Elektronische Messdatenverarbeitung

Klaus Betzler *

Universität Osnabrück

Wintersemester 2007/2008

Inhaltsverzeichnis

* KLAUS.BETZLER@PHYSIK.UNI-OSNABRUECK.DE 20. NOVEMBER 2007

1 Detektoren und Sensoren

Experimentelle Messgrößen liegen im Regelfall nicht in 'EDV-kompatibler' Form vor. Diese herzustellen, d. h. die physikalische Größe in eine geeignete elektrische umzuwandeln – geeignet letztlich zur Weiterverarbeitung mit einem *Interface* und einem Computer – ist die Aufgabe von Detektoren und Sensoren. Die Abgrenzung zwischen den beiden Begriffen ist nicht immer eindeutig; will man abgrenzen, so kann man den Begriff Detektor für den Nachweis von Teilchen (Elektronen, Photonen), den Begriff Sensor für die Wandlung anderer physikalischer Größen (Temperatur, Lichtintensität) benutzen.

Bei der Anwendung eines Detektors oder Sensors sollte man sich in jedem Fall zunächst die zugrunde liegende physikalische Wirkungsweise klar machen, daher:

> Behandeln sie einen Detektor oder Sensor erst dann als *black box*, wenn sie wirklich wissen, was drin ist.
>
> Wichtig sind die Fähigkeiten *und* die Grenzen eines Systems.

Das gesamte Gebiet ist sicherlich zu groß, um es in einer Vorlesung auch nur annähernd vollständig behandeln zu können, wir beschränken uns daher auf einige Beispiele aus typischen Bereichen.

1.1 Temperatur

Praktisch alle physikalischen Eigenschaften sind mehr oder weniger stark temperaturabhängig, können also prinzipiell zur Temperaturmessung eingesetzt werden; anschaulichstes Beispiel ist die thermische Ausdehnung bei Festkörpern, Flüssigkeiten oder Gasen. Für die Anwendung in Sensoren besonders interessant und fast ausschließlich verwendet sind temperaturabhängige Änderungen der *elektrischen* Eigenschaften, die auf einfache Weise automatisch detektiert und elektrisch weiterverarbeitet werden können.

1.1.1 Widerstände

Speziell zur Temperaturmessung und -regelung entwickelte PTC- und NTC-Thermistoren, d. h. Widerstände mit positivem oder negativem Temperaturkoeffizienten (Kaltleiter, Heißleiter), fanden und finden in der technischen Elektrik breite Verwendung. Ihr Anwendungsbereich ist allerdings meist auf Temperaturen beschränkt, in denen technische Geräte, Haushaltsgeräte etc. arbeiten.

In einem weiteren Temperaturbereich einsetzbar und daher für physikalische Experimente interessanter (aber auch teurer) sind Platin-Widerstände mit besonderen Spezifikationen (Pt 100), die eine sehr ausgeprägte, gut definierte und dokumentierte Temperaturabhängigkeit zwischen etwa 10 K und 1000 K aufweisen.

Für tiefe Temperaturen (1 K bis 100 K) geeignet sind Kohle-Widerstände und spezielle Halbleiter-Widerstände (Ge) oder -Dioden (Si), die man auch mit genauer, individuell erstellter Eichung (dann sehr teuer) kaufen kann.

Bei der Temperaturmessung mit Widerständen ist – insbesondere bei tiefen Temperaturen – darauf zu achten, dass die zur Messung benötigte Leistung und damit die Wärmezufuhr möglichst gering ist.

1.1.2 Thermoelemente

Thermoelemente nutzen die materialspezifische Temperaturabhängigkeit der Ladungsträgerverteilung in Metallen oder Halbleitern aus (detaillierte Beschreibung in Lehrbüchern zur Festkörperphysik [1] oder Halbleiterphysik [2]). Obwohl der thermoelektrische Effekt (Thermokraft) im allgemeinen in Halbleitern größer ist, werden aus naheliegenden praktischen Gründen geeignete Kombinationen aus verschiedenen Metallen oder Metall-Legierungen verwendet. Einige gebräuchliche *Thermopaare* sind in Tabelle 1 zusammengestellt.

Thermopaar	Temperaturbereich	Diff. Thermospannung
Gold Eisen – Nickel Chrom	-270 °C. . . 0 °C	20 μV/K
Kupfer – Konstantan	-200 °C. . . 600 °C	40 μV/K
Eisen – Konstantan	-200 °C. . . 900 °C	52 μV/K
Nickel Chrom – Konstantan	0 °C. . . 1000 °C	63 μV/K
Nickel Chrom – Nickel	-200 °C. . . 1370 °C	40 μV/K
Platin Rhodium – Platin	0 °C. . . 1750 °C	55 μV/K
Wolfram Rhenium 5 – 26	0 °C. . . 2500 °C	10 μV/K

Tabelle 1: Typischer Anwendungstemperaturbereich und differentielle Thermospannung bei 0 °C gebräuchlicher Thermoelemente.

Die Vorteile von Thermoelementen liegen in ihrem großen Anwendungstemperaturbereich, ihrer relativ einfachen Handhabung und dem günstigen Preis. Nachteilig ist das geringe elektrische Signal, das ein empfindliches Messgerät bzw. eine stabile Verstärkung erfordert, und die Notwendigkeit einer Referenzstelle (Eisbad) mit definierter, konstanter Temperatur (andere Möglichkeit s. 1.1.3).

Bei der Herstellung von Thermoelementen ist insbesondere auf guten elektrischen Kontakt zwischen den beiden Materialien zu achten (Punkt- oder Mikroschweißung). Teure Thermoelemente können mit *Ausgleichsleitungen* angeschlossen werden. Genauer ist dies in den Datenblättern der Hersteller beschrieben, die meist auch ausführliche Tabellen der temperaturabhängigen Thermospannungen enthalten.

1.1.3 Band-Gap-Referenz-Diode

Die Strom-Spannungs-Kennlinie einer Halbleiterdiode wird in guter Näherung durch

$$I = I_S(\exp \frac{eU}{kT} - 1) \tag{1.1}$$

beschrieben. Dies wurde schon in der Frühzeit der Halbleiterphysik von Shockley hergeleitet und experimentell verifiziert [3].

Abbildung 1: Strom-Spannungs-Kennlinien einer Diode nach Gleichung 1.1 für zwei verschiedene Temperaturen (10 °C und 50 °C); der Strom im Sperrbereich ist um 10^4 überhöht dargestellt.

Im Durchlassbereich ($U \gg kT/e$) kann man die -1 in Gl. 1.1 gegen die Exponentialfunktion vernachlässigen, bei konstantem Strom I ist dann der Zusammenhang zwischen Spannung U und Temperatur T annähernd linear (die Temperaturabhängigkeit des Sättigungssperrstroms I_S ist bei dieser Betrachtung nicht berücksichtigt). Basierend auf diesem Effekt lassen sich Temperaturfühler mit sehr gut linearer Kennlinie bauen [4]. Ein typisches Beispiel ist der integrierte Schaltkreis AD 592 von Analog Devices.

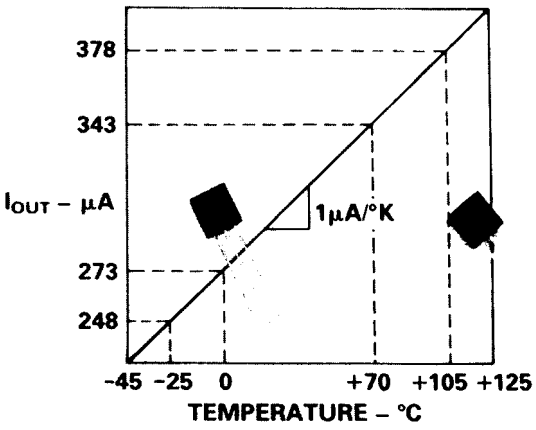


Abbildung 2: Idealisierte Kennlinie und Bauform des IC AD 592; aus [5].

Der AD 592 liefert in einem Betriebsspannungsbereich 3 V. . . 30 V einen sehr gut temperaturproportionalen Strom, der Proportionalitätsfaktor ist genau 1 μA/K (Abb. 2), dies im Temperaturbereich -25 °C. . . 105 °C. An einem Lastwiderstand von beispielsweise 10 kΩ ergibt dies eine gut messbare Spannung von einigen Volt. Die Nichtlinearität der Kennlinie liegt – auf die Temperatur umgerechnet – in der Größenordnung von $\pm$0.5 K. Die Bauform (Plastikgehäuse) bedingt allerdings eine relativ große thermische Zeitkonstante und schlechte Wärmeabfuhr, die diesbezüglichen Daten sind in Tabelle 2 zusammengestellt.

Bevorzugte Anwendungsgebiete sind mithin solche mit langsam veränderlicher

Art der Kühlung	Wärmewiderstand	Therm. Zeitkonstante
Ruhende Luft	175 K/Watt	60 sec
Ruhende Luft + Kühlblech	130 K/Watt	55 sec
Bewegte Luft	60 K/Watt	12 sec
Bewegte Luft + Kühlblech	40 K/Watt	10 sec
Flüssigkeit	35 K/Watt	5 sec
Aluminiumblock + Wärmeleitpaste	30 K/Watt	3 sec

Tabelle 2: Wärmewiderstand und thermische Zeitkonstante des AD 592 für verschiedene Anwendungsarten. Die Erwärmung durch die im Schaltkreis umgesetzte Leistung kann demnach im ungünstigsten Fall bis zu 2 K betragen.

Temperatur, beispielsweise die Temperaturregelung von Halbleiterlasern oder die Messung der Referenztemperatur bei Thermoelementen. Einen Vorschlag zur Referenzstellenkompensation zeigt Abbildung 3.

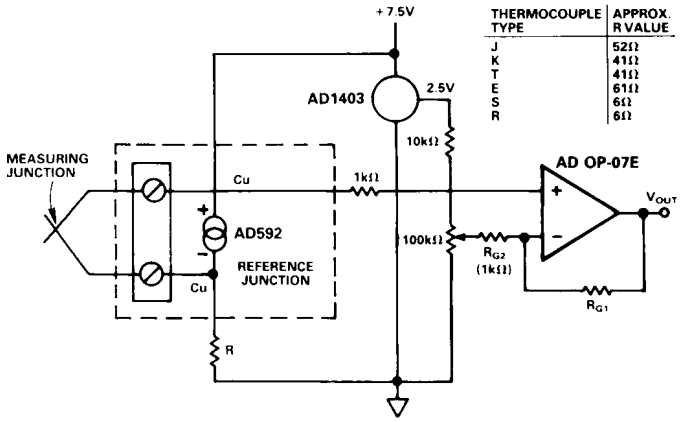


Abbildung 3: Referenzstellenkompensation mit dem IC AD 592 bei der Temperaturmessung mit einem Thermoelement (Schaltungsvorschlag aus dem Datenblatt des IC-Herstellers Analog Devices); aus [5].

1.1.4 Kapazitive Sensoren

Die bisher beschriebenen Temperatursensoren werden ungenau, wenn große Magnetfelder am Messort vorhanden sind. Für diesen Spezialfall können kapazitive Sensoren verwendet werden, die als physikalisches Messprinzip die Abhängigkeit der Dielektrizitätskonstanten von der Temperatur ausnutzen (die Kapazitätsmessung ist allerdings deutlich aufwendiger als etwa eine Widerstandsmessung). Besonders geeignet dafür sind Materialien, die in der Nähe der zu messenden Temperaturen einem strukturellen Phasenübergang (paraelektrisch → ferroelektrisch) zustreben. Ein Beispiel ist Strontiumtitanat bei sehr tiefen Temperaturen, die Dielektrizitätskonstante nimmt gegen 0 K deutlich zu, ohne dass ein Phasenübergang tatsächlich erreicht wird.

1.2 Koordinaten (Ort und Winkel)

Die klassischen Sensoren für diesen Bereich sind Potentiometer, lineare oder Drehwiderstände, die – direkt oder über mehr oder weniger aufwendige Getriebe mit der Messstelle verbunden, mit konstantem Strom oder konstanter Spannung betrieben – eine orts- oder winkelabhängige Spannung liefern. Soll's genauer sein, kann man bei kleinen Wegen kapazitive oder piezoelektrische Prinzipien verwenden, bei größeren optische Phasenmessungen (Interferometer), bei sehr großen optische oder elektrische Laufzeitmessungen.

Im Werkzeugmaschinenbereich werden derzeit hauptsächlich exakte mechanische Maßstäbe oder Teilscheiben verwendet, die optoelektronisch abgelesen werden. Auf Spezialglassubstrate werden hochgenaue Teilungen oder Kodierungen aufgedampft, Beispiele für Teilscheiben zur inkrementellen oder absoluten Winkelmessung zeigt Abbildung 4.

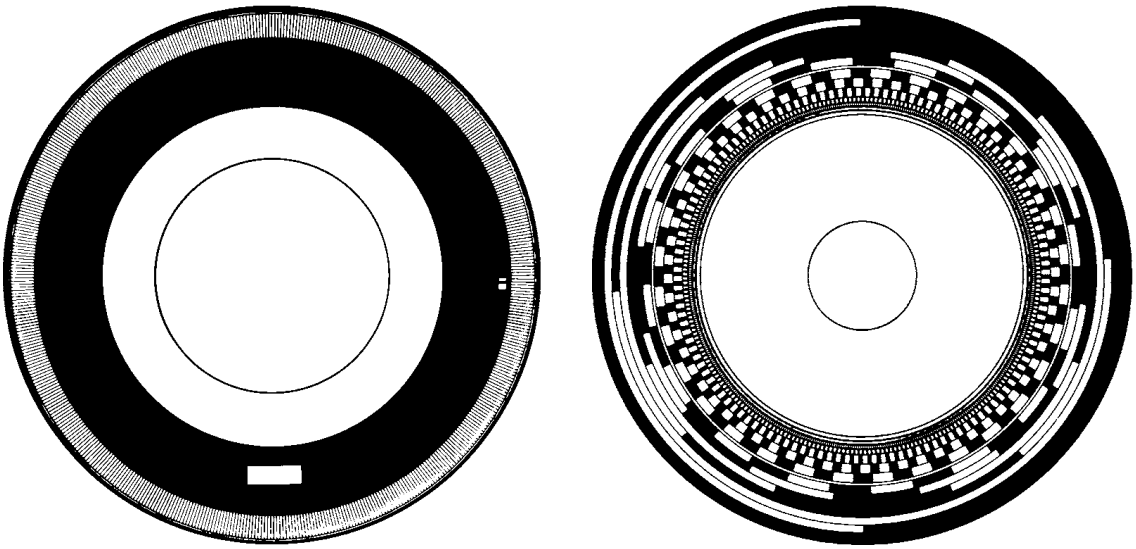


Abbildung 4: Teilscheibe eines inkrementellen Drehgebers (links) und eines Code-Drehgebers (rechts), beide etwa in natürlicher Größe (entnommen einem Katalog der Firma Heidenhain, Traunreut). Die Code-Teilscheibe rechts ist im Gray-Code kodiert, einem binären Code, bei dem sich von einem Wert zum nächsten immer nur ein Bit der Kodierung verändert; auf diese Weise können keine Ablesefehler auftreten, wenn die Scheibe zwischen zwei Werten steht.

Die mit solchen Teilscheiben aufgebauten inkrementellen Drehgeber liefern in der Regel zwei um $\pi/2$ gegeneinander phasenverschobene Sinusspannungen, die in der zugehörigen Anzeigeelektronik ausgewertet werden. Die aktuelle Winkelposition kann über eine Standardschnittstelle (Seriell oder parallel) in einen angeschlossenen Rechner übernommen werden. Benötigt man die Messpunkte in schneller Abfolge, bietet es sich an, die Sinussignale über Analog-Digital-Wandler direkt vom Rechner zu erfassen und auszuwerten. Die mechanische Genauigkeit der Teilscheiben und der optoelektronischen Ablesung ist so gut,

dass pro Sinusperiode mehrere hundert Subschritte interpoliert werden können. Damit kann eine Genauigkeit der Winkelmessung erzielt werden, die in der Größenordnung von 1/1000 Grad liegt.

1.3 Licht

Zum Nachweis von Licht können unterschiedliche physikalische Effekte ausgenutzt werden. Einerseits sind dies Quanteneffekte (äußerer oder innerer Photoeffekt), mit denen sehr empfindliche Detektoren realisiert werden können, andererseits Effekte, die in irgendeiner Form die Temperaturänderung nachweisen, die durch den mit Licht verbundenen makroskopischen Energiestrom bewirkt wird (eine ausführliche Beschreibung der bei Photodetektoren genutzten physikalischen Effekte und der damit realisierten Bauelemente gibt beispielsweise [6]). Wegen der wesentlich besseren Empfindlichkeit sind für physikalische Experimente in erster Linie die auf den Photoeffekten basierenden Detektoren interessant. Thermische Detektoren werden generell nur dort eingesetzt, wo entweder sehr hohe Lichtleistungen zu messen sind oder wo in Wellenlängenbereichen gemessen werden muss, die für andere Detektoren nicht zugänglich sind.

1.3.1 Zur Charakterisierung von Photodetektoren

Innerer und äußerer Photoeffekt sind als 'Bandstruktureffekte' sehr stark material- und energieabhängig. Ladungsträger können – sieht man von in der Regel vernachlässigbaren Mehrquanteneffekten ab – nur von Photonen ab einer bestimmten Quantenenergie angeregt werden. Die Nachweisempfindlichkeit ist auch oberhalb dieser Grenzenergie stark wellenlängenabhängig. Bei thermische Detektoren ist im Vergleich dazu der Empfindlichkeitsverlauf wesentlich weniger dramatisch. Zur Charakterisierung von Detektoren wird die Empfindlichkeit mit ihrer Wellenlängenabhängigkeit angegeben – als *Responsivity* $R(\lambda)$, gemessen beispielsweise in Ampere/Watt, bei Photodioden oder als Quantenausbeute, gemessen in %, bei Kathoden von Photomultipliern.

***NEP*-Wert:** Bei geringen Lichtsignalen stellt das Detektorrauschen eine ganz wesentliche Begrenzung dar, es ist daher üblich, dieses Rauschen bei der Empfindlichkeitsangabe implizit zu berücksichtigen. Als Maß für die so definierte Empfindlichkeit wird die Lichtleistung angegeben, die notwendig ist, um ein dem Rauschen äquivalentes Ausgangssignal zu generieren (*NEP*-Wert – *N*oise *E*quivalent *P*ower).

Detektivität: Detektoren sind umso besser, je kleiner ihr *NEP*-Wert ist. Da große Werte jedoch allgemein beliebter und vor allem werbewirksamer sind, wird häufig die dazu reziproke Größe, die Detektivität $D = 1/NEP$ verwendet.

Spezifische Detektivität: Das Detektorrauschen hängt nicht nur von der Detektorart ab, sondern auch von der Detektorgröße sowie von den Messbedingungen (Detektortemperatur, Verstärkerbandbreite). Weiterhin kann der Erfas-

sungswinkel von Detektoren unterschiedlich sein. Als statistische Größe ist das Rauschsignal in guter Näherung proportional zur Quadratwurzel aus der Detektorfläche A und der Bandbreite B. Man objektiviert daher üblicherweise von den genannten Nebenbedingungen durch Angabe der spezifischen Detektivität D^* (*D-Stern*):

$$D^* = (\mathit{NEP})^{-1} \cdot A^{1/2} \cdot B^{1/2} \,. \tag{1.2}$$

Einige Autoren bzw. Firmen definieren zusätzlich ein *D-Doppelstern*

$$D^{**} = D^* \sin\Theta \,, \tag{1.3}$$

das zusätzlich den Erfassungswinkel (Θ = halber Erfassungswinkel) berücksichtigt.

Einen Überblick der Detektivitätskurven verschiedener Detektortypen geben die Abbildungen 5 und 6, Abbildung 5 für Anwendungen im sichtbaren Spektralbereich sowie im nahen Ultraviolett und Infrarot (Photomultiplier und -dioden), Abbildung 6 für den Infrarotbereich zwischen 1 und 1000 μm (Photoleiter und thermische Detektoren).

Rauschen und Energiebereich: Während bei ausreichenden Lichtintensitäten das Detektorrauschen keine große Rolle spielt, ist es bei der Messung geringer Intensitäten die begrenzende Größe für die minimal messbare Intensität. Bei Photonendetektoren wird das Rauschen bei geringen Lichtintensitäten (Dunkelrauschen) im wesentlichen durch die thermische Anregung von Ladungsträgern verursacht. Die Aktivierungsenergie für diese thermische Anregung ist generell kleiner als die für optische, daher ist das thermisch generierte Detektorrauschen umso größer, je geringer die optische Anregungsenergie ist, d. h. je langwelliger ein Detektor nutzbar ist. Durch Kühlung des Detektors lassen sich die Verhältnisse zwar etwas verbessern, man wird im allgemeinen jedoch immer einen für die jeweilige Messaufgabe optimierten Kompromiss schließen müssen. Für geringe Intensitäten sieht der oft so aus, dass man einen Detektor verwendet, mit dem der gewünschte Wellenlängenbereich gerade noch abzudecken ist.

1.3.2 Photomultiplier

Zum Nachweis extrem geringer Lichtintensitäten im sichtbaren, ultravioletten und nahinfraroten Spektralbereich verwendet man meist Photomultiplier. Ihre Funktionsweise beruht auf dem äußeren Photoeffekt und der Sekundärelektronenemission: aus einer Photokathode (im Vakuum) werden durch Lichtquanten Elektronen ausgelöst, die durch eine Spannung von etwa 100 V zur nächsten Elektrode (Dynode) beschleunigt werden, wo jedes mehrere Sekundärelektronen auslöst. Solche Verstärkungsstufen werden kaskadiert (ca. 10), so dass an der letzten Elektrode (Anode) ein gut messbares Ladungssignal entsteht (Sekundärelektronen-Vervielfacher). Mit dafür optimierten Anordnungen können so problemlos einzelne Photonen nachgewiesen werden (*photon counting*). Den prinzipiellen Aufbau eines Photomultipliers zeigt Abbildung 7.

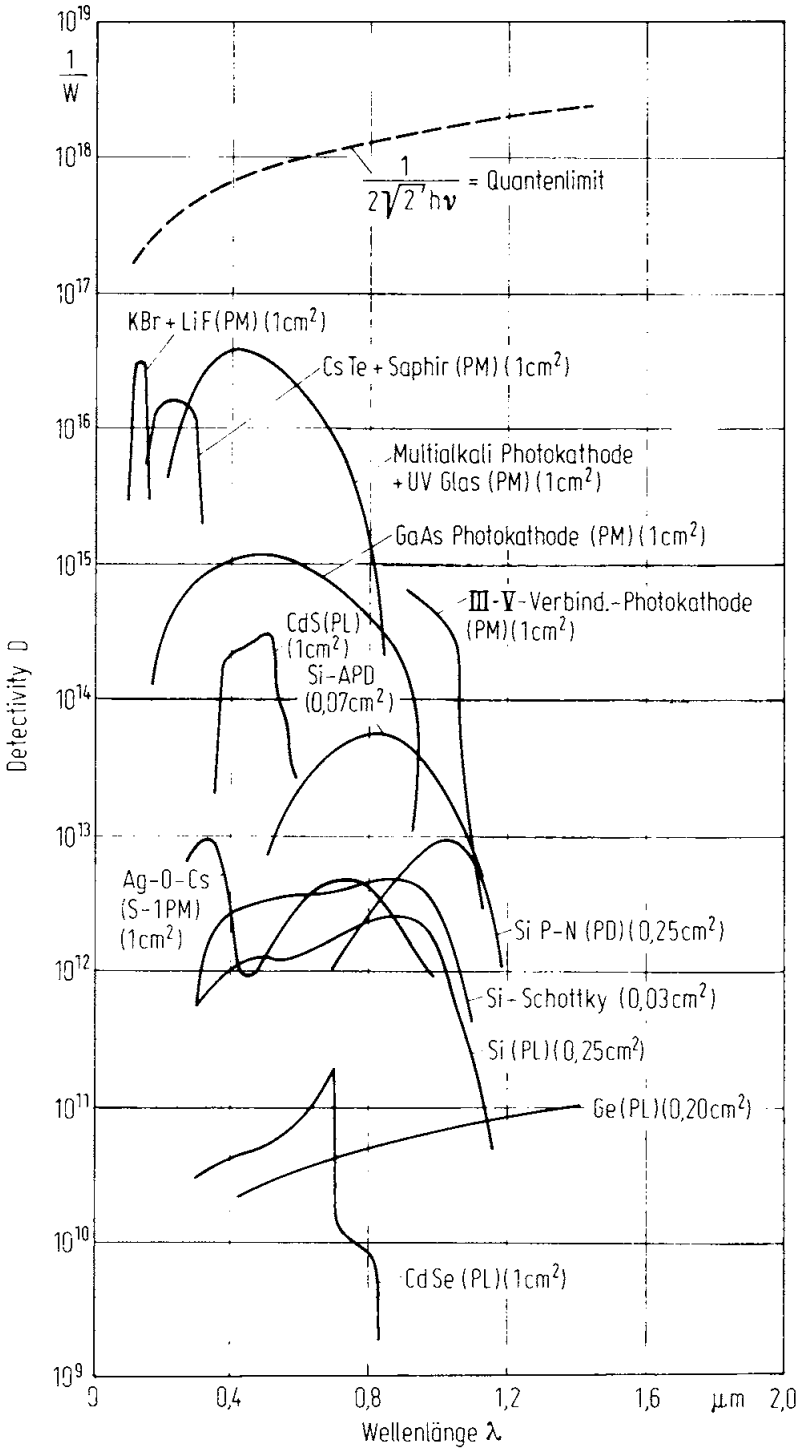


Abbildung 5: Detektivität D von Photomultipliern (PM), Photoleitern (PL), Photodioden (PD) und Lawinenphotodioden (APD); aus [6].

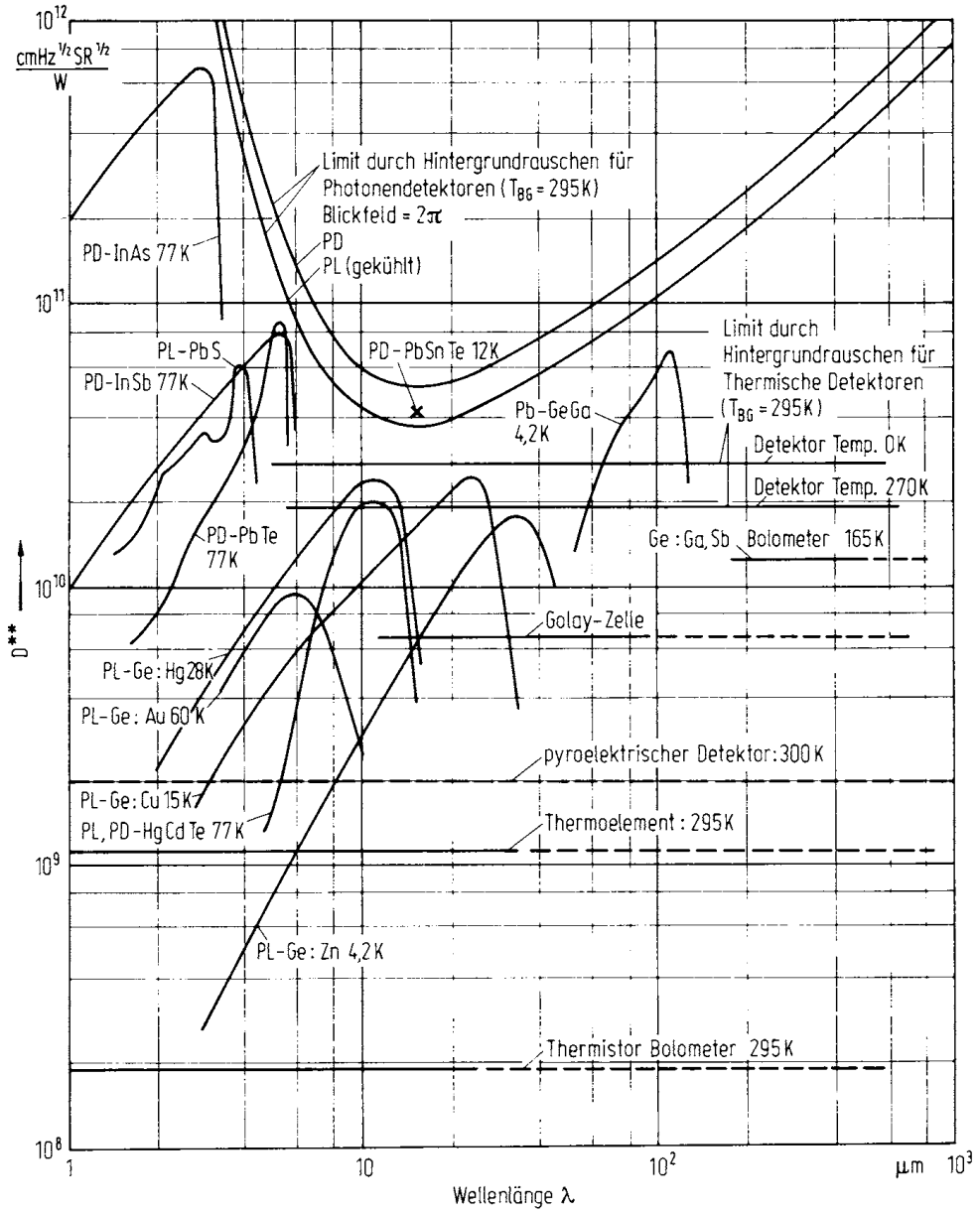


Abbildung 6: Spezifische Detektivität D^{**} von thermischen Detektoren und Photonendetektoren; aus [6].

Kathodenmaterialien: Für Photokathoden werden Materialien aus drei Substanzklassen verwendet: Metalle mit niedriger Austrittsarbeit (meist Mischungen aus Alkalimetallen) für den sichtbaren Spektralbereich, Halbleiter (Telluride, Oxide) für das Ultraviolette und III-V-Halbleiter (GaAs, GaInAs), bei denen mit geeigneter Beschichtung eine *negative Elektronenaffinität* erreicht wurde, für das nahe Infrarot. Die typischen Bandstrukturen (Energienivaues im Ortsraum) sind in Abbildung 8 skizziert.

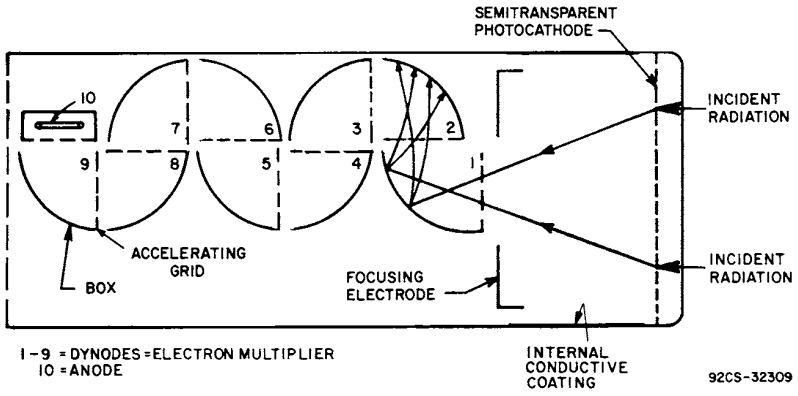


Abbildung 7: Typische Geometrie eines Photomultipliers mit Frontfensterkathode (gebräuchliche Durchmesser liegen zwischen 10 und 100 mm); aus [7].

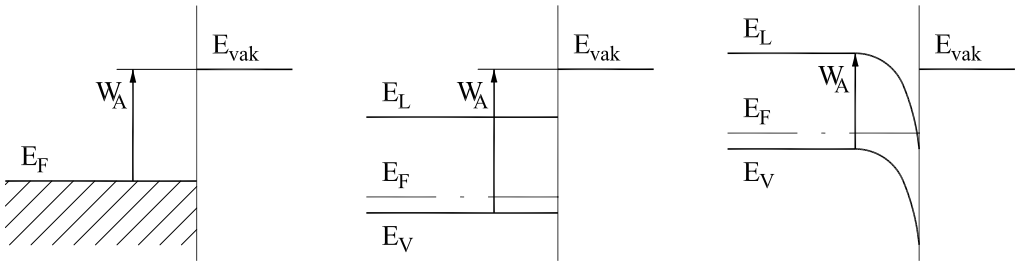


Abbildung 8: 'Optische' Elektronenaustrittsarbeit W_A bei Metallen (linkes Bild), Halbleitern (mittleres Bild) und Halbleitern mit negativer Elektronenaffinität, *d. h. $E_\mathrm{vak} < E_\mathrm{L}$ (rechtes Bild). E_F: Fermi-Energie, E_V: Valenz-, E_L: Leitungsband, E_vak: Vakuumniveau.*

Die Kathoden sind entweder als dünne semitransparente Schicht innen auf das Frontfenster aufgedampft oder als massivere Beschichtung auf ein Metallblech aufgetragen. Die Kathodengrößen liegen für optische Anwendungen zwischen einigen Millimetern (rauscharme Photonenzählanwendungen) und einigen Zentimetern.

Empfindlichkeit: Das Maximum der Quantenausbeute bei Photomultipliern liegt je nach Kathodenmaterial zwischen 0.1 und 30 %, d. h. jedes tausendste bzw. dritte auf die Photokathode treffende Photon löst dort ein Elektron aus. Der spektrale Verlauf ist im langwelligen Bereich durch die Austrittsarbeit bestimmt, im kurzwelligen Bereich in der Regel durch das Fenstermaterial[1]. Eine Übersicht über die spektralen Empfindlichkeitskurven verschiedener Kathodenmaterialien[2] zeigen die Abbildungen 9, 10 und 11 (aus [8]).

[1] Empfindlichkeit im ferneren UV kann auch durch Szintillatormaterialien erreicht werden, die kurzwelliges Licht absorbieren und längerwellig lumineszieren.

[2] Die teilweise verwendeten Bezeichnungen S4, S11 usw. sind gebräuchliche Trivialnamen für bestimmte Materialmischungen.

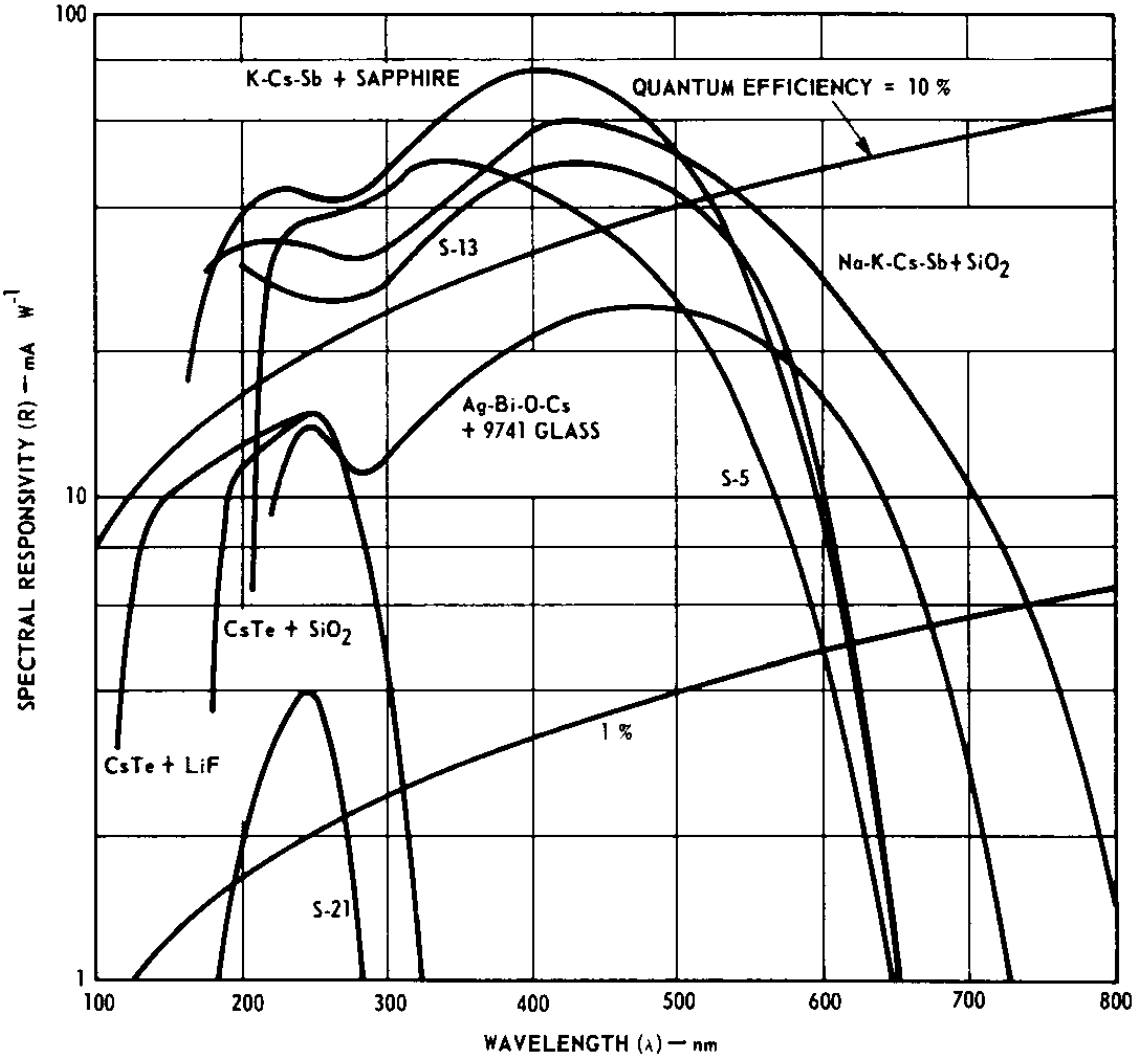


Abbildung 9: Empfindlichkeitskurven von Photomultipliern für Ultraviolettanwendungen.

Dynoden werden aus Materialien mit guter Sekundärelektroneneffizienz hergestellt, vorwiegend aus BeO (gute Hochtemperatureigenschaften) oder Cs_3Sb.

Zeitverhalten: Die *Gesamtlaufzeit* der Elektronen von der Kathode zur Anode beträgt je nach Bauform zwischen 10 und 100 nsec. Durch eine Optimierung der Dynodenanordnung kann man erreichen, dass die *Laufzeitstreuung*, damit die Verbreiterung eines Pulses, deutlich unter diesen Zeiten bleibt. Erreichbar sind Werte in der Größenordnung einer Nanosekunde. Dies begrenzt grundsätzlich die mit Photomultipliern erreichbare Zeitauflösung, aber auch die für Photonenzählanwendungen maximal mögliche Zählrate.

Verstärkung: Typische Verstärkungen der verwendeten Dynodenanordnungen liegen zwischen 10^4 und 10^7. Dass dies ausreicht, zeigt eine Abschätzung des für

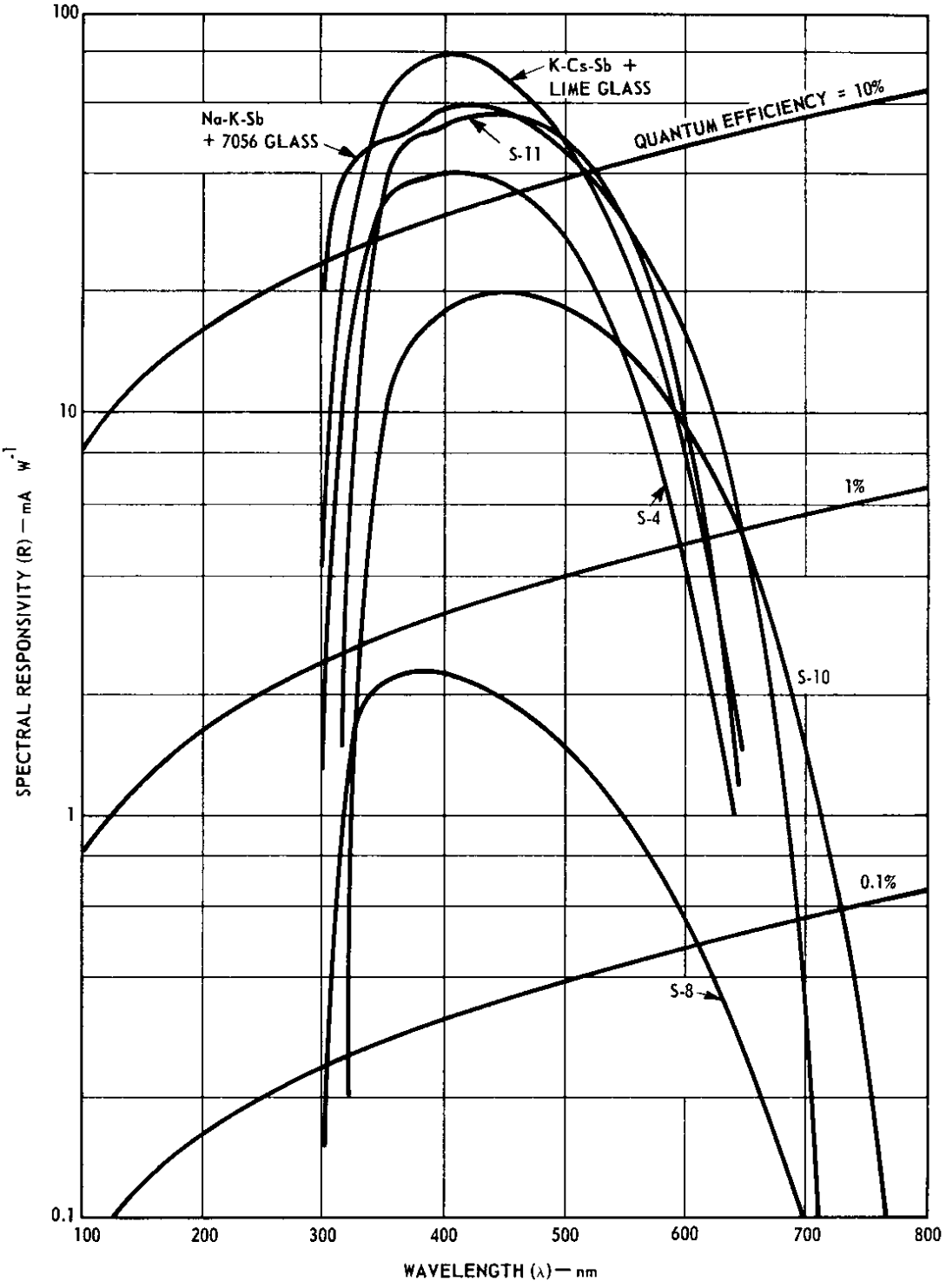


Abbildung 10: Empfindlichkeitskurven von Photomultipliern für Anwendungen im sichtbaren Spektralbereich.

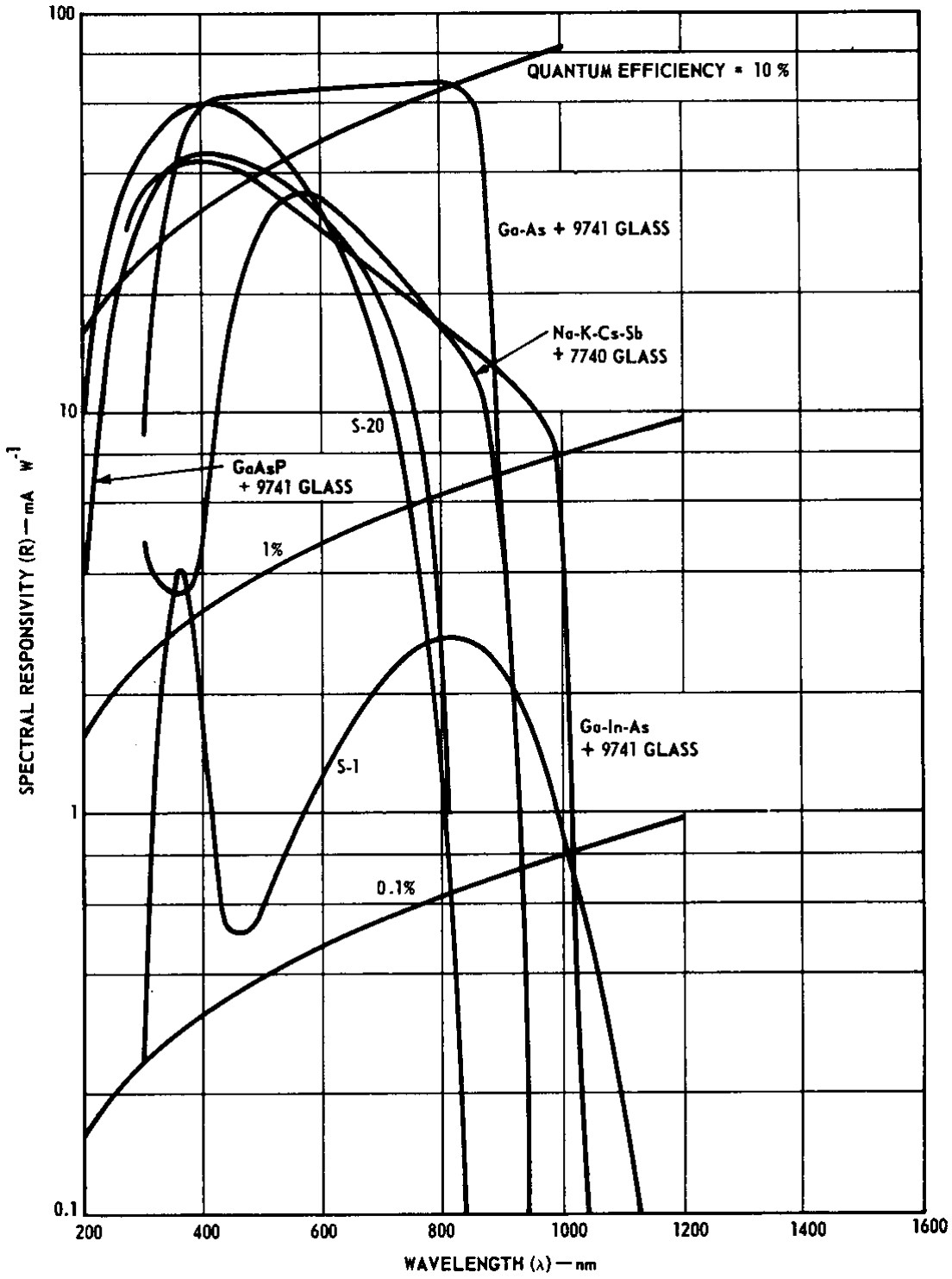


Abbildung 11: Empfindlichkeitskurven von rot- und infrarotempfindlichen Photomultipliern.

ein einzelnes nachgewiesenes Photon an der Anode entstehenden Spannungsimpulses

$$U_{\text{puls}} = e \cdot V \cdot R \cdot \tau^{-1} \ . \tag{1.4}$$

An einem Lastwiderstand $R = 75\,\Omega$ werden bei einer Verstärkung $V = 10^7$ und einer Laufzeitstreuung $\tau = 10^{-9}$ sec etwa 100 mV erreicht.

Versorgungsspannung: Die Verstärkung einer einzelnen Dynodenstrecke ist leicht sublinear von der Beschleunigungsspannung abhängig, $V_1 \propto U_1^x$ mit $x = 0.7 \ldots 0.9$, bedingt durch die bei höheren Energien größere Eindringtiefe. Bei n Dynoden ist die Gesamtverstärkung

$$V \propto (U_1^x)^n = U_1^{nx} \propto U_{\text{B}}^{nx} \ , \tag{1.5}$$

mithin sehr empfindlich von der Betriebsspannung U_{B} abhängig. Wegen dieser Abhängigkeit muss die Versorgungsspannung sehr gut konstant und störungsfrei gehalten werden.

Die einzelnen Dynodenspannungen werden durch einen hochohmigen Spannungsteiler eingestellt (meist im Gehäuse direkt am Sockel aufgebaut). Der Spannungsteiler soll einerseits hochohmig sein, um wenig Verlustwärme zu produzieren, andererseits niederohmig genug, um zu gewährleisten, dass sich bei den im Betrieb auftretenden Lichtintensitäten die Spannungsverhältnisse an den Dynoden nicht merklich verändern. An der Kathode liegt die (negative) Betriebsspannung an, der Anodenstrom wird zur Betriebsspannungsmasse hin gemessen[3].

Mikrokanalplatten: Eine Sonderform der Sekundärelektronenvervielfacher sind Mikrokanalplatten. Die Vervielfachung findet dort nicht diskretisiert auf Dynoden sondern quasikontinuierlich in geeignet beschichteten 'Kanälen' statt. Vorteile

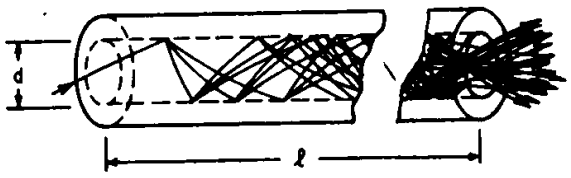


Abbildung 12: Sekundärelektronenvervielfachung in einem 'Kanal' einer Mikrokanalplatte.

der Mikrokanalplatten sind kompakte Bauform, Unempfindlichkeit gegen Magnetfelder und die Möglichkeit einer ortsaufgelösten Vervielfachung (Bildverstärker), Nachteil ist der durch den großen Herstellungsaufwand bedingte hohe Preis.

[3] Bei Impulsanwendungen (Szintillationszähler) macht man's im allgemeinen umgekehrt, die Kathode liegt auf Masse (größere Betriebssicherheit), der Ladungsimpuls an der auf Hochspannung liegenden Anode wird durch einen Kondensator ausgekoppelt. Fast alle Hochspannungsversorgungsgeräte für Photomultiplier können zwischen diesen beiden Betriebsarten umgeschaltet werden.

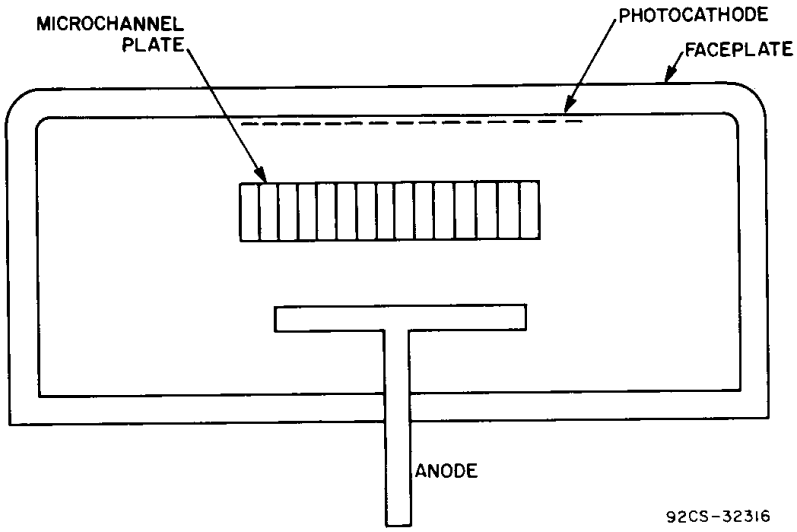
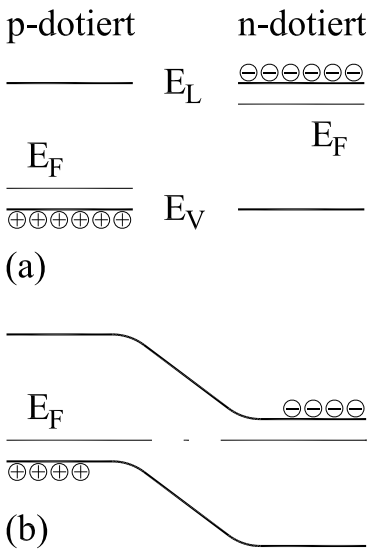


Abbildung 13: Photomultiplier mit einer Mikrokanalplatte als Sekundärelektronenvervielfacher; aus [7].

1.3.3 Photodioden, Phototransistoren

Die Funktionsweise von Photodioden und Phototransistoren ist in Lehrbüchern zur Halbleiterphysik meist recht ausführlich beschrieben [9]. Die Skizzenfolge der Abbildung 14 soll die grundlegenden Mechanismen verdeutlichen, skizziert sind die Energieverhältnisse im Ortsraum (Bandstruktur).



p-n-Übergang im Gleichgewicht: Bringt man (was so natürlich nur theoretisch möglich ist) einen p- und einen n-dotierten Halbleiter miteinander in Kontakt, so stellt man dadurch einen 'abrupten' p-n-Übergang her. Die Fermi-Energie E_F bzw. das chemische Potential liegt beim p-Halbleiter in der Nähe des Valenzbandes (E_V), beim n-Halbleiter in der Nähe des Leitungsbandes (E_L) – Teilbild (a). Beim Kontakt fließen solange bewegliche Ladungen (Elektronen und Löcher) aus dem kontaktnahen Bereich ab, bis die Fermi-Energie im gesamten System konstant ist (b).

Die ionisierten Akzeptoren (Dichte N_A) und Donatoren (Dichte N_D) bleiben als ortsfeste *Raumladungen* in der *Raumladungszone* zurück (c). Die genauen Verhältnisse lassen sich durch die Integration der Poisson-Gleichung

$$\Delta\varphi = -\frac{\rho}{\varepsilon\varepsilon_0} \quad \text{bzw. eindimensional} \quad \frac{d^2\varphi}{dx^2} = -\frac{\rho(x)}{\varepsilon\varepsilon_0} \tag{1.6}$$

berechnen. Einmalige Integration liefert den Feldverlauf (Felder außerhalb der Raumladungszone = 0), nochmalige Integration den Potentialverlauf (Potentialdifferenz = ursprüngliche Differenz der beiden Fermi-Energien).

Einstrahlung von Licht: Werden Lichtquanten eingestrahlt, deren Energie ausreicht, um Elektronen aus dem Valenzband ins Leitungsband anzuheben ($h\nu > E_L - E_V$), so werden zusätzliche Elektron-Loch-Paare gebildet. Geschieht dies im Bereich der Raumladungszone, werden sie durch das dort vorhandene Feld rasch getrennt (d), *Drift*-Strom fließt. Außerhalb der Raumladungszone ist die Trennung relativ unwahrscheinlich, da dort kein Feld vorhanden ist (e); ein geringer *Diffusions*-Strom kann fließen durch Ladungsträger, die zum p-n-Übergang diffundieren, überwiegend findet jedoch *Rekombination* statt. Durch intensivere Lichteinstrahlung werden die Potentialverhältnisse am p-n-Übergang merklich geändert, die Ladungsträgerkonzentrationen, die im Gleichgewichtsfall mit einer einheitliche Fermi-Energie berechnet werden konnte, werden nun formal durch zwei *Quasiferminiveaus* beschrieben (f). Die von der Photodiode gelieferte Spannung entspricht der Differenz der beiden Fermi-Energien.

Die in Teilbild (f) dargestellten Potentialverhältnisse stellen sich bei *photovoltaischer* Verwendung einer Photodiode ein (Photoelement, Solarzelle). Bei Detektoranwendungen betreibt man die Diode meist mit angelegter Sperrspannung (g). Durch diese Betriebsart wird die Raumladungszone verbreitert und das dort vorhandene elektrische Feld erhöht. Beides verbessert die Detektoreigenschaften (größeres empfindliches Volumen, geringere Kapazität, schnellere Ladungsträgersammlung).

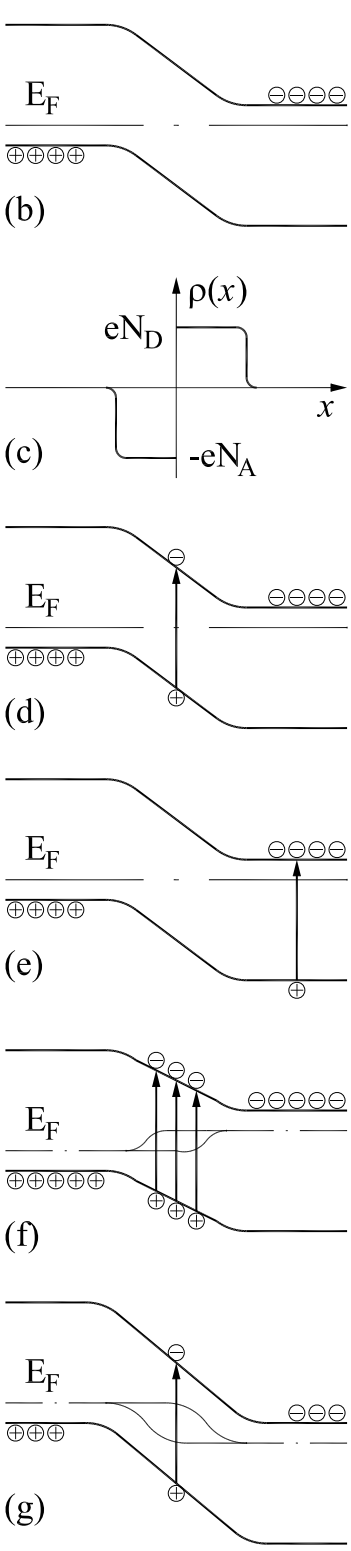


Abbildung 14: p-n-Photodiode, Bandstrukturen.

Kennlinie und Arbeitspunkt: Die Strom-Spannungs-Kennlinien einer Photodiode sind in Abbildung 15 skizziert. Bei Beleuchtung verschiebt sich die Kennlinie über einen sehr großen Bereich linear mit der Lichtintensität zu negativen Strömen.

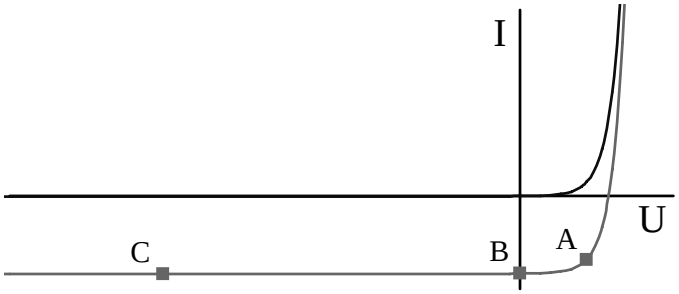


Abbildung 15: Strom-Spannungs-Kennlinien einer Photodiode ohne (obere Kurve) und mit Lichteinstrahlung (untere Kurve). A, B und C sind typische Arbeitspunkte.

Wichtige Arbeitspunkte sind:
A: Leistungsoptimierung, Diode als Spannungsquelle, Produkt aus Strom und Spannung möglichst groß, wird verwendet bei Solarenergieanwendungen.
B: Kurzschluss, einfachste Detektorbetriebsart, Strommessung ohne zusätzlichen Aufwand, der Kurzschlustrom ist proportional zur Lichtintensität.
C: Sperrspannung, die Diode wird in Sperr-Richtung betrieben, der Strom setzt sich aus Kurzschlussstrom (proportional zur Lichtintensität) und Sperrstrom zusammen. Standardbetriebsart insbesondere für p-i-n-Strukturen (Teilchendetektoren, vgl. 1.4.2).

Lawinenphotodioden: Bei hohen Sperrspannungen werden durch das große Feld in der Raumladungszone die Ladungsträger so sehr beschleunigt, dass bei Stößen weitere Ladungsträger angeregt werden, es kommt zum *Lawinendurchbruch*, der Sperrstrom erhöht sich drastisch. Die Diodenkennlinie knickt im Sperrbereich nach unten, d. h. zu hohen Sperrströmen hin ab. Dieser üblicherweise unerwünschte Effekt wird bei speziell dafür konstruierten Photodioden zur Stromverstärkung genutzt, die Verstärkung liegt zwischen 10 und 1000. Durch die zusätzliche Verstärkung in der Raumladungszone erreicht man mit Lawinenphotodioden in günstigen Fällen Einzelphotonenempfindlichkeit, dies mit deutlich höheren Quantenausbeuten (60...80 %) als bei Photomultipliern. Bei der Herstellung sind über die gesamte Detektorfläche sehr enge Toleranzen einzuhalten, damit der Verstärkungseffekt nicht zu sehr örtlich variiert, großflächige Detektoren sind dadurch nicht herstellbar. Eine interessante Anwendung (bei dem die Detektorfläche nur eine untergeordnete Rolle spielt) sind hochempfindliche Empfängerdioden bei der optischen Nachrichtenübertragung.

Halbleitermaterialien: Der für optische Anwendungen wesentlichste Materialparameter bei Halbleitern ist die Breite der verbotenen Zone, die Bandlücke (Bandgap, E_g). Sie bestimmt einerseits die langwellige Grenze des nutzbaren Wellenlängenbereichs, andererseits aber auch die Aktivierungsenergie für die thermische Anregung von Ladungsträgern und damit den Dunkelstrom. Für eine große Empfindlichkeitsbandbreite muss man auch bei Sperrschichtphotodetektoren mit hohem Dunkelstrom, damit verbundenem hohen Rauschsignal

bezahlen. Durch Kühlung kann man die Verhältnisse insbesondere bei langwelligen Detektoren deutlich verbessern. Einige gebräuchliche Materialien mit ihren Anwendungsbereichen sind in Abbildung 16 zusammengestellt.

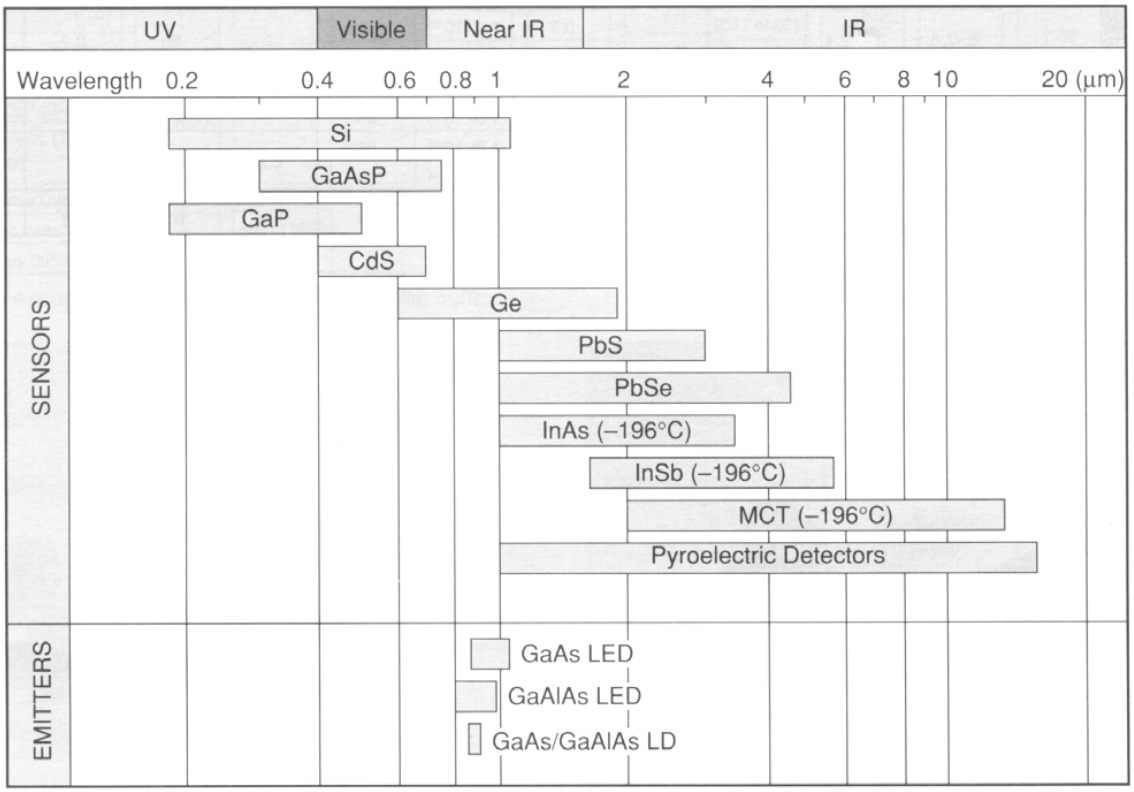


Abbildung 16: Anwendungsbereiche für verschiedene zur Herstellung von Photodetektoren verwendete Halbleitermaterialien; aus [10]. Als derzeit im Bereich der optischen Nachrichtenübertragung (1.3 oder 1.55 μm) sehr prominenter Halbleiter fehlt die III-V-Mischverbindung InGaAs mit einem Anwendungsbereich, der etwa dem von Ge entspricht. Die Abkürzung MCT steht für Mercury Cadmium Telluride *(*$\mathrm{Hg_{1-x}Cd_xTe}$*), einer II-VI-Mischverbindung mit stark zusammensetzungsabhängiger Bandlücke.*

Der Verlauf des Absorptionkoeffizienten für Licht in der Nähe der Bandlücke ($h\nu \gtrapprox E_\mathrm{g}$), d. h. nahe der langwelligen Empfindlichkeitsgrenze eines Halbleiters wird wesentlich von der Form der Energiebänder im k-Raum bestimmt. Bei Halbleitern mit *indirekter* Bandstruktur (Si, Ge) steigt der Absorptionskoeffizient zunächst nur mäßig an, bei solchen mit *direkter* (praktisch alle anderen) dagegen sehr steil. Für die Herstellung bedeutet das, dass bei Halbleitern mit indirekter Bandstruktur eine Optimierung für den jeweiligen Verwendungszweck erforderlich ist: gute Empfindlichkeit in der Nähe von E_g erfordert z. B. eine breite Raumladungszone. Einen Überblick über unterschiedlich optimierte Si-Photodioden eines Herstellers gibt Abbildung 17.

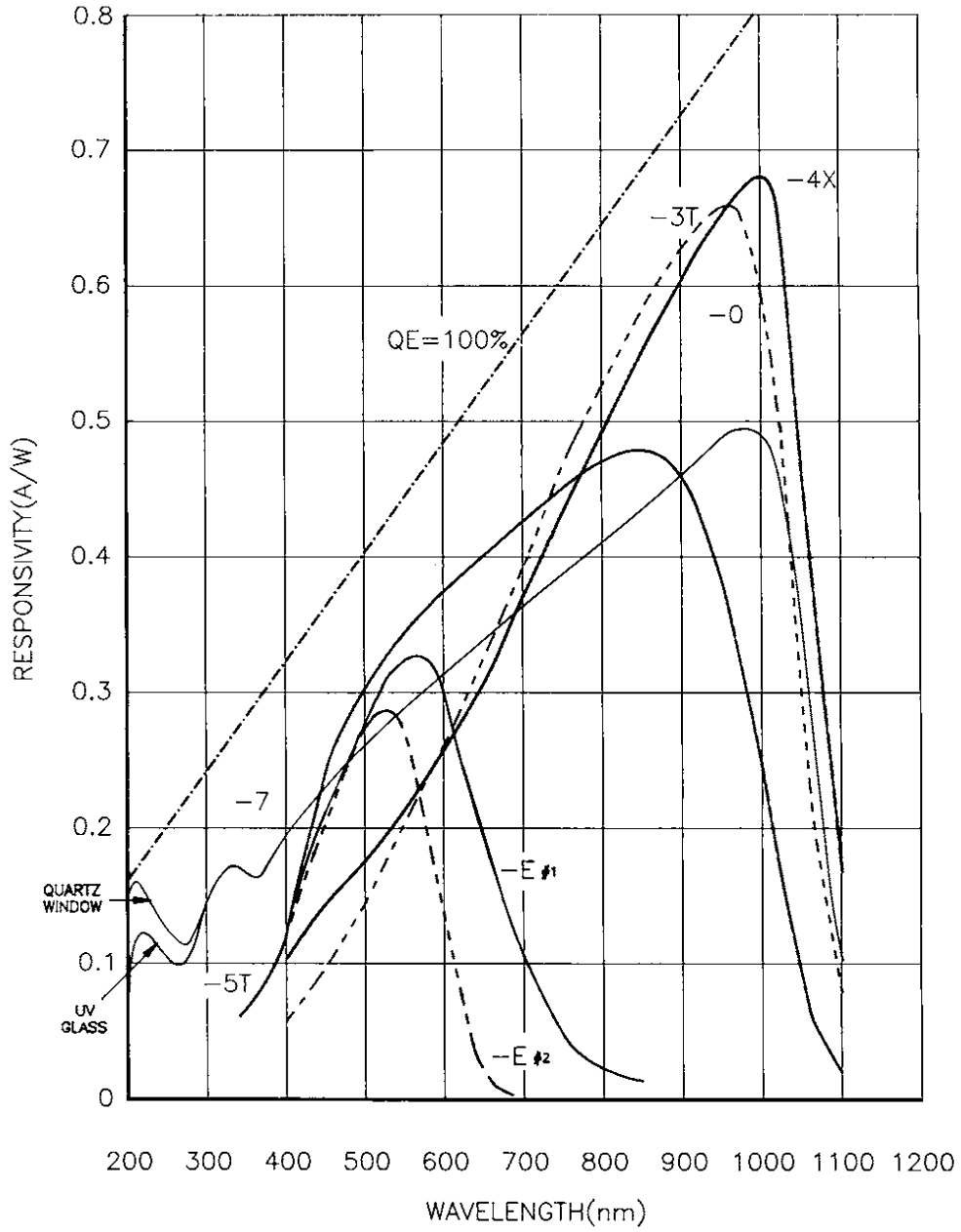


Abbildung 17: Empfindlichkeitskurven für unterschiedlich optimierte Silizium-Photodioden; aus [11]. Mit eingezeichnet ist die 'Idealkurve' für 100 % Quantenausbeute. Im UV-Bereich ist die Empfindlichkeit zusätzlich vom Fenstermaterial abhängig.

Bei Halbleitern mit direkter Bandstruktur ist eine solche Optimierung in der Regel nicht nötig, der Absorptionskoeffizient ist knapp oberhalb von E_g schon sehr hoch. Eine exemplarische Empfindlichkeitskurve für eine Photodiode aus einem Halbleiter mit direkter Bandstruktur (InGaAs) zeigt Abbildung 18.

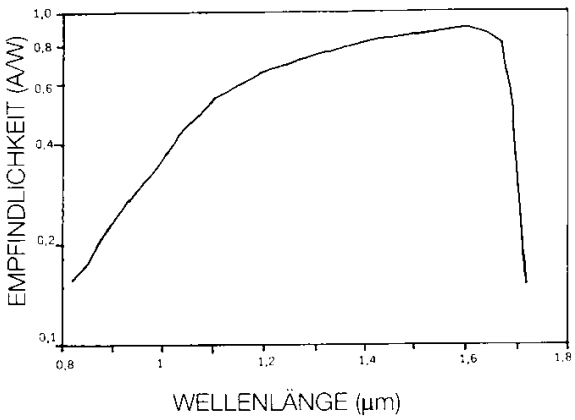


Abbildung 18: Typische Empfindlichkeitskurve für InGaAs-Photodioden (Halbleiter mit direkter Bandstruktur); aus [12].

Phototransistoren: Die Funktionsweise von Phototransistoren wird aus dem Ersatzschaltbild (Abbildung 19) deutlich: Der Photostrom einer Photodiode (beleuchtete Basis-Kollektor-Sperrschicht des Transistors) wird um den Faktor der Stromverstärkung des Transistors verstärkt. Der Signalstrom ist zwar wesentlich höher, die Linearität jedoch schlechter als bei Photodioden, da die Stromverstärkung vom Basisstrom abhängig ist.

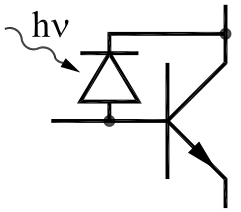


Abbildung 19: Phototransistor, Ersatzschaltbild.

Benötigt man eine Verstärkung des Messsignals mit guter Linearität direkt am Detektor, sollte man statt Phototransistoren besser integrierte Bausteine verwenden, bei denen eine Photodiode mit einem geeigneten Operationsverstärker im gleichen Gehäuse kombiniert ist.

Detektor-Arrays: Zur ortsaufgelösten Messung von Lichtintensitäten werden zunehmend ein- oder zweidimensionale Anordnungen von Photosensoren benutzt. Eindimensionale beispielsweise zur Messung eines kompletten Spektrums an einem Spektrometer, zweidimensionale vorzugsweise zur Bilderfassung. Als Detektormaterial wird wegen der dafür vorhandenen Technologie in erster Linie Silizium verwendet, zumindest für den sichtbaren Spektralbereich, allerdings nicht in bipolarer, sondern in MOS-Technologie. Für infrarotempfindliche Arrays sind neben Spezialmaterialien wie PtSi auch miniaturisierte Photoleiteranordnungen oder auch Mikrothermoelemente gebräuchlich. Zum Auslesen der Arrays werden zwei Techniken verwendet:

- Bei CCDs (*C*harge *C*oupled *D*evices) werden durch getakteten Ladungstransfer die in den Pixelelementen gesammelten Ladungen durch die Spalten und Reihen des Arrays zum Ausgang verschoben. Relativ billig in der Herstellung, nicht allzu genau.
- CMOS-Auslesearrays lesen dagegen jedes Pixelelement individuell durch eine Anordnung von einzelnen integrierten CMOS-Transistoren aus. Relativ aufwendig und teuer, aber sehr genau.

1.3.4 Photoleiter

Die Bedeutung von Photoleitern hat mit der Entwicklung neuer Materialien für Photodioden deutlich abgenommen. Eine neue Anwendung fanden sie jedoch in jüngster Zeit – miniaturisiert und integriert – im Bereich der infrarotempfindlichen Bildsensoren.

Photoleiter sind Materialien – in der Regel Halbleiter, deren Leitfähigkeit durch die optische Anregung von Ladungsträgern verändert wird. Diese Anregung kann über drei Prozesse erfolgen:

Intrinsische Photoleiter: Die Ladungsträger werden durch Anregung vom Valenzband ins Leitungsband erzeugt, dies entspricht dem Anregungsprozess bei Photodioden. Gebräuchliche Materialien sind III-V-Verbindungen wie InAs oder InSb und II-VI-Verbindungen wie CdS, $Hg_{1-x}Cd_xTe$ sowie PbS oder PbSe.

Extrinsische Photoleiter: Die Ladungsträgerdichte wird durch optische Übergänge von Störstellenzuständen ins Valenz- oder Leitungsband geändert. Ein klassisches Material war Germanium mit unterschiedlichen Dotierungen (Hg, Au, Cu, Ga); die minimale Anregungsenergie (Abstand vom Störstellenniveau zum Leitungs- bzw. Valenzband) – damit die langwellige Grenze des Detektors – ist abhängig vom Störstellentyp. Derzeit noch interessant ist dotiertes Silizium, da es mit moderner Technologie (*Planartechnik*) strukturierbar ist.

Intraband-Photoleiter: Durch optische Anregung werden Ladungsträger innerhalb eines Bandes zwischen Zuständen unterschiedlicher Beweglichkeit verschoben; es wird nicht die *Dichte* der Ladungsträger verändert, sondern deren *Beweglichkeit*. Die Leitfähigkeit ändert sich als Produkt aus Dichte und Beweglichkeit. Wenig verwendet, praktisch einziges bekanntes Material ist n-Typ-InSb.

1.3.5 Thermische Detektoren

Wie bei der Temperaturmessung können auch zur Messung von Strahlungsintensitäten (insbesondere im infraroten Wellenlängenbereich – *Wärme*-Strahlung) alle temperaturabhängigen physikalischen Größen (und das sind praktisch alle) genutzt werden. Der große Vorteil thermischer Detektoren liegt in dem weitgehend wellenlängenunabhängig konstanten Empfindlichkeitsverlauf, ein Nachteil in der gegenüber Quantendetektoren geringeren Maximalempfindlichkeit (vgl. Abbildung 6). Auch hier wieder nur eine Auswahl typischer Detektorprinzipien, Ausführlicheres liefert z. B. [6] und die darin zitierte Primärliteratur.

Pneumatische Detektoren: Ausgenutzt wird die thermische Ausdehnung von Gasen. Am bekanntesten ist die Golay-Zelle [13], die mit Xe gefüllt ist (Gas mit geringer Wärmeleitfähigkeit). Das Gasvolumen ist mit einer Membran abgeschlossen, deren Auslenkung optisch gemessen wird. Obwohl vor über 50 Jah-

ren entwickelt, ist die Golay-Zelle immer noch einer der empfindlichsten IR-Detektoren für Raumtemperaturbetrieb.

Pyroelektrische Detektoren beruhen auf der Änderung der spontanen Polarisierung mit der Temperatur. Der Effekt tritt grundsätzlich bei allen Ferroelektrika auf. Man benutzt kondensatorähnliche Anordnungen, ein ferroelektrisches Material zwischen 2 aufgedampften Metallelektroden. Temperaturänderungen ändern die Polarisierung, mithin die zwischen den Elektroden messbare Spannung. Gebräuchliche Materialien sind Triglyzinsulfat (TGS), Strontium-Barium-Niobat (SBN), Lithiumniobat und Lithiumtantalat.

Bolometer sind temperaturabhängige Widerstände, zur Strahlungsmessung sind sie umso besser geeignet, je ausgeprägter die Temperaturabhängigkeit ihres Widerstandswerts ist. Verschiedene physikalische Konzepte sind naheliegend:

Speziallegierungen und *Metalloxide* in Dünnschichttechnik, dadurch mit kurzen Ansprechzeiten und geringer Wärmekapazität. Als Arrays teilweise auch in IR-Bildsensoren.

Supraleiter knapp unterhalb des Sprungpunkts, sehr empfindlich aber auch sehr aufwendig im Betrieb, da die Temperatur sehr genau konstant gehalten werden muss.

Kryo-Bolometer meist aus Halbleitern, die bei tiefen Temperaturen (4 K) sehr große Widerstandsänderungen zeigen.

Thermoelemente zeichnen sich durch Robustheit und einfache Handhabung aus, leider aber auch durch relativ geringe Empfindlichkeit. Durch Serienschaltung kann man die Größe des Messsignals deutlich erhöhen (Thermosäulen, meist in Dünnfilmtechnik), neuere Entwicklungen arbeiten mit miniaturisierten Anordnungen auf Halbleiterbasis (deutlich größere Thermokraft als bei Metallen).

1.4 Teilchen

Teilchendetektoren werden in der Hochenergiephysik, der Kernphysik und der Oberflächenphysik, außerdem als wichtige Nachweisgeräte in der Röntgentechnik verwendet. Unter dem Begriff 'Teilchen' sind einerseits Elementarteilchen und Ionen, andererseits aber auch Quanten elektromagnetischer Strahlung oberhalb einer nicht exakt definierbaren Mindestenergie zu verstehen. Die Funktionsprinzipien einiger typischer Detektortypen aus den Bereichen der Kern- und Oberflächenphysik werden kurz erläutert.

1.4.1 Szintillationszähler

Zu den ältesten Nachweistechniken für radioaktive Strahlung gehören *Szintillations*-Vorgänge, d. h. die Erzeugung von schwachen Lichtblitzen in geeigneten Materialien (Zinksulfid-Schirme zum Nachweis von Alpha-Strahlen etc.). Wäh-

rend in den Anfangszeiten der Kernphysik das dunkeladaptierte Auge des Experimentators eine große Rolle spielte, werden die Szintillationen inzwischen fast ausschließlich mit Photomultipliern nachgewiesen. Szintillatormaterial und -geometrie wählt man passend zur nachzuweisenden Strahlung, das Kathodenmaterial des Photomultipliers passend zum Emissionsspektrum des Szintillators.

Geladene Teilchen (Alpha-, Betastrahlung) regen im Szintillatormaterial – einem Isolator – durch Coulomb-Wechselwirkung Elektronen an und verlieren dadurch Energie. Gamma- oder Röntgenquanten verlieren einen Großteil oder ihre gesamte Energie in einem primären Stoßprozess (Compton- oder Photo-Effekt), das dabei erzeugte schnelle Elektron verhält sich wie ein Beta-Teilchen. Die zunächst hoch ins Leitungsband angeregten sekundären Elektronen thermalisieren zur Bandkante (Phononenanregung) und rekombinieren mit Defektelektronen an der Valenzbandoberkante. Mit gewisser Wahrscheinlichkeit wird dabei jeweils ein Lichtquant erzeugt; die Übergangswahrscheinlichkeit für diese *strahlende* Rekombination wird häufig durch Dotierungssubstanzen[4] erhöht, Thermalisierung und Rekombination laufen dann über geeignete Energieniveaus der Störstellen. Der mittlere Energieverlust pro generiertem Elektron-Loch-Paar ist über einen weiten Energiebereich eine nur vom Szintillatormaterial abhängige nahezu konstante Größe (typischerweise 10. . . 100 eV). Damit ist die Gesamtzahl der Lichtquanten proportional zur Anfangsenergie des Teilchens, somit auch die Anzahl der aus der Kathode des Photomultipliers ausgelösten Elektronen und die Größe des Ladungsimpulses an der Anode[5]: Energiespektroskopie ist möglich.

Energieverlust pro Elektron-Loch-Paar, konkurrierende nichtstrahlende Rekombinationsprozesse, Lichtsammelwirkungsgrad und Quantenausbeute der Photokathode haben insgesamt einen Energieaufwand von etwa 1 keV pro an der Photokathode erzeugtem Photoelektron zur Folge. Bei charakteristischen Teilchenenergien von 1 MeV entstehen somit ca. 1000 Photoelektronen. Deren Wahrscheinlichkeitsverteilung (Poisson- bzw. Gauß-Verteilung) begrenzt prinzipiell die Enegieauflösung der Szintillationszähler auf günstigstenfalls etwa 5 %, einen Wert, der deutlich schlechter ist als bei Halbleiterdetektoren. Für spektroskopische Anwendungen (Gammaspektroskopie) haben Szintillationszähler daher inzwischen nur noch eine geringe Bedeutung, wohl aber für solche Messanwendungen, bei denen es auf einfache Handhabung (keine Kühlung notwendig) oder große Detektorvolumina (Flüssig- oder Kunststoffszintillatoren) ankommt.

[4] Natriumjodid, das wichtigste Szintillatormaterial für Gammadetektoren, wird mit etwa 1 % Thallium dotiert, um eine kurze Lumineszenzabklingzeit, damit auch eine gute Lumineszenzausbeute zu erreichen.

[5] Durch eine geeignete Bauform ist sicherzustellen, dass das Szintillatorvolumen groß genug ist, damit das nachzuweisende Teilchen seine Energie vollständig verliert, und dass reproduzierbar alle Lichtquanten oder zumindest ein konstanter Bruchteil die Photokathode erreichen.

1.4.2 Halbleiterdetektoren

Halbleiterdetektoren zum Teilchennachweis werden als p-i-n-Diodenstrukturen gebaut. Die p- und n-Gebiete begrenzen eine undotierte eigenleitende (*i*ntrinsic) Zone (Abbildung 20). Durch eine an die Diode angelegte Sperrspannung wird ein elektrisches Feld erzeugt, das die vom ionisierenden Teilchen generierten Elektron-Loch-Paare quantitativ trennt, der Ladungsimpuls ist proportional zur Teilchenenergie bzw. zum Energieverlust. Da der mittlere Energieaufwand pro erzeugtem Elektron-Loch-Paar bei etwa 3 eV liegt, ist die Energieauflösung um mehr als eine Größenordnung besser als bei Szintillationszählern. Dicke von i-Schicht und Bedeckung sowie verwendetes Halbleitermaterial hängen von der Anwendung ab. Die Dicke der i-Schicht sollte der Reichweite der Teilchen entsprechen, die Dicke der Deckschicht deutlich kleiner sein. Für den Nachweis von Alpha- und Beta-Teilchen, teilweise auch für Röntgenstrahlung werden großflächige Silizium-Dioden verwendet, bei denen die i-Schicht nur wenige Mikrometer unter der Oberfläche liegt (Oberflächensperrschichtzähler). Für Gamma-Detektoren ist man dagegen auf Germanium angewiesen, das wegen seiner größeren Elektronendichte eine entsprechend größere Absorptionskonstante für die notwendigen Primärprozesse (Compton- und Photo-Effekt) aufweist. Bei diesen Detektoren liegt die Dicke der i-Schicht in der Größenordnung von Zentimetern. Der große Nachteil von Germanium ist seine große Leitfähigkeit bei Raumtemperatur, die Detektoren werden deshalb im Betrieb mit flüssigem Stickstoff (77 K) gekühlt.

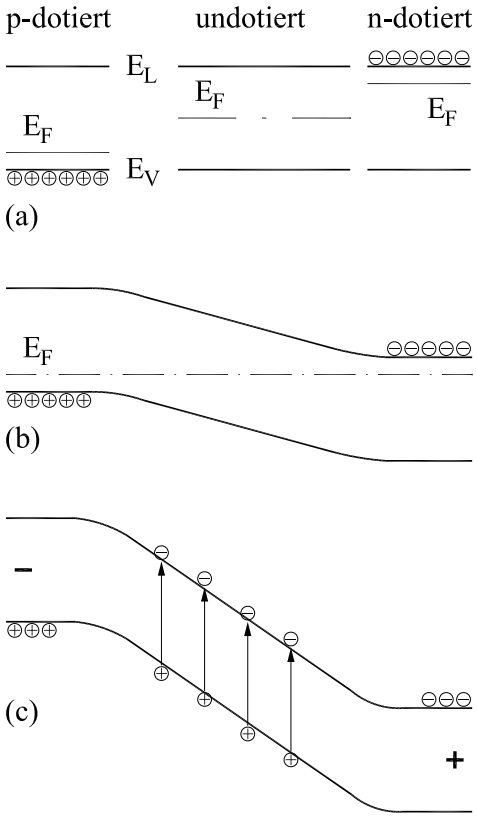


Abbildung 20: p-i-n-Detektor, Energiebandstrukturen: E_L, E_V: Leitungs- und Valenzbandenergien.
(a) Lage der Fermi-Energien E_F in p-, i- und n-Halbleitern.
(b) Spannungslose p-i-n-Diode.
(c) p-i-n-Diode mit angelegter Sperrspannung; die in der i-Schicht generierten Elektron-Loch-Paare werden durch das starke Feld getrennt.

1.4.3 Sekundärelektronenvervielfacher

Zum hochempfindlichen Nachweis langsamer geladener Teilchen (Elektronen bei der Photoelektronenspektroskopie, Ionen in Massenspektrometern) werden offe-

ne Sekundärelektronenvervielfacher im Ultrahochvakuum eingesetzt. Sie entsprechen in Technik und Aufbau in etwa den in Photomultipliern verwendeten (vgl. 1.3.2); da jedoch keine Photokathode vorhanden ist, entfällt die Problematik der thermisch generierten Hintergrundereignisse, ein praktisch untergrundfreier Nachweis von Einzelereignissen ist möglich. Auch in diesen Anwendungsbereichen sind Mikrokanalplatten eine interessante Alternative zur herkömmlichen Technik, insbesondere dann, wenn deren Ortsauflösung zusätzliche physikalische Informationen liefern kann (Elektronenbeugung an Oberflächen).

1.5 Kraft

Verbreitete Kraftsensoren sind Dehnungsmessstreifen (DMS). Das sind Widerstände, die bei einer Längenänderung – hervorgerufen durch Zug oder Druck – ihren Widerstandswert definiert und großzügig ändern. Die Widerstandsänderung ist ein Maß für die Längenänderung und damit in guter Näherung für die ausgeübte Kraft. DMS werden meist nicht direkt verwendet, sondern auf ein geeignetes Trägermaterial aufgeklebt, dessen Verzerrung dann damit gemessen wird. Abbildung 21 zeigt als Beispiel einen Kraftmesser, der mit DMS arbeitet. Die Verzerrung und damit der Messbereich (in diesem Fall laut Aufschrift '3 kg') wird durch die Geometrie des Trägermaterials im Bereich der DMS eingestellt.

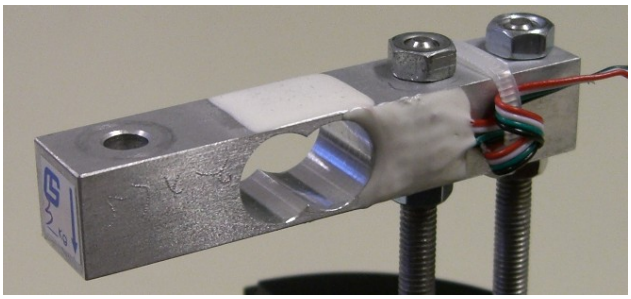


Abbildung 21: Kraftmesser mit Dehnungsmessstreifen (DMS) in Brückenschaltung. Zwei DMS sind auf der Oberseite, ebenfalls zwei auf der Unterseite angebracht.

Brückenschaltung als Messprinzip: Die Widerstandsänderungen ΔR bei Dehnungsmessstreifen sind immer klein gegenüber dem Widerstandswert R – eine Eigenschaft, die sie mit vielen anderen Sensoren gemeinsam haben[6]. Grundsätzlich ist das keine Schwierigkeit, da Widerstände mit großer Genauigkeit gemessen werden können. Probleme ergeben sich aber dadurch, dass auch andere Einflüsse den Widerstandswert verändern, beispielsweise die Temperatur. Bei robusten Messsysteme sollte man das kompensieren. Für die Temperatur lässt sich das dadurch erreichen, dass man einen Spannungsteiler aus zwei identischen Widerstandssensoren baut und nur einen davon der Messgröße aussetzt

[6]Es gibt nur ganz wenige Ausnahmen, zum Beispiel den soeben mit dem Nobelpreis ausgezeichneten *Riesenmagnetowiderstand*.

(oder – bei DMS – einen auf Zug und den anderen auf Druck beansprucht). Die Schaltung ist in Abbildung 22 skizziert.

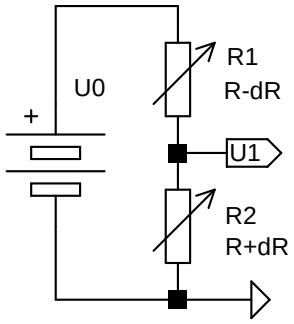


Abbildung 22: Zwei gleiche Sensoren (z. B. Dehnungsmessstreifen) in Spannungsteilerschaltung zur Kompensation von äußeren Einflüssen wie Temperatur u. ä.

Wir gehen von gleichen Widerständen R_1 und R_2 aus, dann wird die Messspannung U_1 ohne Anliegen einer Messgröße (Kraft o. ä.)

$$U_1 = U_0 \frac{R_2}{R_1 + R_2} = \frac{U_0}{2} \ . \tag{1.7}$$

Widerstandsänderungen ΔR werden durch Temperaturänderungen ΔT und durch Krafteinwirkung ΔF hervorgerufen, der Aufbau soll hier so sein (vgl. Abbildung 21), dass die Temperaturänderung auf beide Widerstände in gleicher Weise, die mechanische Verzerrung dagegen komplementär (Zug und Druck) wirkt

$$\Delta R_1 = \frac{\partial R}{\partial T}\Delta T - \frac{\partial R}{\partial F}\Delta F \ , \quad \Delta R_2 = \frac{\partial R}{\partial T}\Delta T + \frac{\partial R}{\partial F}\Delta F \ . \tag{1.8}$$

Die Änderung von U_1 wird dann

$$\Delta U_1 = \frac{\partial U_1}{\partial R_1}\Delta R_1 + \frac{\partial U_1}{\partial R_2}\Delta R_2 = \frac{U_0}{2R}\left(\Delta R_2 - \Delta R_1\right) = \frac{U_0}{R}\frac{\partial R}{\partial F}\Delta F \ . \tag{1.9}$$

Der Temperatureinfluss wird offensichtlich durch die Spannungsteilerschaltung kompensiert. Und die Spannungsänderung ist proportional zur Kraft, zumindest in einem Bereich, in dem $\partial R/\partial F$ als Konstante angesehen werden kann.

Aber: Man misst nicht ΔU_1 sondern $U_1 + \Delta U_1$, und das ist

$$U_1 + \Delta U_1 = \frac{U_0}{2} + \frac{U_0}{R}\frac{\partial R}{\partial F}\Delta F \ . \tag{1.10}$$

Es ist also eine sehr kleine Änderung der Spannung $U_0/2$ zu messen, da die Widerstandsänderungen nicht besonders groß sind. Kein grundsätzliches Problem, jedoch sehr anfällig gegen Schwankungen von U_0; man müsste mithin U_0 extrem gut stabilisieren, um eine hinreichende Messgenauigkeit zu erreichen. Um das zu vermeiden, kompensiert man $U_0/2$ durch einen zweiten Spannungsteiler, der exakt gleich aufgebaut ist. Man kommt so zu der Brückenschaltung der Abbildung 23.

In der Schaltung ist auch schon angedeutet, wie die Sensorwiderstände geometrisch anzuordnen sind, um optimale Empfindlichkeit und Robustheit zu erreichen. In allen vier Brückenzweigen liegen gleichartige Sensoren, alle Widerstandswerte werden gleich groß gemacht. Jeweils gegenüberliegende (R_1 und R_4

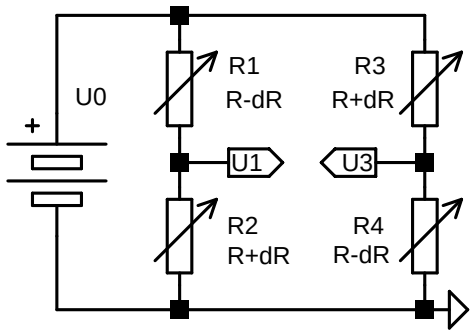


Abbildung 23: Messbrücke aus vier Sensorwiderständen.

bzw. R_2 und R_3) werden zusammen und geometrisch gleich angeordnet, man erreicht so eine Verdopplung der Messspannung ΔU_1 - ΔU_3 gegenüber dem ΔU_1 des einfachen Spannungsteilers. Der konstante Anteile $U_0/2$ ist durch die Differenzmessung U_1 - U_3 kompensiert, Schwankungen in U_0 wirken sich daher nicht mehr direkt additiv aus, sondern gehen nur differentiell in die Messgröße ein.

$$U_3 + \Delta U_3 = \frac{U_0}{2} - \frac{U_0}{R}\frac{\partial R}{\partial F}\Delta F \tag{1.11}$$

und mit Gleichung 1.10

$$U_{13} = (U_1 + \Delta U_1) - (U_3 + \Delta U_3) = 2\frac{U_0}{R}\frac{\partial R}{\partial F}\Delta F \ . \tag{1.12}$$

Das skizzierte Prinzip, Sensoren durch Brückenschaltungen genauer und robuster zu machen, finden Sie nicht nur bei Dehnungsmessstreifen, sondern auch in vielen anderen Sensoranwendungen. Als generelle Strategie ist es natürlich auch nicht auf Widerstandssensoren beschränkt, sondern lässt sich auf fast alle Sensortypen anwenden.

2 Aktoren

Interessant an Messgrößen im physikalischen Experiment ist fast immer nicht nur ihr punktueller Wert, sondern auch ihre Abhängigkeit von variablen Messparametern. Solche gemessenen funktionalen Abhängigkeiten können dann auf Modellvorstellungen übertragen werden, die dadurch initialisiert, überprüft und weiterentwickelt werden. So werden im Bereich der Festkörperphysik aus der wellenlängenabhängig gemessenen Absorption oder Lumineszenz von Kristallen Aussagen über Energiezustände von Störstellen gemacht, aus der winkelabhängig gemessenen Einkopplung von Licht in Wellenleiter die optische Moden und der Brechungsindexverlauf berechnet, aus der temperaturabhängig gemessenen Dielektrizitätskonstanten oder der temperaturabhängig gemessenen optischen Frequenzverdopplung Informationen über strukturelle Phasenübergänge gewonnen, aus der winkel- und frequenzabhängig gemessenen Elektronenspinresonanz die Art und Symmetrie von Störstellen und deren Umgebung bestimmt.

Wünschenswert ist es immer, die Parameteränderungen weitgehend automatisiert – d. h. vom Rechner gesteuert – ablaufen zu lassen, um einerseits den Experimentator von ermüdender und fehleranfälliger 'Handarbeit' zu entlasten, andererseits den Verlauf des Experiments reproduzierbar, aber gleichzeitig einfach änderbar vorzugeben.

> Durch Rechnersteuerung soll vorrangig die Reproduzierbarkeit und damit die *Qualität* des Experiments verbessert werden.

2.1 Externe Geräte

Meist werden für die Einstellung von Parametern kommerzielle externe Geräte verwendet – Temperaturregler, optische Spektrometer, Netzgeräte für Magnetspulen, Hochspannungsversorgungen und viele andere. Üblicherweise sind diese Geräte mit *Standardschnittstellen* (RS 232, HPIB, USB) ausgestattet, über die – mehr oder weniger aufwendig – eine Steuerung möglich ist. Der Befehlsumfang ist fast immer sehr individuell auf das Gerät zugeschnitten, zur Erstellung eines Steuerungsablaufs ist ein genaues Studium der Gerätebeschreibung unerlässlich. In den ersten Generationen intelligenter Geräte wurden aus Performance-Gründen oft kryptische Binärkommandos verwendet, modernere Geräte dagegen 'hören' in der Regel auf Textkommandos, die ohne umständliche Kodierung erstellt werden können und weitgehend selbstdokumentierend sind.

Manche Aktoren lassen sich unter Umständen besser und schneller, immer jedoch billiger, relativ direkt vom Rechner ansteuern, nachstehend einige Beispiele.

2.2 Leistungsschalter

Um die Netzspannung von Elektrogeräten mittlerer und höherer Leistung potentialgetrennt zu schalten, werden Relais verwendet. Klassische elektromechanische Relais haben jedoch für die direkte Ansteuerung durch Rechner verschiedene Nachteile: relativ hohen Stromverbrauch, induktive Belastung, elektrische Störungen beim Schalten. Besser zu verwenden sind *Halbleiterrelais*, Bausteine mit dem in Abbildung 24 skizzierten Funktionsschema: mit einem geringen Steuerstrom (ca. 10 mA) werden zwei Leuchtdioden betrieben, die zwei entgegengesetzt gepolte Fototransistoren schalten[7]. Die reale Schaltung ist natürlich komplizierter: um Störungen zu verringern, wird üblicherweise dafür gesorgt, dass nur im Nulldurchgang der Wechselspannung ein- und ausgeschaltet wird, von der Nulldurchgangslogik werden als Lastschalter Thyristoren angesteuert.

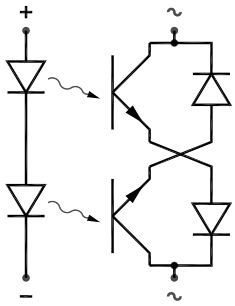


Abbildung 24: Halbleiterrelais, Funktionsschema.

Aufgrund des geringen Ansteuerstroms und eines weiten Bereichs (3. . . 30 V) der Ansteuerspannung können Halbleiterrelais problemlos an Standardschnittstellen des Rechners betrieben werden (Quittungsleitung der seriellen Schnittstelle, Einzelbit der Druckerschnittstelle). Mit geringem Aufwand können so auch Verbraucher im Kilowattbereich (Heizwiderstände in Kristallzuchtöfen o. ä.) vom Rechner geschaltet werden (Zweipunktregelungen, Puls-Pausen-Steuerungen).

2.3 Schrittmotoren

Mechanische Stellelemente im physikalischen Experiment (Verschiebetische, Hubtische, Drehtische), die im Mikro- bis Zentimeterbereich arbeiten, lassen sich relativ einfach mit dafür passenden Minimotoren ausrüsten und somit ferngesteuert bedienen. Zwei Systeme sind gebräuchlich: mit Inkrementalgebern gekoppelte Gleichstrommotoren und Schrittmotoren. Für beide gibt es komfortable komplette kommerzielle Steuergeräte, gerade aber bei Schrittmotoren ist es naheliegend, die Steuerung direkt dem Rechner zu übertragen.

Beim Schrittmotor wird ein magnetisierter Anker von einem durch geeignete Spulenströme erzeugten Drehfeld weitergedreht. Im Gegensatz zum *Synchronmotor* dreht sich das Magnetfeld in diskreten Schritten und mit beliebiger, auch wechselnder, Geschwindigkeit (bis zu einer durch die Bauart bedingten Höchstgeschwindigkeit). Die Winkelauflösung ist primär durch die *Polzahl* festgelegt, Werte zwischen 10 und 1000 sind gebräuchlich, bei Vollschrittbetrieb bedeutet das eine entsprechende Anzahl von Schritten pro ganzer Umdrehung. Die typi-

[7] **Wichtiger Sicherheitsaspekt:** Wie die Prinzipschaltung zeigt, schalten Halbleiterrelais nur einpolig ab, außerdem ohne wirkliche mechanische Trennung. Auch im abgeschalteten Zustand fließt noch ein kleiner Versorgungsstrom für die Schaltung. Wenn an einem so abgeschalteten Gerät gearbeitet wird, muss eine weitergehende Abtrennung der Stromversorgung erfolgen, beispielsweise durch einen zusätzlichen mechanischen Schalter.

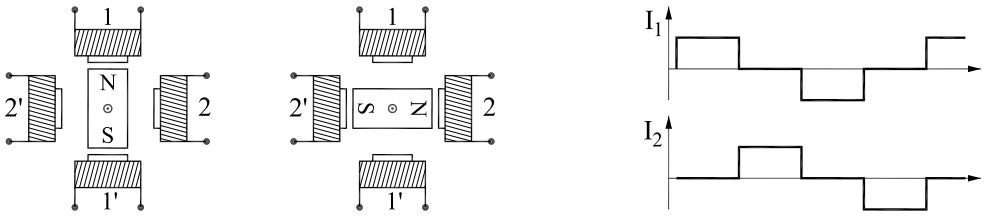


Abbildung 25: ***Vollschrittbetrieb:*** *Der Anker dreht sich bei einem Schritt zum nächsten Pol, im skizzierten Fall eines 4-Pol-Motors entspricht dies einer Rotation um 90°. Im rechten Teilbild der Verlauf der beiden Spulenströme für eine volle Umdrehung (4 Schritte), die Spulen 1 und 1' bzw. 2 und 2' sind jeweils geeignet in Serie geschaltet (⇒ Zwei-Phasen-Motor, bipolare Betriebsart).*

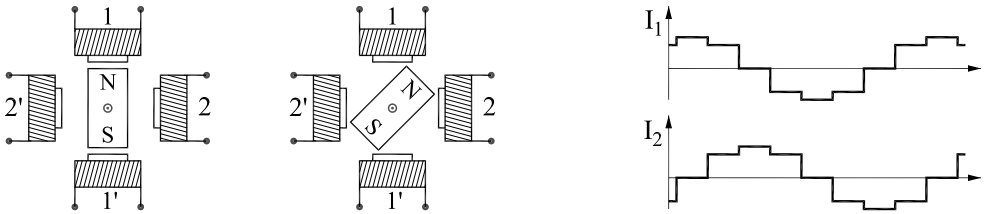


Abbildung 26: ***Halbschrittbetrieb:*** *Der Anker dreht sich bei einem Halbschritt um einen halben Polabstand weiter (hier 45°). Rechts der Stromverlauf (8 Halbschritte pro voller Umdrehung).*

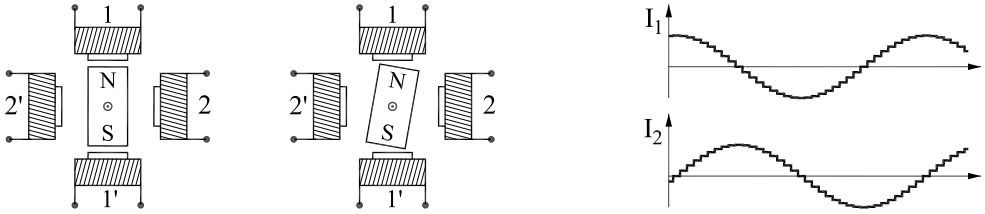


Abbildung 27: ***Mikroschrittbetrieb:*** *Der Anker dreht sich bei jedem Mikroschritt um einen Bruchteil des Polabstands weiter (hier 1/9 ≙ 10°). Die Stromverläufe nähern sich Sinus- bzw. Cosinusfunktionen an. Mikroschrittbetrieb setzt voraus, dass der Schrittmotor von seiner Bauform her dafür geeignet ist.*

schen Betriebsarten von Schrittmotoren sind in den Abbildungen 25, 26 und 27 skizziert. Zur einfacheren Darstellung des Prinzips ist die Polzahl auf 4 reduziert.

In den Abbildungen 25, 26 und 27 wird angenommen, dass der Schrittmotor jeweils *bipolar* betrieben wird. Der Spulenstrom nimmt positives und negatives Vorzeichen an. Dadurch kommt man mit 2 Phasen aus, benötigt aber etwas aufwändigere Treiberendstufen als bei *unipolarer* Betriebsart. Diese ist in Abbildung 28 schematisiert (dort für Vollschrittbetrieb, Halbschrittbetrieb ist in ähnlicher Weise wie bei der bipolaren Betriebsart möglich, grundsätzlich auch Mikroschrittbetrieb). Bei bipolarer Betriebsart werden (mindestens) 3 Zustände der Treiberstufen benötigt (positiv, null, negativ), bei unipolarer Betriebsart nur 2 (Strom, stromlos). Abbildung 29 zeigt die schematisierte Schaltung der

jeweiligen Treiberendstufen.

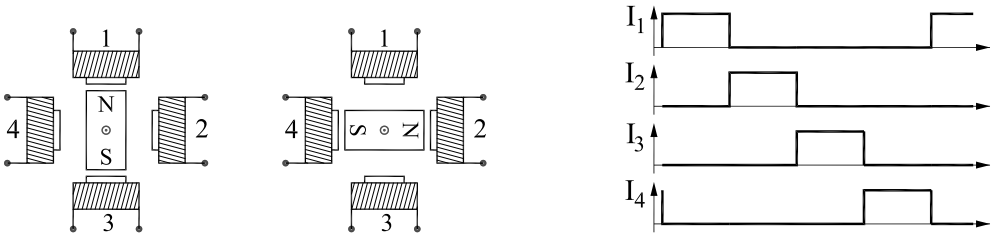


Abbildung 28: ***Unipolare Betriebsart:*** *Bei dieser Betriebsart führen die vier dargestellten Spulen jeweils einzeln Strom. Rechts der zeitliche Verlauf der 4 Spulenströme.*

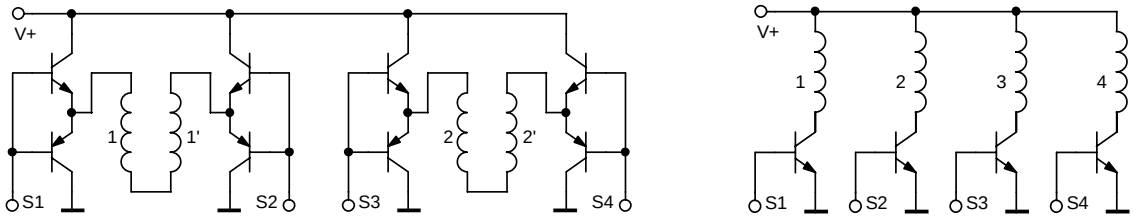


Abbildung 29: Treiberendstufen (schematisiert) für bipolaren (links) und unipolaren Betrieb (rechts) von Schrittmotoren. S1. . . S4 sind die Steuereingänge.

Rechnersteuerung: Für die Ansteuerung von Schrittmotoren durch den Rechner bieten sich vier Varianten an:

Intelligente Steuergeräte erhalten vom Rechner eine Zielvorgabe (n Schritte vorwärts) und erledigen das dazu notwendige selbständig.

Einfache Steuergeräte erwarten vom Rechner *Takt-* und *Richtungs*-Impulse (meist TTL-kompatibel) und generieren nur die zugehörige Spulenstromabfolge. Jeder einzelne Motorschritt muss vom Rechner veranlasst werden.

Treiberstufen mit Darlington-Transistoren (als ICs mit 8fach Treibern erhältlich) lassen sich über eine Parallel-Ein/Ausgabe-Karte oder über den Druckerausgang des Rechners ansteuern. In diesem Fall muss das Steuerprogramm die Abfolge der Ströme für die einzelnen Spulen (Abbildung 28) als Binärwerte erzeugen und sich den jeweiligen Status merken.

D/A-Wandler mit Stromverstärkern können verwendet werden, um einen rechnergesteuerten Mikroschrittbetrieb zu realisieren.

Geschwindigkeit: Wichtig ist es, die spezifizierte *Start-Stop*-Geschwindigkeit (bauartabhängig zwischen etwa 100 und 1000 Hz) nicht zu überschreiten, da nur dann eine schrittgenaue Positionierung gewährleistet ist. Wenn größere Wege zurückzulegen sind, kann – falls nötig – die Geschwindigkeit in einer definierten Beschleunigungsphase (mit einer dazu korrespondierenden Bremsphase) bei fast allen Motoren auf das fünf- bis zehnfache erhöht werden.

Stromabsenkung: Bei Voll- und Halbschrittbetrieb ist es sinnvoll, im Ruhezustand den Spulenstrom auf einen niedrigeren *Haltestrom* abzusenken, die mei-

sten Steuerungen sehen einen solchen Betrieb vor (zusätzliche Steuerleitung). Dies führt zu einer deutlich geringeren thermischen Belastung des Motors und vor allem auch der Umgebung.

2.4 Servos

Vor allem für Anwendungen in der Fernsteuerung (Modellbau, mechanisches Spielzeug, Robotik) wurden kompakte Stellmotoren entwickelt, die mit einigermaßen standardisierten Signalen angesteuert werden können. Diese sogenannten *Servos* können auch in Experimenten überall dort eingesetzt werden, wo einfache Verstellaufgaben automatisiert werden sollen (Blenden, Shutter, Klappspiegel). Servos bestehen aus kleinen leistungsfähigen Motoren mit einem Untersetzungsgetriebe, über das eine Welle oder Scheibe am Ausgang gedreht wird. Deren Winkelstellung wird mit einem Drehwiderstand gemessen und dem Ansteuersignal entsprechend eingestellt (geregelt). Als Winkelverstellbereich ist etwa eine halbe Umdrehung üblich. Das Ansteuersignal besteht aus Impulsen mit einer festen Folgefrequenz (oft 50 Hz) und variabler Länge (z. B. 1...2 ms). Die Pulslänge legt die Winkelposition innerhalb des Verstellbereichs fest (Abbildung 30). Das Steuersignal braucht nur kurzzeitig angelegt zu werden, der Servo behält danach die vorgewählte Position bei.

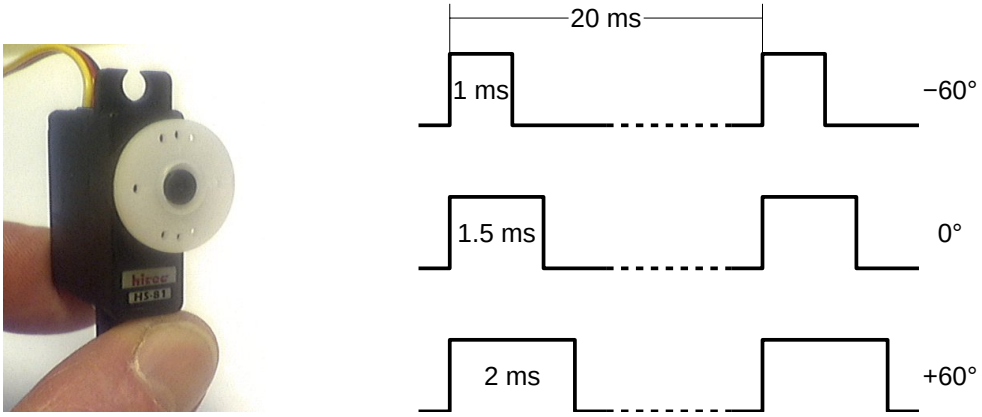


Abbildung 30: Links ein typischer Servo aus dem Fernsteuerungsbereich. Rechts die Ansteuersignale (TTL-kompatibel zwischen ≈0 V und > 3 V wechselnd) für die angegebenen Winkelpositionen.

2.5 Piezostellelemente

Mechanische Bewegungen im Mikro- bis Nanometerbereich lassen sich mit Piezoaktoren ausführen. Ihre Funktion beruht auf dem piezoelektrischen Effekt (genauer dessen Umkehrung): Polare Festkörper reagieren auf ein elektrisches Feld mit Längenänderung oder Scherverformung. Typische Bauformen von drei-

dimensionalen Piezoverstellern, wie sie beispielsweise bei der Rastersondenmikroskopie eingesetzt werden, zeigt Abbildung 31.

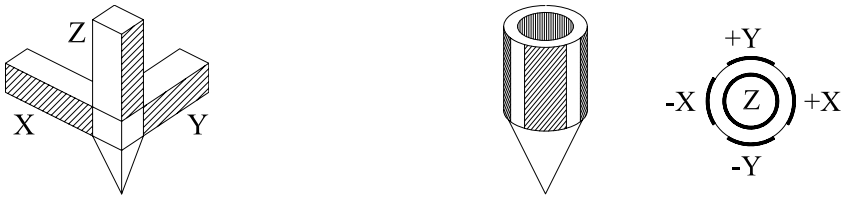


Abbildung 31: Bauformen von dreidimensionalen Piezoverstellern: Dreibein (links) und Biegeröhrchen (rechts, ganz rechts Schnitt durch die Elektrodenanordnung). Beim Biegeröhrchen bewirken symmetrische Spannungsänderungen an den X- oder Y-Elektroden eine Verbiegung, eine Spannungsänderung an der Z-Elektrode eine Längenänderung.

Mit einfachen keramischen Piezostellelementen können Bewegungen im Mikrometerbereich mit Auflösungen im Nanometerbereich erreicht werden. Die notwendigen Betriebsspannungen liegen bei einigen hundert Volt; eine direkte Ansteuerung vom Rechner ist über eine Kombination von D/A-Wandler und Spannungsverstärker möglich. Bei sehr genauen Anwendungen sollte die thermische Ausdehnung von Piezokeramik und Halterung durch eine geeignete Lageregelung kompensiert werden (Regelung auf konstanten Tunnelstrom bei der Rastertunnelmikroskopie, auf maximales Signal bei piezobetriebenen Fabry-Perot-Interferometern).

2.6 Entstörung

Beim Schalten größerer elektrischer Leistungen, aber auch beim Betrieb von Schrittmotoren[8] können elektromagnetische Störfelder auftreten, die sinnvolle Messungen im Experiment erschweren, manchmal verhindern. Solche Störungen sollten schon an der Störquelle soweit wie möglich reduziert werden. Die dazu nötigen Maßnahmen hängen von der Art der Störung ab, diese sollte zunächst analysiert werden. Eine unvollständige Auflistung möglicher Entstörmaßnahmen:

Kurze elektrische Verbindungsleitungen: Trivial, aber immer wieder vergessen, je weniger (Sende- oder Empfangs-) Antenne, umso besser.

Erdung: Wichtig, kann aber auch stören, wenn Erdschleifen auftreten; oft hilft dann, alle Erdleitungen sternförmig zusammenzuführen.

[8] Einfache Schrittmotorsteuergeräte stellen den Motorstrom durch eine Puls-Pausen-Taktung der Betriebsspannung ein. Da diese mit deutlich höherer Frequenz als der maximalen Schrittfrequenz erfolgen muss, sind relativ steile Schaltflanken die Folge.

Abschirmung: Abgeschirmte Kabel und Metallgehäuse gegen elektrische, weichmagnetische Materialien hoher Permeabilität (sehr aufwendig) gegen magnetische Felder.

Symmetrisierung: Ferritringe um Steuerkabel sorgen für eine Hochfrequenzsymmetrisierung und erniedrigen deutlich die HF-Abstrahlung ($\rightsquigarrow$ Standard bei hochwertigen Monitor- oder SCSI-Kabeln).

Potentialtrennung: Eine einfache Trennung durch Optokoppler vermeidet Probleme mit Erdschleifen und mit Potentialunterschieden zwischen Geräten.

Optische Verbindungen: Eine Signalübertragung via Lichtleiter ist zwar etwas aufwendig, aber an Störsicherheit kaum zu überbieten.

3 Signalverarbeitung

Detektoren und Sensoren wandeln physikalische Messgrößen in korrespondierende elektrische Größen wie Strom, Spannung, Ladung oder Widerstand um. Die Signalverarbeitungselektronik muss in ihrer Eingangsschaltung gezielt für diese spezifische elektrische Größe ausgelegt sein.

> Bei der Signalverarbeitung ist zunächst zu überlegen, in welcher Signaleigenschaft sich die gesuchte physikalische Messgröße etabliert.

Analoge elektrische Größen wandelt man zunächst in eine dazu proportionale Spannung, Ereignisgrößen in geeignete einheitliche Zählimpulse. Einige der dazu verwendeten Schaltungsprinzipien werden nachfolgend anhand von vereinfachten Operationsverstärkerschaltungen kurz skizziert.

3.1 Der ideale Operationsverstärker

In einem Operationsverstärker sind eine Reihe von Transistorverstärkerstufen zusammengefasst, die in der Regel gleichstromgekoppelt sind. Als Eingangsstufe wird dabei meist eine Differenzverstärkerschaltung verwendet. Verstärkt wird mithin die Spannungsdifferenz zwischen den beiden Eingängen — dem nichtinvertierenden (‘+’) und dem invertierenden (‘−’) Eingang. Abbildung 32 zeigt die gebräuchlichen Schaltsymbole für Operationsverstärker.

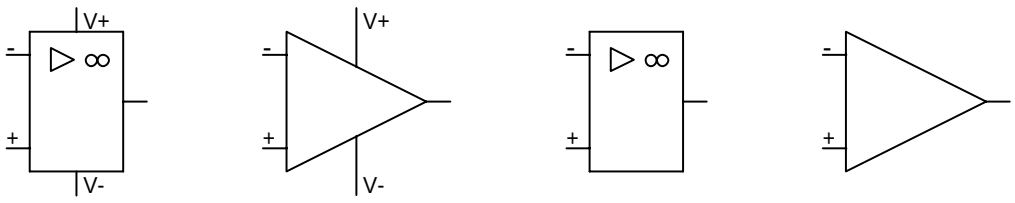


Abbildung 32: Schaltsymbole für Operationsverstärker. Links das Schaltzeichen nach DIN 40900, daneben das international gebräuchliche Symbol. Rechts vereinfachte Darstellungen ohne die Spannungsversorgungsanschlüsse.

Bei der Berechnung der in den folgenden Abschnitten diskutierten Schaltungen gehen wir von idealisierten Verhältnissen aus, dem *idealen Operationsverstärker*. Dessen Eigenschaften sind in Tabelle 3 zusammengestellt, zum Vergleich die typischen Daten realer Operationsverstärker (eine vertiefte Einführung in die Halbleiterschaltungstechnik mit detaillierter Diskussion der Eigenschaften typischer Operationsverstärker und ihrer Schaltungsdimensionierung findet sich z. B. in [14] oder [15], eine Übersicht finden Sie auch bei Wikipedia [16].).

Das Verhalten einer idealen Operationsverstärkerschaltung wird durch das Gegenkopplungsnetzwerk bestimmt (ein allgemeiner Fall ist beispielsweise das Netzwerk der Abbildung 35). Die Schaltungsberechnung folgt einfachen Bedingungen: Die Ausgangsspannung U_A stellt sich so ein, dass die Spannungsdifferenz

Verstärkereigenschaft	Ideal	Real
Differenzverstärkung (dB)	∞	90 . . . 120
Gleichtaktverstärkung	0	0.1 . . . 1
Gleichtaktunterdrückung (dB)	∞	100 . . . 120
Eingangswiderstand (MΩ)	∞	1 . . . 1000
Ausgangswiderstand (Ω)	0	10 . . . 1000
Untere Grenzfrequenz (Hz)	0	0
Obere Grenzfrequenz (MHz)	∞	1 . . . 100
Offsetspannung (mV)	0	0.01 . . . 1
Eingangsruhestrom (μA)	0	0.01 . . . 1
Rauschen am Ausgang (μV)	0	1 . . . 10

Tabelle 3: Operationsverstärker, ideale Eigenschaften im Vergleich mit realen.

zwischen invertierendem und nichtinvertierendem Eingang verschwindet. Unter der Nebenbedingung, dass kein Eingangsstrom am Operationsverstärker fließt, kann man U_A aus den übrigen Größen der Schaltung berechnen. Die Widerstände des Gegenkopplungsnetzwerks sind oft reell, im allgemeinen Fall komplex. Sie können auch nichtlinear oder durch umfangreichere Netzwerke ersetzt sein.

3.2 Strom

Viele Detektoren und Sensoren liefern ein Stromsignal, das der physikalischen Messgröße proportional ist (Anodenstrom bei Photomultiplieren, Sperrstrom bei Photodioden). Dieser Strom wird mit einem als Strom-Spannungs-Wandler beschalteten Operationsverstärker[9] (Abbildung 33) in ein Spannungssignal $U = -R \cdot I$ umgesetzt.

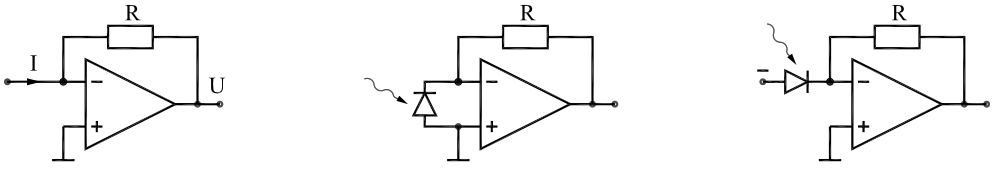


Abbildung 33: Strom-Spannungs-Wandler: Links die Grundschaltung, in der Mitte mit einer Photodiode im Kurzschlussbetrieb, rechts im Sperrbetrieb.

Bei Photodioden sind die beiden skizzierten Schaltungen gebräuchlich, welche man im Einzelfall wählt, hängt von den experimentellen Anforderungen ab:

[9] Verschiedene Hersteller bieten Bausteine an, bei denen Photodiode und Operationsverstärker im gleichen Gehäuse integriert sind.

Im *Kurzschlussbetrieb* fließt kein Dunkelstrom, interessant für den rauscharmen Nachweis sehr geringer Lichtintensitäten.

Im *Sperrbetrieb* ist die Diodenkapazität geringer, das Feld in der Raumladungszone größer als im Kurzschlussbetrieb, ideal für höhere Signalbandbreiten; allerdings fließt auch ohne Beleuchtung ein – wenn auch geringer – Dunkelstrom.

3.3 Spannung

Zur Spannungsverstärkung wird der Operationsverstärker *nichtinvertierend* betrieben (Abbildung 34). Diese Betriebsart hat den Vorteil eines sehr hohen Eingangswiderstands, belastet somit die Signalquelle nur minimal. Bei sehr hochohmigen Quellen und/oder kleinen Signalpegeln sollte man Operationsverstärker mit FET-Eingang verwenden, der Eingangswiderstand ist deutlich größer, das Eingangsrauschen deutlich kleiner als bei konventionellen mit bipolarem Eingang.

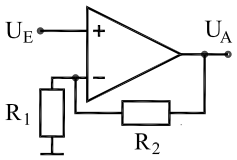


Abbildung 34: Spannungsverstärker.

Die Verstärkung V wird wie üblich aus den idealen Eigenschaften des Operationsverstärkers (Eingangsspannungdifferenz und Eingangsstrom gleich null) berechnet:

$$V = \frac{U_\mathrm{A}}{U_\mathrm{E}} = 1 + \frac{R_2}{R_1} \tag{3.1}$$

Ist nur eine Impedanzwandlung, keine Signalverstärkung notwendig, erreicht man dies durch $R_2 = 0$ (Spannungsfolger).

3.4 Spannungsdifferenz

Zur Messung von Spannungsdifferenzen (beispielsweise aus einer Messbrücke – vgl. Kapitel 1.5 – oder bei anderen symmetrischen Signalquellen) wird der Operationsverstärker als *Differenzverstärker* betrieben (Abbildung 35). Diese Betriebsart ist eine Kombination aus invertierendem und nichtinvertierendem Verstärker. Bei hochohmigen Quellen können jeweils noch Spannungsfolger vorgeschaltet werden (Abbildung 36 im folgenden Abschitt).

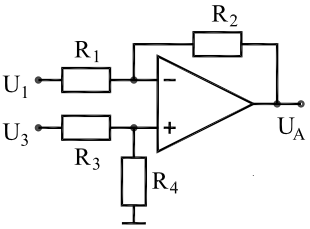


Abbildung 35: Differenzverstärker.

Zur Berechnung der Ausgangsspannung U_A geht man wie üblich von idealen Verhältnissen aus, kein Eingangsstrom, keine Differenz zwischen den beiden Spannungen U_+ und U_- am Eingang des Verstärkers. Das ergibt für den oberen Zweig

$$U_- = U_1 + (U_\mathrm{A} - U_1)\frac{R_1}{R_1 + R_2} = U_\mathrm{A}\frac{R_1}{R_1 + R_2} + U_1\frac{R_2}{R_1 + R_2} \tag{3.2}$$

und für den unteren Zweig

$$U_+ = U_3\frac{R_4}{R_3 + R_4}\ . \tag{3.3}$$

Damit wird

$$U_{\mathrm{A}} = \frac{R_1 + R_2}{R_1} \left(U_3 \frac{R_4}{R_3 + R_4} - U_1 \frac{R_2}{R_1 + R_2} \right) \ . \tag{3.4}$$

Um die Differenzspannung $U_3 - U_1$ zu messen, müssen die beiden Faktoren $R_4/(R_3 + R_4)$ und $R_2/(R_1 + R_2)$ gleich groß sein (oft wählt man $R_4 = R_2$ und $R_3 = R_1$). Somit

$$U_{\mathrm{A}} = \frac{R_2}{R_1} \left(U_3 - U_1 \right) \ , \tag{3.5}$$

die Differenzspannung wird um den Faktor $V = R_2/R_1$ verstärkt.

3.5 Widerstand

Sind die Widerstandssensoren in einer Brückenschaltung angeordnet (vgl. Kapitel 1.5), so verwendet man beispielsweise die im Vorkapitel beschriebene Schaltung zur Differenzspannungsmessung mit geeignet eingestellter Verstärkung V. Bei hochohmiger Brücke sollten Spannungsfolger vorgeschaltet werden (Abbildung 36).

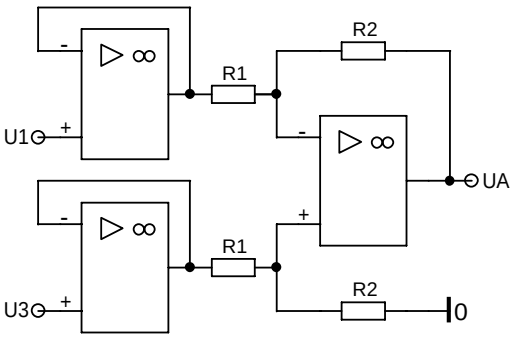


Abbildung 36: Differenzverstärker mit vorgeschalteten Spannungsfolgern zur hochohmigen Messung von Spannungsdifferenzen (praktisch keine Belastung der Quelle).

Zur Messung von Einzelwiderständen (vgl. Abschnitt 1.1.1 zur Temperaturmessung) wird die Widerstandsmessung mit Hilfe einer Konstantstromquelle auf eine Spannungsmessung zurückgeführt[10]. Um die Messung wenig zu stören, insbesondere auch um wenig Wärme zu produzieren, sollte der Betriebsstrom möglichst niedrig eingestellt werden. Trotzdem kann bei Widerstandsdetektoren eine langsame zeitliche Veränderung (Drift) im Widerstandswert zu Messungenauigkeiten führen. Eine Verbesserung erreicht man in solch einem Fall meist durch eine modulierte Messung: Ein Messparameter wird periodisch verändert (bei optischen Messungen zum Beispiel die Intensität des Anregungslichts), das zugehörige Wechselspannungssignal wird gemessen, die (langsame) Gleichspannungsdrift wird durch einen Hochpass – im einfachsten Fall ein Koppelkondensator – unterdrückt. Noch besser geht's mit dem im Abschnitt 3.8 beschriebenen Lock-In-Verfahren, einer Korrelationsmesstechnik.

[10] Auch möglich, aber weniger üblich ist die Messung des reziproken Widerstands über den Strom (Konstantspannungsquelle).

3.6 Ladung

Bei Teilchendetektoren ist die elektrische Ladung pro nachgewiesenem Teilchen die primär interessierende Größe, zumindest dann, wenn der Detektor zur Energiespektroskopie eingesetzt wird (Szintillationszähler und Halbleiterdetektoren in der Alpha- oder Gamma-Spektroskopie). Die Ladung ist das Integral über einen Stromimpuls, eine ladungsproportionale Spannung erzielt man mit einem als Integrator beschalteten Operationsverstärker (Abbildung 37).

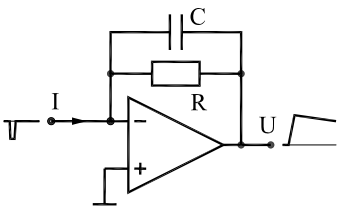


Abbildung 37: Integrator als Ladungs-Spannungs-Wandler.

Für die Ausgangsspannung des Integrators gilt die Differentialgleichung

$$RC \cdot \frac{dU}{dt} + U + R\,I = 0 \tag{3.6}$$

mit der allgemeinen Lösung

$$U = -R\,I + K \exp\left(-\frac{t}{RC}\right). \tag{3.7}$$

Im idealisierten Fall eines Rechteckimpulses der Länge τ, d. h. $I = 0$ für $t < 0$, $I = I_0$ für $0 < t < \tau$ und $I = 0$ für $t > \tau$ ergibt sich für die Spannung am Ende des Impulses

$$U_{\text{peak}} = -R\,I_0 + R\,I_0 \exp\left(-\frac{\tau}{RC}\right) \tag{3.8}$$

und mit der Reihenentwicklung der Exponentialfunktion

$$U_{\text{peak}} = -\frac{I_0\tau}{C}\left(1 - \frac{\tau}{2RC} + \ldots - \ldots\right). \tag{3.9}$$

Gute Ladungsproportionalität (Ladung = $I_0\tau$) ist folglich dann gewährleistet, wenn die Schaltung so ausgelegt wird, dass für typische Signalimpulse $RC \gg \tau$ gilt. Allerdings sollte die Integrationszeitkonstante RC auch nicht größer gewählt werden als für die erforderliche Genauigkeit nötig, da ansonsten *pile-up*-Probleme bei höheren Impulsraten auftreten.

3.7 Ereignis

Sollen Einzelereignisse nur gezählt werden (photon counting u. ä.), ohne dass weitere Signalinformationen auszuwerten sind, werden – gegebenenfalls nach Verstärkung in einem schnellen Verstärker – die Signalimpulse in einem als Schwellwertdiskriminator fungierenden Komparator zu einheitlichen Zählimpulsen geformt (Abbildung 38).

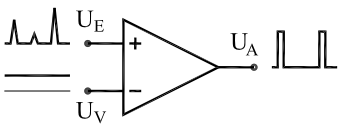


Abbildung 38: Komparator als Diskriminator.

Der Komparator hat neben der Impulsformung die Funktion, Rauschimpulse – insbesondere vom Verstärkerrauschen herrührend – zu unterdrücken (Impulse, die kleiner als U_{V} sind, werden zurückgehalten).

3.8 Lock-In-Verfahren

Die Lock-In-Technik ist eines der wichtigsten Korrelationsmessverfahren: Das zu messende Signal wird geeignet moduliert, das Produkt aus Signalspannung und Modulationsspannung (Referenz) wird über möglichst große Zeiten integriert. Im Prinzip wird dadurch eine sehr schmalbandige Messung bei der Modulationsfrequenz ausgeführt, dadurch werden andere Frequenzen, insbesondere auch niederfrequente Anteile im Rauschen (Drift), effizient unterdrückt. Darüber hinaus kann für die Modulation ein Frequenzbereich gewählt werden, in dem das Rauschen minimal ist. Eine Signalmodulation kann auf unterschiedlichste Art realisiert werden, bei optischen Experimenten (Absorptions- oder Lumineszenzmessungen) wird das Anregungslicht moduliert, bei Spinresonanzexperimenten das Magnetfelds (allerdings nur geringfügig). Das so modulierte, meist stark verrauschte Signal wird *phasenrichtig* gleichgerichtet – wie in Abbildung 39 schematisch dargestellt durch abwechselnde Multiplikation mit +1 oder -1. Diese phasenrichtige Multiplikation wird durch eine mit der Modulation korrelierte Referenzspannung gesteuert. Der Integrator am Ausgang sorgt für die gewünschte Bandbreitenbegrenzung.

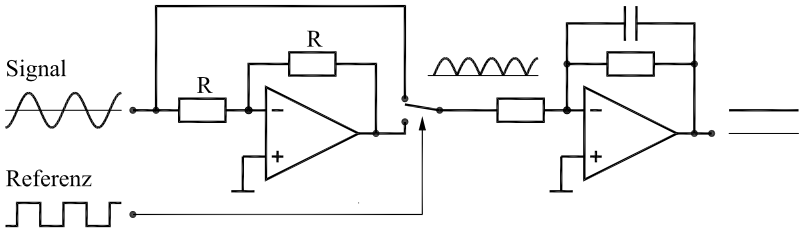


Abbildung 39: Lock-In-Verstärker, Funktionsprinzip.

Ein idealer Lock-In-Verstärker liefert das Integral über das Produkt aus Messspannung U_E und Referenzspannung U_R für $T \to \infty$

$$U_A(t) = \lim_{T\to\infty} \left[\frac{1}{T} \int_{t-T}^{t} U_R(\tau) U_E(\tau) d\tau \right] . \tag{3.10}$$

Aus verschiedenen Gründen ist dieser Idealfall nicht erreichbar. Mit sinusförmiger Referenzspannung (Amplitude 1, Kreisfrequenz ω_R, Phase ϕ_R) wird das Integral

$$U_A(t) = \frac{1}{T} \int_{t-T}^{t} \sin(\omega_R \tau + \phi_R) U_E(\tau) d\tau . \tag{3.11}$$

Die Eingangsspannung U_E möge sich aus verschiedenen Anteilen (Amplituden A, Frequenzen ω_E, Phasen ϕ_E) zusammensetzen, ein beliebiger dieser Anteile liefert

$$U_A(t) = \frac{1}{T} \int_{t-T}^{t} \sin(\omega_R \tau + \phi_R) A \sin(\omega_E \tau + \phi_E) d\tau \tag{3.12}$$

oder mit $\sin\alpha \sin\beta = \frac{1}{2}\left[\cos(\alpha-\beta) - \cos(\alpha+\beta)\right]$

$$U_A(t) = \frac{A}{2T}\int_{t-T}^{t} \left[\cos((\omega_R-\omega_E)\tau + \phi_R - \phi_E) - \cos((\omega_R+\omega_E)\tau + \phi_R + \phi_E)\right] d\tau \ . \tag{3.13}$$

Für große T gehen alle Anteile gegen Null außer dem ersten für den Fall $\omega_E = \omega_R$. Dann wird

$$U_A(t) = \frac{A}{2T}\int_{t-T}^{t} \cos(\phi_R - \phi_E) d\tau = \frac{A}{2}\cos(\phi_R - \phi_E) \ . \tag{3.14}$$

Für eine strengere mathematische Behandlung mache man sich klar, dass Gleichung 3.13 das Faltungsintegral einer Rechteckfunktion (Breite T) mit den Kosinusfunktionen darstellt. Fouriertransformiert wird daraus das Produkt aus der Fouriertransformierten des Rechtecks und der Fouriertransformierten der Kosinusfunktionen. Erstere ergibt $\sin(\omega T/2)/(\omega T/2)$, mithin näherungsweise einen Bandpass mit der ungefähren Breite $\Delta\omega = 1/T$. Die Fouriertransformierten der Kosinusfunktionen sind die zugehörigen *Spektrallinien*. Das Produkt filtert somit aus dem gesamten Signalspektrum ein schmales Spektralband der Breite $1/T$ um die Mittenfrequenz ω_R heraus. Vergrößert man T, so verringert man die Bandbreite und somit den Einfluss von Störungen.

Eine Simulation der Wirkungsweise (durchgeführt mit MATLAB®) ist in Abbildung 40 am Beispiel eines Funktionsverlaufs mit einer Lorentzkurve dargestellt (könnte die Intensität einer Spektrallinie als Funktion der Wellenlänge oder Frequenz sein). Das linke Teilbild zeigt das (ideale) Signal und die beiden bei der Simulation verwendeten Rauschspannungen, im mittleren Teilbild die Messergebnisse bei Anwendung eines einfachen Tiefpassfilters auf die beiden verrauschten Signale, rechts die Messergebnisse bei zusätzlicher Anwendung des Lock-In-Verfahrens.

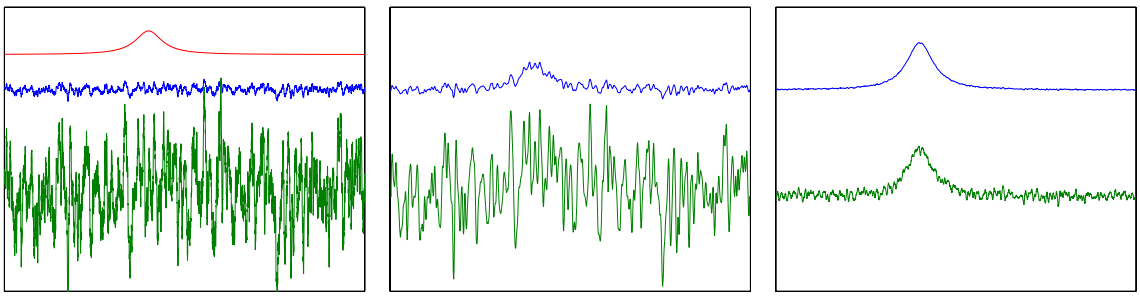


Abbildung 40: Lock-In-Verstärker, Simulation. Links: Ausgangssignale. Mitte: Tiefpassfilterung. Rechts: Lock-In-Verfahren.

Gleichung 3.14 zeigt, dass die Amplitude des Ausgangssignals von der Phasendifferenz zwischen Referenz- und Signalspannung abhängt. Bei der Bedienung solcher *Single Phase Lock-Ins* ist daher stets auf die richtige Einstellung der Phase zu achten. Modernere *Dual Phase Lock-Ins* (oft auch als *Vector Lock-Ins* bezeichnet) vereinfachen dies dadurch, dass auf zwei Kanälen die Produktbildung mit Referenzspannungen unterschiedlicher Phasenlage erfolgt, sinnvollerweise

wählt man als Phasendifferenz $\pi/2$. Damit werden die beiden Ausgangsspannungen

$$U_{A,1} = \frac{A}{2}\cos(\phi_E) \qquad \text{und} \qquad U_{A,2} = \frac{A}{2}\sin(\phi_E)\ . \tag{3.15}$$

Ohne dass die Phase nachgeführt werden muss, errechnen sich Amplitude und Phase aus den beiden Ausgangsspannungen

$$A = 2\sqrt{U_{A,1}^2 + U_{A,2}^2} \qquad \text{und} \qquad \phi_E = \arctan\frac{U_{A,2}}{U_{A,1}}\ . \tag{3.16}$$

Komplexere Lock-In-Verstärker enthalten geeignete Digitale Signalprozessoren (DSP) zur Berechnung.

Da das Lock-In-Prinzip für sehr vielfältige unterschiedliche Messaufgaben eingesetzt werden kann, sind auch viele kommerzielle Geräte verfügbar, die hinsichtlich ihrer technischen Daten wie Empfindlichkeit, Genauigkeit, Frequenzbereich etc., hinsichtlich ihrer Ausstattung und Benutzerfreundlichkeit, aber auch preislich beträchtliche Unterschiede aufweisen. Ein vergleichsweise aufwändiges Geräte, das seit Beginn der 90er Jahre des vergangenen Jahrhunderts hergestellt wird und immer noch von *Stanford Research Systems* gebaut wird, zeigt Abbildung 41.

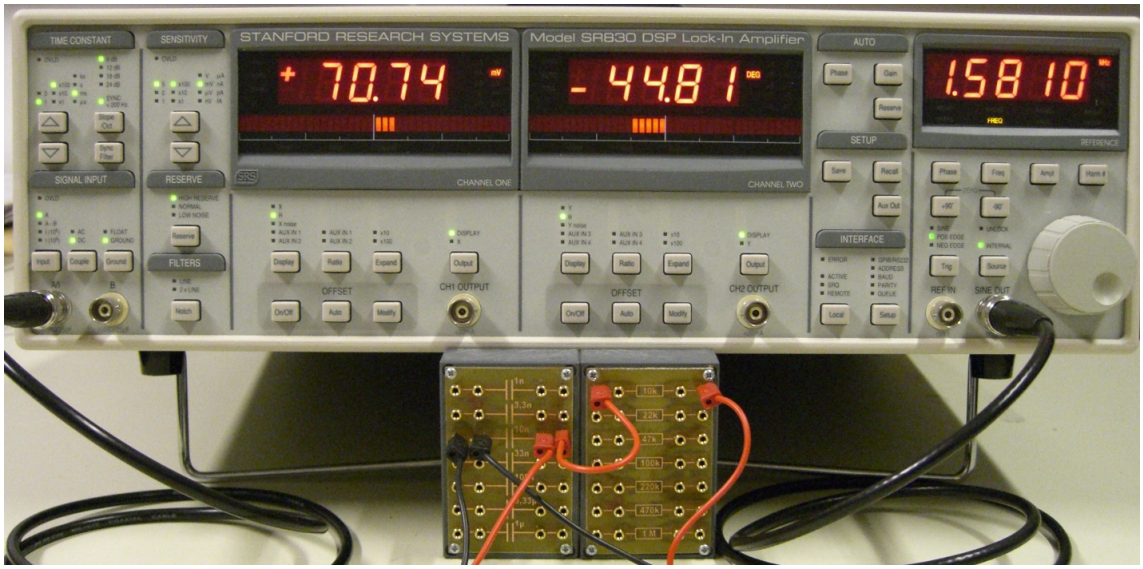


Abbildung 41: Lock-In-Verstärker SR830 von Stanford Research Systems.

Lock-In-Verstärker geben in der Regel die Referenzspannung als Sinussignal aus. Diese Sinusspannung kann man direkt für Messaufgaben verwenden. So lässt sich ein Lock-In beispielsweise als Netzwerk-Analysator für Vierpole einsetzen. Die Referenzspannung wird an den Vierpol-Eingang gelegt, die Amplitude und Phasenlage der Ausgangsspannung wird vom Lock-In analysiert. In Abbildung 41 ist dies für einen sehr einfachen Vierpol, einen Tiefpass aus einer Kombination von 10 kΩ und 10 nF, aufgebaut. Die Frequenz ist so eingestellt, dass

$\omega = 1/(RC)$. Amplitude (71 mV bei 100 mV am Eingang) und Phase (-45 °) zeigen das erwartete Ergebnis.

Der Verlauf einer kompletten Messung, gesteuert von einem MATLAB-Skript, zeigt Abbildung 42.

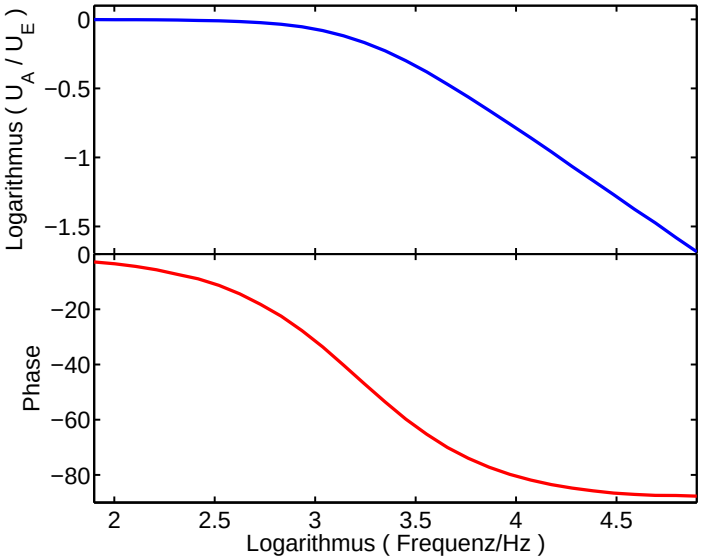


Abbildung 42: Amplituden- und Phasengang an einem Tiefpass. Die Messung wurde mit dem Lock-In-Verstärker der Abbildung 41 durchgeführt.

3.9 Zeit

Bei vielen Messgrößen ist die zeitliche Entwicklung – z. B. nach einer gepulsten Anregung – interessant, da dadurch die Dynamik von Prozessen untersucht werden kann. Geht es dabei um längere Zeiträume (Zerfallszeiten von Hologrammen), kann die Zeitmessung über ein Rechnerprogramm realisiert werden, bei kurzen Zeiten dagegen sind in der Regel spezialisierte Messgeräte erforderlich.

3.9.1 Transientenspeicher

Die Entwicklung schneller Analog-Digital-Wandler und entsprechend schneller Speicher ermöglicht die direkte Erfassung von zeitabhängigen Analogsignalen in Digitalspeicheroszilloskopen oder *Transientenrecordern*. Es können sowohl einmalige Vorgänge gespeichert wie auch – zur Rauschunterdrückung – sich wiederholende Messsignale aufsummiert werden. Die möglichen Zeitauflösungen erreichen den Subnanosekundenbereich.

Ohne Analog-Digital-Wandler, dafür mit extrem schneller *Add-One*-Arithmetik, gibt es die gleiche Technik zur zeitaufgelösten Ereigniszählung (*Multi-Channel Scaler*).

3.9.2 Boxcar-Technik

Eine relativ alte Messtechnik, die bei sich wiederholenden Signalen angewendet werden kann, ist die Boxcar-Technik. Das zeitabhängige Signal wird in einem kurzen Zeitfenster abgetastet, das langsam über das Signal verschoben wird (Abbildung 43). Ein Tiefpassfilter (Integrator) am Ausgang glättet das zunächst kammförmige Abtastsignal zu einer langsam veränderlichen zeitabhängigen Ausgangsspannung (Y). Ein zweiter Ausgang liefert eine Spannung, die der Zeitverschiebung der Abtastung proportional ist (X).

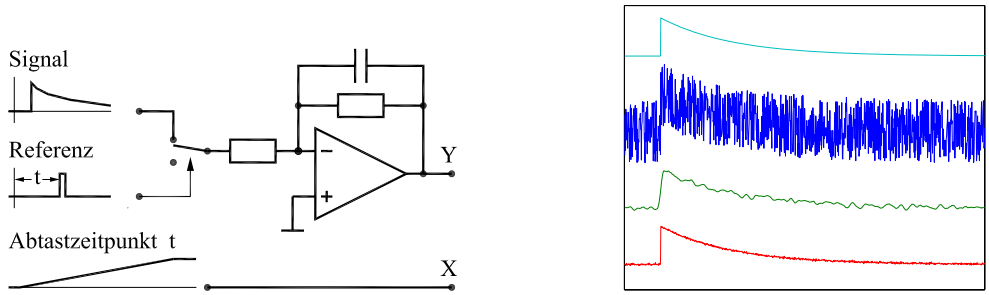


Abbildung 43: Boxcar-Verfahren, links Funktionsschema, rechts Simulation.

Die Simulation des Verfahrens (wieder durchgeführt mit MATLAB®) zeigt die Wirkung – rechtes Teilbild der Abbildung 43: Oben unverrauschtes und verrauschtes Idealsignal, darunter das verrauschte Signal nach Boxcar-Abtastung und -Integration. Ganz unten zum Vergleich dasselbe Signal mit einem summierenden Transientendigitalisierer integriert (bei gleicher Messzeit und hier in der Simulation sehr viel besserer Zeitauflösung ist das Rauschen deutlich geringer als bei der Boxcar-Technik).

Abgesehen von der Zeitmessung kann das Boxcar-Verfahren – dann mit festem Abtastzeitpunkt – als Korrelationsmesstechnik bei gepulst angeregten Experimenten (nichtlineare Optik mit Pulslasern) eingesetzt werden. Gemessen wird nur während der Pulsanregung, das Rauschen dazwischen wird unterdrückt. Digital, d. h. in Verbindung mit Ereigniszählung (photon counting), ist dies derzeit eines der empfindlichsten Messverfahren.

3.9.3 Zeit-Impulshöhen-Wandlung

Die zeitliche Verzögerung seltener Zählereignisse gegenüber einem Anregungsimpuls kann durch Zeit-Impulshöhen-Wandlung mit relativ guter Zeitauflösung bestimmt werden. Bei dieser Technik wird eine analoge Größe, die Zeitdifferenz zwischen zwei Impulsen, in eine dazu proportionale andere, einen Spannungswert, gewandelt, indem während der zu messenden Zeit eine Kapazität mit konstantem Strom aufgeladen wird. Der dabei erreichte Spannungswert ist ein Maß für die Zeitdifferenz. Eine Auftragung der Häufigkeitsverteilung der gemessenen Zeiten ($\rightsquigarrow$ A/D-gewandelte Spannungswerte) ergibt den Zeitverlauf des durch die

Zählereignisse repräsentierten Signals. Da nach einem Anregungsimpuls das jeweils erste Stop-Ereignis zum Tragen kommt, eignet sich die Methode nur für seltene Ereignisse, da ansonsten kurze Zeiten überbewertet werden.

3.10 Kabel

Ergänzend ein paar Sätze zu typischen Kabeln im Bereich der Messdatenverarbeitung. Signale können symmetrisch – massefrei – oder asymmetrisch – massebezogen – übertragen werden. Wofür man sich entscheidet, hängt von der Art der Signalquelle bzw. von der Art des angeschlossenen Verbrauchers ab, darüber hinaus vom Schnittstellentyp, von der Signalhöhe, von möglichen Störungen (vgl. Kapitel 2.6), von der Leitungslänge und verschiedenen weiteren Nebenbedingungen. Den Aufbau einiger typischer Kabel zeigt Abbildung 44.

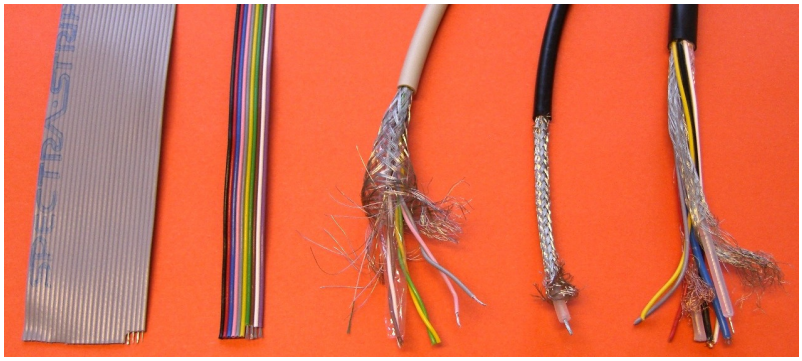


Abbildung 44: Das linke Bild zeigt verschiedene Kabeltypen, links Flachbandleitungen, in der Mitte eine Twisted-Pair-Leitung mit gemeinsamer Abschirmung, rechts daneben ein Koaxialkabel (50Ω-Kabel), ganz rechts ein Verbindungskabel, in dem verschiedene Signal-, Steuer- und Versorgungsleitungen zusammengefasst sind. Rechtes Bild: Bei Koaxialkabeln in Labors leicht zu verwechseln sind Koaxialstecker (BNC) für Hochspannung (links, erkennbar am längeren Isolator) und für Signalspannungen (rechts, kurzer Isolator).

3.10.1 Wellenwiderstand

Bei homogenen (idealerweise unendlich langen) Leitungen kann man einen Wellenwiderstand definieren, unter dem die Leitung einem Signal erscheint. Innenwiderstand der Signalquelle und Wellenwiderstand des Kabels stellen für das Signal einen Spannungsteiler dar, der die Signalhöhe verringert.

Der Wellenwiderstand lässt sich aus dem Ersatzschaltbild eines Kabels berechnen (Abbildung 45).

Geht man von einer Eingangsspannung U und einem Strom I aus, so wird deren

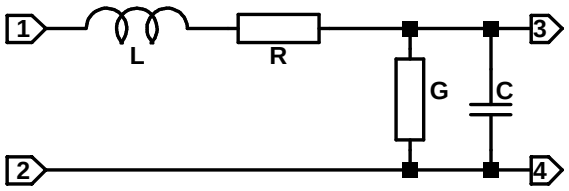


Abbildung 45: Ersatzschaltbild für ein inkrementelles Kabelstück. Der Wellenwiderstand ist durch die differentiellen Größen Induktivität L, Längswiderstand R, Querleitwert G und Querkapazität C bestimmt.

differentielle Änderung durch das Kabelstück der differentiellen Länge dx

$$\frac{dU}{dx} = -I(R + j\omega L) \tag{3.17}$$

$$\frac{dI}{dx} = -U(G + j\omega C)\ . \tag{3.18}$$

Daraus wird eine lineare Differentialgleichung zweiter Ordnung für U oder I, hier für U

$$\frac{d^2U}{dx^2} = (R + j\omega L)(G + j\omega C)U\ . \tag{3.19}$$

Deren Lösung ist

$$U = k\exp(ax) \qquad \text{mit} \qquad a = \pm\sqrt{(R + j\omega L)(G + j\omega C)}\ . \tag{3.20}$$

Wir nehmen an, dass die Spannung U links angelegt wird und die x-Koordinate nach rechts läuft, dann macht nur das negative Vorzeichen von a Sinn. Mit Gleichung 3.17 lässt sich dann ein komplexer Widerstand Z definieren

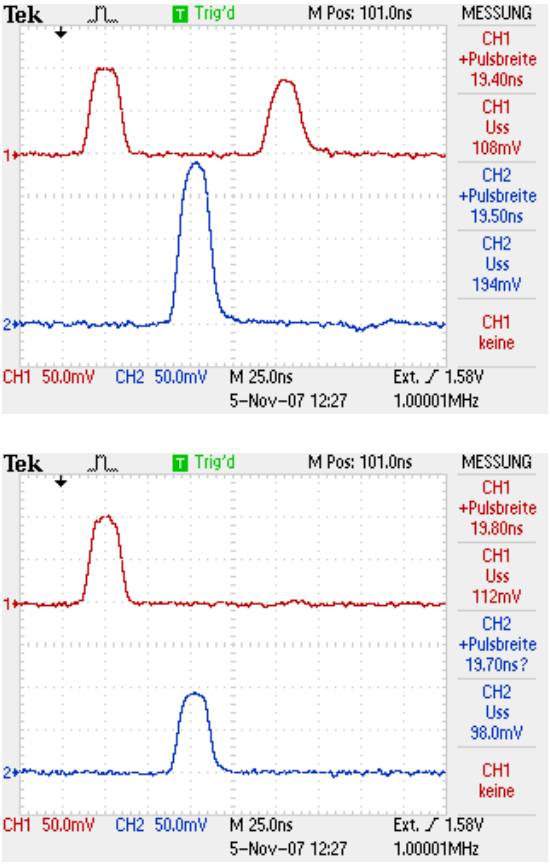
$$Z = \frac{U}{I} = \sqrt{\frac{R + j\omega L}{G + j\omega C}}\ . \tag{3.21}$$

Bei Gleichspannung ($\omega = 0$) ist dieser Widerstand reell, bei niedrigen Frequenzen können meist G und L vernachlässigt werden. Bei hohen Frequenzen sind R und G zu vernachlässigen, die Leitung ist durch L und C definiert und der Widerstand wird somit wieder reell. Typische Werte liegen zwischen 10 und 300 Ω je nach Bauart der Leitung. In der Messtechnik oft verwendete Koaxialkabel weisen einen Wellenwiderstand von 50 Ω auf, Antennenkabel liegen häufig bei 60 Ω.

3.10.2 Reflexionen, Signalgeschwindigkeit

Der oben hergeleitete Wellenwiderstand setzt unendlich lange Kabel voraus. Man kann zeigen, dass sich endlich lange Kabel ebenso ideal verhalten, wenn man sie am Ende mit ihrem Wellenwiderstand abschließt. Tut man das nicht, so werden Signale am Kabelende reflektiert. Dies ist dann störend, wenn man mit schnellen Impulsen oder hohen Frequenzen arbeitet. Was 'schnell' und 'hoch' hier bedeutet, ist von der Laufzeit und damit von der Leitungslänge und der Signalgeschwindigkeit abhängig. Dies sei am Beispiel eines kurzen Impulses (20 ns) auf

einem 10 m langen Koaxialkabel erläutert. Abbildung 46 zeigt die Oszilloskopbilder für drei Fälle.



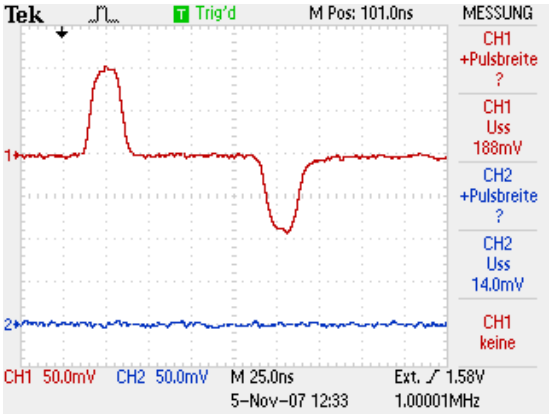


Abbildung 46: Spannungsimpulse auf einem Koaxialkabel, Kabellänge 10 m. Der obere Oszilloskop-Kanal zeigt jeweils die Situation am Kabelanfang, der untere am Kabelende für drei typische Fälle. Links oben offenes Kabelende, rechts oben Kabelende kurzgeschlossen, links unten Kabelabschluss mit dem Wellenwiderstand.

Wird das Kabel ohne jeglichen Abschlusswiderstand – offen – betrieben, so wird das Signal am Kabelende gleichphasig reflektiert. Einlaufendes Signal und reflektiertes sind am Kabelanfang zu sehen, das Signal am Kabelende ist durch die Reflexion in der Amplitude verdoppelt (Teilbild links oben). Wird das Kabel am Ende kurzgeschlossen, so wird das Signal am Kabelende gegenphasig reflektiert. Einlaufendes und gegenphasig reflektiertes sind wieder am Kabelanfang zu sehen, am Kabelende wegen des Kurzschlusses kein Signal (Teilbild rechts oben). Bei Abschluss mit dem Wellenwiderstand des Kabels (hier 50 Ω) treten kein störenden Reflexionen auf, die Impulshöhe am Kabelende entspricht der am Kabelanfang (Teilbild links unten).

Aus der Kabellänge und der am Oszilloskop gemessenen Signallaufzeit kann die Signalgeschwindigkeit berechnet werden. Die etwa 53 ns für 10 m ergeben eine Geschwindigkeit von $1.9 \cdot 10^8$ m/s, also etwa zwei Drittel der Vakuumlichtgeschwindigkeit. Koaxialkabel eignen sich mithin auch sehr gut dazu, Signale definiert zu verzögern oder zeitlich aneinander anzupassen, soweit es um Zeiten zwischen etwa 1 und 100 ns geht. 5 ns Laufzeit pro 1 m Koaxialkabel sind als Faustformel meist hinreichend genau.

4 D/A- und A/D-Wandler

Experimentelle Steuergrößen (Heizstrom, elektrisches Feld, ...), die vom Rechner vorgegeben werden sollen, müssen dazu meist zuerst in Analogwerte umgesetzt werden. Diese Digital/Analog-Wandlung erfolgt in digital ansteuerbaren Peripheriegeräten oder auf PC-Karten durch spezielle Bausteine – D/A-Wandler. Experimentelle Messwerte andererseits (Spannung, Widerstand, Temperatur, Lichtintensität, Druck, ...) liegen am Experiment im allgemeinen analog vor und müssen – gegebenenfalls nach Umwandlung in elektrische Größen – zur Bearbeitung durch den Rechner in Digitalwerte umgeformt werden. Auch diese Analog/Digital-Wandlung erfolgt – in externen Messgeräten wie Digitalvoltmetern oder auf PC-Karten – durch spezielle Bausteine, A/D-Wandler.

> Die Grenze zwischen Digital und Analog ist auch eine Grenze zwischen scheinbarer Exaktheit und realer Ungenauigkeit.

Im folgenden sollen die Verfahren, nach denen die Wandlerbausteine arbeiten, etwas näher betrachtet werden.

4.1 Digital/Analog-Wandler

Das praktisch ausschließlich verwendete technische Konzept für Digital/Analog-Wandler (D/A-Wandler, D/A-Converter, DAC) ist das der Stromsummation: Am Eingang eines Operationsverstärkers werden Ströme aufsummiert, deren Größe der Wertigkeit der einzelnen Bits in dem umzuwandelnden Digitalwert entspricht. Die schematische Schaltung zeigt Abbildung 47.

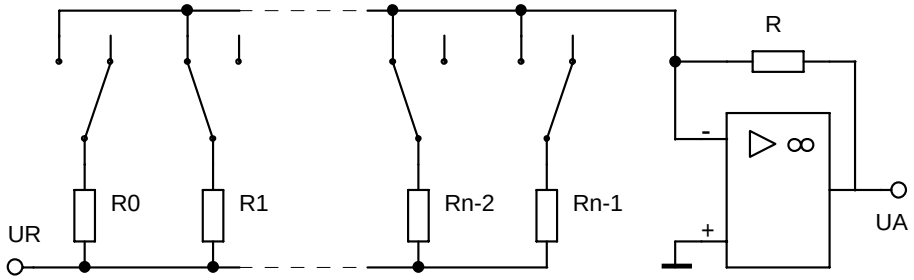


Abbildung 47: Prinzipschaltung eines D/A-Wandlers: Stromsummation.

Beim idealen Operationsverstärker sind Differenzeingangsspannung und Eingangsströme gleich null, daraus ergibt sich für die Ausgangsspannung U_A:

$$U_\mathrm{A} = -U_\mathrm{R} \cdot R \cdot \sum_{i=0}^{n-1} S_i/R_i \tag{4.1}$$

$S_i = 1$, wenn der entsprechende Schalter geschlossen, $S_i = 0$, wenn der entsprechende Schalter geöffnet ist. Die Schalter (meist als Feldeffekttransistoren

realisiert) werden durch die einzelnen Bits des umzusetzenden Digitalwerts angesteuert, die Widerstände R_i haben die Widerstandswerte $R_i = R_0 \cdot 2^{-i}$. Damit erhält man eine Ausgangsspannung, die proportional zum Digitalwert ist.

Die Schaltung Abb. 47 hat den großen Nachteil, dass man sehr unterschiedliche Widerstandswerte benötigt. In der Praxis verwendet man daher eine modifizierte Schaltung, das R/2R-Netzwerk. Das Funktionsprinzip wird aus Abb. 48 deutlich, wenn man sich klarmacht, dass an den Punkten $i = 0, 1, \ldots, n-1$ jeweils die Spannung $U_\mathrm{R} \cdot 2^{-i}$ anliegt.

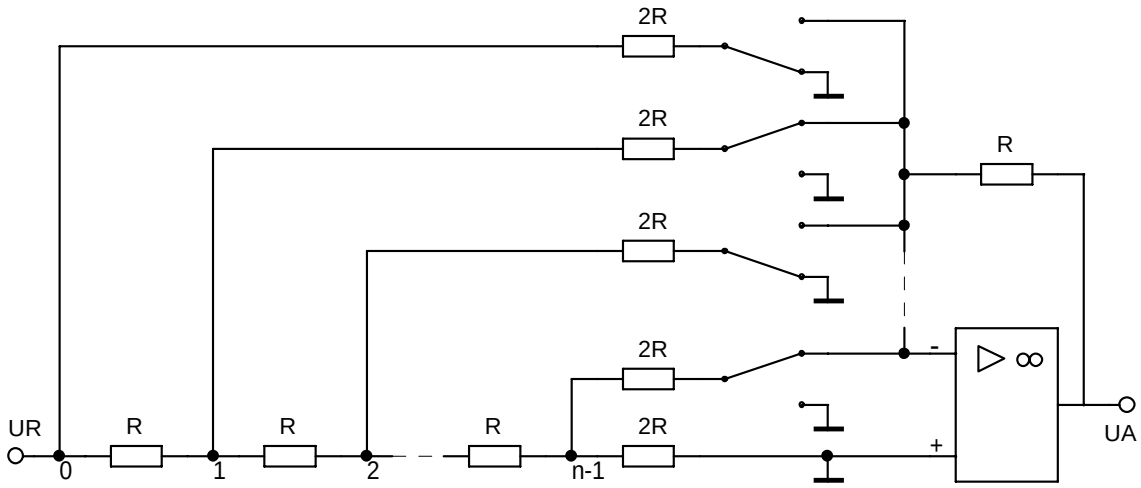


Abbildung 48: Digital/Analog-Wandler mit R/2R-Netzwerk

Die Güte eines D/A-Wandlers und damit seine Verwendungsmöglichkeit für eine bestimmte Steuerungsaufgabe lässt sich durch einige Kenngrößen charakterisieren:

- Die **Auflösung** gibt die Bitbreite des gewandelten Digitalwertes an. Sie liegt im allgemeinen zwischen 8 und 16 Bit, durch die neueren Entwicklungen im Audiobereich (CD-Player) sind zur Zeit auch D/A-Wandler mit hoher Auflösung (14 Bit, 16 Bit) relativ preisgünstig. Die Auflösung bestimmt die minimale Schrittweite, mit der eine Steuergröße vom Rechner vorgegeben werden kann.

- **Genauigkeit** und **Linearität** sowie deren Temperaturabhängigkeit geben an, wie gut die vorhandene (digitale) Auflösung in die Realität der analogen Welt umgesetzt wird. Diese Größen sind von den Toleranzen und dem Temperatur- und Alterungsverhalten des internen Widerstandsnetzwerks abhängig. Hohe Anforderungen hier bedingen hohen Herstellungsaufwand.

- Die **Geschwindigkeit**, oft angegeben als Anstiegszeit (10%–90%) für einen Ausgangsspannungssprung von der minimalen zur maximalen Ausgangsspannung, kann als Kriterium im allgemeinen außer acht gelassen werden,

da die Datenausgabe vom PC fast immer langsamer ist und da die typischen Zeitkonstanten der zu steuernden Geräte (Heizung etc.) meist sehr viel größer sind.

Die Ansteuerung von D/A-Wandlern durch den Rechner hängt davon ab, wie der Digitalteil des Wandlers ausgeführt ist. Im einfachsten Fall sind die 'Schalter' durch TTL-kompatible Logiksignale direkt zugänglich, dann muss ein Baustein vorgeschaltet werden, der den Digitalwert zwischenspeichert und auch nach dem Ausgabebefehl statisch am D/A-Wandler anliegen lässt (z. B. ein paralleler E/A-Baustein 8255).

Ist ein internes Speicherregister im D/A-Wandler vorhanden, wird der Digitalwert entweder parallel oder getaktet seriell übergeben. Parallele D/A-Wandler sind ähnlich wie parallele E/A-Bausteine zu sehen: Der Digitalwert wird – in Bytes aufgeteilt – durch Ausgabebefehle übergeben, ein Trigger-Befehl veranlasst die interne Übernahme des kompletten Werts.

Bei seriellen D/A-Wandlern wird der Digitalwert durch einen Daten- und einen Takteingang seriell (bitweise) in ein Schieberegister eingetaktet, die interne Übernahme wird durch einen weiteren Eingang getriggert. Drei Leitungen reichen also zum Anschluss – z. B. an ein Parallelport im PC – aus. Ein Ansteuerprogramm für diese Art von D/A-Wandlern muss den umzuwandelnden Digitalwert in eine Bitfolge umsetzen und diese zusammen mit richtig gesetztem Taktbit und Übernahmebit z. B. über 3 Leitungen einer parallelen Schnittstelle seriell ausgeben.

4.2 Analog/Digital-Wandler

Im Gegensatz zu den D/A-Wandlern werden unterschiedliche technische Konzepte verwendet, die sich in Auflösung, Geschwindigkeit und Aufwand zum Teil beträchtlich unterscheiden.

4.2.1 Parallel-A/D-Wandler

auch als Flash-A/D-Wandler bezeichnet, vergleichen die anliegende Analogspannung in einer Reihe von 2^n Komparatoren[11] mit 2^n Vergleichsspannungen (Abbildung 49), n ist die Bitbreite des Digitalwertes.

Das Prinzip ist sehr aufwendig (für einen 8-Bit-Wandler benötigt man 256 Komparatoren), allerdings auch sehr schnell. Die schnellsten Flash-Wandler erreichen Wandlungsraten im Gigahertzbereich. Eingesetzt werden solche Wandler in Digitalisierern, mit denen sehr schnelle Vorgänge aufgezeichnet werden sollen (Transientenrecorder, Digitalspeicheroszilloskope). Diese Digitalisierer spei-

[11] Ein Komparator kann als speziell beschalteter Operationsverstärker angesehen werden, der die beiden Eingangsspannungen vergleicht und das Vergleichsresultat in ein Logiksignal umsetzt. $U_+ > U_- \Rightarrow 1$, $U_+ < U_- \Rightarrow 0$.

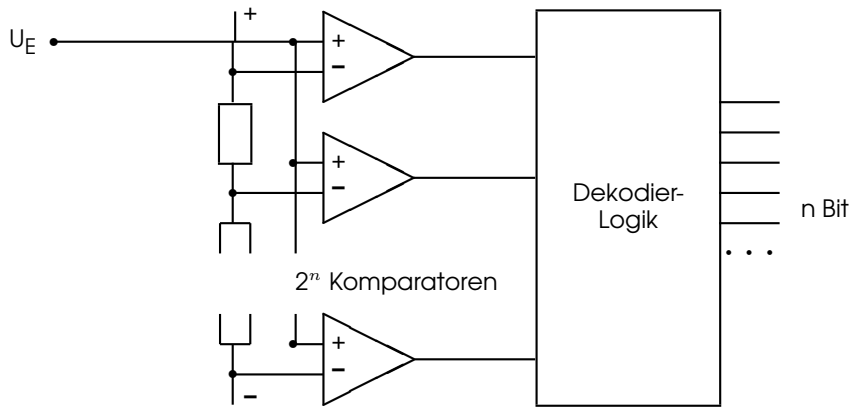


Abbildung 49: Parallel-Analog/Digital-Wandler

chern die Daten zunächst intern in einem entsprechend schnellen Speicher, aus dem sie dann langsamer abgerufen werden können. Im PC sind Flash-Wandler zum Beispiel zur Digitalisierung von Videosignalen sinnvoll einsetzbar (Frame-Grabber-Karten), die Geschwindigkeit des direkten Speicherzugriffs (DMA) über den PCI-Bus ist schnell genug, um Videosequenzen ohne Zwischenspeicherung direkt in den Rechnerspeicher zu transferieren.

4.2.2 Kaskaden-Wandler

Um mit der Geschwindigkeit von Parallel-Wandlern eine höhere Auflösung zu erreichen, ohne gleichzeitig den massiven Aufwand von 2^n parallel arbeitenden Komparatoren treiben zu müssen, wurden Kaskadierungskonzepte für Parallel-Wandler entwickelt. Die Eingangsspannung wird bei diesen Wandlertypen in einer ersten Stufe grob digitalisiert, z. B. mit einer Genauigkeit von 6 Bit. In einem schnellen D/A-Wandler wird aus diesem groben Digitalwert eine Kompensationsspannung generiert, die von der Eingangsspannung subtrahiert wird. Die Differenz wird definiert verstärkt und in einem zweiten Digitalisierungsschritt einem weiteren D/A-Wandler zugeführt. Das Gesamtergebnis wird aus den beiden Teilergebnissen zusammengesetzt.

4.2.3 Integrations- und Zählverfahren

Bei diesem Verfahren wird durch die Messspannung der Strom einer Kondensatoraufladung gesteuert (Operationsverstärker als Integrator geschaltet). Die Zeit, die benötigt wird, um eine bestimmte Aufladespannung zu erreichen, ist umgekehrt proportional zum Ladestrom und damit zur Messgröße. Durch Abzählen der Taktimpulse eines hochgenauen Taktgenerators (quarzstabilisiert) kann diese Zeit und damit die Messgröße sehr genau bestimmt werden.

Eine Verbesserung dieses ‘Single-Slope’-Verfahrens ist das ‘Dual-Slope’-Verfahren,

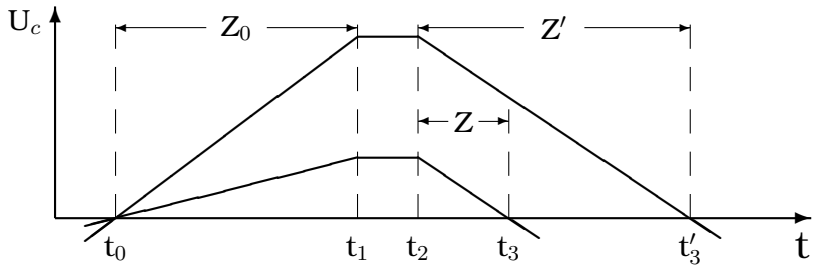


Abbildung 50: Dual-Slope-Verfahren: Zeitlicher Spannungsverlauf an der Integratorkapazität für zwei verschiedene Messspannungen.

dessen Funktionsweise in Abb. 50 dargestellt ist. Die Integration über die Messspannung erfolgt hier für eine fest vorgegebene Zeit $t_1 - t_0$, die durch Abzählen von Z_0 Taktimpulsen gemessen wird. Die nach dieser Zeit erreichte Ladespannung ist proportional zur Messgröße. Zum Zeitpunkt t_1 wird die Messspannung abgeschaltet, zum Zeitpunkt t_2 stattdessen eine hochgenaue Referenzspannung umgekehrten Vorzeichens am Integrator angelegt. Die Zeit bis zur Entladung des Kondensators ($t_3 - t_2$) ist nun – da die steuernde Referenzspannung konstant ist und damit die Steigung der Entladegeraden fest – proportional zur Ladespannung bei t_2. Auch diese Zeit wird durch Zählen von Taktimpulsen gemessen ($Z,\ Z'$). Die Messspannung kann dann auf sehr einfache Weise berechnet werden:

$$U_{\text{mess}} = -U_{\text{ref}} \cdot Z/Z_0 \tag{4.2}$$

Zweckmäßigerweise werden $-U_{\text{ref}}$ und Z_0 so gewählt, dass ihr Quotient der geforderten Auflösung entsprechen (z. B. $-U_{\text{ref}}/Z_0 = 1\ \mu V$), dann ist Z ohne weitere Umrechnung der Zahlenwert der Messspannung.

Der große Vorteil des Dual-Slope-Verfahrens besteht darin, dass Ungenauigkeiten des Taktgebers und des Integrators nur eine geringe Rolle spielen, da sie in Auflade- und Entladevorgang in gleicher Weise eingehen und damit das Messergebnis in erster Näherung nicht verfälschen. Zur Unterdrückung von Störspannungen kann die Aufladezeit $t_1 - t_0$ so festgesetzt werden, dass sie einem ganzzahligen Vielfachen der Periode der Störspannung entspricht. Üblich sind Vielfache von 20 millisec, um 50 Hz-Störungen auszuschalten.

Integrierende A/D-Wandler sind die weitaus genauesten, das Verfahren wird insbesondere bei Digitalmultimetern eingesetzt. Ein kleiner Nachteil sind die relativ langen Integrationszeiten, die keine allzu schnelle Messfolge zulassen.

Eine etwas modifiziertes Verfahren wird bei der Impulshöhenanalyse in Vielkanalanalysatoren benutzt. Eine Kapazität wird auf eine Spannung aufgeladen, die dem Spitzenwert des zu analysierenden Impulses entspricht, die Entladung erfolgt dann mit konstantem Strom. Die Entladezeit wird wie beim Dual-Slope-Verfahren gemessen, der Zahlenwert ist proportional zur Impulshöhe und dient zur Adressierung des Vielkanalanalysatorspeichers. Hierbei sind Taktfrequen-

zen von einigen 100 MHz Stand der Technik, die Entladezeit und damit der notwendige Zeitabstand zum nächsten Impuls beträgt etwa 10 μsec.

4.2.4 Wägeverfahren

Beim Wäge- oder Kompensationsverfahren wird die Messspannung in einem Komparator mit einer Vergleichsspannung verglichen, die durch einen Digital/Analog-Wandler erzeugt wird. Die vereinfachte Schaltung zeigt Abb. 51.

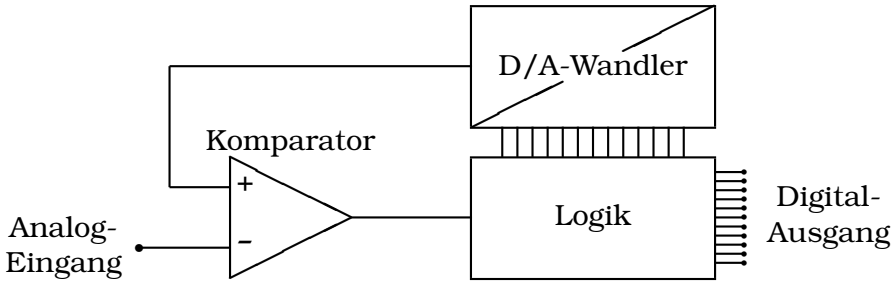


Abbildung 51: Grundschaltung von Analog/Digital-Wandlern, die nach dem Vergleichsprinzip arbeiten.

Im Logikteil wird der Digitalwert der Vergleichsspannung generiert und der Ausgang des Komparators beobachtet. Dieser Teil kann durch Hardware auf dem Baustein oder durch Rechnersoftware realisiert sein. Die Vergleichsspannung kann durch unterschiedliche Strategien variiert werden:

- Bei der einfachen Abzählstrategie wird der Digitalwert, bei 0 beginnend, solange hochgezählt, bis der Komparatorausgang von logisch 0 auf 1 wechselt. Diese Strategie ist die langsamste.
- Beim Nachlaufverfahren wird – abhängig vom Komparatorausgang – vorwärts oder rückwärts gezählt, bis der richtige Wert erreicht ist. Dies Verfahren ist sehr gut geeignet bei langsam veränderlicher Messgröße, in diesem Fall liegt praktisch immer der richtige Digitalwert am Ausgang an.
- Beim Intervallschachtelungsverfahren (sukzessive Approximation) werden, beginnend beim höchstwertigen Bit, alle Bits nacheinander abgetestet und je nach Komparatorreaktion im Digitalwert auf 0 oder 1 gesetzt. Der Messwert wird durch die fortgesetzte Halbierung des Intervalls sehr schnell erreicht. Voraussetzung ist, dass die Messspannung während der Intervallschachtelung einigermaßen konstant bleibt. Dies wird im allgemeinen durch eine vorgesetzte 'Sample-and-Hold'-Schaltung erreicht, die auf ein Startsignal hin den Messwert abtastet (sample) und dann festhält (hold).

Die meisten der auf (internen oder externen) Messkarten verwendeten Analog/Digital-Wandler arbeiten nach diesem letztgenannten Verfahren. Ein Programm für sol-

che Wandler muss zunächst eine A/D-Wandlung starten, sodann eine festgelegte Mindestzeit (Wandlungszeit) oder ein 'Ready'-Signal abwarten, dann den Digitalwert einlesen. Die Wandlungszeit liegt, abhängig vom A/D-Wandler, zwischen 0.1 und 100 μsec. Häufig ist dem A/D-Wandler ein Analogmultiplexer vorgesetzt, der zwischen mehreren Messstellen umschalten kann, dann muss das Programm zuallererst den richtigen Kanal anwählen.

4.2.5 Spannungs-Frequenz-Wandlung

Bei diesem Verfahren wird die Messspannung in eine Wechselspannung oder eine Impulsfolge umgesetzt, deren Frequenz zur Messgröße proportional ist. Dies leisten spezielle ICs, Spannungs/Frequenz-Wandler. Die Spannungsmessung ist damit in eine Frequenzmessung transformiert. Die Frequenz wird mit einem Zähler gemessen, der auf eine feste Zählzeit eingestellt wird, fortlaufend misst und die jeweiligen Messwerte zum Auslesen in einem 'Latch' zwischenspeichert (Ratemeter).

Der Vorteil des Verfahrens liegt u. a. darin, dass es integrierend ist, damit sehr rauschunempfindlich und wenig störanfällig. Allerdings zählt es zu den langsameren A/D-Wandlungsprinzipien, Zählzeiten von 0.01 – 1 sec sind gebräuchlich. Ein weiterer Vorteil – oft wesentlich – ist die einfach durchzuführende Potenzialtrennung zwischen Messstelle und Rechner (nächster Abschnitt).

4.3 Potenzialtrennung

Das Arbeiten mit analogen Größen erfordert ein stabiles und vor allem einheitliches Referenzpotential im gesamten Messsystem ('Messerde'). Die Problematik wird deutlich, wenn man sich klarmacht, dass bei 12 Bit Auflösung und einem Spannungsbereich 0...5 V die einem Bit entsprechende Spannung etwa 1 mV beträgt. Störspannungen in dieser Größenordnung lassen sich in einem größeren System oft nicht verhindern. Ein Ausweg, der Störungen meist abhilft, ist die elektrische Trennung von Mess- bzw. Steuerungssystem und Datenverarbeitung. Einheitliche Bezugspotentiale brauchen dann nur noch in den Teilsystemen vorhanden zu sein.

Informationen zwischen den Teilsystemen werden am einfachsten auf optischem Wege übertragen. Dies kann durch Optokoppler[12] oder – bei höheren Anforderungen, z. B. in einer Hochspannungsumgebung – mit Lichtleiterverbindungen realisiert werden. Für die optische Ankopplung sind insbesondere Bausteine und Geräte geeignet, die über wenige Leitungen angebunden werden können, da ansonsten der Aufwand hoch wird. Man verwendet daher für die analoge Ankopplung in einem gestörten Bereich Spannungs/Frequenz-Wandler, die sehr nahe an der Messstelle angeordnet sind und ihre Messfrequenz optisch an einen

[12] Optokoppler sind Schaltkreise, die eine Lumineszenzdiode und eine Photodiode – elektrisch getrennt, optisch gekoppelt – enthalten. Die Isolationsspannungen liegen im Bereich von einigen hundert bis etwa 2000 Volt.

Zähler im Rechner weitergeben bzw. seriell ansteuerbare D/A-Wandler, die vom Rechner aus optisch angesteuert werden.

4.4 Digitale Regelung

Ein wichtiges Anwendungsgebiet für A/D- und D/A-Wandler ist der Bereich der Regelungstechnik. Das grundlegende Prinzip, nach dem alle Regler – analoge wie digitale – arbeiten, ist das des geschlossenen Regelkreises (Abbildung 52).

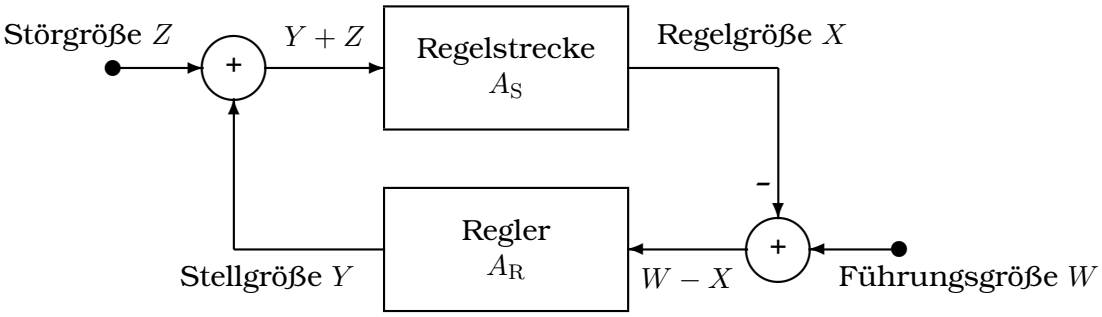


Abbildung 52: Regelkreis

Die **Regelstrecke** ist das Gerät, das geregelt werden soll, z. B. ein Ofen oder Kryostat. Sie wird beschrieben durch ihre Übertragungsfunktion A_S, die im Idealfall eine Konstante ist (lineare Regelstrecke).

Die **Regelgröße** ist der physikalische Parameter, der geregelt werden soll, z. B. die aktuelle Temperatur (Ist-Temperatur, Ist-Wert).

Die **Führungsgröße** ist der Vorgabewert für die Regelgröße (Soll-Temperatur, Soll-Wert).

Der **Regler** ist ein Verstärker im ganz allgemeinen Sinne mit bestimmten Eigenschaften, die durch die Übertragungsfunktion A_R beschrieben werden. Beim **Zwei-Punkt-Regler** ist die Übertragungsfunktion eine Stufenfunktion mit Hysterese (Bimetallthermostat o. ä.), beim **Proportionalregler** eine Konstante, die Verstärkung des Reglers.

Die **Stellgröße** ist die physikalische Größe, mit der die Strecke betrieben wird (Heizleistung, -strom, -spannung).

In der **Störgröße** sind die von außen einwirkenden Störungen zusammengefasst (Schwankungen der Umgebungstemperatur, zusätzliche Wärmezufuhr durch Lichteinstrahlung o. ä.).

Idealisiert gilt bei linearer Regelstrecke für einen Proportionalregler

für die Stellgröße

$$Y = A_{\mathrm{R}} \cdot (W - X), \tag{4.3}$$

für die Regelgröße

$$X = A_{\mathrm{S}} \cdot (Y + Z) \tag{4.4}$$

und damit für die Abhängigkeit der Regelgröße von Führungsgröße und Störung

$$X = \frac{A_{\mathrm{R}} A_{\mathrm{S}}}{1 + A_{\mathrm{R}} A_{\mathrm{S}}} \cdot W + \frac{A_{\mathrm{S}}}{1 + A_{\mathrm{R}} A_{\mathrm{S}}} \cdot Z. \tag{4.5}$$

Um den Einfluss der Störung Z gering zu halten, muss man A_{R} sehr groß machen. Dies hat dann Nachteile, wenn Regelstrecke und Regler sich nicht so ideal verhalten wie angenommen. Beispielsweise führen die immer vorhandenen Verzögerungszeiten in Regler und Regelstrecke dazu, dass die Regelung beim Überschreiten einer bestimmten Gesamtverstärkung schwingt. Ein Ausweg ist, die reine Proportionalregelung dadurch zu erweitern, dass man die Vorgeschichte und den Trend mit berücksichtigt. Zum Proportionalteil $A_{\mathrm{R}} \cdot (W - X)$ wird ein Integralteil

$$A_{\mathrm{I}} \cdot \frac{1}{\tau_{\mathrm{I}}} \cdot \int_{-\infty}^{t_0} (W(t) - X(t)) \cdot \exp \frac{t - t_0}{\tau_{\mathrm{I}}} \cdot dt \tag{4.6}$$

und ein Differentialteil

$$A_{\mathrm{D}} \cdot \tau_{\mathrm{D}} \cdot \frac{d(W - X)}{dt} \tag{4.7}$$

hinzugenommen (**PID-Regler**).

PID-Regler sind im analogen Bereich heute Stand der Technik. Ein Analogregler kann als (linearer) Verstärker angesehen werden, bei dem die Verstärkung, sowie Integral- und Differentialanteil variiert werden können.

Beim digitalen Regler wird die gemessene Regelgröße in einen Digitalwert umgesetzt (A/D-Wandler), die Führungsgröße liegt digital als Konstante, Funktion oder Tabelle vor und der eigentliche Regelalgorithmus ist durch ein Programm realisiert. Die Stellgröße wird durch einen D/A-Wandler wieder analog gemacht und – nach entsprechender Verstärkung – der Regelstrecke zugeführt. Wichtig ist eine sinnvolle Diskretisierung, der Zeittakt dafür kann intern (Timer) oder extern (Messtakt eines Digitalmultimeters) vorgegeben werden.

Der Vorteil des Analogreglers ist sicher im geringeren Aufwand und damit auch geringeren Preis zu sehen; wo dies keine allzu große Rolle spielt, sprechen einige wesentliche Punkte für die Verwendung von digitalen Reglern (Aufzählung ohne Anspruch auf Vollständigkeit):

- Die Führungsgröße kann eine beliebige zeitliche Funktion sein. So lassen sich in Schmelzöfen (Kristallzüchtung) sehr komplexe Temperaturprogramme realisieren.
- Störgrößen (Umgebungstemperatur etc.) können separat gemessen und in der richtigen Weise berücksichtigt werden, bevor ihr Einfluss – zeitlich verzögert – in der Regelgröße zu sehen ist.

- Die Übertragungsfunktion darf auch sehr kompliziert sein, den Rechner stört das nicht. Insbesondere lassen sich Nichtlinearitäten der Regelstrecke berücksichtigen und Grenzbedingungen für die Stellgröße oder deren Änderung festlegen.
- Driftprobleme spielen nur noch in den Wandlern eine Rolle, nicht mehr im Reglerteil.
- Die Übertragungsfunktion kann während der Regelung geändert werden, es lassen sich Regelalgorithmen konstruieren, die sich während der Regelung an die Eigenschaften der Regelstrecke anpassen (adaptive Regler).
- Man kann mit unscharf formulierten Regelalgorithmen arbeiten – *Fuzzy-Regler*.
- Mehrere Stellgrößen und mehrere Regelgrößen lassen sich kombinieren.

5 MATLAB I : Messdatenerfassung

Wenn man einigermaßen komfortable Programme zur Datenerfassung und Steuerung an Experimenten zu erstellen hat, kann das zu einer sehr langwierigen und aufwendigen Aufgabe werden, insbesondere dann, wenn neben der reinen Datenerfassung auch noch weiter gehende Funktionen wie Datenanalyse, Graphikerstellung, Anpassungsrechnungen integriert sein sollen. Allerdings sind die Teilprobleme in vielen Bereichen gleichartig und lassen sich sehr gut modularisieren. Bestimmte Peripheriegeräte sind zu bedienen, Messdaten müssen gepeichert werden, Transformationen oder Fits sind nötig. Das legt es nahe, für die Teilaufgaben fertige Module bereitzustellen, die dann nur noch geeignet kombiniert werden müssen. Verschiedene Hersteller bieten dafür spezialisierte graphische Entwicklungsumgebungen an, mit denen man Mess- und Steuerprogramme relativ einfach bausteinartig zusammensetzen kann. Führend auf diesem Markt sind derzeit die Umgebungen LabView® von National Instruments [17] und VEE® von Agilent [18]. Verwendet man solche Werkzeuge, ist es allenfalls noch notwendig, kurze Treiberroutinen für exotische Hardware im klassischen Sinne selbst zu programmieren. Fast alle kommerziellen Hardwarehersteller liefern zu ihrer Hardware fertige Module für eine oder beide der genannten Entwicklungswerkzeuge mit. Im industriellen Umfeld hat sich diese Art der graphischen Messprogrammerstellung inzwischen weitgehend durchgesetzt, da man damit die Entwicklungszeiten beträchtlich verkürzen kann und da die Kosten der Werkzeuge meist keine allzu große Rolle spielen.

> Effizient zu progammieren heißt auch,
> effizient von fertigen Produkten Gebrauch zu machen.

Ein anderes, in Bereichen wie Physik oder Informatik näher liegendes Konzept zur Arbeitserleichterung ist die Kombination von Numerikprogrammen mit kurzen selbstgeschriebenen Routinen. Die selbstgeschriebenen Teile übernehmen die Kopplung an die Messperipherie, das Numerikprogramme alle weiter gehenden Aufgaben. Dabei ist es sinnvoll, ein Numerikprogramm zu verwenden, das auch auf anderen Gebieten (Theorie, Simulation) gut einsetzbar ist. Die umfangreichsten und modernsten Möglichkeiten bietet hier seit geraumer Zeit MATLAB®, ein Produkt der Firma MathWorks [19]. Ebenfalls recht vielseitig, aber nicht ganz so professionell und benutzerfreundlich ist Scilab [20], ein Programm aus dem *Public-Domain*-Bereich.

Zur Informationsübertragung zwischen den Progammteilen – Daten und Steuerungsanweisungen müssen ausgetauscht werden – können unterschiedliche Mechanismen implementiert sein, betriebssystemspezifische (Pipes, DDE[13], OLE[14],

[13] *D*ynamic *D*ata *E*xchange.
[14] *O*bject *L*inking and *E*mbedding.

ActiveX, COM[15], DCOM[16], CORBA[17] usw.), aber auch weitgehend betriebssystemunabhänge (Dateien – binär oder Text – oder direkte Parameterübergabe zwischen den beteiligten Funktionen beipielsweise mit TCP/IP-Mechanismen).

Am Beispiel MATLAB sollen verschiedene Konzepte zur Steuerung und Datenerfassung genauer betrachtet werden. Auf einige der Standardschnittstellen (insbesondere die seriellen Schnittstellen) des Rechners ist ein Zugriff mit MATLAB-eigenen Objekten möglich, ansonsten werden externe Programme oder Funktionen benötigt. Mit der Windows-Version von MATLAB können unter anderem die windowstypischen Kommunikationsstandards DDE und ActiveX verwendet werden, wenn ein geeignetes Partnerprogramm vorhanden ist. So ist beispielsweise ein Datenaustausch mit Programmen wie *Excel* per DDE möglich, mit dem ActiveX-Mechanismus kann Hardware dann angesprochen werden, wenn geeignete Treiber verfügbar sind (Näheres zu beiden Konzepten im MATLAB-Hilfesystem). Wir werden hier die MATLAB-eigenen Standards der MEX-Unterprogramme (*MATLAB EX*tension) und der 'MATLAB Engine' unter Windows näher betrachten, die zwar nicht ganz betriebssystemunabhängig sind, aber in ähnlicher Form auch unter anderen Betriebssystemen (UNIX, Linux) implementiert sind. Das MATLAB-Handbuch *External Interfaces* [21] meint dazu:

> *Although MATLAB is a complete, self-contained environment for programming and manipulating data, it is often useful to interact with data and programs external to the MATLAB environment. MATLAB provides an interface to external programs written in the C and Fortran languages.*

MATLAB ist ursprünglich als reines Text-*Frontend* für Numerik-Pakete entwickelt worden, daher ist eine Bedienung über Texteingaben oder Skripte nahe liegend. Es wurden aber auch schon früh Möglichkeiten integriert, graphische Benutzeroberflächen zu erstellen. Dies ist besonders für Anwendungen in der Messdatenerfassung interessant, bei denen sich gleichartige Abläufe (Messungen) häufig wiederholen können. Auch einige dieser Möglichkeiten sollen an Beispielen diskutiert werden.

5.1 Hardware-Zugriff mit MATLAB-Funktionen

Ein Teil der PC-Hardware kann von MATLAB aus direkt angesprochen werden, so die seriellen und die Druckerschnittstellen.

Die Funktion `serial` erstellt ein MATLAB-Objekt für die serielle Schnittstelle, mit der Anweisung

```
S1 = serial('COM1', 'BaudRate', 9600);
```

beispielsweise wird die erste serielle Schnittstelle auf eine Geschwindigkeit von 9600 Baud eingestellt und in `S1` bereitgestellt. Weiter verwendet wird `S1` dann

[15] *C*omponent *O*bject *M*odel.
[16] *D*istributed COM.
[17] *C*ommon *O*bject *R*equest *B*roker *A*rchitecture.

ähnlich wie ein Datei-Objekt. Nach `fopen(S1)` können Daten mit `fprintf(S1,...)` ausgegeben, mit `x=fscanf(S1)` gelesen werden. Weitere Schnittstelleneigenschaften werden mit `set(S1,...)` eingestellt, `fclose(S1)`) schließlich beendet die Verbindung.

Ebenfalls wie Datei-Objekte werden die Druckerschnittstellen gehandhabt. Man hat die Wahl zwischen C-artigen und Java-artigen Funktionen[18]:

```
Init = sprintf ('%cU1%c8%cs%c', 27, 27, 27, 1);
Pos  = sprintf ('%cl%c.%c\r\n', 27, Position, 7);
printer = fopen ('lpt1:', 'w');
fprintf (printer, Init);
fprintf (printer, Pos);
fclose (printer);
```

oder

```
ESC = 27; BELL = 7; CR = 13; LF = 10;
Init = [ESC 'U1' ESC '8' ESC 's' 1];
Pos  = [ESC 'l' Position '.' BELL CR LF];
printer = java.io.FileWriter('lpt1:');
printer.write(Init);
printer.write(Pos);
printer.close; .
```

Beide Fragmente positionieren einen Epson-kompatiblen Drucker auf `'Position'`.

Darüber hinaus ist es mit Java-Objekten auch möglich, die Ethernet-Schnittstelle des Rechners von MATLAB aus anzusprechen. Für Netzwerk-Verbindungen ist das *Package* `java.net` zuständig, das explizit importiert werden muss.

Ein MATLAB-Skript für einen Ethernet-Client, der an den Server `ipc1` zum TCPIP-Port `1234` eine Zeichenfolge `sendStr` schickt und die Zeile `recvStr` empfängt, könnte somit etwa diese Anweisungen enthalten:

```
import java.net.*
socket = java.net.Socket('ipc1.physik.uni-osnabrueck.de', 1234);
str = java.lang.String(sendStr);
out = socket.getOutputStream;
out.write(str.getBytes);
in = socket.getInputStream;
isr = java.io.InputStreamReader(in);
ibr = java.io.BufferedReader(isr);
recvStr = ibr.readLine;
socket.close; .
```

Die Verbindung wird mit dem `Socket`-Objekt etabliert, mit der `write`-Methode des zugehörigen `OutputStream` wird die Anfrage abgeschickt. Die Antwort wird mit dem `InputStream` gelesen, dem ein `BufferedReader` angehängt wurde, um bequem mit `readLine` lesen zu können.

[18] Die neueren Versionen von MATLAB (in Release 11 inoffiziell und rudimentär, ab Release 12 offiziell) stellen ein Software-Interface zur *Java Virtual Machine* des Rechners bereit, über das einerseits komplette Java-Programme in MATLAB eingebunden werden können, andererseits aber auch einzelne Java-Anweisungen an die *VM* geschickt werden können.

5.2 Externe Programme

Sollen – von MATLAB aus gesehen – nur einfache Aktionen angestoßen werden, ohne dass umfangreiche Informationen zurückgeliefert werden, kann man das über ein externes Programm erledigen, dem die benötigten Parameter in der Kommandozeile übergeben werden. Ein C++-Programm `mul`, das etwa folgende `main`-Funktion enthält

```
int main(int argc, char* argv[])
{
  if (argc<3) {
    printf("Need two input numbers!\n");
    return -1;
  }
printf("%g\n",atof(argv[1])*atof(argv[2]));
return 0;
}
```

könnte aus MATLAB mit

```
r = 2;
[R,A] = dos(['mul ',num2str(pi),' ',num2str(r*r)]);
```

`R` enthält dann den `return`-Wert des Programms (0 bei Erfolg, -1 bei Misserfolg), `A` den Ausgabetext. Mit `str2double(A)` würde man daraus einen Zahlenwert machen.

5.3 MEX-Funktionen

Von MATLAB aufrufbare C/C++- oder Fortran-Subroutinen (MEX-Dateien) werden unter Windows als DLLs (*D*ynamic *L*ink *L*ibraries) erstellt und beim Aufruf aus MATLAB vom Betriebssystem automatisch geladen und ausgeführt:

> *You can call your own C or Fortran subroutines from MATLAB as if they were built-in functions. MATLAB callable C and Fortran programs are referred to as MEX-files. MEX-files are dynamically linked subroutines that the MATLAB interpreter can automatically load and execute. MEX-files have several applications:*

- ◇ *Large pre-existing C and Fortran programs can be called from MATLAB without having to be rewritten as M-files.*
- ◇ *Bottleneck computations (usually for-loops) that do not run fast enough in MATLAB can be recoded in C or Fortran for efficiency.*

Der dem MATLAB-Handbuch [21] entnommenen Auflistung wäre noch der Bereich der Kopplung ans Experiment hinzuzufügen.

Zur Realisierung von MEX-Dateien ist im MATLAB-System das Kommando *mex* vorgesehen; nach richtiger Konfiguration mit `mex -setup` ist es in der Lage, aus C- oder C++-Dateien die DLLs zur Verwendung unter MATLAB zu erstellen. Die Quelldatei wird dazu im jeweils aktuellen Verzeichnis – z. B. mit dem MATLAB-Texteditor – angelegt. Voraussetzung ist, dass auf dem Rechner einer der von

MATLAB unterstützten C/C++-Compiler vorhanden ist. Ein einfacher frei verfügbarer C-Compiler (LCC) wird bei MATLAB mitgeliefert, damit können in jedem Fall MEX-Programme compiliert werden (allerdings nur in C).

MATLAB kommuniziert mit einer MEX-Datei über die Interface-Funktion `mexFunction()`, die mithin in jeder MEX-Datei vorhanden sein muss. Diese Funktion hat die Signatur

```
void mexFunction ( int nlhs, mxArray *plhs[],
                   int nrhs, const mxArray *prhs[] ),
```

die 4 Parameter beschreiben die Eingabe- und Ausgabeobjekte der MEX-Funktion (*l*eft *h*and side, *r*ight *h*and side). Die Felder `prhs` und `plhs` enthalten Zeiger auf die Objekte, `nrhs` und `nlhs` sind deren Anzahl. Bei einer MEX-Funktion `func.dll`, die aus MATLAB mit

```
[u, v] = func (a);
```

aufgerufen wird, ist `nlhs = 2`, `plhs[0]` zeigt auf `u`, `plhs[1]` auf `v`, `nrhs = 1`, `prhs[0]` zeigt auf `a`.

Der C++-Quellcode einer MEX-DLL, die als einziges Objekt ein zweidimensionales Messdatenfeld zurückgibt, enthält etwa die folgenden Zeilen:

```
#include "mex.h"
...
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
    plhs[0] = mxCreateDoubleMatrix(YSIZE, XSIZE, mxREAL);
    double * z = mxGetPr(plhs[0]);
    ...
    // put data into array z
}  .
```

Das Matrix-Objekt wird mit `mxCreateDoubleMatrix()` angelegt, `plhs[0]` zeigt darauf. Mit `mxGetPr(plhs[0])` beschafft man sich im Programm einen Zeiger auf den Anfang des eigentlichen Datenfelds.

5.4 MEX, *Microsoft Foundation Classes*, andere Compiler

Will man sich nicht auf einfache Funktionen beschränken oder andere als die unterstützten Compiler verwenden, dann ist etwas mehr Programmieraufwand nötig. So lassen sich beispielsweise die Möglichkeiten der *Microsoft Foundation Classes* in MEX-Funktionen verwenden, um komfortable Benutzeroberflächen zu erstellen. In seinen neueren Versionen bietet MATLAB inzwischen genügend eigene Funktionalität in diesem Bereich, so dass die Verwendung externer Programmierung meist nicht erforderlich ist. Daher werden die Vorgehensweise im vorliegenden Skriptum nicht näher erläutert. Falls Sie dennoch Informationen dazu benötigen, finden Sie Ausführlicheres dazu im Vorgängerskriptum (2006 oder früher).

5.5 Externe Bibliotheken

Mit der Funktion `loadlibrary` ist MATLAB in der Lage, externe Funktionsbibliotheken zu laden (*D*ynamic *L*ink *L*ibraries, DLLs, unter Windows, *Shared Libraries* unter Linux oder UNIX). Als Parameter werden beim Aufruf der Name der externen Bibliothek und der Name der Header-Datei mit den Signaturen der Bibliotheksfunktionen mitgegeben. Die Funktionen der externen Bibliothek können dann mit `calllib` aufgerufen werden.

Eine einfache Beispiel-DLL soll die Vorgangsweise verdeutlichen. Es werden drei C++-Funktionen implementiert, die Zahlen multiplizieren.

```
#include "dllmul.h"
#include <stdio.h>
#include <stdlib.h>

double multiply(const double x, const double y) {
  return x*y;
}
double* nmultiply(const double* x, const double* y, const int N) {
  static double* z = (double*)malloc(N*sizeof(double));
  for (int n=0;n<N;n++)
    *(z+n) = *(x+n)**(y+n);
  return z;
}
char* cmultiply (const char* s1, const char* s2) {
  static char a[20];
  sprintf(a,"%9.2f", atof(s1)*atof(s2));
  return a;
}
```

Die Funktionen unterscheiden sich in der Art und Anzahl der Parameter. Die erste, `multiply`, multipliziert zwei Zahlen, die Parameter sind vom Typ `double`. Die zweite, `nmultiply`, multipliziert zwei Zahlenfelder, die Parameter sind Zeiger auf die Felder. Die dritte, `cmultiply`, multipliziert zwei Zahlen, die als Strings vorliegen, Rückgabewert ist ebenfalls ein String.

Die zugehörige Header-Datei beschreibt die Funktionen und sorgt dafür, dass die Funktionsnamen nicht C++-artig verändert werden (`extern "C"`) und dass der Compiler sie auf die Export-Liste setzt (`__declspec(dllexport)`)[19]. Im Compiler-System ist weiterhin einzustellen, dass eine DLL erstellt werden soll.

```
#ifdef __cplusplus
    extern "C" {
#endif
__declspec(dllexport) double multiply (const double, const double);
```

[19] Diese Spracherweiterung zu C++ ist nicht allgemeiner Standard. Das Beispielprogramm wurde mit dem Gnu-Compiler DEV-C++ von Bloodshed Software [22] übersetzt, der – wie auch der Microsoft-Compiler – damit zurecht kommt. Falls ein Compiler die Erweiterung nicht akzeptiert, muss die EXPORT-Liste in der .DEF-Datei explizit vermerkt werden.

```
__declspec(dllexport) double* nmultiply
    (const double*, const double*, const int);
__declspec(dllexport) char* cmultiply (const char*, const char*);
#ifdef __cplusplus
  }
#endif
```

Um die Funktionen in MATLAB zu verwenden, wird zunächst die Bibliothek geladen, etwa mit

```
if (~libisloaded('dllmul')),
    loadlibrary('dllmul','dllmul.h');
end;
```

Die Abfrage, ob die Bibliothek schon vorhanden ist, vermeidet unnötige Fehlermeldungen.

Mit

```
Z =  = calllib('dllmul','multiply',X,Y);
```

würden dann `X` und `Y` multipliziert, das Ergebnis landet in `Z`.

Zwei Datenfelder `U` und `V` der Länge `N` würden mit

```
pW = calllib('dllmul','nmultiply',U,V,N);
setdatatype(pW,'doublePtr',1,N);
W = pW.Value;
```

multipliziert, als Rückgabewert wird ein Zeiger geliefert, dem in MATLAB der richtige Datentyp zugewiesen werden muss, damit das Ergebnisfeld `W` zugänglich wird.

Variablen in Textform würden nach dem Schema

```
a = calllib('dllmul','cmultiply','3.5','1.2e4');
```

bzw.

```
a = calllib('dllmul','cmultiply',s1,s2);
```

multipliziert, `a` liefert das Ergebnis ebenfalls als String.

5.6 MATLAB als 'Engine'

MEX-Funktionen sind Erweiterungen des MATLAB-Befehlsumfangs und werden aus MATLAB aufgerufen. Umgekehrt kann man auch den Befehlsumfang des eigenen Programms durch MATLAB erweitern, MATLAB-Funktionen z. B. aus einem Messprogramm aufrufen:

MATLAB provides a set of routines that allows you to call MATLAB from your own programs, thereby employing MATLAB as a computation engine. MATLAB engine programs are C or Fortran programs that communicate with a separate MATLAB process via pipes (in UNIX) and through ActiveX on Windows. There is a library of functions provided with MATLAB that allows you to start and end the MATLAB process, send data to and from MATLAB, and send commands to be processed in MATLAB. Some of the things you can do with the MATLAB engine are:

- ◇ *Call a math routine to invert an array or to compute an FFT from your own program. When employed in this manner, MATLAB is a powerful and programmable mathematical subroutine library.*
- ◇ *Build an entire system for a specific task, for example, radar signature analysis or gas chromatography, where the front end (GUI) is programmed in C and the back end (analysis) is programmed in MATLAB, thereby shortening development time.*

Die Nutzung von MATLAB als 'Engine' veranschaulicht die Funktion `OnFilter()`, die MATLAB aufruft, um ein zweidimensionales Datenfeld (quadratisches Bild `b` der Größe `BSIZE*BSIZE`) zu filtern:

```
#include "engine.h"
...
void CMulDoc::OnFilter()
{
  int i, k;
  Engine * ep = engOpen(NULL);
  engEvalString(ep, "f=ones(5);");
  mxArray * z = mxCreateDoubleMatrix (BSIZE, BSIZE, mxREAL);
  double * Z = mxGetPr (z);
  for ( i=0; i<BSIZE; i++ )
    for ( k=0; k<BSIZE; k++ )
      * (Z + i*BSIZE + k) = b[k][i];
  engPutVariable (ep, "z", z);
  engEvalString(ep, "z=filter2(f,z);");
  engEvalString(ep, "z=z-min(min(z));");
  engEvalString(ep, "z=z*255/max(max(z));");
  mxArray * res = engGetVariable (ep, "z");
  Z = mxGetPr (res);
  for ( i=0; i<BSIZE; i++ )
    for ( k=0; k<BSIZE; k++ )
      b[k][i] = (BYTE) (* (Z + i*BSIZE + k));
  mxDestroyArray (res);
  mxDestroyArray (z);
  engClose(ep);
  UpdateAllViews(NULL);
}
```

MATLAB wird mit `engOpen()` gestartet, mit `f=ones(5)` wird in MATLAB ein einfaches zweidimensionales Rechteckfilter definiert. Das Datenfeld wird an MATLAB übergeben, mit der zweidimensionalen Filterfunktion `filter2()` gefiltert, wie-

der auf 8-Bit-Dynamik normiert und zurückkopiert. Lässt man die `engClose`-Anweisung am Ende weg, kann man nach dem Motto 'MATLAB, übernehmen Sie!' mit den übergebenen Daten in MATLAB weiterarbeiten.

Auch bei Engine-Anwendungen sind gewisse Vorarbeiten nötig: ähnlich wie bei den MEX-Anwendungen müssen zusätzliche Bibliotheken eingebunden werden, diesmal `libmx.lib`, `libeng.lib`. Darüber hinaus sind die Pfade zur Header-Datei `engine.h` und zu den Bibliotheksdateien dem Compiler und Linker bekannt zu machen. Unter Windows muss MATLAB als COM-Server registriert werden, damit alles funktioniert, das ist mit dem Aufruf

```
matlab /regserver
```

im MATLAB-Programmverzeichnis zu machen.

Einen ausführlichen und vollständigen Überblick über die von MATLAB bereitgestellten Schnittstellenfunktionen geben die zuständigen Handbücher (Benutzerhandbuch gedruckt [21] und als PDF-Datei [23]; Referenzhandbuch in HTML [24] und PDF [25]) oder das MATLAB-Hilfesystem. Ein weiteres Engine-Beispiel ist im Skriptum zur Vorlesung 'Graphik-Workshop' [26] beschrieben. Dort geht es um den Tranfer von Bilddaten, die aus einer Twain-Quelle stammen, an MATLAB.

5.7 ActiveX

In der Windows-Version kann MATLAB ActiveX-Komponenten[20] verwenden, sowohl so genannte *Controls* wie auch *Server*. Damit wird es sehr einfach, mit Rechnerperipherie zu arbeiten, für die ActiveX-Controls vom Hersteller mitgeliefert werden. Dies ist inzwischen bei sehr vielen kommerziellen Steckkarten und auch bei externen Geräten zur Datenerfassung und Steuerung der Fall. Auch bei selbst programmierten Software-Komponenten sollte man die Möglichkeit in Betracht ziehen, den ActiveX-Standard zu benutzen. Dies zumindest dann, wenn man mit einem Software-Entwicklungssystem arbeitet, das dies einigermaßen einfach ermöglicht.

MATLAB-Objekte für ActiveX-Controls werden mit

```
h = actxcontrol('Programm-ID',...);
```

erstellt. `Programm-ID` ist der Name, unter dem die ActiveX-Komponente in der *Registry* eingetragen ist. Als weitere Parameter können beim Aufruf angegeben werden (s. MATLAB-Hilfe): Position und Größe des Controls, Handle zum *Parent*-Objekt und eine oder mehrere *Callback*-Funktionen für Ereignisse (Events) des Controls. Die möglichen Ereignisse können mit `events(h)` erfragt werden.

Eigenschaften können, wie bei allen MATLAB-Objekten üblich, mit `set` und `get`

[20] ActiveX ist eine der innerhalb des Microsoft Component Object Model (COM) definierten Software-Schnittstellen. Software-Komponenten – in der Regel DLLs – unterschiedlicher Herkunft können über diese Interface-Definition relativ problemlos zu variablen größeren Systemen zusammengefasst werden.

festgelegt und abgefragt werden, interaktiv geht das einfacher mit dem *Property Editor*, der mit `inspect(h)` aufgerufen wird.

Methoden – Funktionen – des Objekts ruft man entweder mit

```
a = invoke(h, 'Function', Parameter1, Parameter2, ...);
```

oder mit

```
a = Function(h, Parameter1, Parameter2, ...);
```

auf. Ein leeres `invoke(h)` liefert eine Liste der mit dem Objekt verbundenen Methoden.

5.8 Dateiformate

MATLAB kann eine ganze Anzahl von Dateiformaten direkt erstellen und lesen, daneben sind verschiedene Funktionen implementiert, mit denen die Ein- und Ausgabe sehr flexibel gesteuert werden kann. Reicht dies alles nicht aus, so bleibt noch die Möglichkeit, eigene Funktionen in C/C++, Fortran oder Java zu implementieren.

5.8.1 SAVE und LOAD

Die Standardfunktionen zum Schreiben und Lesen von Daten sind `save` und `load`. Ohne Argumente speichert `save` alle Variablen der aktuellen MATLAB-Sitzung binär in der Datei `matlab.mat`, mit `load` wird diese Datei wieder gelesen, die Variablen werden wiederhergestellt. Durch Argumente kann man bei `save` den Namen der Datei, die zu speichernden Variablen, die Formatierung und die Datengenauigkeit vorgeben. So werden mit

```
save  <fname>  X Y Z
```

die Variablen (Matrizen) `X`, `Y` und `Z` in der Datei `<fname>` im MATLAB-eigenen Datenformat gespeichert, mit

```
save  <fname>  Z  -ASCII  -DOUBLE
```

wird MATLAB angewiesen, `Z` in Textformat mit hoher Genauigkeit (16 Nachkommastellen) zu speichern. Bei Textdateien (Option `-ASCII`) ist es meist nicht sinnvoll, mehrere Datenfelder anzugeben, da die Matrizen nacheinander geschrieben werden, ohne dass die Größe und der Variablenname vermerkt werden. Dies kann beim Wiedereinlesen Probleme machen, insbesondere dann, wenn die Spaltenanzahl der Matrizen unterschiedlich ist. Will man – wie bei Messdaten üblich – als Spaltenvektoren vorliegende Messwerte tabellenartig schreiben, muss man sie in MATLAB vorher zu einer Matrix zusammensetzen (`Z = [X,Y]`).

In ähnliche Weise lassen sich auch bei `load` nähere Spezifikationen angeben.

```
load  <fname>
```

liest die Datei `<fname>`. Ohne Dateinamenerweiterung oder mit `.MAT` als Erweiterung geht MATLAB von binärem MATLAB-Format aus, ansonsten von Textformat. Bei Binärdateien werden die Daten unter ihren ursprünglichen Namen wiederhergestellt, durch Angabe von einzelnen Variablennamen als Parameter kann man `load` darauf beschränken. Handelt es sich um eine Textdatei, wird der Inhalt in der Matrix `<fname>` abgelegt. Durch Verwendung der funktionalen Form kann man das verändern. So können durch Leerzeichen getrennte Zahlen aus einer Textdatei `test.dat` mit

```
Z = load('test.dat')
```

in die Matrix `Z` gelesen werden. Deren Spalten- und Zeilenzahl entspricht der Anordnung der Daten in der Textdatei.

5.8.2 Weitere Formate

Ein Überblick über weitere Dateiformate, die in MATLAB zum Austausch von Messdaten mit anderen Programmen verwendet werden können, ist im Skriptum zur Vorlesung 'Graphik-Workshop' [26] enthalten. Informationen dazu findet man auch in der MATLAB-Hilfe mit `help fileformats` und `help iofun`.

5.9 Graphische Benutzeroberflächen

Die Bedienung von (Mess-)Programmen wird dadurch beträchtlich erleichtert, dass man für bestimmte Aktionen Eingabehilfen wie Drucktasten oder Menüs zur Verfügung hat. MATLAB bietet zur Erstellung solcher graphischer Oberflächen umfangreiche Hilfsmittel. Basis ist immer ein `figure`-Objekt, das über die Funktion `uimenu` mit zusätzlichen Menüpunkten und über die Funktion `uicontrol` mit Tasten, Eingabefenstern, Auswahllisten und ähnlichem ausgestattet werden kann.

Abbildung 53 zeigt ein einfaches Beispiel, in dem mathematische Funktionen mit den Einfachkommandos `ezplot`, `ezsurf` oder `ezmesh` gezeichnet werden können. Die gewünschte Funktion wird im Textfeld eingegeben, nach Drücken der jeweiligen Taste geplottet.

Das zugehörige MATLAB-Skript erstellt zunächst ein `figure`-Objekt als Rahmen, darin mit `axes` eine Zeichenfläche.

```
figure( ...
    'Name','User Interface in MATLAB', ...
    'NumberTitle','off');
axes( ...
    'Units','normalized', ...
    'Position',[0.10 0.12 0.6 0.78], ...
    'FontUnits','normalized','FontSize',0.055);
Pos = [0.75 0.8 0.2 0.05];
dPos = [0 -0.1 0 0];
hText = uicontrol( ...
```

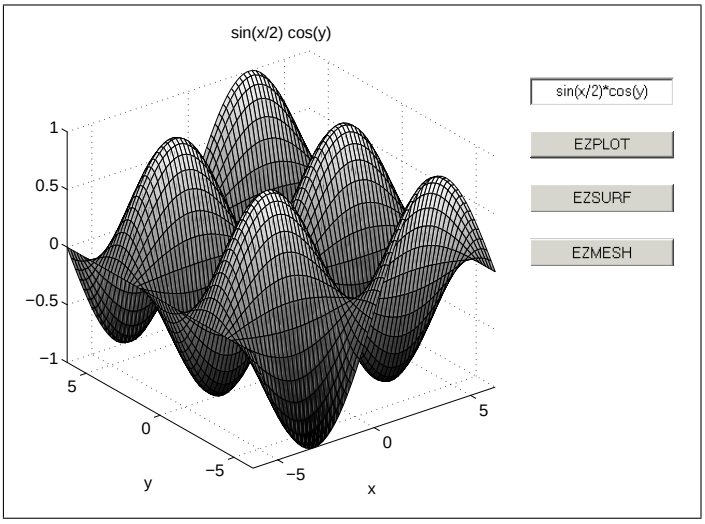


Abbildung 53: Graphische Benutzeroberfläche zum Test von Funktionsplots.

```
    'Style','edit', ...
    'Units','normalized', ...
    'Position',Pos, ...
    'BackgroundColor',[1 1 1], ...
    'String','sin(x/2)*cos(y)');
uicontrol( ...
    'Style','pushbutton', ...
    'Units','normalized', ...
    'Position',Pos+dPos, ...
    'String','EZPLOT', ...
    'Callback','ezplot(get(hText,''String''))');
...
```

Das Textfeld wird vorbesetzt, beim Drücken der Taste (hier `EZPLOT`) wird das Textfeld gelesen und die gewünschte Funktion (`ezplot`) damit aufgerufen.

Die Funktion `uicontrol` erstellt Graphik-Objekte, die durch Eigenschafts-Werte-Paare näher definiert werden. So gibt der Wert für `Style` an, was für ein Typ `uicontrol` erstellt werden soll, `Pos` die Lage und Größe, `String` die Beschriftung. Aktionselemente brauchen eine `Callback`-Eigenschaft, die die auszuführende Aktion festlegt. Der Rückgabewert von `uicontrol` ist ein Objekt-Pointer (*Handle*), mit dem Eigenschaften des Objekts später erfragt (`get`) oder verändert (`set`) werden können.

Das zweite Beispiel implementiert eine einfache Messumgebung (Abbildung 54). Für Messparameter sind Eingabefelder vorgesehen, verschiedene Aktionen können per Tastendruck gestartet werden.

Die Aktionen sind jeweils durch die `Callback`-Eigenschaft festgelegt:

```
uicontrol( 'Style', 'pushbutton', ...
    'String', 'MEASURE', ...
    'Callback', measurecb);
uicontrol( 'Style', 'pushbutton', ...
    'String', 'CALCULATE', ...
```

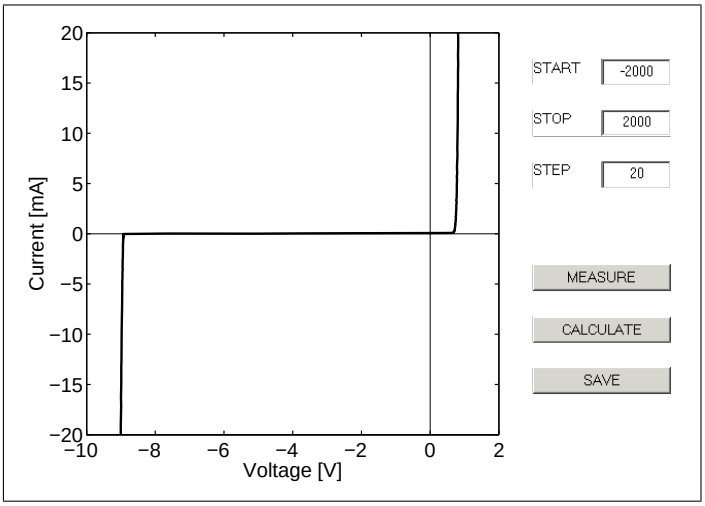


Abbildung 54: Graphische Benutzeroberfläche zur Messdatenerfassung und -darstellung (gemessen wurde gerade die Kennlinie einer 9-V-Zenerdiode).

```
    'Callback', 'errordlg(''Not implemented'',''Error'')');
uicontrol( 'Style', 'pushbutton', ...
    'String', 'SAVE', ...
    'Callback', savecb);
```

Beim Drücken der `CALCULATE`-Taste wird mit `errordlg` ein Dialogfenster mit Fehlermeldung aufgemacht, die beiden weiteren Aktionen sind an anderer Stelle im MATLAB-Skript implementiert, `measurecb` als

```
measuring = 0;
measurecb = [...
    'if measuring==0,' ...
    'measuring = 1;' ...
    'x0 = str2num(get(hStart,''String''));' ...
    'x1 = str2num(get(hStop,''String''));' ...
    'dx = str2num(get(hStep,''String''));' ...
    '[x,y] = domeas(x0,x1,dx);' ...
    'measuring = 0;' ...
    'end;']; .
```

Die Zahlenwerte (`str2num`) der Texteingaben (`hStart`, `hStop`, `hStep` sind die Handles der Textfelder) werden als Parameter an die eigentliche Messfunktion `domeas` übergeben, zurückgeliefert werden die Messwerte. Die *Semaphore* (Ampel) `measuring` sorgt dafür, dass ein zweiter Tastendruck während einer Messung ohne Folgen bleibt. Falls die Messung länger dauert, sollten innerhalb der Messfunktion regelmäßig aktuelle Messdaten gezeichnet werden, um über den Messablauf zu informieren. Dazu wird die Graphik zunächst konfektioniert (Datenbereich, Achsenbeschriftungen usw.) und mit `hold on` eingefroren. In der Messschleife innerhalb der Messfunktion `domeas` wird dann laufend nur noch der aktuelle Datenpunkt geplottet:

```
for n = 1:nmax,
  ...            % Slow Measurement
  plot(x(n), y(n), 'ko');
  if mod(n,10)==0, drawnow; end;
end; .
```

Der Aufruf von `drawnow` sorgt dafür, dass konkret gezeichnet wird, im obigen Fall nach jedem zehnten Punkt. MATLAB schiebt ansonsten alle Zeichenaktionen auf, bis nichts anderes mehr anliegt, würde also ohne diese Zeile erst am Ende der Messung zeichnen.

Die Aktion `savecb` macht zunächst mit `uiputfile` ein Standarddialogfenster zur Auswahl eines Dateinamens auf, Rückgabewerte sind Dateiname und vollständiger Pfad:

```
savecb = ...
    ['[fn,fp]=uiputfile(''*.*'');', ...
    'if fn~=0,', ...
    '  chdir(fp);', ...
    '  pairs=[x'',y''];' ...
    '  save(fn,''pairs'',''-ascii'');' ...
    'end;']; .
```

Vor dem Abspeichern mit `save` werden die im Messprogramm als Zeilenvektoren vorliegenden Messwerte in die gewünschte Tabellenform gebracht (zweispaltige Matrix `pairs`).

GUIDE

Als Hilfsmittel zur graphischen Erstellung von GUIs enthält MATLAB einen graphischen Editor (GUI Design Environment – GUIDE), der mit `guide` aufgerufen wird. Der Editor erstellt eine MATLAB-Graphik-Datei (`.fig`) und eine dazu korrespondierende Skript-Datei (`.m`). Seit Release 13 (Juni 2002) ist es auch möglich, alles in eine einzige Skript-Datei zu exportieren. Damit sind nachträgliche Änderungen und Anpassungen sowohl mit dem graphischen Editor wie auch mit einem Text-Editor möglich. Die Dateien sind allerdings etwas umfangreicher und unübersichtlicher als Skript-Dateien, in denen ein GUI direkt programmiert wird.

6 MATLAB II : Messdatenverarbeitung

MATLAB ist die Abkürzung für *MAT*rix *LAB*oratory, Vektoren und Matrizen sind die Variablentypen, mit denen MATLAB am besten umgehen kann. Das sind aber auch genau die Datentypen, in denen Messdaten üblicherweise vorliegen. Meist eindimensional, eine physikalische Größe wird in ihrer Abhängigkeit von einer anderen, unabhängig veränderlichen gemessen. So wird beim Freien Fall der zurückgelegte Weg in Abhängigkeit von der Zeit bestimmt, die Kennlinie einer Diode ist der Strom als Funktion der anliegender Spannung, die optische Absorption eines Materials wird meist wellenlängenabhängig gemessen, in der Kernspektroskopie interessiert man sich für die Energieverteilung der bei Zerfalls- oder Wechselwirkungsprozessen involvierten Teilchen. Zwei- oder mehrdimensional dagegen liegen Daten beispielsweise überall dort vor, wo physikalische Größen ortsabhängig gemessen werden. Alle bildgebenden Messverfahren – wie die Rastermikroskopie oder auch viele der Untersuchungsmethoden aus dem medizinischen Bereich – liefern solche Daten. Da in MATLAB Rechenverfahren für Matrizen sehr effizient implementiert sind, ist es gerade auch für die Weiterverarbeitung von ein- und mehrdimensionalen Messdaten hervorragend geeignet[21].

> Die in Messdaten enthaltene physikalische Information
> wird oft erst nach geschickter Bearbeitung sichtbar.

6.1 Filterung

Filterverfahren werden verwendet, um Datenfelder zu glätten, aber auch, um spezielle Informationen hervorzuheben. MATLAB stellt dafür verschiedene Funktionen zur Verfügung, die wichtigsten sind `filter` zur Filterung eines eindimensionalen Felds und `filter2` für zweidimensionale Felder. Die Verfahren berechnen die Faltung aus zwei Vektoren oder Matrizen, dem Datenfeld D und der Filterfunktion F.

So berechnet `Df = filter(F,norm,D)` an jedem Punkt des Ergebnis-Feldes `Df` die mit `F` gewichtete und mit `norm` normierte Summe der Werte des Ausgangsfeldes `D`, die in der Umgebung dieses Punktes liegen

$$Df(n) = \frac{D(n)F(1) + D(n-1)F(2) + \ldots + D(n-m+1)F(m)}{\text{norm}} \tag{6.1}$$

$$= \sum_{k=1}^{m} \frac{D(n-k+1)F(k)}{\text{norm}} \,. \tag{6.2}$$

Den Mittelwert aus jeweils 5 benachbarten Datenpunkten würde man demnach durch die Anwendung der Filterfunktion `F = ones(1,5)` und `norm = 5` bzw. `norm = sum(F)` erhalten.

[21] Für bestimmte Anwendungsbereiche bietet The MathWorks spezialisierte *Toolboxes* zu MATLAB an, so die 'Image Processing Toolbox' mit vielen Funktionen für die digitale Bildbearbeitung.

6.1.1 Gewichteter Mittelwert

Die oben beschriebene einfache Mittelwertbildung entspricht einer rechteckigen Filterfunktion. Oft ist es zweckmäßig, einen geeignet gewichteten Mittelwert zu berechnen, um nahe liegende Punkte stärker zu bewerten als entfernte. Die Form und Breite der konkreten Filterfunktion muss jeweils an die gewünschte Anwendung angepasst werden. In den meisten Fällen ist es sinnvoll, symmetrische Funktionen zu verwenden, die an ihren Rändern einigermaßen stetig verlaufen (Trapez, Dreieck, Cosinus o. ä.).

Das folgende Fragment wendet eine parabelförmige Filterfunktion `weight` auf ein verrauschtes Signal an:

```
HalfWidth = 20;
linear = [-HalfWidth:HalfWidth]/HalfWidth;
weight = 1 - linear.*linear;
Filtered = filter(weight, sum(weight), Signal); .
```

Im zweiten Parameter von `filter` kann ein zusätzlicher Wertevektor angegeben werden, mit der vorhergehende Werte des Ergebnisvektors berücksichtigt werden können (Näheres dazu in der Online-Hilfe). Im obigen Fall wird nur eine Konstante zur Normierung eingesetzt.

Die Wirkung zweier unterschiedlich breiter Filterfunktionen zeigt Abbildung 55. Die verwendeten Filterfunktionen sind mit eingezeichnet; deutlich ist die Abhängigkeit der Glättungswirkung, aber auch des Informationsverlusts von der Filterbreite.

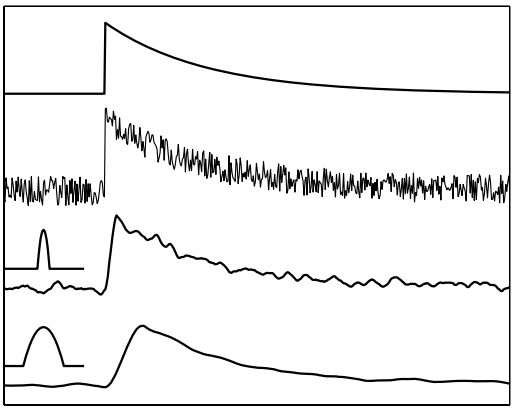


Abbildung 55: Filterung von verrauschten Messdaten mit Filterfunktionen unterschiedlicher Breite. Obere Kurve: idealer Verlauf, zweite Kurve: reales Messsignal, dritte und vierte Kurve: gefilterte Daten (Halbwertsbreite der Filterfunktionen: 6 bzw. 20 Punkte bei insgesamt 500 Datenpunkten).

Eine zweidimensionale Filterung mit `filter2` ist insbesondere bei zweidimensional ortsabhängigen Messwerten interessant. Die Glättung eines Rastertunnelmikroskopbildes veranschaulicht Abbildung 56. Links die Originalmessung, atomare Auflösung an einer Graphitprobe (HOPG[22]) mit dem bei Einfachgeräten üblichen Rauschen. Daneben das gefilterte Bild, als Filterfunktion wurde ein Rotationsparaboloid verwendet, dessen Basisdurchmesser in etwa dem atomaren Abstand entsprach.

[22] *H*ighly *O*rdered *P*yrolytic *G*raphite

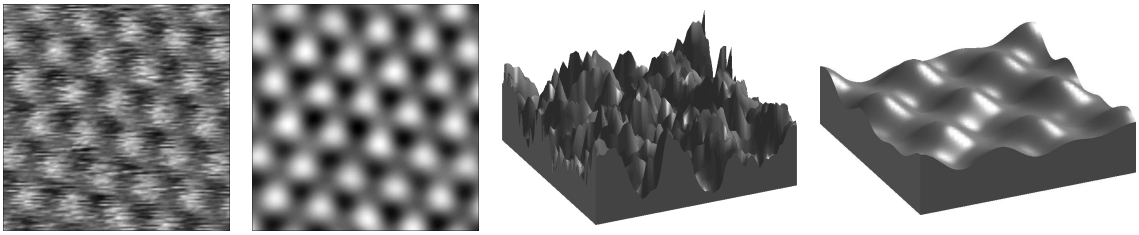


Abbildung 56: Rastertunnelmikroskopbild einer HOPG-Probe (atomare Ausflösung). Links die Originaldaten, daneben das gefilterte Bild, rechts Ausschnitte in dreidimensionaler Darstellung.

Besonders deutlich erkennt man die Wirkung des relativ drastischen Filters in den dreidimensional dargestellten Bildausschnitten rechts.

Die Konstruktion der Filterfunktion `ff` und ihre Anwendung auf die Bildmatrix `z` zeigt das folgende Fragment:

```
r0 = 10;
[xf,yf] = meshgrid(-r0:r0);
ff = r0*r0-xf.*xf-yf.*yf;
ff = ff.*(ff>0);
ff = ff/sum(sum(ff));
z = filter2(ff,z); .
```

Die Filtermatrix `ff` ist quadratisch, die Multiplikation mit `(ff>0)` in der vierten Zeile setzt die bei der Paraboloiddefinition $r_0^2 - r^2$ entstandenen negativen Matrixelemente auf null.

6.1.2 Gradientenfilter

Will man Daten nicht glätten, sondern bestimmte Detailinformationen verstärken, kann man dies mit speziell darauf zugeschnittenen Filtern erreichen. Mit einem einfachen Gradientenfilter beispielsweise lassen sich Flächen in einem Bild auf ihre Begrenzungslinien reduzieren. Das folgende Beispiel filtert ein Bild `b` mit einem Gradientenfilter für vertikale (Zeilenvektor `[-1,1]`) und einem für horizontale Gradienten (Spaltenvektor `[-1;1]`) und fasst die Ergebnisse durch eine Oder-Verknüpfung zusammen:

```
fver = [-1,1];
fhor = [-1;1];
bver = abs(filter2(fver, b));
bhor = abs(filter2(fhor, b));
bcontour = bver | bhor; .
```

Abbildung 57 zeigt die Wirkung, links das Originalbild mit seinen flächenhaften Objekten, rechts das gefilterte, auf dem nur noch die Umrisse sichtbar sind.

Filter dieser Art kann man dann verwenden, wenn geometrische Strukturen wie z. B. Begrenzungslinien zwischen Kristallbereichen (Domänen) automatisiert erkannt werden sollen. Man transformiert dann nach der Filterung in den Parameterraum des gesuchten geometrischen Objekts (Hough-Transformation [27]),

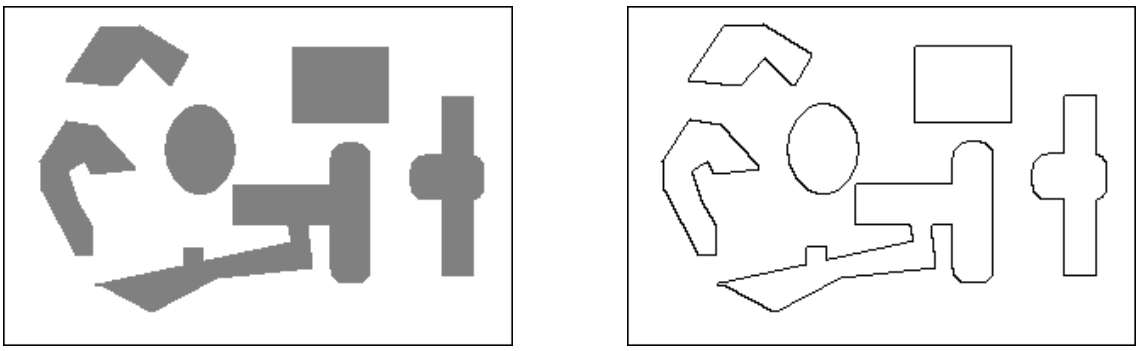


Abbildung 57: Gradientenfilter, links Original-, rechts gefiltertes Bild.

das Maximum dort liefert die Parameter des gefundenen Objekts. Falls das Ausgangsbild nicht kontrastreich genug für das Verfahren ist, kann man sich ein Bild mit extremem Kontrast dadurch generieren, dass man alle Helligkeitswerte oberhalb eines bestimmten Grenzwerts auf 1, die darunter auf 0 setzt. MATLAB erledigt das mit

```
HighContrast = LowContrast > Threshold; .
```

Darin ist `LowContrast` die Bildmatrix mit geringem Kontrast, `Threshold` der verwendete Grenzwert und `HighContrast` das resultierende Binärbild.

6.1.3 Savitzky-Golay-Filter

In Abschnitt 6.1.1 wurde beschrieben, wie man einen geeignet gewichteten Mittelwert zur Glättung von Messdaten verwenden kann. Dabei wurde auch ein gravierender Nachteil deutlich, scharfe Strukturen in Messdaten werden verbreitert. Besser einstellen lässt sich das, wenn man statt des Mittelwerts ein Polynom geeigneter Ordnung[23] verwendet, um Daten zu glätten. Das Prinzip ist in Abbildung 58 dargestellt.

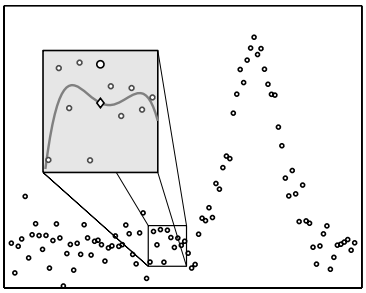


Abbildung 58: Gleitender Polynomfit: Für die Datenwerte in der Umgebung eines Punktes wird das Regressions-Polynom berechnet, der Datenpunkt (großer Kreis) wird durch den Polynomwert (Raute) ersetzt.

Das Verfahren scheint sehr aufwendig zu sein, für jeden Datenpunkt muss ein Regressions-Polynom berechnet werden. Man kann jedoch zeigen, dass dies nicht nötig ist und dass stattdessen eine geeignete Filterfunktion verwendet werden kann. Diese Art der Filterung wird nach den Autoren der ersten umfassen-

[23] Der einfache Mittelwert ist ein Fit-Polynom *nullter* Ordnung.

den Arbeit zu diesem Thema [28] – Abraham Savitzky und Marcel J. E. Golay – benannt.

Das Regressions-Polynom für die Umgebung des Datenpunktes i sei definiert durch

$$p_i(x) = \sum_{k=0}^{M} B_k \, (x - x_i)^k \; . \tag{6.3}$$

Die Polynomkoeffizienten B_k ergeben sich aus der Minimalisierung der Summe der Abstandsquadrate (*least squares fit*)

$$D_i = \sum_{j=i-n_L}^{i+n_R} (p_i(x_j) - y_j)^2 \stackrel{!}{=} \text{Min.} \tag{6.4}$$

Dabei wurde angenommen, dass das Polynom für n_L linksseitige und n_R rechtsseitige zusätzliche Punkte berechnet wird. Das Minimum wird dadurch bestimmt, dass man die partiellen Ableitungen nach allen B_m berechnet und zu Null setzt

$$\frac{\partial D_i}{\partial B_m} = 0 \, , \qquad m = 0 \ldots M \, . \tag{6.5}$$

Bei Messdaten kann man als Vereinfachung fast immer annehmen, dass die Datenpunkte äquidistant sind

$$x_j - x_i = (j - i)\Delta x \, , \quad j - i = n \, , \quad n = -n_L \ldots 0 \ldots n_R \, . \tag{6.6}$$

Damit wird dann aus Gleichung 6.3

$$p_i(x_j) = \sum_{k=0}^{M} B_k n^k \Delta x^k = \sum_{k=0}^{M} b_k n^k \, , \qquad n = j - i \, . \tag{6.7}$$

Die partiellen Ableitungen (Gl. 6.5) liefern $M+1$ Gleichungen ähnlicher Struktur

$$\sum_{n=-n_L}^{n_R} n^m \left(\sum_{k=0}^{M} b_k n^k - y_{i+n} \right) = 0 \, , \qquad m = 0 \ldots M \, , \tag{6.8}$$

$$\sum_{n=-n_L}^{n_R} n^m \sum_{k=0}^{M} b_k n^k = \sum_{n=-n_L}^{n_R} n^m y_{i+n} \, . \tag{6.9}$$

Das lineare Gleichungssystem 6.9 für die b_k könnte nun mit den üblichen Methoden gelöst werden. Das müsste für jeden Datenpunkt i gemacht werden.

Zur weiteren formalen Betrachtung schreibt man das lineare Gleichungssystem 6.9 in Matrixschreibweise:

$$\begin{pmatrix} (-n_L)^M & \cdots & n_R^M \\ \vdots & \ddots & \vdots \\ -n_L & \cdots & n_R \\ 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} (-n_L)^M & \cdots & -n_L & 1 \\ \vdots & \ddots & \vdots & \vdots \\ n_R^M & \cdots & n_R & 1 \end{pmatrix} \begin{pmatrix} b_M \\ \vdots \\ b_0 \end{pmatrix} = \\ = \begin{pmatrix} (-n_L)^M & \cdots & n_R^M \\ \vdots & \ddots & \vdots \\ -n_L & \cdots & n_R \\ 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} y_{i-n_L} \\ \vdots \\ y_{i+n_R} \end{pmatrix} \quad . \tag{6.10}$$

Man sieht, dass die ersten beiden Matrizen zueinander transponiert sind, mit

$$A = \begin{pmatrix} (-n_L)^M & \cdots & -n_L & 1 \\ \vdots & \ddots & \vdots & \vdots \\ n_R^M & \cdots & n_R & 1 \end{pmatrix} \quad , \qquad A^T = \begin{pmatrix} (-n_L)^M & \cdots & n_R^M \\ \vdots & \ddots & \vdots \\ -n_L & \cdots & n_R \\ 1 & \cdots & 1 \end{pmatrix} \tag{6.11}$$

$$\text{und} \qquad b = \begin{pmatrix} b_M \\ \vdots \\ b_0 \end{pmatrix} \qquad \text{sowie} \qquad y = \begin{pmatrix} y_{i-n_L} \\ \vdots \\ y_{i+n_R} \end{pmatrix} \tag{6.12}$$

vereinfacht sich die Schreibweise deutlich:

$$A^T A\, b = A^T y\,. \tag{6.13}$$

A ist eine Rechteckmatrix mit $M+1$ Spalten und $n_L + n_R + 1$ Zeilen. Jede Rechteckmatrix lässt sich in eine Orthogonalmatrix Q und eine rechte Dreiecksmatrix R aufspalten (QR-Zerlegung):

$$A = Q\,R \qquad \text{und} \qquad A^T = (Q\,R)^T = R^T Q^T\,, \tag{6.14}$$

und mit

$$A^T A = R^T Q^T Q\,R = R^T I\,R = R^T R \tag{6.15}$$

erhält man schließlich

$$R\,b = Q^T y\,. \tag{6.16}$$

Der Polynomwert am Punkt $x_j = x_i(n=0)$ ist b_0, er ergibt sich durch Multiplikation

$$b_0 = \frac{Q^T(M+1,:)}{R(M+1,M+1)}\,y\,. \tag{6.17}$$

Die Matrix A ist unabhängig von den Datenwerten y_j, damit sind dies auch die Matrizen Q und R. Die Polynomwerte ergeben sich durch eine gleitende Multiplikation (Filterung) mir einer festen Filterfunktion F_0, die Filterkoeffizienten

müssen nur einmal berechnet werden

$$F_0 = \frac{Q^T(M+1,:)}{R(M+1,M+1)} = \frac{Q(:,M+1)}{R(M+1,M+1)}\,. \tag{6.18}$$

Entsprechend kann man Ableitungen berechnen, so ist die erste Ableitung am Punkt $x_j = x_i$ durch den Polynomkoeffizienten b_1 gegeben, die Filterkoeffizienten dafür sind

$$F_1 = \frac{Q(:,M) - F_0 R(M,M+1)}{R(M,M)}\,. \tag{6.19}$$

Die konkrete Berechnung der Koeffizienten war zur Zeit der Originalarbeit von Savitzky und Golay noch etwas aufwändig, der größte Teil der Veröffentlichung besteht daher aus den tabellierten Filterkoeffizienten für verschiedene Polynomordnungen und unterschiedliche Anzahl von Stützpunkten. Abbildung 59 zeigt ein Beispiel einer solchen Tabelle.

CONVOLUTES	SMOOTHING		QUADRATIC	CUBIC		A20	A30			
POINTS	25	23	21	19	17	15	13	11	9	7	5
-12	-253										
-11	-138	-42									
-10	-33	-21	-171								
-09	62	-2	-76	-136							
-08	147	15	9	-51	-21						
-07	222	30	84	24	-6	-78					
-06	287	43	149	89	7	-13	-11				
-05	322	54	204	144	18	42	0	-36			
-04	387	63	249	189	27	87	9	9	-21		
-03	422	70	284	224	34	122	16	44	14	-2	
-02	447	75	309	249	39	147	21	69	39	3	-3
-01	462	78	324	264	42	162	24	84	54	6	12
00	467	79	329	269	43	167	25	89	59	7	17
01	462	78	324	264	42	162	24	84	54	6	12
02	447	75	309	249	39	147	21	69	39	3	-3
03	422	70	284	224	34	122	16	44	14	-2	
04	387	63	249	189	27	87	9	9	-21		
05	322	54	204	144	18	42	0	-36			
06	287	43	149	89	7	-13	-11				
07	222	30	84	24	-6	-78					
08	147	15	9	-51	-21						
09	62	-2	-76	-136							
10	-33	-21	-171								
11	-138	-42									
12	-253										
NORM	5175	8059	3059	2261	323	1105	143	429	231	21	35

Abbildung 59: Tabelle der Filterkoeffizienten für einen gleitenden Polynomfit zweiter Ordnung und unterschiedliche Anzahl von Stützpunkten (aus der Originalarbeit [28]).

Das Arbeiten mit solchen Tabellenwerten ist relativ fehleranfällig (schon in der anscheinend computergenerierten obigen Tabelle ist mindestens ein Zahlenwert falsch). Besser ist es, die Filterkoeffizienten dann zu berechnen, wenn sie gebraucht werden. MATLAB kennt die QR-Zerlegung, dafür ist die Funktion `qr` zuständig, man muss mithin nur die Matrix A definieren [29].

Die folgende MATLAB-Funktion berechnet die benötigten Filterkoeffizienten für

die Daten und die erste Ableitung, Polynomordnung M und Zahl der links- und rechtsseitigen Punkte n_L und n_R werden vorgegeben:

```
function [F0,F1,F2] = savgol(nl,nr,M)
    % coefficients for Savitzky-Golay fits
    % F0: smooth curve, F1/F2: derivatives
    % nl/nr: number of left/right points
    % M: polynomial order
A = ones(nl+nr+1,M+1);
for j = M:-1:1,
    A(:,j) = [-nl:nr]'.*A(:,j+1);
end;
[Q,R] = qr(A);
F0 = Q(:,M+1)/R(M+1,M+1);
F1 = (Q(:,M)-F0*R(M,M+1))/R(M,M);
F2 = 2*(Q(:,M-1)-F0*R(M-1,M+1)-F1*R(M-1,M))/R(M-1,M-1);
```

Das folgende Skript zeigt die Anwendung auf ein Datenfeld y:

```
nl = 15;
nr = 15;
M = 6;
F0 = savgol(nl,nr,M);
sgy = filter(F0(nl+nr+1:-1:1),1,y);
sgy(1:nl) = y(1:nl);
sgy(1+nl:end-nr) = sgy(1+nl+nr:end);
sgy(end-nr+1:end) = y(end-nr+1:end);
```

Die Funktion `filter` arbeitet rückwärts gewandt, daher ist das Filter-Array `F0` gespiegelt zu verwenden. Außerdem müssen die Daten um die rechtsseitige Breite des Filters verschoben werden, um wieder zu den ursprünglichen Abszissenwerten zu passen. Zu überlegen ist auch, wie man die Randbereiche behandelt, im obigen Fragment wurden dort die Ausgangsdaten y eingesetzt.

Abbildung 60 zeigt die Anwendung von Savitzky-Golay-Filtern unterschiedlicher Ordnung auf ein verrauschtes Datenfeld im Vergleich zur einfachen Mittelung.

6.2 Interpolation

Interpolationsverfahren werden dann verwendet, wenn man Zwischenwerte zwischen benachbarten Datenwerten benötigt. Dies ist unter anderem dann erforderlich, wenn in verwendeten Eichtabellen der erforderliche Wert nicht exakt repräsentiert ist. So enthalten Tabellen über Thermospannungen von Thermoelementen, die man zur Temperaturmessung verwendet, deren Werte nicht beliebig dicht, sondern mit irgendeiner praktikablen Schrittweite, z. B. in 10-K-Abständen, Werte dazwischen muss man geeignet interpolieren. Oft sind auch Messdaten nicht in der gewünschten Fülle gemessen worden; um dem Betrachter dennoch einen 'vollständigen' Eindruck zu vermitteln, zeichnet man interpolierte Kurven.

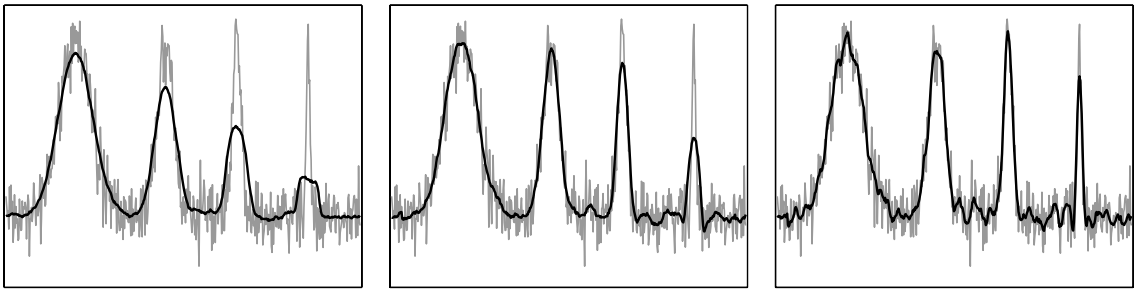


Abbildung 60: Einfache Mittelwertbildung (links) und Savitzky-Golay-Filterung zweiter (Mitte) und sechster Ordnung (rechts) bei Daten mit unterschiedlich breiten Strukturen. Grau: ursprüngliche Daten, schwarz: gefilterte Daten, Filterbreite jeweils gleich ($n_L = n_R = 15$).

Zur Realisierung der Interpolation stellt MATLAB ein Reihe von Funktionen mit jeweils ähnlicher Signatur zur Verfügung.Die Anweisung

```
VI = interpN(X1,..,XN,V,XI1,..,XIN)
```

mit `N = 1,2,3,n` interpoliert die Ausgabewerte `VI` `N`-dimensional zwischen den Tabellenwerten `V`. `V` ist an den Koordinaten `X1..XN` vorgegeben, `VI` wird an den Koordinaten `XI1..XIN` berechnet.

Als weiterer Parameter kann (und sollte) die Interpolationsstrategie angegeben werden, möglich sind die folgenden Verfahren:

`'nearest'`: Zielwert wird auf den Wert des nächstliegenden Tabellenwerts gesetzt. Funktionswerte sind unstetig.

`'linear'`: Lineare Interpolation zwischen benachbarten Werten, die Funktionswerte sind stetig.

`'cubic'`: Kubische Interpolation, Funktionswerte und erste Ableitung stetig.

`'spline'`: Spline-Interpolation, Stetigkeit der Funktionswerte sowie der ersten und zweiten Ableitung.

Die Ergebnisse der verschiedenen Interpolationsverfahren hängen von der Art der zu interpolierenden Funktion bzw. der zu interpolierenden Messdaten ab. Die Abbildungen 61 und 62 veranschaulichen dies an zwei relativ extremen Beispielen: In Abbildung 61 wird eine in den Funktionswerten und Ableitungen stetige Sinusfunktion interpoliert, in Abbildung 62 eine Rechteckfunktion mit ausgeprägten Unstetigkeiten. Die Messwerte (`X,V`) werden durch die Punkte, die interpolierten Werte (`XI,VI`) durch die Kurven repräsentiert.Die interpolierte Funktionswerte wurden mit

```
VI = interp1(X,V,XI,'Strategy');
```

berechnet, für `Strategy` wurden die angegebenen Verfahren eingesetzt.

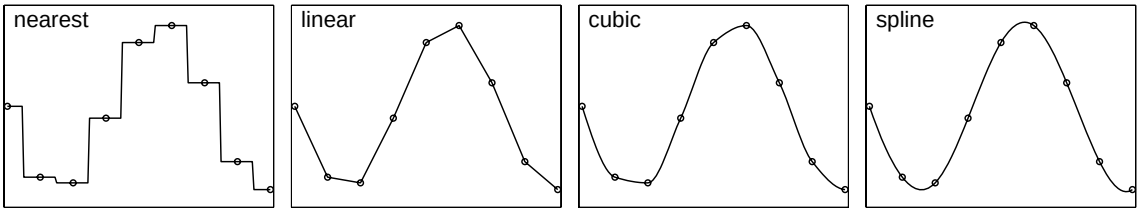


Abbildung 61: Wirkung verschiedener Interpolationsstrategien bei einer stetigen Funktion (Sinus).

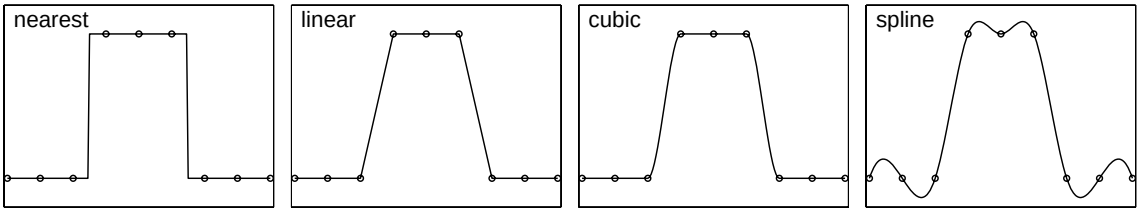


Abbildung 62: Wirkung verschiedener Interpolationsstrategien bei einer Funktion mit Unstetigkeitsstellen (Rechteck).

Der Vergleich zeigt, dass bei einer in den Werten und in den Ableitungen stetigen Funktion (Sinus) die Interpolation mit Splines sehr gut ist. Die kubische Interpolation ist ein guter Kompromiss, wenn Unklarheit über die Stetigkeit herrscht. Die `nearest`-Strategie ist dann anzuwenden, wenn Zwischenwerte nicht sinnvoll, nicht möglich oder nicht erwünscht sind. Dass sich auch die Rechenzeiten für die unterschiedlichen Verfahren beträchtlich voneinander unterscheiden, ist offensichtlich; dies kann insbesondere bei sehr großen Datenfeldern ein zusätzliches Kriterium sein.

6.3 Fouriertransformation

Messdaten, insbesondere elektrische Signale, werden meist in Abhängigkeit von der Zeit gemessen. Periodizitäten darin zeigen charakteristische Frequenzen an. Einem zeitabhängig streng periodischen Verlauf entspricht eine einzelne Frequenz, ein Punkt im Frequenzspektrum. Umgekehrt entspricht einem zeitlich einmaligen Ereignis ein Frequenzkontinuum. Die beiden Koordinaten – Zeit und Frequenz – sind zueinander komplementär.[24] Zwischen solchen komplementären Darstellungen vermittelt die Fouriertransformation. Naturgemäß spielt bei Messdaten nur die endliche diskrete Fouriertransformation eine Rolle. Zwischen den beiden komplementären Bereichen kann man Daten beliebig hin und her transformieren, jeweils die für einen bestimmten Zweck sinnvollere oder nützlichere Darstellungsform wählen.

Zur Berechnung der diskreten Fouriertransformation enthält MATLAB die Funktionen `fft`, `fft2` und `fftn`, sowie deren inverse `ifft`, `ifft2` und `ifftn`

[24] Ein ähnlich komplementäres Paar in der Physik ist beispielsweise Ort und Raumfrequenz.

zur Rücktransformation. Die eindimensionalen Formen berechnen die Transformation in einer Dimension, auch wenn die Variable eine Matrix ist (spaltenweise bzw. in der ersten nicht singulären Dimension). Die zwei- und die mehrdimensionalen Formen rechnen entsprechend in zwei oder allen Dimensionen.

Zu den Anwendungsbereichen der Fouriertransformation gehört die Frequenzanalyse und die Datenfilterung, dazu je ein Beispiel.

6.3.1 Frequenzanalyse

Die Fouriertransformierten von streng periodischen Funktionen sind Deltafunktionen im dazu jeweils reziproken Raum. Einer zeitlich periodische Funktion entspricht mithin ein Punkt im (komplexen) Frequenzraum. Es liegt daher nahe, periodische Messdaten zu transformieren und in ihrer Frequenzdarstellung zu analysieren. Abbildung 63 zeigt als Beispiel die zu zwei verschiedenen Wähltasten – 1 und 5 – gehörenden Töne eines Telefons[25]. In der zeitlichen Auftragung ist der Unterschied nur schwer zu erkennen (linkes Bild), nach der Fouriertransformation wird die Analyse einfach (rechtes Bild).

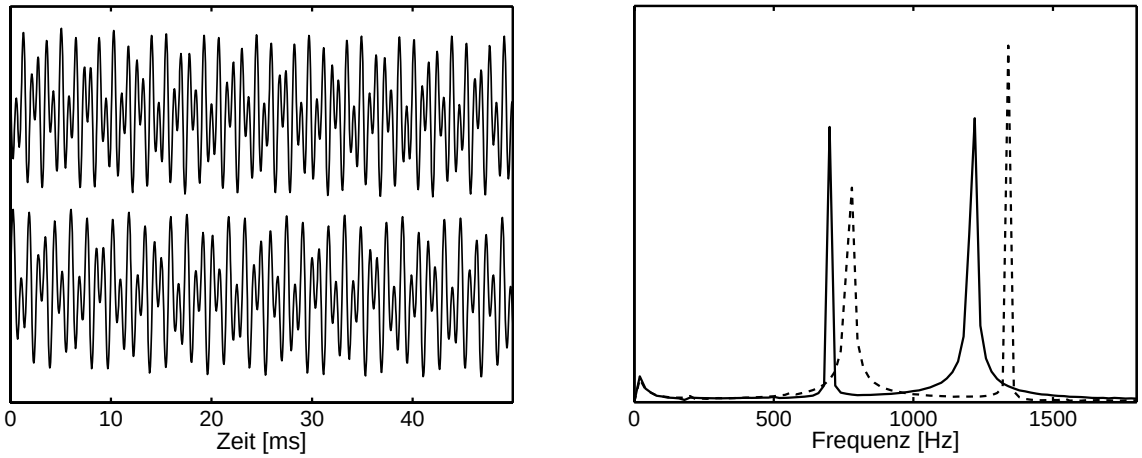


Abbildung 63: Wähltöne 1 *und* 5 *am Telefon: Links die zeitliche Auftragung der Amplituden (Oszilloskop), unten* 1*, oben* 5*, rechts die Absolutwerte der Fouriertransformierten, durchgehende Linie:* 1*, gestrichelt:* 5*.*

Wie zu erwarten sind keine idealen Deltafunktionen entstanden. Die Breite der Spektrallinien entspricht der reziproken Messzeit, die hier auf 50 ms begrenzte Messzeit verursacht eine Breite von 20 Hz in der Frequenzauftragung. Ideale Deltafunktionen wären nur durch unendlich lange Messzeiten zu erreichen.

Ein zweiter bei der Fouriertransormation auftretender Effekt ist in Abbildung 63 ebenfalls erkennbar: unterschiedliche Linienformen, Restintensität neben den

[25] Bei der Tastenmatrix am Telefon sind Reihen und Spalten mit festen Frequenzen kodiert, die Reihen mit 697, 770, 852 und 941 Hz, die Spalten mit 1209, 1336 und 1477 Hz. Beim Tastendruck werden die beiden zugehörigen Frequenzen als Ton übertragen.

Linien. Hier zwar nicht sonderlich störend, da die Intensitäten der Messsignale deutlich größer sind, kann diese Untergrundintensität dann Probleme machen, wenn Signale analysiert werden sollen, die verschiedene Frequenzanteile mit sehr unterschiedlichen Intensitäten enthalten. Der Untergrund wird von dem stufenartigen Verlauf der Messdaten am Beginn und Ende der Messung hervorgerufen, die zusätzliche Frequenzen im Spektrum verursachen. Man vermeidet den Effekt bei der Fourieranalyse dadurch, dass man das zu analysierende Signal zuvor mit einer geeigneten Fensterfunktion multipliziert, die an den Endpunkten ganz oder nahezu verschwindet. Zur Realisierung verwendet man u. a. Dreiecks-, Trapez-, Cosinus-, Parabel- oder Gaußfunktionen. Das Verfahren hat als Nebeneffekt immer eine leichte Verbreiterung der Strukturen im Spektrum zur Folge, da durch die geringere Gewichtung der Randpunkte die effektive Messzeit verringert wird.

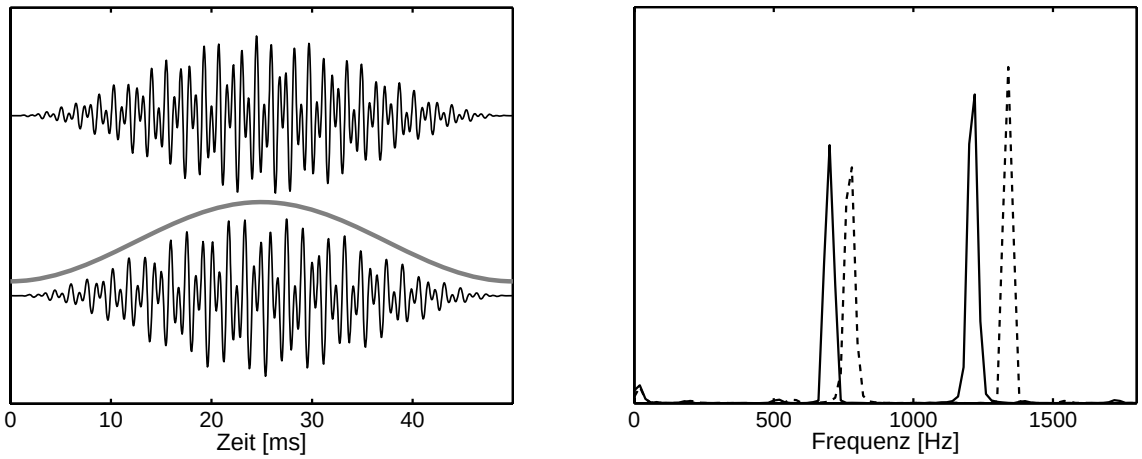


Abbildung 64: Wirkung einer Fensterfunktion bei der Fouriertransformation: Links die mit der cosinusförmigen Funktion multiplizierten Messdaten, rechts die untergrundfreien Spektren.

Die Anwendung auf das vorliegende Beispiel zeigt Abbildung 64. Es wird eine cosinusförmige Fensterfunktion verwendet, im linken Bild als graue Kurve dargestellt. Die Wirkung ist deutlich, im Spektrum ist keine Untergrundintensität mehr enthalten, die Form der Spektrallinien ist nahezu identisch geworden.

Macht man die Fensterfunktion schmaler, verkürzt damit die effektive Messzeit, so verliert man Information, die spektralen Strukturen werden breiter (Abbildung 65).

6.3.2 Datenfilterung

Außer zur Analyse von Messdaten kann man die Fouriertransformation auch dazu verwenden, Messdaten gezielt zu verändern. Ein Anwendungsbereich ist die Filterung von Daten, sofern es darum geht, Signale bestimmter Frequenzen oder ganzer Frequenzbereiche in Messdaten zu manipulieren. Man transformiert

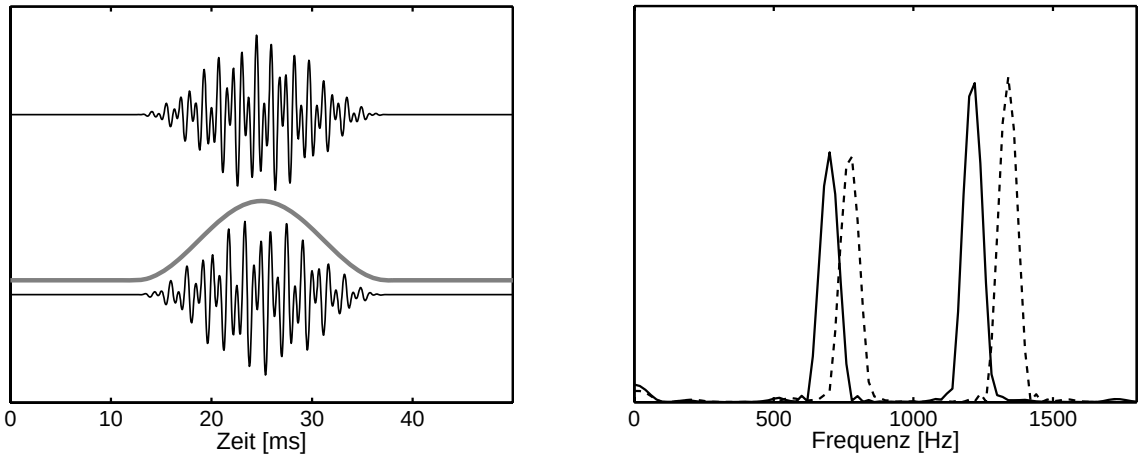


Abbildung 65: Verkürzung der effektiven Messzeit durch eine schmalere Fensterfunktion, die Spektrallinien verbreitern sich.

dazu den gesamten Datensatz, macht im Fourierspektrum die gewünschten Veränderungen und transformiert anschließend mit der inversen Fouriertransformation wieder zurück.

Die Abbildung 66 zeigt als idealisiertes Anwendungsbeispiel Messdaten, denen eine intensive periodische Störung überlagert ist[26]. Eine herkömmliche Filterung (vgl. 6.1) wirkt erst dann zufriedenstellend, wenn eine relativ breite Filterfunktion verwendet wird. Dann sind aber auch die Strukturen in den Messdaten deutlich verbreitert.

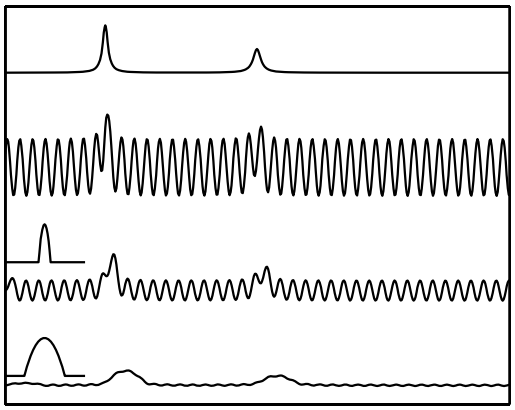


Abbildung 66: Messdaten mit einer überlagerten periodischen Störung: oben die ungestörten Daten, darunter mit Störung. Die beiden unteren Kurven zeigen das Ergebnis der in Kapitel 6.1 beschriebenen herkömmlichen Filterung der Daten mit unterschiedlich breiten Filterfunktionen (vgl. auch Abbildung 55).

Abbildung 67 zeigt die Wirkung einer Fourierfilterung. Vor der Fouriertransformation werden die Messdaten mit einer Fensterfunktion multipliziert, im linken Bild als graue Kurve dargestellt. In diesem Fall wird eine Trapezfunktion verwendet, die so gewählt wird, dass sie im interessierenden Bereich der Messdaten

[26] Störungen dieser Art können beispielsweise durch die Netzwechselspannung (50 oder 100 Hz) verursacht werden. Die Frequenz muss dabei nicht unbedingt der Störfrequenz entsprechen, sondern kann durch Schwebungseffekte zwischen Messtakt und Störfrequenz auch in völlig andere Bereiche verschoben sein.

konstant ist. Dadurch ist gewährleistet, dass die Intensitätsverhältnisse dort die Transformationen ungeändert überstehen.

Das rechte Bild (obere Kurve) zeigt das Ergebnis der Fouriertransformation (`fft`), die Störfrequenz wird durch die beiden intensiven Linien repräsentiert. Zur Unterdrückung der Störung wird im Fourierraum mit einer Funktion multipliziert, die zwischen 0 (im Bereich der Linien) und 1 (außerhalb davon) variiert (graue Kurve im rechten Bild). Auch hier wird eine trapezartige Funktion gewählt, damit keine abrupten Übergänge entstehen. Diese würden bei der Rücktransformation zu störenden Artefakten führen. Die untere Kurve im rechten Bild zeigt das Ergebnis der Multiplikation. Die Rücktransformation (inverse Funktion `ifft`) ergibt dann ein fast störungsfreies Ergebnis (untere Kurve im linken Bild). Die

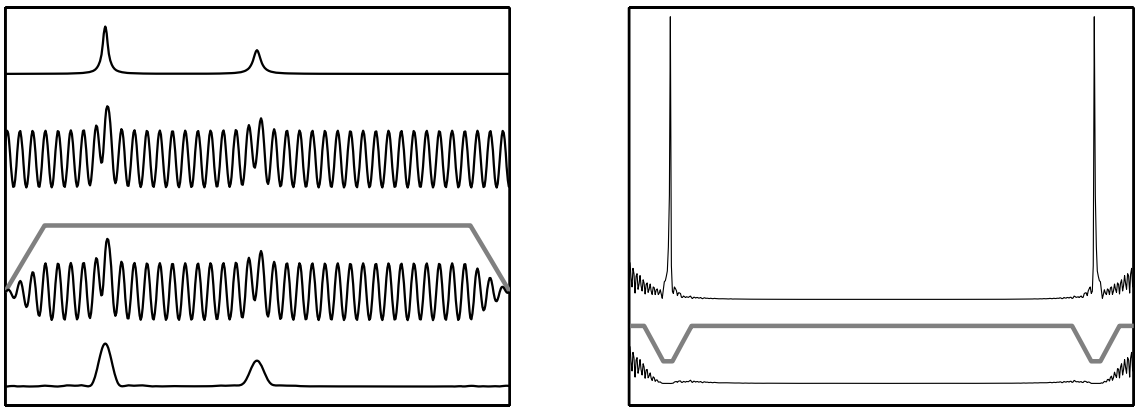


Abbildung 67: Fourierfilterung von Messdaten mit periodischer Störung. Links oben die Daten ohne und mit Störung, darunter die verwendete Fensterfunktion und die damit multiplizierten Daten. Rechts die Fouriertransformierte und die Filterfunktion, die die Störfrequenz unterdrückt. Rechts unten die damit multiplizierte Fouriertransformierte, links unten das Ergebnis der Rücktransformation.

Strukturen in den Messdaten sind etwas breiter geworden; das lässt sich nicht verhindern, da durch die Manipulation im Fourierraum auch die Bandbreite des Nutzsignals verringert wird.

6.4 Fits, Anpassung an Funktionen

Vermutet man, dass sich Messdaten durch einen mathematischen Zusammenhang beschreiben lassen, dann interessieren die Parameter, die die bekannte oder vermutete funktionale Abhängigkeit bestimmen. In diesen Parametern steckt meist die physikalische Eigenschaft, die man messen will. So kann man beispielsweise die Federkonstante einer Feder dadurch bestimmen, dass man den linearen Zusammenhang zwischen Kraft und Auslenkung misst.

Man berechnet diese Parameter dadurch, dass man die Abweichung zwischen Daten und Funktionszusammenhang möglichst klein macht (man minimalisiert

die Summe der Quadrate der Differenzen – *least squares fit*). Besonders einfach ist das, wenn die vermutete Funktion ein Polynom ist oder wenn die Anpassparameter in der Funktion nur linear auftreten. Dann lässt sich das Fit-Problem exakt lösen, das Minimum ergibt sich als Lösung eines linearen Gleichungssystems. Ansonsten muss man mit geeigneten Iterationsverfahren arbeiten.

6.4.1 Polynome

Für Polynomanpassungen ist in MATLAB die Funktion `polyfit` zuständig. Der Aufruf

```
P = polyfit(x, y, N);
```

liefert im Vektor `P` die Koeffizienten des Regressionspolynoms `N`-ter Ordnung für den Datensatz `x,y`.

```
[P, S] = polyfit(x, y, N);
```

definiert eine zusätzliche Struktur `S` mit weiteren Informationen, die beispielsweise für die Fehlerabschätzung nützlich sind.

```
Yp = polyval(P, Xp);
```

berechnet die Polynomwerte an den Punkten `Xp`,

```
[Yp, dYp] = polyval(P, Xp, S);
```

zusätzlich Fehlergrenzen, innerhalb derer bei Daten mit einigermaßen zufälligem Fehler mindestens 50 % der Datenwerte liegen. Abbildung 68 zeigt als Beispiel für einen Polynomfit den einfachsten Fall, die Anpassung an eine lineare Abhängigkeit (`polyfit(x,y,1)`).

6.4.2 Parameterlineare Fits

Mit MATLAB lassen sich die Anpassparameter auch immer dann besonders einfach bestimmen, wenn sie nur linear in der funktionalen Abhängigkeit enthalten sind, mit der die Daten gefittet werden sollen. Wenn also etwa der über x gemessene Datensatz y mit der Funktion $f(x)$ gefittet werden soll, die Anpassparameter a_i nur in linearer Form enthält:

$$y \approx f(x) = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_n f_n(x) \quad . \tag{6.20}$$

Die Teilfunktionen $f_i(x)$ können darin beliebig aufwändig sein.[27]

Die Messdaten x und y sind 2 Vektoren, die je m Elemente enthalten. Da beide bekannt (gemessen) sind, kann man Gleichung 6.20 als ein lineares Gleichungssystem für die a_i auffassen. Für $m > n$, den Regelfall bei der Anpassung von Messdaten, ist das Gleichungssystem allerdings überbestimmt.

[27] Polynome sind ein einfacher Spezialfall mit $f_1(x) = x^0$, $f_2(x) = x^1$ usw.

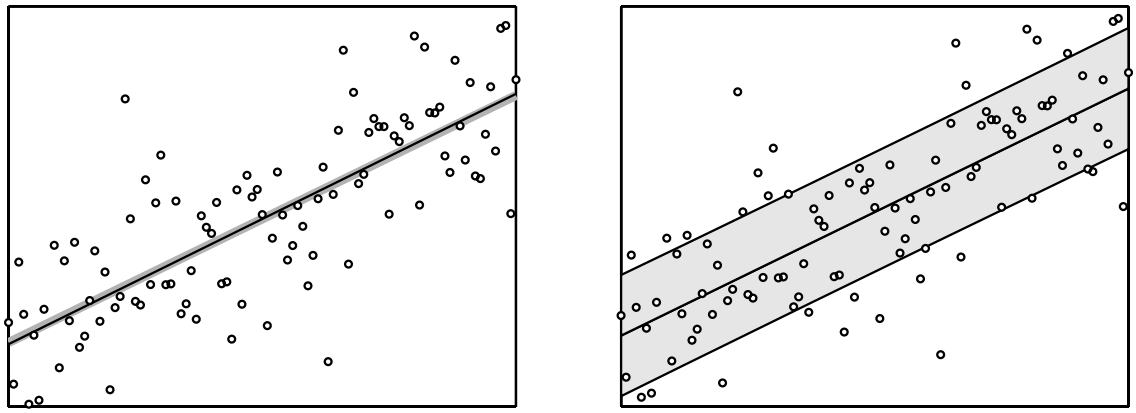


Abbildung 68: Regressionsgerade an zufallsverteilte Messdaten. Die Berechnung erfolgte mit `polyfit`*. Im linken Bild zeigt die schwarze Gerade die Ausgangsfunktion, die dann mit* `randn` *gestört wurde (Punkte), grau die Fitgerade (die leichte Abweichung deutet darauf hin, dass die Zufallszahlen nicht ganz ideal sind). Im rechten Bild ist der Fehlerbereich* `Yp-dYp...Yp+dYp` *grau hinterlegt, es wird deutlich, dass mehr als 50 % der Datenpunkte in diesem Bereich liegen.*

Zur Lösung linearer Gleichungssysteme hat MATLAB die Funktion `mldivide`, 'left matrix division', die verkürzt auch mit '\' (Backslash) geschrieben werden kann. Die gleiche formale Schreibweise kann auch verwendet werden, wenn das Gleichungssystem überbestimmt ist. MATLAB errechnet dann eine Näherungslösung, in der die quadratischen Abweichungen minimiert sind (least squares fit). Das ist genau das, was man benötigt.

Die Parameter a_i in Gleichung 6.20 werden berechnet durch:

$$F = [f_1(x), f_2(x), \ldots f_n(x)]; \tag{6.21}$$

$$a = F\backslash y; \quad . \tag{6.22}$$

Das Ergebnis a ist ein Vektor mit den Parametern a_i.

Abbildung 69 zeigt ein Beispiel aus dem Bereich der nichtlinearen Optik, eine neue Substanz wurde mit Laserimpulsen unterschiedlicher Leistung beleuchtet, das erzeugte frequenzverdoppelte Licht wurde gemessen. Man erwartet einen quadratischen Zusammenhang zwischen den beiden Intensitäten $I_{2\omega}$ und I_ω:

$$I_{2\omega} = d_{\text{eff}}^2 G I_\omega^2 \quad . \tag{6.23}$$

Neben einem Geometriefaktor G enthält der Ausdruck die wirksame Nichtlinearität d_{eff}, die interessierende physikalische Größe. Die Berechnung der Fitparameter erledigt das MATLAB-Skript

```
data = load('p15_2.int');
x = data(:,1);
y = data(:,2);
F = [ones(size(x)), x.*x];
a = F\y; .
```

Zunächst werden die Messdaten gelesen, die als Tabelle in der Datei `p15_2.int` abgelegt sind, daraus werden die `x`- und `y`-Werte extrahiert. Die Funktion `F` besteht aus dem quadratischen Anteil und einer zusätzlichen Konstanten, die eine eventuell vorhandene konstante Untergrundintensität berücksichtigt. Dazu ist der mit Einsen besetzte Vektor gleicher Größe nötig. In Abbildung 69 sind Messwerte und Anpassungskurve zusammen geplottet.

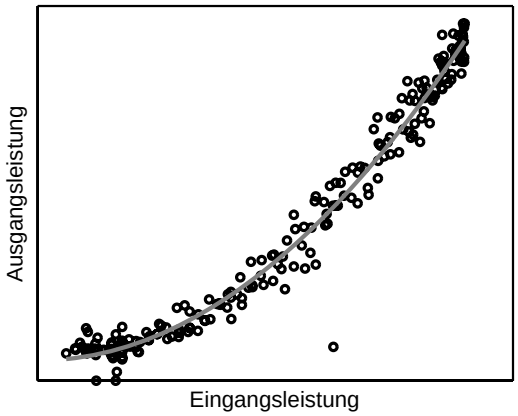


Abbildung 69: Zusammenhang zwischen Laserleistung und frequenzverdoppelter Intensität an einer nichtlinearen Probe, Messwerte (○) und quadratische Anpassung (grau).

6.4.3 Anpassung an beliebige Funktionen

Soll eine Funktion angepasst werden, die die Fitparameter in allgemeiner Form enthält, kann das durch eine Minimalwertsuche mit der MATLAB-Funktion `fminsearch` erledigt werden. Diese Funktion wendet den Simplex-Algorithmus an, um ein Minimum zu finden. Sie wird aufgerufen mit

```
a = fminsearch(f, a0, options, p1, p2, ...) .
```

Darin ist `a` das Ergebnis, die berechneten Parameter, `f` die zu minimierende Funktion, `a0` Startwerte[28] für `a`, `options` einstellbare Optionen wie Genauigkeit oder Maximalzahl der Iterationen, `pn` weitere Variable, die an `f` übergeben werden. Die Funktion `f` hat die Form

```
d = f(a, p1, p2, ...) .
```

Bei der Bestimmung von Fitkurven berechnet man in `f` zum Beispiel die Summe der quadrierten Differenzen zwischen Messwerten und zugehörigen Funktionswerten der Fitfunktion.

Zur Veranschaulichung wieder ein einfaches Beispiel aus der Optik. Die Abhängigkeit des Brechungsindex von der Wellenlänge kann mit guter Genauigkeit durch eine Summe von Resonanztermen ausgedrückt werden:

$$n(\lambda)^2 = 1 + \sum_i \frac{A_i}{\lambda_i^{-2} - \lambda^{-2}} \quad . \tag{6.24}$$

[28] Beim verwendeten Simplex-Verfahren ist es sehr wichtig, sinnvolle Startwerte vorzugeben, da der Algorithmus auch nach Auffinden eines *lokalen* Minimums abbricht.

Für diese Terme sind jeweils Absorptionsübergänge (Elektronen, Phononen etc.) bei den Wellenlängen λ_i verantwortlich. Im sichtbaren Spektralbereich genügt es oft, nur *einen* derartigen – im Ultravioletten liegenden – Term zu berücksichtigen.

Im vorliegenden Beispiel wurden die Brechungsindizes von Benzophenon, einem optisch anisotropen Material, für die 3 Hauptpolarisationsrichtungen bei verschiedenen Wellenlängen gemessen. Die Wellenlängen waren durch die verwendeten Spektrallampen vorgegeben. Brechungsindizes in den Bereichen dazwischen könnte man durch geeignete Interpolation erhalten, sehr viel besser ist es jedoch, den durch physikalische Modellvorstellungen begründeten Funktionsverlauf anzupassen, der durch ganz wenige Parameter (in diesem Fall jeweils ein Resonanzterm mit zwei Parametern für jede der drei Polarisationsrichtungen) beschrieben werden kann.

Das MATLAB-Skript für die Anpassungsrechnung:

```
function para = coeff(fname)
start = [ 1e14, 2e-7 ];
options = optimset('TolX', 1e-10, 'TolFun', 1e-10, ...
                   'MaxIter', 1e4, 'MaxFunEvals', 1e4);
data = load(fname);
para = fminsearch('devsum', start, options, data(:,1), data(:,2)); .
```

Der Minimalisierungsalgorithmus wird mit Startwerten und Optionen aufgerufen, die Optionen werden zuvor mit `optimset` festgelegt (Toleranzen und Maximalzahl der Iterationen). An die zu minimierende Funktion `devsum` werden die Messdaten aus der Datei `fname` weitergereicht, Wellenlängen und zugehörige Brechungsindizes.

Die Funktion `devsum` berechnet die Summe der quadrierten Abstände der Messwerte von der jeweils aktuellen Fitkurve:

```
function f = devsum(para, lambda, nexp)
f = sum((nexp - nfit(para, lambda)).^2); .
```

Die Fitkurve selbst wird in einer weiteren Funktion berechnet, um sie auch für andere Zwecke (Graphik) verwenden zu können:

```
function n = nfit(para, lambda)
n = sqrt(1+para(1)./((para(2))^(-2)-lambda.^(-2))); .
```

Die errechneten Fitkurven sind zusammen mit den Messwerten in Abbildung 70 dargestellt. Deutlich wird hierin auch ein weiterer Vorteil von (wenigparametrigen) Fitkurven gegenüber einer Interpolation: Kleine statistische Messfehler werden weitgehend ausgemittelt.

6.5 Graphische Darstellung

Obwohl MATLAB in erster Linie für Anwendungen in der Numerik konzipiert ist, bietet es auch eine Fülle von Visualisierungsfunktionen, die zur Erstellung von Graphiken eingesetzt werden können. Darunter sind 2D-Funktionen wie `plot`

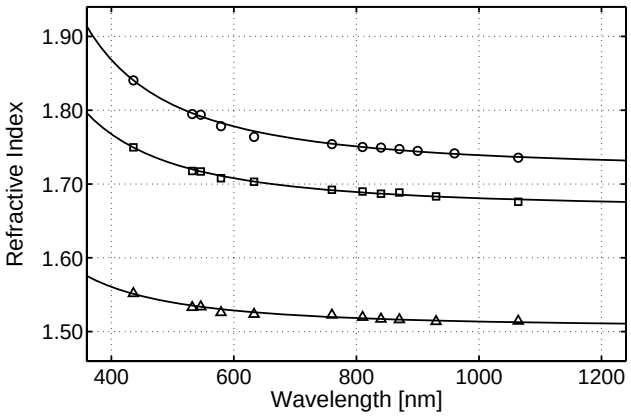


Abbildung 70: Brechungsindizes von Benzophenon, oben n_a, in der Mitte n_b, unten n_c. Die Punkte repräsentieren die Messwerte, Linien die Fitkurven.

für lineare 2D-Plots, `loglog` für doppelt logarithmische oder `semilogx` und `semilogy` für halblogarithmische Darstellung und 3D-Funktionen wie `plot3` für dreidimensionale Linienplots oder `surf` für Flächendarstellungen.

Alle Graphik-Objekte sind in MATLAB in einer hierarchischen Objekt-Struktur angeordnet, die einzelnen Objekte sind über ihre *Handles*[29] zugänglich. Alle Objekteigenschaften können während der Erstellung oder nachträglich mit Hilfe der *Handles* fast beliebig eingestellt werden. So wurde beispielsweise die Kurve und die Punkte für n_a in Abbildung 70 konfiguriert mit:

```
hfita = plot(L, na, 'k-');
set(hfita, 'LineWidth', 1.5);
hold on;
data = load('n_a.dat');
hna = plot(1e9*data(:,1), data(:,2), 'ko');
set(hna, 'LineWidth', 1.5);
set(hna, 'MarkerSize', 7)
```

Darin sind die ersten beiden Parameter in den `plot`-Anweisungen jeweils die Daten, `'k-'` ordnet eine durchgezogene schwarze (blac*k*) Linie an, `'ko'` schwarze kreisförmige Marker. Mit `set` werden anschließend weitere Eigenschaften festgelegt.

Eine Einführung in die Graphik-Möglichkeiten von MATLAB gibt das Skriptum zu Vorlesung 'Graphik-Workshop' [26], eine Fülle von Informationen findet man außerdem im MATLAB-Hilfesystem und in der Online-Hilfe unter den Einstiegspunkten `help graphics`, `help graph2d` und `help graph3d`.

[29] 'Handle Graphics' ist ein eingetragenes Warenzeichen von *The MathWorks Inc.*

7 Schnittstellen und Programmierung

Die Kopplung zwischen Experiment und Steuerungs- bzw. Auswerterechner wird im allgemeinen durch Schnittstellen (*Interfaces*) vermittelt – Standardschnittstellen, die an praktisch jedem Rechner gleichartig vorhanden sind, oder aber speziellen Schnittstellenkarten, die für bestimmte Aufgaben gebaut sind. Eigenschaften und Progammierung der Rechnerschnittstellen sind Gegenstand dieses Kapitels, das Schwergewicht liegt dabei auf den Standardschnittstellen. Die Programmbeispiele werden in MATLAB, C/C++ oder Java erstellt.

> Programmiersprachen sind keine Weltanschauungen sondern Werkzeuge;
> es gibt geeignete und ungeeignete und man darf sie ungestraft wechseln.

Wichtig im Rahmen von Steuer- und Messaufgaben ist oft auch ein gut definierter zeitlicher Ablauf, daher zunächst eine kurze Diskussion der Zeitsteuerungsmöglichkeiten in einem Betriebssystem wie Windows.

7.1 Zeit und Windows

Windows ist kein 'Echtzeit'-Betriebssystem, daher zur exakten zeitlichen Steuerung eines Ablaufs im Prinzip nicht geeignet. Wenn man aber kleinere Ungenauigkeiten in Kauf nehmen kann, sind die Windows-Funktionen zur Zeitmessung und Zeittaktsteuerung recht vielseitig verwendbar.

7.1.1 Systemzeit

Die Rechneruhr wird unter Windows im Abstand von 10 ms inkrementiert, mit dieser Auflösung und Genauigkeit arbeiten auch alle Zeitfunktionen, die darauf zugreifen (auch wenn die Zeitangaben in Millisekunden sind). Unter anderem sind dies:

`Sleep` – setzt die Programmausführung für die als Parameter (in Millisekunden) angegebene Zeit aus.

`clock` – berechnet die seit dem Programmstart vergangenen Zeittakte. Die formale Auflösung ist durch die Konstante `CLOCKS_PER_SEC` gegeben, die reale Auflösung ist die der Rechneruhr. Die Zeit in Sekunden ergibt sich durch Division von `clock` durch `CLOCKS_PER_SEC`.

`GetTickCount` – liefert die seit dem Start von Windows vergangene Zeit (in Millisekunden) als 32-Bit-Wert (`DWORD`)[30].

[30] Bei Rechnersystemen, die länger in Betrieb sind, muss gegebenenfalls berücksichtigt werden, dass der Millisekunden-Zähler etwa alle 50 Tage überläuft und wieder bei Null anfängt.

7.1.2 Zeitmessung in MATLAB

Dazu äquivalente Funktionen sind auch in MATLAB verfügbar, ebenfalls – abgesehen von `tic, toc` – mit einer Auflösung von 10 ms:

`pause` – wartet für die (diesmal in Sekunden anzugebende) Zeit.

`clock` – liefert Datum und Zeit als Vektor mit 6 Elementen `[Jahr Monat Tag Stunden Minuten Sekunden]`.

`etime` – berechnet die Differenz aus zwei `clock`-Werten.

`tic, toc` – starten und stoppen eine Zeitmessung, `toc` liefert die Zeitdifferenz zum vorhergehenden `tic`. Die Genauigkeit wurde vor einigen Jahren deutlich verbessert (zumindest unter Windows), sie liegt nun im Mikrosekunden-Bereich.

`cputime` – liefert die Zeit (in Sekunden) seit dem Start von MATLAB.

7.1.3 Performance-Counter

Zur genaueren Zeitmessung kann unter Windows ein spezieller Zähler verwendet werden, der ein hochfrequentes Taktsignal zählt. Dieser Zähler wird mit der Funktion

```
QueryPerformanceCounter(LARGE_INTEGER * N)
```

ausgelesen, die Frequenz des zugehörigen Taktsignals kann man mit

```
QueryPerformanceFrequency(LARGE_INTEGER * F)
```

erfragen. Ist kein entsprechender Zähler in der Rechner-Hardware vorgesehen, wird als Frequenz 0 gemeldet. Typischerweise liegen die verwendeten Frequenzen zwischen 1 MHz und der Prozessortaktfrequenz. Mithin sind Zeitmessungen mit Mikrosekundengenauigkeit oder besser möglich.

Die Messung einer Zeitdifferenz `delta` mit dem Performance Counter veranschaulicht das nachstehende Programmfragment:

```
LARGE_INTEGER f, n1, n2;
QueryPerformanceFrequency(&f);
QueryPerformanceCounter(&n1);
...
QueryPerformanceCounter(&n2);
double delta = (double) (n2.QuadPart-n1.QuadPart)/f.QuadPart; .
```

Will man nicht die Differenz der Ablesezeitpunkte messen, sondern nur die dazwischen liegende Zeit, beispielsweise um die Ausführungszeit von Programmcode zu bestimmen, dann sollte man noch mit einer Leermessung korrigieren. Eine Beispielanwendung für den Performance-Counter ist in Anhang A gelistet. Dort wird der Performance-Counter dafür verwendet, die Steuerimpulse für einen Servo zwischen 0.9 und 2.1 ms zu variieren (vgl. Abschnitt 2.4).

7.1.4 Timer

Prozesse unter Windows können sich Taktgeber (Timer) einrichten, die in festem zeitlichen Abstand entweder eine dafür bereitgestellte Funktion aufrufen oder an das zugehörige Fenster die `WM_TIMER`-Nachricht verschicken. Sie sind immer an ein Fenster-Objekt (Window) gebunden, können daher nicht in reinen Konsolprogrammen eingerichtet werden. Timer werden mit `SetTimer()` eingerichtet, mit `KillTimer()` wieder entfernt. Die Zeitauflösung und damit auch die minimal einstellbare Taktrate beträgt derzeit 10 msec. Nachstehendes Programmfragment verdeutlicht die Verwendung:

```
static UINT MyTimer = 1;
const UINT Elapse = 100;                              // timeout 100 msec
...
void CALLBACK TiProc (HWND hWnd, UINT msg, UINT timer, DWORD systime)
{ Do what you want to do every 100 msec }
...
MyTimer = SetTimer (hWnd, MyTimer, Elapse, TiProc); // install timer
...
KillTimer (hWnd, MyTimer);                            // deinstall timer
```

7.1.5 Timer in MATLAB

Seit der Version 6.5 (Release 13) stehen auch in MATLAB Timer-Objekte zur Verfügung, die mit dem Aufruf

```
t = timer('TimerFcn', @mycallback, 'Period', 10.0);
```

eingerichtet werden. Eigenschaften des Timer-Objekts werden direkt bei der Einrichtung festgelegt oder später – MATLAB-üblich – entweder mit

```
set(t,'Eigenschaft1',Wert1,'Eigenschaft2',Wert2,...);
```

oder mit

```
t.Eigenschaft = Wert; .
```

`get(t)` und `set(t)` zeigen die aktuellen und die möglichen Werte für die Objekteigenschaften an.

Die Zeitauflösung entspricht der des umgebenden Betriebssystems, bei Windows derzeit 0.01 s.

Die Verwendung von *Callback*-Funktionen ist sehr flexibel, neben der üblichen *TimerFcn* können Funktionen zugewiesen werden, die beim Starten (*StartFcn*), Stoppen (*StopFcn*) oder Fehlern (*ErrorFcn*) des Timers aufgerufen werden. Nachdem ein Timer-Objekt mit allen Eigenschaften eingerichtet ist, wird es mit `start(t)` gestartet, mit `stop(t)` wieder angehalten. Weitere Informationen dazu in der MATLAB-Hilfe.

7.1.6 Multimedia-Timer

Für Anwendungen, bei denen die Zeitauflösung der 'normalen' System-Timer nicht ausreicht, stellt Windows einen weiteren Timer-Typ bereit, den Multimedia-Timer mit einer Auflösung von 1 ms. Neben der höheren Auflösung hat dieser Timertyp den Vorteil, nicht an Fenster-Objekte gebunden zu sein, er kann somit auch in Konsolprogrammen, fensterlosen DLLs (MEX) u. ä. verwendet werden. Gestartet wird mit

```
MMRESULT timeSetEvent( UINT uDelay, UINT uResolution,
                       LPTIMECALLBACK lpTimeProc,
                       DWORD dwUser, UINT fuEvent );
```

`uDelay` ist die Verzögerung (Taktrate) in Millisekunden, `uResolution` die Auflösung (Genauigkeit), `lpTimeProc` die Callback-Funktion, die vom Timer aufgerufen werden soll, `dwUser` Daten, die an die aufgerufene Funktion übergeben werden, `fuEvent` legt fest, ob der Timer einmalig (`TIME_ONESHOT`) oder periodisch arbeiten soll (`TIME_PERIODIC`).

Der Rückgabewert identifiziert den Timer und wird zum Stoppen des Timers mit

```
MMRESULT timeKillEvent( UINT uTimerID );
```

benötigt (`uTimerID`).

Die Callback-Funktion, die vom Multimedia-Timer aufgerufen wird, muss dem folgenden Prototyp entsprechend implementiert werden

```
void CALLBACK TimeProc( UINT uID, UINT uMsg,
                        DWORD dwUser, DWORD dw1, DWORD dw2 );
```

darin identifiziert `uID` den Timer und `dwUser` enthält die beim Timer-Start übergebenen Daten, die übrigen Parameter sind reserviert oder undefiniert.

Mit

```
MMRESULT timeGetDevCaps( LPTIMECAPS ptc, UINT cbtc );
```

können die Fähigkeiten (minimale und maximale Zeit) der Multimedia-Timers erfragt werden.

Ein Beispielprogramm, in dem Funktionen des Multimedia-Timers verwendet werden, ist in Anhang A gelistet. Der Multimedia-Timer wird dort dazu verwendet, eine 50-Hz-Impulsfolge zum Betrieb eines Servos zu generieren.

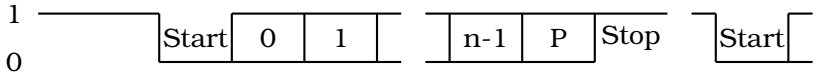
7.2 Die serielle Schnittstelle

Serielle Verbindungen sind die vom Leitungsaufwand her einfachsten genormten 2-Punkt-Verbindungen in der DV-Technik. Ursprünglich sind sie für langsame, zeichenorientierte Datenübertragung zwischen Fernschreibern oder zwischen Rechner und Terminal konzipiert. Am PC ist meist zumindest eine serielle Schnittstelle vorhanden, an die Peripheriegeräte wie Maus, Modem, Plotter, (serielle) Drucker, Datenerfassungsgeräte oder auch ein anderer Rechner angeschlossen werden können.

7.2.1 Grundlagen und Schnittstellennorm

Damit Geräte verschiedener Hersteller ohne weitere Anpassungsarbeit miteinander kommunizieren können, müssen alle relevanten Parameter herstellerübergreifend festgelegt sein. Für serielle Verbindungen sind mehrere Normen definiert, die sich in der Hardwareauslegung unterscheiden. Je nach Norm sind unterschiedliche Maximalentfernungen und Maximalübertragungsgeschwindigkeiten möglich. In der zur Zeit gebräuchlichsten Norm (USA: RS 232 C, D: DIN 66020, 66022, Europa: CCITT V24)[31], die auch bei der PC-Schnittstelle verwendet wird, sind unter anderem die folgenden Parameter vereinbart:

- Der High-Pegel einer Leitung liegt zwischen +3 und +15 Volt, der Low-Pegel zwischen -3 und -15 Volt.
- Daten werden in negativer Logik (1 = Low-Pegel), Steuersignale in positiver Logik (True, aktiv = High-Pegel) übertragen.
- Der Ruhezustand der Datenleitung ist logisch 1. Bei der asynchronen Übertragung werden Zeichen einzeln übertragen, die einzelnen Bits eines Zeichens in Folge. Jedes Bit hat die gleiche zeitliche Länge. Die Übertragung wird durch ein auf logisch 0 gesetztes Bit, das Startbit, eingeleitet, dann folgen die Datenbits in steigender Wertigkeit, danach eventuell ein Paritätsbit P, am Ende mindestens 1–2 Bits logisch 1 (Stopbits, = Ruhezustand der Leitung).



  Die Zeichenlänge (Wortlänge) n liegt zwischen 5 (Fernschreiber) und 8 Bit (8 Bit ASCII[32]).
- Die Übertragungsgeschwindigkeit, das ist die Zahl der übertragenen Bits pro Sekunde[33], muss zwischen Sender und Empfänger vereinbart werden, damit sich der Empfänger nach dem Startbit auf die ankommende Bitfolge synchronisieren kann. Üblich sind $75 \cdot 2^N$ bit/sec mit $N = 0 \cdots 9$, d. h. 75, 150, 300, ... 9600, 19200, 38400 bit/sec, in Sonderfällen werden aber auch andere Übertragungsgeschwindigkeiten eingestellt[34].
- Neben den beiden Datenleitungen für Senden (TxD) und Empfangen (RxD) sind einige Steuerleitungen definiert, die anzeigen, ob das betreffende Gerät eingeschaltet ist (DSR, DTR), empfangsbereit ist (CTS, RTS), die Leitung in Ordnung ist (DCD) oder ein Anruf angekommen ist (RI). Ein Teil

[31] Die Nummern der Normblätter werden häufig auch zur Bezeichnung der Schnittstelle verwendet: RS-232-Interface, V24-Schnittstelle.

[32] ASCII = American Standard Code for Information Interchange

[33] Für bit/sec ist die Einheit Baud gebräuchlich, die Übertragungsgeschwindigkeit wird oft als Baud-Rate bezeichnet.

[34] Moderne Bausteine können mit Übertragungsraten bis zu 1 MBaud betrieben werden.

dieser Leitungen ist für den Modem-Betrieb und damit die Datenfernübertragung gedacht, im lokalen Betrieb können sie zur Datenflusssteuerung (Quittungsbetrieb) benutzt werden.

- Als Steckverbindungen wurden bis vor einigen Jahren 25-polige D-Stecker und -Buchsen benutzt, bei neueren PCs fast ausschließlich 9-polige, auf Spezialkarten teilweise auch kleinere. Die Steckerbelegung ist davon abhängig, ob es sich um ein Datenendgerät (DTE, Data Terminal Equipment) oder um ein Modem-artiges Gerät (DCE, Date Communication Equipment) handelt. PCs fungieren immer als DTE-Geräte, die Steckerbelegung dafür ist in Tabelle 4 angegeben. Bei DCE-Geräten ist die Signalrichtung (Ein/Aus) komplementär.

Pin 9-pol.	Pin 25-p.	Ein/ Aus	Signal
1	8	E	DCD, Data Carrier Detected, Leitung in Ordnung
2	3	E	RxD, Receive Data, Dateneingang
3	2	A	TxD, Transmit Data, Datenausgang
4	20	A	DTR, Data Terminal Ready, Terminal betriebsbereit
5	7		GND, Signal Ground, Masseanschluss
6	6	E	DSR, Data Set Ready, externes Gerät betriebsbereit
7	4	A	RTS, Request To Send, Sendeanforderung zum externen Gerät
8	5	E	CTS, Clear To Send, Sendeanforderung vom externen Gerät
9	22	E	RI, Ring Indicator, Wählsignal vom MODEM

Tabelle 4: Steckerbelegung der seriellen Schnittstelle im PC

Verbindungskabel zwischen einem DTE- und einem DCE-Gerät verbinden jeweils gleiche Pin-Nummern, bei Kabeln zwischen 2 DTE-Geräten (z. B. zwischen PC und Plotter) müssen die sich entsprechenden Pins (RxD↔TxD, CTS↔RTS) jeweils 'gekreuzt' verbunden werden(Abbildung 71). Oft genügen zum Anschluss die Datenleitungen und Masse, teilweise werden die Steuerleitungen zur Datenflusskontrolle benutzt.

7.2.2 Quittungsbetrieb

Werden die Daten über eine serielle Leitung potentiell schneller übertragen, als sie von einem der Teilnehmer verarbeitet werden können (Drucker, Plotter), muss eine Möglichkeit vorgesehen werden, den Datenfluss zu steuern (Quittungsbetrieb, Handshake). Zwei Arten der Datenflusssteuerung werden überwiegend verwendet:

- Beim 'Hardware-Handshake' wird durch den Status der Steuerleitungen mitgeteilt, ob ein Gerät empfangsbereit ist. Die meisten Geräte benutzen

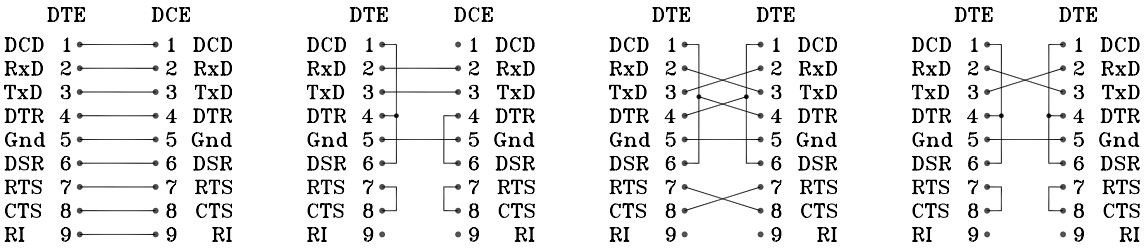


Abbildung 71: Die wichtigsten Kabeltypen für serielle Verbindungen vom PC zu einem Peripheriegerät, Pinbelegungen für 9-polige Stecker bzw. Buchsen. Von links: Volle DTE-DCE-Verbindung (Rechner⇄Modem), DTE-DCE ohne Steuerleitungen, volle DTE-DTE-Verbindung (Nullmodemkabel zwischen zwei PCs), DTE-DTE ohne Steuerleitungen.

zu diesem Zweck die CTS- bzw. RTS-Leitung (CTS-RTS-Handshake). Das Sendergerät muss vor der Ausgabe jedes einzelnen Zeichens den Steuerleitungsstatus überprüfen und gegebenenfalls den 'Aktiv'-Zustand der Handshake-Steuerleitung abwarten.

- Beim 'Software-Handshake' (auch XON-XOFF-Handshake) sind zwei Zeichen vereinbart, die vom Empfängergerät zum Stop und zur Wiederaufnahme der Übertragung ans Sendergerät geschickt werden. Gestoppt wird mit 'XOFF' (meist CTRL-S), wiedergestartet mit 'XON' (meist CTRL-Q). Das Sendergerät sollte insbesondere auf XOFF prompt reagieren, d. h. die Datenausgabe sofort anhalten.

Bei der Übertragung von größeren Datenmengen zwischen Rechnern ist es oft sinnvoll, im Blockbetrieb zu arbeiten. Dabei bildet das Sendergerät nach einem vereinbarten Algorithmus einen Block aus einer bestimmten Anzahl von Zeichen, der zusammen mit einer Prüfsumme verschickt wird. Der Empfänger prüft den Block anhand der Prüfsumme auf Richtigkeit und quittiert positiv oder negativ. Bei negativer Quittung wird der Block nochmals geschickt. Auf diese Weise lässt sich eine hohe Datensicherheit erreichen. Die meisten klassischen Datenübertragungsprogramme arbeiten mit solchen 'Protokollen' (Kermit, Crosstalk, X-Modem, . . .).

7.2.3 Andere Übertragungsnormen

Die RS 232 C-Schnittstellen-Norm ist relativ alt und ursprünglich für niedrige Übertragungsraten konzipiert. Laut Spezifikation ist sie für Leitungslängen bis zu 15 m und für Übertragungsraten bis zu 20 kbit/s ausgelegt. Inzwischen sind deutlich höhere Übertragungsraten möglich und gebräuchlich (MODEMs, Rechner-Rechner-Kopplung). Neben der RS 232 C existieren zwei wesentlich leistungsfähigere Normen, die allerdings im PC-Bereich wenig verwendet werden.

Die Norm RS 423 A definiert eine unsymmetrische Schnittstelle, bei der die Datenübertragung über ein (mit dem Wellenwiderstand) abgeschlossenes Koaxialkabel erfolgt. Die maximale Übertragungsrate ist 300 kbit/s, die maximale Leitungslänge 600 m.

Die Norm RS 422 A benutzt symmetrische Leitungstreiber und -empfänger und abgeschlossene, verdrillte Zweidrahtleitungen. Die maximale Leitungslänge ist 1200 m, die maximale Übertragungsrate 2 Mbit/s (bei dann allerdings verringerter Leitungslänge von max. 60 m).

Noch höhere Übertragungsraten sind durch Lichtleiterverbindungen, größere Übertragungsstrecken durch Modulationsverfahren (Telefon-Modem) oder über die einschlägigen digitalen Postdienste (Datex-P, ISDN, DSL) realisierbar.

Für alle Normen sind Treiberbausteine verfügbar, die eine Umsetzung des vom Schnittstellenbaustein generierten TTL-Signals auf die Norm-Signale vornehmen.

7.2.4 Programmierung unter Windows

Bis vor einigen Jahren konnte die serielle Schnittstelle des PC mit akzeptabler Geschwindigkeit nur durch direkte Programmierung der Schnittstellenbausteine betrieben werden. Zum einen lag das an der beschränkten Funktionalität der Bausteine, zum andern an den wenig leistungsfähigen Betriebssystemfunktionen. Beides hat sich deutlich geändert: in der Entwicklung von den 'klassischen' ICs 8250 und 16450 zum 16550 bzw. Multifunktionsbausteinen mit integriertem 16550 wurde der Zeichenpuffer im Baustein vergrößert, und moderne Betriebssysteme wie Windows 32 bieten alle zum Betrieb notwendigen Funktionen. Direkte Portprogrammierung [30] ist daher nicht mehr sinnvoll.

Vom Betriebssystem wird die serielle Schnittstelle logisch wie eine Datei gehandhabt, die gelesen und geschrieben werden kann. Folglich muss zunächst ein Datei-Objekt angelegt werden, um die Eigenschaften der Schnittstelle einstellen zu können. Unter Win32 gibt es dafür die Funktion `CreateFile`:

```
HANDLE hCOM = CreateFile("COM1:", 0, 0, NULL, OPEN_EXISTING, 0, NULL); .
```

Danach können mit den Methoden `ReadFile` und `WriteFile` Datenblöcke von der Schnittstelle gelesen bzw. zur Schnittstelle geschickt werden, mit `CloseHandle` wird die Schnittstelle wieder freigegeben.

Die Schnittstellenparameter sind in einem im Betriebssystem vorgehaltenen umfangreichen Bit-Feld, dem *Device Control Block*, definiert:

```
typedef struct _DCB { // dcb
DWORD DCBlength;           // sizeof(DCB)
DWORD BaudRate;            // current baud rate
DWORD fBinary: 1;          // binary mode, no EOF check
DWORD fParity: 1;          // enable parity checking
DWORD fOutxCtsFlow:1;      // CTS output flow control
```

```
DWORD fOutxDsrFlow:1;      // DSR output flow control
DWORD fDtrControl:2;       // DTR flow control type
DWORD fDsrSensitivity:1;   // DSR sensitivity
DWORD fTXContinueOnXoff:1; // XOFF continues Tx
DWORD fOutX: 1;            // XON/XOFF out flow control
DWORD fInX: 1;             // XON/XOFF in flow control
DWORD fErrorChar: 1;       // enable error replacement
DWORD fNull: 1;            // enable null stripping
DWORD fRtsControl:2;       // RTS flow control
DWORD fAbortOnError:1;     // abort reads/writes on error
DWORD fDummy2:17;          // reserved
WORD wReserved;            // not currently used
WORD XonLim;               // transmit XON threshold
WORD XoffLim;              // transmit XOFF threshold
BYTE ByteSize;             // number of bits/byte, 4-8
BYTE Parity;               // 0-4=no,odd,even,mark,space
BYTE StopBits;             // 0,1,2 = 1, 1.5, 2
char XonChar;              // Tx and Rx XON character
char XoffChar;             // Tx and Rx XOFF character
char ErrorChar;            // error replacement character
char EofChar;              // end of input character
char EvtChar;              // received event character
WORD wReserved1;           // reserved; do not use
} DCB;
```

Die Bedeutung der einzelnen Komponenten sowie vordefinierte Konstanten für die Zuweisung von Werten sind z. B. in der Online-Hilfe zu Visual C++ ausführlich beschrieben (suchen unter 'DCB').

Um Einstellungen zu ändern, wird der *Device Control Block* kopiert, geändert und anschließend wieder zurückkopiert, etwa so:

```
DCB dcb;
GetCommState ( hCOM, &dcb );
dcb.BaudRate = CBR_38400;             // set to 38400 Baud
...
SetCommState ( hCOM, &dcb );
```

Durch eine entsprechende Kontrollstruktur wird das Verhalten der Schnittstelle bei Zeitüberschreitung (*Timeout*) definiert (Online-Hilfe unter 'COMMTIMEOUTS'):

```
typedef struct _COMMTIMEOUTS {   // ctmo
DWORD ReadIntervalTimeout;
DWORD ReadTotalTimeoutMultiplier;
DWORD ReadTotalTimeoutConstant;
DWORD WriteTotalTimeoutMultiplier;
DWORD WriteTotalTimeoutConstant;
} COMMTIMEOUTS,*LPCOMMTIMEOUTS;
```

Änderungen ähnlich wie oben durch:

```
COMMTIMEOUTS ctmo;
GetCommTimeouts ( hCOM, &ctmo );
```

```
ctmo.ReadIntervalTimeout = 100;      // allow 100 msec between
...                                  //   arriving characters
SetCommTimeouts ( hCOM, &ctmo );
```

Zusätzlich zu dieser umfassenden Art der Schnittstellenprogrammierung gibt es vereinfachte Funktionen, um die Ausgangsleitungen der Schnittstelle statisch zu setzen:

```
BOOL SetCommBreak ( HANDLE hFile );   // handle of comm. device
BOOL ClearCommBreak ( HANDLE hFile ); // handle of comm. device
```

setzen die Datenleitung auf logisch 0 bzw. logisch 1.

```
BOOL EscapeCommFunction ( HANDLE hFile,   // handle of comm. device
                          DWORD dwFunc ); // function to perform
```

mit `dwFunc` = `SETRTS, CLRRTS, SETDTR, CLRDTR, ...` setzt die Quittungsleitungen in definierte Zustände.

Mit diesen Funktionen können die Ausgangsleitungen der Schnittstelle relativ einfach zur Steuerung von primitiven Geräten genutzt werden, beispielsweise wie in folgendem Programmfragment zum Schalten eines Halbleiterrelais:

```
void CAnyDlg::SwitchOn()
{ EscapeCommFunction ( hCOM, SETRTS ); };

void CAnyDlg::SwitchOff()
{ EscapeCommFunction ( hCOM, CLRRTS ); };
```

Neben den bisher beschriebenen Funktionen gibt es eine Anzahl weiterer, die dazu dienen, die serielle Schnittstelle ereignisgesteuert zu verwenden (`SetCommMask`, `WaitCommEvent`, ...). Da sie im Umfeld der Messdatenerfassung relativ selten benötigt werden, sollen sie hier nicht näher diskutiert werden (sie sind in der Online-Hilfe ausführlich dokumentiert).

7.2.5 C++ und Microsoft Foundation Classes

Objektorientiert in C++ kann man die Schnittstelle – z. B. unter Visual-C++ – programmieren, wenn man die Klassenbibliothek der *Microsoft Foundation Classes* verwendet. Man konstruiert ein `CFile`-Objekt, etwa

```
CFile  SerialLine ( "COM1:", CFile::modeReadWrite ); ,
```

dessen Methoden `Read` und `Write` dann zur Ein- und Ausgabe von Datenblöcken aufgerufen werden.

Die zum Zugriff auf den *Device Control Block* und die *Timeout*-Struktur benötigte *Handle* ist in der Klasse `CFile` als Variable definiert:

```
HANDLE hCOM = SerialLine.m_hFile .
```

7.2.6 C-Programmierung mit *Stream-IO*-Funktionen

Die Windows-Funktionen `ReadFile` und `WriteFile` sind für binäre Datenblöcke gedacht, ebenso die entsprechenden Methoden der Klasse `CFile`. Die

Formatierung erfolgt getrennt davon, etwa mit `sprintf` bzw. `sscanf`. Statt der Windows-Funktionen kann man aber auch die von C gewohnten *Stream-IO*-Funktionen verwenden, die die Formatierung mit erledigen (`fprintf`, ...). Die Schnittstelle wird dabei wie eine Datei gehandhabt. Im folgenden Fragment werden Textanweisungen formatiert ausgegeben und Daten binär gelesen (Tektronix-Oszilloskop):

```
FILE * com = fopen("COM1:", "r+");
setvbuf(com, NULL, _IONBF, 0);
fprintf(com, "DAT:SOU CH%d\r\n", channel);
...
fprintf(com, "CURV?\r\n");
fflush(com);
fread(buffer, sizeof(char), 10, com);
...
fclose(com); .
```

Die Funktion `setvbuf` weist den Compiler an, keinen Datenpuffer bereitzustellen, die Daten also direkt zur Schnittstelle weiterzuleiten. Da dies nicht von allen Compilern strikt befolgt wird, ist zusätzlich ein `fflush` zwischen Schreiben und Lesen zu empfehlen, um den eventuell doch vorhandenen Puffer zu leeren.

Sollen Schnittstellenparameter über den *Device Control Block* eingestellt werden, muss dazu eine kurze Programmsequenz

```
HANDLE hCOM = CreateFile ( "COM1:",  ... );
DCB dcb;
GetCommState ( hCOM, &dcb );
...
CloseHandle ( hCOM );
```

– wie in 7.2.4 beschrieben – vorgeschaltet werden.

7.2.7 Linux-Spezifisches

Unter Linux kann die serielle Schnittstelle ebenfalls mit *Stream-IO*-Funktionen betrieben werden, statt `"COM1:"`, ... sind als Gerätenamen die entsprechenden Linux-*Devices* `"/dev/ttyS0"`, ... einzusetzen.

Desweiteren ändert sich die Programmierung der Schnittstellenparameter, dem *Device Control Block* entspricht unter Linux die etwas andere Struktur `termios`; hier ein Anwendungsbeispiel:

```
FILE * com = fopen("/dev/ttyS0", "r+");
int fn = fileno(com);
struct termios options;
tcgetattr(fn, &options);              // get current options
cfsetispeed(&options, B9600);         // inputrate
cfsetospeed(&options, B9600);         // outputrate
cfmakeraw(&options);                  // raw input
options.c_cflag |= (CLOCAL | CREAD); // local usage, receive
options.c_cflag |= CRTSCTS;           // enable HardwareFlowControl
```

```
options.c_cc[VTIME] = 10;            // timeout (*0.1s)
options.c_cc[VMIN] = 0;              // minimum received chars
tcsetattr(fn, TCSANOW, &options);    // activate the changes now.
```

7.2.8 Programmierung in MATLAB

Seit dem Release 12 (September 2000) enthält MATLAB eine Funktionsbibliothek für die seriellen Schnittstellen. Mit der Funktion `serial` wird zunächst ein Objekt erstellt, dessen Parameter MATLAB-üblich mit `set` und `get` eingestellt oder erfragt werden können[35]. Daneben ist auch ein objektartiger Zugriff in Punktschreibweise möglich.

Inzwischen (ab Release 13 – Juni 2002) ist in MATLAB auch ein *Property Inspector* integriert, mit dem Objekt-Eigenschaften interaktiv inspiziert und verändert werden können. Zumindest in der Testphase ist das ein Hilfsmittel, das einem viel Arbeit ersparen kann. Der *Property Inspector* wird mit

```
inspect(ser);
```

aufgerufen, nachdem das serielle Objekt beispielsweise durch

```
ser = serial('COM1');
```

erstellt wurde.

Auf das Schnittstellenobjekt werden dann transparent die C-ähnlichen MATLAB-I/O-Funktionen angewendet. Das folgende Beispiel – eine Funktion `osc`, die Daten aus einen Digitalspeicheroszilloskop (Tektronix 210) in MATLAB einliest – verdeutlicht die Verwendung:

```
function y = osc(channel)
ser = serial('COM1');
ser.FlowControl = 'hardware';
ser.InputBufferSize = 3000;
ser.Terminator = 'CR/LF';
fopen(ser);
fprintf(ser, 'SEL:CH%d ON\n', channel);
fprintf(ser, 'DAT:SOU CH%d\n', channel);
fprintf(ser, 'DAT:ENC RPB\n');
fprintf(ser, 'CURV?\n');
a = 0;
while char(a)~='#',
    a = fread(ser, 1, 'uchar');
end;
b = fread(ser, 1, 'uchar');
n = fread(ser, str2num(char(b)), 'uchar');
y = fread(ser, str2num(char(n')), 'uint8');
fclose(ser);
delete(ser);
clear ser;
```

[35] Die komplette Liste der möglichen Parameter erhält man mit `set(handle)`, die Liste der aktuellen Werte mit `get(handle)`, `handle` ist die von `serial` zurückgelieferte *Object Handle*.

Das Schnittstellenobjekt wird mit `serial` erstellt, anschließend werden einige Parameter festgelegt – Datenflusskontrolle, Puffergröße, Zeilenendezeichen. Dann werden nach `fopen` mit `fprintf` einige Anweisungen an das Oszilloskop gegeben (Kanalauswahl, Datenformat), die letzte (`CURV?`) fordert die aktuellen Daten an. Gelesen wird solange, bis ein #-Zeichen kommt, danach werden die Daten relevant. Zunächst eine Ziffer (`b`), die angibt, aus wieviel Ziffern die folgende Zahl – `n` – besteht[36]. Diese wiederum informiert über die Größe des Datenblocks `y`. Die Folge von `fread`-Funktionen entspricht dieser Datenstruktur. Die letzten 3 Zeilen räumen auf.

7.2.9 Verallgemeinerte Software-Standards für Peripheriegeräte

In gemischten Messumgebungen ist es oft sinnvoll, die serielle Schnittstelle über das gleiche Software-Interface anzusprechen wie die übrigen Schnittstellen. Zwei von der Messgeräte-Industrie gut eingeführte Standards sind die *Standard Instrument Control Library* (*SICL*) der Firma Agilent und der herstellerübergreifende Standard *Virtual Instrument Software Architecture* (*VISA*). Mehr zu den beiden Software-Standards in den Kapiteln zum IEC-BUS (7.5) und zum USB (7.6).

7.3 Direkte Port-Ein/Ausgabe unter Windows 32

Messkarten im Rechner sind entweder über den Ein-/Ausgabe-Adressbereich des Prozessors (*port mapped I/O* oder *isolated I/O*) oder über den Speicher-Adressbereich (*memory mapped I/O*) zugänglich (Kurzdefinitionen dazu finden Sie im Internet, beispielsweise bei Wikipedia [31, 32, 33]). Ein direkter Zugriff auf solche Hardware verbietet sich im Allgemeinen, moderne Betriebssyteme sind in der Regel gut dagegen geschützt.

Messkarten-Hersteller liefern fast immer geeignete Treiber-Funktionen mit, die aus Programmen oder Bibliotheken (*DLLs* oder *Shared Libraries*) aufgerufen werden können. Insofern ist eine sehr hardwarenahe Programmierung (*Kernel*-Treiber) meist nicht erforderlich. Sollten Sie dennoch einmal vor dem Problem stehen, direkte Port- oder Speicherzugriffe zu programmieren, ist auch dies auf einfache Weise zu realisieren.

Das Schreiben eines eigenen Kernel-Treibers kann man umgehen, indem man einen generischen Treiber verwendet. Ein Beispiel für einen solchen Gerätetreiber ist `UNIIO`, der von der Firma *BBD SOFT* als freie Software zur Verfügung gestellt wurde [34]. *BBD SOFT* arbeitete daran, *'Unified IO - C++ interface for industrial IO cards'* zu entwickeln. Ziel war es, plattformunabhängige C++-Klassen für industrielle Peripheriekarten zu entwickeln. Plattform- und Compiler-abhängig müssen dabei nur noch kurze Routinen für den direkten

[36] Mit `fread` wird `n` als Spaltenvektor gelesen, daher das transponierte `n'` bei der Umwandlung in eine Zahl in der folgenden Zeile.

Hardwarezugriff implementiert werden. Die Software wird inzwischen nicht mehr weiter entwickelt, ist aber noch als freier Download bei der Herstellerfirma erhältlich. Meine Tests ergaben, dass zumindest noch unter Windows XP damit gearbeitet werden kann. Näheres zur Anwendung in älteren Versionen des Skriptums (2006 und früher).

7.4 Parallele Schnittstellen

werden benutzt, um Peripheriegeräte über mehrere Leitungen gleichzeitig anzusprechen und um Daten schnell byte- oder wortweise einzulesen oder auszugeben. Ein Standardgerät, das früher auf diese Weise angeschlossen wurde, ist der Drucker. Die klassische Druckerschnittstelle arbeitet mit 8 Datenleitungen (ein Byte wird jeweils komplett zum Drucker übertragen), einer Leitung, die anzeigt, dass die Daten gültig sind und einer Leitung, die meldet, ob der Drucker beschäftigt ist. Weitere Leitungen sind für zusätzliche Statusmeldungen zuständig. Zwischenzeitlich wurde die Funktionalität der Druckerschnittstelle beträchtlich erweitert, so dass bei damit ausgestatteten Rechnern eine sehr schnelle bidirektionale Datenübertragung zu Peripheriegeräten wie externen Disk-Laufwerken, Scannern o. ä. möglich war.

Inzwischen sind diese parallelen Schnittstellen weitgehend von USB-Schnittstellen abgelöst. Sie werden daher im vorliegenden Skriptum nicht mehr berücksichtigt. Ausführliche Informationen zu den parallelen Schnittstellen finden Sie – falls benötigt – in der älteren Versionen des Skriptums (2006 und früher).

7.5 Der IEC-Bus

Die in der vorhergehenden Kapiteln beschriebenen Schnittstellen (parallel, seriell) werden im wesentlichen für 2-Punkt-Verbindungen benutzt (PC↔Plotter, PC↔Drucker). Diese Verbindungstechnik wird sehr aufwendig, wenn eine größere Anzahl von Messgeräten eingebunden werden soll, da der PC für jedes Messgerät eine individuelle Schnittstelle zur Verfügung stellen muss.

Im Gegensatz zu den 2-Punkt-Verbindungen zeichnen sich Bussysteme dadurch aus, dass eine Vielzahl von Geräten an eine Schnittstelle angeschlossen werden kann, und über den Bus eine Kommunikation mit dem PC wie auch zwischen den angeschlossenen Geräten möglich ist.

7.5.1 Grundlagen

Ein Bussystem, das auf dem Messgerätesektor noch sehr verbreitet ist, ist der IEC-Bus. Wikipedia schreibt dazu:

Der Bus wurde in den 1960er Jahren von der Firma Hewlett-Packard[37] *(HP)*

[37] Der Messgerätesektor firmiert inzwischen unter *Agilent* (KB).

als HP-IB entwickelt, und von dieser Ende der 1970er Jahre zur IEEE-Standardisierung eingereicht. 1975 erfolgte die Standardisierung als IEEE-488-1975. 1978 wurde der Standard überarbeitet und als IEEE-488-1978 veröffentlicht (später umbenannt zu IEEE 488.1, 2004 umbenannt zu IEEE 60488.1:2004). HP-IP entspricht diesem Standard. ANSI übernahm den Standard als ANSI Standard MC 1.1. Das IEC übernahm den Standard als IEC-625.

1987 verabschiedete das IEEE eine Ergänzung, IEEE 488.2 (alias IEEE 60488.2:2004), die den Originalstandard erweitert, nicht ersetzt. Der Originalstandard definierte keine Datenübertragungsprotokolle oder Gerätekommandos. Um den Wildwuchs von Herstellerlösungen einzudämmen, wurde 1990 IEEE 488.2 um eine standardisierte Kommandosprache SCPI erweitert. Viele Geräte erfüllen auch heute noch nicht den IEEE488.2-Standard.

Schnellere Varianten sind z. B. als HS488 bekannt, welcher 2003 als IEEE 488.1-2003 standardisiert wurde. Ältere Geräte sind jedoch teilweise nicht in der Lage, das HS488-Protokoll zu verarbeiten. Bei einer Mischbestückung muss dies daher entsprechend berücksichtigt werden.

Dieser inzwischen somit sehr gut genormte Schnittstellenbus[38] besteht aus 8 Daten-, 3 Handshake- und 5 allgemeinen Steuerleitungen (Abb. 72).

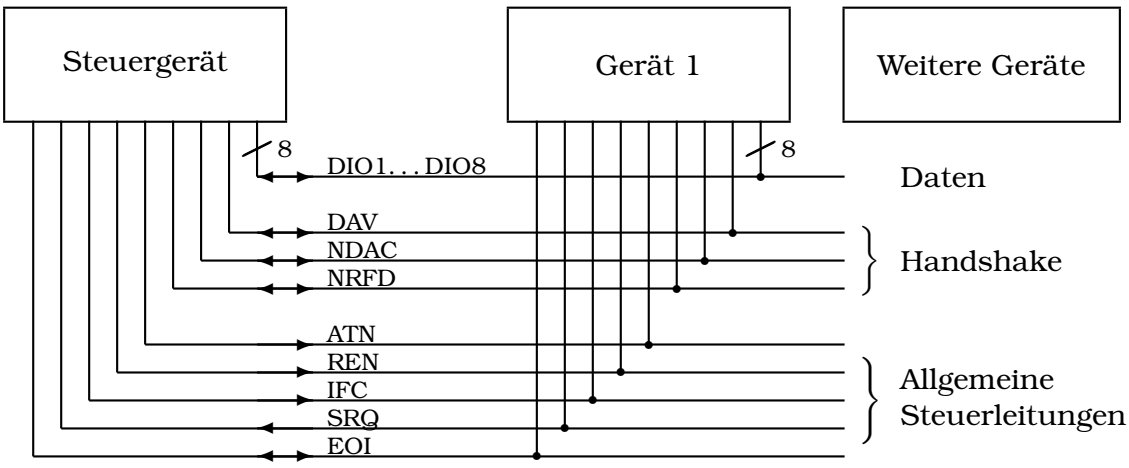


Abbildung 72: Die 16 Signalleitungen des IEC-Bus.

Der Bus wird in negativer Logik mit TTL-Spannungen betrieben, der aktive Leitungszustand bzw. logisch 1 entspricht somit ca. 0 V, inaktiv oder logisch 0 dagegen ca. 5 V. Die Signalausgänge sind als 'offene Kollektor' Ausgänge realisiert, durch diese 'Wired-Or'-Verknüpfung wird sichergestellt, dass auch dann keine Buskonflikte auftreten, wenn mehrere Geräte gleichzeitig auf eine bestimmte Busleitung zugreifen. Andererseits kann ein einzelnes Gerät den Aktiv-Status einer Leitung erzwingen.

[38] Neben den oben genannten Bezeichnungen ist auch *GPIB* (General Purpose Interface Bus) bei manchen Herstellern gebräuchlich.

Gegenüber einem Rechnerbus weist der IEC-Bus einige Unterschiede auf:

Die Datenübertragung erfolgt asynchron, d. h. sie wird nicht — wie beim PC-Bus — durch ein zentrales Taktsignal synchronisiert, sondern durch ein Quittungsverfahren mit den 3 Handshakeleitungen. Das Signal für gültige Daten (DAV: Data Valid) wird vom jeweiligen Sprechergerät nur dann aktiviert, wenn alle Hörergeräte am Bus ihre Bereitschaft durch das deaktivierte NRFD-Signal (Not Ready For Data) anzeigen. Die Daten bleiben danach so lange gültig, bis alle Hörergeräte das NDAC-Signal (Not Data Accepted) deaktiviert und damit gemeldet haben, dass die Daten empfangen wurden. Die Datenübertragungsgeschwindigkeit richtet sich also dynamisch nach dem jeweils langsamsten aktiven Gerät am Bus. Dennoch sind — bei hinreichend schnellen Teilnehmern — maximale Übertragungsraten von ca. 1 MByte/sec möglich.

Die 8 Datenleitungen (DIO1. . . DIO8) werden sowohl für Daten als auch für Kommandos benutzt, die Unterscheidung erfolgt durch die Steuerleitung Attention (ATN). Nur ein Gerät am Bus — als Steuergerät (Controller) konfiguriert — darf die Attention-Leitung bedienen und somit Kommandos erteilen. Ein Teil der Kommandos dient dazu, andere Geräte am Bus als Sprecher- (Talker) oder Hörer-Geräte (Listener) zu adressieren. Durch diese Adressierkommandos wird eine Datenübertragung zwischen einem Sprecher und einem oder mehreren Hörern eingeleitet. Insgesamt sind 31 Hörer- und 31 Sprecheradressen in einem IEC-Bus-System möglich, die übrigen Kommandobytes sind für weitere Befehle reserviert. Unter anderem ist eine weitergehende Adressunterteilung über Sekundäradressen möglich.

Die Funktion der weiteren Steuerleitungen:

REN (Remote Enable) schaltet die lokale Bedienungsmöglichkeit der angeschlossenen Geräte ab.

IFC (Interface Clear) bringt das Schnittstellensystem in einen definierten Anfangszustand.

SRQ (Service Request) wird von einem angeschlossenen Gerät aktiviert, wenn es eine Bedienung wünscht (z. B. am Ende einer Messung zum Datentransfer). Die dazu nötige Unterbrechung wird vom Steuergerät veranlasst, erfolgt daher nicht unbedingt prompt. Die Reaktion des Steuergeräts muss mit einer Umfrage (Serial Poll, Parallel Poll) beginnen, um festzustellen, welches Gerät die SRQ-Leitung aktiviert hat.

EOI (End Or Identify) hat zwei Funktionen: Zum einen zeigt der gerade aktive Sprecher damit das Ende des Datentransfers an. Zum anderen wird EOI zusammen mit ATN vom Controller benutzt, um eine Parallelabfrage (Parallel Poll) anzuordnen.

Für weitere technische Details sei an dieser Stelle auf die weiterführende Literatur zum IEC-Bus verwiesen [35, 36].

7.5.2 Datenformat

Die Datenübertragung auf dem IEC-Bus erfolgt 'bit-parallel' und 'byte-seriell'. Hinsichtlich der Länge des Datenstroms und der Codierung der einzelnen Bytes bestehen keine Einschränkungen. Daher muss die Art der Datencodierung jeweils zwischen Talker- und Listener-Gerät vereinbart werden. Da aber bei praktisch allen Peripheriegeräten das Datenformat für den IEC-Bus nicht veränderbar ist, bedeutet 'vereinbaren', dass sich der PC als flexibelstes Gerät am Bus auf das jeweilige Datenformat einstellen muss. Zwei Codierungsarten werden überwiegend verwendet:

- Binäre Codierung dort, wo große Datenmengen möglichst schnell übertragen werden sollen. Einige Digitalspeicheroszilloskope übertragen ihre Daten auf diese Weise — bei einer Auflösung von 8 Bit wird für einen Datenpunkt nur 1 Byte benötigt.
- ASCII-Codierung dort, wo es eher auf Sicherheit und Verständlichkeit ankommt und die Geschwindigkeit nicht im Vordergrund steht. Die Daten werden als Text übertragen, somit ist eine sofortige Kontrolle möglich. Fast alle Digitalmultimeter benutzen diese Codierungsart sowohl für ihre Programmierung als auch für die Messdaten. Da jeweils nur ein Messwert übertragen wird, werden keine hohen Geschwindigkeiten benötigt.

7.5.3 Programmierung

PC-Karten für den IEC-Bus[39] enthalten im allgemeinen einen intelligenten Schnittstellenbaustein, der einen großen Teil des Busmanagements (insbesondere den Quittungsablauf) selbständig erledigt. Meist können die Karten sowohl als Controller (überwiegende Betriebsart im PC) wie auch als Talker/Listener konfiguriert werden.

Fast alle Kartenhersteller (jedenfalls die teureren) liefern Softwarebibliotheken zum Betrieb der Karte mit[40], üblich ist zumindest eine C-Bibliothek. Der Datenaustausch mit einem externen Gerät erfolgt dann hochsprachlich analog zum Dateizugriff nach dem Schema:

1. Logische Verbindung zum externen Gerät herstellen (*open*),
2. Daten lesen (*read*) oder schreiben (*write*),
3. Verbindung zum externen Gerät schließen (*close*).

Softwarebibliothek SICL: Agilent, neben National Instruments und Keithley einer der wichtigeren Hersteller für IEC-Bus-Karten, liefert zum Betrieb der Karten die *Agilent Standard Instrument Control Library* mit, eine Bibliothek mit Schnittstellen für Basic und C. Agilent meint dazu:

[39] Auf dem Markt sind unter anderem Steckkarten für verschiedene Mainboard-Busse, PCMCIA-Karten für Notebooks, aber auch USB-Adapter.

[40] Die aktuellsten Versionen sind meist über die Herstellerwebseiten erreichbar.

> *The SICL library provides very complete and flexible control of instruments. However, this API is primarily implemented and supported by Agilent; it is not an industry standard. SICL is optimized for use from C and C++ and can be used from Visual Basic and other environments that can call arbitrary Windows DLLs. SICL provides complete access to GPIB, USB, serial, LAN, VXI message-based, and VXI register-based products.*

Die Software muss zunächst auf dem PC installiert werden, anschließend wird die Schnittstellenkarte logisch eingebunden, d. h. unter einem symbolischen Namen werden die physikalischen Daten der Karte (E/A-Adresse usw.) vermerkt (zuständiges Programm: `iocfg32.exe`). Der bei der Konfiguration festgelegte Name wird – zusammen mit der Adresse des externen Geräts – beim Verbindungsaufbau benötigt. Die Bibliotheksfunktionen werden in C bzw. C++ durch die Header-Datei `sicl.h` bekanntgemacht und aus `sicl32.lib` mit dem Linker ins Programm eingebunden. Aus den Routinen in `sicl32.lib` erfolgen Zugriffe auf die zentrale System-DLL `sicl32.dll`, die das eigentliche Ein/Ausgabe-Management abwickelt. Durch diese Softwarestruktur ist gewährleistet, dass alle Zugriffe auf die IEC-Bus-Karte(n) im System über eine zentrale Stelle laufen; die verantwortliche DLL ist 'Multi-Thread'-fähig, somit können mehrere Programme gleichzeitig mit dem Bus arbeiten.

Unter anderem stellt SICL die folgenden Definitionen und Funktionen für den Betrieb von Geräten bereit (aus `sicl.h`):

```
typedef int INST;
INST iopen (char *addr);
int iclose (INST id);  .
```

Auf ein Geräte wird über eine 'Instrument-Handle' zugegriffen, die in Typ und Verwendungsart in etwa der 'File-Handle' in C entspricht. `iopen()` eröffnet die Verbindung zum Gerät, `addr` enthält Kartenname und Geräteadresse, also z. B. "hpib,7".

Die Funktionen für formatiertes Schreiben und Lesen entsprechen den C-Bibliotheksfunktionen `fprintf()` und `fscanf()`:

```
int iprintf (INST id, const char *fmt, ...);
int iscanf (INST id, const char *fmt, ...);  .
```

Der Rückgabewert enthält die Anzahl der tatsächlich konvertierten Argumente.

Zum unformatierten Schreiben und Lesen aus einem oder in ein Bytefeld `buf` der Länge `datalen` bzw. `bufsize` sind

```
int iwrite (INST id, char *buf, unsigned long datalen,
            int endi, unsigned long *actual);
int iread (INST id, char *buf, unsigned long bufsize,
           int *reason, unsigned long *actual);
```

zuständig, `endi` legt die Art des Übertragungsendes fest ($= 0$: ohne **EOI**, $\neq 0$: mit **EOI**), `actual` gibt die Zahl der tatsächlich übertragenen Bytes an und `reason`

den Grund für Erfolg oder Misserfolg. Die Rückgabewerte sind C-üblich Null bei Erfolg, ungleich Null bei Misserfolg.

Ein Programm, das ein Digitalvoltmeter auf einen bestimmten Betriebszustand setzt und dann einen Messwert liest, könnte die folgenden Zeilen enthalten:

```
#include "c:/hp/siclnt/c/sicl.h"
...
INST DVM = iopen("hpib,7");
iprintf (DVM, "L0VAT0\n");
iscanf (DVM, "%lg", &Voltage);
iclose(DVM);  .
```

Die Details der Funktionen sowie die Vielzahl weiterer SICL-Funktionen sind in den bei den HP-Karten mitgelieferten umfangreichen Handbüchern [37, 38] ausführlich beschrieben.

SICL und MATLAB: Die durch die *Standard Instrument Control Library* verfügbare Funktionalität lässt sich von MATLAB aus über MEX-Funktionen nutzen, die ihrerseits *SICL*-Funktionen verwenden. Ein einfaches Beispiel ist im Anhang B gelistet. Die dort implementierte `mexFunction` kann als `siclio` mit unterschiedlicher Signatur aufgerufen werden (`siclio` ist der Dateiname der C++-Quelldatei und der daraus kompilierten DLL). Als erster Parameter muss immer der *SICL*-Gerätename angegeben werden. Der Inhalt eines zweiten Parameters wird auf das so adressierte Gerät ausgegeben. Wird stattdessen oder zusätzlich ein Rückgabewert angefordert, so werden Daten (Text- oder Binär-) vom Gerät gelesen und an MATLAB übergeben.

`siclio('hpib,10','APPL:SIN 3000,10');` würde einen Funktionsgenerator auf der Adresse 10 des IEC-Busses auf 10 V Sinusspannung mit 3000 Hz einstellen.

`d = siclio('hpib,7');` würde Daten vom Gerät auf der Adresse 7 des IEC-Busses lesen (z. B. von einem Multimeter).

`d = siclio('hpib,7','L0VAT0');` würde vor dem Lesen der Daten am Multimeter einen bestimmten Betriebszustand einstellen (im Beispiel ein Altgerät der Firma *Prema*, das auf kurze Antwort – `L0`, Wechselspannungsmessung – `VA` und eine bestimmte Integrationszeit – `T0` eingestellt wird).

Softwarebibliothek VISA: Während *SICL* im Wesentlichen von Agilent entwickelt wurde und immer noch regelmäßig an neue Hardware angepasst wird, hat sich zwischenzeitlich herstellerübergreifend ein neuer Industrie-Standard etabliert, die *Virtual Instrument Software Architecture* (*VISA*). Auch dazu nochmals Agilent

> *The VISA API conforms to industry standards. A program written to work with the VISA library will work with implementations of VISA from other vendors. VISA is optimized for use from C and C++ and can be used from Visual Basic and other environments that can call arbitrary Windows DLLs. Agilent provides header files to facilitate the use of VISA in Microsoft® Visual Basic 6, Visual Basic .NET and C#. VISA provides complete access to GPIB, USB, seri-*

al, LAN, VXI message-based, and VXI register-based products.

Mehr zu *VISA* und Programmierbeispiele dazu folgen im Kapitel zum Universal Serial Bus (7.6).

7.6 Universal Serial Bus (USB)

Von einem Herstellerkonsortium bestehend aus Compaq, Digital Equipment Corporation, IBM PC Company, Intel, Microsoft, NEC und Northern Telecom wurde nach mehrjähriger Vorarbeit im Januar 1996 eine neue Schnittstellennorm für den PC vorgeschlagen [39], der *Universal Serial Bus* (USB). Zunächst zögerlich (*U*seless, *S*enseless, *B*rainless), später lawinenartig haben sich alle Hersteller dem Vorschlag angeschlossen, alle PCs (und auch verwandte Geräte wie beispielsweise Videorecorder) enthalten inzwischen diese Schnittstelle (gekennzeichnet durch das nebenstehende Logo) – in der Regel mehrfach; fast alle neueren Peripheriegeräte sind damit ausgestattet. Seit einiger Zeit werden auch Messgeräte vermehrt mit dieser einfachen und preisgünstigen Schnittstelle ausgestattet, zum Teil zusätzlich zu anderen Standardschnittstellen, aus Kostengründen aber auch zum Teil nur noch ausschließlich damit. In allen neueren PC-Betriebssystemen ist die Basisfunktionalität für den USB integriert, darauf aufsetzende spezialisierte Treiber für Peripheriegeräte werden von den Herstellern in praktisch allen Fällen mitgeliefert, umfangreiche Eigenprogrammierung ist daher meist nicht erforderlich.

7.6.1 Überblick

Der USB wurde so universell konzipiert, dass er die meisten früheren Standardschnittstellen weitgehend überflüssig macht. In der Norm sind verschiedene Geschwindigkeitsbereiche vorgesehen, so dass auf dem Bussystem langsame und schnelle Peripheriegeräte koexistieren können, ohne dass überall ein hoher Aufwand für schnellen Schnittstellenbetrieb nötig ist. Einen Überblick über die Geschwindigkeitsbereiche des USB gibt die Zusammenstellung in Abbildung 73. Zunächst waren die beiden langsamen Geschwindigkeitsbereiche eingeführt worden (USB 1.1), heute werden alle drei Geschwindigkeitsbereiche von Hard- und Software unterstützt (USB 2.0). Allenfalls bei etwas älteren Geräten muss man damit rechnen, dass sie mit den engeren Spezifikationen von USB 2.0 nicht zurecht kommen. Viele der neueren Geräte sind jedoch in der Lage, sich an dieses Problem anzupassen (USB-1.1-Kompatibilität).

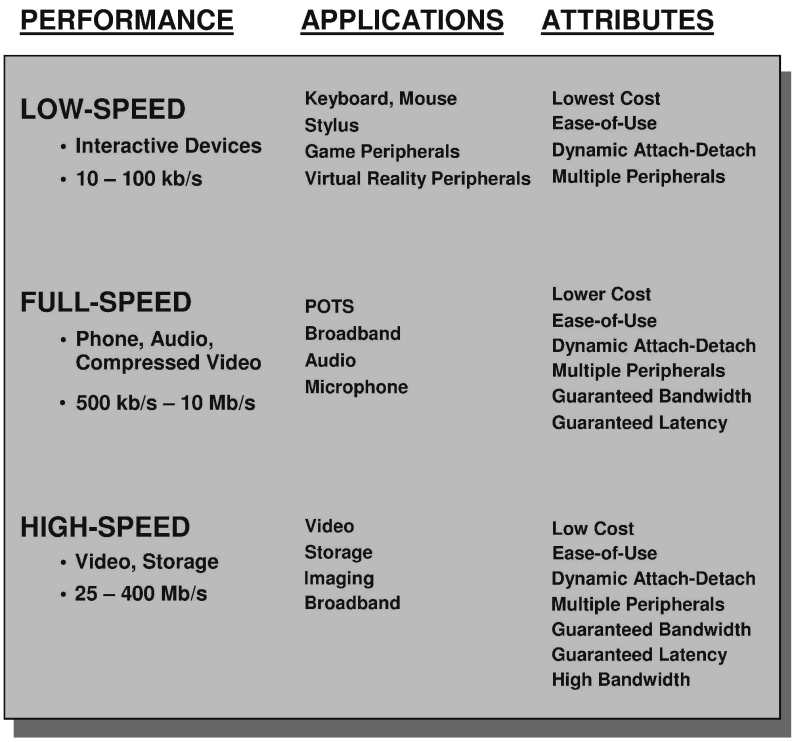


Abbildung 73: Geschwindigkeitsbereiche für den USB, Anwendungen und typische Eigenschaften (Stand USB 2.0, 2002).

Das Konzept des USB sieht eine hierarchisch vernetzte Peripheriestruktur vor, in der die einzelnen Geräte durch logische Adressen angesprochen werden.

Ähnlich wie beim Twisted-Pair-Ethernet ist die logische Busstruktur physikalisch durch Zweipunktverbindungen realisiert, die über Verteiler (Hubs) verbunden sind. An einem Root-Hub (in der Regel im PC) können bis zu 127 Geräte angeschlossen werden. Das Prinzip ist in Abbildung 74 skizziert.

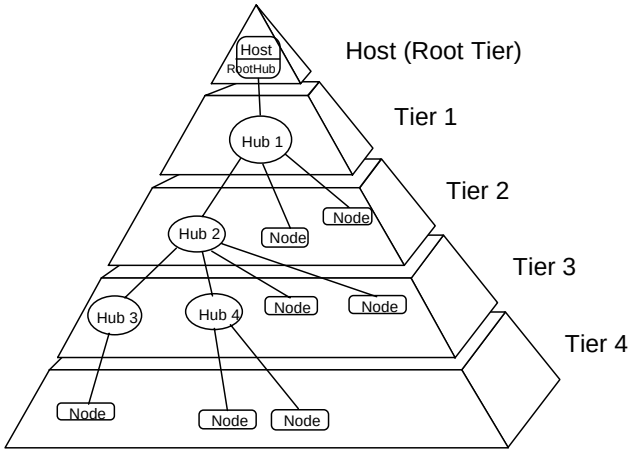


Abbildung 74: Vernetzungsstruktur eines USB-Systems.

In den Endgeräten kann die Hub-Funktionalität mit integriert sein, somit kann ein einfaches USB-System auch ohne abgesetzte Hubs aufgebaut werden. Eine mögliche Realisierung für ein PC-System zeigt Abbildung 75: Die Hauptperiphe-

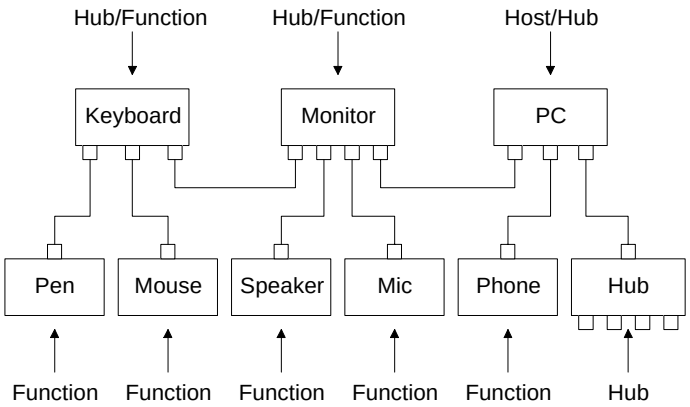


Abbildung 75: PC-System, das über USB-Verbindungen betrieben wird.

riegeräte wie Monitor und Keyboard fungieren gleichzeitig als USB-Hub, an den jeweils weitere Geräte angeschlossen sind.

Als serielles Bussystem benötigt der USB nur relativ einfache Kabelverbindungen; für den Datentransfer wird eine verdrillte Zweidrahtleitung verwendet, daneben wird noch die Versorgungsspannung von 5 V über die Buskabel geführt (Abbildung 76). Dadurch können Geräte mit geringem Stromverbrauch (max. 500 mA nach USB-2.0-Norm) auch ohne eigenes Netzteil am USB betrieben werden.

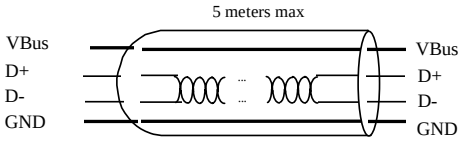


Abbildung 76: USB-Kabel: Verdrillte Datenleitungen und zwei Leitungen für die Versorgungsspannung reichen aus.

In der Spezifikation weiterhin vorgesehen ist eine *Hot-Plug*-Fähigkeit: Peripheriegeräte können während des normalen Rechnerbetriebs hinzugefügt oder entfernt werden, der lästige und zeitraubende Neustart des Systems entfällt. Die Geräte werden automatisch erkannt, der zugehörige Treiber wird geladen bzw. aktiviert oder nach dem Entfernen eines Geräts wieder deaktiviert.

Für den Bereich der Datenerfassung und Experimentsteuerung weist der USB einige interessante Vorteile auf:

- Als breit akzeptierter Standard ist die Schnittstelle überall verfügbar (Notebooks). Der Einbau von speziellen Schnittstellenkarten entfällt.
- Die notwendigen Komponenten sind billig herzustellen (einfache standardisierte Kabel, intelligente Standardperipheriebausteine).
- Einfache Systeme lassen sich aus wenigen ICs zusammenbauen, oft reicht *ein* intelligenter Peripheriebaustein aus, die Stromversorgung wird vom Bus geliefert.
- Da bis zu 127 Geräte am Bus betrieben werden können, muss man nicht

mehr möglichst viele Teilfunktionen zusammenfassen, um Platz zu sparen (Multi-IO-Karten), sondern kann die Funktionen getrennt und damit flexibler realisieren.

Für die Programmierung werden jedoch immer USB-konforme, ins Betriebssystem eingebundene Gerätetreiber nötig sein, wegen der komplexen Fähigkeiten des USB verbietet sich ein direkter Zugriff auf Geräte.

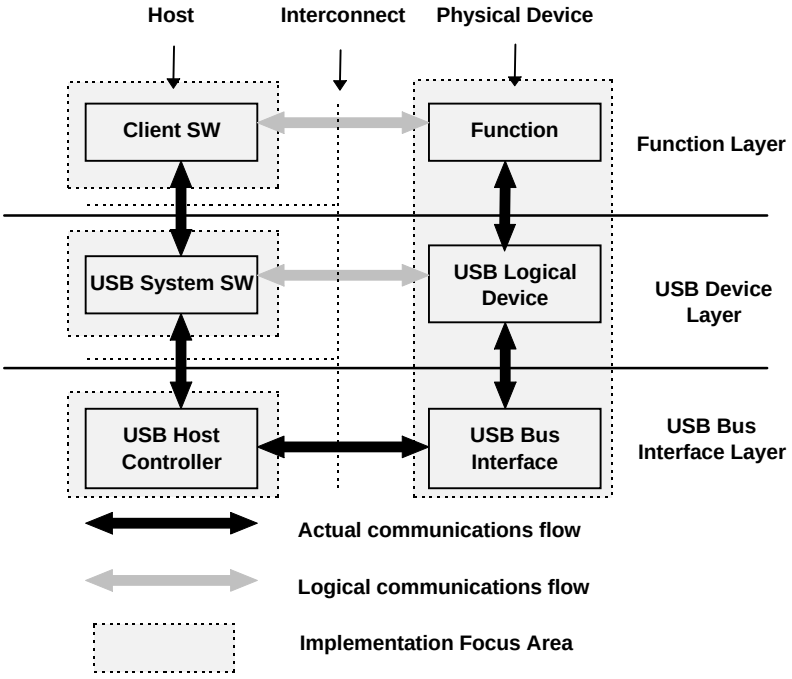


Abbildung 77: Logischer und physikalischer Datenfluss in einem USB-System.

Der Datenfluss in einem USB-System ist in Abbildung 77 dargestellt. Im intelligenten Peripheriegerät wird in der Regel nur die oberste Ebene (Function Layer) für die spezifische Aufgabe neu entwickelt, die beiden unteren Ebenen können aus vorgefertigten Moduln entnommen werden. Ähnlich ist es im Rechnersystem, auch dort wird die Software auf fertigen Moduln (DLLs und Gerätetreiber) aufsetzen. Konkret bedeutet das, dass bei kommerziellen Peripheriegeräten im Anwenderprogramm die Funktionen der mitgelieferten DLLs benutzt werden, mithin nur der logische Datenfluss der obersten Ebene programmiert wird, bei Eigenbaugeräten wird man allerdings DLL, Gerätetreiber und (zumindest) die oberste Softwareebene im Peripheriegerät selbst realisieren.

Die ursprünglichen technischen Spezifikationen des Universal Serial Bus sind sehr detailliert in [40] zusammengestellt, alle Abbildungen in diesem Einführungskapitel sind dieser Dokumentation entnommen.

7.6.2 Aktuelle Standards und Entwicklungen

Der aktuelle Entwicklungsstand des USB wird durch die Logos in Abbildung 78 gekennzeichnet.

Abbildung 78: USB-Logos für verschiedene aktuelle Gerätestandards, Beschreibung im Text (Logos aus Wikipedia).

Certified USB: Damit sind Geräte gekennzeichnet, die für USB-Low-Speed oder USB-Full-Speed zertifiziert sind.

Certified High-Speed USB: Geräte, die für den Hochgeschwindigkeitsbereich zertifiziert sind und diesen Bereich auch tatsächlich nutzen. USB 2.0 bedeutet genau genommen nur, dass Geräte mit der neueren Norm USB 2.0 kompatibel sind, die deutlich engere Toleranzanforderungen an die Übertragungsgeschwindigkeiten setzt als vorher USB 1.1. Es bedeutet nicht unbedingt, dass solche Geräte auch die hohe Geschwindigkeit ausnutzen.

USB On-The-Go: Mit solchen Schnittstellen ausgerüstete Geräte können in begrenztem Host-Funktionalität übernehmen. Dadurch sind diese Geräte in der Lage, unmittelbar, d. h. ohne die Vermittlung eines Rechners, miteinander kommunizieren. So wird beispielsweise direktes Drucken von einer Digitalkamera auf einen Drucker möglich.

Wireless USB: Derzeit in der Entwicklung, Ziel ist es, im USB-High-Speed-Bereich, d. h. mit einer Geschwindigkeit von 480 MBit/s drahtlos zu übertragen. Geplante Reichweiten sind etwa 10 m.

USB 3.0: Ein Konsortium aus mehreren Firmen (vgl. Abbildung rechts) bereitet derzeit einen neuen sehr schnellen USB-Standard vor. Die Übertragungsgeschwindigkeit wird etwa 10 mal so hoch sein wie beim HighSpeed-USB, geplant sind 5 GBit/s. Die Übertragung wird mit Lichtleitern erfolgen, die zusätzlich in USB-Stecker integriert werden. Die Spezifikationen für USB 3.0 sollen nach Angaben des Firmenkonsortiums 2008 veröffentlicht werden.

In den folgenden Abschnitten soll an einigen Beispielen die Software-Einbindung von USB-Geräten in Messsysteme (Messprogramme) erläutert werden.

7.6.3 Grundsätzliche Aspekte

Die seriellen und die parallelen Standard-Schnittstellen des PC sind vergleichsweise transparent integriert und dadurch auch meist ohne umfangreiche Zusatzsoftware zu bedienen. Bei GPIB und USB ist dies nicht mehr der Fall. GPIB benötigt spezialisierte Hardware, die nur über spezialisierte Software angesprochen werden kann – Kernel-Treiber und darauf aufsetzende DLLs. USB ist zwar grundsätzlich standardisiert, jedoch werden auch hier gerätespezifische Treiber benötigt, die auf die Eigenschaften der externen Geräte abgestimmt sind.

Die jeweils benötigte Software wird von den Hardware-Herstellern mitgeliefert, allerdings nicht auf die Verwendung in selbstgeschriebenen Messprogrammen oder in Skripten unter MATLAB abgestimmt[41]. Fast immer ist eine unter C oder C++ verwendbare Software-Bibliothek als DLL (*D*ynamic *L*ink *L*ibrary) vorhanden. Diese wird bei der Software-Installation in aller Regel schon an die richtige Stelle kopiert (z. B. ins entsprechende Betriebssystemverzeichnis oder – bei freundlicheren Programmen – in ein Installationsverzeichnis, das dann in der Umgebungsvariablen *Path* mit vermerkt wird). Damit kann ein anderes Programm problemlos auf die Funktionen der DLL zugreifen. Die bereitgestellten Funktionen sind in einer Deklarations-Datei (*Header*-Datei `*.h`) beschrieben, die im verwendenden Programm mit einer `#include`-Anweisung eingebunden wird.

Die beiden Komponenten, DLL und Deklaration in der Header-Datei, reichen grundsätzlich schon aus, um die Hardware von einem C-Programm aus zu verwenden. Für gängige C-Compiler (Microsoft Visual Studio, seltener auch Borland C) wird jedoch meist noch eine LIB-Datei mitgeliefert, die ein komfortables Interface zwischen Progamm und DLL liefert. Man braucht sich dann beispielsweise nicht mehr darum zu kümmern, dass die DLL rechtzeitig geladen wird. Fast alle Hersteller liefern auch Beispielprogramme mit, deren Analyse das Schreiben von eigenen Programmen sehr erleichtert. Wir werden in den im Folgenden beschriebenen Anwendungen immer davon ausgehen, dass beides – LIB-Datei und ausführliche Beispielprogramme – vom Hersteller mitgeliefert wurden und werden beides verwenden.

Die Programmierung einer konkreten Applikation – beispielsweise für die Verwendung in einem MATLAB-Messprogramm – wird in etwa in den folgenden Schritten erfolgen:

[41] Mathworks bietet zwei Toolboxes zu MATLAB an, die die Bedienung der Hardware einiger Hersteller und die Kommunikation mit Messgeräten mit einfachen MATLAB-Anweisungen ermöglichen, die *Instrument Control Toolbox* und die *Data Acquisition Toolbox*.

1. Analyse von Beispielprogrammen, Studium der Dokumentation zur DLL, Studium der Deklarationsdatei(en).
2. Studium der Dokumentation des Messgeräts (welche Anweisungen gehen ans Messgerät, was schickt das Gerät zurück?).
3. Kompilieren und Binden eines Beispielprogramms in der konkreten Entwicklungs-Umgebung (z. B. M$ Visual Studio). Fehlerfrei lauffähig?
4. Modifikation des Beispielprogramms auf das konkrete Messgerät und auf die konkrete Aufgabe hin. Immer noch fehlerfrei?
5. DLL für die Verwendung unter MATLAB.

7.6.4 *Uneigentliche* USB-Geräte

Der Wegfall serieller Schnittstellen an PCs und vor allem an Notebooks stimulierte die Entwicklung preisgünstiger integrierter Bausteine, die eine *Übersetzung* zwischen serieller Schnittstellennorm und USB ermöglichen (für die dort benötigten Übertragungsgeschwindigkeiten ist schon der langsame Standard USB 1.1 ausreichend). Um sich eine zusätzliche serielle Schnittstelle zu verschaffen, genügt ein Konverterkabel zwischen USB und 9-poligem seriellem Anschluss. Die komplette Konverter-Elektronik ist üblicherweise im Gehäuse des 9-poligen D-Steckers eingegossen (Abbildung rechts). Zur Verwendung muss natürlich noch die mitgelieferte Treibersoftware installiert werden, dann erscheint der Anschluss für den Rechner wie eine der seriellen Schnittstellen. Deren Konfiguration und Betrieb ist für die Software, die darauf zugreift, völlig transparent, es sind keinerlei USB-Spezifika zu berücksichtigen.

Abbildung 80: USB-Seriell-Konverter.

Die günstigen ICs haben verschiedene Messgerätehersteller veranlasst, die ursprüngliche serielle Schnittstelle an ihren Messgeräten durch eine (scheinbare) USB-Schnittstelle zu ersetzen. Leider wird die Verbindung dadurch weder beschleunigt noch auf andere Weise verbessert. Man hat nach wie vor die Nachteile der seriellen Verbindung, muss beispielsweise selbst erkennen, wann die Übertragung zu Ende ist, für den richtigen Quittungsbetrieb (Handshake) sorgen, die Übertragungsgeschwindigkeit (Baud-Rate) passend einstellen u. ä. Ein großer Vorteil ist allerdings, dass man auf beiden Seiten der Verbindung die für die serielle Schnittstelle ansonsten benötigten zusätzlichen positiven und negativen Betriebsspannungen einspart.

Sie erkennen solche *uneigentlichen* USB-Schnittstellen, wenn Sie die Treibersoftware installieren, danach haben Sie eine weitere serielle Schnittstelle im System. Die Bedienung erfolgt, wie im Abschnitt zur seriellen Schnittstelle beschrieben, es sind keinerlei USB-Spezifika zu berücksichtigen.

7.6.5 National Instruments NI USB-6008

Verschiedene Hersteller von PC-Einsteckkarten zur Ein- und Ausgabe von analogen und digitalen Signalen bieten inzwischen externe Module mit ähnlicher Funktionalität an, die über USB an Rechner angeschlossen werden. Der Anschluss über USB bietet dabei den großen Vorteil, dass keine zusätzlichen Betriebsspannungen benötigt werden.

Als ein Beispiel wollen wir uns das Ein/Ausgabe-Modul NI USB-6008 der Firma *National Instruments* ansehen. Dieses Interface bietet eine preiswerte Möglichkeit, A/D- und D/A-Wandler sowie digitale Ein- und Ausgänge am PC einzurichten. Es enthält 8 gemultiplexte A/D-Kanäle mit 12 Bit Auflösung, die auch differentiell betrieben werden können, 2 D/A-Kanäle mit ebenfalls 12 Bit Auflösung und 12 digitale E/A-Leitungen, eine davon kann als Zähleingang verwendet werden. Abbildung 81 zeigt den Innenaufbau, Abbildung 82 das Blockschaltbild.

Abbildung 81: Innenansicht des NI USB-6008, oben die USB-B-Buchse zum Anschluss an den PC.

Mit dem Interface erhält man eine CD mit allen notwendigen Treibern und mit einer umfangreichen Bibliothek zum Betrieb des NI USB-6008. Neueste Software findet man beim Hersteller National Instruments (www.ni.com).

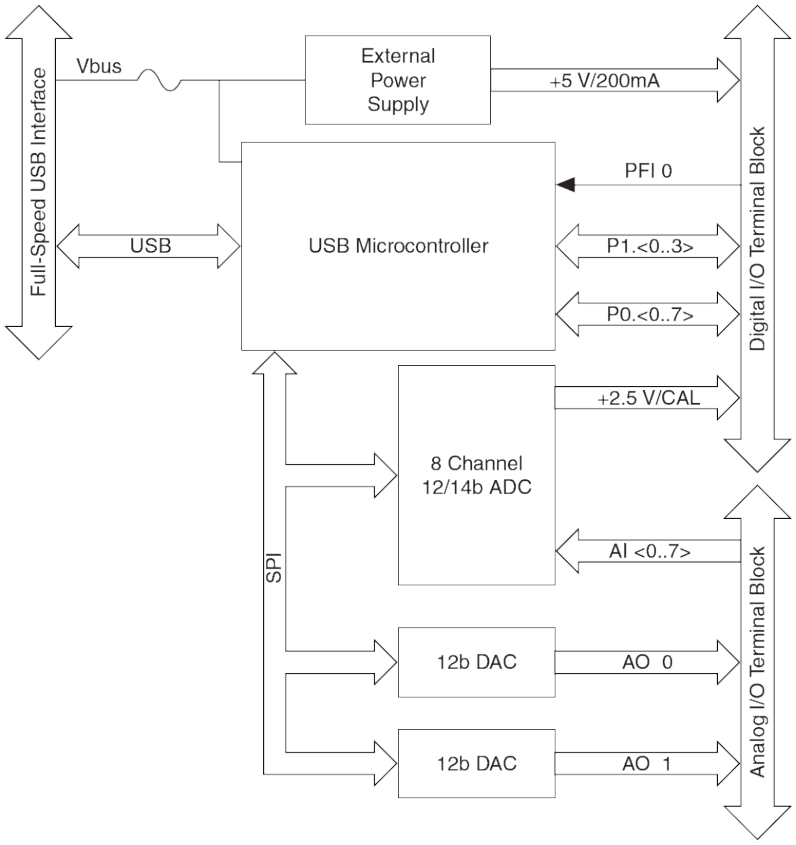


Abbildung 82: Blockschaltbild des NI USB-6008.

7.6.5.1 Beispielprogramm für die Analog-Ausgabe Vom Hersteller sind verschiedene Beispielprogramme mitgeliefert, eines davon für die Ausgabe eines Analogsignals. Das Listing finden Sie in Anhang C. Abgesehen von der erst im Programm definierten Wrapper-Funktion für die Fehlerbehandlung (`DAQmxErrChk`) sind alle `DAQmx...`-Funktionen sehr ausführlich in der Dokumentation beschrieben. Abbildung 83 zeigt als Beispiel die Beschreibung der Funktion `DAQmxWriteAnalogF64`.

Offenbar muss bei der Programmierung eine feste funktionale Abfolge eingehalten werden, die im Kommentar am Beginn des Beispiels in Anhang C beschrieben ist.

7.6.5.2 Übersetzen und Binden mit Visual Studio Nach dem Start von Visual Studio wird ein neues C/C++-Projekt angelegt, Typ *WIN32 Console Application*.

Dies wird im ersten Konfigurationsfenster (Abbildung 84 festgelegt. Im folgenden Fenster (Abbildung 85 werden dann weitere Optionen ausgewählt: *Console application* und *Empty project*. Damit ist nun ein leeres Projekt begonnen, die für ein einfaches Konsolprogramm benötigten Optionen sind eingestellt.

Über *Project* ⇒ *Add Existing Item* wird nun das Beispielprogramm ins Projekt

DAQmxWriteAnalogF64

int32 DAQmxWriteAnalogF64 (TaskHandle taskHandle, int32 numSampsPerChan, bool32 autoStart, float64 timeout, bool32 dataLayout, float64 writeArray[ ], int32 *sampsPerChanWritten, bool32 *reserved);

Purpose

Writes multiple floating-point samples to a task that contains one or more analog output channels.

Note Buffered writes require a minimum buffer size of two samples.

Parameters

Input

Name	Type	Description
taskHandle	TaskHandle	The task to write samples to.
numSampsPerChan	int32	The number of samples, per channel, to write. You must pass in a value of 0 or more in order for the sample to write. If you pass a negative number, this function returns an error.
autoStart	bool32	Always set to FALSE.
timeout	float64	The amount of time, in seconds, to wait for this function to write all the samples. This function returns an error if the timeout elapses.
dataLayout	bool32	Specifies how the samples are arranged, either interleaved or non-interleaved.
		Value / **Description**: DAQmx_Val_GroupByChannel — Group by channel (non-interleaved); DAQmx_Val_GroupByScanNumber — Group by sample (interleaved)
writeArray	float64[ ]	The array of 64-bit samples to write to the task.
reserved	bool32 *	Reserved for future use. Pass NULL to this parameter.

Output

Name	Type	Description
sampsPerChanWritten	int32 *	The actual number of samples per channel successfully written to the buffer.

Return Value

Name	Type	Description
status	int32	The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

Abbildung 83: Beschreibung der Funktion `DAQmxWriteAnalogF64` in der Dokumentation zum NI USB-6008.

eingefügt. Sinnvollerweise wurde es vorher in das aktuelle Arbeitsverzeichnis kopiert. Ein Übersetzungsversuch liefert Fehlermeldungen, die Deklarationsdatei `NIDAQmx.h` ist nicht zu finden. Den Pfad dorthin muss man Visual Studio über *Project* ⇒ *Properties* mitteilen (Abbildung 86).

In ähnlicher Weise werden die Spezialitäten für den Linker eingestellt: Die LIB-Datei `nidaqmx.lib` bei *Additional Dependencies* auf der *Input*-Karte für den Linker und das zugehörige Verzeichnis unter *Additional Library Directories* auf der *General*-Karte für den Linker.

Damit sollte das Projekt fehlerfrei übersetzen und linken. Wenn es dann auch noch richtig läuft (*Debug* ⇒ *Start Without Debugging*) und das NI USB-6008 das richtige tut, kann man davon ausgehen, dass alles richtig installiert ist – Hardware und Software.

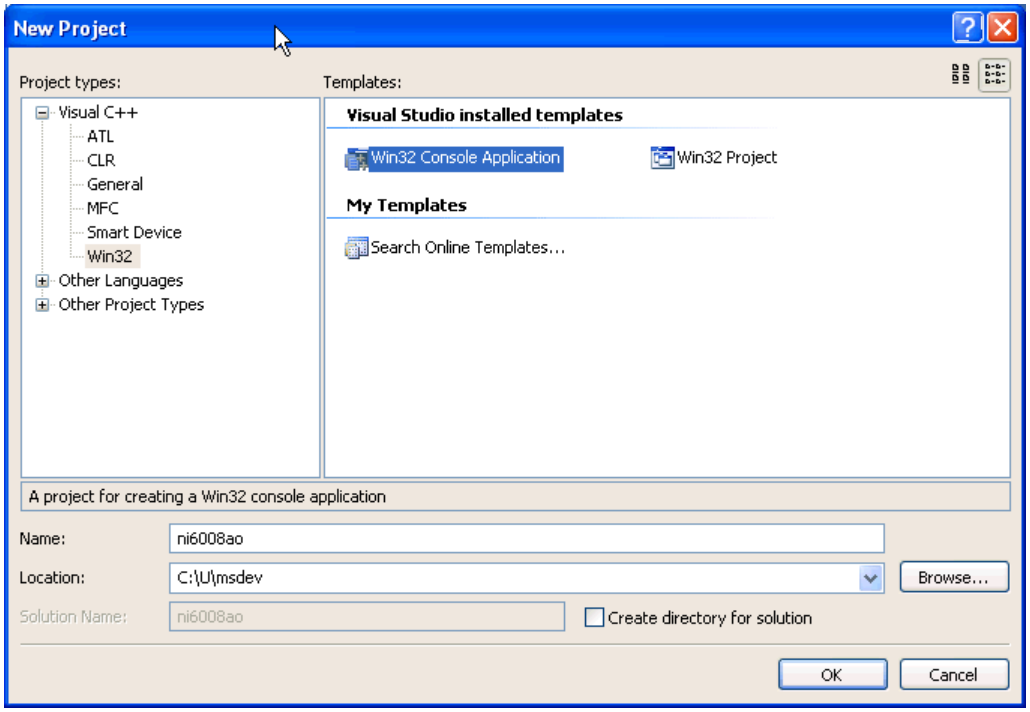


Abbildung 84: Konfigurationsfenster für ein neues Projekt unter Visual Studio

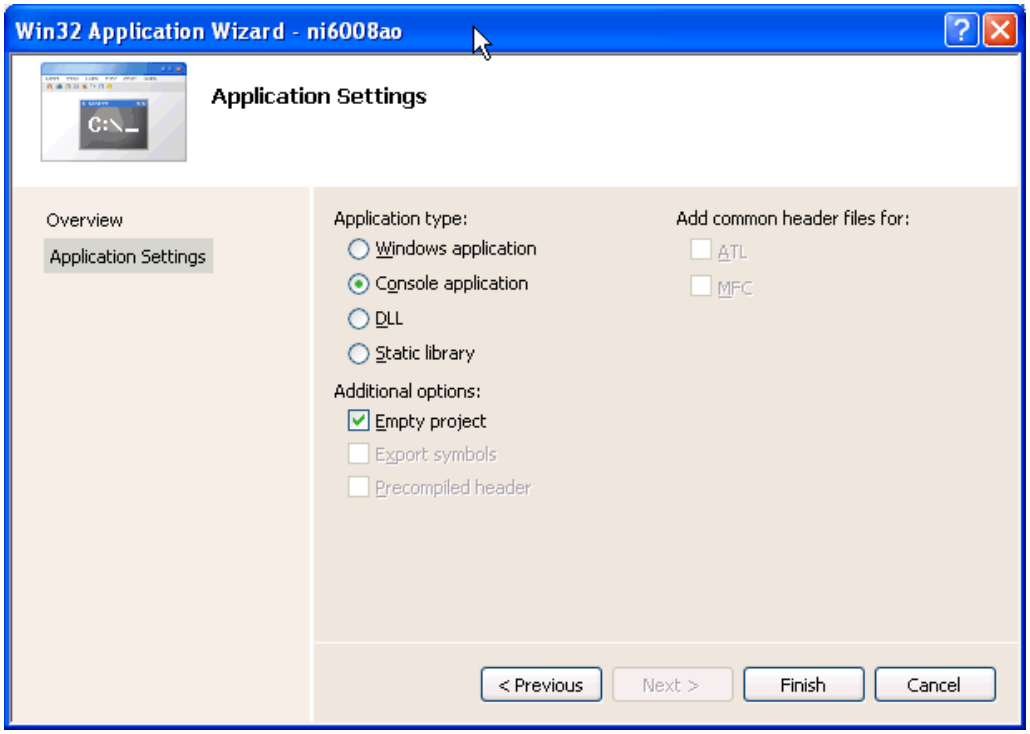


Abbildung 85: Detailkonfiguration des Programms, das erstellt werden soll.

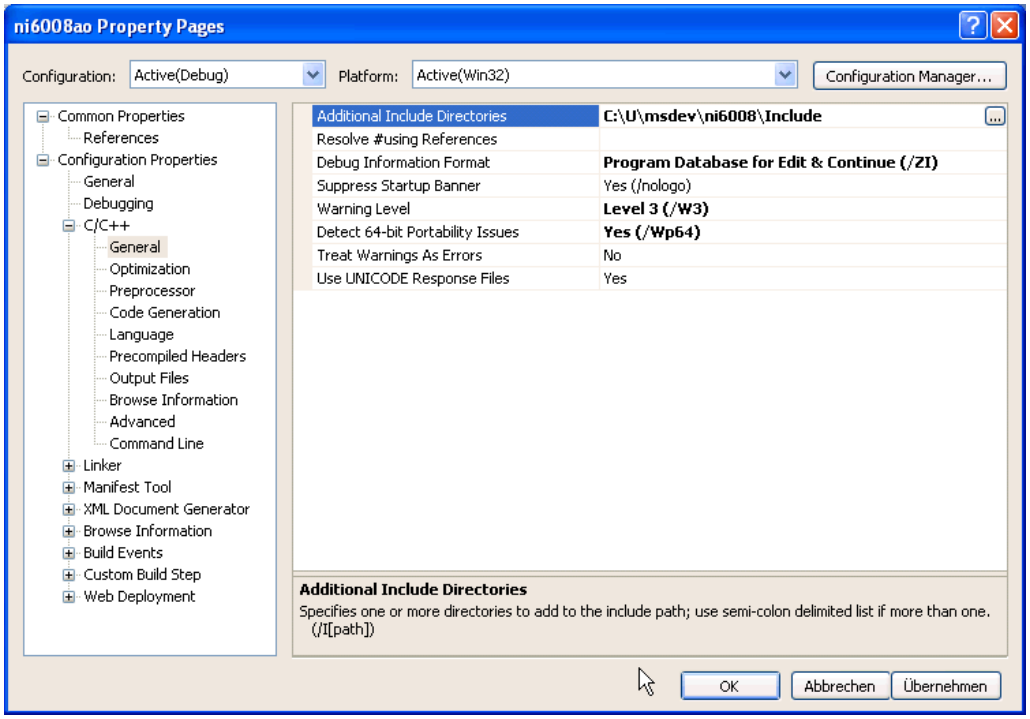


Abbildung 86: Einstellung zusätzlicher Optionen für das Projekt, hier das Include-Verzeichnis, in dem die Deklarationsdatei `NIDAQmx.h` *zu finden ist.*

7.6.5.3 Programm-Modifikation Wenn alles läuft, macht die Hardware das, was im Beispielprogramm vorgesehen ist, Ausgabe eines festen Spannungswerts von 5 V auf D/A-Kanal 0. Das ist uns natürlich zu langweilig, wir müssen das Programm so ändern, dass das produziert wird, was wir gerne hätten. Es soll beispielsweise eine Sinus-Spannung ausgegeben werden, auf Kanal 0 direkt, auf Kanal 1 um π phasenverschoben.

Die zusätzlich benötigten bzw. modifizierten Programmzeilen könnten etwa so aussehen:

```
#include <conio.h>
#include <math.h>

#define  N (int32)100
#define  A (float64)2.5
const float64 pi = 4*atan(1);
char          chan[] = "Dev1/ao0:1";
float64       data[2*N] = {0};
int           k;

for (k=0;k<N;k++)
  data[k] = A + A*sin(2*pi*k/N);
for (k=N;k<2*N;k++)
  data[k] = A - A*sin(2*pi*k/N);
```

```
while (!_kbhit())
  DAQmxWriteAnalogF64(taskHandle,samplesPerChan,0,timeout,
      DAQmx_Val_GroupByChannel,data,&pointsWritten,NULL);
```

Die Sinus-Werte sind im Feld `data` in zwei Gruppen für die beiden Kanäle abgelegt, `DAQmx_Val_GroupByChannel` sorgt dafür, dass das Datenfeld richtig interpretiert wird. Das Programm läuft endlos, bis im Eingabe-Fenster eine Taste gedrückt wird.

Damit hat man zwei Sinus-Generatoren, die gegenphasig zwischen 0 und 5 V arbeiten. Eine Überprüfung mit dem Oszilloskop bestätigt das (Abbildung 87).

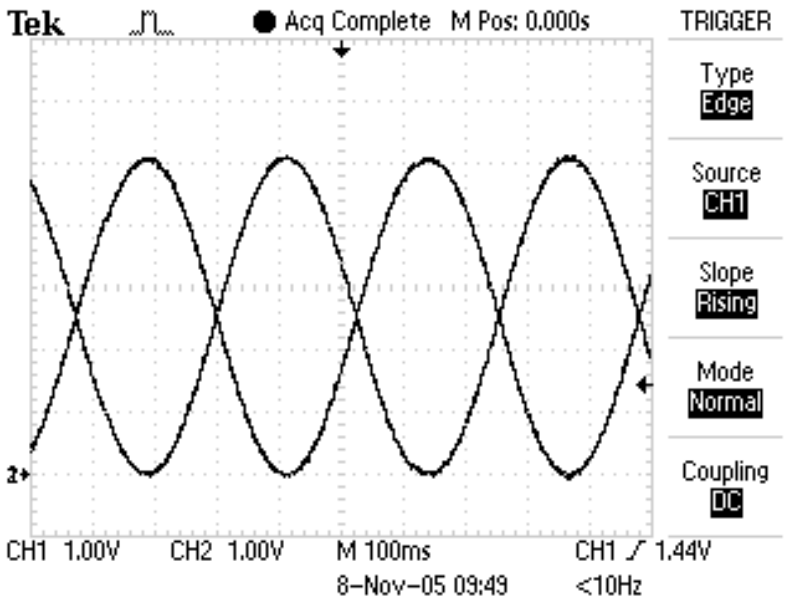


Abbildung 87: Ausgangssignale der beiden D/A-Wandler nach der Änderung des Beispielprogramms.

7.6.5.4 Programm für die Analog-Eingabe Das Listing für das Beispielprogramm ist in Anhang D abgedruckt. Übersetzen und Binden des Programms schaffen wir nach den Vorkapiteln problemlos. Wir modifizieren das Programm so, dass zwei differentielle Kanäle jeweils eine Datenfolge aufnehmen:

```
uInt32      samplesPerChan = 500;
uInt32      bufferSize = 2*samplesPerChan;
uInt32      pointsToRead = samplesPerChan;
float64     sampleRate = 5000.0;
char        chan[] = "Dev1/ai0:1";

DAQmxReadAnalogF64 (taskHandle, pointsToRead,
    timeout, 0, data, bufferSize, &read, NULL);
```

Eingestellt sind somit 500 Datenpunkte pro A/D-Kanal und eine Messrate von 5 kHz.

Die Daten könnten wir als Tabelle nach `stdout` oder in eine Datei schreiben. Schöner wäre eine direkte Verbindung zu MATLAB.

7.6.5.5 DLL für MATLAB: MEX-Datei Dafür sind nur ein paar kleine Änderungen nötig. Statt `main` wird `mexFunction` als Einsprungfunktion benötigt, und die Daten müssen geeignet an MATLAB übergeben werden:

```
#include "mex.h"

void mexFunction( int nlhs, mxArray *plhs[],
            int nrhs, const mxArray *prhs[] )

if (nrhs>0)
    samplesPerChan = mxGetScalar(prhs[0]);

plhs[0] = mxCreateDoubleMatrix(samplesPerChan,2, mxREAL);
double * data = mxGetPr(plhs[0]);
```

Das ist alles, was an Änderungen nötig ist. Übersetzt und zusammengepackt wird das Ganze zweckmäßigerweise aus MATLAB mit einem `mex`-Aufruf:

```
mex -I../Include -L../Lib -lnidaqmx.lib ai.cpp
```

`ai.cpp` ist der Name der C++-Quelldatei, `-I` und `-L` erweitern Include- und Linker-Pfad, `-l` benennt zusätzliche Bibliotheksdateien. Näheres dazu unter `help mex` in MATLAB.

Von MATLAB aus können wir die Funktion mit `A=ai(N)` aufrufen. `N` legt die Zahl der Messpunkte fest, in `A` erhalten wir die Messdaten, eine `N*2`-Matrix.

7.6.5.6 Messbeispiel Mit den nun verfügbaren Programmen bauen wir eine kleine Messumgebung auf. Die Ausgangssignale kommen an eine Reihenschaltung aus Widerstand und Diode, gemessen werden die an den beiden Bauelementen anliegenden Spannungen (Abbildung 88).

Die gemessenen Spannungen entsprechen den Erwartungen (Abbildung 89). Man erkennt die Auswirkungen der Diodenkennlinie.

Die Diodenkennlinie selbst erhält man, wenn man den Diodenstrom über der Diodenspannung aufträgt. Der Strom wird aus der Spannung am Widerstand und dem Widerstandswert berechnet. Ein MATLAB-Fragment, das alles Nötige erledigt, könnte so aussehen:

```
dos('ao &');              % analog output, background task
R = 0.470;                % resistor to measure current
xlim = [-5,5];
```

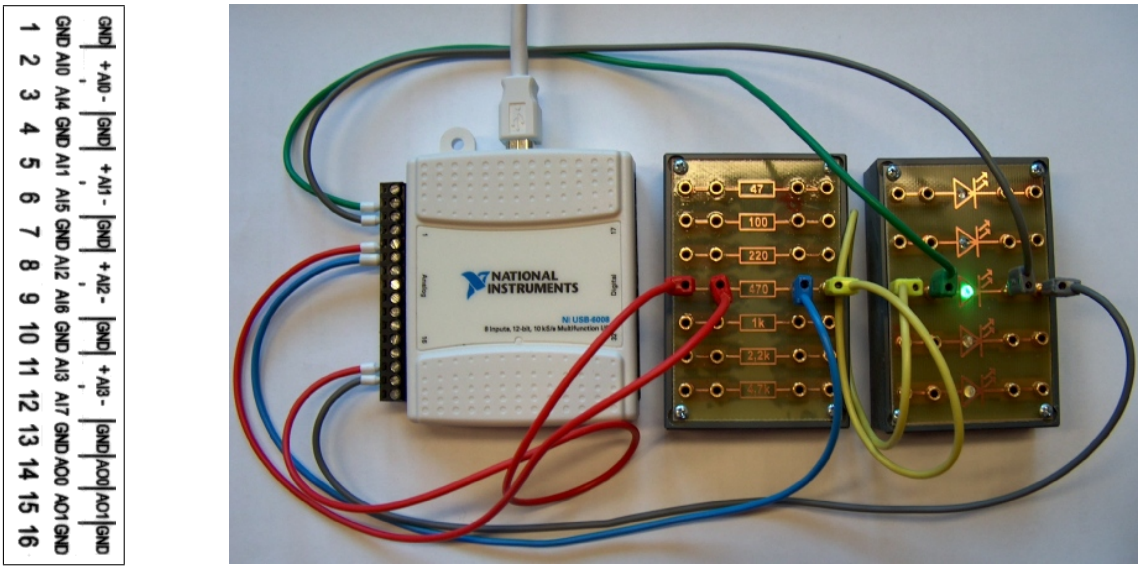


Abbildung 88: Messschaltung mit dem NI USB-6008: Die beiden Ausgänge liegen an der Reihenschaltung Widerstand und Diode an, die Eingänge messen die beiden Teilspannungen. Links die Steckerbelegung des Analogteils.

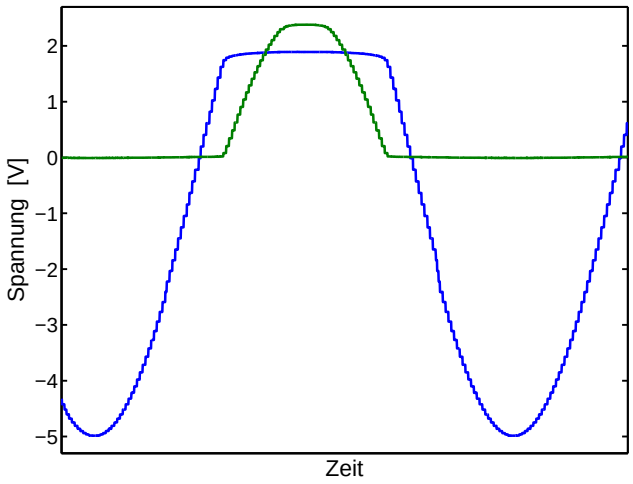


Abbildung 89: Spannungen an Diode (blau) und Widerstand (grün) bei sinusförmiger Gesamtspannung.

```
ylim = [-1,3];
while 1,                    % loop until Ctrl-C
    a = ai(2000);           % analog input
    x = sortrows(a,1);      % sort data for plotting
    plot(x(:,1),x(:,2)/R,'Linewidth',3);
    set(gca,'XLim',xlim,'YLim',ylim);
    hold on;
    plot(xlim,[0,0],'k',[0,0],ylim,'k');
    hold off;
    drawnow;
end;
```

`dos` ruft ein anderes Programm auf und lässt es mit `&` weiter laufen, ohne auf die Beendigung zu warten. Mit `ai(2000)` werden 2000 Datenpunkte gemessen, die dann mit `sortrows` nach Spannungswerten sortiert werden. Die damit gemessenen Kennlinien unterschiedlicher Lumineszenzdioden und einer Photodiode bei unterschiedlichen Beleuchtungsstärken zeigen die Abbildungen 90 und 91.

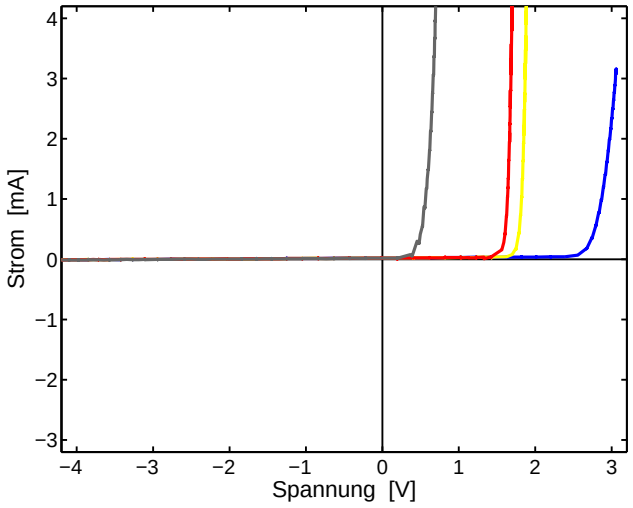


Abbildung 90: Kennlinien von Lumineszenzdioden unterschiedlicher Farbe. Zum Vergleich in Grau die Kennlinie einer Siliziumdiode.

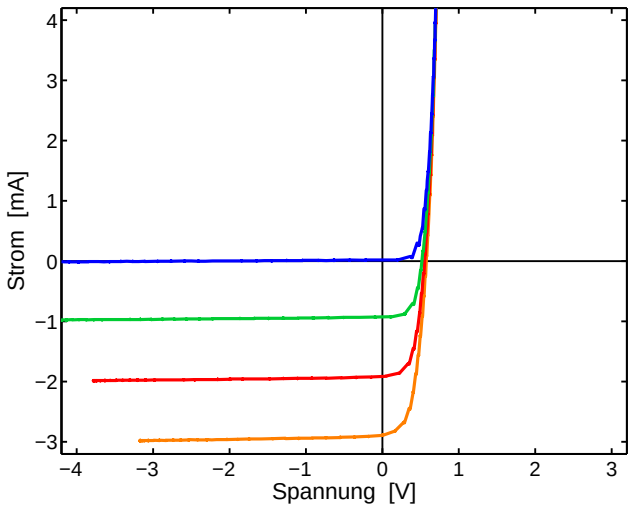


Abbildung 91: Kennlinien einer Si-Photodiode bei unterschiedlichen Beleuchtungsstärken (Blau: Ohne Beleuchtung).

7.6.6 Messgeräte-Standard VISA

Verschiedene Messgeräte-Hersteller haben sich darauf verständigt, zu ihren Messgeräten Treibersoftware zu liefern, die von Programmen, die darauf zugreifen, weitgehend einheitlich (d. h. Hersteller übergreifend) angesprochen werden kann. Die Gerätespezifika – insbesondere die hardwarenahen Spezialitäten der Geräteanbindung – werden in den jeweiligen DLLs implementiert und brauchen vom Anwender nicht im Einzelnen beachtet zu werden. Die Treibersoftware erledigt somit auch die Anbindung der Schnittstellen, über die die Messgeräte mit dem Rechner verbunden sind.

Dieser Standard – *V*irtual *I*nstrument *S*oftware *A*rchitecture (VISA) – wird von der *Interchangeable Virtual Instrument Foundation* [41], einem offenen Zusammenschluss von Herstellerfirmen, spezifiziert und laufend an neue Entwicklungen angepasst. Sich selbst definiert die IVI-Foundation auf ihrer Eingansseite [41] wie folgt:

> *The IVI Foundation is an open consortium founded to promote specifications for programming test instruments that simplify interchangeability, provide better performance, and reduce the cost of program development and maintenance.*
>
> ***IVI Overcomes Industry Challenges***
>
> *In today's world two factors hinder efficient test system setup and support: 1) the high cost of developing and maintaining test system software and, 2) rapidly evolving technology. The IVI Foundation addresses these needs through new driver technology:*
>
> - *IVI drivers define a new level of quality, completeness, usability, and functionality that reduces the cost of test system development and ownership.*
> - *IVI drivers simplify upgrading or replacing components in complex test systems intended to be used over a long period of time.*
>
> *The IVI Foundation was formed in 1998 and officially incorporated in 2001. Its membership includes end-users, instrument vendors, software vendors, system suppliers, and system integrators.*

Um VISA-Geräte in Messprogramme zu integrieren, studiert man sinnvollerweise zunächst Beispielprogramme und ändert diese dann in gewünschter Weise ab.

Ein einfaches Beispielprogramm, das bei VISA-Installationen mitgeliefert wird, ist in Anhang E gelistet. Das Programm `FindRsrc.c` sucht nach Messgeräten am PC, die über VISA-Gerätetreiber erreichbar sind. Am Programmlisting erkennt man die Vorgehensweise, die Programmabfolge ist kommentiert. Das Übersetzen und Binden des Programms läuft analog zu 7.6.5.2, die Pfade zur Include-Datei `visa.h` und zur LIB-Datei `visa32.lib` sind geeignet einzustellen, letztere ist als Bibliotheksdatei explizit einzubinden.

Bei Ausführung meldet das Programm die angeschlossenen Geräte, ein Messgerät ist über USB angeschlossen:

```
ASRL1::INSTR
ASRL3::INSTR
ASRL10::INSTR
USB0::0x0957::0x0407::MY43003008::INSTR
```

`ASRLx` sind die seriellen Schnittstellen, bei denen nicht geprüft wird, ob tatsächlich Messgeräte dort angeschlossen sind. Dafür gibt es keine eindeutige Strategie. `USB` ist die USB-Schnittstelle, dort wird geprüft, ob Geräte angeschlossen sind, die USB-Kennung der angeschlossenen Geräte wird angezeigt.

Diese Kennung ist zu verwenden, um die Geräte an der Schnittstelle unter VISA zu adressieren. Im Folgenden zwei kurze Beispiele zur Gerätekommunikation, die Programmierung des Funktionsgenerators 33220A von Agilent und das Einlesen von Messdaten vom Oszilloskop TDS 2012B von Tektronix.

7.6.7 Agilent Funktionsgenerator 33220A

Agilent Technologies

Der Funktionsgenerator 33220A (Abbildung 92) ist sehr vielseitig verwendbar, neben Standardfunktionen wie Sinus, Rechteck, Dreieck, Impuls sind auch beliebige Kurvenformen programmierbar. Der Frequenzbereich umfasst 0 bis 20 MHz. Ausführliche Informationen liefert das Handbuch zum Gerät.

Abbildung 92: Frontansicht des Funktionsgenerators 33220A der Firma Agilent.

Als Basis zur Programmierung wird ein Beispielprogramm für das Schreiben und Lesen auf ein VISA-Gerät verwendet (`RdWrt.c`, Listing in Anhang F). Es sind nur kleine Modifikationen erforderlich. Man benötigt nur die Ausgabe einer Anweisung an den Funktionsgenerator, die Anweisung soll variabel beim Programmaufruf als Parameter mitgegeben werden können. Das erreicht man beispielsweise mit (`genwr.c`)

```
int main(int argc, char * argv[])
{
  status = viOpenDefaultRM (&defaultRM);
  status = viOpen (defaultRM, "USB0::0x0957::0x0407::MY43003008::INSTR",
      VI_NULL, VI_NULL, &instr);
  status = viSetAttribute (instr, VI_ATTR_TMO_VALUE, 5000);
  if (argc<2)
    strcpy_s(stringinput, sizeof(stringinput), "APPL:SIN 3000,10");
  else
    strcpy_s(stringinput, sizeof(stringinput), argv[1]);
  status = viWrite (instr, (ViBuf)stringinput,
            (ViUInt32)strlen(stringinput), &writeCount);
  viClose(instr);
  viClose(defaultRM);
```

```
    return 0;
}
```

Fehlerbehandlung und Kommentare sind im Listing weggelassen. Ohne Kommandozeilenparameter wird 3000 Hz Sinusspannung mit einer Amplitude von 10 V eingestellt (zur genauen Befehlssyntax ist das Studium des Gerätehandbuchs empfehlenswert). Der Bereich mit `viRead` aus dem Beispielprogramm wird weggelassen.

7.6.8 Tektronix Oszilloskop TDS 2012B

Die Oszilloskope TDS 10XX und TDS 20XX von Tektronix sind Digitalspeicheroszilloskope, die für Messaufgaben bis etwa 200 MHz ein relativ gutes Preis-Leistungs-Verhältnis bieten. Zwei oder vier Kanäle, Monochrom- oder Farb-Darstellung und (Analog-)Bandbreiten zwischen 40 und 200 MHz bieten eine breite Auswahl an Möglichkeiten (und Preisen). Abbildung 93 zeigt das Modell TDS 2012B mit einer Analog-Bandbreite von 100 MHz, zwei Kanälen und Farb-Display.

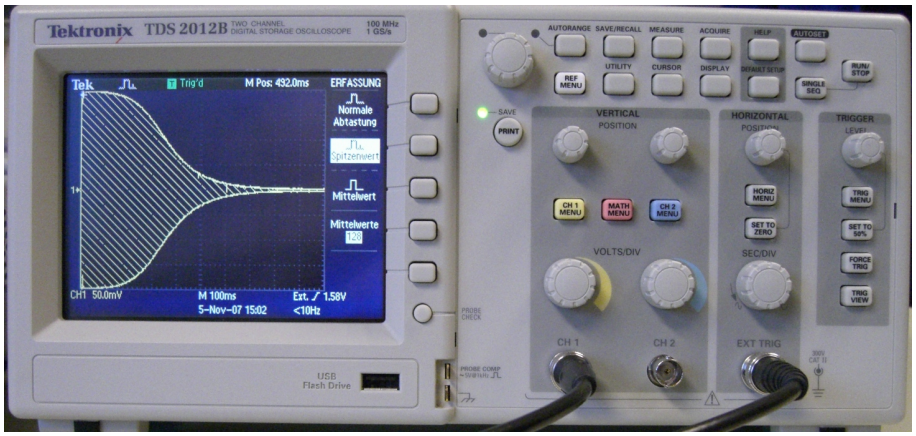


Abbildung 93: Frontansicht des Digitalspeicheroszilloskops TDS 2012B der Firma Tektronix.

Auch für das Oszilloskop wird ein kurzes Leseprogramm aus dem Beispielprogramm des Anhang F entwickelt. Ergänzend zu vorstehendem Listing für den Funktionsgenerator werden die Daten hierbei mit `viRead` gelesen.

```
status = viOpen (defaultRM, "USB0::0x0699::0x0367::C030104::INSTR",
    VI_NULL, VI_NULL, &instr);
strcpy_s(stringinput, sizeof(stringinput), "CURVE?");
status = viWrite (instr,( ViBuf)stringinput,
          (ViUInt32)strlen(stringinput),& writeCount);
status = viRead (instr, buffer, sizeof(buffer), &retCount);
viClose(instr);
```

Weitergehende Einstellungen (Kanalwahl, X- und Y-Ablenkung etc.) sind der Übersichtlichkeit halber weggelassen; die zuständigen Anweisungen finden Sie im ausführlichen Handbuch.

Will man die Daten weiter verarbeiten, ist der Inhalt von `buffer` noch näher zu untersuchen. Hat man das Oszilloskop auf binäre Daten eingestellt, so werden diese als Block aus Datenbytes übertragen, ein solcher Block hat laut Tektronix-Handbuch das Format

<Block> ::= #<NZDig><Dig>(<Dig> ...)(<DChar> ...)

Nach einem # folgt ein *Non-Zero Digit*, das angibt, wieviele *Digits* folgen, die als Zahl gelesen wiederum angeben, wieviele *DatenCharakter* folgen.

Ein Block aus 500 Datenbytes B würde mithin so beginnen

#3500BBBBBBBBB ...

Dieser Datenblock kann entweder gleich im C++-Programm interpretiert werden oder auch unverarbeitet gespeichert bzw. an ein übergeordnetes Messprogramm weiter gegeben werden. Die Dekodierung in einem MATLAB-Skript wird im folgenden Abschnitt beschrieben.

7.6.9 Messbeispiel: MATLAB und USB-Geräte

Um von MATLAB auf USB-Geräte zuzugreifen, wurde eine einfache MEX-Funktion entwickelt, die ähnlich wie die im Abschnitt 7.5.3 beschriebene Funktion `siclio` arbeitet. Die Funktion `visaio` ist im Anhang G gelistet. Sie wird im Messbeispiel verwendet, um einen Funktionsgenerator zu programmieren und Daten von einem Oszilloskop abzurufen. Gemessen wird der Frequenzgang eines RC-Tiefpasses, die Messanordnung ist in Abbildung 94 skizziert. Für die Messung wird der Funktionsgenerator im *SWEEP*-Modus betrieben, innerhalb einer Sekunde wird die Frequenz von 1 kHz bis 10 MHz geändert.

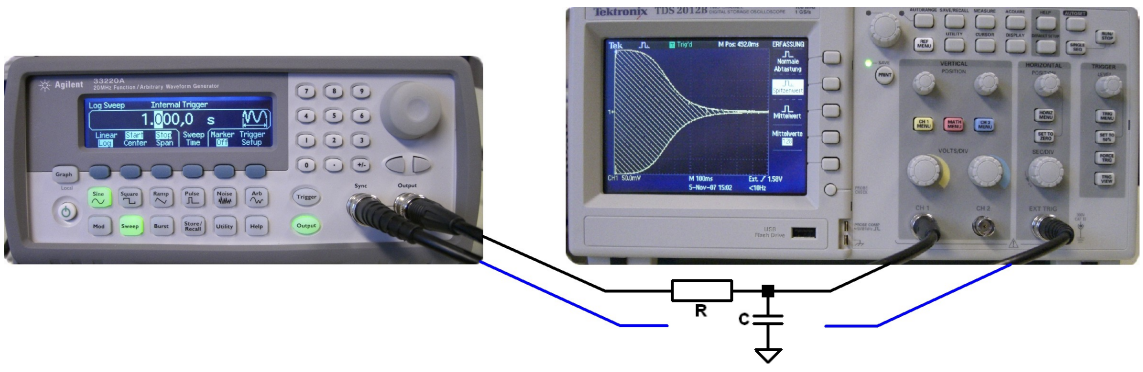


Abbildung 94: Messanordnung für den Frequenzgang (Amplitude) eines RC-Tiefpasses. Der Funktionsgenerator wird im SWEEP-Modus betrieben, das Oszilloskop in Spitzenwert-Darstellung.

Das MATLAB-Skript für die Messung programmiert zunächst den Funktionsgenerator

```
func = 'USB0::0x0957::0x0407::MY43003008::INSTR';
visaio(func, 'APPL:SIN 1000,0.2');
visaio(func, 'FREQ:START 1000');
visaio(func, 'FREQ:STOP 1.0E+7');
visaio(func, 'SWE:SPAC LOG');
visaio(func, 'SWE:TIME 1');
visaio(func, 'SWE:STAT ON');
```

Es wird eine Sinusspannung von 0.2 V eingestellt, dann Start- und Stop-Frequenz (1 kHz und 10 MHz) für einen Frequenz-Sweep, der auf logarithmischer Skala in 1 s den gewünschten Frequenzbereich durchlaufen soll.

Die Einstellungen am Oszilloskop werden manuell vorgenommen (auch das könnte über eine geeignete Befehlsabfolge automatisiert werden). So müssen von dort nur die Daten eingelesen und interpretiert werden. Zum Einlesen wird Kanal 1 angewählt und binäres Datenformat eingestellt

```
osc ='USB0::0x0699::0x0367::C030104::INSTR';
visaio(osc, 'SEL:CH1 ON');
visaio(osc, 'DAT:SOU CH1');
visaio(osc, 'DAT:ENC RPB');
d = visaio(osc, 'CURVE?');
```

Zur Interpretation der Daten wird zunächst auf das Startsymbol (#) gewartet, dann die folgenden Bytes ihrer Bedeutung entsprechend dekodiert, schließlich wird das eigentliche Datenfeld in *Double*-Format gewandelt.

```
for k=1:length(d)
    if (char(d(k))=='#') break; end
end
L = str2double(char(d(k+1)));
N = str2double(char(d(k+1+[1:L])));
D = double(d(k+1+L+[1:N]));
```

Ein Plot dieser Daten mit

```
plot(D,'ko','Markersize',1);
set(gca,'XLim',[0,2500],'YLim',[20,236],...
  'XTick',500*[0,1,2,3,4,5],'XTicklabel',...
  {'0','0.2','0.4','0.6','0.8','1'});
```

ist in Abbildung 95 gezeigt. Es sind die vom Oszilloskop dargestellten Spitzenwerte. Eine genauere Betrachtung der Daten zeigt, dass im Datenfeld Maximal- und Minimalwerte abwechseln. Man kann den Frequenzgang mithin sehr einfach mit

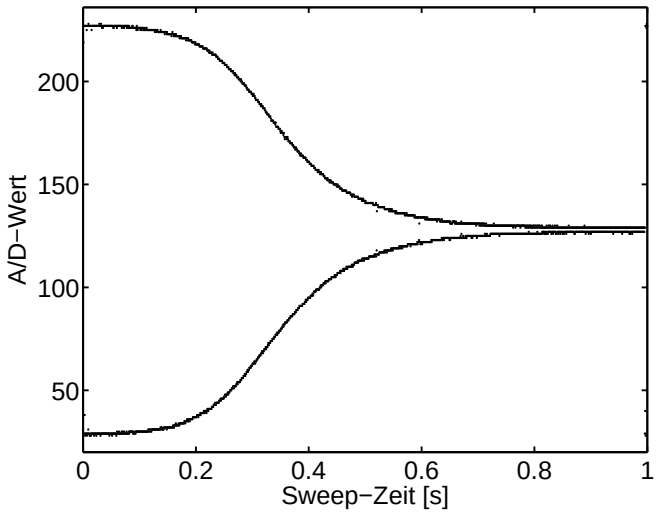


Abbildung 95: Frequenzsweep am Tiefpass: Rohdaten vom Oszilloskop bei Spitzenwertanzeige.

```
Y = D(2:2:N)-D(1:2:N);
Y = log10(Y/max(Y));
```

produzieren. Plotten mit nachstehenden Anweisungen liefert dann die gewohnte doppelt-logarithmische Darstellung der Frequenzgangs (Abbildung 96).

```
plot(Y,'k-','Linewidth',1.5);
set(gca,'XLim',[0,1050],'XTick',1250/4*[0,1,2,3,4],...
  'XTickLabel',{'1kHz','10kHz','100kHz','1MHz','10MHz'},...
  'YLim',[-2.05,0.05]);
```

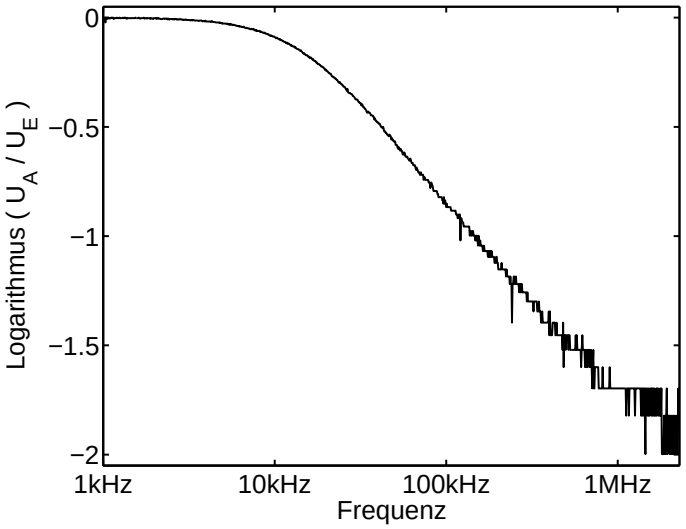


Abbildung 96: Frequenzsweep am Tiefpass: Doppelt-logarithmische Auftragung der Differenz aus benachbarten Maxima und Minima der Spitzenwertanzeige.

Apropos: **R** war 10 kΩ, welchen Wert hatte **C**?

7.7 LAN-Schnittstelle, TCP/IP-Programmierung

Neben der ständig wachsenden Zahl der rein windowsorientierten Kommunikationsmöglichkeiten (DDE, NetDDE, OLE1, OLE2, COM, DCOM, ActiveX, ...), steht – wenn TCP/IP als Protokoll installiert ist – unter den Windows-Systemen auch die weitgehend systemunabhängige Socket-Schnittstelle zum Informationsaustausch zwischen Prozessen auf einem oder auf verschiedenen vernetzten Rechnern (Internet) zur Verfügung. Über diese Schnittstelle ist auch eine Verbindung mit Rechnern unter anderen Betriebssystemen einfach realisierbar. Darüber hinaus sind seit einiger Zeit verschiedene Microcontroller auf dem Markt, die eine Ethernet-Schnittstelle und die zugehörige TCP/IP-Software enthalten und damit den Bau einfacher und billiger Messsysteme mit Internet-Anschluss ermöglichen.

7.7.1 Sockets und Ports

Ein Socket ist der Endpunkt einer Netzwerkverbindung unter dem TCP/IP-Protokoll; diese Schnittstelle kann überall dort benutzt werden, wo TCP/IP als Netzwerkprotokoll installiert ist. Die Kommunikation über Sockets kann sowohl zwischen Programmen innerhalb eines Rechners wie auch – das ist der üblichere Fall – zwischen unterschiedlichen Rechnern ablaufen.

Der Verbindungsaufbau erfolgt (bei TCP-Verbindungen, auf die wir uns hier beschränken wollen) nach einem Client-Server-Prinzip: ein Server-Socket wartet am Netzwerk (`Listen`), ein Client-Socket betreibt den Verbindungsaufbau (`Connect`), der Server-Socket nimmt die Verbindung an (`Accept`). Über die Verbindung werden Daten mit `Send` und `Receive` ausgetauscht, die Verbindung wird schließlich von einem der Partner mit `Close` beendet.

Die Art der Verwendung bzw. das Subprotokoll der Verbindung wird durch die Port-Nummer des Server-Sockets spezifiziert. Dadurch können unterschiedliche Verbindungstypen (TELNET, FTP, WWW, ...) quasigleichzeitig auf eine physikalische Netzschnittstelle zugreifen – soweit dafür ein Server-Programm vorgehalten wird. So hat beispielsweise ein TELNET-Server die Port-Nummer 23, ein SMTP-Server 25, ein WWW-Server 80. Der Client gibt beim Connect-Versuch die gewünschte Port-Nummer an, um mit dem zuständigen Server verbunden zu werden. Die üblichen Client-Programme machen dies ohne Zutun richtig – ein TELNET-Programm versucht einen Verbindungsaufbau mit Port 23 des Host-Rechners. Zu Testzwecken kann die Verbindung auch zu einer anderen Port-Nummer hergestellt werden (z. B. TELNET-Client $\hookrightarrow$ SMTP-Server, um das Protokoll zu studieren).

Generell ist vorgesehen, dass von einem Server mehrere Verbindungen vom gleichen Typ gemanagt werden können, dazu wird der Listener-Socket bei der Verbindungsannahme jeweils dupliziert und die Datenübertragung wird auf dem Duplikat abgewickelt.

Die Liste der festgelegten Port-Nummern ('well-known ports') ist in RFC1700 [42] veröffentlicht ('Request for Comments is a series of documents published by the Internet Engineering Task Force [43] and cover a broad range of topics. The core topics are the Internet and the TCP/IP protocol suite.'). Die vom Rechnerbetriebssystem belegten Ports bzw. Dienste finden sich in der Datei `services` (bei UNIX-Systemen im Verzeichnis `/etc`, bei Windows-NT- oder -2000-Systemen im Verzeichnis `...\System32\drivers\etc`, bei Windows-95/98/ME-Systemen im Windows-Hauptverzeichnis). Bei der Programmierung von Socket-Verbindungen müssen Kollisionen von Port-Nummern mit 'well-knowns' vermieden werden (> 1024 im allgemeinen problemlos).

7.7.2 Socket-Server in C/C++

In C/C++ unter Linux oder Windows ist eine Socket-Verbindung meist mit relativ einfachen Anweisungen realisierbar, die zuständigen Funktionen ähneln den üblichen Datei-Ein/Ausgabe-Funktionen.

Bei einem Server-Socket wird die Verbindung mit `socket` und `bind` bereitgestellt, `listen` wartet dann auf ankommende Verbindungen, die mit `accept` angenommen werden. Dabei wird ein Duplikat des Sockets erstellt, mit dem die Datenübertragung durch die Funktionen `recv` und `send` abgewickelt wird. Der ursprüngliche Socket bleibt empfangsbereit, kann weitere Verbindungen abweisen, in eine Warteschlange stellen oder über weitere Duplikate sofort annehmen. Nach dem Ende der Verbindung wird das Socketduplikat mit `closesocket` geschlossen, der Server-Socket geht wieder in den normalen *Listen*-Zustand zurück.

Im folgenden Beispiel ist ein einfacher Echo-Server programmiert, ein Programm, das an einem bestimmten Port auf eine Verbindungsanforderung wartet und die ankommenden Daten reflektiert.

Port und Rechnername werden als Konstanten fest eingestellt:

```
const unsigned short PORT = 1234;
const char HOSTNAME[] = "localhost"; .
```

Unter Windows ist es zuallererst notwendig, das Socket-System zu initialisieren, das macht die Funktion `WSAStartup`:

```
WORD wVersionRequested = MAKEWORD( 1, 0 );
WSADATA wsaData;
WSAStartup( wVersionRequested, &wsaData ); .
```

Die Informationen über die Verbindungsart, Portnummer und IP-Adresse werden in einer Struktur von Typ `sockaddr_in` abgelegt, die Funktion `gethostbyname` übersetzt den Rechnernamen in eine IP-Adresse (lokal oder durch Anfrage bei einem Nameserver):

```
struct sockaddr_in ad, adcli;
int adlen = sizeof(sockaddr_in);
ad.sin_family = PF_INET;
```

```
ad.sin_port = htons(PORT);
struct hostent *he = gethostbyname(HOSTNAME);
if (he==0) {
  printf("Error: gethostbyname.\n");
return -1;
}
memcpy(&ad.sin_addr, he->h_addr, 4); .
```

Diese Struktur wird bei der Erstellung des Sockets in der Funktion `bind` als Parameter benötigt:

```
SOCKET S = socket(AF_INET, SOCK_STREAM, 0);
WSASetLastError(0);
if (bind(S, (sockaddr *) &ad, sizeof(ad))!=0) {
  printf("Bind Error: %d\n", WSAGetLastError());
  return -1;
  } .
```

In der folgenden Endlosschleife wird auf die Verbindungsanforderung durch einen Client-Socket gewartet, `listen` stoppt das Programm solange. `accept` nimmt die Verbindung auf dem Duplikat `T` an, in der Struktur `adcli` werden die Client-Daten abgelegt, man kann feststellen, mit wem man verbunden ist. Die zweite Schleife empfängt Daten, gibt sie auf dem Bildschirm aus und schickt sie wieder zurück, bis der Client die Verbindung beendet (`recv` liefert ein Ergebnis $<= 0$):

```
for (;;) {
  listen(S, 5);
  SOCKET T = accept(S, (sockaddr *) &adcli, &adlen);
  for (;;) {
    const int BUFSIZE = 1024;
    char buffer[BUFSIZE+1];
    int n = recv(T, buffer, BUFSIZE, 0);
    if (n<=0)
      break;
    buffer[n] = '\0';
    printf(buffer);
    send(T, buffer, strlen(buffer), 0);
  }
  closesocket(T);
}
return closesocket(S);  // never reached .
```

Nach der zweiten Schleife wird das Socket-Duplikat `T` mit `closesocket` geschlossen. Die äußere Endlosschleife kann nie verlassen werden, die letzte Zeile ist mithin nur ein Tribut an guten Programmierstil.

In einem Messsystem würde man statt der `printf`-Zeile eine geeignete Aktion implementieren, d. h. die Nachricht im `buffer` interpretieren, darauf reagieren, die Antwort im `buffer` ablegen.

Nach minimalen Änderungen ist das Programm auch unter Linux lauffähig:

- Alle Funktionen, die mit `WSA` beginnen, sind Windows-spezifisch, fallen unter Linux weg.

- Die Casts von `sockaddr_in` nach `sockaddr` sind unter Linux unnötig.
- Statt `closesocket` genügt unter Linux ein `close`.

7.7.3 Socket-Client in C/C++

Ein zu dem Socket-Server-Programm komplementäres Client-Programm, das Konsol-Eingaben an den Server schickt und die Antworten ausgibt, lässt sich durch kleine Änderungen realisieren.

Nach dem ungeänderten ersten Teil wird die Verbindung vom Client mit `connect` aufgebaut:

```
SOCKET S = socket(AF_INET, SOCK_STREAM, 0);
WSASetLastError(0);
if (connect(S, (sockaddr *) &ad, sizeof(ad))!=0) {
  printf("Connect Error: %d\n", WSAGetLastError());
  return -1;
  } .
```

Dann die Endlosschleife mit Tastatureingabe, Verschicken, Empfangen, Ausgabe:

```
for (;;) {
  const int BUFSIZE = 1024;
  char buffer[BUFSIZE+1];
  fgets(buffer, BUFSIZE, stdin);
  if ((buffer[0]=='e')||(buffer[0]=='E'))
    break;
  send(S, buffer, strlen(buffer), 0);
  int n = recv(S, buffer, BUFSIZE, 0);
  buffer[n] = '\0';
  printf(buffer);
}
return closesocket(S); .
```

7.7.4 Socket-Client in MATLAB/Java

Durch die Java-Integration ist es sehr einfach geworden, die TCP/IP-Kommunikation direkt in MATLAB zu formulieren. Das zuständige *Package* `java.net` wird importiert und es wird ein `Socket`-Objekt mit den benötigten Eigenschaften (Zielrechner, Port) formuliert:

```
import java.net.*
so = java.net.Socket('131.173.11.215', 1234);
...
V(i) = measure(so, C(i));
...
so.close; .
```

Die Messroutine `measure` übermittelt einen Vorgabewert `C(i)` an das Messsystem und liest einen Messwert `V(i)`:

```
function volt = measure(so, curr)
  sData = sprintf('da4 %d', curr);
  SocketIO(so, sData);
  rData = SocketIO(so, 'ad4');
  volt = str2double(rData); .
```

Die eigentliche Kommunikation wird von der Funktion `SocketIO` erledigt:

```
function recvStr = SocketIO(socket, sendStr)
  str = java.lang.String(sendStr);
  out = socket.getOutputStream;
  out.write(str.getBytes);
  in = socket.getInputStream;
  isr = java.io.InputStreamReader(in);
  ibr = java.io.BufferedReader(isr);
  recvStr = ibr.readLine; .
```

Der MATLAB-String wird in einen Java-String gewandelt, der `OutputStream` des Sockets wird etabliert, der Java-String wird verschickt. Auf den `InputStream` des Sockets wird ein `BufferedReader` aufgesetzt, mit dem eine ankommende Zeile gelesen wird.

7.7.5 LXI – ein neuer Standard

Da LAN-Schnittstellen an allen Computersystemen vorhanden sind, dadurch auch die Bausteine dafür relativ preisgünstig sind, beginnen viele Herstellerfirmen damit, ihre Messgeräte mit LAN-Verbindungen auszustatten (vgl. Abbildung 97). Es ist abzusehen, dass dieser Verbindungstyp aus Kostengründen den IEC-Bus (GPIB) ablösen wird.

Abbildung 97: Rückseite eines Messgeräts (Agilent Funktionsgenerator 33220A): Noch sind drei Standards, USB, LAN und GPIB, als Anschlussmöglichkeiten vorgesehen.

Dazu ist noch eine detailliertere Standardisierung notwendig, an der zurzeit intensiv gearbeitet wird. Eine Arbeitsgruppe begann 2004 mit der Standardisierung, 2006 erschienen die ersten nach einem vorläufigen Standard arbeitenden Geräte auf dem Markt.

Übliche LAN-Verbindungen sind meist nicht zeitkritisch, daher sind in der Regel keine harten Synchronisationsmechanismen vorgesehen. Verbindungen und

Netze sind auf weitgehend gleichberechtigten Vielfachzugriff ausgelegt, es sind daher Mechanismen eingebaut, die bei Kollisionen dafür sorgen, dass Daten nicht verloren gehen. Das geht üblicherweise nicht ohne Verzögerungen. Bei Messgeräten ist dagegen sehr oft eine exakte Zeitsteuerung notwendig, beim IEC-Bus beispielsweise kann dies durch den sehr direkten exklusiven Zugriff des Controllers auf den Bus realisiert werden.

Der neu entwickelte Standard *LXI - Lan eXtension for Instrumentation* [44] definiert derzeit drei bzw. vier Geräteklassen, die sich im Wesentlichen durch unterschiedliche Mechanismen zur Zeitsynchronisation unterscheiden.

Class C erfordert keine speziellen Synchronisationsmechanismen, ist mithin vergleichbar mit bekannten Netzwerk-Verbindungen.

Class B ermöglicht eine exakte Zeitsynchronisation durch Implementierung des IEEE 1588 Standards [45] für *A Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. Je nach Aufwand sind damit Genauigkeiten im Mikrosekunden- bis Nanosekundenbereich möglich.

Class A sieht darüber hinaus Hardware-Trigger-Möglichkeiten vor, die durch zusätzliche Leitungen in den Verbindungskabeln realisiert werden müssen (*Wired Trigger Bus*, *WTB*). Die Genauigkeit ist dann nur noch durch den Hardware-Aufwand begrenzt.

Um nicht IEEE 1588 implementieren zu müssen und dennoch gute Synchronisationsmöglichkeiten zu bieten, haben verschiedene Hersteller eine **Class C Plus** definiert, die die Class C um die Hardware-Trigger-Möglichkeiten der Class A erweitert.

Alle LXI-Geräte müssen ein Web-Interface bereitstellen, über das die Geräte in ihren wesentlichen Funktionen bedient werden können und über das sie gegebenenfalls auch ihre Daten darstellen (beispielsweise bei Oszilloskopen). Der LXI-Standard erfordert weiterhin, dass eine automatische Geräteerkennung möglich ist (*Broadcast*-Abfrage im Netz). Die Herstellerfirmen müssen darüber hinaus IVI-kompatible Treiber liefern (vgl. Abschnitt 7.6.6), so dass die Geräte in C- oder Visual-Basic-Messprogramme eingebunden werden können.

8 Windows-Programmierung mit Visual Studio

Einfache Messprogramme, Praktikumsanwendungen, Prototypprogramme, leicht veränderbare Programme oder solche mit nur kurzer Lebensdauer werden zweckmäßigerweise mit einer graphischen Messumgebung wie LabView bzw. VEE oder – wie in Kapitel 5 beschrieben – mit einem Numerikprogramm wie MATLAB programmiert. Für Aufgaben, die deutlich darüber hinaus gehen, beispielsweise die Erstellung von professionellen Messprogrammen für feste apparative Aufbauten, kann es sinnvoll sein, Windows-Programme komplett in einer leistungsfähigen Programmierumgebung zu erstellen.

Auf eine Einführung in die Windows-Programmierung wird im diesjährigen Skriptum aus Platzgründen verzichtet, bei Bedarf können Sie auf die Kurzeinführung in den Vorgängerskripten zurückgreifen (2006 und früher).

Wenn man Windows-Programme schreiben will, ist es auf jeden Fall empfehlenswert, sich mit Microsoft Visual Studio[42] als Programmierumgebung auseinanderzusetzen. Man hat dort die Wahl zwischen verschiedenen Programmiersprachen (C, C++, C#, Visual Basic, J#) und wohl das umfangreichste On-Line-Hilfesystem im Bereich der Windows-Programmierung.

> Man mag über die Firma denken wie man will;
> es gibt niemanden, der Windows besser kennt.

[42] Microsoft Visual Studio ist im Rahmen der MSDN-AA (Microsoft Developer Network – Academic Alliance) für Hochschulangehörige frei erhältlich.

Literatur

Die Literaturhinweise sind – insbesondere die zitierten Lehrbücher betreffend – exemplarisch, willkürlich und zufällig, eine Vollständigkeit ist weder möglich noch angestrebt.

[1] H. Ibach, H. Lüth. *Festkörperphysik*. Springer, 1990.

[2] D. Geist. *Halbleiterphysik I: Eigenschaften homogener Halbleiter*. Vieweg, 1969.

[3] F. S. Goucher, G. L. Pearson, M. Spark, G. K. Teal, W. Shockley. *Theory and Experiment for a Germanium p-n Junction*. Phys. Rev. **81**, 637 (1951).

[4] W. Heywang. *Sensorik*. Springer, 1986.

[5] Analog Devices. *AD 592*. Datenblatt.

[6] G. Winstel, C.Weyrich. *Optoelektronik II – Photodioden, Phototransistoren, Photoleiter und Bildsensoren*. Springer, 1986.

[7] Burle Electronics. *Photomultiplier Handbook*. Burle, 1987.

[8] Burle Electronics. *Electro-Optics Handbook*. Burle, 1987.

[9] D. Geist. *Halbleiterphysik II: Sperrschichten und Randschichten – Bauelemente*. Vieweg, 1970.

[10] Hamamatsu Photonics. *Opto-Semiconductors, Condensed Catalog*. Datenblatt.

[11] Centronic Ltd. *High Performance Silicon Photodetectors*. Datenblatt.

[12] Ortel Vertriebs GmbH. *InGaAsP / InP-IR-LED's, InGaAs-Fotodioden (Telcom Devices Corp.)*. Datenblatt.

[13] M. J. E. Golay. *A pneumatic infrared detector*. Rev. Sci. Instr. **18**, 357 (1947).

[14] U. Tietze, C. Schenk. *Halbleiterschaltungstechnik*. Springer, 1991.

[15] Hermann Hinsch. *Elektronik: Ein Werkzeug für Naturwissenschaftler*. Springer, 1996.

[16] http://de.wikipedia.org/wiki/Operationsverst%C3%A4rker.

[17] http://www.ni.com.

[18] http://www.agilent.com.

[19] http://www.mathworks.com.

[20] http://www.inria.fr.

[21] The MathWorks. *MATLAB 7 External Interfaces*. Handbuch, 2007.

[22] http://www.bloodshed.net.

[23] Bei einer Standard-Installation:
<matlabroot>/help/pdf_doc/matlab/api/apiguide.pdf.

[24] Bei einer MATLAB-Standard-Installation findet sich das Inhaltsverzeichnis der API-Referenz in <matlabroot>/help/techdoc/apiref/apireftoc.html.

[25] Bei einer Standard-Installation:
<matlabroot>/help/pdf_doc/matlab/api/apiref.pdf.

[26] http://www.physik.uni-osnabrueck.de/kbetzler/gw/gw.html.

[27] Bässmann, Besslich. *Konturorientierte Verfahren in der digitalen Bildverarbeitung*. Springer Verlag, 1989.

[28] A. Savitzky, M. J. E. Golay. *Smoothing and Differentiation of Data by Simplified Least Squares Procedures*. Analytical Chemistry **36**, 1627–1639 (1964).

[29] W. Gander, J. Hřebíček. *Solving Problems in Scientific Computing using Maple and MATLAB*. Springer Verlag, 1995.

[30] Klaus Betzler. *Experimentsteuerung*. Vorlesungsskriptum, 1990.
http://www.physik.uni-osnabrueck.de/kbetzler/win32/es.pdf.

[31] http://de.wikipedia.org/wiki/Isolated_I/O.

[32] http://de.wikipedia.org/wiki/Memory_Mapped_I/O.

[33] http://en.wikipedia.org/wiki/Memory-mapped_IO.

[34] http://www.bbdsoft.com/.

[35] Georg Walz. *Grundlagen und Anwendung des IEC-Bus*. Markt & Technik, 1982.

[36] Lothar Preuss, Harald Musa. *Computerschnittstellen : Dokumentation der Hard- und Software mit Anwendungsbeispielen CENTRONICS, IEC-BUS, V.24*. Hanser, 1989.

[37] Agilent. *Agilent IO Libraries Suite 14 SICL User's Guide*. Handbuch, 2005.

[38] Agilent. *Agilent IO Libraries Suite 14 USB/LAN/GPIB Interfaces Connectivity Guide*. Handbuch, 2006.

[39] http://www.usb.org.

[40] http://www.physik.uni-osnabrueck.de/kbetzler/win32/usb/usbspec.pdf.

[41] http://www.ivifoundation.org/.

[42] http://www.ietf.org/rfc/rfc1700.txt.

[43] http://www.ietf.org/.

[44] http://www.lxistandard.org.

[45] http://ieee1588.nist.gov/.

A Beispielprogramm: PerformanceCounter und Multimedia-Timer

```
// usbservo.cpp  2007 (c) Klaus Betzler
// Steuerung eines Servos mit Tastatureingaben

#include <conio.h>
#include "windows.h"
#include <NIDAQmx.h>

static UINT uTimer = 0;
const short DIVISOR = 20;
static short ACTUAL = 30;         // 1.5 ms (ACTUAL/DIVISOR)
const short MIN = 18, MAX = 42;   // 0.9 to 2.1 ms

class NI6008DigOut {
private:
  TaskHandle taskHandle;
public:
  NI6008DigOut(char DEV[]="Dev1/port0/line0:7") {
    taskHandle = 0;
    DAQmxCreateTask("",&taskHandle);
    DAQmxCreateDOChan(taskHandle,DEV,"",DAQmx_Val_ChanForAllLines);
    DAQmxStartTask(taskHandle);
  }
  ~NI6008DigOut() {
    if( taskHandle!=0 ) {
      DAQmxStopTask(taskHandle);
      DAQmxClearTask(taskHandle);
    }
  }
  void Pattern(short data) {
    DAQmxWriteDigitalScalarU32(taskHandle,1,10.0,data,NULL);
  }
} DigOut;

void CALLBACK Pulse ( UINT ID, UINT, DWORD, DWORD, DWORD )
{
  LARGE_INTEGER f, t1, t2;
  QueryPerformanceFrequency(&f);
  LONGLONG INC = f.QuadPart/1000*ACTUAL/DIVISOR;
  QueryPerformanceCounter(&t1);
  t1.QuadPart += INC;
  DigOut.Pattern(1);
  do {
    QueryPerformanceCounter(&t2);
    } while (t2.QuadPart<t1.QuadPart);
  DigOut.Pattern(0);
}

void main()
{
  char c;
  timeBeginPeriod(20);
  uTimer = timeSetEvent ( 20, 20, Pulse, 0, TIME_PERIODIC );
```

```
    while ((c=_getch())!='e')
      switch (c) {
        case 'a': ACTUAL = MIN;
          break;
        case 's': ACTUAL = (ACTUAL<=MIN) ? MIN : (ACTUAL-1);
          break;
        case 'd': ACTUAL = (ACTUAL>=MAX) ? MAX : (ACTUAL+1);
          break;
        case 'f': ACTUAL = MAX;
          break;
        default: /* DO NOTHING */ ;
      }
    timeKillEvent ( uTimer );
    timeEndPeriod(20);
    Sleep (100);
  }
```

B MEX-Funktion: Schreiben und Lesen auf ein SICL-Gerät

```
// siclio.cpp (c) 20071117 KB
// Write and/or Read SICL Instrument

#include "mex.h"
#include "sicl.h"
#include <string.h>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
     const mxArray *prhs[])
{
  /* Check for proper number of arguments. */
  if ((nrhs < 1)||((nrhs+nlhs)<2)||(nlhs>1))
    mexErrMsgTxt("Invalid number of arguments:\n"
      "Two outputs (SICL name and data) for writing,\n"
      "one output (SICL name) and one input for reading,\n"
      "two outputs and one input for writing and reading.");

  /* Inputs 1 and 2 must be strings. */
  if (mxIsChar(prhs[0]) != 1)
    mexErrMsgTxt("Input 1 must be a string.");
  if (mxGetM(prhs[0]) != 1)
    mexErrMsgTxt("Input 1 must be a row vector.");
  if ((nrhs > 1) && (mxIsChar(prhs[1]) != 1))
    mexErrMsgTxt("Input 2 must be a string.");
  if ((nrhs > 1) && (mxGetM(prhs[1]) != 1))
    mexErrMsgTxt("Input 2 must be a row vector.");

  /* Open instrument connection with SICL name string */
  char* devName = mxArrayToString(prhs[0]);
  INST id = iopen(devName);
  mxFree(devName);

  /* Write data string */
  if (nrhs > 1) {
    unsigned char* data = mxArrayToString(prhs[1]);
```

```
      iwrite(id, data, (unsigned long) strlen(data), 1, 0);
      mxFree(data);
    }

    /* Read string or uint8 data from instrument */
    if (nlhs == 1) {
      long retCount;
      const unsigned long BUFFERSIZE = 10000;
      unsigned char data[BUFFERSIZE];
      iread(id, data, BUFFERSIZE, 0, &retCount);
      plhs[0] = mxCreateNumericMatrix(1, retCount, mxUINT8_CLASS, mxREAL);
      memcpy((unsigned char) mxGetData(plhs[0]), data, retCount);
    }

    /* Close connection. */
    iclose(id);
  }
```

C Beispielprogramm: Analog-Ausgabe mit NI USB-6008

```
/*********************************************************************
*
* ANSI C Example program:
*    VoltUpdate.c
*
* Example Category:
*    AO
*
* Description:
*    This example demonstrates how to output a single Voltage Update
*    (Sample) to an Analog Output Channel.
*
* Instructions for Running:
*    1. Select the Physical Channel to correspond to where your
*       signal is output on the DAQ device.
*    2. Enter the Minimum and Maximum Voltage Ranges.
*    Note: Use the Acq One Sample example to verify you are
*          generating the correct output on the DAQ device.
*
* Steps:
*    1. Create a task.
*    2. Create an Analog Output Voltage Channel.
*    3. Use the Write function to Output 1 Sample to 1 Channel on the
*       Data Acquisition Card.
*    4. Display an error if any.
*
* I/O Connections Overview:
*    Make sure your signal output terminal matches the Physical
*    Channel I/O Control. For further connection information, refer
*    to your hardware reference manual.
*
*********************************************************************/

#include <stdio.h>
```

```
#include <NIDAQmx.h>

#define DAQmxErrChk(functionCall) if( DAQmxFailed(error=(functionCall)) )\
  goto Error; else

int main(void)
{
  int         error=0;
  TaskHandle  taskHandle=0;
  char        errBuff[2048]={'\0'};
  float64     data[1] = {0.0};

  /*********************************************/
  // DAQmx Configure Code
  /*********************************************/
  DAQmxErrChk (DAQmxCreateTask("",&taskHandle));
  DAQmxErrChk (DAQmxCreateAOVoltageChan(taskHandle,"Dev1/ao0","",0.0,5.0,
    DAQmx_Val_Volts,""));

  /*********************************************/
  // DAQmx Start Code
  /*********************************************/
  DAQmxErrChk (DAQmxStartTask(taskHandle));

  /*********************************************/
  // DAQmx Write Code
  /*********************************************/
  DAQmxErrChk (DAQmxWriteAnalogF64(taskHandle,1,1,10.0,
    DAQmx_Val_GroupByChannel,data,NULL,NULL));

Error:
  if( DAQmxFailed(error) )
    DAQmxGetExtendedErrorInfo(errBuff,2048);
  if( taskHandle!=0 ) {
    /*********************************************/
    // DAQmx Stop Code
    /*********************************************/
    DAQmxStopTask(taskHandle);
    DAQmxClearTask(taskHandle);
  }
  if( DAQmxFailed(error) )
    printf("DAQmx Error: %s\n",errBuff);
  printf("End of program, press Enter key to quit\n");
  getchar();
  return 0;
}
```

D Beispielprogramm: Analog-Eingabe mit NI USB-6008

```
/*********************************************************************
*
* ANSI C Example program:
*    Acq-IntClk.c
*
```

```
 * Example Category:
 *    AI
 *
 * Description:
 *    This example demonstrates how to acquire a finite amount of data
 *    using the DAQ device's internal clock.
 *
 * Instructions for Running:
 *    1. Select the physical channel to correspond to where your
 *       signal is input on the DAQ device.
 *    2. Enter the minimum and maximum voltages.
 *    Note: For better accuracy try to match the input range to the
 *          expected voltage level of the measured signal.
 *    3. Select the number of samples to acquire.
 *    4. Set the rate of the acquisition.
 *    Note: The rate should be AT LEAST twice as fast as the maximum
 *          frequency component of the signal being acquired.
 *
 * Steps:
 *    1. Create a task.
 *    2. Create an analog input voltage channel.
 *    3. Set the rate for the sample clock. Additionally, define the
 *       sample mode to be finite and set the number of samples to be
 *       acquired per channel.
 *    4. Call the Start function to start the acquisition.
 *    5. Read all of the waveform data.
 *    6. Call the Clear Task function to clear the task.
 *    7. Display an error if any.
 *
 * I/O Connections Overview:
 *    Make sure your signal input terminal matches the Physical
 *    Channel I/O Control. For further connection information, refer
 *    to your hardware reference manual.
 *
 ********************************************************************/

#include <stdio.h>
#include <NIDAQmx.h>

#define DAQmxErrChk(functionCall) if( DAQmxFailed(error=(functionCall)) ) \
  goto Error; else

int main(void)
{
  int32       error=0;
  TaskHandle  taskHandle=0;
  int32       read;
  float64     data[1000];
  char        errBuff[2048]={'\0'};

  /*********************************************/
  // DAQmx Configure Code
  /*********************************************/
  DAQmxErrChk (DAQmxCreateTask("",&taskHandle));
  DAQmxErrChk (DAQmxCreateAIVoltageChan(taskHandle,"Dev1/ai0","",
    DAQmx_Val_Cfg_Default,-10.0,10.0,DAQmx_Val_Volts,NULL));
```

```
    DAQmxErrChk (DAQmxCfgSampClkTiming(taskHandle,"",10000.0,DAQmx_Val_Rising,
      DAQmx_Val_FiniteSamps,1000));

    /*********************************************/
    // DAQmx Start Code
    /*********************************************/
    DAQmxErrChk (DAQmxStartTask(taskHandle));

    /*********************************************/
    // DAQmx Read Code
    /*********************************************/
    DAQmxErrChk (DAQmxReadAnalogF64(taskHandle,1000,10.0,
      DAQmx_Val_GroupByChannel,data,1000,&read,NULL));

    printf("Acquired %d points\n",read);

  Error:
    if( DAQmxFailed(error) )
      DAQmxGetExtendedErrorInfo(errBuff,2048);
    if( taskHandle!=0 )  {
      /*********************************************/
      // DAQmx Stop Code
      /*********************************************/
      DAQmxStopTask(taskHandle);
      DAQmxClearTask(taskHandle);
    }
    if( DAQmxFailed(error) )
      printf("DAQmx Error: %s\n",errBuff);
    printf("End of program, press Enter key to quit\n");
    getchar();
    return 0;
  }
```

E Beispielprogramm: VISA-Geräte suchen

```
/*****************************************************************/
/* This example demonstrates how you might query your system for     */
/* a particular instrument.  This example queries for all            */
/* GPIB, serial or VXI instruments.  Notice that VISA is able to     */
/* find GPIB and VXI instruments because the instruments have a      */
/* predefined protocol.  But serial instruments do not.  Hence,      */
/* VISA merely indicates that a serial port is available.            */
/*                                                                   */
/* The general flow of the code is                                   */
/*      Open Resource Manager                                        */
/*      Use viFindRsrc() to query for the first available instrument */
/*      Open a session to this device                                */
/*      Find the next instrument using viFindNext()                  */
/*      Open a session to this device.                               */
/*      Loop on finding the next instrument until all have been found*/
/*      Close all VISA Sessions                                      */
/*****************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>

#include "visa.h"

static char instrDescriptor[VI_FIND_BUFLEN];
static ViUInt32 numInstrs;
static ViFindList findList;
static ViSession defaultRM, instr;
static ViStatus status;

int main(void)
{
   /* First we will need to open the default resource manager. */
   status = viOpenDefaultRM (&defaultRM);
   if (status < VI_SUCCESS)
   {
      printf("Could not open a session to the VISA Resource Manager!\n");
      exit (EXIT_FAILURE);
   }

    /*
     * Find all the VISA resources in our system and store the number of resources
     * in the system in numInstrs.  Notice the different query descriptions a
     * that are available.

         Interface         Expression
    --------------------------------------
         GPIB              "GPIB[0-9]*::?*INSTR"
         VXI               "VXI?*INSTR"
         GPIB-VXI          "GPIB-VXI?*INSTR"
         Any VXI           "?*VXI[0-9]*::?*INSTR"
         Serial            "ASRL[0-9]*::?*INSTR"
         PXI               "PXI?*INSTR"
         All instruments   "?*INSTR"
         All resources     "?*"
    */
   status = viFindRsrc (defaultRM, "?*INSTR", &findList, &numInstrs, instrDescriptor);
   if (status < VI_SUCCESS)
   {
      printf ("An error occurred while finding resources.\nHit enter to continue.");
      fflush(stdin);
      getchar();
      viClose (defaultRM);
      return status;
   }

   printf("%d instruments, serial ports, and other resources found:\n\n",numInstrs);
   printf("%s \n",instrDescriptor);

   /* Now we will open a session to the instrument we just found. */
   status = viOpen (defaultRM, instrDescriptor, VI_NULL, VI_NULL, &instr);
   if (status < VI_SUCCESS)
   {
      printf ("An error occurred opening a session to %s\n",instrDescriptor);
   }
```

```
      else
      {
        /* Now close the session we just opened.                          */
        /* In actuality, we would probably use an attribute to determine  */
80      /* if this is the instrument we are looking for.                  */
        viClose (instr);
      }

      while (--numInstrs)
85    {
         /* stay in this loop until we find all instruments */
         status = viFindNext (findList, instrDescriptor);  /* find next desriptor */
         if (status < VI_SUCCESS)
         {   /* did we find the next resource? */
90          printf ("An error occurred finding the next resource.\nHit enter to continue.");
            fflush(stdin);
            getchar();
            viClose (defaultRM);
            return status;
95       }
         printf("%s \n",instrDescriptor);

         /* Now we will open a session to the instrument we just found */
         status = viOpen (defaultRM, instrDescriptor, VI_NULL, VI_NULL, &instr);
100      if (status < VI_SUCCESS)
         {
             printf ("An error occurred opening a session to %s\n",instrDescriptor);
         }
         else
105      {
            /* Now close the session we just opened.                       */
            /* In actuality, we would probably use an attribute to determine */
            /* if this is the instrument we are looking for.               */
            viClose (instr);
110      }
      }    /* end while */

      status = viClose(findList);
      status = viClose (defaultRM);
115   printf ("\nHit enter to continue.");
      fflush(stdin);
      getchar();

      return 0;
120 }
```

F Beispielprogramm: Schreiben und Lesen auf ein VISA-Gerät

```
1  /*****************************************************************/
   /*              Read and Write to an Instrument Example          */
   /*                                                               */
   /* This code demonstrates synchronous read and write commands to a  */
5  /* GPIB, serial or message-based VXI instrument using VISA.      */
   /*                                                               */
```

```
/* The general flow of the code is                                  */
/*      Open Resource Manager                                       */
/*      Open VISA Session to an Instrument                          */
/*      Write the Identification Query Using viWrite                */
/*      Try to Read a Response With viRead                          */
/*      Close the VISA Session                                      */
/*******************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "visa.h"

static ViSession defaultRM;
static ViSession instr;
static ViStatus status;
static ViUInt32 retCount;
static ViUInt32 writeCount;
static unsigned char buffer[100];
static char stringinput[512];

/*
* In every source code or header file that you use it is necessary to prototype
* your VISA variables at the beginning of the file. You need to declare the VISA
* session, VISA integers, VISA strings, VISA pointers, and VISA floating variables.
* Remember that if you are prototyping variables that are to be used as part of the
* VISA session that need this prototyping. As an example, above retCount has been
* prototyped as a static variable to this particular module.   It is an integer of
* bit length 32. If you are uncertain how to declare your VISA prototypes refer
* to the VISA help under the Section titled Type Assignments Table. The VISA
* help is located in your NI-VISA directory or folder.
*/

int main(void)
{
    /*
     * First we must call viOpenDefaultRM to get the resource manager
     * handle.  We will store this handle in defaultRM.
     */
   status=viOpenDefaultRM (&defaultRM);
   if (status < VI_SUCCESS)
   {
      printf("Could not open a session to the VISA Resource Manager!\n");
      exit (EXIT_FAILURE);
   }

    /*
     * Now we will open a VISA session to a device at Primary Address 2.
     * You can use any address for your instrument. In this example we are
     * using GPIB Primary Address 2.
     *
     * We must use the handle from viOpenDefaultRM and we must
     * also use a string that indicates which instrument to open.  This
     * is called the instrument descriptor.  The format for this string
     * can be found in the NI-VISA User Manual.
```

```
        * After opening a session to the device, we will get a handle to
        * the instrument which we will use in later VISA functions.
65      * The two parameters in this function which are left blank are
        * reserved for future functionality.  These two parameters are
        * given the value VI_NULL.
        *
        * This example will also work for serial or VXI instruments by changing
70      * the instrument descriptor from GPIB0::2::INSTR to ASRL1::INSTR or
        * VXI0::2::INSTR depending on the necessary descriptor for your
        * instrument.
        */
      status = viOpen (defaultRM,  "GPIB0::2::INSTR", VI_NULL, VI_NULL, &instr);
75    if (status < VI_SUCCESS)
      {
           printf ("Cannot open a session to the device.\n");
           goto Close;
      }
80
       /*
        * Set timeout value to 5000 milliseconds (5 seconds).
        */
      status = viSetAttribute (instr, VI_ATTR_TMO_VALUE, 5000);
85
       /*
        * At this point we now have a session open to the instrument at
        * Primary Address 2.  We can use this session handle to write
        * an ASCII command to the instrument.  We will use the viWrite function
90      * to send the string "*IDN?", asking for the device's identification.
        */
      strcpy(stringinput,"*IDN?");
      status = viWrite (instr, (ViBuf)stringinput, strlen(stringinput), &writeCount);
      if (status < VI_SUCCESS)
95    {
         printf("Error writing to the device\n");
         goto Close;
      }

100    /*
        * Now we will attempt to read back a response from the device to
        * the identification query that was sent.  We will use the viRead
        * function to acquire the data.  We will try to read back 100 bytes.
        * After the data has been read the response is displayed.
105     */
      status = viRead (instr, buffer, 100, &retCount);
      if (status < VI_SUCCESS)
      {
         printf("Error reading a response from the device\n");
110   }
      else
      {
         printf("Data read: %*s\n",retCount,buffer);
      }
115


      /*
       * Now we will close the session to the instrument using
```

```
        * viClose. This operation frees all system resources.
120     */
    Close:
       printf("Closing Sessions\nHit enter to continue.");
       fflush(stdin);
       getchar();
125    status = viClose(instr);
       status = viClose(defaultRM);

       return 0;
    }
```

G MEX-Funktion: Schreiben und Lesen auf ein VISA-Gerät

```
 1  // visaio.cpp (c) 20071030 KB

    // Write and/or Read VISA Instrument

 5  #include "mex.h"
    #include "visa.h"
    #include <string.h>

    static ViSession defaultRM;
10  static ViSession instr;
    static ViUInt32 retCount;
    static ViByte obuf[10000];

    void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
15       const mxArray *prhs[])
    {
      /* Check for proper number of arguments. */
      if ((nrhs < 1)||((nrhs+nlhs)<2)||(nlhs>1))
        mexErrMsgTxt("Invalid number of arguments:\n"
20        "Two outputs (VISA name and data) for writing,\n"
          "one output (VISA name) and one input for reading,\n"
          "two outputs and one input for writing and reading.");

      /* Inputs 1 and 2 must be strings. */
25    if (mxIsChar(prhs[0]) != 1)
        mexErrMsgTxt("Input 1 must be a string.");
      if (mxGetM(prhs[0]) != 1)
        mexErrMsgTxt("Input 1 must be a row vector.");
      if (nrhs > 1) {
30      if (mxIsChar(prhs[1]) != 1)
          mexErrMsgTxt("Input 2 must be a string.");
        if (mxGetM(prhs[1]) != 1)
          mexErrMsgTxt("Input 2 must be a row vector.");
      }
35
      /* Get resource manager*/
      if (viOpenDefaultRM(&defaultRM) < VI_SUCCESS)
        mexErrMsgTxt("Could not open a session to the VISA Resource Manager!\n");
      /* VISA name string */
40    ViChar* DEV = mxArrayToString(prhs[0]);
```

```
    /* Open device */
    if (viOpen(defaultRM, DEV, VI_NULL, VI_NULL, &instr) < VI_SUCCESS)
      mexErrMsgTxt("Cannot open a session to the device.\n");
    mxFree(DEV);
45  viSetAttribute (instr, VI_ATTR_TMO_VALUE, 3000);

    /* Write data string */
    if (nrhs > 1) {
      ViChar* DATA = mxArrayToString(prhs[1]);
50    viWrite (instr, (ViBuf)DATA, (ViUInt32)strlen(DATA), &retCount);
      mxFree(DATA);
    }

    /* Read string or uint8 data from instrument */
55  if (nlhs == 1) {
      viRead (instr, obuf, sizeof(obuf), &retCount);
      plhs[0] = mxCreateNumericMatrix(1, retCount, mxUINT8_CLASS, mxREAL);
      memcpy((ViPByte)mxGetData(plhs[0]), obuf, retCount);
    }
60
    viClose(instr);
    viClose(defaultRM);
  }
```