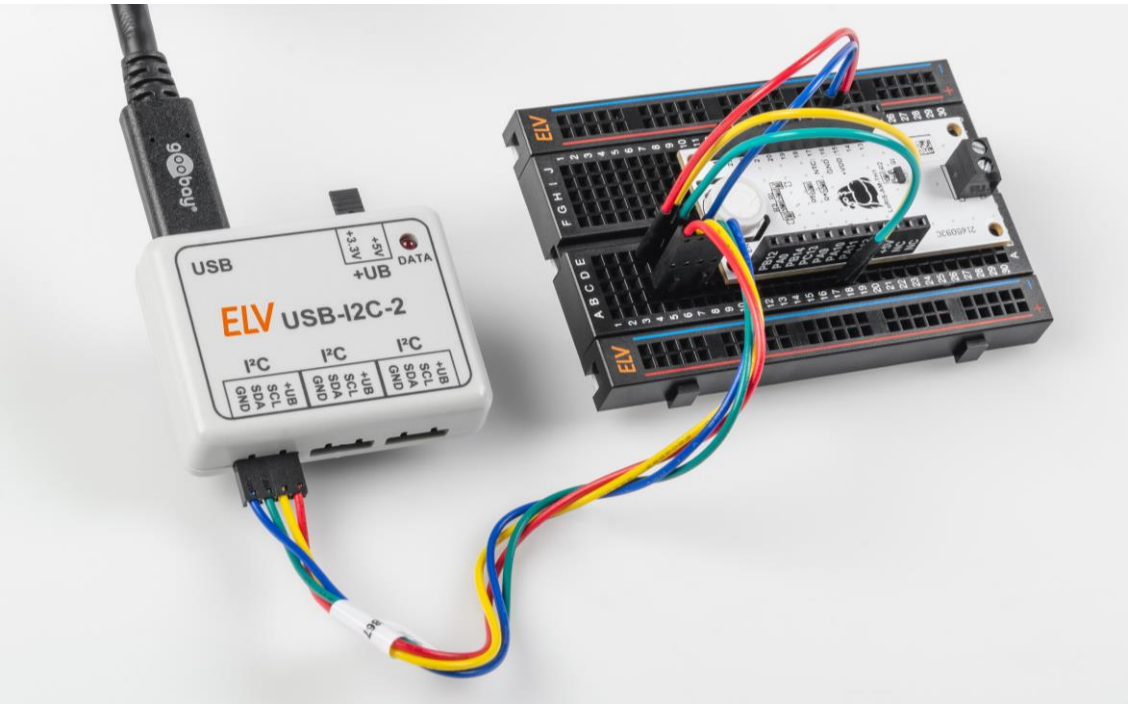


USB-I²C-Interface

Dokumentation und Anwendungsbeispiele für USB-I2C und USB-I2C-2

Stand:	April 2024
ELV Artikel-Nr.:	160452
Firmware-Version:	1.7

ELV Elektronik AG · Maiburger Straße 29-36 · 26789 Leer · Germany
Telefon 0491/6008-88 • Telefax 0491/6008-7016 • www.elv.com

Inhaltsverzeichnis

1. Einleitung

Diese Dokumentation beinhaltet zusätzliche Informationen zur Bedienungs-/Bauanleitung der USB-I²C-Interface USB-I2C und USB-I2C-2. Neben einer aktualisierten Auflistung aller Steuer- und Konfigurationsbefehle, werden diese detailliert beschrieben und ihre Verwendung anhand vieler einfacher Beispiele demonstriert.

Zusätzliche Hinweise weisen auf Besonderheiten und häufige Fehlerquellen hin, deren Beachtung eine lange Fehlersuche ersparen kann.

Im Anschluss an die Befehlsdokumentation werden eine Reihe konkreter Anwendungsbeispiele gegeben.

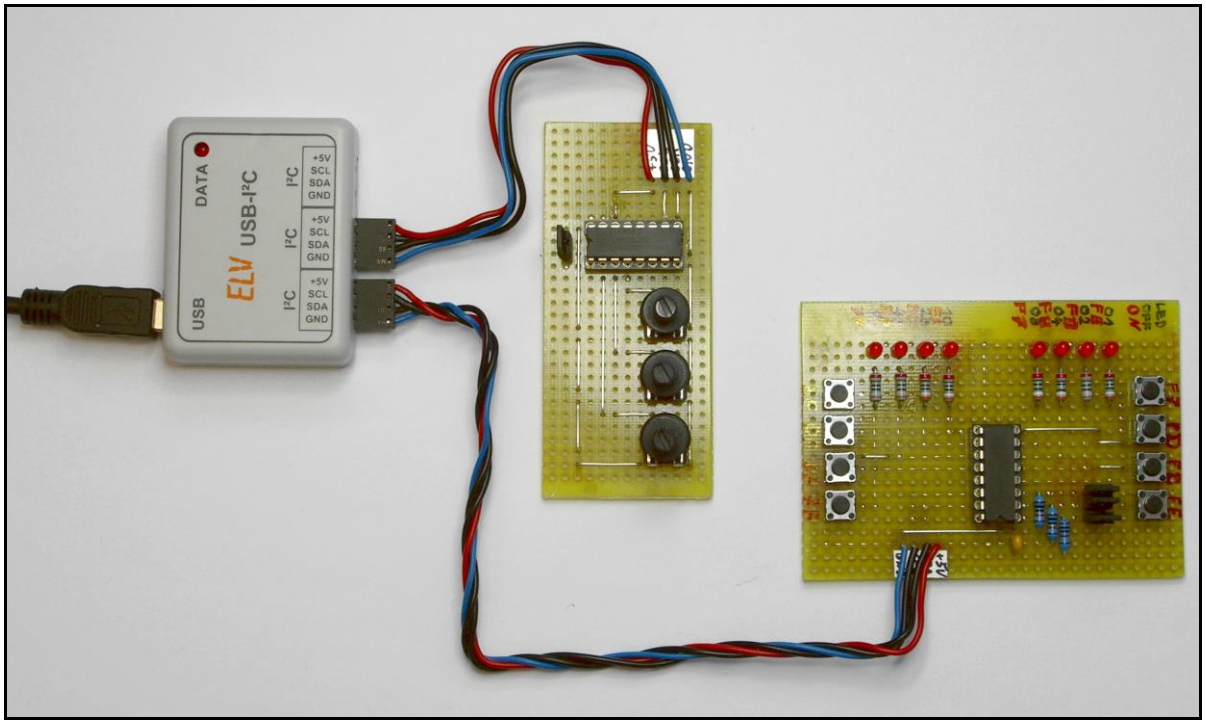


Bild: USB-I²C-Interface mit PCF8591-A/D-Wandler- und PCF8574 I/O-Interface-Testplatine

2. Installation des USB-I²C-Interface

Für die Verbindung des USB-I²C-Interfaces mit dem Computer wird im Interface der USB-Baustein „CP2102N“ der Firma Silicon Labs (SiLabs) verwendet. Für diesen Chip bietet SiLabs passende Windows-, Linux- und Macintosh-Treiber an, die entweder über die Produktseite des USB-I²C-Interfaces im Bereich Downloads oder direkt auf der Webseite von SiLabs erhältlich sind:
https://www.silabs.com/interface/usb-bridges/usbxpress/device.cp2102n-gqfn20?tab=softwareandtools

Aktuell (Stand: April 2024) sind das die folgenden “VCP Driver Kit” Versionen:

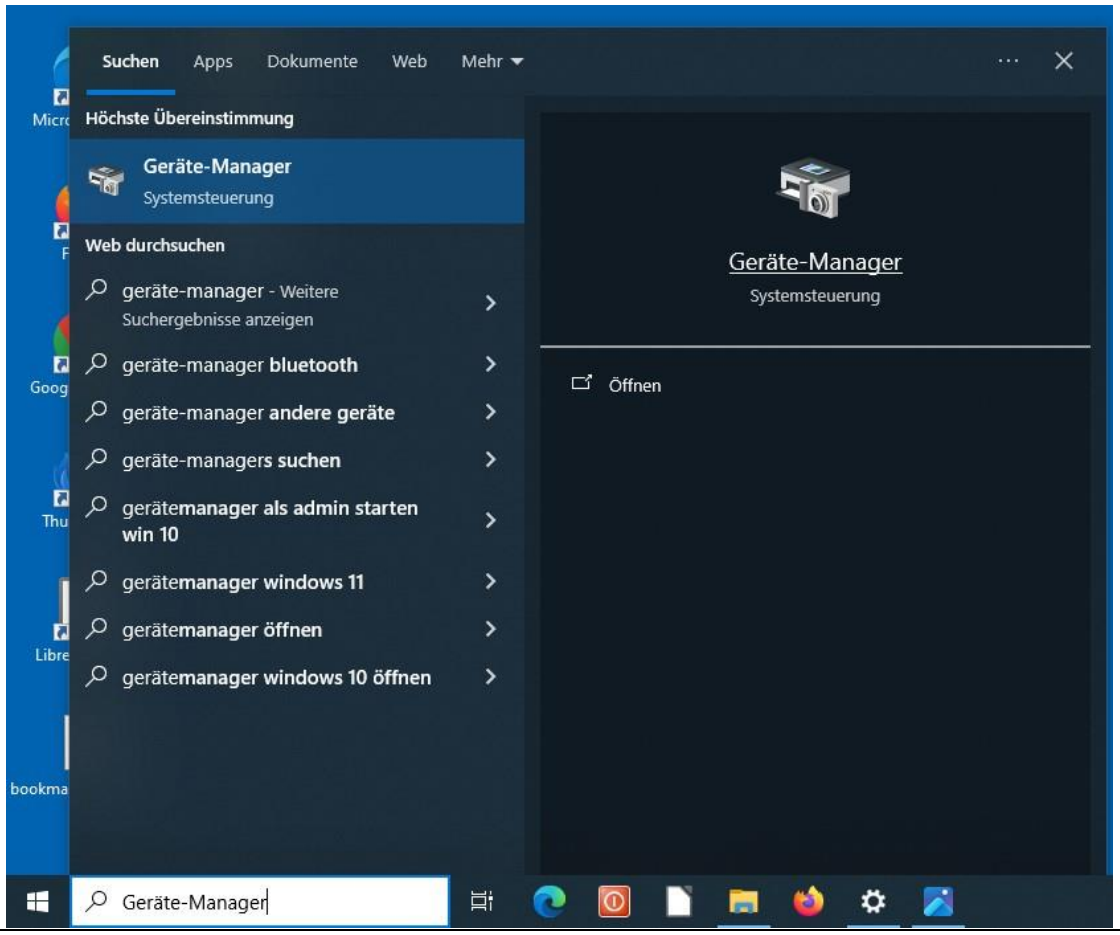
- Windows (Version 11.3.0)
- Macintosh OSX (Version 6.7.4 / 6.0.2)
- Linux (Version 5.x.x)

2.1. Installation des SiLabs-VCP-Treibers unter Windows

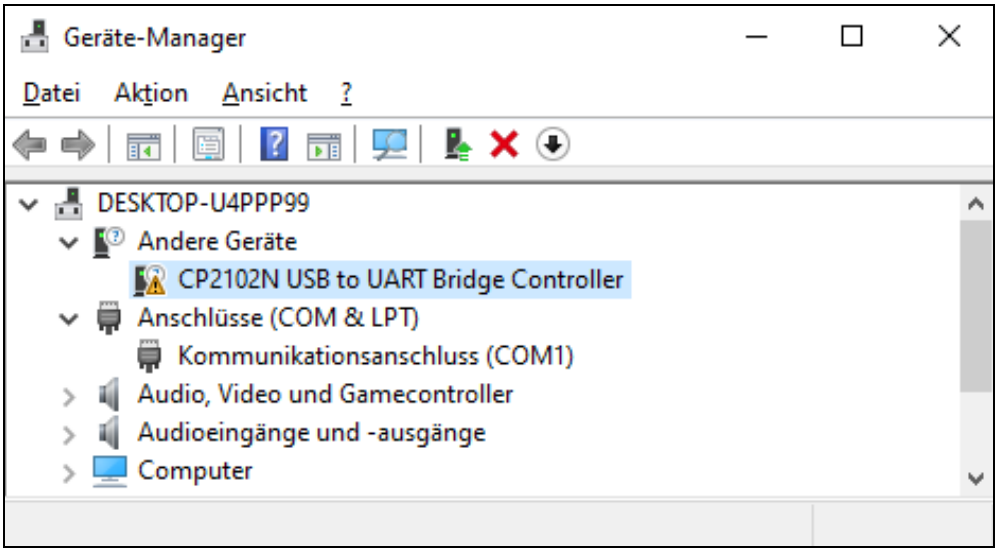
Bei älteren Windows-Versionen (vor Windows 10) ist eine manuelle Installation des Treibers aus den heruntergeladenen und entpackten **CP210x Windows Drivers** über die passende VCP-Installer-exe vor dem Anstecken des Interface am sinnvollsten. Nach der Installation des Treibers erfolgt der Anschluss an den PC gemäß Punkt 2.2.

Bei aktuellen Windows-Versionen ab Windows 10 kann das Interface direkt an den Computer angeschlossen werden und der Treiber aus dem heruntergeladenen und entpackten **CP210x Universal Windows Driver** gemäß nachfolgender Schritte installiert werden. Anschließend kann mit Punkt 2.3. fortgefahren werden.

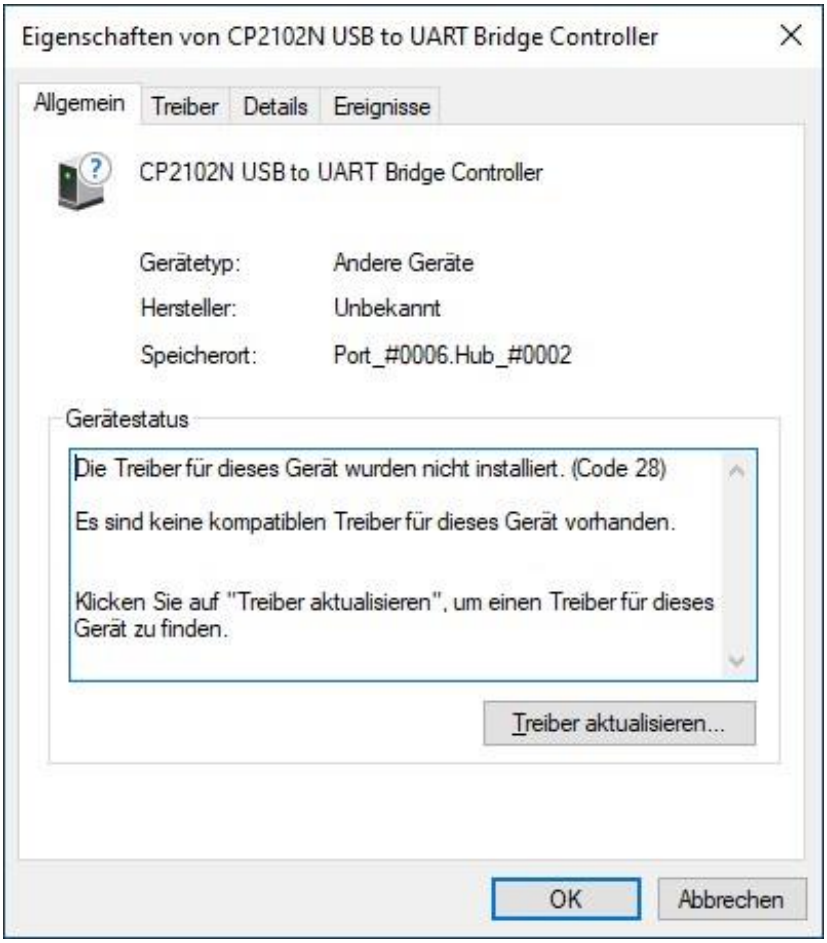
a) Aufruf des Geräte-Managers



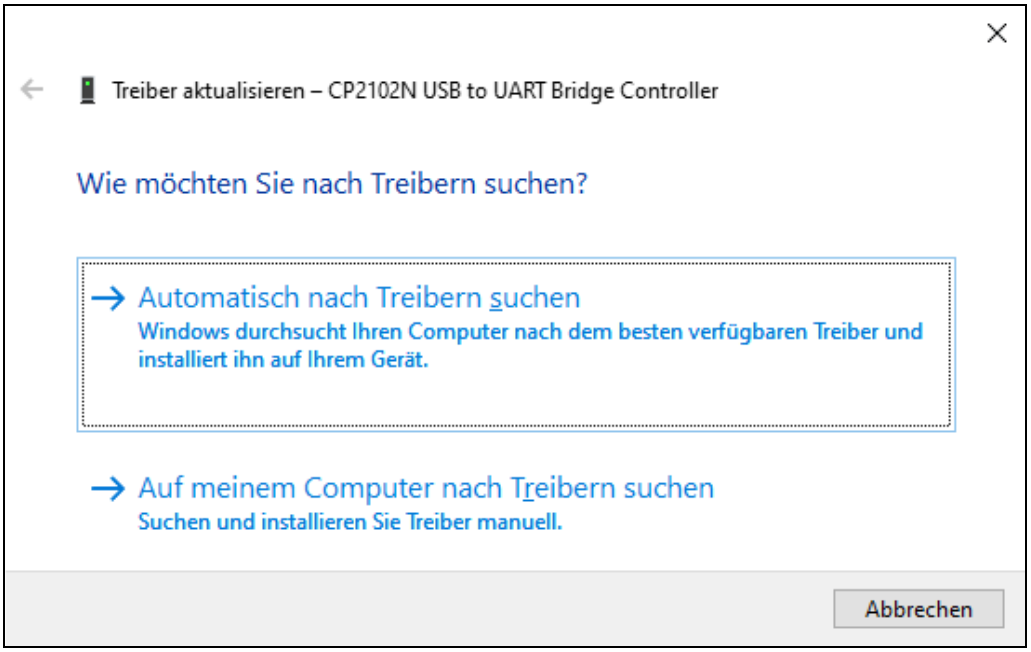
b) Auswählen und Öffnen des CP2102N Eintrags



c) Treiber aktualisieren auswählen

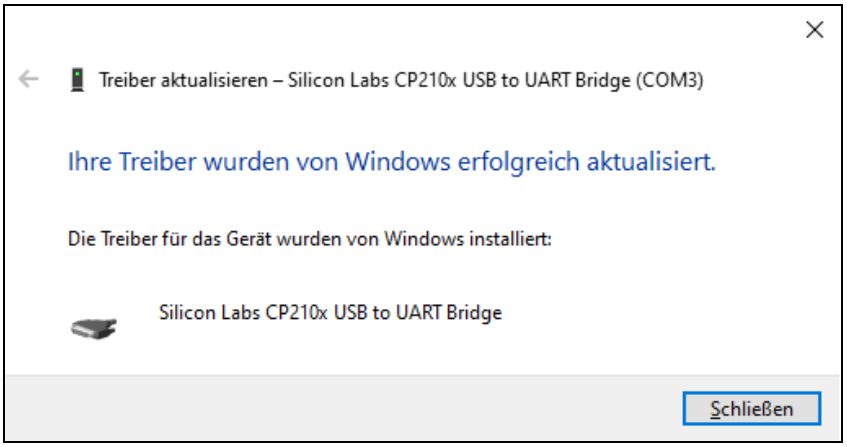


d) Auswählen von „Auf meinem Computer nach Treibern suchen“

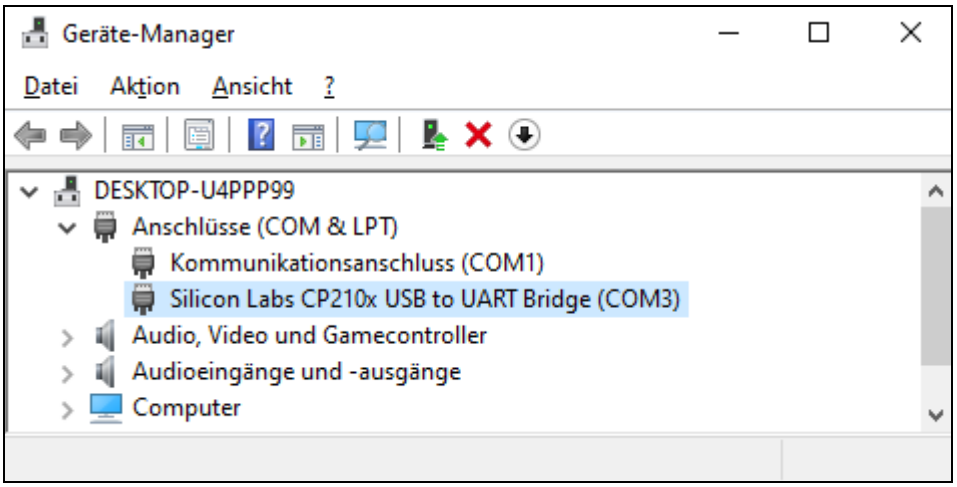


e) Den Ordner mit dem entpackten Treiberpaket auswählen

f) Beenden der Installation



g) Fertig eingerichteter virtueller COM-Port



2.2. Anschluss des USB-I²C-Interface am PC

Erst **nach der erfolgreichen Installation des SiLabs-VCP-Treibers** wird das USB-I²C-Interface über das mitgelieferte USB-Kabel an den PC angeschlossen!

Windows sollte nun das USB-I²C-Interface automatisch erkennen und den Treiber einrichten. Dies dauert eine Weile und wird im Erfolgsfall mit der folgenden Meldung bestätigt.

Damit ist die Installation des USB-I²C-Interface abgeschlossen.

Sollte sich nach dem Anschließen des USB-I²C-Interface der „**Assistent für das Suchen neuer Hardware**“ öffnen und nach einer Software fragen, obwohl zuvor der SiLabs-VCP-Treiber installiert worden ist, so sollte man die Option „**Software automatisch installieren (empfohlen)**“ auswählen und dem Assistenten folgen. Im Normalfall sollte das Einrichten aber automatisch erfolgen.

Das USB-I²C-Interface ist hiermit sofort betriebsbereit und kann über ein geeignetes Programm (siehe Kapitel 3.0.) sofort verwendet werden.

2.3. Ändern und Abfragen der COM-Port-Nummer

Bei der Installation weist Windows dem USB-I²C-Interface automatisch einen freien COM-Port zu.
Aber welchen?

Je nach Anzahl der vorhandenen seriellen Schnittstellen sind COM-Port 1 und 2 bei PCs fast immer fest vergeben. Bei vielen PCs ist daher erst der COM-Port 3 frei verfügbar und kann so dem USB-I²C-Interface zugewiesen werden. Verfügt der PC allerdings über einen Bluetooth-Adapter, so sind meist weitere COM-Ports an diesen vergeben.

Welcher Port nun tatsächlich dem USB-I²C-Interface zugewiesen wurde, lässt sich sehr einfach im Gerätemanager überprüfen und dort gegebenenfalls sogar ändern. Dazu sind diese Schritte wie in der Abbildung demonstriert durchzuführen:

a) Geräte-Manger gemäß 2.1. a) aufrufen
b) Unter „“**Anschlüsse (COM und LPT)**“ findet man nun den Eintrag:
„**Silicon Labs CP210x USB to UART Bridge (COMxx)**“ Hier steht der zugewiesene COM-Port.
Möchte man diesen ändern, so kann man durch einen Doppelklick auf diesen Eintrag zu den „Eigenschaften von Silicon Labs...“ gelangen.
c) Die hier stehenden Einstellungen „Bit pro Sekunde, Datenbits, Parität, Stoppbits und Flussteuerung“ sollte man ignorieren, da diese keinen Einfluss auf die spätere Schnittstellenkonfiguration haben. Gleichzeitig findet sich dort aber der Button „**Erweitert...**“ über den man zur eigentlichen Auswahl gelangt.
d) Unter „**COM-Anschlussnummer**“ kann dem USB-I²C-Interface nun ein freier Port zugewiesen werden. Dabei ist noch zu beachten, dass manche Terminalprogramme COM-Ports mit hohen Nummern nicht ansprechen können („HTerm“ hat damit aber kein Problem).

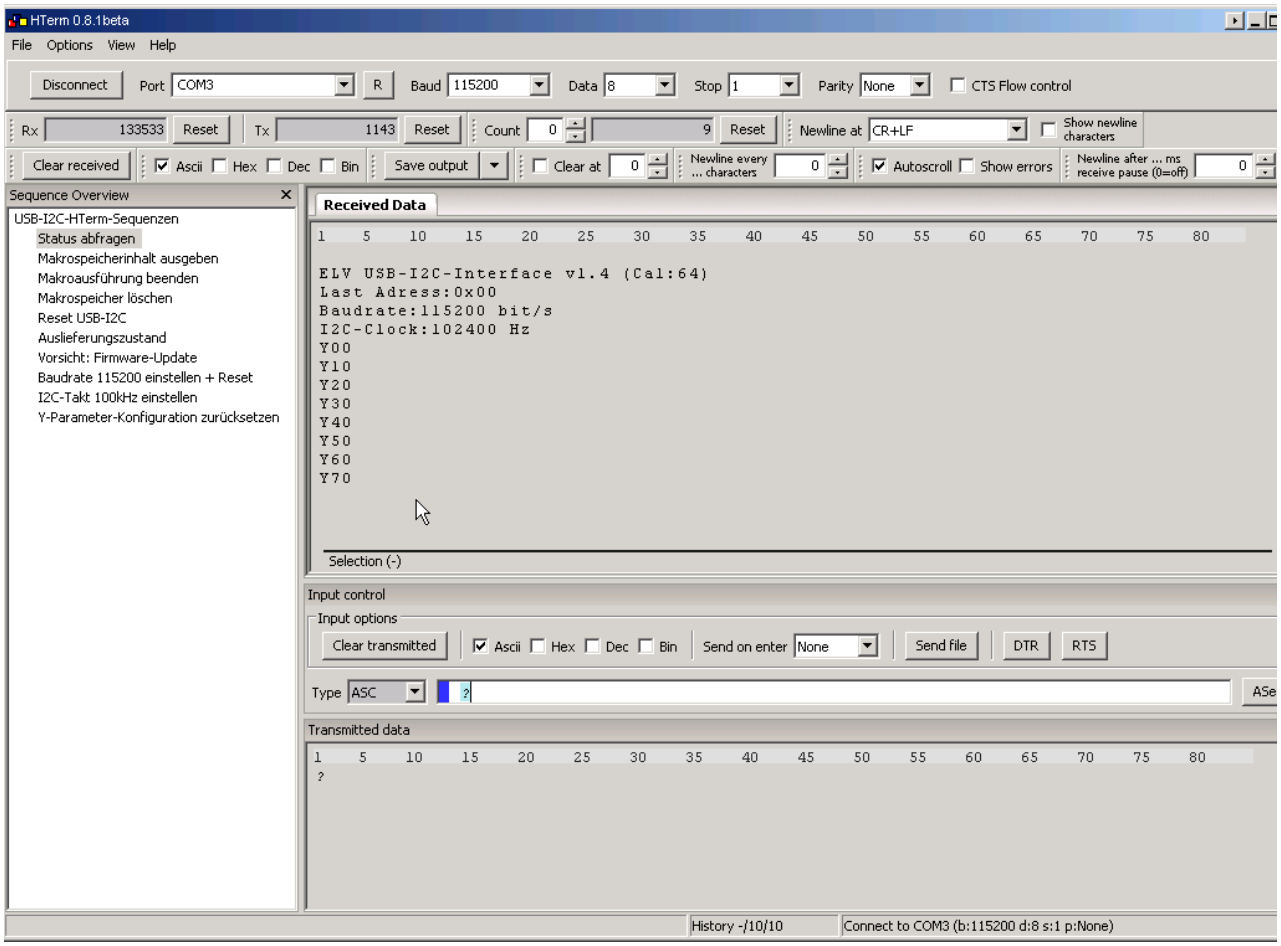
3. Programme zur Kommunikation (Terminalprogramm)

Für die Kommunikation mit dem USB-I²C-Interface können beliebige Terminal-Programme oder eigene Software-Entwicklungen verwendet werden, da die Kommunikation über einen Standard-COM-Port geschieht.

Beispielsweise kann das sehr gute Terminalprogramme „**HTerm**“ verwendet werden. Das von Tobias Hammer entwickelte Programm ist unter dem folgenden Link **kostenfrei** (auch für kommerzielle Nutzung) als Download erhältlich:
http://www.der-hammer.info/terminal/

Es existiert sowohl eine **Windows**- als auch eine **Linux-Version**.

Die im Weiteren beschriebenen Beispiele wurden unter Zuhilfenahme dieses sehr empfehlenswerten Programms erstellt und getestet.



Ab der Version 0.8.0 vom 12.11.2008 ermöglicht „HTerm“ das Erstellen, Laden und Speichern einer sogenannten Sequenz-Datei, in der ganze Befehls-Sequenzen gespeichert und durch einen Doppelklick gesendet werden können.
Dargestellt werden die Sequenzen im linken weißen Feld „Sequence Overview“. Das Speichern und Laden von Sequenz-Einstellungen erfolgt durch einen rechten Mausklick innerhalb des Sequenz-Feldes.

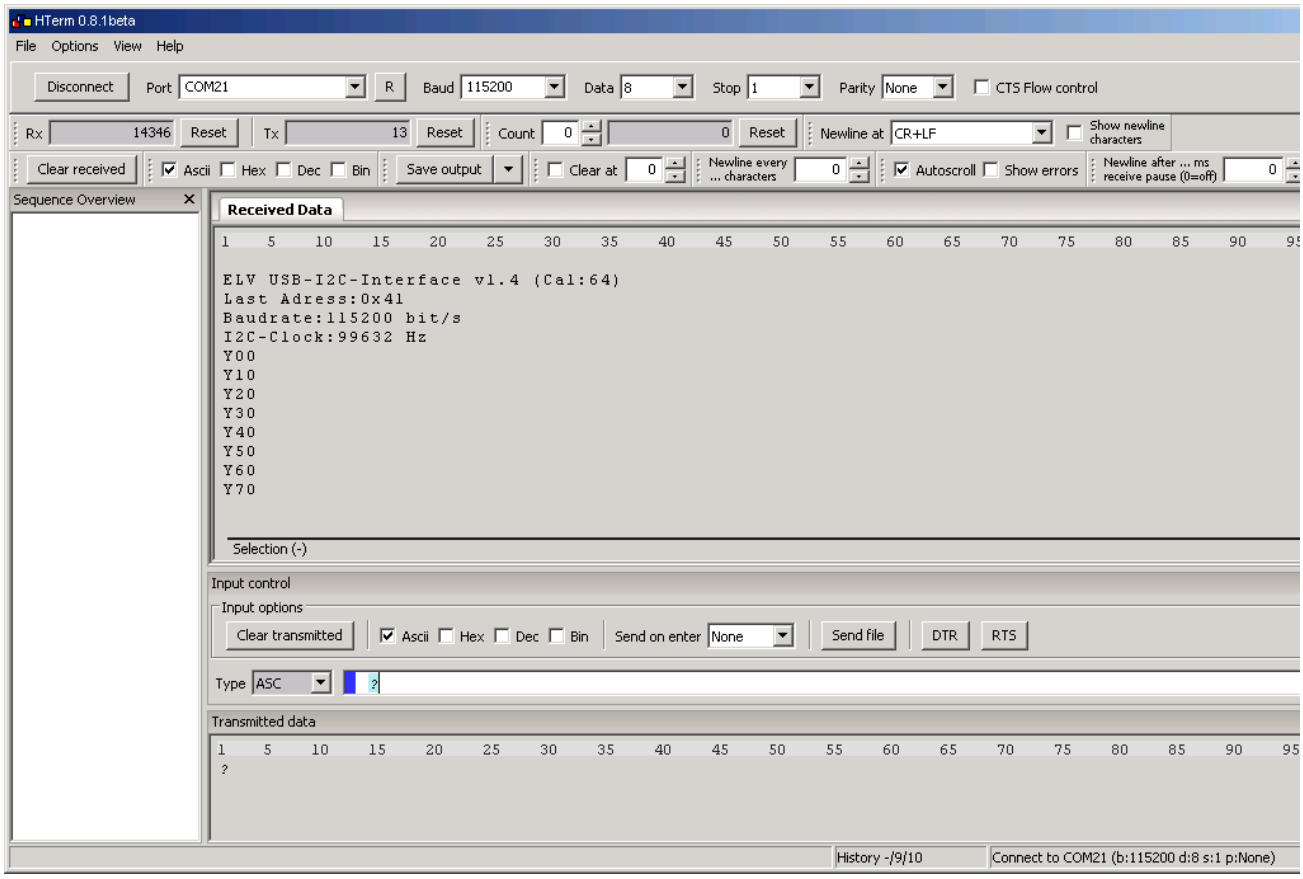
Zum Senden einer einmal eingegebenen Sequenz muss man lediglich den zugehörigen Namen im linken Feld anklicken.

Eine solche Sequenz-Datei („USB-I2C-HTerm-Sequenzen.hts“) für HTerm mit einigen Beispiels-Sequenzen für das USB-I²C-Interface findet sich im Download-Verzeichnis zum USB-I²C-Interface auf www.elv.de.

3.1. Bedienung des USB-I²C-Interface über „HTerm“

Die folgende Beschreibung geht davon aus, dass sich die Einstellungen des USB-I²C-Interface noch im Auslieferungszustand befinden – dies gilt insbesondere für die eingestellte Baudrate.
Die Einstellungen finden sich auch im folgenden Bildschirmprint wieder und können so direkt übernommen werden.

a) Richtigen **COM-Port** auswählen (siehe auch Kapitel 2.3.)
b) Baud: **115200**
c) Data: **8**
d) Stop: **1**
e) Parity: **None**
f) Newline at: **CR+LF**
g) Häkchen löschen neben: „Show newline characters“
h) Häkchen neben „**Ascii**“ setzen und neben „Hex“, „Dec“ und „“Bin“ löschen (sowohl oben über dem „Received Data“-Fenster als auch unten über dem „Transmit Data“-Fenster)
i) Häkchen neben „**Autoscroll**“ setzen
j) Unten neben dem Eingabefeld für die zu sendenden Daten muss als Type „**ASC**“ ausgewählt sein.
k) Oben links auf „**Connect**“ drücken (die Beschriftung des Buttons sollte danach auf Disconnect“ wechseln, erst dann ist der COM-Port geöffnet)
l) Wird im Eingabefeld unten jetzt ein Fragezeichen (**?)** eingegeben und per Eingabetaste gesendet, so antwortet das USB-I²C-Interface mit seiner aktuell eingestellten Konfiguration (siehe Kapitel 4.2.1.).



Alle Anweisungen, die an das USB-I²C-Interface gesendet werden, sind in das untere Eingabefeld einzutippen. Die Antworten vom USB-I²C-Interface erhält man im oberen „Received Data“ Fenster.

Viele Beispiele aus dieser Dokumentation können zum Testen direkt über die Windows-Zwischenablage kopiert und ins „HTerm“-Eingabefeld eingefügt werden. So spart man sich das Abtippen der Anweisungsfolgen (beachten sie dabei den max. 400 Zeichen großen Empfangspuffer des USB-I²C-Interface).

4. Befehle und Einstellungen des USB-I²C-Interface

In den folgenden 3 Tabellen sind alle verfügbaren Befehle aufgelistet, mit denen man das USB-I²C-Interface ansprechen und verwenden kann.

In Tabelle 1 finden sich die wichtigsten Befehle, die für die I²C-Kommunikation mit den am Interface angeschlossenen Geräten sind. Für die richtige Verwendung sind Grundkenntnisse über die I²C-Schnittstelle und den Ablauf der Kommunikation sehr nützlich. Infos dazu finden sich beispielsweise in der I2C-Bauanleitung.

Anhand der folgenden Befehlsbeschreibungen und den Beispielen in diesem Dokument stellt sich aber sehr schnell der Erfolg ein, da die ganze Kommunikation über den I²C-Bus relativ einfach ist.

Tabelle 1: Befehle für die I²C-Kommunikation

ASCII-Zeichen	Folgebyte(s)*	Funktions-Beschreibung
S		initiiert Start-Ereignis auf I²C-Bus
	7-Bit-Adresse + Write-Bit **(0)** + Datenbyte(s)	nachfolgende Daten ins adressierte I²C-Gerät schreiben (geringstwertiges Bit [LSB] des Adressbytes muss 0 sein)
	7-Bit-Adresse + Read-Bit **(1)** + Byteanzahl**	Daten aus dem adressierten I²C-Gerät lesen, dem Adressbyte folgt die Anzahl der zu lesenden Bytes (geringstwertiges Bit [LSB] des Adressbytes muss 1 sein)
P		initiiert Stopp-Ereignis auf I²C-Bus (Bus im Leerlauf = idle)
W	Byte1 Byte2 Byte3...	schreibt „Byte1, Byte2, Byte3..." ins zuletzt adressierte I²C-Gerät
R	Byteanzahl**	liest Datenbytes (1...255) aus dem zuletzt adressierten I²C-Gerät
:		wartet mit der Ausführung der nachfolgenden Befehle bis zum nächsten Zeilenumbruch (0x0D oder 0x0A); ist sinnvoll, wenn das Terminal-Programm jedes Zeichen sofort nach der Eingabe überträgt; Ausführung erst nach Abschluss mit Eingabetaste
L	Byte1 Byte2	fügt eine Wartepause von 1 bis 65.535 ms (0001...FFFF) in Hex-Schreibweise (16 Bit) in die Befehlsausführung ein (z. B. innerhalb von Makros)
N		lässt Master nach letztem gelesenen Byte mit NACK antworten, wenn die automatische NACK-Antwort mit Y21 deaktiviert wurde

* Jedes Byte (Hexadezimal) wird mit 2 ASCII-Zeichen geschrieben, z. B.: 0x1F = 1F.
** Wird beim Lesen als Byteanzahl 00 angegeben, erwartet das USB-I2C-Interface einen Pascal-String. Dieser macht es möglich, eine Zeichenkette mit variabler Länge vom Slave auszulesen. Definiert ist die Länge im ersten gelesenen Byte.

In Tabelle 2 finden sich spezielle Anweisungen, die einer übersichtlichen Befehlsdarstellung und einer sinnvollen Formatierung der Rückgabewerte (oder der Messergebnisse) vom USB-I²C-Interface dienen. Diese wenigen sehr einfachen Befehlszeichen stellen ein erstaunlich mächtiges Werkzeug dar, durch das die unterschiedlichsten Aufgaben und Datenformate möglich werden.

Hierfür finden sich im Folgenden auch einige Beispiele.

Tabelle 2: Kommentarbefehle für die Befehls- und Rückgabewerte

ASCII-Zeichen	Funktions-Beschreibung
.	ein Punkt bewirkt einen Zeilenumbruch (0x0D 0x0A) in der Rückgabe
,	fügt ein Komma in die Rückgabe ein (für kommagetrennte Daten für Excel o. Ä.)
;	fügt ein Semikolon in die Rückgabe ein (für semikolongetrennte Daten für Excel)
[...]	ASCII-Zeichen zwischen eckigen Klammern werden zum PC zurückgegeben –> zum Kommentieren von Rückgabewerten Mögliche Steuerzeichen: \r (Carriage Return), \n (Line Feed), \t (Horizontal Tab)
(...)	ASCII-Zeichen zwischen runden Klammern werden ignoriert –> zum Kommentieren von Befehlsanweisungen
Leerstelle	Leerstellen in den Anweisungen werden ignoriert (ausgenommen innerhalb von eckigen Klammern)

In der Tabelle 3 finden sich zum einen die Befehle, durch die das USB-I²C-Interface komplexe Makros ausführen kann und zum anderen finden sich hier alle Konfigurations-Parameter mit denen das USB-I²C-Interface für die unterschiedlichsten Anwendungen angepasst werden kann.

Tabelle 3: Befehlsübersicht zur Konfiguration des USB-I2C-Interfaces			
ASCII-Zeichen	**Folgebyte(s)*/Zeichen**		**Funktions-Beschreibung**
>	Makroadresse (1 Byte)		startet Makro-Ausführung an der Makro-Speicheradresse
<			beendet Makro-Ausführung und wartet auf neue Anweisungen vom PC
V	Makroadresse { Zeichen1 Zeichen2... }		schreibt die ASCII-Zeichen (Befehle und Daten) zwischen den geschweiften Klammern in den Makro-Speicher ab der übergebenen Makroadresse; Makrospeicher löschen mit: **V00{}** (Speicher wird mit Leerzeichen (Hex:0x20) überschrieben)
U			Inhalt des Makrospeichers (256 ASCII-Zeichen) wird vollständig ausgegeben (zum PC)
?			Systemstatus und Einstellungen werden ausgegeben (zum PC)
T	I²C-Taktrate (6 Zeichen)		I²C-Bustakt von min. 226 Hz bis max. 409,6 kHz (000226...409600), z. B.: 400000 = 400 kHz (Fast-Mode), **100000 = 100 kHz # (Standard-Mode)**, 000226 = 226 Hz (geringst mögliche Frequenz)
X	Baudrate (4 Zeichen)		COM-Port Baudrate: 0048 = 4800 bit/s, 0096 = 9600 bit/s, 0192 = 19.200 bit/s, 0384 = 38.400 bit/s, 0576 = 57.600 bit/s, 0768 = 76.800 bit/s, **1152 = 115.200 bit/s #**, 2304 = 230.400 bit/s (diese Einstellung wird erst nach einem Neustart wirksam)
Y	0	0 #	dem letzten Datenbyte, das der Master aus dem Slave liest, folgt ein Zeilenumbruch (0x0D 0x0A) in der Rückgabe zum PC
		1	Daten, die der Master aus dem Slave ausliest, folgt **kein Zeilenumbruch** in der Rückgabe zum PC
	1	0 #	Master **ignoriert** beim Schreiben die **NACK**-Antwort des Slaves (die bedeutet, dass der Slave keine weiteren Daten akzeptiert) und schreibt weiter
		1	Master stoppt Schreiben, wenn Slave mit NACK antwortet
	2	0 #	nachdem der Master das letzte Byte vom Slave gelesen hat, antwortet er mit NACK (dadurch weiß der Slave, dass keine weiteren Daten folgen)
		1	beim Lesen sendet Master **kein NACK** nach letztem Byte (in diesem Fall ist mit der N-Anweisung manuell NACK zu senden)
	3	0 #	ACK/NACK-Antworten des Slaves werden nicht zum PC übertragen
		1	beim Schreiben wird für jedes erhaltene **ACK** ein **K** zum PC übertragen und für ein **NACK** ein **N** (gut zum Debuggen)
	4	0 #	jeder Daten-Rückgabe (2 Zeichen) folgt ein Leerzeichen (0x20)
		1	einer Daten-Rückgabe zum PC folgt kein Leerzeichen
	5	0 #	nach einem Reset das Makro ausführen, wenn eines im Makrospeicher steht
		1	nach einem Reset kein Makro ausführen
	6	0 #	die Makroadresse (nach > und V) wird mit 2- und die Wartepause (nach L) wird mit 4-ASCII-Zeichen in Hexadezimalschreibweise angegeben
		1	die Makroadresse (nach > und V) wird mit 3- und die Wartepause (nach L) wird mit 5-ASCII-Zeichen in Dezimalschreibweise angegeben
	7	0 #	Gelesene Daten werden als Ascii-Zeichen im Hexadezimalformat zum PC gesendet .
		1	Gelesene Daten werden als Ascii-Zeichen im Dezimalformat (ohne führende 0) zum PC gesendet (ab v1.6)
Z	4B		startet das USB-I2C-Interface neu (Reset); der Inhalt des Makrospeichers bleibt erhalten
	AA		Konfiguration, Baudrate und Bustakt auf Auslieferungszustand zurücksetzen, Makrospeicher löschen und USB-I2C-Interface neu starten

* Jedes Byte (Hexadezimal) wird mit 2 ASCII-Zeichen geschrieben, z. B.: 0x1F = 1F # Standardwert im Auslieferungszustand

In den folgenden Kapiteln ist jeder Befehl einzeln mit Beispielen und weiteren Hinweisen genau beschrieben.

Allgemeines zu den Befehlen:

- Die Befehle können wahlweise in Großbuchstaben, Kleinbuchstaben oder in gemischter Schreibweise eingegeben werden. Dies hilft eventuell die Anweisungen übersichtlicher zu gestalten.
- Zwischen den Befehlen können Leerzeichen eingefügt werden. Befehle und Daten dürfen aber auch direkt aufeinander folgen – dies gilt auch für den Makrospeicher.
- Nicht in der Tabelle 3 aufgelistet: Wird der Befehl Z gefolgt von FF (also: **ZFF**) eingegeben, wird das USB-I²C-Interface in den Firmware-Update-Zustand versetzt (siehe Kapitel 4.2.4.).
- Neu ab Firmware v1.6 ist der Parameter Y71, der zum Umschalten des Ausgabeformats dient.

4.1. Befehle für die I²C-Kommunikation

Mit den in diesem Kapitel beschriebenen Befehlen aus der Tabelle 1, können Daten über den I²C-Bus geschrieben und gelesen werden. Zudem werden hier einige Zusatzbefehle (z.B. Wartepausen, verzögerte Ausführung) erläutert, die über die normale I²C-Kommunikation hinausgehen und dadurch weitaus komplexere Funktionen ermöglichen.

4.1.1. „S“ – Start-Bedingung

Wie im I²C-Grundlagen-Teil der Bauanleitung beschrieben, wird jeder Schreib-/Lesezugriff auf dem I²C-Bus durch die sogenannte Start-Bedingung eingeleitet. Dies geschieht beim USB-I²C-Interface durch das ASCII-Zeichen **S**. Der Startbedingung muss die Adresse des angesprochenen I²C-Gerätes folgen, die gleichzeitig das Read/Write-Bit (LSB) enthält. Die Adresse wird Hexadezimal mit 2 Zeichen (also z. B. **41** oder **40**) angegeben.

Beispiele:
Schreibe das Daten-Byte 0xFF in das Gerät mit der Geräte-Adresse 0100 000X (binär).
Bildung der Adresse: 0100 000X + Schreib-Bit (0) = 0100 0000 (binär) = 40 (hexadezimal)

[Start] [Write-Adr:0x40] [Daten:0xFF] [Stopp]
```
S 40 FF P
```

Gleichwertig sind auch die folgenden Schreibweisen in Groß/Kleinschreibung und mit/ohne Leerzeichen:
```
S40FFP
S 40 ff p
s40 ff P
```

Sollen **mehrere Daten in denselben Slave** geschrieben werden, geht das beispielsweise so:
[Start] [Write-Adr:0x40] [Daten:0xFF] [Stopp] [Start] [Write-Adr:0x40] [Daten:0x0F] [Stopp] [Start] [Write-Adr:0x40] [Daten:0xF0] [Stopp]
```
S40 FF P S40 0F P S40 F0 P
```

Hinweis 1:
Im Auslieferungszustand gibt das USB-I²C-Interface bei Schreibzugriffen (wie in den Beispielen) keine Antwort zum PC zurück. Möchte man aber wissen, ob die Schreibanweisungen erfolgreich waren, kann man sich die Acknowledge/Not-Acknowledge Antworten (ACK/NACK) vom Slave zum PC weiterleiten lassen. Dies geschieht mit Hilfe des Konfigurationsbefehls **Y31** (ACK=K, NACK=N, siehe Kapitel 4.2.5).
Hinweis 2:
Strenggenommen muss das **P** nicht am Ende jeder Übertragung verwendet werden. Ohne die Stopp-Bedingung wird der I²C-Bus nicht wieder freigegeben, was nicht immer nötig ist. Für den Abschluss einer Übertragung wird das **P** aber empfohlen.

Möglich ist es aber auch, die Daten ohne eingefügte Stopp-Bedingungen zum Slave zu schreiben.
Dafür dient die **Repeated-Start-Bedingung** (ein erneutes **S**, ohne dass zuvor ein **P** kam):
[I²C-Start] [Write-Adr:0x40] [Daten:0xFF] [Start] [Write-Adr:0x40] [Daten:0x0F] [Start] [Write-Adr:0x40] [Daten:0xF0] [Stopp]
```
S40 FF S40 0F S40 F0 P
```

Wenn alle Daten in denselben Slave geschrieben werden, können auch **mehrere Datenbytes direkt nacheinander** eingegeben werden. Dies verkürzt die Anzahl der zu übertragenen Zeichen:
[Start] [Write-Adr:0x40] [Daten:0xFF, 0x0F, 0xF0, 0x00, 0x10, 0x46, 0x3A] [Stopp]
```
S40 FF 0F F0 00 10 46 3A P
```
Oder noch kürzer ohne Leerzeichen:
```
S40FF0FF00010463AP
```

Auch eine **Lese-Anweisung** wird entsprechend den obigen Beispielen mit dem Start-Befehl eingeleitet. In diesem Fall folgt der Geräte-Adresse jedoch die Anzahl der vom Slave zu lesenden Datenbytes. Die ausgelesenen Datenbytes werden vom USB-I²C-Interface direkt zum PC weitergeleitet. Die Bytes werden dabei in Hexadezimalschreibweise in Form von 2 ASCII-Zeichen ausgegeben (z. B. wird der Dezimalwert 254 als FE ausgegeben).

Lese 11 Bytes vom Gerät mit der Geräte-Adresse 0100 000X (binär).
[Bildung der Adresse: 0100 000X + Lese-Bit (1) = 0100 0001 (binär) = 41 (hexadezimal)]
[Start] [Read-Adr:0x41] [Byteanzahl] [Stop]
```
S41 0A P
```

Die gelesenen Daten könnten z.B. so aussehen (abhängig vom I²C-Gerät und dessen Zustand):
```
FF FF FF FF FF FF FF FF FF FF FF
```

Eine weitere Möglichkeit Daten zu schreiben und zu lesen, bieten die Befehle **W** und **R** (siehe Kapitel 4.3./4.4.).

4.1.2. „P“ – Stopp-Bedingung

Laut dem I²C-Protokoll werden Schreib-/Lesezugriff auf dem Bus durch die sogenannte Stopp-Bedingung beendet. Dies geschieht beim USB-I²C-Interface durch das ASCII-Zeichen **P**.

Mit der Stopp-Bedingung wird der Bus wieder freigegeben, damit ein zweiter Master auf den Bus zugreifen kann. (Das USB-I²C-Interface ist aber nicht für den Multi-Master-Betrieb entwickelt worden.)

Beispiele für die Verwendung von **P** finden sich unter 4.1.1.

Hinweis:
Im Falle einer **I²C-Verklemmung** (Deadlock, siehe Kapitel 4.6.) kann der Master die Stopp-Bedingung nicht mehr ausführen. In dem Fall ist eine weitere Kommunikation mit angeschlossenen Slaves nicht mehr möglich. Um dies zu verhindern, versucht das USB-I²C-Interface eine erkannte Verklemmung automatisch wieder aufzulösen. Dazu gibt es 8 zusätzliche Takt-Pulse auf die SCL-Leitung, sobald die Stopp-Bedingung nicht ausführbar sein sollte (also wenn die SDA-Leitung vom Slave auf Low-Pegel gehalten wird).

4.1.3. „W“ – Schreibe Daten an zuletzt adressierten Slave

Der Schreib-Befehl **W** (Write) ermöglicht eine verkürzte Schreibweise. Er kann erst verwendet werden, nachdem die Startbedingung **S** und die Geräte-Adresse mindestens einmal übertragen worden sind.
Der Befehl **W** lässt das USB-I²C-Interface eine Start-Bedingung auf dem I²C-Bus ausführen und die nachfolgend eingegebenen Daten an das zuletzt adressierte Gerät schreiben. Die Geräte-Adresse wird beim **W**-Befehl also nicht erneut eingegeben.

Beispiele:
Es sollen 3 Datenbytes in den Slave geschrieben werden.
Statt nun das Folgende zu senden...
[Start] [Write-Adr:0x40] [Daten: 0xFF] [Start] [Write-Adr:0x40] [Daten:0x0F] [Stopp] [Start] [Write-Adr:0x40] [Daten:0xF0] [Stopp]
```
S40 FF  S40 0F P  S40 F0  P
```

...ist mit dem **W**-Befehl eine kürzere Schreibweise möglich, die folgendermaßen aussieht (der **W**-Befehl funktioniert sowohl mit als auch ohne vorherige Stopp-Bedingung):
[Start] [Write-Adr:0x40] [Daten:0xFF] [Write] [Daten:0x0F] [Stopp] [Write] [Daten:0xF0] [Stopp]
```
S40 FF  W 0F P  W F0  P
```

Auch mit dem **W**-Befehl können **mehrere Daten direkt nacheinander** (ohne Repeated-Start) in den Slave geschrieben werden:
```
S40 00 01 02 03  W A0 A1 A2 A3 A4  P
```

In diesem Beispiel mag die Verwendung des **W**-Befehls noch nicht sinnvoll erscheinen, da die dem **W**-Befehl folgenden Daten ja auch ohne den **W**-Befehl direkt den ersten 4 Datenbytes folgen könnten. Dies ändert sich allerdings, sobald abwechseln Daten zum Slave geschrieben und gelesen werden sollen. In dem Fall müsste jedes Mal die Start-Bedingung plus Schreibadresse (z.B. **S40**) und dann wieder die Start-Bedingung plus Leseadresse (z.B. **S41**) eingegeben werden. Dies ist mit den **W**- und **R**-Befehlen nicht nötig und geht z.B. folgendermaßen:

Abwechselndes Schreiben und Lesen von Daten:
[Start] [Write-Adr:0x40] [Daten:0xFF] [Read] [Byteanzahl] [Write] [Daten:0xA0] [Stopp]
```
S40 FF R 01 W A0 P
```

Die mit der Start-Bedingung **S** zu Beginn einmal übertragene Geräte-Adresse unterscheidet sich zwar je nachdem ob Daten geschrieben (z.B. **S40**) oder gelesen (z.B. **S41**) werden sollen, was aber für die nachfolgenden Schreib- (**W**) und Lese-Befehle (**R**) egal ist, da das USB-I²C-Interface bei diesen Befehlen die Geräte-Adresse automatisch richtig verwendet.

Hinweis:
Welche Geräte-Adresse zuletzt verwendet wurde, lässt sich mit dem Status-Befehl (**?**) prüfen.

4.1.4. „R“ – Lese Daten vom zuletzt adressierten Slave

Der Lese-Befehl **R** (Read) ermöglicht eine verkürzte Schreibweise. Er kann erst verwendet werden, nachdem die Startbedingung **S** und die Geräte-Adresse mindestens einmal übertragen worden sind. Der Befehl **R** lässt das USB-I²C-Interface eine Start-Bedingung auf dem I²C-Bus ausführen und liest die nachfolgend angegebene Anzahl von Bytes aus dem zuletzt adressierten Gerät aus. Die Geräte-Adresse wird beim **R**-Befehl also nicht erneut eingegeben.

Beispiele:
Es sollen dreimal nacheinander je 5 Datenbytes aus dem Slave ausgelesen werden.
Statt nun das Folgende zu senden...
[Start] [Read-Adr:0x41] [Bytezahl:0x05] [Start] [Read-Adr:0x41] [Bytezahl:0x06] [Stopp] [Start] [Read-Adr:0x41] [Bytezahl:0x07] [Stopp]
S41 05 S41 06 P S41 07 P

...ist mit dem **R**-Befehl eine kürzere Schreibweise möglich, die folgendermaßen aussieht (der **R**-Befehl funktioniert sowohl mit als auch ohne vorherige Stopp-Bedingung):
[Start] [Read-Adr: 0x41] [Byteanzahl: 0x05] [Read] [Byteanzahl: 0x06] [Stopp] [Read] [Byteanzahl: 0x07] [Stopp]
S41 05 R06 P R07 P

Die Ausgabe der gelesenen Daten zum PC ist in beiden Fällen exakt gleich und könnte z.B. so aussehen:

```
FF FF FF FF FF
FF FF FF FF FF FF
FF FF FF FF FF FF FF
```

Hinweis 1:
Die übersichtliche Formatierung der Rückgabewerte mit Leerzeichen zwischen den Einzelwerten und mit einem Zeilenumbruch nach Abschluss jedes Lese-Befehls, erfolgt vollautomatisch durch das USB-I²C-Interface (Konfiguration im Auslieferungszustand).
Möchte man beispielsweise den Datenverkehr reduzieren und auf die Leerzeichen verzichten, so ist dies mit dem Konfigurationsbefehl **Y41** möglich (siehe Kapitel 4.2.5.).
Möchte man zudem die automatischen Zeilenumbrüche abschalten, so ist dies mit dem Konfigurationsbefehl **Y01** möglich (siehe Kapitel 4.2.5.).

Hinweis 2:
Welche Geräte-Adresse zuletzt verwendet wurde, lässt sich mit dem Status-Befehl (**?**) prüfen.

4.1.5. „:“ – Warte mit Ausführung bis Zeilenumbruch

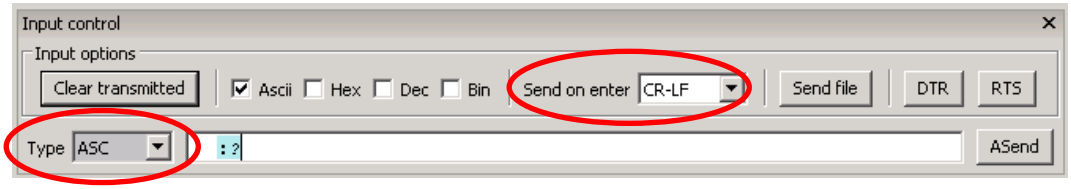
Der Doppelpunkt-Befehl (**:**) wird nur dann benötigt, wenn eine Befehlskette erst vollständig in den Eingabepuffer des USB-I²C-Interface übernommen werden soll, bevor das Interface deren Ausführung startet.
Sobald das USB-I²C-Interface einen Doppelpunkt empfangen hat, wartet es solange mit der Ausführung aller nachfolgenden Befehle, bis es entweder das Steuerzeichen für Zeilenvorschub (Line Feed = LF = 0x0A) oder für Wagenrücklauf (Carriage Return = CR = 0x0D) empfängt. Diese Zeichen entsprechen dem Drücken der Eingabe/Enter-Taste.

Dies kann sinnvoll sein, wenn ein Terminalprogramm verwendet wird, das jedes eingegebene Zeichen sofort überträgt. In dem Fall wäre die Ausführung also abhängig von der Geschwindigkeit der Zeicheneingabe, was in vielen Fällen sicherlich nicht erwünscht ist. Gibt man dagegen als erstes einen Doppelpunkt ein, wird die Ausführung der nachfolgenden Befehle solange unterbunden, bis ein Zeilenumbruch (Enter/Eingabe-Taste bzw. das ASCII-Steuerzeichen 0x0D oder 0x0A) übertragen wird.
Sinnvoll kann der Doppelpunkt-Befehl auch bei einer relativ geringen Verbindungsgeschwindigkeit (z. B. 4800 bit/s) sein, wenn bei zeitkritischen Anwendungen Fehler durch die Übermittlungsverzögerungen ausgeschlossen werden und die Anweisungen möglichst unmittelbar nacheinander ausgeführt werden sollen.

Hinweis 1:
Der **Eingangspuffer** des USB-I²C-Interface ist **400 Zeichen** groß. Werden mehr als 400 Zeichen in den Puffer geschrieben, führt das zu einem Überlauf bei dem Zeichen verloren gehen. Dadurch wäre die Ausführung der Anweisungen nicht mehr sichergestellt. Aus diesem Grund führt das USB-I²C-Interface bei einem Puffer-

Überlauf (also bei mehr als 400 Zeichen) selbstständig einen **Reset** aus, bei dem der Eingangspuffer komplett geleert wird!
Da jede Anweisung, die das USB-I²C-Interface ausführt, automatisch aus dem Puffer gelöscht wird, füllt sich der Puffer nur, wenn die Anweisungen schneller eingegeben werden, als das Interface sie abarbeiten kann oder wenn die Ausführung zuvor mit dem Doppelpunkt-Befehl unterbrochen wurde.

Hinweis 2:
Normalerweise überträgt das Terminalprogramm „HTerm“ die im Eingabefeld eingetippten Zeichen erst nach dem Drücken der Eingabe-Taste (oder dem Drücken des Buttons „ASend“). Dabei überträgt es kein Steuerzeichen für die Eingabe-Taste. Für dieses Verhalten von „HTerm“ steht die Einstellung **„Send on enter“** (direkt über dem Eingabefeld) auf „None“.
Soll „HTerm“ beim Absenden der Anweisungsfolge durch das Drücken der Eingabetaste automatisch bestimmte Steuerzeichen anhängen, so muss die Einstellung im **„Send on enter“**-Auswahlmenü auf das gewünschte Steuerzeichen (z. B. „CR-LF“ – siehe folgendes Bild) umgestellt werden.
„HTerm“ bietet zudem die Möglichkeit, Steuerzeichen auch direkt im Eingabefeld einzugeben. Dazu muss man sich zuvor den zugehörigen Hex-Wert raussuchen (z.B. 0x0D für „CR“ oder 0x0A für „LF“ – siehe auch http://goascii.de/ - dann im „**Type**“-Auswahlmenü (links neben dem Eingabefeld) von „**ASC**“ (für ASCII) auf „**HEX**“ umstellen und z. B. 0D eintippen. Die Anweisungen können sogar gemischt (also „ASC“ und „HEX“ abwechseln) eingegeben werden und werden dafür im Eingabefeld farblich markiert.



Beispiel:

1. In „HTerm“ soll in diesem Beispiel „**Send on Enter**“ auf „**None**“ und „**Type**“ auf „**ASC**“ stehen.
2. Die mit dem Doppelpunkt beginnende Anweisungsfolge wird eingegeben und zum USB-I2C gesendet. Das USB-I²C-Interface führt sie jetzt noch nicht aus (die DATA-LED blinkt jetzt schnell):
   `:S40 00 L00AF wFF L00AF w00 L00AF wFF L00AF w00 L00AF wFF P`
3. Nun wird in „HTerm“ die Einstellung „**Type**“ von „ASC“ auf „**HEX**“ umgestellt und **`0D`** im Eingabefeld eingegegeben. Erst nach dem Drücken der Eingabetaste wird dies zum USB-I²C-Interface übertragen. Das Interface führt danach sofort die zuvor empfangene Anweisungsfolge aus.

4.1.6. „L“ – Warte mit Ausführung für angegebene Zeit

Der Befehl **`L`** lässt das USB-I²C-Interface die Befehlsausführung für die angegebene Zeit anhalten. Mit diesem Befehl können also Wartepausen in die Ausführung eingefügt werden. Dabei können Zeiten zwischen 1 ms und 65535 ms (ca. 65 Sekunden) angegeben werden.
Dies ist immer dann nützlich, wenn etwas eine bestimmte Zeit andauern soll, bevor das nächste passiert. Also z.B. eine LED eine Zeit lang leuchten soll oder wenn eine Messung eine bestimmte Zeit benötigt, bevor anschließend der Messwert ausgelesen werden kann.
Besonders nützlich ist dieser Befehl im Zusammenhang mit den Makrofunktionen des USB-I²C-Interface, da dadurch ganze Programme periodisch ausgeführt werden können (z. B. A/D-Wandler-Messwerte auslesen, Lauflichter usw.)

Je nach Einstellung des Konfigurationsparameters **`Y6`** können die Zeitangaben mit 4 Zeichen in Hexadezimalschreibweise (**`Y60`** = Auslieferungszustand) oder mit 5 Zeichen in Dezimalschreibweise (**`Y61`**) angegeben werden.

Schreibweise der Zeitangabe	Kleinster Wert (1ms)	Größter Wert (ca. 65 s)
Hexadezimal (Auslieferungszustand)	0001	FFFF
Dezimal	0001	65535

Beispiel:
Mit diesem Befehl ist es z.B. möglich, etwas für eine Sekunde (1s = 1000 ms = 0x3E8) einzuschalten und danach wieder auszuschalten.

Mit dem Befehl **S4000** können die 8 Portpins eines PCF8574-Chips auf High-Pegel geschaltet werden. Wenn an jeden Ausgang eine LED angeschlossen ist, beginnen diese zu leuchten. Mit **WFF** werden die LEDs dann wieder ausgeschaltet:
[Start] [Write-Adr: 0x40] [Daten] [Warte:1000ms] [Write] [Daten] [Stopp]
```
S40 00 L03E8 WFF P
```

Auf diese Weise kann beispielsweise auch ein LED-Lauflicht programmiert werden:
```
s40 ff L002f wfe L002f wfc L002f wf8 L002f wf0 L002f we0 L002f wc0 L002f w80
L002f w00 L002f w80 L002f wc0 L002f we0 L002f wf0 L002f wf8 L002f wfc L002f wfe
L002f wff l002f P
```

Im Kapitel „Makrofunktion“ (Kapitel 4.4.2.) findet sich eine Beschreibung, wie solch ein Lauflicht ganz einfach im Makrospeicher abgelegt und dann periodisch ausgeführt werden kann.

4.1.7. „N“ – Master antwortet NACK nach letztem Read

Der **N**-Befehl wird nur für sehr spezielle Anwendungen benötigt. Dieser Befehl kann VOR eine Lese-Anweisung geschrieben werden und lässt dadurch den Master (also das USB-I²C-Interface) beim letzten gelesenen Byte mit NACK (Not Acknowledge) antworten, anstatt mit ACK (Acknowledge), wie sonst üblich. Dies signalisiert dem Slave, dass dieser Lese-Befehl der Letzte war und dass erstmal keine weiteren Bytes gelesen werden.

Im Normalfall (Konfiguration im Auslieferungszustand: **Y20**) gibt der Master nach dem letzten gelesenen Byte automatisch ein NACK als Antwort zum Slave zurück. Wird mit dem Konfigurationsbefehl **Y21** die automatische NACK-Antwort des USB-I²C-Interface deaktiviert, muss sich der Anwender selber um die ACK/NACK-Antwort kümmern und zum richtigen Zeitpunkt den **N**-Befehl verwenden. Dieser Befehl muss vor der letzten Lese-Anweisung ans USB-I²C-Interface gesendet werden, damit dieses den letzten Lesezugriff mit einem NACK beantworten kann.

Sinnvoll ist das Abschalten der automatischen NACK-Antwort (mit **Y21**) und die daraus folgende Verwendung des **N**-Befehl eventuell dann, wenn mehr als die maximal möglichen 255 Bytes in einer Folge vom Slave gelesen werden sollen (mit dem Befehl **RFF** werden 255 Bytes gelesen). „*In einer Folge*“ bedeutet dabei, ohne eine Repeated-Start-Bedingung und ohne eine erneute Adressierung des Slaves, wie sie mit **S41 FF** oder **R FF** erfolgen würde.
Mit dem **N**-Befehl können also mit höherer Geschwindigkeit viele Daten vom Slave gelesen werden.

Beispiel:
Es sollen 1000 Datenbytes, also 4 mal 250 (0xFA) Bytes, mit nur einer Slave-Adressierung und nur einer Startbedingung vom Slave gelesen werden (genauso möglich: 3x 255 (0xFF) und 1x 235 (0xEB)):

[Deaktiviert automatische NACK-Antwort]
```
Y21
```

[Start] [Read-Adr: 0x41] [Read 0xFA Byte] [Read 0xFA Byte] [Read 0xFA Byte] [NACK nach letztem Read] [Read 0xFA Byte] [Stopp]
```
S41 FA FA FA N FA P
```

[Aktiviert automatische NACK-Antwort] (nur sicherheitshalber hier, damit dies nicht vergessen wird)
```
Y20
```

Ausgabe (4x 250 Bytes):
```
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ...
```

4.2. Befehle zur Konfiguration des USB-I²C-Interface

Mit den in diesem Kapitel beschriebenen Befehlen aus der Tabelle 3 (Seite 11) kann das Verhalten des USB-I²C-Interface je nach Einsatzgebiet angepasst werden.

Neben der Einstellung der I²C-Bus-Taktrate (Befehl: **T**) und der Verbindungsgeschwindigkeit zwischen PC und Interface (Befehl: **X**), dienen einige Befehle der Formatierung der Rückgabewerte des USB-I²C-Interface zum PC (Befehle: **Y0, Y3, Y4**) und der Eingabewerte (Befehl: **Y6**) im Terminalprogramm.

Weitere Parameter definieren das Verhalten des Masters bei der I²C-Kommunikation (Befehle: **Y1, Y2**) und nach einem Reset (Befehl: **Y5**).

Abgefragt werden können diese Einstellungen alle mit Hilfe des Statusbefehls (Befehl: **?**).

Mit dem **Z**-Befehl lässt sich ein Reset des USB-I²C-Interface durchführen (Befehl: **Z4B**), die Auslieferungs-Konfiguration wieder herstellen (Befehl: **ZAA**) und der Vorgang starten, der für ein Firmware-Update nötig ist (Befehl: **ZFF**).

4.2.1. „?“ – Systemstatus und Einstellungen abfragen

Mit Hilfe des Statusbefehls **?** kann die aktuelle Konfiguration des USB-I²C-Interface überprüft werden. Dieser Befehl kann auch nützlich sein, wenn man wissen möchte, ob das Interface richtig angeschlossen ist und ob am PC der korrekte COM-Port ausgewählt und geöffnet wurde (siehe folgender Hinweis). Zudem kann mit diesem Befehl geprüft werden, welche Geräte-Adresse zuletzt verwendet wurde.

Zeile 1: Enthält die Firmware-Versionsnummer des USB-I²C-Interface und den werkseitig fest eingestellten Kalibrierungswert des internen RC-Oszillators.
Zeile 2: Zeigt die zuletzt verwendete I²C-Geräte-Adresse an. Die hier angezeigte Adresse nimmt das USB-I²C-Interface zur Adressierung bei der Verwendung des **W**- oder **R**-Befehls. Diese Adresse ändert sich erst, wenn zusammen mit dem Startbefehl (also z.B. **S40**)eine neue Adresse eingegeben wird. Nach einem Reset und nach einem Systemstart, wenn noch keine Adresse eingegeben wurde, lautet die Adresse **0x00**.
Zeile 3: Zeigt die aktuell eingestellte Verbindungsgeschwindigkeit zum PC. Wird diese mit dem **T**-Befehl umgestellt, so steht hier sofort danach die neue Geschwindigkeit – übernommen wird diese Geschwindigkeit aber erst nach einem Reset (Befehl: **Z4B**).
Zeile 4: Zeigt die Taktrate mit der das USB-I²C-Interface die Datenübertragung auf dem I²C-Bus taktet. Die angezeigten Werte können von den Eingabewerten abweichen, da nur bestimmte Taktraten in vorgegebenen Schritten möglich sind. Prüfen sie bitte immer, ob alle angeschlossenen Slave-Geräte die eingestellte Geschwindigkeit auch unterstützen (Datenblätter)!
Zeile 5-12: Aktuell eingestellte Konfigurationsparameter **Y0** bis **Y7** (siehe Kapitel 4.2.5.).

Beispiel:
[Systemstatus abfragen]
?

Antwort:
```
ELV USB-I²C-Interface v1.6 (Cal:64)
Last Adress:0x1E
Baudrate:115200 bit/s
I2C-Clock:99632 Hz
Y00
Y10
Y20
Y30
Y40
Y50
Y60
Y70
```

Hinweis:
Wird die Verbindungsgeschwindigkeit des USB-I²C-Interface umgestellt und die neue Einstellung anschließend vergessen, kann mit Hilfe des Statusbefehls (**?**) die richtige Geschwindigkeit durch Ausprobieren herausgefunden werden. Dazu stellt man in „HTerm“ nacheinander unterschiedliche Verbindungsgeschwindigkeiten ein, öffnet jeweils einmal den COM-Port, sendet ein Fragezeichen zum Interface und schaut, ob das USB-I²C-Interface antwortet.
Achtung: Solange ein Makro ausgeführt wird, kann das USB-I²C-Interface nicht auf Anfragen antworten. Um sicher zu gehen, dass ein eventuell laufendes Makro (während der Ausführung blinkt dann meist die DATA-LED) erst beendet wird, sollte bei jedem Testdurchgang zuerst der Beende-Makro-Befehl („**<<**“ – siehe Kapitel 4.4.4.) eingegeben, abgesendet und danach erst das Fragezeichen verwendet werden.

4.2.2. „T“ – I²C-Bustakt einstellen

Das USB-I²C-Interface erzeugt als Master für die Datenübertragung den Takt auf dem I²C-Bus. Die entsprechende Geschwindigkeit kann je nach angeschlossenem Slave und Anforderung variabel zwischen 226 Hz und 409,6 kHz eingestellt werden.
Im Auslieferungszustand beträgt die Taktrate ca. 100 kHz. Der genaue Wert, der auch über den Statusbefehl (**?**) angezeigt wird, liegt knapp darunter bei 99632 Hz, da nicht alle Taktwerte möglich sind. Es können zwar im Frequenzbereich von 226 bis 409.600 Hz alle Werte eingegeben werden, allerdings wählt das USB-I²C-Interface immer den nächstmöglichen höheren Wert.
Die Einstellung der Taktrate geschieht mit dem Befehl **T**, dem die Taktfrequenz in Hz als 6-stelliger Dezimalwert folgt. Die Einstellung wird sofort für alle nachfolgenden Datenübertragungen verwendet. Auf eine korrekte Einstellung antwortet das USB-I²C-Interface nicht. Die Einstellung lässt sich jederzeit mit dem Statusbefehl (**?**) überprüfen.

Beispiel 1:
[I²C-Bus-Taktrate auf 50 kHz (50000 Hz) stellen]
T050000

[Systemstatus abfragen]
?

Antwort:
```
ELV USB-I²C-Interface v1.6 (Cal:64)
Last Adress:0x00
Baudrate:115200 bit/s
I2C-Clock: 50498 Hz
Y00
Y10
Y20
Y30
Y40
Y50
Y60
Y70
```

Beispiel 2:
[I²C-Bus-Taktrate auf 250 Hz stellen]
T000250

Hinweis 1:
Bis zur Firmware-Version 1.5 wird die neu eingestellte I²C-Taktrate erst nach einem Reset (Befehl: **Z4B**) übernommen. Ab v1.6 geschieht das sofort nach der Eingabe einer neuen Taktrate.

Hinweis 2:
Werden Taktraten außerhalb des zulässigen Wertebereichs eingegeben, antwortet das USB-I²C-Interface mit „`Err:WRONG VALUE`“ und behält den alten Wert.

4.2.3. „X“ – PC-Verbindungsgeschwindigkeit einstellen

Damit das USB-I²C-Interface möglichst flexibel mit unterschiedlichen Programmen und Betriebssystem zusammenarbeiten kann, nutzt es für die Verbindung zum PC eine serielle Verbindung über einen virtuellen COM-Port. Diese Schnittstelle muss vor einem Verbindungsaufbau entsprechend den Einstellungen vom USB-I²C-Interface konfiguriert werden. Im Auslieferungszustand sind das die folgenden Einstellungen:

- **Übertragungsrate: 115200 bit/s**
- **1 Start-Bit, 8 Daten-Bits, 1 Stopp-Bit**
- **keine Parität, keine Flusssteuerung (Handshake)**

Die Übertragungsrate kann dabei in mehreren Stufen an eigene Bedürfnisse angepasst werden. Dazu dient der Befehl **X**, der zusammen mit 4 Ziffern (Dezimalschreibweise) eingegeben werden muss. Folgende Einstellungen sind möglich:

Befehl	Geschwindigkeit
X0048	4800 bit/s
X0096	9600 bit/s
X0192	19,2 kbit/s
X0384	38,4 kbit/s
X0576	57,6 kbit/s
X0768	76,8 kbit/s
X1152	115,2 kbit/s
X2304	230,4 kbit/s

Die Einstellung wird **nicht sofort übernommen**, damit die neue Konfiguration zuvor mit dem Statusbefehl (**?**) überprüft und gegebenenfalls korrigiert werden kann. Um die neue Einstellung zu aktivieren, muss ein Reset durchgeführt werden, was mit dem Reset-Befehl **Z4B** ausgeführt werden kann. Anschließend muss auch auf der PC-Seite in „HTerm“ die neue Verbindungsgeschwindigkeit („Baud“) eingestellt werden.

Beispiel:
[Systemstatus abfragen]
?

Antwort:
```
ELV USB-I²C-Interface v1.6 (Cal:64)
Last Adress:0x00
Baudrate:115200 bit/s
I2C-Clock: 99632 Hz
...
```

[Verbindungsgeschwindigkeit auf **57,6 kbit/s** umstellen]
X0576

[Systemstatus abfragen]
?

Antwort:
```
ELV USB-I²C-Interface v1.6 (Cal:64)
Last Adress:0x00
Baudrate:57600 bit/s
I2C-Clock: 99632 Hz
....
```

[Reset ausführen]
Z4B
Antwort:
Reset... (anschließend folgt eine Reihe unlesbarer Sonderzeichen, da die Übertragungsgeschwindigkeit von „HTerm“ nun nicht mehr übereinstimmt)

Die unleserlichen Zeichen beinhalten die normale Startmeldung des USB-I²C-Interface, die jedoch aufgrund der nicht übereinstimmenden Verbindungsgeschwindigkeit falsch empfangen wurden. Damit Empfang und Senden wieder korrekt funktionieren, muss auch „HTerm“ auf die neue Verbindungsgeschwindigkeit umgestellt werden. Wählen sie dafür in „HTerm“ neben „Baud“ jetzt die Einstellung „57600“. Wenn jetzt der Statusbefehl (**?**) eingegeben und ans USB-I²C-Interface gesendet wird, antwortet es mit einer lesbaren Antwort.

Hinweis:
Wird die Verbindungsgeschwindigkeit des USB-I²C-Interface umgestellt und die neue Einstellung anschließend vergessen, kann mit Hilfe des Statusbefehls (**?**) die richtige Geschwindigkeit durch Ausprobieren herausgefunden werden. Dazu stellt man in „HTerm“ nacheinander unterschiedliche Verbindungsgeschwindigkeiten ein, öffnet jeweils einmal den COM-Port, sendet ein Fragezeichen zum Interface und schaut, ob das USB-I²C-Interface antwortet.
Achtung: Solange ein Makro ausgeführt wird, kann das USB-I²C-Interface nicht auf Anfragen antworten. Um sicher zu gehen, dass ein eventuell laufendes Makro (während der Ausführung blinkt dann meist die DATA-LED) erst beendet wird, sollte bei jedem Testdurchgang zuerst der Beende-Makro-Befehl („**<<**“ – siehe Kapitel 4.4.4.) eingegeben, abgesendet und danach erst das Fragezeichen verwendet werden.

4.2.4. „Z“ – Reset, Auslieferungszustand, Firmware-Update

Mit dem **Z**-Befehl können je nach angehängter Zeichenfolge drei unterschiedliche Funktionen ausgeführt werden. Die möglichen Varianten werden in der folgenden Tabelle beschrieben:

Befehl	Beschreibung
Z4B	Führt sofort einen Reset aus (alle Einstellungen bleiben).
ZAA	1.) Setzt alle Einstellungen auf den Auslieferungszustand zurück. 2.) Löscht den Makrospeicher. 3.) Führt einen Reset aus.
ZFF	1.) Setzt alle Einstellungen auf den Auslieferungszustand zurück. 2.) Löscht den Makrospeicher. 3.) Geht in den Modus zum Updaten der Firmware.

Beispiel 1:
[Reset ausführen]
Z4B
Antwort:
```
Reset...
ELV USB-I²C-Interface v1.6 (Cal:64)
```

Beispiel 2:
[Auslieferungszustand herstellen]
ZAA
Antwort:
```
Full Init...
ELV USB-I²C-Interface v1.6 (Cal:64)
```

Beispiel 3:
[Auslieferungszustand herstellen und in Firmware-Update-Modus wechseln]
ZFF
Antwort:
```
Close COM-Port and start FW-update!
```

Hinweis 1:
Ist ein Makro im USB-I²C-Interface gespeichert, so wird dieses nach jedem Reset automatisch ausgeführt. Das USB-I²C-Interface reagiert in dem Fall kurzzeitig oder dauerhaft nicht mehr auf Befehle. Bevor es neue Befehle empfangen und ausführen kann, muss die Makroausführung erst mit dem Beende-Makro-Befehl („**<<**“ – siehe Kapitel 4.4.4.) gestoppt werden.
Die Makro-Start-Automatik kann mit dem **Y51**-Befehl deaktiviert werden (siehe Kapitel 4.2.5.).

Hinweis 2:
Wird zu irgendeinem Zeitpunkt vor einem Reset die Verbindungsgeschwindigkeit des USB-I²C-Interface mit dem **T**-Befehl verändert, so verwendet es die neue Einstellung sofort nach dem Reset und ist daher erst wieder ansprechbar, wenn am PC die gleiche Verbindungsgeschwindigkeit eingestellt wird.
Dasselbe gilt für den Befehl **ZAA**: Verwendet man zum Zeitpunkt der Eingabe dieses Befehls eine andere Verbindungsgeschwindigkeit als 115,2 kbit/s, so muss nach dem Reset das Terminalprogramm ebenfalls auf 115200 bit/s eingestellt werden.

Hinweis 3:
Die Beschreibung, wie bei einem Firmware-Updates vorzugehen ist, liegt jedem Update bei, und ist genau zu befolgen. Sobald das USB-I²C-Interface nach Eingabe des Befehls **ZFF** in den Update-Modus wechselt (die DATA-LED blinkt), nimmt es keine weiteren Anweisungen mehr entgegen. Möchte man doch kein Update durchführen und den Modus wieder verlassen, muss die USB-Verbindung kurz getrennt werden, indem der USB-Stecker aus dem Interface gezogen wird. Anschließend befinden sich alle Einstellungen des USB-I²C-Interface wieder im Auslieferungszustand.

4.2.5. „Y“ – Konfigurations-Befehle

Der **Y**-Befehl dient zur Konfiguration einer Reihe ganz unterschiedlicher Funktionen und wird zusammen mit zwei weiteren Ziffern eingegeben. Die erste nachfolgende Ziffer (**0** bis **7**) steht für die jeweilige Funktion (siehe Tabelle) und die zweite Ziffer aktiviert oder deaktiviert diese. Die in der Tabelle mit einem Kreuz (#) gekennzeichneten Werte (also z.B. **Y00**) sind die im Auslieferungszustand aktiven Einstellungen.
Die jeweils eingestellte Konfiguration kann mit Hilfe des Statusbefehls (**?**) überprüft werden.

0	0 #	dem letzten Datenbyte, das der Master aus dem Slave liest, folgt ein Zeilenumbruch (0x0D 0x0A) in der Rückgabe zum PC
	1	Daten, die der Master aus dem Slave ausliest, folgt **kein Zeilenumbruch** in der Rückgabe zum PC
1	0 #	Master **ignoriert** beim Schreiben die **NACK**-Antwort des Slaves (die bedeutet, dass der Slave keine weiteren Daten akzeptiert) und schreibt weiter
	1	Master stoppt Schreiben, wenn Slave mit NACK antwortet
2	0 #	nachdem der Master das letzte Byte vom Slave gelesen hat, antwortet er mit NACK (dadurch weiß der Slave, dass keine weiteren Daten folgen)
	1	beim Lesen sendet Master **kein NACK** nach letztem Byte (in diesem Fall ist mit der N-Anweisung manuell NACK zu senden)
3	0 #	ACK/NACK-Antworten des Slaves werden nicht zum PC übertragen
	1	beim Schreiben wird für jedes erhaltene **ACK** ein **K** zum PC übertragen und für ein **NACK** ein **N** (gut zum Debuggen)
4	0 #	jeder Daten-Rückgabe (2 Zeichen) folgt ein Leerzeichen (0x20)
	1	einer Daten-Rückgabe zum PC folgt kein Leerzeichen
5	0 #	nach einem Reset das Makro ausführen, wenn eines im Makrospeicher steht
	1	nach einem Reset kein Makro ausführen
6	0 #	die Makroadresse (nach > und V) wird mit 2- und die Wartepause (nach L) wird mit 4-ASCII-Zeichen in Hexadezimalschreibweise angegeben
	1	die Makroadresse (nach > und V) wird mit 3- und die Wartepause (nach L) wird mit 5-ASCII-Zeichen in Dezimalschreibweise angegeben
7	0 #	Gelesene Daten werden als Ascii-Zeichen im Hexadezimalformat zum PC gesendet
	1	Gelesene Daten werden als Ascii-Zeichen im Dezimalformat zum PC gesendet (ab Ver. 1.6)

Besonderheiten, Beispiele, Vorteile und Nachteile der Einstellungen Y0 bis Y7:

Y00: Übersichtliche Datenrückgabe, da die Daten eines abgeschlossenen Lese-Befehls mit einem Zeilenumbruch kenntlich gemacht werden.
Beispiel:
```
S41 05 R05 P
```
Antwort:
```
FF FF FF FF FF
FF FF FF FF FF
```

Y01: Gut, wenn weniger Zeichen übertragen werden sollen oder wenn man die Darstellung der Datenrückgabe mit Kommentar-Befehlen selber gestalten möchte.
Beispiel:
```
S41 05 R05 P
```
Antwort:
```
FF FF FF FF FF FF FF FF FF FF
```

Y10: Ein Schreib-Befehl wird komplett ausgeführt – egal wie ein Slave darauf reagiert.
(Die ACK/NACK-Antworten der Slaves sind nur sichtbar, wenn der Parameter **Y3** auf **Y31** gesetzt wird.)
Beispiel mit der Annahme, dass der Slave mit **NACK** antwortet:
```
S40 00 FF 00 FF 00 FF P
```
Antwort, wenn der Slave nicht bereit oder gar nicht angeschlossen ist:
```
NNNNNNN   (eventuell auch: KNNNNNN)
```

Beispiel mit der Annahme, dass der Slave mit **ACK** antwortet:
```
S40 00 FF 00 FF 00 FF P
```
Antwort, wenn Slave angeschlossen und bereit ist:
```
KKKKKKK
```

Y11: Ein Schreib-Befehl wird sofort abgebrochen, wenn der adressierte Slave nicht bereit ist.
(Die ACK/NACK-Antworten der Slaves sind nur dann sichtbar, wenn der Parameter **Y3** auf **Y31** gesetzt wird.)
Beispiel mit der Annahme, dass der Slave mit **NACK** (nicht bereit) antwortet:
```
S40 00 FF 00 FF 00 FF P
```
Antwort, wenn Slave nicht bereit oder nicht angeschlossen ist:
```
NN   (eventuell auch: KN)
```

Y20/Y21: Dieser Parameter bleibt normalerweise auf **Y20**. Die Verwendung der **Y21**-Einstellung ist im Kapitel 4.1.7. in Verwendung mit dem **N**-Befehl beschrieben.

Y30/Y31: Siehe hierzu die Beispiele zum **Y10**/**Y11**-Parameter, wo diese Einstellung auf **Y31** gestellt wurde, um die ACK/NACK-Antworten eines Slaves auf dem PC sichtbar zu machen.

Y40: Übersichtliche Datenrückgabe, da jedem Datenbyte ein Leerzeichen folgt und die Datenbytes dadurch sichtbar getrennt sind.
Beispiel:
```
S41 0A P
```
Antwort:
```
FF FF FF FF FF FF FF FF FF FF
```

Y41: Gut, wenn möglichst wenig Zeichen übertragen werden sollen und/oder wenn die Darstellung der Datenrückgabe mit Kommentar-Befehlen selbst gestaltet wird. Dabei ist oft die Verwendung zusammen mit der Einstellung **Y01** (kein Zeilenumbruch am Ende eines Lese-Befehls) sinnvoll.
Unübersichtliches Beispiel ohne Leerzeichen zwischen den Datenbytes:
```
S41 0A P
```
Antwort:
```
FFFFFFFFFFFFFFFFFFFF
```
Übersichtliches Beispiel (kommagetrennte Messwerte für Excel) durch die zusätzliche Verwendung von Kommentar-Befehlen (Semikolon-Befehl = Semikolon, Punkt-Befehl = Zeilenumbruch) und **Y01** (deaktiviert automatische Zeilenumbrüche):
```
S41 01;R01;R01;R01;R01.R01;R01;R01;R01;R01.P
```
Antwort:
```
FF;FF;FF;FF;FF
FF;FF;FF;FF;FF
```

Y50: Achtung: Im Auslieferungszustand wird nach jedem Reset ein im Speicher stehendes Makro automatisch ausgeführt. In Falle, dass eines ausgeführt wird, reagiert das USB-I²C-Interface nur noch auf den Befehl zum Beenden der Makroausführung (siehe Kapitel 4.4.4.). Diese automatische Makroausführung ist sehr nützlich und stellt eines der wichtigsten Features des USB-I²C-Interface dar. Sie kann das Interface völlig unabhängig von einem PC machen. Ein USB-I²C-Interface mit einem programmierten Makro (welches am Ende wieder zurück zum Anfang springt) benötigt nur noch ein USB-Netzgerät und kann dann eigenständig Geräte über I²C steuern.

Y51: Nach jedem Reset wartet das USB-I²C-Interface auf eine Eingabe vom angeschlossenen PC.
Ein Makro muss manuell vom PC aus (mit dem Befehl **>00**) gestartet werden.

Y60: Die Verwendung der Hexadezimalschreibweise ermöglicht eine kürzere Schreibweise der Befehle **L** (Wartepause), **V** (schreibe Makro an Adresse...) und **>** (starte Makro an Adresse...).
Beispiel 1 (zwei Wartepausen mit je 170 ms):
```
S40 FF L00AA W00 L00AA P
```
Beispiel 2 (obige Befehlskette in Makrospeicher schreiben und nach Start immer wieder ausführen):
```
V00{S40 FF L00AA W00 L00AA P >00}
```

Y61: Die Verwendung der Dezimalschreibweise für die Angabe der Wartepausen und Makroadressen ermöglicht eine leichter verständliche Befehlsschreibweise – vergrößert aber die beiden Befehle aber um je ein Zeichen.
Beispiel (wie vorheriges Beispiel mit zwei 170 ms-Pausen - jetzt in Dezimalschreibweise):
```
V000{S40 FF L00170 W00 L00170 P >000}
```

Y70: Gelesene Daten werden als Ascii-Zeichen im **Hexadezimalformat** zum PC gesendet.
Beispiel: `S41 0A P`
Antwort: `FF 00 01 5A AA 1F 00 FF FF FF`

Y71: **(Neu ab Firmware v1.6)** Gelesene Daten werden als Ascii-Zeichen im **Dezimalformat** (ohne führende Nullen) zum PC gesendet (siehe hierzu auch „LogView"-Beispiel 4.5.5.)
Beispiel (wie zuvor): `S41 0A P`
Antwort: `255 0 1 90 170 31 0 255 255 255`

4.3. Kommentar-Befehle

Obwohl die Kommentar-Befehle auf den ersten Blick eher unscheinbar wirken, sind diese sehr leistungsfähig und ermöglichen erst viele praktische Anwendungen. Zum einen dienen sie einer übersichtlicheren Gliederung von Rückgabewerte und Anweisungsfolgen. Sie erhöhen die Verständlichkeit und machen die Ziffernfolgen überhaupt erst verständlich. Zum anderen können die Daten mit Hilfe von Semikolon, Komma und Zeilenumbrüche so formatiert werden, dass die Daten auch in Excel und ähnlichen Visualisierungsprogrammen automatisch ausgewertet werden können.

4.3.1. „ " – Leerzeichen in Anweisungen einfügen

Das Leerzeichen ermöglicht eine sichtbare Trennung der Datenbytes und der Befehle untereinander. Leerzeichen werden vom USB-I²C-Interface ignoriert und können daher überall beliebig eingefügt werden.

Beispiel einer sehr unübersichtlichen Befehlsfolge ohne Leerzeichen:
```
S40F04BA7L1EA0W00L00AAW012EE64A53FB638564E523P
```

Beispiel derselben Befehlsfolge mit übersichtlichen Leerzeichen:
```
S40 F0 4B A7  L0EA0  W 00  L00AA  W 01 2E E6 4A 53 FB 63 85 64 E5 23  P
```

4.3.2. „(...)" – Anweisungen kommentieren

In Anweisungsfolgen können Kommentare eingefügt werden, die das USB-I²C-Interface ignoriert, wenn sie von runden Klammern eingeschlossen werden. Dadurch lassen sich beispielsweise für Schulungszwecke alle Bestandteile einer Anweisungsfolge verständlich beschreiben.
Auch wenn man ein Makro ins USB-I²C-Interface speichert, ist es eventuell sinnvoll, einen kurzen Kommentar dazu zu schreiben.

Beispiel mit Kommentaren in der Anweisungsfolge:

```
(Start-Bedingung mit Write-Adresse=>)S40 F0 (<=Datenbyte) (Stopp-Bedingung=>)P
```

Hinweis:
Ein Befehl darf nicht wie im folgenden Beispiel von einem Kommentar in zwei Teile getrennt werden!

```
S40 (hier ist kein Text erlaubt)F0  L(hier auch nicht)0EA0  W01 P
```

4.3.3. „ . “ – Punkt in Rückgabe einfügen

Erhält das USB-I²C-Interface einen Punkt, gibt es dafür einen Zeilenumbruch zurück, der aus den Steuerzeichen für Wagenrücklauf (Carriage Return = `0x0D`) und Zeilenvorschub (Line Feed = `0x0A`) besteht.
Hierdurch können die Daten-Rückgabewerte gezielt formatiert werden.

Beispiel:
(zuvor wurden die automatischen Zeilenumbrüche mit **`Y01`** deaktiviert)

```
.S41 05 . R05 . R05 . P
```

Antwort:

```
FF FF FF FF FF
FF FF FF FF FF
FF FF FF FF FF
```

Hinweis 1:
Ein Punkt darf nicht, wie im folgenden Beispiel, innerhalb eines Befehls stehen!

```
S.40.F0  L.0E.A0   W.01  P    (<= die Punkte sind an diesen Stellen un zulässig)
S40 F0. .L0EA0.  W01. P       (<= so ist die Verwendung der Punkte richtig)
```

Hinweis 2:
Die Verwendung der manuellen Zeilenumbrüche ist meistens zusammen mit dem Parameter **`Y01`** sinnvoll, der die automatischen Zeilenumbrüche deaktiviert (siehe 4.2.5.).

4.3.4. „ , “ – Komma in Rückgabe einfügen

Erhält das USB-I²C-Interface ein Komma, gibt es auch wieder ein Komma zurück. Dies ist praktisch, wenn die Daten so formatiert werden sollen, dass sie beispielsweise mit Excel ausgewertet werden können.

Beispiel einer Rückgabe mit kommagetrennten Datenbytes:
(zusätzlich wurden hier zuvor die Parameter **`Y01`** und **`Y41`** eingeben, wodurch automatische Zeilenumbrüche und Leerzeichen deaktiviert sind)

```
S41 01, R01, R01, R01, R01.  R01, R01, R01, R01, R01.  P
```

Antwort:

```
F1,F2,F3,F4,F5
1F,2F,3F,4F,5F
```

Hinweis:
Ein Komma darf nicht, wie im folgenden Beispiel, innerhalb eines Befehls stehen!

```
S,40,F0  L,0E,A0   W,01  P    (<= die Kommata sind an diesen Stellen unzulässig)
S40 F0, ,L0EA0,  W01, P       (<= so ist die Verwendung richtig)
```

4.3.5. „ ; “ – Semikolon in Rückgabe einfügen

Die Verwendung des Semikolons funktioniert entsprechend dem Komma in Kapitel 4.3.4., nur dass das USB-I²C-Interface ein Semikolon statt einem Komma zurückgibt.

Beispiel: (entsprechend dem Beispiel in 4.3.4.)

Hinweise: (es gelten dieselben Hinweise wie in 4.3.4.)

4.3.6. „[...]“ - Kommentar in Rückgabe einfügen

Alle Zeichen, die innerhalb von eckigen Klammern an das USB-I²C-Interface gesendet werden, gibt das Interface unverändert wieder zurück. Dies ist eine sehr komfortable und nützliche Funktionen, da die Datenwerte hierdurch direkt lesbar werden.

Eckige Klammern dürfen nicht innerhalb der eckigen Klammern verwendet werden. Mit den folgenden drei Sonderfunktionen lassen sich beispielsweise auch Zeilenumbrüche erzeugen:

\r	Carriage Return
\n	Line Feed
\t	Horizontal Tab

Beispiel:
(zusätzlich wurden hier zuvor die Parameter Y01 und Y41 eingeben, wodurch automatische Zeilenumbrüche und Leerzeichen deaktiviert sind)

```
 .[D/A-Wandler:\r\n] S90 05 [Wert 0:]R01. [Wert 1:]R01. [Wert 2:]R01.[Wert 3:]R01. P
```

Antwort:

```
D/A-Wandler:
Wert 0: A2
Wert 1: EA
Wert 2: F5
Wert 3: FF
```

Hinweis:
[\r\n] bewirkt dasselbe wie ein (deutlich kürzerer) Punkt-Befehl außerhalb der eckigen Klammern.
[,] bewirkt dasselbe wie ein Komma-Befehl außerhalb der eckigen Klammern.
[;] bewirkt dasselbe wie ein Semikolon-Befehl außerhalb der eckigen Klammern.

4.4. Makro-Funktion

Die Makrofunktionalität ermöglicht ein autarkes Arbeiten des USB-I²C-Interface und der angeschlossenen Slave-Geräte, ohne von einem PC-Programm gesteuert werden zu müssen, bzw. sogar gänzlich ohne einen angeschlossenen PC, wenn ein USB-Netzteil oder ein KFZ-Adapter zur Stromversorgung verwendet wird.

Der Makrospeicherbereich des USB-I²C-Interface umfasst **256 Zeichen** in dem eine lange Anweisungsfolge gespeichert werden kann. Diese führt das USB-I²C-Interface nach dem Start einmal oder immer wieder aus. Da der Makrospeicher in EEPROM-Technik ausgeführt ist, bleibt eine gespeicherte Anweisungsfolge auch über lange Zeiträume ohne Spannungsversorgung erhalten und kann automatisch sofort wieder ausgeführt werden, sobald das USB-I²C-Interface mit Spannung versorgt wird (Parameter **Y50** beachten).

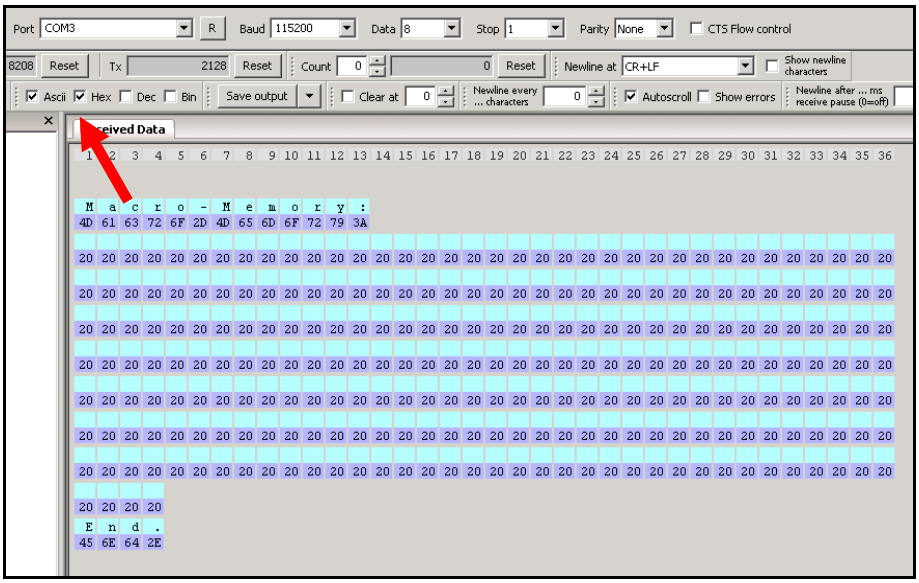
Mit einem als Datenlogger konfigurierten USB-I²C-Interface und einem angeschlossenem I²C-A/D-Wandler oder einem I²C-Temperatursensor können direkt Messdaten empfangen und in ein grafisches Datenloggerprogramm übernommen werden.

Sollen keine Messwerte aufgenommen werden, kann das USB-I²C-Interface auch ohne PC völlig autark z. B. mehrere I²C-Busexpandern ansteuern, an denen wiederum Leuchtdioden, Motoren oder andere Geräte angeschlossen werden. So sind komplette Steuerungsabläufe einfach und schnell zu realisieren.

4.4.1. „U“ – Makrospeicherinhalt zum PC ausgeben

Der **256 Zeichen** große Makrospeicherbereich des USB-I²C-Interface wird über den Befehl **U** zum PC ausgegeben. Dadurch kann die gespeicherte Anweisungsfolge überprüft und gegebenenfalls korrigiert werden.

Auf die **U**-Anweisung hin, wird der Makrospeicher immer komplett ausgegeben. Leere Speicherzellen werden als Leerzeichen (`0x20`) ausgegeben. Um diese Leerzeichen in „HTerm“ sehen zu können, ist es praktisch bei der Makroerstellung die Ausgabe in „Ascii“ und „Hex“ darzustellen (siehe Abbildung).



Beispiel:
[Makrospeicher auslesen]
U

Antwort (leerer Makrospeicher = 256 Leerzeichen):

```
Macro-Memory:
................................................................................
................................................................................
................................................................................
End.
```

Antwort (voller Makrospeicher):

```
Macro-Memory:
(Dies Makro nimmt analoge Messwerte auf und sendet sie formatiert zum PC).S90 05 (<= Ein
PCF8591 A/D-Wandler wird initialisiert) [Wert 0:]R01.[Wert 1:]R01.[Wert 2:]R01.[Wert
3:]R01.P L03E8 (<=warte 1s) >00 (<= Beginne von vorn) =>Hier kommen wir nie hin!<=
End.
```

4.4.2. „V“ – Anweisungsfolge in Makrospeicher schreiben

Der **V**-Befehl dient zum Speichern einer Anweisungsfolge an eine bestimmte Stelle im Makrospeicher. Dabei überschreiben die neuen in geschweiften Klammern stehenden Anweisungen die alten im jeweiligen Speicherbereich stehenden Zeichen.

Dem **V** folgt die Speicherzellen-Adresse ab der die neuen Anweisungen geschrieben werden sollen. Die Adresse wird dabei im Auslieferungszustand (Parameter **Y60**) in Hexadezimalschreibweise mit zwei Ziffern angegeben. Wird der Parameter **Y61** eingestellt, so erfolgt die Angabe der Speicheradresse in Dezimalschreibweise mit 3 Ziffern.

Die Makrospeicher-Adresse wird von 00 bis FF (Hexadezimalschreibweise) bzw. von 000 bis 255 (Dezimalschreibweise) gezählt.

Auf einen erfolgreichen Speicherbefehl hin, gibt das USB-I²C-Interface keine Antwort. Versucht man mehr Zeichen in den Speicher zu schreiben, als hineinpassen, so erhält man die Fehlermeldung „Err:MEMORY FULL“ und die „überlaufenden Zeichen“ werden vom USB-I²C-Interface direkt ausgeführt, was zu weiteren Fehlern führen wird.

Einzelne Speicherzellen können gelöscht werden, indem man sie mit Leerzeichen überschreibt, da Leerzeichen in der Programmausführung immer vom USB-I²C-Interface ignoriert werden.
Den **kompletten Speicher** kann man **löschen**, indem die Adresse 00 (bzw. 000 in Dezimalschreibweise) angegeben wird und die geschweiften Klammern komplett leer bleiben: **V00{}**

Beispiel mit Parameter Y60:
[Anweisungsfolge ab Adresse 0x00 in Makrospeicher schreiben]
```
V00{S40 00 L00FF WFF P L00FF >00}
```

Gleiches Beispiel mit Parameter Y61:
[Anweisungsfolge ab Adresse 000 in Makrospeicher schreiben]
```
V000{S40 00 L00255 WFF P L00255 >000}
```

Beispiel zum Ersetzen eines bestimmten Wertes im Makrospeicher (vorheriges Beispiel als Ausgangspunkt):
[Zahlenwert in erster Wartepause von 255ms auf 100ms ändern]
```
V010{100}
```

Hinweis:
Zusätzliche geschweifte Klammern innerhalb der geschweiften Befehls-Klammern sind unzulässig. Runde Klammern und eckige Klammern sind jedoch auch für Makroanweisungen erlaubt.
Beispiel: `V00{ (erlaubt).{nicht erlaubt}.[erlaubt] }`

4.4.3. „>“ – Makro-Ausführung starten

Sobald im Makrospeicher eine Anweisungsfolge abgelegt worden ist, kann diese mit dem „**>**“-Befehl direkt ausgeführt werden. Dem „**>**“-Zeichen folgt dabei die Adresse ab der das USB-I²C-Interface die Anweisungen abarbeiten soll. Die Adresse wird dabei im Auslieferungszustand (Parameter **Y60**) in Hexadezimalschreibweise mit zwei Ziffern angegeben. Wird der Parameter **Y61** eingestellt, so erfolgt die Angabe der Speicheradresse in Dezimalschreibweise mit 3 Ziffern.

Äußerst nützlich für periodisch ablaufende Programme ist es, wenn am Ende einer Anweisungsfolge im Makrospeicher **>00** steht (**>000** in Dezimalschreibweise), da die Abarbeitung dann immer wieder automatisch von vorne beginnt. Natürlich kann hier auch eine andere Einsprungsadresse angegeben werden, damit nach dem ersten Durchlauf beispielsweise nicht alles, sondern nur ein Teil periodisch wiederholt wird. So kann zwischen einem einmaligen Konfigurationsteil und einem periodischen Ausführungsteil im Programm unterschieden werden (siehe folgendes Beispiel, dass auf dem in der ELV-Bauanleitung beschriebenen A/D-Wandler basiert, der zum Programmstart einmal konfiguriert wird).

Beispiel:
[Anweisungsfolge ab Adresse 0x00 in Makrospeicher schreiben, die später ab 0x06 wiederholt werden soll]
```
V00{S90 05 R04 L00FF . P >06}
```

[Ausführung in Adresse 0x00 starten]
```
>00
```

Antwort:
```
02 01 FE 89
09 03 FF 89
0E 01 FF 8A
...
```

Hinweis:
Während das USB-I²C-Interface ein Makro ausführt, reagiert es auf keine neuen Befehle vom PC. Einzige Ausnahme ist der Befehl zum Beenden einer Makroausführung („**<**“ oder „**<<**“). Weitere Infos zum Beenden eines Makros finden sich im Kapitel 4.4.4.

4.4.4. „<“ – Makro-Ausführung beenden

Der Befehl „**<**“ beendet eine laufende Makroausführung, sobald diese bis zum Ende der Anweisungsfolge ausgeführt worden ist. Sprungbefehle (z. B. **>00**) werden nach der Eingabe des Ende-Befehls nicht mehr ausgeführt, so dass eine periodisch ablaufende Anweisungsfolge nicht immer wieder aufgerufen wird.
Je nach Inhalt der gespeicherten Anweisungsfolge kann es also längere Zeit dauern, bis das Programm fertig abgearbeitet wurde. Dies gilt insbesondere beim Einsatz von Wartepause (Befehl: **L**), die jeweils bis zu 65 Sekunden dauern können.

Soll das Ende der Anweisungsfolge nicht abgewartet werden, kann die Ausführung auch augenblicklich mit einer zweiten Stopp-Anweisung (also mit „**<<**“) beendet werden. Da bei dem Sofort-Stopp-Befehl („Instant-Stop“) der Zustand des I²C-Busses in einem undefinierten Zustand bleiben kann, führt das USB-I²C-Interface nach dem Sofort-Stopp einen Reset aus. Nach dem Reset wird ein im Makrospeicher befindliches Makro auch dann nicht ausgeführt, wenn der Parameter **Y50** eingestellt ist. Das USB-I²C-Interface wartet anschließend also wieder auf neue Anweisungen.

<	Beendet Makro-Ausführung nach komplettem Durchlauf oder vor nächster „>“-Anweisung.
<<	Beendet das Makro sofort, führt einen Reset aus und startet das Makro **nicht** wieder neu.

Auf eine einfache Stopp-Anweisung („**<**“) gibt das USB-I²C-Interface keine Antwort zurück.
Auf eine Sofort-Stopp-Anweisung („**<<**“) gibt das USB-I²C-Interface die folgende Antwort zurück:
`„Instant Stop -> Reset...“`

4.5. Praktische Beispiele mit verschiedenen I²C-Geräten

In den folgenden Kapiteln sollen einige Beispielsapplikationen aufgezeigt werden, die mit dem USB-I²C-Interface realisiert werden können. Dazu sind jeweils einige grundlegende Anweisungsfolgen angegeben mit denen die jeweiligen Bausteine direkt angesprochen werden können.

Die meisten der am Markt erhältlichen I²C-Bausteine verwenden ganz ähnliche Befehle, weshalb auch diese mit nur kleinen Änderungen an den hier gegebenen Beispielen schnell verwendet werden können.

4.5.1. 8-Bit I/O-Interface mit dem PCF8574

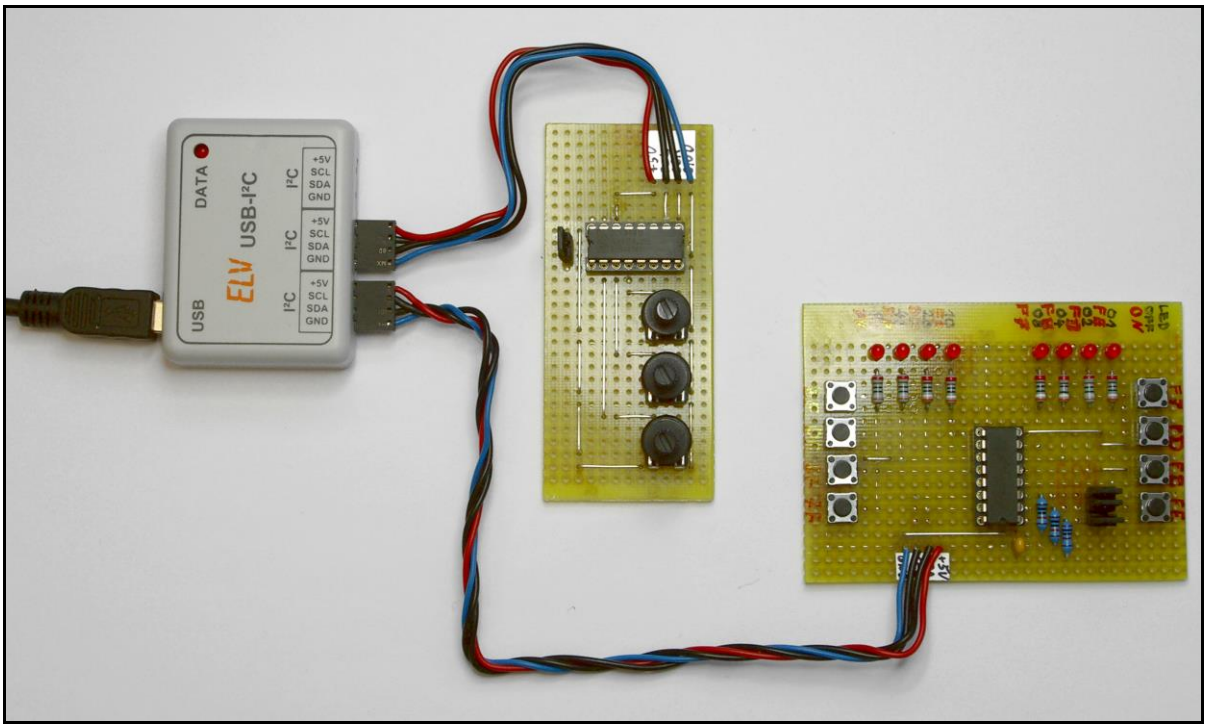
Der 8-Bit-I/O-Expander Baustein PCF8574 von Texas Instruments bzw. NXP ist ein sehr universeller Schaltkreis, der für erste Experimente und für viele Anwendungen ideal ist.

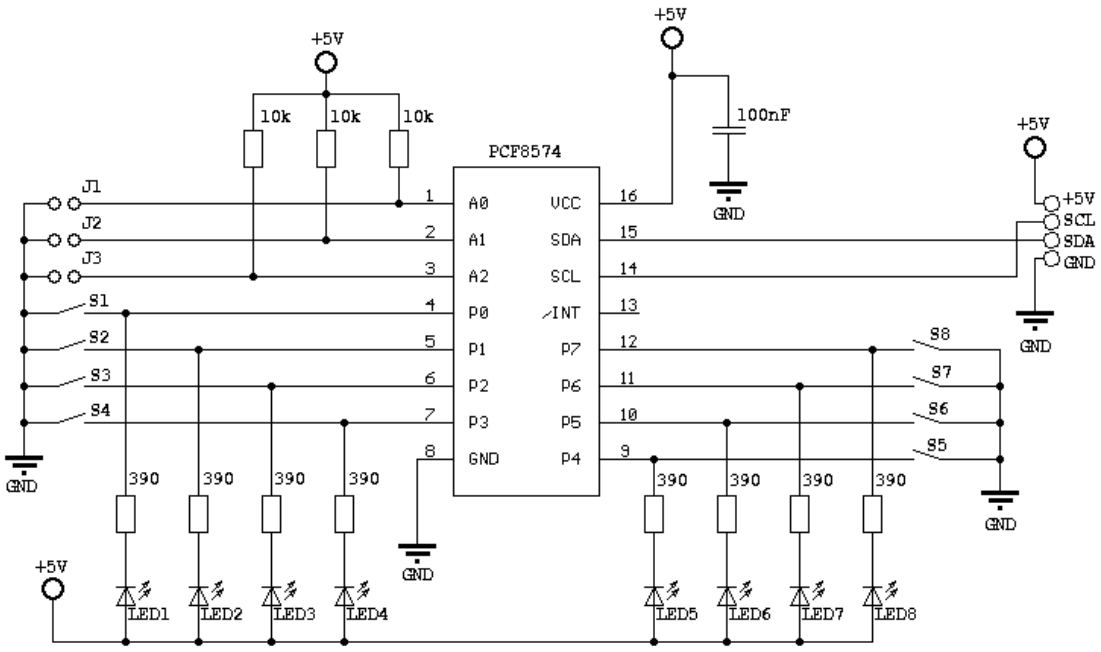
Eine Beispielsschaltung ist rechts im folgenden Bild zu sehen. (Bei der zweiten Schaltung in der Bildmitte handelt es sich um einen Beispielsaufbau mit dem A/D-Wandler-Baustein aus Kapitel 4.5.4.) Der Schaltplan der 8-Bit-I/O-Schaltung ist der nachfolgenden zweiten Abbildung skizziert.

In dem Beispiel können die 8 Ports gleichzeitig als Ein- und Ausgang verwendet werden. Allerdings kann der Zustand eines Schaltausgangs nicht am Leuchten der LED erkannt werden, solange der zugehörige Taster S1 - S8 gedrückt wird. Zu einem Kurzschluss kann es dabei aber aufgrund der Open-Drain-Ausgänge nicht kommen.
Die Zustände der 8 Taster lassen sich über den PCF8574 und das angeschlossene USB-I²C-Interface einlesen und die Leuchtdioden LED1 - LED8 können angesteuert werden.

Die Jumper J1 - J3 dienen zur Einstellung der Geräte-Adresse des Bausteins (siehe dazu den Grundlagenteil der USB-I²C-Interface-Bauanleitung und das Datenblatt zum PCF8574).

Datenblatt von Texas: [focus.ti.com/lit/ds/symlink/**pcf8574**.pdf](focus.ti.com/lit/ds/symlink/pcf8574.pdf)





Die Geräte-Adresse des PCF8574 bildet sich folgendermaßen:

0	1	0	0	A2	A1	A0	R/W-Bit

Sind die drei Jumper J1 - J3 gesetzt (A2=0, A1=0, A0=0), so hat der Baustein die folgenden Adressen:
Lesezugriff (Read): **0x41**
Schreibzugriff (Write): **0x40**

Viele der Anweisungs-Beispiele in dieser Dokumentation, basieren übrigens auf dieser Beispielsschaltung und können daher direkt ausprobiert werden.

Beispiele zum Ansteuern der LED (Write):

[LED1 – LED8 einschalten (LEDs leuchten, wenn Ausgänge Low sind)]
```
S40 00 P
```

[LED1 – LED8 ausschalten (LEDs leuchten nicht, wenn Ausgänge High sind)]
```
S40 FF P
```

[nur LED1 leuchtet]
```
S40 FE P
```

[ein bei LED1 startendes Lauflicht; nur je eine LED an; Licht läuft hin und wieder zurück, extrem schnell]
```
S40 FE FD FB F7 EF DF BF 7F 7f bf df ef f7 fb fd fe P
```

[ein bei LED1 startendes Lauflicht; nur je eine LED an; Licht läuft hin und wieder zurück, langsamer]
```
S40 FE L001f W FD L001f W FB L001f W F7 L001f W EF L001f W DF L001f W BF L001f W 7F L001f W 7f L001f W bf L001f W df L001f W ef L001f W f7 L001f W fb L001f W fd L001f W fe L001f P
```

[vorheriges Lauflicht als periodische Anweisungsfolge im Makrospeicher]
```
V00{ S40 FE L001f W FD L001f W FB L001f W F7 L001f W EF L001f W DF L001f W BF L001f W 7F L001f W 7f L001f W bf L001f W df L001f W ef L001f W f7 L001f W fb L001f W fd L001f W fe L001f P >00 }
```

Makrospeicher vorher löschen mit: **v00{}**
Nach dem Speichern der obigen Folge prüfen, ob alles drinnen ist: **U**
Anweisungsfolge im Makrospeicher starten: **>00**
Programmablauf wieder beenden: **<**

Beispiele zum Auslesen der Taster (Read):

[Taster1 – Taster8 einmal auslesen]
```
S41 01 P
```
Antwort (kein Taster gedrückt und keine LED leuchtet):
```
FF
```

[Taster1 – Taster8 zwölfmal nacheinander auslesen]
```
S41 0C P
```
Antwort (nur Taster 1 ist gedrückt):
```
FE FE FE FE FE FE FE FE FE FE FE FE
```

Beispiel zum abwechselnden Setzen der LEDs und Auslesen der Taster (Write & Read):

[LED1 bis LED8 nacheinander einschalten und gleich danach wieder auslesen]
```
S40FE [Taster 1: ]R01 WFD [Taster 2: ]R01 WFB [Taster 3: ]R01 WF7 [Taster 4: ]R01 WEF [Taster 5: ]R01 WDF [Taster 6: ]R01 WBF [Taster 7: ]R01 W7F [Taster 8: ]R01 P
```

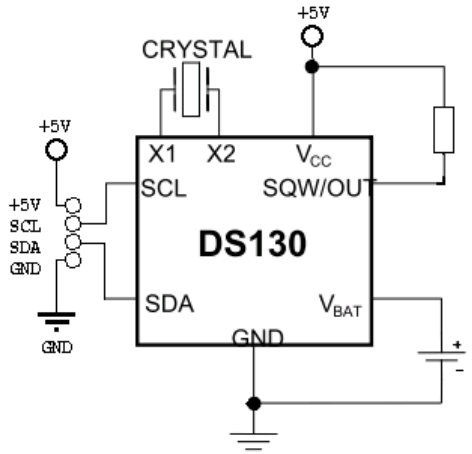
Antwort:
```
Taster 1: FE
Taster 2: FD
Taster 3: FB
Taster 4: F7
Taster 5: EF
```

```
Taster 6: DF
Taster 7: BF
Taster 8: 7F
```

4.5.2. Echtzeituhr mit dem DS1307

Bei dem DS1307 handelt es sich um einen Uhrenbaustein von der Firma Maxim. Dieser kann über den I²C-Bus gestellt werden und liefert dann Sekunde, Minute, Stunde, Monat, Wochentag und Jahr. Er berücksichtigt Schaltjahre und kann gestützt von einer Batterie über lange Zeit die gewünschten Informationen bereitstellen.
Die Daten des Bausteins liegen im BCD-Format vor, was bedeutet, dass sie in „HTerm“ direkt lesbar sind und nicht erst von Hexadezimal nach Dezimal umgewandelt werden müssen.
Da die genaue Beschreibung aller Funktion des DS1307 den Umfang dieser Dokumentation sprengen würde, sei an dieser Stelle auf das Hersteller-Datenblatt hingewiesen:
https://www.analog.com/media/en/technical-documentation/data-sheets/DS1307.pdf

Der folgende Schaltplan zeigt einen typischen Aufbau mit dem DS1307. Als Quarz muss ein Uhrenquarz mit 32768 Hz verwendet werden. Als Pufferbatterie ist eine Lithium-Knopfzelle (z.B. CR2032) zu verwenden.



In der folgenden Tabelle aus dem Datenblatt des DS1307 lassen sich die Register des Bausteins und deren Funktionen ersehen. Beim Schreiben von Zeitangaben bzw. von Konfigurationseinstellungen in den Baustein muss nach der Geräte-Adresse `0xD0` nur das erste Funktionsregister `0x00` adressiert werden. Die Daten werden dann nacheinander zum Baustein gesendet und dieser inkrementiert selbstständig die Adresse des Funktionsregisters.

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

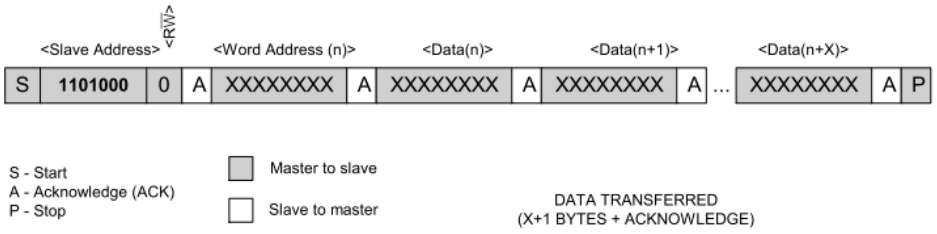
Die Uhr lässt sich starten, indem `0x00` ins Sekunden-Register an der Adresse `0x00` geschrieben wird, da dadurch auch das Bit 7 (CH) auf 0 gesetzt wird. Das CH-Bit steht für „Clock Halt“ und stoppt die Uhr, solange

es auf 1 gesetzt ist. Dem ersten Datenbyte folgen jetzt die Bytes zum Stellen der Minuten und der Stunden. Hierbei wird automatisch die 24-h-Betriebsart gewählt.

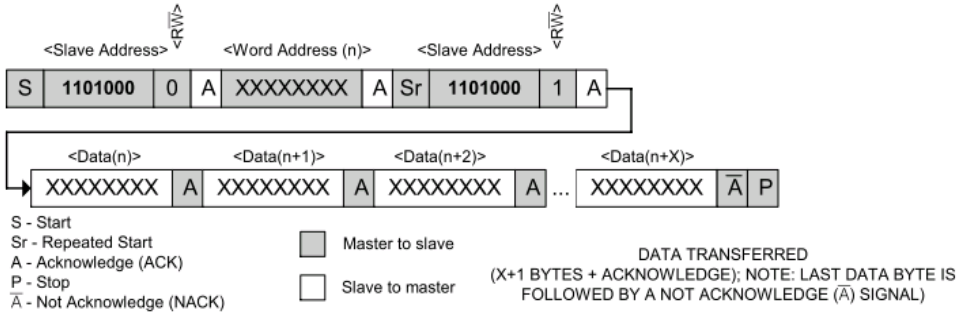
Ausgelesen wird die Uhr, indem zuerst einmal dem Uhrenbaustein das Register angegeben wird, ab dem die Zeitdaten später ausgegeben werden sollen. Daher soll die Uhr erstmal ein Byte lesen (das Register `0x00`) und danach 3 Datenbytes (Sekunden, Minuten und Stunden) auf den I²C-Bus schreiben.

Beispiele:

[Auf 16:45:00 Uhr stellen und starten (die Uhr soll die Daten vom Master lesen = Read)]
`SD0 00 00 45 16 P`



[Auslesen der Uhrzeit (die Uhr soll die Uhrzeit zum Master schreiben = Write)]
`SD0 00 P  SD1 03 P`
Antwort (Sekunden, Minuten, Stunden):
`11 45 16`



4.5.3. EEPROMs beschreiben/auslesen (24C01, 24C02...)

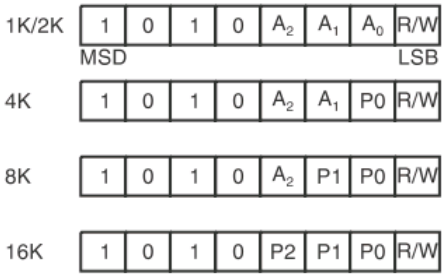
Das Auslesen und Beschreiben verschiedener EEPROM-Bausteine unterscheidet sich je nach Speichergröße und -aufteilung. Nach der Geräte-Adresse (Achtung, 24C04/08/16-EEPROMs haben mehrere) wird dem EEPROM die Adresse der auszulesenden bzw. der zu beschreibenden Speicherzelle mitgeteilt. Werden mehrere Datenbytes nacheinander ohne Stopp-Bedingung beschrieben (Page-Write) oder ausgelesen, so inkrementiert das EEPROM innerhalb einer Page die Adressen der Speicherzellen selbstständig.
Für weitergehende Informationen sei auch hier auf die zugehörigen Datenblätter der jeweiligen Chip-Hersteller verwiesen.

Die folgenden verbreiteten EEPROM-Typen haben folgende Speicheraufteilung:

- **24C01 (128 x 8 Bit = 1Kbit):**
  16 Pages mit je 8 Bytes, 7-bit Datenwort-Adresse (**1 Byte Speicheradresse**)
- **24C02 (256 x 8 Bit = 2Kbit):**
  32 Pages mit je 8 Bytes, 8-bit Datenwort-Adresse (**1 Byte Speicheradresse**)
- **24C04 (512 x 8 Bit = 4Kbit):**
  32 Pages mit je 16 Bytes, 9-bit Datenwort-Adresse (**P0-Bit in Geräteadresse + 1 Byte Speicheradresse**)
- **24C08 (1024 x 8 Bit = 8Kbit):**
  64 Pages mit je 16 Bytes, 10-bit Datenwort-Adresse (**P1/P0-Bit in Geräteadresse + 1 Byte Speicheradresse**)
- **24C16 (2048 x 8 Bit = 16Kbit):**
  128 Pages mit je 16 Bytes, 11-bit Datenwort-Adresse (**P2/P1/P0-Bit in Geräteadr. + 1 Byte Speicheradr.**)

Page-Write: Die 1K und 2K EEPROM-Typen können in einem Page-Write bis zu 8 Bytes nacheinander schreiben und die 4K, 8K und 16K Typen können in einem Page-Write bis zu 16 Bytes schreiben. Das Schreiben jeder Page dauert bis zu 5ms, weshalb hier der Wartebefehl (**L0005**) eingesetzt werden solle.

Die Zusammensetzung der **Geräte-Adressen** der EEPROMs ist in der folgenden Abbildung (nach Speichergröße) gezeigt. Es gibt das Read/Write-Bit (LSB), einen fest vorgegebenen 4-Bit großen Teil (binär: **1010** xxxx, hexadezimal: **A**x), je nach Speichergröße bis zu 3 Bit (A2, A1, A0), die sich frei per Hardware konfigurieren lassen und bis zu 3 Bit (P2, P1, P0), die je nach Speichergröße die MSB (Most Significant Bit) der Speicheradresse enthalten. Je größer ein Speicherbaustein ist, desto mehr Bit der Geräteadresse (P2, P1, P0) dienen der Adressierung der Speicherbereiche. Hieraus ergibt sich die Tatsache, dass von einem 24C16 nur einer, während von einem 24C01 oder einem 24C02 bis zu 8 Bausteine an einem I²C-Bus zusammen betrieben werden können.
In der nachfolgenden Tabelle sind die jeweils möglichen Geräte-Adressen angegeben, die mit Hilfe der Hardware-Kodierung gewählt werden können. Dabei ist jeweils die hexadezimale Geräte-Adresse zur Adressierung der ersten 256-Byte-Page angegeben. Die angegebenen Adressen ändern sich bei der Adressierung der anderen Speicherbereiche.



EEPROM	Speicheradressierungs-Bit	Mögliche Basis-Geräte-Adressen für die erste 256-Byte-Page (Write-Adresse/Read-Adresse)
24C01, 24C02		0xA0/0xA1, 0xA2/0xA3, 0xA4/0xA5, 0xA6/0xA7, 0xA8/0xA9, 0xAA/0xAB, 0xAC/0xAD, 0xAE/0xAF
24C04	P0=0	0xA0/0xA1, 0xA4/0xA5, 0xA8/0xA9, 0xAC/0xAD
24C08	P1=0, P0=0	0xA0/0xA1, 0xA8/0xA9
24C16	P2=0, P1=0, P0=0	0xA0/0xA1

Beispiele zum Beschreiben:

[Speicherzelle 0x00 mit 0xAA beschreiben (24C01 und 24C02 mit Geräteadresse A2=A1=A0=0]
SA0 00 AA P

[Speicherzelle 0x0100 auf 2. Page mit 0xAA beschreiben (24C04 mit Hardware-Adressbit A2=A1=0, 24C08 mit Hardware-Adressbit A2=0 und 24C16 ohne Hardware-Adr.bit, alle EEPROMs mit Geräteadresse: 0xA0]
SA2 00 AA P

[32 Bytes im Page-Write-Modus schreiben (24C01 und 24C02)]
sA0 00 A1 A2 A3 A4 A5 A6 A7 A8 p L0005 w08 B1 B2 B3 B4 B5 B6 B7 B8 p L0005 w10 C1 C2 C3 C4 C5 C6 C7 C8 p L0005 w18 D1 D2 D3 D4 D5 D6 D7 D8 p

[32 Bytes im Page-Write-Modus in 1.Page schreiben (24C04, 24C08 und 24C16) – Achtung: mit den Befehlen „**w**“ und „**r**“ wird immer dieselbe Page adressiert, die zuvor mit der Startadresse adressiert wurde!]
sA0 00 A1 A2 A3 A4 A5 A6 A7 A8 B1 B2 B3 B4 B5 B6 B7 B8 p L0005 w10 C1 C2 C3 C4 C5 C6 C7 C8 D1 D2 D3 D4 D5 D6 D7 D8 p

[Mit 256 Bytes den Speicher eines 24C02 im Page-Write-Modus komplett füllen (3 Anweisungsfolgen]
[1. Eingabe:]
w00 00 01 02 03 04 05 06 07 p L0005 w08 08 09 0A 0B 0C 0D 0E 0F p L0005 w10 10 11 12 13 14 15 16 17 p L0005 w18 18 19 1A 1B 1C 1D 1E 1F p L0005 w20 20 21 22 23 24 25 26 27 p L0005 w28 28 29 2A 2B 2C 2D 2E 2F p L0005 w30 30 31 32 33 34 35 36 37 p L0005 w38 38 39 3A 3B 3C 3D 3E 3F p L0005 w40 40 41 42 43 44 45 46 47 p L0005 w48 48 49 4A 4B 4C 4D 4E 4F p L0005 w50 50 51 52 53 54 55 56 57 p L0005

[2. Eingabe:]
w58 58 59 5A 5B 5C 5D 5E 5F p L0005 w60 60 61 62 63 64 65 66 67 p L0005 w68 68 69 6A 6B 6C 6D 6E 6F p L0005 w70 70 71 72 73 74 75 76 77 p L0005 w78 78 79 7A 7B 7C 7D 7E 7F p L0005 w80 80 81 82 83 84 85 86 87 p L0005 w88 88 89 8A 8B 8C 8D 8E 8F p L0005w90 90 91 92 93 94 95 96 97 p L0005 w98 98 99 9A 9B 9C 9D 9E 9F p L0005 wA0 A0 A1 A2 A3 A4 A5 A6 A7 p L0005 wA8 A8 A9 AA AB AC AD AE AF p L0005

[3. Eingabe:]
wb0 B0 B1 B2 B3 B4 B5 B6 B7 p L0005 wB8 B8 B9 BA BB BC BD BE BF p L0005 wc0 C0 C1 C2 C3 C4 C5 C6 C7 p L0005 wC8 C8 C9 CA CB CC CD CE CF p L0005 wd0 D0 D1 D2 D3 D4 D5 D6 D7 p L0005 wD8 D8 D9 DA DB DC DD DE DF p L0005 we0 E0 E1 E2 E3 E4 E5 E6 E7 p L0005 wE8 E8 E9 EA EB EC ED EE EF p L0005 wf0 F0 F1 F2 F3 F4 F5 F6 F7 p L0005 wF8 F8 F9 FA FB FC FD FE FF

Beispiele zum Auslesen:

[Auslesen aller Speicherzellen eines 24C01 per „Sequential-Read“ (0x80 = 128 Byte)]
SA0 00 R80 P
Antwort:
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B
3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59
5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77
78 79 7A 7B 7C 7D 7E 7F

[Auslesen aller Speicherzellen eines 24C02 per „Sequential-Read“ (256 Bytes)]
SA0 00 R80 R80 P
Antwort:
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B
3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59
5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77
78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95
96 97 98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3
B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1
D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF

[Auslesen einer bestimmten Speicherzelle - „Random-Read“ (24C01 und 24C02)]
```
SA0 00 R01 P
```
Antwort (z. B.):
```
0F
```

[Auslesen einer bestimmten Speicherzelle - „Random-Read“ (24C04, 24C08 und 24C16)]
```
SA0 00 R01 P
```
Antwort (z. B.):
```
0F
```

[Aktuelle Speicherzelle auslesen - „Current-Address-Read“ (24C01, 24C02, 24C04, 24C08 und 24C16)]
```
SA1 01 P
```
Antwort (z. B.):
```
01
```
Hinweis:
Beim Beschreiben eines EEPROM bietet es sich eventuell an, zur Kontrolle testweise mal den Parameter Y31 zu setzen (siehe Kapitel 4.2.5.). Dieser gibt die ACK/NACK-Antworten des EEPROMs weiter an den PC. Dadurch kann man sehen, ob die Schreib-Anweisungen z.B. zu schnell zum EEPROM gesendet werden, und dieses die Daten nicht alle speichern kann (Antwort ist dann N statt K).

4.5.4. Thermometer-Sensor (DS75, LM75, TMP101)

Eine ganze Reihe von I²C-Bausteinen dient der Messung von Umwelteinflüssen, wie z.B. Temperatur, Luftfeuchtigkeit und Luftdruck. Zur Messung der Temperatur gibt es beispielsweise die sich recht ähnlichen DS75, LM75 und TMP101.

Neben ähnlichen Messeigenschaften, gleicher Geräte-Adresse und einer weitestgehend gleichen Ansteuerung, besitzen alle ein Register zur Konfiguration. Deren Einstellungen sollten vor der eigentlichen Messung beachtet werden. Dazu finden sich detaillierte Informationen in den zugehörigen Datenblättern der Hersteller.

Um dieses Register konfigurieren zu können, muss es (nach der Adressierung des Gerätes) mit dem Pointer-Register `0x01` adressiert werden. In den folgenden Beispielen sind die 3 frei belegbaren Adresspins am Baustein mit GND verbunden (A0=0, A1=0, A2=0). Daraus ergeben sich für die drei Temperatursensor-Bausteine LM75, DS75 und den TMP101 die Lese-Adresse `0x91` und die Schreib-Adresse `0x90`.

Die Messergebnisse werden in einem 2 Byte großen Temperatur-Registern bereitgestellt, dessen Format sich von Baustein zu Baustein unterscheidet und daher für die Auswertung im jeweiligen Datenblatt nachzulesen ist.

Am Beispiel des **LM75** sieht die Konfiguration des Sensors mit der **Einstellung** `0x00` beispielsweise folgendermaßen aus:
```
S90 01 00 P
```

Der **LM75** schreibt alle 100 bis 300ms einen aktuellen Temperaturwert (Hexadezimal) in seine Temperatur-Register. Zum **Ausgelesen** muss das Pointer-Register `0x00` sein. Das erste Temperatur-Register enthält die Temperatur in vollen Grad Celsius (MSB enthält das Vorzeichen) und im zweiten Register gibt das MSB (Bit 8) den Wert der Nachkommastelle in 0,5°C an (die restlichen 7 Bits im 2. Byte sind undefiniert):
```
S90 00 R 02 P
```
Antwort (gemessen: 20,5 Grad Celsius):
```
14 80
```

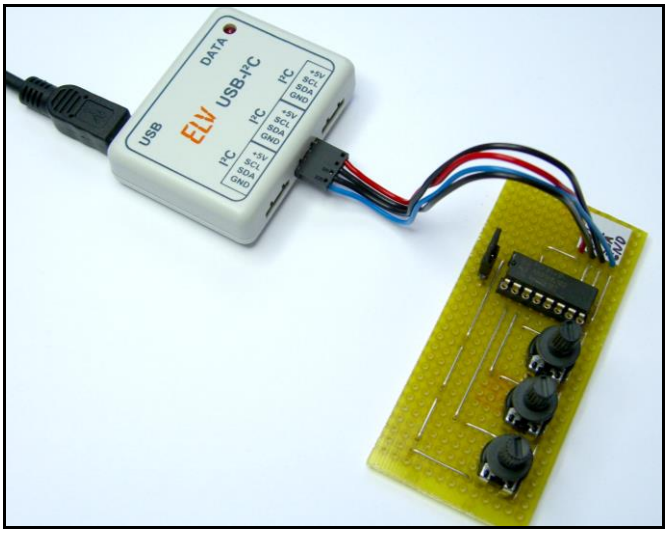
Geschrieben als selbst laufendes Makro, welches **jede Sekunde eine Messung zum PC sendet**:
```
V00{S90 00 R 02 P L03E8 >00}
```
Gestartet wird das Makro mit:
```
>00
```

Antwort (gemessen: +1,5; +1,0; +0,5; 0,0; -0,5; -1,0...):

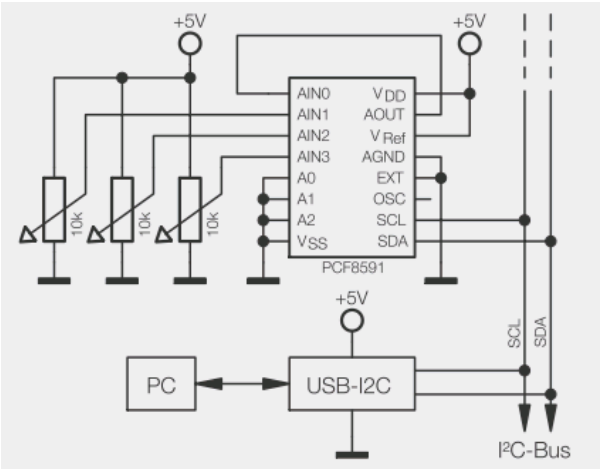
```
01 80
01 00
00 80
00 00
80 80
81 00
```

4.5.5. A/D-Wandler mit dem PCF8591

In der folgenden Abbildung ist ein Testaufbau mit dem 8-Bit-A/D-D/A-Konverter PCF8591 dargestellt. Die zugehörige Beispielschaltung darunter abgebildet. Der PCF8591 kann, am USB-I²C-Interface angeschlossen, zur analogen Datenaufnahme über 4 Kanäle und zur Ausgabe einer analogen Spannung genutzt werden.



Im Beispiel sind die drei Adresspins A2, A1 und A0 des PCF8591 mit GND verbunden, wodurch die jeweiligen Bits auf 0 gesetzt sind. Es ergeben sich die Leseadresse `0x91` und die Schreibadresse `0x90`.
Zur Konfiguration des PCF8591 muss erst die Schreibadresse `0x90` und danach das Konfigurationsbyte `0x05` übertragen werden. Dessen genaue Bedeutung sollte im zugehörigen Datenblatt nachgelesen werden.



Zum **Auslesen** der 4 A/D-Wandler-Kanäle ergibt sich die folgende Anweisungsfolge:

```
S90 05 R04 P
```

Antwort (AIN0 ist undefiniert, AIN1 und AIN2 sind auf GND, AIN3 ist auf +5V):

```
A2 00 00 FF
```

Sollen die Messwerte noch mit Kommentaren formatiert werden, dann lautet die Eingabe z. B. so:

```
S90 05 [AIN0: ]R01 [AIN1: ]R01 [AIN2: ]R01 [AIN3: ]R01 P
```

Antwort (AIN0 ist undefiniert, AIN1 und AIN2 sind auf GND, AIN3 ist auf +5V):

```
AIN0: A2
AIN1: 00
AIN2: 00
AIN3: FF
```

Wenn die Messung nun periodisch alle 500 ms (kürzere Wartezeit = 478ms = 0x01DE, da die Programmausführung auch etwas Zeit benötigt) durchgeführt werden soll, kann die Anweisungsfolge leicht angepasst und in den Makrospeicher kopiert werden:

```
V00{S90 05 [AIN0: ]R01 [AIN1: ]R01 [AIN2: ]R01 [AIN3: ]R01 P.L01DE >00}
```

Soll mit derselben Schaltung eine analoge Spannung auf den **D/A-Wandler-Ausgang** (AOUT) ausgegeben werden, ist eine Schreibroutine notwendig. Möchte man beispielsweise die halbe Referenzspannung (Vref / 2 = 2,5 V; 0xFF / 2 = 0x7E) ausgeben, ergibt sich die folgende Befehlsfolge:

```
S90 45 7E P
```

Da der Ausgangswerte gleichzeitig am Eingang AIN0 anliegt, kann die neu eingestellte analoge Ausgangsspannung quasi wieder „zurück gelesen“ werden:

```
S90 05 R04 P
```

Antwort (an AIN0 liegen nun in etwa 2,5 V; AIN1-AIN2 sollen jetzt auf GND liegen):

```
81 00 00 00
```

4.6. Verklemmung (Deadlock) auf dem I²C-Bus

Im Falle einer **I²C-Verklemmung** („Deadlock“) kann der Master normalerweise keine Stopp-Bedingung mehr ausführen, wodurch jede weitere Kommunikation mit den angeschlossenen Slaves unmöglich wird.

Eine solche Verklemmung auf dem I²C-Bus kann beispielsweise geschehen, wenn ein angeschlossener Slave erwartet, dass als nächstes weitere Bits von ihm gelesen werden sollen (durch falsche Anweisungsfolge oder durch Störung auf Signalleitung) und er die SDA-Leitung dafür auf Low-Pegel hält. Der Master möchte wiederum auch etwas auf die SDA-Leitung schreiben und wartet nun darauf, dass die SDA-Leitung vom Slave freigegeben wird. Das System ist nun verklemmt (alle warten auf etwas) und eine weitere Datenübertragung ist nicht mehr möglich.

Um das zu verhindern, versucht das USB-I²C-Interface eine erkannte Verklemmung (bei der Ausführung der Start-Bedingung) automatisch wieder aufzulösen. Dazu gibt es 8 zusätzliche Takt-Pulse auf die SCL-Leitung, sobald die Stopp-Bedingung nicht mehr ausführbar sein sollte (also wenn die SDA-Leitung vom Slave auf Low-Pegel gehalten wird). Dadurch „schreibt“ ein Slave seine Datenbits heraus und gibt anschließend die SDA-Leitung wieder frei.

Die Kommunikation kann nun weitergehen und das USB-I²C-Interface sendet zur Kenntnisnahme die folgende Fehlermeldung zum PC:

```
Solve I2C-Bus-Lock
```

5. Verzeichnis der Korrekturen seit Februar 2009

Datum	Kapitel/FW	Änderung
22.01.2010	4.5.3	Korrektur in der Beschreibung der Speicheraufteilung 24C04/08/16
19.01.2011	4.5.5	Erweiterung der Beschreibung des Beispiels mit LogView
04.03.2011	FW 1.7	Mikrocontroller getauscht auf ATmega88PA, ohne Funktionsänderungen
28.03.2011	4.1.1	Korrektur der Adresse in binärer Schreibweise
06.04.2011	4.4.1	Korrektur von Schreib- und Leseadresse
06.04.2011	4.5.3	Beschreibung und Beispiele zur Speicheradressierung korrigiert und erweitert
19.08.2011	4.2.	Reset-Befehl war an einer Stelle fälschlicherweise als 4F beschrieben
23.05.2012	4.5.2	Korrektur der Antwort im Beispiel des RTC-Bausteins
02.04.2024	2. - 2.3	Anpassung der Installationsbeschreibung an aktuelle Systeme
02.04.2024	4.1.1	Adresse in Beispiel korrigiert
02.04.2024	4.4, 4.5.5	Entfernen von Beschreibungen zum nicht mehr gepflegten LogView
02.04.2024	4.5.1, 4.5.2	Aktualisierung von Links