TechnoIO.library 4.1

COLLABORATORS			
	TITLE : TechnoIO.library 4.1		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 27, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

TechnoIO.library 4.1

1.1 TechnoIO.library Release 4.1

```
                    ================================
          TechnoIO.library

    (C)1996-1999 by J.Reinert
 ================================


   Funktionen der Library

   (Sortiert)

   (Thematisch)

   Allgemeine Hinweise

   Programmiermodell

   Zeitmessung

   Programmierer
```

1.2 Die Library

Die "TechnoIO.library" ist die Schnittstelle zwischen Ihrer Anwendung und der TechnoIO-Hardware. Sie "sagen" der Library, welche Kanäle aktiviert oder deaktiviert werden sollen und die Library übermittelt diese Information an die Hardware.

In Gegenrichtung liest die Library ständig Daten von der Hardware ein, sodaß der Programmierer jederzeit Zugriff auf diese Daten hat. Die Datenübermittlung zwischen Library und Hardware erfolgt per Interrupt (Unterbrechung). Die Interrupt-Routine hat dabei die im Amigasystem übliche Priorität Null und ist damit relativ neutral im System. Die Übertragungsgeschwindigkeit kann mit einem

separaten Programm bei Bedarf verändert werden. Da die Interrupt-
Routine recht umfangreich ist, kann es passieren, daß der Amiga
ein wenig ausgebremst wird.

Funktionsübersicht (Sortiert)

Funktionsübersicht (Thematisch)
Unter BlitzBASIC kann die schreibweise wie unter "Anwendung"
beschrieben benutzt werden. Assemblerprogrammierer müssen vor
dem Funktionsaufruf die angegebenen Register mit bestimmten
Werten laden. Das Adressregister A6 muss immer die
Basis-Adresse der "TechnoIO.library" enthalten. Wenn von einer
Funktion ein Ergebnis geliefert wird, so ist dieses im Datenre-
gister D0 zu finden. Alle Parameter und Ergebnisse sind ULong.
Die Prozessorregister D0, D1, A0 und A1 können durch die
Library-Funktionen zerstört werden. Wenn Sie deren Inhalt
benötigen, sollte Sie diesen vor Aufruf einer Funktion retten.

Weiterhin besteht die indirekte Möglichkeit, die Library sowohl
per ARexx als auch per Exec-Messagesystem anzusprechen. Lesen
Sie dazu bitte die Anleitung zum Commoditie "TechnoIORexx".

1.3 Beschreibung alphabethisch

```
32 Eingangskanäle auslesen

LongReadOutput ....
Ausgangszustand von 32 Kanäle holen

LongWriteMaskOutput
32 bestimmte Kanäle ein/ausschalten

LongWriteOutput ...
Ein Langwort (32 Bit) ausgeben

ReadIn32 ..........
Die untersten 32 Eingangskanäle einlesen

ReadMem ...........
Eingangsdaten in Speicherbereich kopieren

ReadOut32 .........
Zustände der Kanäle 0 bis 31 einlesen

ReadStaticInput ...
Pin 1 bis 3 des Gameports abfragen

ReadTimer .........
Zähler der Zeitmessung auslesen

ResetTimer ........
Zähler der Zeitmessung auf Null setzen

SaveConfig ........
Timing als Vorgabe sichern

SaveIO ............
Ausgabekanäle in Datei sichern

SetTiming .........
Timing für Datenrefresh und Zeitmessung setzen

ShowLight .........
Kanal 0 ein- bzw. ausschalten

SwitchChannel .....
Einen bestimmten Kanal ein/ausschalten

SwitchOff .........
Einen bestimmten Kanal ausschalten

SwitchOn ..........
Einen bestimmten Kanal einschalten

TestInputbit ......
Einen Eingangskanal auf Signal testen

TestOutputbit .....
Prüfen, ob ein Ausgang ein- oder ausgeschaltet ist

TestStaticInput ...
```

```
Pin 1,2 oder 3 auf Signal testen

WordReadInput  ....
16 Eingangskanäle auslesen

WordReadOutput ....
Ausgangszustand von 16 Kanäle holen

WordWriteMaskOutput
16 bestimmte Kanäle ein/ausschalten

WordWriteOutput ...
Ein Wort (16 Bit) ausgeben

WriteChars ........
Daten im 7-Segment-Code ausgeben

WriteMaskOutput ...
Bestimmte Kanäle der ersten 32 ein/ausschalten

WriteMem ..........
Speicherbereich als Daten ausgeben

WriteOut32 ........
Kanal 0 bis 31 ein/ausschalten
```

1.4 Beschreibung thematisch

```
Allgemeine Funktionen:

GetOpenErrors .....
Fehler der Library abfragen

GetOpenErrorText ..
Fehlertext zu einem Fehler holen

SetTiming .........
Timing für Datenrefresh und Zeitmessung setzen

SaveIO ............
Ein- Ausgabekanäle in Datei sichern

SaveConfig ........
Timing als Vorgabe sichern


Zeitmessung:

ShowLight .........
Kanal 0 ein- bzw. ausschalten

EnableRace ........
Zeitmessung ermöglichen
```

```
                DisableRace .......
                Zeitmessung abbrechen

                GetStatus .........
                Zustand der Zeitmessung abfragen

                ReadTimer .........
                Zähler der Zeitmessung auslesen

                ResetTimer ........
                Zähler der Zeitmessung auf Null setzen


Datenausgabe:

                ShowLight .........
                Kanal 0 ein- bzw. ausschalten


                SwitchChannel .....
                Einen bestimmten Kanal ein/ausschalten

                SwitchOn ..........
                Einen bestimmten Kanal einschalten

                SwitchOff .........
                Einen bestimmten Kanal ausschalten

                TestOutputbit .....
                Prüfen, ob ein Ausgang ein- oder ausgeschaltet ist


                WriteOut32 ........
                Kanal 0 bis 31 ein/ausschalten

                WriteMaskOutput ...
                Bestimmte Kanäle der ersten 32 ein/ausschalten

                ReadOut32 .........
                Zustände der Kanäle 0 bis 31 einlesen


                WriteMem ..........
                Speicherbereich als Daten ausgeben

                WriteChars ........
                Daten im 7-Segment-Code ausgeben


                ByteWriteOutput ...
                Ein Byte (8 Bit) ausgeben

                WordWriteOutput ...
                Ein Wort (16 Bit) ausgeben

                LongWriteOutput ...
                Ein Langwort (32 Bit) ausgeben
```

```
                ByteWriteMaskOutput
                8 bestimmte Kanäle ein/ausschalten

                WordWriteMaskOutput
                16 bestimmte Kanäle ein/ausschalten

                LongWriteMaskOutput
                32 bestimmte Kanäle ein/ausschalten


                ByteReadOutput ....
                Ausgangszustand von 8 Kanäle holen

                WordReadOutput ....
                Ausgangszustand von 16 Kanäle holen

                LongReadOutput ....
                Ausgangszustand von 32 Kanäle holen


 Dateneingabe:

                ReadStaticInput ...
                Pin 1 bis 3 des Gameports abfragen

                TestStaticInput ...
                Pin 1,2 oder 3 auf Signal testen


                ReadIn32  .........
                Die untersten 32 Eingangskanäle einlesen

                TestInputbit ......
                Einen Eingangskanal auf Signal testen


                ByteReadInput  ....
                8 Eingangskanäle auslesen

                WordReadInput  ....
                16 Eingangskanäle auslesen

                LongReadInput  ....
                32 Eingangskanäle auslesen


                ReadMem ...........
                Eingangsdaten in Speicherbereich kopieren
```

1.5 EnableRace

```
               Name......: TIO_EnableRace

Funktion..: Zeitmessung ermöglichen (Start freigeben)

Anwendung.: Success = TIO_EnableRace_()

Offset....: -30

Parameter.: Keine

Ergebnis..: TRUE  = Start ist freigeschaltet
            FALSE = Zeitmessung läuft bereits. Nur mit
                    DisableRace abzubrechen.

Bemerkung.: Die Strecke (Rennstrecke) gilt als belegt, wenn der
            Zähler aktiviert ist. Dies erfolgt nach Startfreigabe
            mit durchbrechen der Start-Lichtschranke (Pin 1 am
            GamePort). Eine angeschlossene Start-Ampel (Kanal 0)
            wird automatisch gesteuert.

Siehe auch:
               DisableRace

               ShowLight
```

1.6 DisableRace

```
               Name......: TIO_DisableRace

Funktion..: Zeitmessung abbrechen

Anwendung.: Success = TIO_DisableRace_(Always)

Offset....: -36

Parameter.: Always (Register D0)
              =0   wenn Strecke belegt ist, wird nicht
                   abgebrochen
             <>0   es wird auf jeden Fall abgebrochen

Ergebnis..: TRUE    Lauf bzw Start wurde abgebrochen
            FALSE   Strecke belegt. Nicht abgebrochen

Bemerkung.: Für freundliches Abbrechen einer Zeitmessung sollte auf
            jeden Fall zuerst mit Always=0 abgebrochen werden. War
            der Teilnehmer noch nicht gestartet, wird das Rotlicht
            aktiviert und die Start-Lichtschranke nicht mehr abge-
            fragt. Wenn der Teinehmer bereits unterwegs ist, wird
            dies mit FALSE gemeldet. Wenn tatsächlich abgebrochen
            werden soll, muss der Funktionsaufruf mit Always<>0
            erfolgen.

Siehe auch:
```

```
                EnableRace

                ShowLight
```

1.7 GetStatus

```
                Name......: TIO_GetStatus

Funktion..: Zustand der Zeitmessung abfragen

Anwendung.: Status = TIO_GetStatus_()

Offset....: -42

Parameter.: Keine

Ergebnis..: 0 = Nichts passiert
            1 = Start wurde freigegeben
            2 = Strecke belegt (Zähler läuft)
            3 = Teilnehmer hat Ziel durchlaufen (Zähler steht still)

Bemerkung.: Keine

Siehe auch:
                ResetTimer
```

1.8 ReadTimer

```
                Name......: TIO_ReadTimer

Funktion..: Auslesen des aktuellen Zählerstandes

Anwendung.: Time100 = TIO_ReadTimer_()

Offset....: -48

Parameter.: Keine

Ergebnis..: Aktueller Zählerstand

Bemerkung.: Der Zählerstand wird in hundertstel Sukunden geliefert.
            Um die Sekunden zu erhalten, muss das Ergebnis also
            noch durch 100 geteilt werden.

Siehe auch:
                ResetTimer
```

1.9 GetOpenErrors

```
                Name.......: TIO_GetOpenErrors

Funktion..: Fehler, die bei öffnen der Library aufgetreten sein
            könnten, ermitteln

Anwendung.: Fehler = TIO_GetOpenErrors_()

Offset....: -54

Parameter.: Keine

Ergebnis..: Bit 0 gesetzt = Keine CIA-Resource. Kann eigentlich nur
                            passieren,wenn keine CIAs vorhanden sind
                            (Draco ?)

            Bit 1 gesetzt = CIA-Basisadresse nicht ermittelt. Wenn
                            schon keine CIA-Resource, dann auch kein
                            CIA

            Bit 2 gesetzt = Fehler mit Potgo-Resource. Ähnlicher Fall
                            wie mit Bit 0

            Bit 3 gesetzt = Keine Datenausgabe möglich. Ein anderes
                            Programm könnte den Gameport blockieren

            Bit 4 gestezt = Interrupt-Routine wurde nicht installiert.
                            Einanderes Programm hat den Timer belegt

Bemerkung.: Gemeldet werden Fehler, die beim öffnen der Library auf-
            treten, dort aber nicht weiter verarbeitet werden konnten.

Siehe auch:
                GetOpenErrorText
```

1.10 ResetTimer

```
                Name.......: TIO_ResetTimer

Funktion..: Zähler für Zeitmessung auf Null

Anwendung.: Input = TIO_ResetTimer_()

Offset....: -60

Parameter.: Keine

Ergebnis..: Letzter Zählerstand

Bemerkung.: Der Zähler wird automatisch auf Null gestellt, wenn eine
            neue Zeitmessung gestartet wird. Der Status wird mit
            dieser Funktion ebenfalls auf Null gesetzt
```

```
Siehe auch:
                ReadTimer

                GetStatus
```

1.11 GetOpenErrorText

```
                Name......: TIO_GetOpenErrorText

Funktion..: Fehlertexte holen, falls Fehler beim öffnen der Library
            aufgetreten sind

Anwendung.: *Text = TIO_GetOpenErrorText_(BitNr)

Offset....: -66

Parameter.: BitNr   (Register D0)
                  Nummer des Fehlerbits von 0 bis 4. Falls Bit 4
                  gesetzt ist, kann mit BitNr=99 der Name des Tasks
                  ermittelt werden, der den Timer nutzt

Ergebnis..: Zeiger auf Text mit Zusatzinformationen oder Null, falls
            dem Bit kein Fehler zugeordnet ist.

Bemerkung.: Falls die Funktion
                GetOpenErrors
                einen Wert ungleich
            Null lieferte, kann die Funktion in einer Schleife alle
            Fehlertexte liefern.

            Beispiel für BlitzBASIC V2.1:

            for i=0 to 4
               err.l=TIO_GetOpenErrorText_(i)
               if err.l<>0
                  x$=peek$(err.l)
                  if i=4
                    x$+"|Belegt durch:"+peek$(TIO_GetOpenErrorText_(99))
                  endif
                  Request "Fehler",peek$(err),"Okay"
               endif
            next i

Siehe auch:
                GetOpenErrors
```

1.12 ShowLight

```
               Name......: TIO_ShowLight

Funktion..: Schaltet Start-Ampel an Kanal 0, Modul 0 um

Anwendung.: Dummy = TIO_ShowLight_(Light)

Offset....: -72

Parameter.: Light  (Register D0)
                = 0  Rotes Licht einschalten
                <>0  Grünes Licht einschalten

Ergebnis..: Nichts wichtiges

Bemerkung.: Für grünes Licht wird der Ausgang 0 (Kanal 0) auf
            Hi-Pegel (5Volt) gesetzt. Rotlicht sollte bei Null
            Volt an diesem Ausgang leuchten

Siehe auch:
               EnableRace

               DisableRace
```

1.13 ReadStaticInput

```
               Name......: TIO_ReadStaticInput

Funktion..: Abfragen der Gameport-Pins 1 bis 3

Anwendung.: StaticInput = TIO_ReadStaticInput_()

Offset....: -78

Parameter.: Keine

Ergebnis..: Bit 0 gesetzt    Pin 1 hat Null Volt
            Bit 1 gesetzt    Pin 2 hat Null Volt
            Bit 2 gesetzt    Pin 3 hat Null Volt

Bemerkung.: Es ist zu beachten, daß die drei Eingänge Low-Active
            sind. Ein bit im Ergebnis ist also gesetzt, wenn der
            zugehörige Anschluss (Pin) auf Masse liegt. Dies hat
            den Vorteil, daß fehlende Kontakte nicht als betätigt
            gemeldet werden.

Siehe auch:
               TestStaticInput
```

1.14 TestStaticInput

```
               Name......: TIO_TestStaticInput

Funktion..: Testet die einzelnen Static-Eingäne (Pin 1 bis 3) auf
            Signal

Anwendung.: Signal = TIO_TestStaticInput_(BitNr)

Offset....: -84

Parameter.: BitNr   (Register D0)
                =0, Pin 1 testen
                =1, Pin 2 testen
                =2, Pin 3 testen

Ergebnis..: FALSE wenn kein Signal
            TRUE  wenn Signal vorhanden ist

Bemerkung.: Da die drei statischen Eingänge des Gameports direkt
            abgefragt werden, ist mit dieser Funktion ein schneller
            Signaltest möglich. Die Abtastrate wird durch Timer-A
            von CIA-B bestimmt

Siehe auch:
               ReadStaticInput

               SetTiming
```

1.15 WriteOut32

```
               Name......: TIO_WriteOut32

Funktion..: Untersten 32 Kanäle (Modul 0 bis 3) gleichzeitig ein-
            bzw ausschalten

Anwendung.: Dummy = TIO_WriteOut32_(Outdata)

Offset....: -90

Parameter.: Outdata  (Register D0)

                Bitkombination, die in die untersten 32 Kanäle
                (Kanal 0 bis 31) geschrieben werden soll. Jedes
                gesetzte Bit bedeutet, daß der korrespondierende
                Kanal eingeschaltet werden soll (+ 5Volt). Das
                Low-Byte landet dabei in Modul 0

Ergebnis..: Nichts wichtiges

Bemerkung.: Mit dieser Funktion ist es sehr einfach, eine 32-Kanal
            Relaiskarte zu steuern. Bit 0 wird übrigens auch von der
            integrierten Zeitmessung beeinflusst. Hierüber soll eine
            Start-Ampel gesteuert werden.
```

```
Siehe auch:
                ShowLight
                  ReadOut32
                SwitchChannel
```

1.16 ReadOut32

```
                Name......: TIO_ReadOut32

Funktion..: Ausgangszustand der untersten 32 Kanäle ermitteln

Anwendung.: Outdata = TIO_ReadOut32_()

Offset....: -96

Parameter.: Keine

Ergebnis..: Bitkombination der Kanäle 0 bis 31 (Bit0=Kanal0)

Bemerkung.: Da diese Library von mehreren Programmen zeitgleich
            genutzt werden könnte, ist es sinnvoll, den Ausgangs-
            zustand der Kanäle vorab zu ermitteln. Die Reser-
            vierung einzelner Kanäle habe ich noch nicht vorgesehen.

Siehe auch:
                WriteOut32
```

1.17 SwitchChannel

```
                Name......: TIO_SwitchChannel

Funktion..: Einen bestimmten Kanal ein- oder ausschalten

Anwendung.: Outdata = TIO_SwitchChannel_(Channel,Flag)

Offset....: -102

Parameter.: Channel  (Register D0)
                  gibt an, welcher Kanal geschaltet werden soll.
                  Zugelassen sind Werte von 0 bis 511

            Flag     (Register D1)
                  gibt an, ob der gewünschte Kanal ein- oder
                  ausgeschaltet werden soll. Der Wert 0 schaltet
                  aus und mit einem Wert ungleich 0 wird der
                  betreffende Kanal eingeschaltet.

Ergebnis..: Nummer des Modules, in welchem ein Bit verändert wurde

Bemerkung.: Keine
```

```
Siehe auch:
                SwitchOn

                SwitchOff

                Programmiermodell
```

1.18 SwitchOn

```
                Name......: TIO_SwitchOn

Funktion..: Einen bestimmten Kanal einschalten

Anwendung.: Outdata = TIO_SwitchOn_(Channel)

Offset....: -108

Parameter.: Channel    (Register D0)
                  gibt an, welcher Kanal eingeschaltet werden
                  soll. Zugelassen sind Werte von 0 bis 511

Ergebnis..: Nummer des Modules, an welchem ein Bit gesetzt wurde

Bemerkung.: Keine

Siehe auch:
                SwitchChannel

                SwitchOff

                Programmiermodell
```

1.19 SwitchOff

```
                Name......: TIO_SwitchOff

Funktion..: Einen bestimmten Kanal ausschalten

Anwendung.: Outdata = TIO_SwitchOff_(Channel)

Offset....: -114

Parameter.: Channel    (Register D0)
                  gibt an, welcher Kanal eingeschaltet werden
                  soll. Zugelassen sind Werte von 0 bis 511

Ergebnis..: Nummer des Modules, an welchem ein Bit gelöscht wurde.

Bemerkung.: Keine
```

```
Siehe auch:
                SwitchChannel

                SwitchOn

                Programmiermodell
```

1.20 WriteMaskOut32

```
                Name......: TIO_WriteMaskOut32

Funktion..: Bestimmte Kanäle des unteren Bereiches gleichzeitig
            ein- bzw ausschalten

Anwendung.: Outdata = TIO_WriteMaskOut32_(Outdata,Outmask)

Offset....: -120

Parameter.: Outdata    (Register D0)

                  Enthält Ein-/Aus-Informationen für jeden Kanal.
                  Jedes Bit in diesem Langwort repräsentiert
                  einen Kanal. Ein gesetztes Bit aktiviert den
                  Kanal.


            Outmask    (Register D1)

                  Enthält die Information, welche Kanäle tatsäch-
                  lich beeinflusst werden dürfen. Ein gesetztes
                  Bit verändert den betreffenden Kanal anhand des
                  korespondierenden Bits in Outdata (D0). Ein
                  gelöschtes Bit lässt den Kanalzustand unver-
                  ändert.

Ergebnis..: Aktueller Ausgangszustand

Bemerkung.: Es ist zu beachten, daß mit dieser Funktion nur die
            untersten 32 Kanäle beeinflusst werden können.

Siehe auch:
                WriteOut32

                ReadOut32
```

1.21 ReadIn32

```
                Name......: TIO_ReadIn32
```

```
Funktion..: Einlesen der untersten 32 Eingangskanäle

Anwendung.: Indata = TIO_ReadIn32_()

Offset....: -126

Parameter.: Keine

Ergebnis..: Bitkombination der eingegangenen untersten 32 Kanäle

Bemerkung.: Der Lesevorgang selbst wird per Interrupt durchgeführt.
            Mit dieser Funktion können die eingelesenen Daten
            ermittelt werden. Das niedrigste Bit steht wieder für
            Kanal 0, das höchste für Kanal 31.

Siehe auch:
                TestInputbit

                ReadStaticInput

                TestStaticInput
```

1.22 SetTiming

```
Name......: TIO_SetTiming

Funktion..: Parameter der Interrupt-Routine verändern

Anwendung.: dummy = TIO_SetTiming_(Timer,Divider)

Offset....: -132

Parameter.: Timer      (Register D0)
                  Dieser Wert wird in das Lese-Register von Timer-A
                  in CIA-B geschrieben. Er legt die Frquenz des
                  zugehörigen Interrupts fest. Die Grundfrequenz des
                  CIA liegt bei etwa 709kHz. Ein Timer-Wert von 709
                  würde die Interrupt-Routine etwa 1000 mal pro
                  Sekunde aufrufen. Folgendes Verhalten ist festzu-
                  stellen:

                  - Je kleiner der Wert, um so schneller werden die
                    Ausgabedaten übertragen

                  - Je kleiner der Wert, um so schneller werden die
                    Eingangsdaten aktualisiert (Auch die statischen
                    Eingänge des Gameports)

                  - Ist der Wert zu klein, bleibt der Amiga wegen
                    Überlastung stehen (Werte kleiner als 50 ?)

                  Der Timer-Wert verändert auch die Grundfrquenz
                  für die Zeitmessung
```

```
            Divider    (Register D1)
                   Der hier angegebene Wert beeinflusst nur die Zeit-
                   mess-Einrichtung der Library. Je nach programmiertem
                   Timer-Wert muß für eine genaue Zählung dieser Wert
                   dimensioniert werden. Zwei Werte haben sich für
                   meinen Rechner (A4000-040/40) bewährt:

                   - Timer=172,Divider=41
                     Datenaktualisierung mit ca 4100 Hz
                     Zeitmessung mit 100,011 Hz

                   - Timer=709,Divider=10
                     Datenaktualisierung mit ca 999,12 Hz
                     Zeitmessung mit 99,912 Hz

Ergebnis..: Nichts

Bemerkung.: Der Abstand, in der die Ein- und Ausgabedaten aktualisiert
            werden, kann wie folgt berechnet werden:

            F = 1/((MaxWords*32*3+2)/709379Hz*(Timer+1))

            Beispiel:  MaxWords=16
                       Timer=709
                       F=0.65 Hz oder 1,54 Sek pro Durchlauf

            Wenn für eine Zeitmessung kein genauer Wert zu finden
            ist, so muß Ihr Steuer-Programm für die richtige
            Genauigkeit sorgen

Siehe auch:
```

1.23 TestInputbit

```
                Name......: TIO_TestInputbit

Funktion..: Einen Eingangskanal auf Zustand testen

Anwendung.: Signal = TIO_TestInputbit_(Bit)

Offset....: -138

Parameter.: Bit        (Register D1)
                   Dieser Parameter gibt an, welches Bit bzw
                   welcher Eingangskanal auf Signal getestet
                   werden soll. Der Wertebereich erstreckt
                   sich von 0 bis 511. Bit 0 steht für Ein-
                   gangskanal 0, Bit 511 für Eingang 511.

Ergebnis..: Wenn am getesteten Eingang Hi-Pegel anliegt, wird
            TRUE (ein Wert ungleich Null) zurückgegeben. Bei
            Lo-Pegel erhalten Sie eine  0 (FALSE)

Bemerkung.: Bitte beachten Sie, daß die seriellen Eingangskanäle
```

```
            (die von der TechnoIO-Hardware) im Gegensatz zu den
            drei statischen Eingängen des Gameports Hi-Active
            sind

Siehe auch:
                TestStaticInput
```

1.24 WriteChars

```
                Name......: TIO_WriteChars

Funktion..: ASCII-Zeichen ausgeben

Anwendung.: dummy = TIO_WriteChars_(LowModul,Source,Convert)

Offset....: -144

Parameter.: LowModul   (Register D0)
                Hier muß ein Wert im Bereich 0 bis 63 angegeben
                werden. Er gibt an, welches Ausgabemodul das
                erste (linke) Zeichen ausgeben soll. Alle nach-
                folgenden Zeichen werden in die nächst höheren
                Module geschrieben.

            Source     (Register A0)
                Adresse der auszugebenden Zeichenkette als
                "C"-Text (Abgeschlossen mit einem Null-Byte)

            Convert    (Register D1)
                Wenn Convert==0 ist, werden alle Zeichen direkt
                ab "LowModul" ausgegeben. Wird "Convert" auf einen
                Wert ungleich 0 gesetzt, erfolgt eine Konvertierung
                in 7-Segment-Code zum ansteuern von 7-Segment
                Digits.

Ergebnis..: Nichts

Bemerkung.: Da die Library über eine Zeitmessung verfügt liegt es
            nahe, die gemessene Zeit über ein geeignetes externes
            Display auszugeben. Damit der externe Hardwareaufwand
            auf ein Minimum reduziert werden kann, erledigt diese
            Funktion die Unwandlung in den zur Anzeige notwendigen
            7-Segment-Code. An den digitalen Ausgängen der Hardware
            können dann unter Verwendung eines Treibers die 7-Segment-
            Anzeigen direkt angeschlossen werden. Ein integrierter
            Schaltkreis, der diese Codewandlung vornehmen würde,
            kostet etwa 4,-DM. Ein Treiber (ULN2803) ist schon für
            1,10 DM zu bekommen. Außer Ziffern können auch einige
            Buchstaben dargestellt werden. Experimentieren Sie etwas.

Siehe auch:
                WriteMem

                ReadMem
```

1.25 TestOutputbit

```
               Name......: TIO_TestOutputbit

Funktion..: Testen, ob ein bestimmter Kanal eingeschaltet ist

Anwendung.: Result = TIO_TestOutputbit_(Channel)

Offset....: -150

Parameter.: Channel  (Register D1)
                  Hier muss ein Wert im Bereich 0 bis 511 angegeben
                  werden und repräsentiert den zu testenden Kanal

Ergebnis..: FALSE, wenn Ausgang ausgeschaltet ist bzw
            TRUE, wenn Ausgang eingeschaltet ist

Siehe auch:
               TestInputbit
```

1.26 ByteWriteOutput

```
               Name......: TIO_ByteWriteOutput

Funktion..: 8-Bit Datenausgabe an Hardware

Anwendung.: dummy = TIO_ByteWriteOutput_(Modul,Outdata)

Offset....: -162

Parameter.: Modul      (Register D0)
                  Nummer des Modules, an dem die 8 Bits ausgegeben
                  werden sollen

            Outdata    (Register D1)
                  Datenbyte, welches an das angegebene Modul
                  übertragen wird.

Ergebnis..: Nichts

Bemerkung.:

Siehe auch:
               WriteChars

               WordWriteOutput

               LongWriteOutput
```

```
                WriteMem

                ReadMem
```

1.27 WordWriteOutput

```
                Name......: TIO_WordWriteOutput

Funktion..: 16-Bit Datenausgabe an Hardware

Anwendung.: dummy = TIO_WordWriteOutput_(LowModul,Outdata)

Offset....: -168

Parameter.: LowModul   (Register D0)
                Erste Kanalgruppe von 8 Bit (Modul) wie schon bei
                "TIO_WriteChars" beschrieben.

            Outdata    (Register D1)
                Die untersten 16 Bits dieses Parameters werden
                ab dem angegebenen Ausgabebyte geschrieben.
                Es können also 16 Ausgabekanäle gleichzeitig
                beeinflusst werden.

Ergebnis..: Nichts

Bemerkung.:

Siehe auch:
                WriteChars

                ByteWriteOutput

                LongWriteOutput

                WriteMem

                ReadMem
```

1.28 LongWriteOutput

```
                Name......: TIO_LongWriteOutput

Funktion..: 32-Bit Datenausgabe an Hardware

Anwendung.: dummy = TIO_LongWriteOutput_(LowModul,Outdata)

Offset....: -174

Parameter.: LowModul   (Register D0)
```

```
                Erste Kanalgruppe von 8 Bit wie schon bei
                "TIO_WriteChars" beschrieben.

            Outdata    (Register D1)
                Alle 32 Bits dieses Parameters werden ab dem
                angegebenen Modul ausgegeben. Es können damit
                32 Ausgabekanäle gleichzeitig geschaltet werden

Ergebnis..: Nichts

Bemerkung.:

Siehe auch:
                WriteChars

                ByteWriteOutput

                WordWriteOutput

                WriteMem

                ReadMem
```

1.29 ByteReadOutput

```
                Name......: TIO_ByteReadOutput

Funktion..: Ermitteln der Ausgangszustände von 8 Kanäle

Anwendung.: Outdata = TIO_ByteReadOutput_(Modul)

Offset....: -180

Parameter.: Modul  (Register D0)
                Nummer des Ausgabemodules, dessen Ausgabe-
                zustand ermittelt werden soll

Ergebnis..: 8-Bit Bitkombination von 8 Ausgabekanäle

Bemerkung.: Da mehrere Programme die Library zeitgleich nutzen
            könnten, kann mit dieser Funktion der aktuelle
            Zustand von 8 Kanälen ermittelt werden.

Siehe auch:
                WriteChars

                WordReadOutput

                LongReadOutput

                ReadOut32
```

1.30 WordReadOutput

```
                Name......: TIO_WordReadOutput

Funktion..: Ermitteln der Ausgangszustände von 16 Kanäle

Anwendung.: Outdata = TIO_WordReadOutput_(LowModul)

Offset....: -186

Parameter.: LowModul  (Register D0)
                  Die erste Kanalgruppe von 8 Bit wie schon bei
                  "TIO_WriteChars" beschrieben.

Ergebnis..: 16-Bit Bitkombination von 16 Ausgabekanäle (zwei
            aufeinanderfolgende Module ab angegebenem Modul)

Bemerkung.: Da mehrere Programme die Library zeitgleich nutzen
            könnten, kann mit dieser Funktion der aktuelle
            Zustand von 16 Kanälen ermittelt werden.

Siehe auch:
                WriteChars

                ByteReadOutput

                LongReadOutput
                             ReadOut32
```

1.31 LongReadOutput

```
                Name......: TIO_LongReadOutput

Funktion..: Ermitteln der Ausgangszustände von 32 Kanäle

Anwendung.: Outdata = TIO_LongReadOutput_(LowModul)

Offset....: -192

Parameter.: LowModul  (Register D0)
                  Die erste Kanalgruppe von 8 Bit wie schon bei
                  "TIO_WriteChars" beschrieben. Byte 0 repräsen-
                  tiert die Kanäle 0 bis 7, Byte 1 die Kanäle
                  8 bis 15 usw.

Ergebnis..: 32-Bit Bitkombination von 32 Ausgabekanäle

Bemerkung.: Da mehrere Programme die Library zeitgleich nutzen
            könnten, kann mit dieser Funktion der aktuelle
            Zustand von 32 Kanälen ermittelt werden.
```

```
Siehe auch:
                WriteChars

                ByteReadOutput

                WordReadOutput

                ReadOut32
```

1.32 ByteReadInput

```
                Name......: TIO_ByteReadInput

Funktion..: Lesen von 8 Eingangskanäle

Anwendung.: Indata = TIO_ByteReadInput_(Modul)

Offset....: -196

Parameter.: Modul  (Register D0)
                   Nummer des auszulesenden Modules im
                   Bereich von 0 bis 63

Ergebnis..: 8-Bit Bitkombination 8 Eingabekanäle

Bemerkung.: Jedes gesetzte Bit im eingelesenen Byte bedeutet,
            daß der zugehörige Kanal an Hi-Pegel liegt

Siehe auch:
                WriteChars

                WordReadInput

                LongReadInput

                ReadIn32
```

1.33 WordReadInput

```
                Name......: TIO_WordReadInput

Funktion..: Lesen von 16 Eingangskanäle

Anwendung.: Indata = TIO_WordReadInput_(LowModul)

Offset....: -204

Parameter.: LowModul   (Register D0)
                   Erste Kanalgruppe von 8 Bit wie schon bei
                   "TIO_WriteChars" beschrieben.
```

```
Ergebnis..: 16-Bit Bitkombination 16 Eingabekanäle

Bemerkung.: Jedes gesetzte Bit im eingelesenen Word bedeutet,
            daß der zugehörige Kanal an Hi-Pegel liegt

Siehe auch:
                WriteChars

                ByteReadInput

                LongReadInput

                ReadIn32
```

1.34 LongReadInput

```
                Name......: TIO_LongReadInput

Funktion..: Lesen von 32 Eingangskanäle

Anwendung.: Indata = TIO_LongReadInput_(LowModul)

Offset....: -210

Parameter.: LowModul  (Register D0)
                  Nummer des Eingabemodules, dessen Daten in die
                  untersten 8 Bit des Ergebnisses kopiert werden
                  soll

Ergebnis..: 32-Bit Bitkombination 32 Eingabekanäle

Bemerkung.: Jedes gesetzte Bit im eingelesenen Long-Word bedeutet,
            daß der zugehörige Kanal an Hi-Pegel liegt

Siehe auch:
                WriteChars

                ByteReadInput

                WordReadInput

                ReadIn32
```

1.35 ByteWriteMaskOutput

```
                Name......: TIO_ByteWriteMaskOutput

Funktion..: 8 Bestimmte Kanäle gleichzeitig schalten
```

```
Anwendung.: Outdata = TIO_ByteWriteMaskOutput_(LowModul,Outdata,Outmask)

Offset....: -216

Parameter.: LowModul   (Register D0)
                       Eine Kanalgruppe von 8 Bit wie schon bei

                WriteChars
                beschrieben

            Outdata    (Register D1)
                       Jedes der 8 Bits repräsentiert einen Kanal
                       in der angesprochenen Kanal-Gruppe (Modul
                       mit 8 Bit). Ein gesetztes Bit schaltet den
                       Kanal ein, ein gelöschtes Bit schaltet ihn
                       aus. Welche Bits tatsächlich berücksichtigt
                       werden, hängt von den Bits im Parameter
                       "Outmask" ab

            Outmask    (Register D2)
                       Eine 8-Bit Maske die festlegt, welche Kanäle be-
                       einflusst werden sollen.

Ergebnis..: Neuer Ausgangszustand des angesprochenen Modules

Bemerkung.: Diese Funktion ähnelt der Funktion
                WriteMaskOut32
               . Es
            lässt sich hiermit aber jedes der 64 Module einzeln beeinflussen.

Siehe auch:
                WordWriteMaskOutput

                LongWriteMaskOutput
```

1.36 WordWriteMaskOutput

```
                Name......: TIO_WordWriteMaskOutput

Funktion..: 16 Bestimmte Kanäle gleichzeitig schalten

Anwendung.: Outdata = TIO_WordWriteMaskOutput_(LowModul,Outdata,Outmask)

Offset....: -222

Parameter.: LowModul   (Register D0)
                       Eine Kanalgruppe von 8 Bit wie schon bei

                WriteChars
                beschrieben

            Outdata    (Register D1)
                       Jedes der 16 unteren Bits repräsentiert einen
                       Kanal in der angesprochenen Kanal-Gruppe (Modul
```

```
                        mit 8 Bit) und der darauf folgenden Gruppe. Ein
                        gesetztes Bit schaltet den Kanal ein, ein ge-
                        löschtes Bit schaltet ihn aus. Welche Bits tat-
                        sächlich berücksichtigt werden, hängt von den
                        Bits im Parameter "Outmask" ab

              Outmask   (Register D2)
                        Eine 16-Bit Maske die festlegt, welche Kanäle be-
                        einflusst werden sollen.

Ergebnis..: Neuer Ausgangszustand der angesprochenen Module

Bemerkung.: Diese Funktion ähnelt der Funktion
                WriteMaskOut32
               . Es
            lassen sich hiermit aber zwei beliebige, aufeinander folgende
            Module beeinflussen.

Siehe auch:
                ByteWriteMaskOutput

                LongWriteMaskOutput
```

1.37 LongWriteMaskOutput

```
                Name......: TIO_LongWriteMaskOutput

Funktion..: 32 Bestimmte Kanäle gleichzeitig schalten

Anwendung.: Outdata = TIO_LongWriteMaskOutput_(LowModul,Outdata,Outmask)

Offset....: -228

Parameter.: LowModul  (Register D0)
                        Eine Kanalgruppe von 8 Bit wie schon bei

                WriteChars
                beschrieben

              Outdata   (Register D1)
                        Jedes der 32 Bits repräsentiert einen Kanal in
                        der angesprochenen Kanal-Gruppe (Modul mit 8
                        Bit) und den drei darauf folgenden Gruppen. Ein
                        gesetztes Bit schaltet den Kanal ein, ein ge-
                        löschtes Bit schaltet ihn aus. Welche Bits tat-
                        sächlich berücksichtigt werden, hängt von den
                        Bits im Parameter "Outmask" ab

              Outmask   (Register D2)
                        Eine 32-Bit Maske die festlegt, welche Kanäle be-
                        einflusst werden sollen.

Ergebnis..: Neuer Ausgangszustand der angesprochenen Module
```

```
Bemerkung.: Diese Funktion ähnelt der Funktion
                WriteMaskOut32
               . Es
            lassen sich hiermit aber vier beliebige, aufeinander folgende
            Module beeinflussen.

Siehe auch:
                ByteWriteMaskOutput

                WordWriteMaskOutput
```

1.38 WriteMem

```
                Name......: TIO_WriteMem

Funktion..: Ganzen Speicherbereich als Daten ausgeben

Anwendung.: dummy = TIO_WriteMem_(LowModul,Source,Size,Mode)

Offset....: -234

Parameter.: LowModul    (Register D0)
                Erste Kanalgruppe von 8 Bit, die mit bearbeitet
                werden sollen. Es können Werte von 0 bis 63
                angegeben werden, wobei 0 für die untersten
                8 Kanäle steht, 1 für die Kanäle 8 bis 15 usw.

            Source      (Register D1)
                Anfangsadresse eines Speicherbereiches, der die
                auszugebenden Daten enthält

            Size        (Register D2)
                Anzahl Daten, die an die Hardware ausgegeben
                werden sollen.

            Mode        (Register D3)
                Bestimmt die Art und Weise, wie die Daten ge-
                schrieben werden sollen. Ein Wert von 0 bedeu-
                tet, das die Daten einfach nur geschrieben
                werden, eine 1 dagegen startet eine Synchrone
                Übertragung. Mehr bei Bemerkung

Bemerkung.: Was meine ich mit synchron? Es gibt grundsätzlich
            mindestens zwei Möglichkeiten, Daten von einem Sender
            zum Empänger zu transportieren. Nämlich Synchron und
            asynchron. Bei der asynchronen Übertragung muß der
            Empfänger selbstständig die Daten übernehmen. Dazu
            müssen meistens zusätzlich zu den Daten auch Steuer-
            signale übertragen werden.

            Da meine Interrupt-Routine selbstständig in bestimm-
            ten Abständen die Ausgabedaten an die Hardware sendet,
            könnte es durchaus passieren, daß bestimmte Daten,die
            gerade vom Anwenderprogramm ausgegeben werden sollen,
```

```
            nicht zur richtigen Zeit ausgegeben werden, da der
            "richtige Moment" verpasst wurde. Die synchrone Über-
            tragung der Ausgabedaten an die Hardware läuft dann
            folgendermaßen ab:

            1. Warten, bis alle alten Daten zur Hardware geschickt
               wurden

            2. Interrupt abschalten (Es findet keine Aktualisierung
               mehr statt)

            3. Übertragen der Ausgabedaten in den in der Library
               vorgesehenen Ausgabebereich

            4. Starten des Interrupts, und zwar so, daß alle 64
               Byte (16 Langworte) komplett an die Hardware neu
               übertragen werden.

            Die synchrone Übertragung gestattet also auch Zeit-
            genaue Übertragung. Zwar langsam, aber es geht. Damit
            könnten z.B. RAM-Bausteine mit Daten geladen werden,
            die dann zu einem anderen Zeitpunkt ausgelesen werden.

            Auch Lauflichter lassen sich damit programmieren, weil
            die Interrupt-Routine den Ausgabedaten nicht "davon-
            laufen" kann.

Siehe auch:
                ReadMem

                WriteChars
```

1.39 ReadMem

```
                Name......: TIO_ReadMem

Funktion..: Ganzen Speicherbereich mit Eingabedaten füllen

Anwendung.: dummy = TIO_ReadMem_(LowModul,Destination,Size,Mode)

Offset....: -240

Parameter.: LowModul    (Register D0)
                siehe
            WriteMem
                          Destination (Register D1)
                   Anfangsadresse eines Speicherbereiches, der mit
                   Eingangsdaten gefüllt werden soll

            Size        (Register D2)
                   Anzahl Daten, die von Hardware gelesen sollen.
                   Ist dieser Wert zu groß, werden keine Daten
                   eingelesen
```

```
            Mode        (Register D3)
                   Bestimmt die Art und Weise, wie die Daten gelesen
                   werden sollen. Ähnlich wie bei
               TIO_WriteMem
                                    wird gewartet, bis alle Eingangskanäle von der
                   Interrupt-Routine eingelesen wurden.

Ergebnis..: Nichts

Bemerkung.: Mit dieser Funktione finden Sie eine weitere Möglich-
            keit, Daten von der angeschlossene TechnoIO-Hardware
            einzulesen.

Siehe auch:
               WriteMem

               WriteChars
```

1.40 SaveIO

```
               Name......: TIO_SaveIO

Funktion..: Speichert den aktuellen Ein- und Ausgangszustand aller 512
            Ein- und Ausgabekanäle. Die ersten 64 gespeicherten Byte
            repräsentieren die Ausgabemodule. Die letzten 64 Byte
            in der Datei spiegeln die Eingabekanäle.

            Die gespeicherten Ausgabedaten werden mit aktivieren der
            Library automatisch wieder eingelesen und verwendet.

Anwendung.: Result = TIO_SaveIO_(Name)

Offset....: -246

Parameter.: Name        (Register D1)
                  Normalerweise wird dieser Parameter auf Null
                  gesetzt. Die aktuellen Ein- und Ausgabedaten
                  werden dann zur automatischen Wiederverwendung
                  im Systemordner "ENVARC:" abgelegt.
                  Wenn Sie selber die Daten auf einem anderen
                  Datenträger benötigen, können Sie optional
                  einen Pointer auf einen Null-terminierten
                  Dateinamen angeben.

Ergebnis..: =0 bei Fehler, <>0 bei erfolgreicher Speicherung

Bemerkung.: Die Ausgabedaten werden in der Datei "ENVARC:TechnoIO.OUT"
            gespeichert und liegen in der Reihenfolge vor, wie im
            Abschnitt
               Programmiermodell
               beschrieben.
```

1.41 SaveConfig

```
                Name......: TIO_SaveConfig

Funktion..: Timer-Wert und Teiler, die mit
                SetTiming
                eingestellt
            wurden, speichern

Anwendung.: dummy = TIO_SaveConfig_()

Offset....: -252

Parameter.: Keine

Ergebnis..: Nichts

Bemerkung.: Auch das Timing wird mit aktivieren der Library auto-
            matisch geladen. Nach verändern des Timings können Sie
            diese Einstellungen sichern lassen.
```

1.42 Allgemeine Hinweise

```
Die interruptgesteuerte Datenübertragung ist so lange aktiv, wie
die "TechnoIO.library" im Rechner ist.

Die TechnoIO-Hardware befindet sich so lange in einem undefinierten
Zustand, bis die "TechnoIO.library" aktiviert wird.

Das Commoditie "TechnoIO" kann in die Schublade "SYS:WBStartup"
kopiert werden. Es öffnet die "TechnoIO.library", aktiviert das
System und lässt es bis zum nächsten Reset aktiv.

Das Programm "TechnoIORexx" ist ebenfalls ein Commoditie, welches
die Library aktiviert. Es besitzt aber zusätzlich einen ARexx-Port
zum ansteuern der Hardware.
```

1.43 Programmiermodell

```
                Dieser Abschnitt soll als Ein- und Ausgabemodell dienen. Die hier
enthaltenen Informationen benötigen Sie zum programmieren mit der
Library. Ohne diese Informationen kann es passieren, daß Sie einen
falschen Kanal schalten oder auswerten.

Insgesamt gibt es 512 Ausgabe- und 512 Eingabekanäle. Die Anord-
nung der Kanäle ist bei Aus- und Eingabekanäle identisch, sodaß
ich mich im folgenden nur noch auf die Bezeichnung "Kanäle" be-
schränke.

Die 512 Kanäle sind unterteilt in 64 Gruppen (Byte) zu je 8 Kanäle.
Jede Gruppe von 8 Kanäle befindet sich normalerweise auf einem ange-
schlossenen Modul, wobei sich die Kanäle 0 bis 7 auf Modul Nummer
```

0 befinden, die Kanäle 8 bis 15 auf Modul 1 usw. Die Library richtet einen Speicherbereich ein, der quasi einen Spiegel der Kanäle darstellt. In diesen Speicherbereich wird, je nach Library-Funktion, geschrieben oder aus diesem gelesen. Eine Interrupt-Routine in der Library sorgt dafür, daß der Speicherbereich immer den aktuellen Ein- und Ausgabezustand wiedergibt. Sie als Programmierer haben mit den Library-Funktionen also nur Zugriff auf die Speicherbereiche für die Ein- und Ausgabekanäle. Der Abstand der Datenaktualisierung kann mit der Funktion

SetTiming

verändert werden.

Das erste Bytes im Ausgabespeicher spiegelt die 8 Bit des ersten Ausgabemodules. Bit 0 in diesem Byte entspricht also Kanal 0 von Modul 0. Bit 1 ist demnach Kanal 1 von Modul 0. Der Kanal 2 des zweiten Modules ist über das Bit 2 im zweiten gespeicherten Byte erreichbar.

Ausgabespeicher:

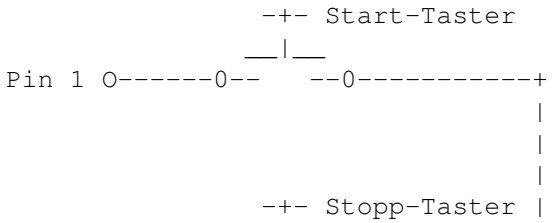
Offset	Bit	Kanal	
0	0	0	
0	1	1	
0	2	2	
0	3	3	Modul Nummer 0
0	4	4	
0	5	5	
0	6	6	
0	7	7	
1	7	8	
1	6	9	
1	5	10	
1	4	11	Modul Nummer 1
1	3	12	
1	2	13	
1	1	14	
1	0	15	
2	0-7	16-23	Modul Nummer 2
3	0-7	24-31	Modul Nummer 3
4	0-7	32-39	4
5	0-7	40-47	5
6	0-7	48-55	6
7	0-7	56-63	7
8	0-7	64-71	8
9	0-7	72-79	9
10	0-7	80-87	10
11	0-7	88-95	11
12	0-7	96-103	12
13	0-7	104-111	13
14	0-7	112-119	14
15	0-7	120-127	Modul Nummer 15

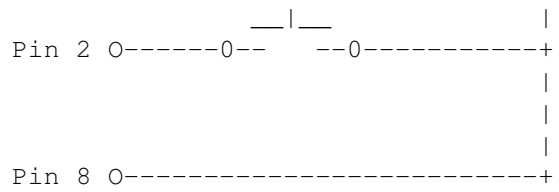
```
  16-19  0-31   128-352           Module 16 bis 19
  20-23  0-31   160-320           Module 20 bis 23
  24-27  0-31   192-288           Module 24 bis 27
  28-31  0-31   224-256           Module 28 bis 31
  32-35  0-31   256-224           Module 32 bis 35
  36-39  0-31   288-192           Module 36 bis 39
  40-43  0-31   320-160           Module 40 bis 43
  44-47  0-31   352-128           Module 44 bis 47
  48-51  0-31   384- 96           Module 48 bis 51
  52-55  0-31   416- 64           Module 52 bis 55
  56-59  0-31   448- 32           Module 56 bis 59
  60-61  0-15   480- 16           Module 60 und 61
  62     0-7    496-  8           Modul  62

------------------------------\
  63      0     504             \
  63      1     505              \
  63      2     506               \
  63      3     507                Modul Nummer 63
  63      4     508
  63      5     509               /
  63      6     510              /
  63      7     511             /
------------------------------/
```

Wenn nur ein Modul mit 8 Ausgabekanäle angeschlossen ist, so werden hier auf jeden Fall die Kanäle 0 bis 7 ausgegeben. Weitere Module müssen in Reihe zum ersten bzw. vorherigen geschaltet werden. Wenn bestimmte Kanäle nicht benötigt werden, so müssen diese "überbrückt" werden. Dies ist wichtig, damit nachfolgende Module die richtigen Kanäle wiedergeben.

1.44 Zeitmessung

Da ich für die Ein-Ausgabe mit meiner Library bereits einen CIA-Timer belegt habe, habe ich die nötigen Unterprogramme für eine Zeitmessung in die Library integriert. Diese Zeitmessung wird mit zwei Tastschalter am Gamport (Pin 1 und 2) gestartet und gestoppt. Die Tastschalter müssen zum starten bzw. stoppen die Anschlüsse auf Masse (GND; Pin 8) ziehen. Die Beschaltung des Gameports sieht dann etwa so aus:





Die Schalter könnten z.B. Lichtschranken sein. Dabei ist zu
beachten, daß die Lichtschranken bei unterbrechen des Licht-
strahles ihren Kontakt schließen. Damit hat man gleichzeitig
eine Funktionskontrolle. Wenn nämlich ständig beide Kontakte
geschlossen sind, sind die Lichtschranken nicht genau ausgerichtet.

Programmtechnisch kann eine Zeitmessung in etwa so durchgeführt
werden:

- Mit der Funktion "TIO_EnableRace" wird der Start freigegeben.d.h.
  der Zähler wird auf Null gesetzt, die Start-Lichtschranke wird
  ab jetzt abgefragt und eine Ampel zeigt grünes Licht (Kanal 0
  der TechnoIO-Hardware hat Hi-Pegel).

- Sobald die Start-Lichtschranke durchfahren wird, fängt der Zähler
  an zu zählen und die Ampel wird auf Rotlicht geschaltet. Außerdem
  werden alle Signale der Start-Lichtschranke ab jetzt ignoriert.

- Sobald die Ziel-Lichtschranke durchfahren wird, wird der Zähler
  angehalten.

Durch wiederholtes abfragen des Zählstatus mit "TIO_GetStatus" können
Sie erfahren, wann der Teilnehmer das Ziel erreicht hat.

Anmerkung: Mit der Funktion "TIO_DisableRace" kann eine Zeitmessung
           jederzeit unterbrochen werden.

- Abschließend kann der Zähler mit "TIO_ReadTimer" ausgelesen
  werden. Dieser Wert gibt die verstrichene Zeit zwischen den
  Lichtschranken in hundertstel Sekunden an und muss entsprechend
  umgerechnet werden.

1.45 Programmierer

Urheber des Programm-Paketes "TechnoIO" und Inhaber aller Rechte an
der in dieser Anleitung beschrieben Hard- und Software ist:

Jürgen Reinert
Am Kirchberg 4
D-31275 Lehrte

Tel   : 05175-3972
E-Mail: AC-Techno@T-Online.de

```
Schriftliche Anfragen werden nur beantwortet,wenn ein ausreichend
frankierter Rückumschlag beigelegt wird. Bitte haben Sie dafür
Verständnis.
```