# Universalist I²C

## oder: Wie ein Bussystem die Elektronik eroberte

**I²C ist ein geniales serielles Bussystem, das bereits eine lange Geschichte hat und vieles in der Elektronik einfacher realisierbar macht. Wir beschäftigen uns mit der I²C-Geschichte, der Topologie, dem Übertragungsprotokoll und vor allem mit der grundlegenden Praxis rund um das heute allgegenwärtige Bussystem.**


I²C-Logo:©NXP

### Master-Slave-Beziehung seit 1982

Als Philips (heute NXP) im Jahr 1982 den I²C-Bus (englisch: Inter-Integrated Circuit, Deutsch umgangssprachlich „I-Zwo-C", richtig: „I-Quadrat-C") als Master-Slave-Bus mit bidirektionaler synchroner serieller Datenübertragung einführte, ging es in erster Linie darum, ein einfach beherrschbares Kommunikationssystem zwischen komplexen Elektronikbausteinen in Consumergeräten zu schaffen. Zuerst wurde I²C in Fernsehgeräten eingesetzt, später auch in anderen komplexen Geräten wie z. B. hochentwickelten Autoradios, wie der Schaltungsausschnitt eines solchen RDS-Radios von 1996 in Bild 1 zeigt. Es entstanden bereits ganze Reihen mit dieser Schnittstelle ausgestatteter Spezialchips, allen voran Mikrocontroller- und Speicherbausteine, Displayansteuerungen, Multiplexer und Portexpander – vor allem überall da, wo in der Zeit davor aufwendige Parallelbussysteme dominierten. Das sicherlich bekannteste Duo ist der kleine Mikro-

controller und sein serielles, nichtflüchtiges EEPROM zur Speicherung von Parametern und Daten. Das Prinzip wurde von zahlreichen anderen Herstellen übernommen, und so kam es u. a. aus lizenzrechtlichen Gründen auch zu unterschiedlichen Bezeichnungen des gleichen Protokolls. Atmel führte dazu die Bezeichnung TWI (Two Wire Interface) ein.

Seit 1982 und insbesondere seit der Einführung als Industriestandard 1992 hat der Bus mehrere Spezifikationen erfahren, die vor allem die Übertragungsraten steigerten und die Datenübertragung perfektionierten. Zwar finden in normalen Anwendungen der Mikrocontrollertechnik immer noch hauptsächlich die Normal-Mode-Rate von 100 kbit/s und die Fast-Mode-Rate von 400 kbit/s ihre Anwendung, aber heute kann man eine unidirektionale Übertragungsrate von bis zu 5 Mbit/s und eine bidirektionale Datenrate bis zu 3,4 Mbit/s realisieren.

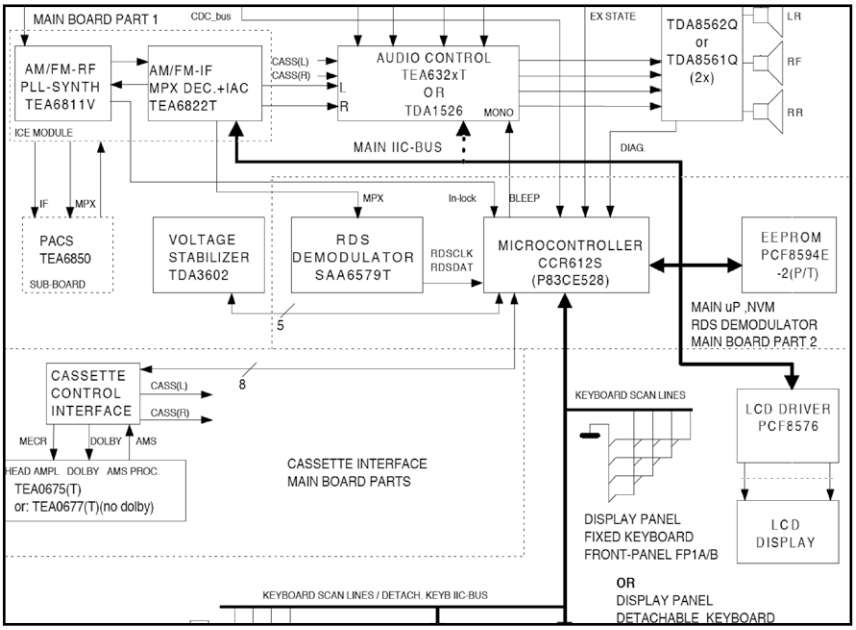


*Bild 1: Einer der Urahnen für die Anwendung des I²C-Busses in der Consumerelektronik, hier in einem Autoradio. Bild: Philips/NXP*

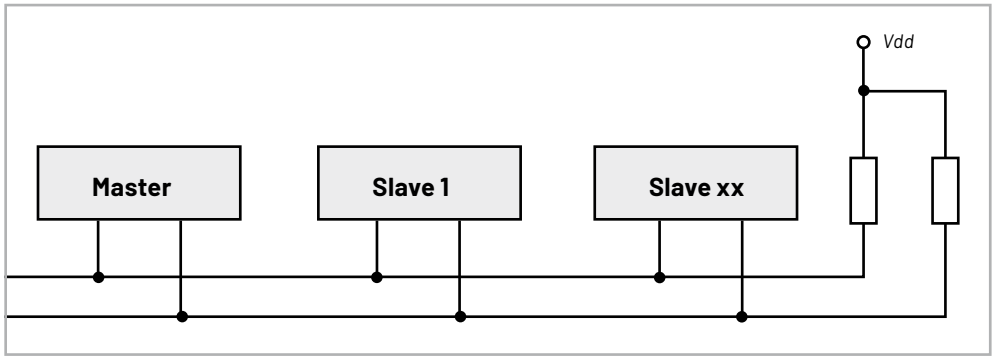


*Bild 2: Die Busarchitektur von I²C*

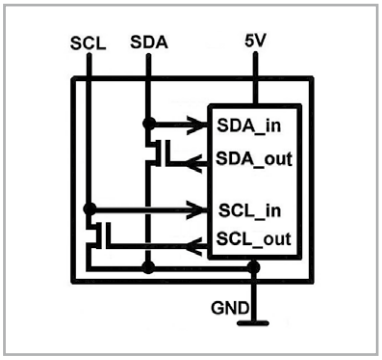


*Bild 3: Das I²C-Hardware-Interface*

## Das Bussystem und die Kommunikation

Die Bustopologie von I²C ist übersichtlich (Bild 2). Das grundlegende Bussystem besteht aus einem Master (Mikrocontroller) und einem oder mehreren Slave-Bausteinen sowie einem Pull-up-Widerstand auf jeder Leitung. Alle Bausteine sind über einen zweiadrigen Bus verbunden, der aus einer Taktleitung (SCL, Serial Clock Line) und einer Datenleitung (SDA, Serial Data) besteht. Der vom Master erzeugte Systemtakt sorgt für eine synchrone Arbeit aller beteiligten Bausteine. Die Informationen (Daten) werden seriell über die SDA-Leitung übertragen.

Das Master-Slave-Prinzip bedeutet auch hier, dass stets der Master die Kommunikation mit den Slaves startet. Das gilt auch für die später noch diskutierten Slaves mit Interruptsignalen. Dabei entscheidet das letzte Bit in der Adressierung der Slaves, ob eine Übertragung (Write=0) an den Slave erfolgen soll oder eine Information aus diesem ausgelesen werden soll (Read=1). Das werden wir noch genauer beleuchten.

Das eigentliche Hardware-Interface sieht schaltungstechnisch überall gleich aus (Bild 3). Als Ausgänge sind immer Open-Collector-Ausgänge (OC) realisiert. Die Eingänge werden oft in der Schaltungspraxis zusätzlich durch einen Schutzwiderstand gesichert. Die Pull-up-Widerstände liegen im Standardfall bei 4,7 kΩ, können in bestimmten Einsatzfällen aber auch niedriger sein. Die Logik im System ist positiv, im Ruhezustand liegen beide Busleitungen also auf HIGH (logisch 1). Es gibt keinen festen Logikpegel, lediglich die Bedingungen: LOW ist <0,3 x $V_{DD}$, HIGHh ist >0,7 x $V_{DD}$. Somit können im I²C-System auch Bausteine mit unterschiedlichen Betriebsspannungen arbeiten, etwa 5 V und 3,3 V. Hier müssen lediglich Level-Shifter für eine Umsetzung sorgen.

| Verfügbare I²C-Bausteine bei ELV | Artikel-Nr. |
|---|---|
| ELV USB-I²C-Interface | 092255 |
| ELV Intelligentes Schrittmotor Treibermodul iSMT8 | 092720 |
| ELV Triggergenerator TG1 für SPI/I2C/UART | 142124 |
| ELV I²C-BUS Displaymodul I²C-LCD | 099253 |
| ELV I²C-Realtime-Clock I²C-RTC | 103413 |
| ELV FM-Receiver-Modul mit Si4705 | 140984 |
| ELV LED-I²C-Steuertreiber, 16 Kanäle | 098377 |
| ELV Lichtsensor OPT3001 mit I²C-Schnittstelle I²C-LS | 152106 |
| ELV 3-Achsen-Beschleunigungssensor 3D-BS | 104893 |
| Joy-IT Analog-Digital-Converter 16Bit I²C ADS1115 | 145166 |
| Joy-IT Display-Modul 6,6 cm (2,6") I²C | 127513 |
| Joy-IT Luftqualitätssensor (VOC), CCS811 Sensor | 251973 |
| Joy-IT 2,44-cm-(0,96")-OLED-Display, 128 x 64 Pixel | 251189 |

Bild 4 zeigt die Grundlage des Zusammenwirkens der Daten- und Taktleitung im Bussystem. Egal, welcher Zustand auf der Datenleitung herrscht, die Daten sind nur gültig, solange das Taktsignal auf HIGH liegt. Gleichermaßen gilt: Ein Wechsel des Pegels auf SDA ist nur möglich, solange SCL auf LOW liegt.

Soll nun eine Datenübertragung starten, muss vom Master (und nur von diesem aus) eine Startbedingung erzeugt werden. Diese ist in Bild 5 links veranschaulicht. Der Master signalisiert den Start mit einem HIGH-LOW-Wechsel auf SDA und hält gleichzeitig SCL auf HIGH. Ähnlich wird die Stopp-Bedingung nach einem Datenübertragungszyklus erzeugt. Wieder muss SCL auf HIGH liegen und SDA dabei von LOW auf HIGH wechseln.

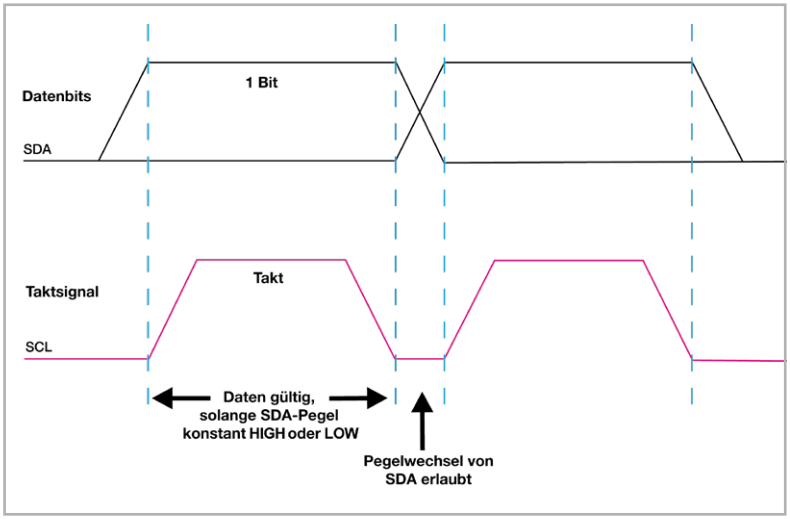


Bild 4: Das grundlegende Zusammenspiel zwischen Daten- und Taktleitung

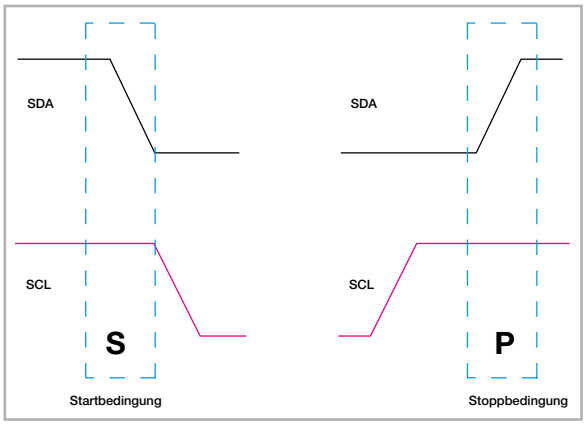


Bild 5: Die Start-/Stopp-Bedingungen am I²C-Bus

Bild 6 zeigt schließlich anhand einer Anforderung eines Datenbytes durch den Master von einem Slave den kompletten Ablauf eines Datenaustauschs. Mit der Startbedingung erfolgt quasi ein „Aufwecken" aller Busteilnehmer, die nun auf Ansprache warten. Der Master beginnt die Datenübertragung also mit der Adresse des Slaves, von dem er Daten auslesen möchte. Und dazu teilt er mit, ob er lesen oder schreiben

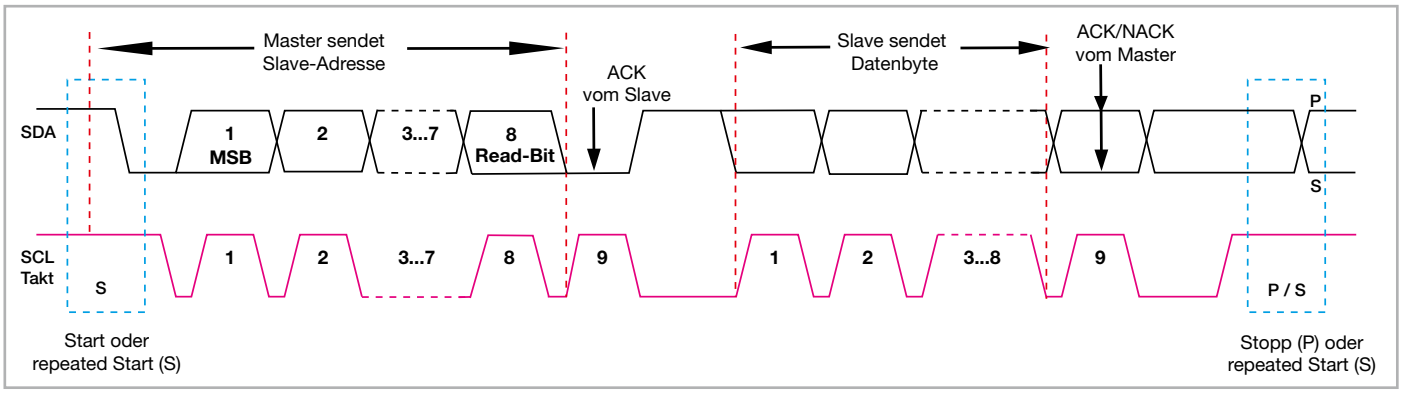


Bild 6: Ein kompletter Datenübertragungszyklus, bei dem der Master vom Slave 1 Datenbyte anfordert und erhält.

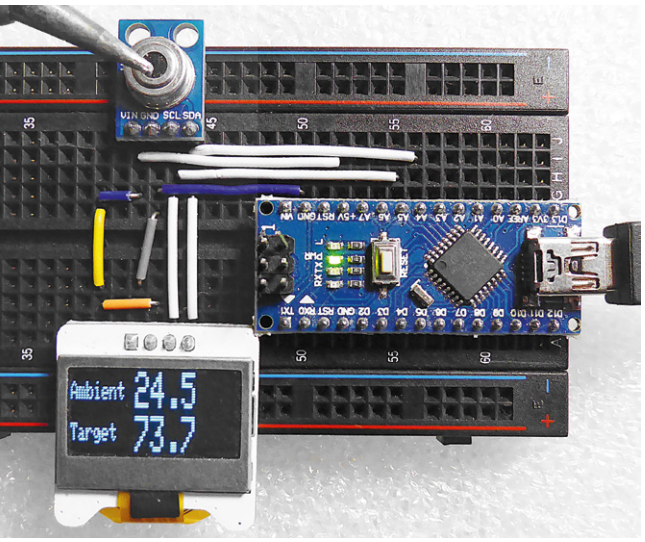


Bild 7: Eine typische, kleine Mikrocontroller-Applikation mit einem Arduino Nano als Master, einem Infrarot-Temperatursensor MLX90614 und einem OLED-Display als Slaves am I²C-Bus

möchte (R/W). Der angesprochene Slave bestätigt mit dem ACK-Signal, dass er bereit ist – alle anderen Slaves beenden die Kommunikation. Dann vervollständigt der Master die Adressierung und der Slave gibt das angeforderte Datenbyte aus. Hat der Master das Datenbyte vollständig empfangen, sendet er eine Bestätigung (ACK), wenn nicht, ein NACK. Die Sequenz endet mit der Stopp-Bedingung. Auf die vollständige Adressierung wollen wir hier nicht näher eingehen, sie ist u. a. unter [1] ausführlich erklärt. Bis zu 128 Geräte am Bus können per I²C angesprochen werden.

## I²C in der Praxis

In der Mikrocontroller-Praxis sind unzählige Sensor-, Interface- und Umsetzer-Chips verfügbar, die über ein I²C-Interface verfügen.

### Adresse ermitteln

Bild 7 zeigt eine typische kleine Microcontroller-Applikation mit einem Arduino Nano als Master und einem Infrarot-Temperatursensor MLX90614 sowie einem OLED-Display als Slaves am I²C-Bus. Insgesamt ergibt dies ein IR-Thermometer, das gleichzeitig die Umgebungs-

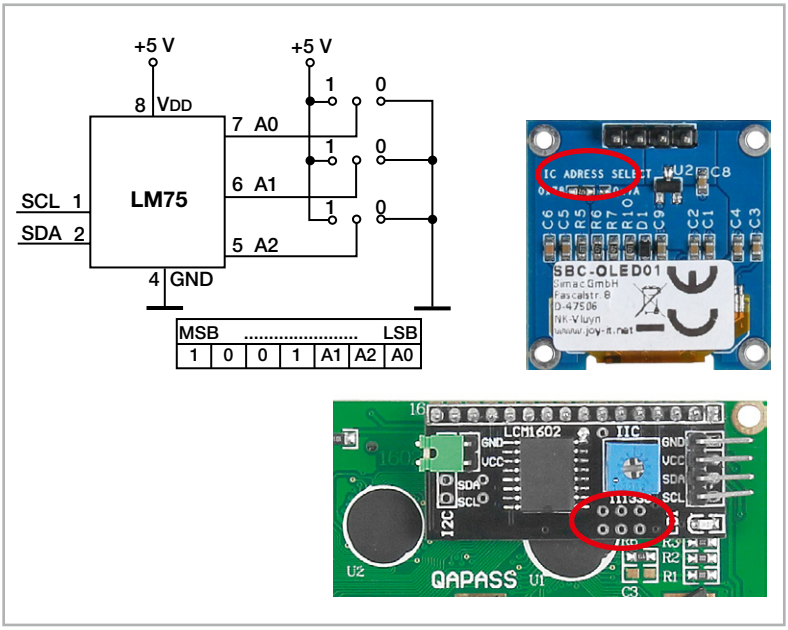


*Bild 8: Die unterschiedlichen Adressierungsmöglichkeiten von I²C-Bausteinen mit Hardware-adressierung*

temperatur (Ambient) und die Objekttemperatur des gemessenen Objekts (Target) anzeigt.

Die meisten dieser Bausteine sind entweder fest adressiert oder ihre Adresssierung kann über Adresspins gewählt werden, wie in Bild 8 für den Temperatursensor LM75 und für ein 0,96"-OLED-Display sowie ein Interface für die Ansteuerung von 1602/2004-LC-Displays gezeigt.

Andere Bausteine, wie unser MLX90614, haben eine feste Adresse, die dann auch in der zugehörigen Chip-Bibliothek, die in Bild 9 zu sehen ist, verankert ist. Diese Bibliotheken (Libraries, abgekürzt: libs) machen es dem Programmierer auch leicht, den Baustein in sein Programm zu integrieren. Sie sind für AVRs, STM32, ESP usw. universell einsetzbar. Somit braucht er sich nach den Set-up-Routinen nur auf sein eigentliches Programm zu konzentrieren.

Bild 10 zeigt einen Ausschnitt eines solchen Set-ups in der Arduino-IDE. Der MLX90614 hat die feste Adresse 0x5A (siehe Bild 9), das Display belegt per in die zugehörige Library eingegliederten Bibliotheken SPI.h und Wire.h die Adresszuweisung 0x3C.

Was aber, wenn man die Adresse des eingesetzten Bausteins nicht kennt? Hier hilft ein I²C-Adress-Scanner (Bild 11), wie man ihn in einfacher Form u. a. unter [2] findet. Er wird auf einen Mikrocontroller, der als Master fungiert, übertragen. Dieser fragt dann alle im Bus vorhandenen Bausteine ab, ermittelt deren Adressen und zeigt diese in einem Terminal, hier dem seriellen Monitor der Arduino-IDE, an. Wenn man diese Aufgabe öfter zu lösen hat, lohnt es sich, einen preiswerten Arduino Pro Mini oder Pro Micro oder einen RP2040 dafür zu programmieren und bei Bedarf schnell auf das Breadboard zu stecken.

```
#include <Adafruit_MLX90614.h>  // Library für MLX90614 aufrufen
#include <Adafruit_GFX.h>       // Grafik-Library für den Zeichensatz
#include <Adafruit_SSD1306.h>   // Library für den Display-Chipsatz SSD1306 aufrufen

Adafruit_SSD1306 display(128, 64);           //Display definieren
Adafruit_MLX90614 mlx = Adafruit_MLX90614(); //MLX90614 definieren


void setup()
{
  delay(100);  // Display initialisieren einleiten
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  //  Display mit der I2C-Adresse 0x3C initialisieren
  display.clearDisplay();  // Display-Buffer leeren
  display.setTextColor(WHITE);  // Textfarbe festlegen
  mlx.begin();  //MLX90614 starten
}
```

*Bild 10: Im Anwendungsprogramm werden die benötigten Libraries eingebunden und Adressierungen sowie Startbedingungen für die beteiligten Komponenten festgelegt.*

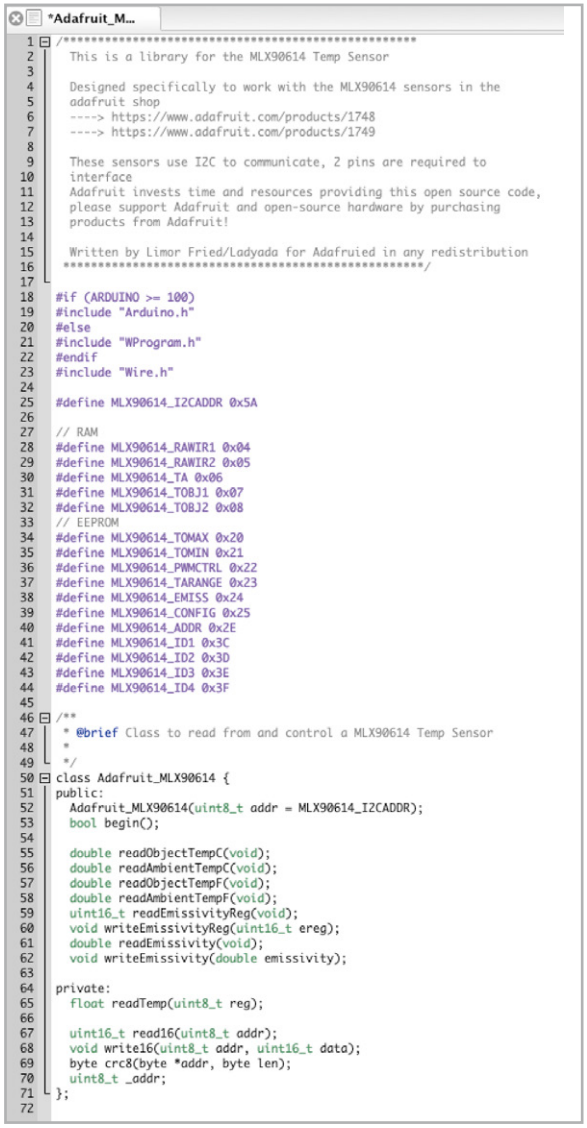


*Bild 9: In solchen Libraries sind alle grundlegenden Adressierungen, Syntaxbeschreibungen und Hardware-Definitionen zusammengefasst. Quelle: Adafruit Industries*

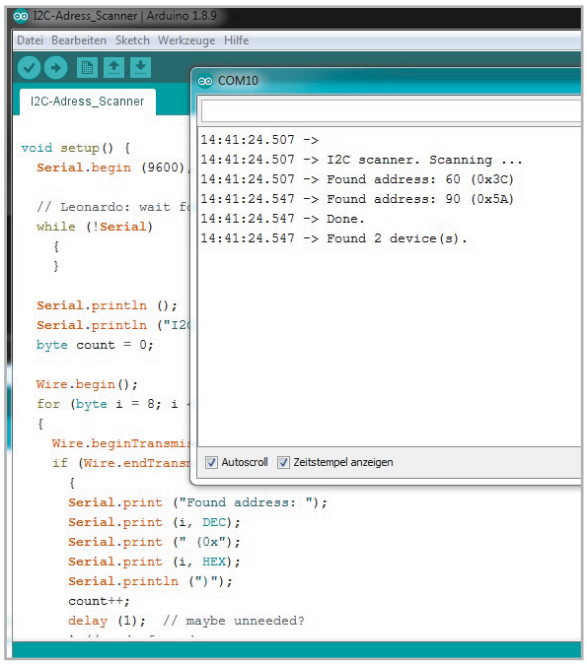


*Bild 11: Mit solch einem I²C-Adress-Scanner kann man die Adressen und die Anzahl sowie bei umfangreicheren Programmen auch den Bustakt der beteiligten Bausteine ermitteln. Programm: arduino.cc/Nick Gammon*

**Mehrere Slaves mit gleicher Adresse?**

Will man etwa ein Bussystem mit mehreren Temperatursensoren, z. B. dem SHT75, oder zahlreichen LED-Displays aufbauen, die eine feste und gleiche Adresse haben, lässt sich das Problem mit einem Bus-Switch mit integriertem I²C-Controller wie dem in Bild 12 dargestellten PCA9646 lösen. Der wird so konfiguriert, dass ihm für jeden Busteilnehmer quasi virtuelle Adressen zugeteilt werden, über deren Aufruf die abgeschlossenen Busteilnehmer jeweils einzeln zugeschaltet werden.

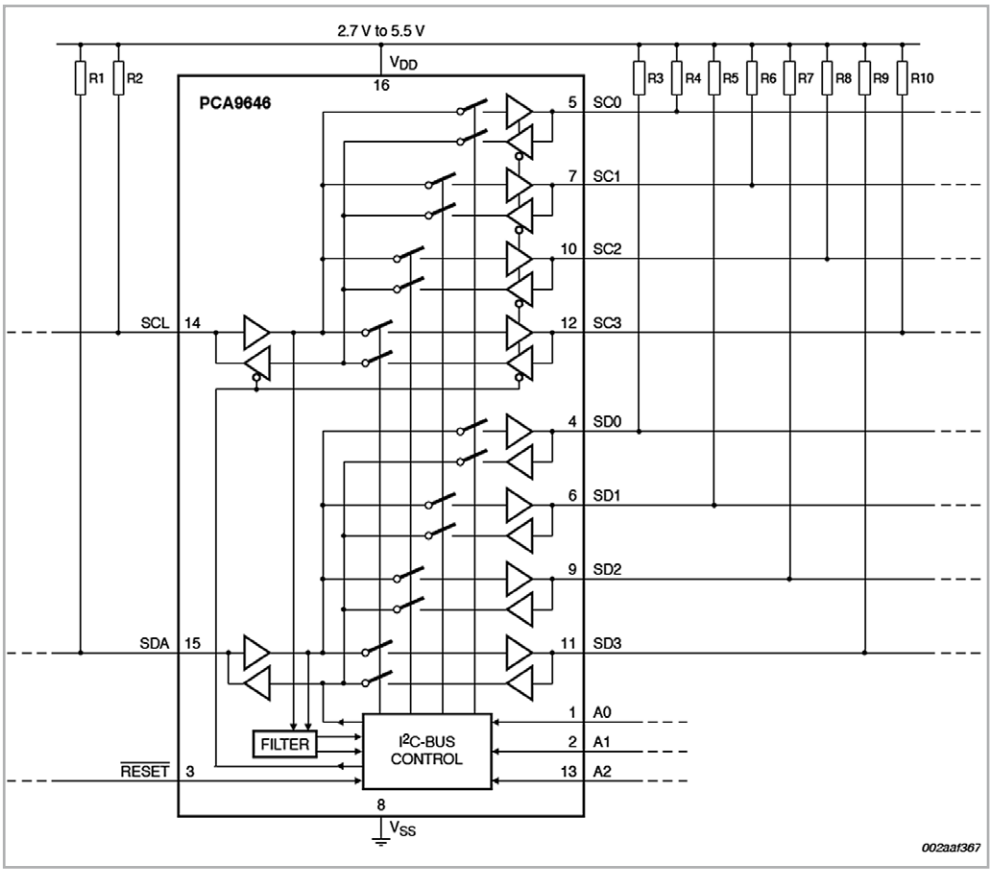


*Bild 12: Der Bus-Switch PCA9646 kann bis zu vier Sub-I²C-Buskonfigurationen bidrektional steuern. Bild: NXP*

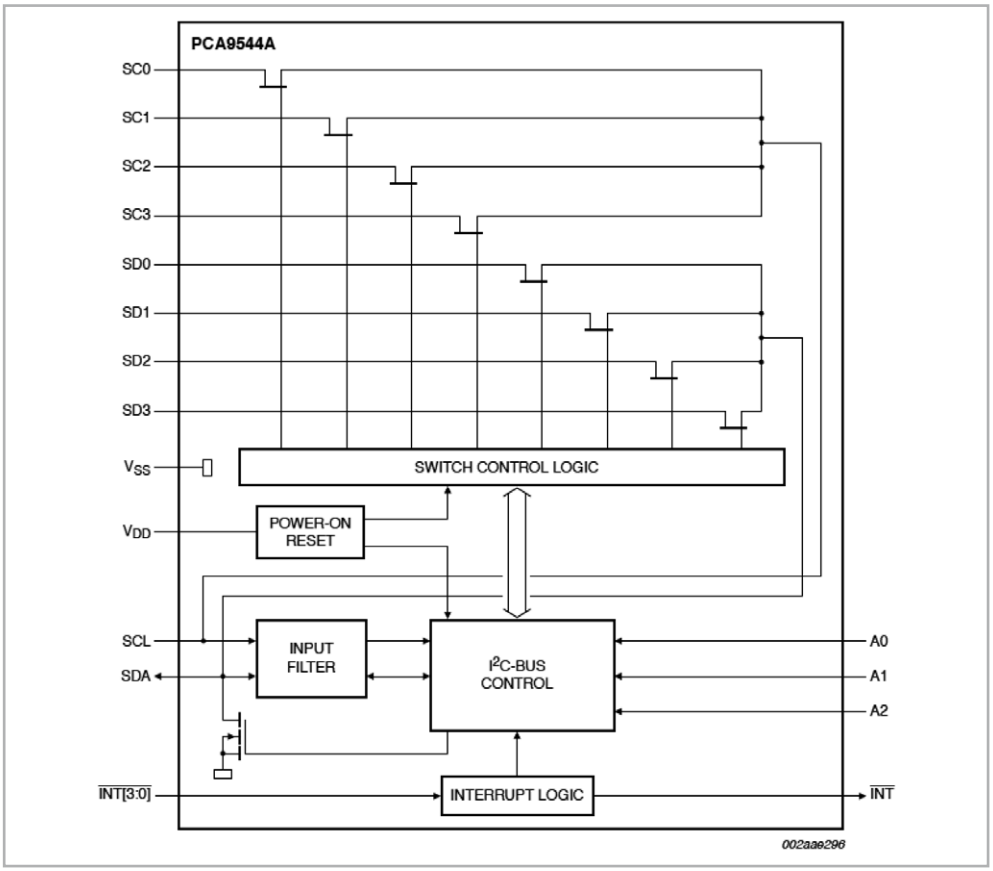


*Bild 13: Das Blockschaltbild des I²C-Multiplexers PCA9544A. Bild: NXP*

Durch eine Hardware-Adressierung des Bus-Switches lassen sich auch mehrere dieser Bausteine gleichzeitig an einem Bus betreiben, sodass man auch große Systeme aufbauen kann.

Ähnlich arbeiten I²C-Multiplexer wie der PCA9544A (Bild 13), der zudem über eine bidirektionale Interruptsteuerung verfügt, die auch von einem Slave wie dem 7-Kanal-Temperatursensor MAX6697 (Bild 14) aus den Master ansprechen kann, und der einen Anschluss von Systemen mit unterschiedlichen Buspegeln erlaubt.

**Zu wenige µC-Ports?**

Das Problem, immer einen freien Port am Mikrocontroller zu wenig zu haben, als man benötigt, tritt besonders bei den kleineren Mikrocontrollern, etwa den ATtinys oder dem ESP-12, immer wieder einmal auf. Muss man beim Controllertyp bleiben, hilft I²C auch hier, und zwar mit einem Portexpander, wie er in Bild 15 mit dem PCF8574 zu sehen ist. Auch hier kann man durch Hardware-Adressierung mehrere dieser Expander kaskadieren.

Die Ports P0 bis P7 sind bidirektional ausgeführt. Als Ausgänge arbeiten sie als Open-Collector-Ausgang, der über einen Schreibzugriff aktiv wird. Sendet man einen Lesezugriff an den PCF8547, so werden die OC-Ausgänge deaktiviert und man kann den dann am Port anstehenden Pegel, etwa eines Logikbausteins oder Tasters, einlesen. In [1] findet man dazu ein ausführliches Programmierbeispiel.

**I²C am ESP und am RP2040 Pico**

Neben dem Arduino sind die Espressif-ESP-Boards weit verbreitet. Bereits mit dem ESP8266EX auf dem ESP-01-Modul oder dem bekannten WEMOS-Modul kann man trotz der wenigen GPIOs eine Software-Schnittstelle zuordnen. Das Datenblatt gibt darüber

Auskunft, in [3] wird die Anbindung praktisch beschrieben.

Weit vielfältiger ist die Verfügbarkeit bei der ESP32-Familie. Neben den Standard-Pins G21 (SDA) und G22 (SCL) kann man weitere GPIOs individuell als zweiten I²C-Port definieren. Dazu sind die entsprechenden Pins inklusive Pull-ups im Programm ebenso zuzuordnen wie, je nach Bibliothek der anzuschließenden I²C-Komponenten, die beiden I²C-Schnittstellen durch Bildung von sogenannten TwoWire-Objekten für I2C_1 und I2C_2. In [4] und [5] sind die Vorgehensweisen detailliert und mit konkreten Programmbeispielen erläutert. Hier muss man jedoch genau hinsehen, welches ESP32-Modul man benutzt, da in den verschiedenen Modulen unterschiedliche GPIOs vorbelegt, nur als Eingang verwendbar bzw. nicht für I²C verfügbar sind. Hierüber gibt [6] detailliert Auskunft.

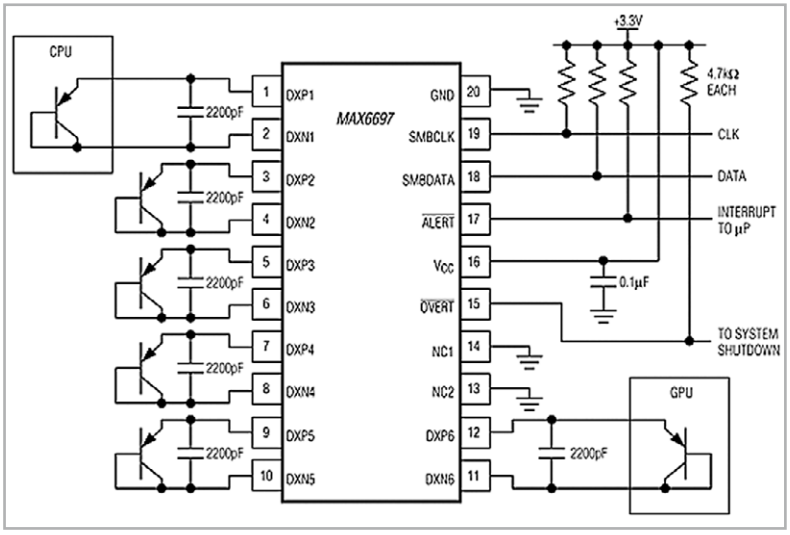


*Bild 14: Manche I²C-Bausteine verfügen über einen Interruptausgang, der z. B. das Erreichen von Grenzwerten an den Master signalisiert, hier der MAX6697. Bild: MAXIM Integrated*

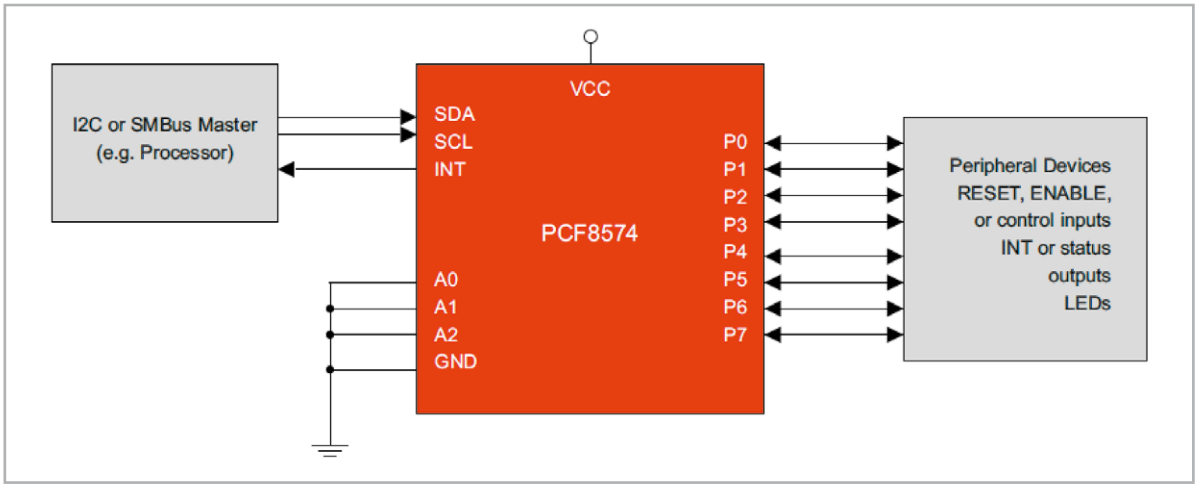


*Bild 15: Mit einem Portexpander wie dem PCF8574 kann man bis zu acht Peripheriebauteile, Anzeigen, Taster, Logikbausteine etc. über den I²C-Bus ankoppeln. Bild: NXP*

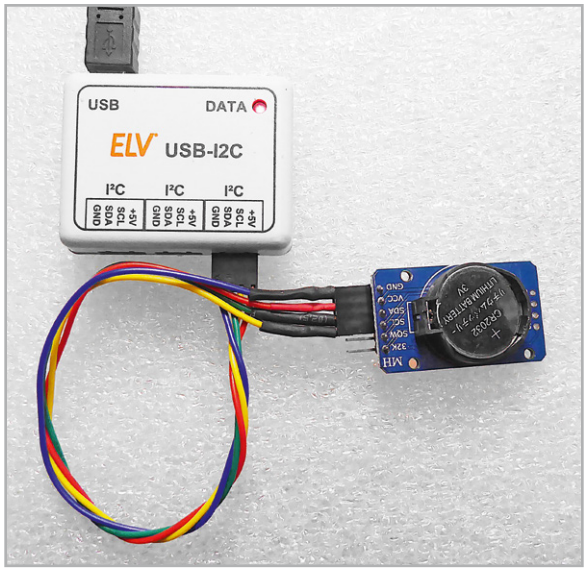


*Bild 16: Das ELV USB-I²C-Interface ist ein universell einsetzbares Werkzeug für die Entwicklung, das Testen und Experimentieren. Hier ist eine DS3231-RTC angeschlossen.*

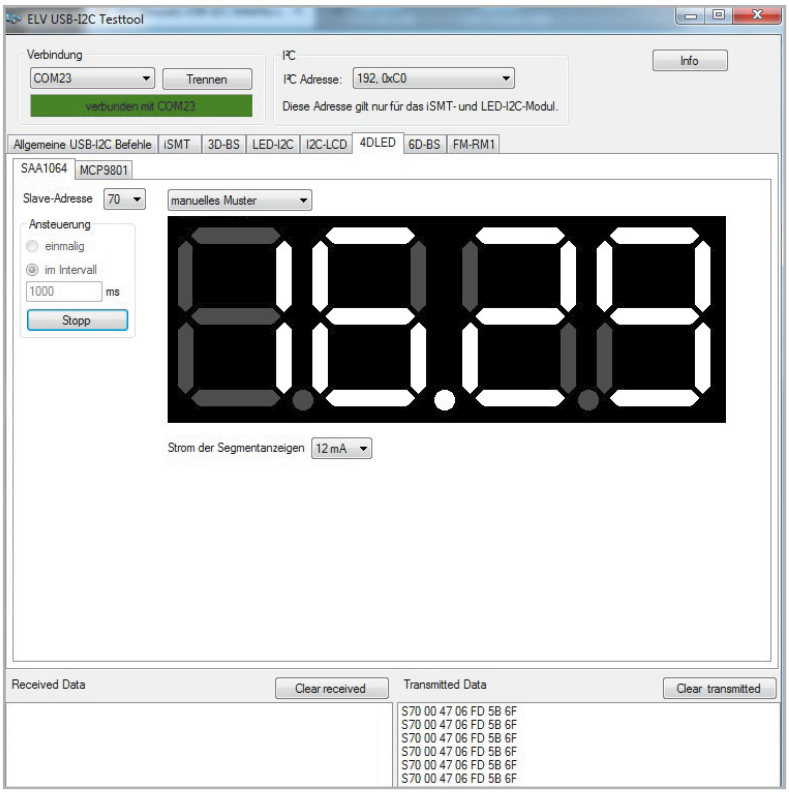


*Bild 17: Das per Download auf der Produktseite verfügbare Demoprogramm zum ELV USB-I²C-Interface ermöglicht sowohl das Programmieren der ELV I²C-Baugruppen ...*

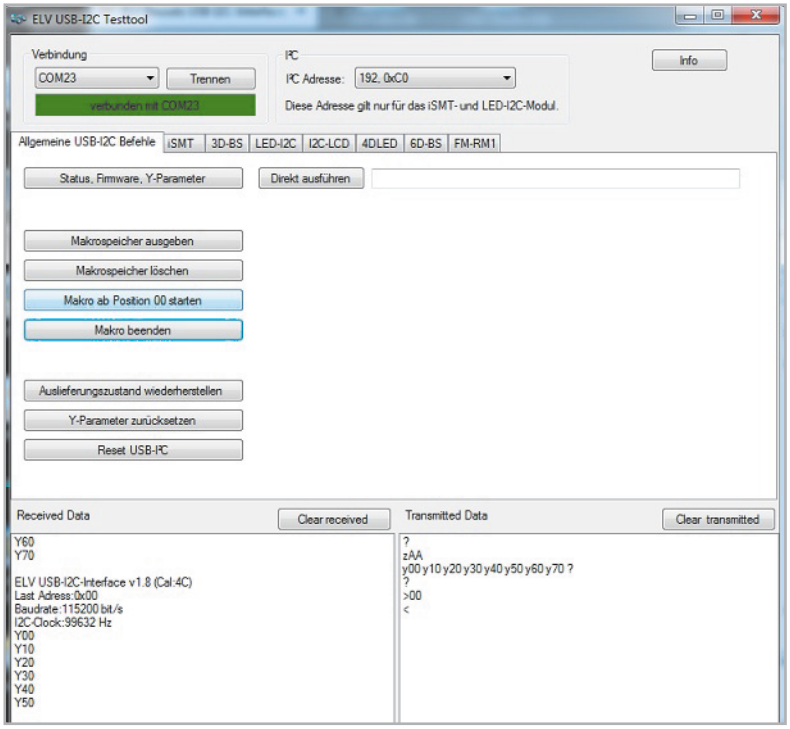


*Bild 18: ... als auch die Kommunikation über allgemeine I²C-Kommandos mit anderen Bausteinen.*

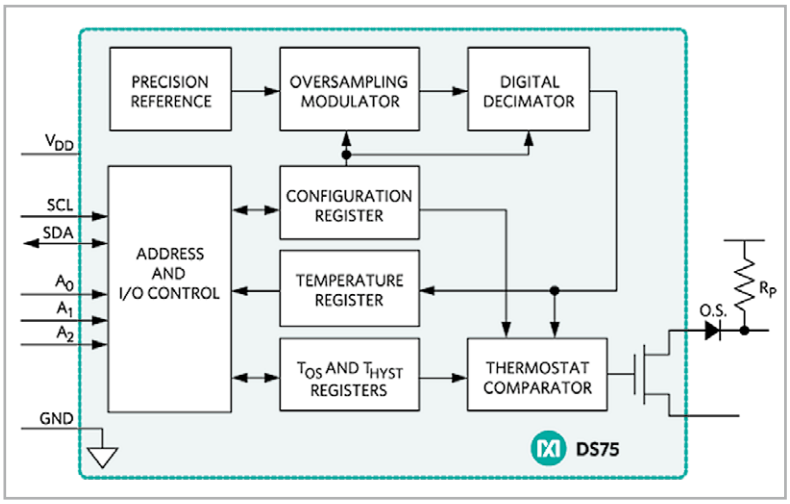


*Bild 19: Der DS75 lässt sich als Thermostat programmieren. Das Vorgehen dazu ist im Datenblatt ausführlich beschrieben. Bild: Maxim Integrated*

Auch der Controller RP2040 des noch relativ jungen Raspberry Pi Pico und seine inzwischen zahlreichen Derivate verfügen über zwei I²C-Controller I²C0 und I²C1. Diese können, wie auch beim ESP32, weitgehend frei den Multifunktions-GPIOs des Raspberry Pi Pico zugeordnet werden, die Standard-Pins sind die Pins 4 (GP2/SDA) und 5 (GP3/SCL).

In der SDK-Dokumentation zum RP2040 sind die Kommandos zur Steuerung der Schnittstelle detailliert beschrieben [7].

## Universalwerkzeug USB-I²C-Interface

ELV bietet mit dem USB-I²C-Interface [1] ein universelles Werkzeug an, das man sowohl zur Nutzung von I²C-Bausteinen an einem PC als auch zum Lernen und Trainieren von Adressierungen und Datentransfers, zum Testen bei der Entwicklung eigener Programme und auch als I²C-Datenlogger einsetzen kann (Bild 16). So kann man neben einer Reihe von I²C-Baugruppen von ELV (siehe Verfügbare I²C-Bausteine bei ELV) – Bild 17 zeigt die Einstellungen für die 4DLED-Baugruppe – auch allgemeine Baugruppen bzw. Chips abfragen und programmieren (Bild 18).

Auch das Einlesen von Analogwerten ist hier möglich, ebenso das Einspeichern von Befehlsfolgen als Makro und dessen Übertragung auf den I²C-Bus für bis zu 128 Geräte. So kann man u. a. auch AD-Wandler abfragen oder Bausteine wie den DS75 von Maxim (Bild 19), der nicht nur Temperaturen misst, sondern auch in einem nichtflüchtigen Register eine obere und untere Temperaturgrenze speichern kann, in deren Bereich er als Thermostat arbeiten und ein Schaltsignal ausgeben kann.

Das USB-I²C-Interface ist neben dem hier gezeigten, zum Download bereitstehenden Demo-Programm auch für die direkte Programmierung von Befehlen über ein Terminalprogramm oder als Datenlogger mit tabellarischer/grafischer Aufzeichnung, z. B. mit LogView, einsetzbar.

Das Interface sichert zusätzlich auch die Spannungsversorgung der angeschlossenen I²C-Bausteine mit insgesamt bis zu 450 mA. ELV

## i Weitere Infos

[1] Handbuch ELV USB-I²C-Interface: im Downloadbereich bei der Artikel-Nr. 084123 im ELVshop

[2] I²C-Adress-Scanner von Nick Gammon im Arduino.cc-Playground: https://playground.arduino.cc/Main/I2cScanner/

[3] ESP8266-I²C -Kommunikation: https://randomnerdtutorials.com/esp8266-0-96-inch-oled-display-with-arduino-ide/

[4] ESP32 I²C-Kommunikation: https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/

[5] ESP-Programming-Guide: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2c.html

[6] ESP-Hardware-Guide: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html

[7] RP2040-Dokumentation – I2C Controller API und Beispiele: https://raspberrypi.github.io/pico-sdk-doxygen/group__hardware__i2c.html#details

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links