



# Peter Beater

---

Physical Computing – Automatisieren mit dem Arduino

Eine kurze Einführung in die Welt eingebetteter Systeme

Prof. Dr.-Ing. Peter Beater  
Fachhochschule Südwestfalen  
Fachbereich Maschinenbau-Automatisierungstechnik  
Lübecker Ring 2  
59494 Soest

Version 1.0  
Manuskript Version vom 3. 1. 2020

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Autors. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

© Dr. Peter Beater, 2020

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften. Der Verlag und der Autor gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag noch der Autor übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen.

ISBN 9783750437203

Herstellung und Verlag  
Books on Demand GmbH  
Gutenbergring 53  
D-22848 Norderstedt

Sage es mir, und ich werde es vergessen.  
Zeige es mir, und ich werde es vielleicht behalten.  
Lass es mich tun, und ich werde es können.

Konfuzius, \*551 v. Chr. †479 v. Chr.



In jedem Gerät und jeder Maschine befindet sich heute Mikroelektronik. Exemplarisch kann man ihren Siegeszug auf 1969 datieren, als die erste Quarzuhr verkauft wurde. Im Laufe der nächsten zwei Jahrzehnte verdrängte sie die mechanische Armbanduhr und verursachte Disruption in der Schweizer Uhrenindustrie. Diese musste sich völlig neu erfinden und ist von einem Anbieter von Gebrauchsgegenständen zu einem Anbieter von Luxusprodukten geworden. Ähnliche Startpunkte und Folgen kann man auch für viele andere Bereiche der Technik angeben. Zum Beispiel Maschinensteuerungen, die anfangs mit mechanischen Relais arbeiteten und dann mit speziell für diesen Zweck entwickelten Rechnern, den sogenannten Speicherprogrammierbaren Steuerungen. Oder aber Waagen, die früher rein mechanisch arbeiteten und heute auch elektronisch. Oder die Kraftstoffversorgung in PKWs, die früher mechanisch arbeitende Vergaser und heute elektronisch gesteuerte Einspritzanlagen besitzen.

Mikroelektronik ist auch Gegenstand vieler Studiengänge. Elektrotechniker und Informatiker befassen sich im Studium in erheblichem Ausmaß mit ihr. Manager und viele andere Nicht-Techniker benutzen sie, haben aber nur wenig Hintergrundwissen aus ihrem Studium. Um dies zu ändern, wurde am Fachbereich Maschinenbau-Automatisierungstechnik in Soest eine Veranstaltung für diese Zielgruppe erarbeitet, deren Inhalte dieses Buch beschreibt.

Wird Mikroelektronik direkt in Geräten und Maschinen eingesetzt, so spricht man von eingebetteten Systemen. Dieses Buch versucht, diese Systeme beispielhaft zu erklären. Dabei geht es vordergründig um Hardware und Software. Im Hintergrund stehen aber auch Konzepte aus der Automatisierungstechnik, um komplexere Aufgaben überhaupt beschreiben zu können.

Manche Menschen verwenden ihr Smartphone ausschließlich zum Telefonieren. Andere organisieren ihr gesamtes Leben damit, weil die Mikroelektronik und dazugehörige Software sehr leistungsfähig sind. Beim Einsatz des Arduinos gehören wir in diesem Buch zu ersten Gruppe. Er ist ein Werkzeug, um Konzepte zu vermitteln, und kann viel mehr, als hier gezeigt werden kann. Daher ist zum Beispiel das Verzeichnis der Programmierung in Anhang A1 nicht vollständig: Es enthält genau das, was wir in diesem Buch brauchen. Allerdings gehen wir auch auf einige Eigenschaften des Prozessors ein, die in vielen Anwendungsbüchern zum Arduino nicht beschrieben werden, z. B. Interrupts.



## Inhaltsverzeichnis

<b>1</b>	<b>Boolesche Algebra .....</b>	<b>1</b>
1.0	Inhalt dieses Kapitels .....	1
1.1	UND-Verknüpfung .....	1
1.2	ODER-Verknüpfung .....	3
1.3	Negation .....	4
1.4	NOR-Verknüpfung .....	4
1.5	NAND-Verknüpfung .....	5
1.6	Exklusiv-ODER-Verknüpfung .....	5
1.7	Priorität .....	6
<b>2</b>	<b>Flipflops und Zeitglieder .....</b>	<b>11</b>
2.0	Inhalt dieses Kapitels .....	11
2.1	Speicherglieder .....	11
2.2	Zeitglieder .....	15
<b>3</b>	<b>Erste Programme für den Arduino .....</b>	<b>21</b>
3.0	Inhalt dieses Kapitels .....	21
3.1	Der Arduino Uno R3 und die Entwicklungsumgebung IDE .....	21
3.2	Der Arduino als Taschenrechner .....	25
3.4	Variablen und ihre Eigenschaften .....	34
3.5	Funktionen .....	38
3.6	Exkurs zu den Anweisungen & und   .....	40
<b>4</b>	<b>Der Arduino und die physikalische Welt .....</b>	<b>43</b>
4.0	Inhalt dieses Kapitels .....	43
4.1	Anschlussmöglichkeiten des Arduino Uno R3 .....	43
4.2	Entscheidungen mit der if-Anweisung .....	58
<b>5</b>	<b>Steuerungen mit dem Arduino .....</b>	<b>61</b>
5.0	Inhalt dieses Kapitels .....	61
5.1	Ein „Betriebssystem“ für den steuernden Arduino .....	61
5.2	Flipflop mit dem Arduino .....	62
5.4	Einschaltverzögerung mit dem Arduino .....	66
5.6	Arduino-Realisierung der Zwei-Hand-Steuerung .....	76
5.7	Bibliothek AT für den Arduino .....	80



<b>6 Ablaufsteuerungen .....</b>	<b>85</b>
6.0 Inhalt dieses Kapitels .....	85
6.1 Ablaufsteuerungen und Darstellung im Funktionsplan .....	85
6.2 Auswahl- und Parallelbetrieb.....	91
6.3 Ein Ablauf für den Arduino .....	92
6.4 Der Arduino hilft Fußgängern über die Straße .....	96
6.5 Programmieren einer SPS in Ablaufsprache (AS).....	100
 <b>7 Dem Arduino wird warm und kalt - Analogwertverarbeitung.....</b>	 <b>103</b>
7.0 Inhalt dieses Kapitels .....	103
7.1 Datenerfassung und Auswertung .....	103
7.2 Programmierung für den Arduino.....	106
7.3 Anzeigen der Werte auf einem LCD-Display .....	109
7.4 Einfache oder schöne Displays .....	112
7.5 Einlesen von Daten über die serielle Schnittstelle .....	113
7.6 Aktive Temperatursensoren entlasten den Arduino.....	114
 <b>8 Der Arduino regelt es.....</b>	 <b>119</b>
8.0 Inhalt dieses Kapitels .....	119
8.1 Die Aufgabe und der Aufbau .....	119
8.2 Dynamisches und stationäres Verhalten .....	121
8.3 PI-Regler .....	127
8.4 Puls-Breiten-Modulation des Leistungsteils .....	130
8.5 Arduino-Programm für den PI-Regler .....	131
8.6 Arduino-Programme zur Simulation des Lötkolbens .....	133
 <b>9 Der Arduino nimmt den Bus.....</b>	 <b>139</b>
9.0 Inhalt dieses Kapitels .....	139
9.1 Der I <sup>2</sup> C-Bus und die Bibliothek wire.....	139
9.2 Datenübertragung zwischen zwei Arduinos mit I <sup>2</sup> C-Bus .....	142
9.3 Adressensuche auf dem I <sup>2</sup> C-Bus.....	144
9.4 Ansteuerung eines D/A-Wandlers über I <sup>2</sup> C-Bus.....	145
9.5 Der Arduino als Double .....	147
9.6 Die serielle Schnittstelle .....	155
9.7 Feldbusse in der Automatisierungstechnik .....	157

<b>10</b>	<b>Zahlendarstellung und Rechnungen im Digitalrechner .....</b>	<b>161</b>
10.0	Inhalt dieses Kapitels .....	161
10.1	Zahlendarstellungen im täglichen Leben .....	161
10.2	Darstellung ganzer Zahlen im Dualsystem .....	162
10.3	Addition und Subtraktion von Dualzahlen .....	164
10.4	Gleitkommazahlen .....	168
<b>11</b>	<b>Programmausführung und Rechnerarchitektur .....</b>	<b>175</b>
11.0	Inhalt dieses Kapitels .....	175
11.1	Arbeitsweise eines Mikrocontrollers .....	175
11.2	Assembler .....	181
11.3	Programmausführung .....	183
11.4	Programmstart und der Bootloader .....	185
11.5	Programmverzweigungen und Entscheidungen .....	186
11.6	Interrupts .....	188
11.7	Typische Merkmale eines Mikrocontrollers .....	191
<b>12</b>	<b>Abgekapselt programmieren .....</b>	<b>193</b>
12.0	Inhalt dieses Kapitels .....	193
12.1	Arduino language und C und C++ .....	193
12.2	Objektorientiertes Programmieren .....	195
12.3	Graphische Programmierung mit ARDUBlock .....	196
<b>13</b>	<b>Literatur.....</b>	<b>199</b>

<b>A Anhang .....</b>	<b>203</b>
A1 Programmieranweisungen.....	203
A1.1 Datentypen und Variablen .....	203
A1.2 Verknüpfungen .....	204
A1.3 Funktionen .....	204
A1.4 if-Anweisung .....	205
A1.5 for-Anweisung.....	206
A1.6 while-Anweisung .....	206
A1.7 switch-Anweisung .....	207
A1.8 Übertragung mit I <sup>2</sup> C-Bus und wire .....	207
A1.9 Serielle Übertragung mit der USB-PC Schnittstelle und Serial .....	208
A1.10 Konfiguration der Pins.....	208
A2 Mechanischer Aufbau .....	210
A3 ASCII-Tabelle.....	212
A4 LCD-Display 16x2 mit Treiber HD44780 .....	213
 <b>Sachverzeichnis .....</b>	 <b>215</b>

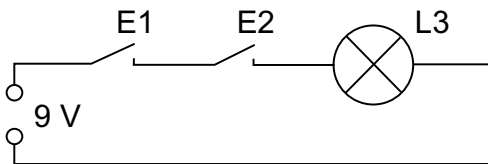
# 1 Boolesche Algebra

## 1.0 Inhalt dieses Kapitels

In der Automatisierungstechnik haben wir es mit Abläufen und logischen Verknüpfungen zu tun. Dabei ist es wichtig, dass wir die Zusammenhänge eindeutig beschreiben, wofür die Mathematik die Werkzeuge liefert. Die Grundlagen dazu werden in diesem Kapitel vorgestellt. Anwenden werden wir sie nicht nur bei Steuerungen, sondern auch bei Programmieraufgaben.

## 1.1 UND-Verknüpfung

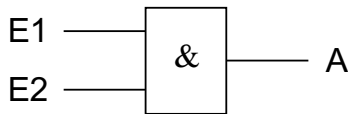
Eine Lampe soll nur leuchten, wenn der linke *und* der rechte als Schließer ausgeführte Handschalter betätigt werden. Dies bezeichnen wir als *UND-Verknüpfung*. Der Aufbau ist sehr einfach: Wir nehmen zwei Klingeltaster und schalten sie in Reihe.



**Bild 1-1** UND-Verknüpfung als elektrische Schaltung

Da wir den Arduino, also einen Digitalrechner, verwenden wollen, betrachten wir im Weiteren nicht die Komponenten, sondern die damit erzeugten Signale. Das Signal E1 steht bei uns für die Betätigung des Tasters. Ist der Taster unbetätigt, ist E1 logisch 0. Ist der Taster betätigt, ist das Signal E1 logisch 1. Entsprechend gilt, dass die Lampe leuchtet, wenn das Signal L3 logisch 1 ist. Beim Arduino verbinden wir logisch 1 mit der Spannung 5 V und logisch 0 mit 0 V. Bei einer industriellen Speicherprogrammierbaren Steuerung sind es 24 V für logisch 1 und 0 V für logisch 0.

Da die Variable E1 *und* die Variable E2 logisch 1 sein müssen, damit die Ausgangsvariable A logisch 1 ist, spricht man von einer *UND-Verknüpfung*. Diese logische Verknüpfung kann man auch ganz abstrakt darstellen, wobei Symbole dafür in DIN EN 60617-12 genormt sind<sup>1</sup>. In der Norm heißt es: „Das allgemeine Funktionskennzeichen des Elements gibt an, wie viele Eingänge sich im internen 1-Zustand befinden müssen, damit die Ausgänge ihre internen 1-Zustände einnehmen.“ Bei der UND-Verknüpfung ist der Ausgang nur dann im 1-Zustand, wenn *alle* Eingänge im 1-Zustand sind.



**Bild 1-2** Schaltzeichen einer UND-Verknüpfung als Funktionsbaustein

Bei einem Funktionsbaustein befinden sich die Eingänge links und der Ausgang rechts. Der Ausgang eines Bausteins darf mit mehreren Eingängen verbunden werden. Ausgänge dürfen *nicht* miteinander verbunden werden.

Das logische UND ist auch für Verknüpfungen mit mehr als zwei Eingängen definiert: Der Ausgang ist nur dann logisch 1, wenn *alle* Eingänge logisch 1 sind. Das logische UND ist in DIN 66000 als mathematisches Zeichen genormt:

$$E1 \wedge E2 = A.$$

Gesprochen wird dies als: „E1 und E2 ist gleich A“.

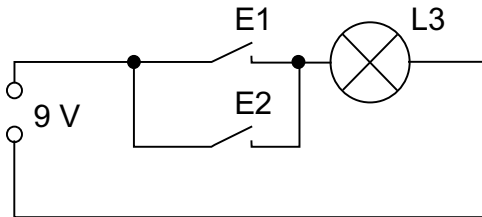
Der Arduino beherrscht auch die Boolesche Algebra und verwendet für die UND-Verknüpfung den Operator `&&`. Wenn wir die Variablen E1, E2 und L3 im Programm definiert haben, können wir folgende Berechnung ausführen:

```
L3 = E1 && E2;
```

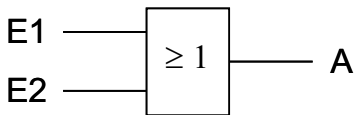
<sup>1</sup> Die Symbole der alten DIN 40700 und der US-Norm (Exportdokumentation) sind anders.

## 1.2 ODER-Verknüpfung

Schaltet man die zwei Taster nicht in Reihe, sondern *parallel*, ergibt sich eine *ODER-Verknüpfung*. Die Lampe leuchtet, wenn der eine Taster *oder* der andere Taster oder beide betätigt sind.



**Bild 1-3** ODER-Verknüpfung als elektrische Schaltung



**Bild 1-4** Schaltzeichen einer ODER-Verknüpfung als Funktionsbaustein

Das Zeichen  $\geq 1$  im Schaltzeichen bedeutet dabei, dass der Ausgang logisch 1 ist, wenn ein oder mehrere Eingänge logisch 1 sind.

Das logische ODER ist auch für Verknüpfungen mit mehr als zwei Eingängen definiert: Der Ausgang ist dann logisch 1, wenn mindestens ein Eingang logisch 1 ist. Das logische ODER ist in DIN 66000 als mathematisches Zeichen genormt:

$$E1 \vee E2 = A.$$

Gesprochen wird dies als: „E1 oder E2 ist gleich A“.

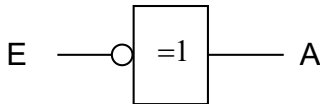
Der Arduino verwendet für die ODER-Verknüpfung den Operator `||`.

```
L3 = E1 || E2;
```

### 1.3 Negation

Wichtig ist noch die *Negation*. Dabei wird aus logisch 1 ein logisch 0 und umgekehrt. Die Negation wird formelmäßig durch einen Strich über der Größe dargestellt, z. B.  $\overline{E}$ . Gesprochen wird dies als: „Nicht E“ oder „E quer“.

$$\overline{\overline{E}} = E.$$



**Bild 1-5** Schaltzeichen einer Negation als Funktionsbaustein

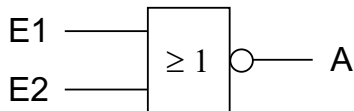
Im Zusammenhang mit weiteren Funktionsbausteinen wird die *Negation* nur durch einen kleinen Kreis bei der Eingangs- bzw. Ausgangsleitung dargestellt.

Der Arduino verwendet für die Negation den Operator `!`.

```
A = ! E;
```

### 1.4 NOR-Verknüpfung

Schaltet man zwei Öffner in Reihe, erhält man eine *NOR-Verknüpfung*. Diese erhalten wir auch, wenn wir zwei Eingangsgrößen mit einer ODER-Verknüpfung verbinden und das Ausgangssignal negieren; NOR steht für **Not OR**, s. auch Bild 1-6.

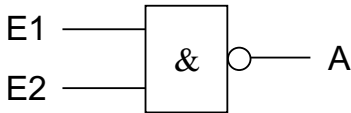


**Bild 1-6** Schaltzeichen einer NOR-Verknüpfung

Man kann zeigen, dass alle kombinatorischen Operationen ausschließlich mit NOR-Verknüpfungen formuliert werden können. Daher bezeichnet man die NOR-Verknüpfung auch als *vollständig*.

## 1.5 NAND-Verknüpfung

Die *NAND-Verknüpfung* erhalten wir, wenn wir das Ausgangssignal eines UND-Gliedes negieren. Oder indem wir zwei Öffner parallel schalten.



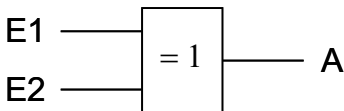
**Bild 1-7** Schaltzeichen einer NAND-Verknüpfung

Man kann zeigen, dass alle kombinatorischen Operationen ausschließlich mit NAND-Verknüpfungen formuliert werden können. Daher bezeichnet man die NAND-Verknüpfung auch als *vollständig*.

Die NAND-Verknüpfung gehörte zu den ersten Logikbausteinen, die als integrierte Schaltung unter der Bezeichnung SN7400 in den 1970er-Jahren von Texas Instruments angeboten wurden, s. a. Beispiel 1-4.

## 1.6 Exklusiv-ODER-Verknüpfung

Das Ausgangssignal der ODER-Verknüpfung ist logisch 1, wenn entweder das eine oder das andere oder aber beide Eingangssignale logisch 1 sind. Bei der *Exklusiv-ODER-Verknüpfung* ist das Ausgangssignal nur dann logisch 1, wenn *entweder das eine oder aber das andere Eingangssignal* logisch 1 ist, nicht aber, wenn beide logisch 1 sind.



**Bild 1-8** Schaltzeichen einer Exklusiv-ODER-Verknüpfung als Funktionsbaustein



## 1.7 Priorität

Beim täglichen Rechnen beachten wir die unterschiedliche Wichtigkeit von Verknüpfungen, ohne weiter darüber nachzudenken. Z. B. wissen wir, dass die Punktrechnung vor der Strichrechnung ausgeführt wird. Bei der Booleschen Algebra nehmen wir uns den Arduino als Maßstab und definieren, dass Klammern die höchste Priorität besitzen und dass dann die UND-Verknüpfung Priorität vor der ODER-Verknüpfung besitzt:

$$X \vee Y \wedge Z = X \vee (Y \wedge Z).$$

In der SPS-Programmiersprache „Strukturierter Text“ nach DIN EN 61131-3 ist ebenfalls diese Priorität vorgesehen. Aber in der Norm DIN 66000 ist dies anders definiert! Vorsichtige Menschen verwenden daher Klammern.

### Beispiel 1–1

Gegeben ist die Schaltfunktion

$$Y = 1 \wedge 1 \vee 0 \wedge 0.$$

Wenn wir, wie vereinbart, *Priorität* der UND-Verknüpfung haben, erhalten wir als Ergebnis:

$$Y = (1 \wedge 1) \vee (0 \wedge 0) = 1 \vee 0 = 1.$$

*Ohne* Priorität der UND-Verknüpfung, also beim Abarbeiten des Ausdrucks von links nach rechts, erhielten wir als Ergebnis:

$$Y = [(1 \wedge 1) \vee 0] \wedge 0 = [(1) \vee 0] \wedge 0 = [1 \vee 0] \wedge 0 = 1 \wedge 0 = 0.$$

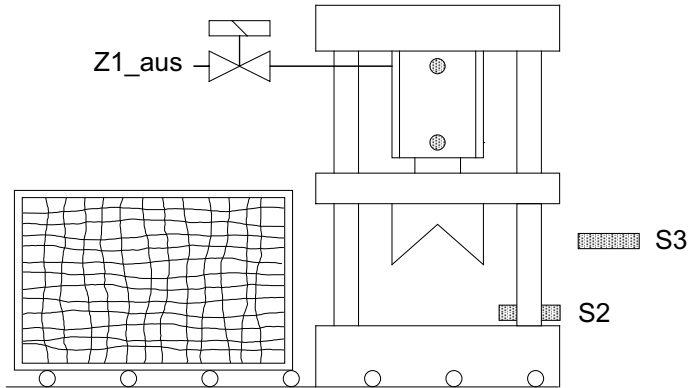
### Beispiel 1–2

In Bild 1-9 ist eine Stanze dargestellt. Für die Gesamtsteuerung ist folgende Teilaufgabe zu lösen:

Der Stanzvorgang wird bei einer der folgenden Bedingungen ausgeführt:

- 1) Die (nicht dargestellten) Handtaster T4 und Handtaster T5 sind gleichzeitig betätigt,
- 2) Sensor S3 meldet, dass das Schutzgitter geschlossen ist, und der Fußtaster T6 ist betätigt,

3) das Schutzgitter ist geschlossen und einer der Handtaster ist betätigt.  
Zusätzlich muss der Sensor S2 anzeigen, dass Stanzgut vorhanden ist.



### Bild 1-9 Schematische Darstellung einer Stanze

Als erstes müssen wir die Zustände der Komponenten unseren Variablen zuordnen, s. Tabelle 1.1.

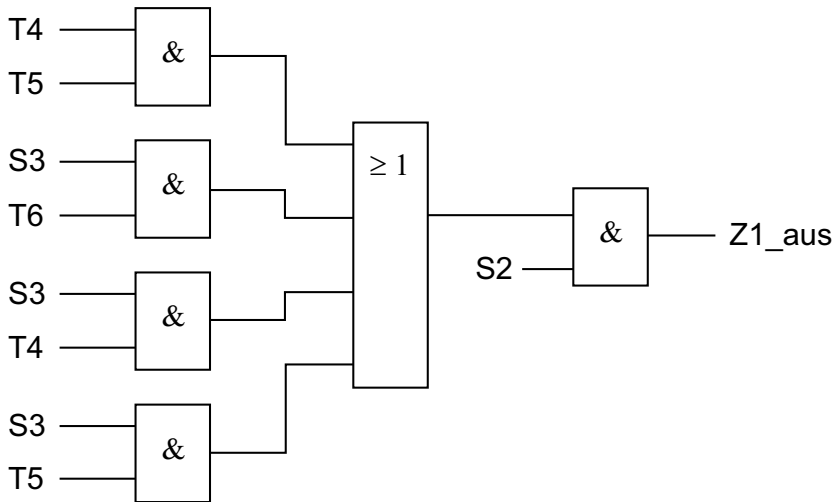
**Tabelle 1.1** Zuordnung der Komponenten zu den Variablen

Komponente	Variable	Definition der logischen Zustände
Sensor	S2	Stanzgut vorhanden $\Rightarrow S2 = 1$ , sonst $S2 = 0$
Sensor	S3	Gitter geschlossen $\Rightarrow S3 = 1$ , sonst $S3 = 0$
Handtaster	T4	Taster gedrückt $\Rightarrow T4 = 1$ , sonst $T4 = 0$
Handtaster	T5	Taster gedrückt $\Rightarrow T5 = 1$ , sonst $T5 = 0$
Fußtaster	T6	Taster gedrückt $\Rightarrow T6 = 1$ , sonst $T6 = 0$
Ventil	Z1_aus	$Z1\_aus = 1 \Rightarrow$ Kolbenstange fährt aus, sonst Stillstand

Als Schaltgleichung ergibt sich:

$$Z1\_aus = [(T4 \wedge T5) \vee (S3 \wedge T6) \vee (S3 \wedge T4) \vee (S3 \wedge T5)] \wedge S2 \text{ .}$$

Dabei haben wir, wie in der gewöhnlichen Algebra, Klammern verwendet. Wir können diese Gleichung auch als Funktionsbaustein-Plan zeichnen:



**Bild 1-10** Funktionsbaustein-Plan der Steuerung zu Bild 1-9

Wenn wir diese Steuerung mit dem Arduino realisieren wollen, müssen wir zusätzlich noch wissen, an welchen Anschlüssen die jeweiligen Sensoren bzw. das Ventil als Stellglied angeschlossen sind und die Signalzustände den Variablen zuordnen. Die Berechnung führen wir aus mit:

```
Z1_aus = ( (T4 && T5) || (S3 && T6) || (S3 && T4) || (S3 && T5) ) && S2;
```

Für eine reale Presse reicht unsere Steuerung aber noch nicht aus, da die Sicherheits-schaltung sehr einfach zu umgehen ist. Ein Holzklotz, der auf den Handtaster  $T_4$  gelegt wird und diesen betätigt, macht aus der Zweihandbedienung eine Einhandbedienung. Um diese Umgehung auszuschließen, werden wir die Steuerung so erweitern, dass die beiden Taster gleichzeitig betätigt werden müssen, damit die Presse arbeitet, s. Beispiel 2–2 und Abs. 5.6.

### Beispiel 1–3

Tabelle 1.2 gibt die Schalttafel eines sogenannten *Halbaddierers* an. Diese Schaltung wird verwendet, um in einem Rechner zwei einstellige Dualzahlen,  $a$  und  $b$ , zu addieren. Ausgangsgrößen sind die Summe  $y$  und das Übertragsbit  $c_{out}$ .

**Tabelle 1.2** Schalttablelle des Halbaddierers

a	b	y	c <sub>out</sub>
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

## Aufgaben

- a) Geben Sie die Schaltgleichung für y an.  
 b) Geben Sie die Schaltgleichung für c<sub>out</sub> an.

- a) Für den Ausgang y ergibt sich als Schaltgleichung:

$$y = (\bar{a} \wedge b) \vee (a \wedge \bar{b}) .$$

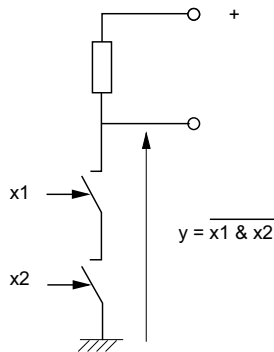
- b) Für das Übertragsbit c<sub>out</sub> ergibt sich als Schaltgleichung:

$$c_{out} = a \wedge b .$$

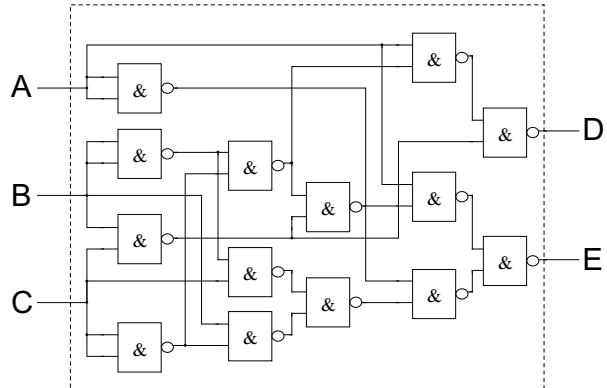
## Beispiel 1–4

Wir werden mit dem Arduino in den nächsten Kapiteln algebraische Berechnungen ausführen. Die verschiedenen Rechenoperationen lassen sich auf Additionen zurückführen. Die dazu benötigten Addierer können wir aus NAND-Bausteinen aufbauen.

Den Aufbau eines NAND-Bausteins zeigt schematisch Bild 1-11a. Hier sind die in der elektronischen Schaltung vorhandenen Transistoren durch Kontakte ersetzt, die von den zwei Eingangssignalen geschlossen werden.



**Bild 1-11a** Schematische Darstellung des Aufbaus eines NAND-Bausteins



**Bild 1-11b** Aufbau eines Addiereres aus NAND-Bausteinen

Wenn die beiden Signale  $x_1$  und  $x_2$  0 sind, liegt am Anschluss die Versorgungsspannung an. Sind beide Signale 1, so ergibt sich eine Ausgangsspannung von 0 V. Aus NAND-Bausteinen können wir einen sogenannten *Volladdierer* aufbauen, der drei Dualzahlen addiert, s. Bild 1-11b. Zusammen mit dem Halbaddierer aus Beispiel 1–3 können wir mit entsprechend vielen Volladdierern eine Schaltung aufbauen, die zwei z. B. 8-stellige Dualzahlen addiert. Die jeweils rechten Bits addieren wir mit dem Halbaddierer, der neben dem Ergebnis auch noch das Übertragsbit  $c_{out}$  liefert. Bei der zweiten bis 8. Stelle müssen wir dann immer drei Dualzahlen addieren: Die beiden Summanden und das Übertragsbit der vorherigen Addition, s. auch Abs. 10.3.

**Tabelle 1.3** Schalttable des Volladdierers

A	B	C	D	E
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
1	0	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

## 2 Flipflops und Zeitglieder

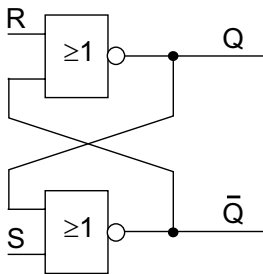
### 2.0 Inhalt dieses Kapitels

Bei den in Kapitel 1 behandelten Verknüpfungssteuerungen hängt das Ausgangssignal nur von den jeweiligen aktuellen Eingangssignalen ab: Es gibt *keine* Speicher. Bei vielen Steuerungsaufgaben ist es aber nötig, sich einen Schaltzustand „zu merken“, das heißt, ein Speicherglied zu verwenden. Dies ist z. B. erforderlich, um eine Bediener-eingabe zu speichern (Einschalten eines Gerätes) oder um einen Prozesszustand festzuhalten. Steuerungen mit inneren Zuständen bezeichnet man auch als Schaltwerk.

Häufig stehen wir vor der Aufgabe, eine Signaländerung um ein paar Sekunden zu verzögern. Dazu stehen uns Zeitglieder zur Verfügung, die wir in Abs. 2.2 untersuchen.

### 2.1 Speicherglieder

Ein typisches Speicherglied ist das *RS-Flipflop*. Wir stellen uns zunächst vor, dass es aus einzelnen Transistoren in klassischer Analogtechnik zusammengelötet ist.



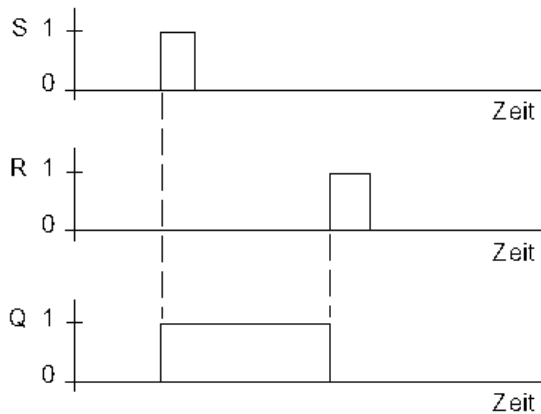
**Bild 2-1** RS-Flipflop, aufgebaut aus NOR-Gliedern in klassischer Analogtechnik

Das Verhalten des in klassischer Analogtechnik aufgebauten RS-Flipflops zeigt die Schritt-Tabelle 2.1. Wir beginnen mit Schritt 1 und einem 1-Signal für den Eingang S und einem 0-Signal für den Eingang R. Es ergibt sich  $Q = 1$ . Setzen wir das Signal am Eingang S auf 0, so bleibt der Ausgang Q dennoch auf 1. Erst wenn wir im Schritt 3 den Eingang R auf 1 setzen, wird der Ausgang Q zu 0. Im Schritt 4 setzen wir auch den Eingang R auf 0 und der Ausgang Q bleibt 0.

**Tabelle 2.1** Schritt-Tabelle zum RS-Flipflop

Schritt	S	R	Q	$\bar{Q}$
	0	0	*	*
1	1	0	1	0
2	0	0	1	0
3	0	1	0	1
4	0	0	0	1

\* unverändert



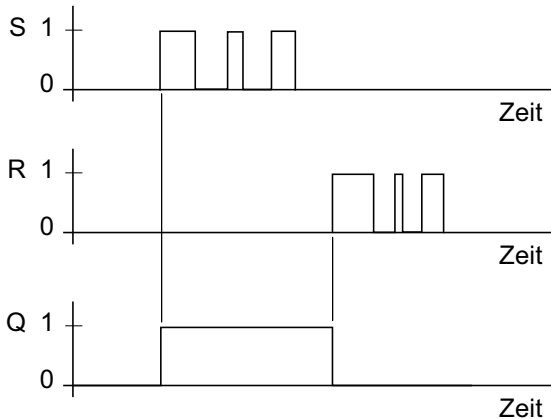
**Bild 2-2** Schaltfolge-Diagramm des RS-Flipflops

Wir können also feststellen:

- Ein 1-Zustand am Eingang S sorgt dafür, dass der Ausgang Q auf 1 gesetzt wird. Ist er bereits 1, so ändert sich nichts. Es handelt sich um den *Setz*-Eingang.
- Ein 1-Zustand am Eingang R sorgt dafür, dass der Ausgang Q auf 0 gesetzt wird. Ist er bereits 0, so ändert sich nichts. Es handelt sich um den *Rücksetz*-Eingang.

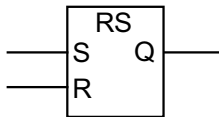
- Ein 1-Zustand an beiden Eingängen ist nicht zulässig, da dann sowohl  $Q = 0$  als auch  $\bar{Q} = 0$  gilt und die Bedingung  $\bar{Q} \neq Q$  nicht erfüllt ist.

Wir können somit sagen: Das RS-Flipflop „speichert“ den letzten Umschalt-Befehl. Ein wiederholtes 1-Signal am gleichen Eingang ändert das Ausgangssignal nicht, s. Bild 2-3.



**Bild 2-3** Zeitdiagramm der Ein- und Ausgangssignale des RS-Flipflops

Das Schaltzeichen des RS-Flipflops zeigt das Bild 2-4.



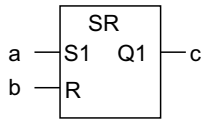
**Bild 2-4** Schaltzeichen des RS-Flipflops

Beim obigen RS-Flipflop ist es „verboten“, gleichzeitig ein 1-Signal an den S- und an den R-Eingang anzulegen, da dann nicht gilt  $\bar{Q} \neq Q$ . Dieser Fall kommt aber bei Steuerungen im wirklichen Leben vor. Daher ist bei den in der Automatisierungstechnik verwendeten Flipflops das Verhalten bei gleichzeitigem Setzen und Rücksetzen definiert. Dabei unterscheidet man zwischen *setzdominanten*<sup>1</sup> bzw. *rücksetzdominanten*<sup>2</sup> Speichergliedern.

<sup>1</sup> Unter dem Begriff „dominieren“ versteht man vorherrschen oder beherrschen. Diese

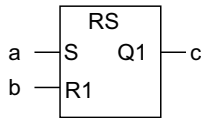


Die jeweilige Dominanz kennzeichnen wir mit einer 1 für den Setzeingang, S1, bzw. für den Rücksetzeingang, R1.



a	b	c
0	0	unverändert
0	1	0
1	0	1
1	1	<b>1</b>

**Bild 2-5** Flipflop mit dominierendem Setzen, Funktionsbaustein SR



a	b	c
0	0	unverändert
0	1	0
1	0	1
1	1	<b>0</b>

**Bild 2-6** Flipflop mit dominierendem Rücksetzen, Funktionsbaustein RS

Wenn wir das Flipflop mit einem Rechnerprogramm realisieren, gibt es nur dominante Flipflops.

Mit dem Arduino können wir ein Flipflop programmieren, indem wir Bedingungen stellen:

```
if (setzen == true)    {Ausgang = true;}
if (ruecksetzen == true) {Ausgang = false;}
```

Die Variable Ausgang wird auf logisch 1, d. h. true, gesetzt, wenn die Variable setzen wahr ist. Die Variable Ausgang wird auf logisch 0, d. h. false, gesetzt, wenn die Variable ruecksetzen wahr ist. Das Programm wird von oben nach unten zeilenweise abgearbeitet. Wenn sowohl setzen als auch ruecksetzen wahr sind, wird Ausgang zunächst auf true, danach auf false gesetzt. Somit dominiert das Rücksetzen. Wenn wir die Dominanz ändern wollen, vertauschen wir die Reihenfolge der zwei Zeilen.

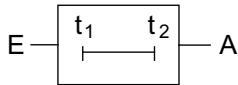
---

Flipflops werden auch „vorrangig setzend“ genannt.

<sup>2</sup> Die rücksetzdominanten Flipflops werden auch „löschdominant“ oder „vorrangig rücksetzend“ genannt.

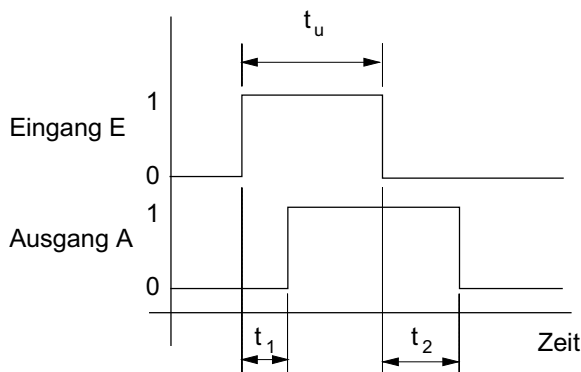
## 2.2 Zeitglieder

Häufig stellt sich die Aufgabe, ein Signal zu verzögern oder aber auch ein Signal eine bestimmte Zeit zu halten. Dazu werden *Zeitglieder* verwendet.



**Bild 2-7** Schaltzeichen eines Zeitglieds

Im Schaltzeichen in Bild 2-7 gibt  $t_1$  die *Einschalt-* und  $t_2$  die *Ausschaltverzögerungszeit* an.



**Bild 2-8** Definition der Verzögerungszeiten  $t_1$  und  $t_2$ ; die Signaldauer beträgt  $t_u$

In SPS-Systemen gibt es das Zeitglied aus Bild 2-7 nicht. Stattdessen gibt es reine Einschaltverzögerungen, Baustein TON, oder reine Ausschaltverzögerungen, Baustein TOF. Für den Arduino und die weiteren Aufgaben werden wir in diesem Buch eine reine Einschaltverzögerung verwenden, die wir in Abs. 5.4 selber programmieren.

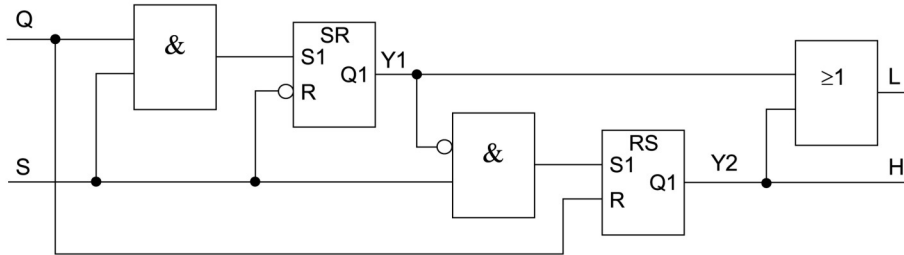
Es gibt eine Anweisung, mit der sehr einfach eine Verzögerung im Arduino programmiert werden kann:

```
delay(1000);
```

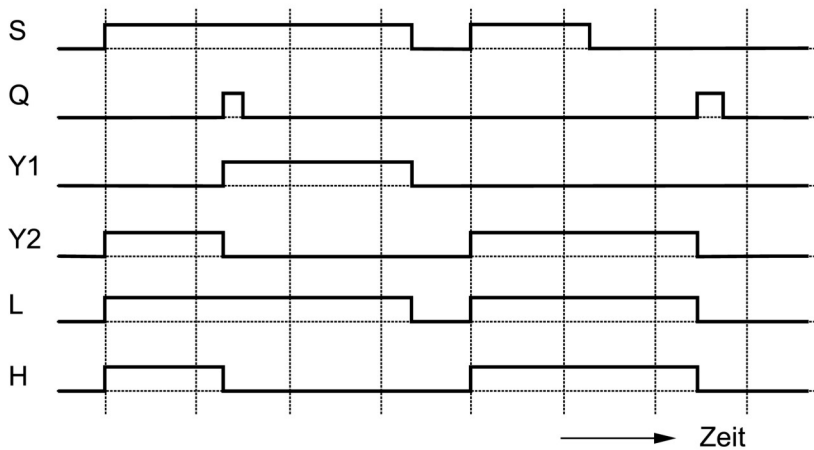
Nach dieser Zeile wartet der Arduino 1000 ms = 1 s, bis er die nächste Anweisung ausführt. Allerdings können wir in dieser Zeit auch keine anderen Aufgaben erledigen, so dass wir noch andere Möglichkeiten kennenlernen werden, eine Verzögerung zu realisieren.

### Beispiel 2-1

Bild 2-9 zeigt eine Alarmschaltung. Bild 2-10 zeigt die zwei Eingangssignale Q und S und die sich daraus ergebenden Zeitverläufe der anderen Signale.



**Bild 2-9** Funktionsbaustein-Plan

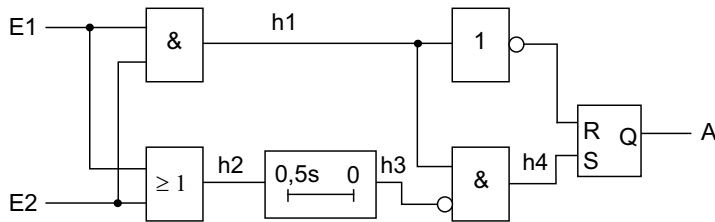


**Bild 2-10** Zeitverlauf für die Eingangssignale S und Q und daraus resultierende Signale

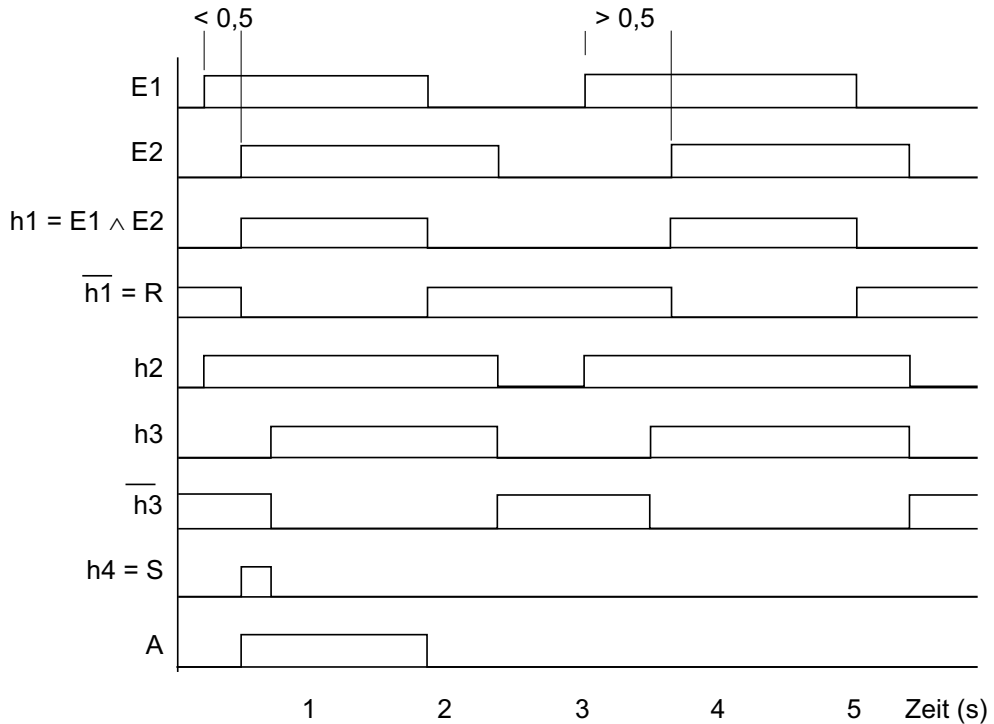
Eine Steuerung wie in Bild 2–9 können wir verwenden, um eine Alarmschaltung für eine Maschine zu realisieren. Wie Bild 2–10 zeigt, werden sowohl die Lampe L als auch die Hupe H eingeschaltet, wenn die Störung S auftritt. Drückt der Monteur dann auf die Taste Quittung Q, so verstummt die Hupe. Ist die Störung beseitigt, so geht auch die Lampe aus. Für den Fall, dass die Störung verschwindet, bevor der Monteur die Taste Q gedrückt hat, bleiben Lampe und Hupe eingeschaltet, bis mit Q quittiert wird.

## Beispiel 2–2

In Bild 2–11 sehen wir den Funktionsbaustein-Plan einer Steuerung. Bild 2–12 zeigt oben den zeitlichen Verlauf der Eingangssignale  $E1(t)$  und  $E2(t)$ . Dann folgen die resultierenden internen Signale  $h1$  bis  $h4$  und schließlich das Ausgangssignal  $A(t)$ . Dabei gehen wir davon aus, dass das Flipflop am Anfang im logischen 0-Zustand ist.



**Bild 2-11** Funktionsbaustein-Plan



**Bild 2-12** Verlauf der Eingangssignale E1 und E2 und der daraus resultierende Signale der Steuerung in Bild 2-11

Eine Steuerung wie in Bild 2–11 können wir verwenden, wenn wir einen sicheren Betrieb einer Presse realisieren wollen. Wie Bild 2–12 zeigt, müssen die Signale E1 und E2 fast gleichzeitig von 0 auf 1 *wechseln*, damit das Ausgangssignal A ebenfalls von 0 auf 1 wechselt. Ist die Zeit zwischen den Flankenwechseln von E1 und E2 größer als 0,5 s, so bleibt das Ausgangssignal  $A = 0$ . Das heißt, dass beim Bedienen die zwei Handtaster, die die Signale E1 bzw. E2 erzeugen, ungefähr gleichzeitig gedrückt werden müssen. Ein Blockieren eines der beiden Taster und Auslösen der Presse mit dem anderen Taster ist so nicht möglich.

Mit dem Arduino können wir diese Steuerung wie folgt realisieren:

```
h1 = E1 && E2;
h2 = E1 || E2;

zeitglied.Update(h2, T500);
h3 = zeitglied.ZeitgliedAusgang;
h4 = h1 && (! h3);

rs.Update(h4, ! h1);
Ausgang = rs.RSAusgang;
```

Benötigt wird vorher die Definition der Variablen und Zuordnung zu den Pins. Ebenso müssen wir noch die Modelle für das Zeitglied und das Flipflop erstellen, s. Kap. 5.

Mit der Anweisung `rs.Update()` werden dem Flipflop die aktuellen Parameter zugewiesen. Mit der Anweisung `Ausgang = rs.RSAusgang;` wird der Zustand des Bausteins aktualisiert und der aktuelle Wert der Variablen `Ausgang` zugewiesen.

In einer kommerziellen Maschine setzt man nicht den Arduino, sondern eine Speicherprogrammierbare Steuerung, SPS, ein. Deren Programm, in der Sprache Strukturierter Text (ST), sieht unserem Arduino-Programm recht ähnlich.

Bei einem SPS-Programm deklarieren wir als erstes die Ein- und Ausgänge als globale Variablen. Dazu dient das Schlüsselwort `VAR_GLOBAL`.

```
VAR_GLOBAL
E1 AT %IX2.0: BOOL;
E2 AT %IX2.1: BOOL;
A AT %QX2.0: BOOL;
```

```
END_VAR
```

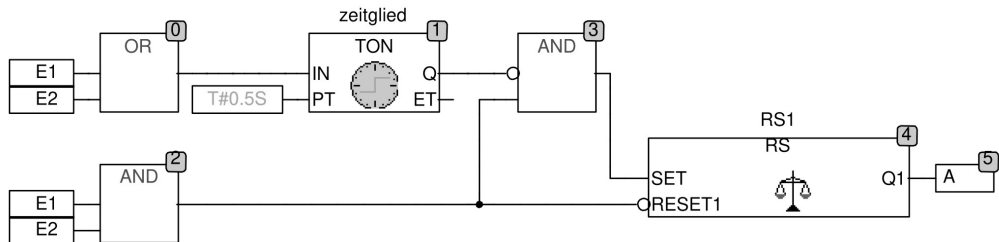
Die Variablen, die wir in unserem Programm benötigen, definieren wir mit dem Schlüsselwort `VAR`. Dazu gehören auch Funktionsbausteine, wie z. B. das Zeitglied.

```
VAR
h1: BOOL;
h2: BOOL;
h3: BOOL;
h4: BOOL;
RS1: RS;
zeitglied: TON;
END_VAR
```

Für das SPS-Programm dient unser Arduino-Programm als Vorlage:

```
PROGRAM ZweiHandST
h1 := E1 AND E2;
h2 := E1 OR E2;
zeitglied( IN := h2, PT := T#0.5s );
h3 := zeitglied.Q;
h4 := h1 AND NOT h3;
RS1( SET := h4, RESET1 := NOT h1 );
A := RS1.Q1;
```

Das Programm des Arduinos besteht aus Text. In der Automatisierungstechnik wird häufig graphisch mit Hilfe von Funktionsbausteinen programmiert, da man dann die Systemstruktur viel einfacher erkennen kann. Eine Sprache dazu ist Continuous Function Chart (CFC, auf Deutsch Signalfussplan), bei der genau angegeben werden kann, in welcher Reihenfolge die Blöcke bearbeitet werden. Daher kann man mit dieser Weiterentwicklung des Funktionsbausteinplans auch Systeme mit Rückführschleifen beschreiben. Bild 2-13 zeigt das Programm, das mit dem Programmsystem CODESYS erstellt wurde.



**Bild 2-13** Graphische Programmierung der Zwei-Hand-Steuerung

## 3 Erste Programme für den Arduino

### 3.0 Inhalt dieses Kapitels

In den Kapiteln 1 und 2 haben wir als Vorschau einige Anweisungen kennengelernt, mit denen der Arduino Steuerungsaufgaben löst. In diesem Kapitel schreiben wir vollständige Programme für den Arduino.

Dieses Kapitel beginnt mit einer kurzen Einführung in den Gebrauch der Soft- und Hardware für den Arduino. In Kapitel 3.2 nutzen wir den Arduino als einfachen Taschenrechner, um das System kennenzulernen. Es folgen etwas umfangreichere Programme, bei denen wir schrittweise weitere Möglichkeiten einsetzen.

### 3.1 Der Arduino Uno R3 und die Entwicklungsumgebung IDE

Bild 3-1 zeigt den aktuellen Arduino Uno R3. Neben dem Uno gibt es noch weitere Systeme. Sie können mit der gleichen Entwicklungsumgebung programmiert werden. Zum Teil besitzen sie erweiterte Anschlussmöglichkeiten, mehr Speicherplatz oder leistungsfähigere Prozessoren. Das R3 steht für die dritte Generation des Uno. Neben dem originalen Uno gibt es deutlich billigere Nachbauten, die bei Direktimport aus Asien für einstellige Eurobeträge erworben werden können. Sie sollen im Wesentlichen kompatibel zum Vorbild sein. Ein bekannter, möglicher Unterschied betrifft den Baustein, der die Kommunikation mit dem PC übernimmt. U. U. ist es erforderlich, noch einen Treiber im PC zu installieren. Für gewerbliche Anwendungen gibt es z. B. von der Fa. Siemens das System SIMATIC IOT2000, das die üblichen Eigenschaften industrieller Elektronik aufweist, wie z. B. eine Versorgungsspannung von 24 V.

Herzstück des Uno ist der Mikrocontroller *ATmega328P*. Wichtig sind noch der Chip für die serielle Schnittstelle, *ATmega16U2*, und die Elektronik für die Aufbereitung der Versorgungsspannung und Takterzeugung. Und natürlich die vielen Anschlüsse für analoge und digitale Signale, die sogenannten *Pins*.





**Bild 3-1** Arduino Uno R3 - aktuelle Version mit austauschbarem Prozessor

Um den Arduino in Betrieb zu nehmen, müssen wir als erstes die dazugehörige Software installieren<sup>1</sup>. In diesem Buch wird die Version 1.8.10 verwendet. Im Folgenden wird sie *Entwicklungsumgebung* oder *IDE* genannt, angelehnt an die von den Entwicklern verwendete Bezeichnung „Arduino Software (IDE)<sup>2</sup>“.

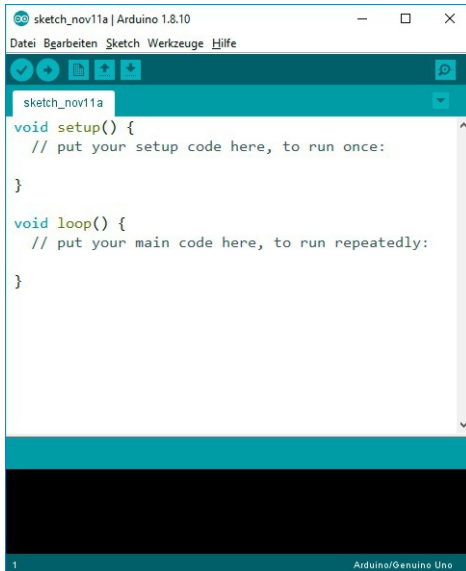
Wir installieren das Programm und verbinden danach den Arduino über die USB-Schnittstelle mit dem PC. Beim ersten Mal werden Treiber installiert. Wenn dies erfolgreich abgeschlossen ist, rufen wir die IDE auf und stellen die Schnittstelle ein. Bei meinem System ist es COM3, s. Bild 3-4. Wenn ein zweiter Arduino in einer zweiten IDE an dem PC angeschlossen ist, nimmt man die nächste Schnittstelle, bei meinem System COM4. Wir überprüfen auch, ob die Software den richtigen Typ, z. B. Uno, erkannt hat und korrigieren ggf. Am Arduino leuchtet jetzt die grüne Leuchtdiode am rechten Rand der Platine. Dies ist die Anzeige dafür, dass das Board mit Strom versorgt wird<sup>3</sup>.

---

<sup>1</sup> Sie ist in der aktuellen Version unter <https://www.arduino.cc/en/Main/Software> zu finden.

<sup>2</sup> IDE steht für Integrated Development Environment.

<sup>3</sup> Der Arduino kann auch unabhängig vom PC von einem Netzteil mit Hohlstecker über die links auf der Platine befindliche Buchse mit Strom versorgt werden; die Spannung des Netzteils muss zwischen 7 V und 12 V liegen, wobei 7 V bis 9 V empfohlen werden.



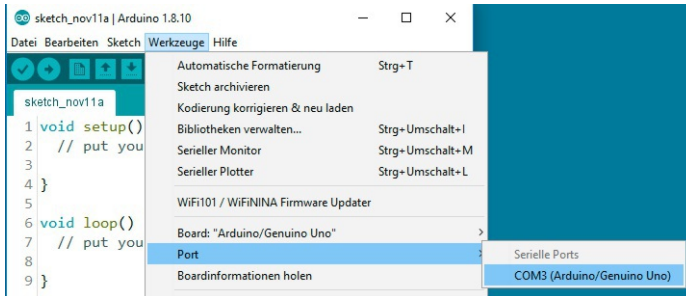
**Bild 3-2** Erster Aufruf der Entwicklungsumgebung

Häufig benötigte Funktionen besitzen eigene Schaltflächen in der Werkzeugleiste. Ein Klick auf die Taste *Hochladen* sorgt dafür, dass das aktuelle Programm gespeichert, übersetzt, in den Arduino geladen und dort gestartet wird.



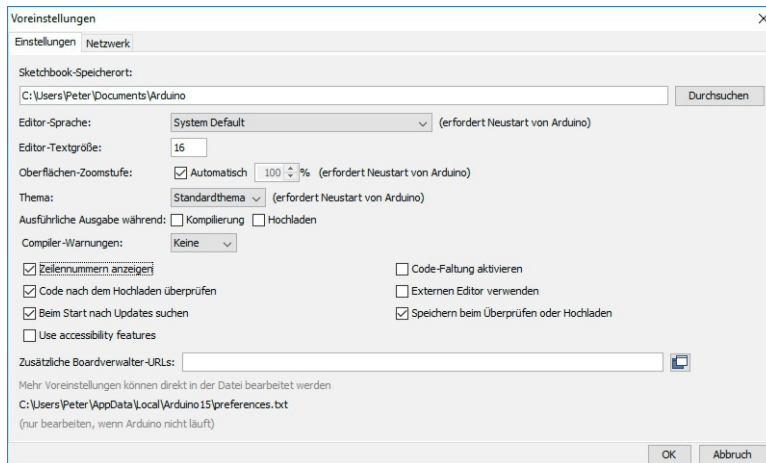
**Bild 3-3** Werkzeugleiste der Entwicklungsumgebung

Die Menüleiste führt unter Datei -> Beispiele zu einem Auswahlménü, das alle vorhandenen Beispiele auflührt. Dazu gehören sowohl Beispiele für den Arduino als auch Beispiele zu den installierten Bibliotheken. Immer dann, wenn man z. B. einen neuen Sensor ausprobieren möchte, ist es sinnvoll, die dazu passende Bibliothek zu installieren, mit deren Beispielen zu beginnen und diese dann schrittweise anzupassen.



**Bild 3-4** Einstellen der Schnittstelle; Ändern auf COM3

Die Entwicklungsumgebung kann an die eigenen Vorstellungen angepasst werden. Unter Datei/Voreinstellungen kann man z. B. eine größere Schriftgröße oder Zeilennummern anzeigen wählen, Bild 3-5.



**Bild 3-5** Voreinstellungen der Entwicklungsumgebung

Der Speicherort für die erstellten Programme steht ganz oben unter Sketchbook-Speicherort. Bei mir ist es: C:\Users\Peter\Documents\Arduino. Wenn wir ein Häkchen bei Ausführliche Ausgabe machen, erhalten wir im unteren Teil der IDE ausführliche Meldungen über die Zwischenschritte im Hintergrund. Dabei erfahren wir auch, wo temporäre Dateien abgelegt werden.

## 3.2 Der Arduino als Taschenrechner

Es ist ein bisschen exzentrisch, aber wir können den Arduino auch als simplen Rechner verwenden. Im Folgenden wollen wir die Kosten für eine einfache Laborausstattung berechnen. In Tabelle 3-1 sind die erforderlichen Teile aufgeführt.

**Tabelle 3-1** Liste der benötigten Teile

Bezeichnung	Anzahl	Stückpreis
Arduino Uno R3	5	€ 29,80
Steckbrett	5	€ 5,50
Kabelsatz	1	€ 3,90

### 3.2.1 Das erste Programm

Wir starten die Entwicklungsumgebung und geben folgende Anweisungen ein:

```
1 void setup() {
2   // Serielle Schnittstelle im Arduino konfigurieren
3   // Uebertragungsrate 9600, muss mit der Einstellung im PC uebereinstimmen
4   // 9600 Baud
5
6   Serial.begin(9600);
7   Serial.print(5 * 29.80 + 5 * 5.50 + 3.9);
8 }
9
10 void loop() {
11   // put your main code here, to run repeatedly:
12 }
13 // Prog_3_a
```

**Bild 3-6** Erstes Arduino-Programm

Dieses Programm Prog\_3\_a und die jeweils neueste Version der meisten anderen Programme in diesem Buch finden Sie unter

<https://c.gmx.net/@334699682150220409/WMwWqN5aTSexsRCKVMzDw>

Wichtig bei der Eingabe ist, dass Sie die Groß- bzw. Kleinschreibung beibehalten! Eingaben in die IDE sind „case sensitive“, d. h. es wird zwischen Groß- und Kleinschreibung unterschieden.

Ein Programm ist besser lesbar, wenn wir Anweisungen und Variablen jeweils durch ein Leerzeichen trennen.


Wie alle Programmiersysteme erfordert auch diese Entwicklungsumgebung Sorgfalt bei der Eingabe von Zahlen. Erwartet wird die im englischsprachigen Raum übliche Darstellung mit einem Dezimalpunkt, statt des bei uns üblichen Kommas. Also hier als Eingabe 29.80 statt wie bei Preisen üblich € 29,80. In der Entwicklungsumgebung sieht das dann wie folgt aus:

```
1 void setup() {
2 // Serielle Schnittstelle im Arduino konfigurieren
3 // Uebertragungsrate 9600, muss mit der Einstellung im PC uebereinstimmen
4 // 9600 Baud
5 Serial.begin(9600);
6 Serial.print(5 * 29.80 + 5 * 5.50 + 3.9);
7 }
8 void loop() {
9 // put your main code here, to run repeatedly:
10 }
11 // Prog_3_a
```

**Bild 3-7** Bildschirmausdruck der Entwicklungsumgebung

In der Entwicklungsumgebung werden Schlüsselworte wie `void`, `setup` oder `Serial` automatisch farbig markiert, um sie hervorzuheben.


Kommentare kennzeichnen wir mit zwei Schrägstrichen `//`. Alles, was danach in der Zeile steht, wird von der IDE ignoriert.

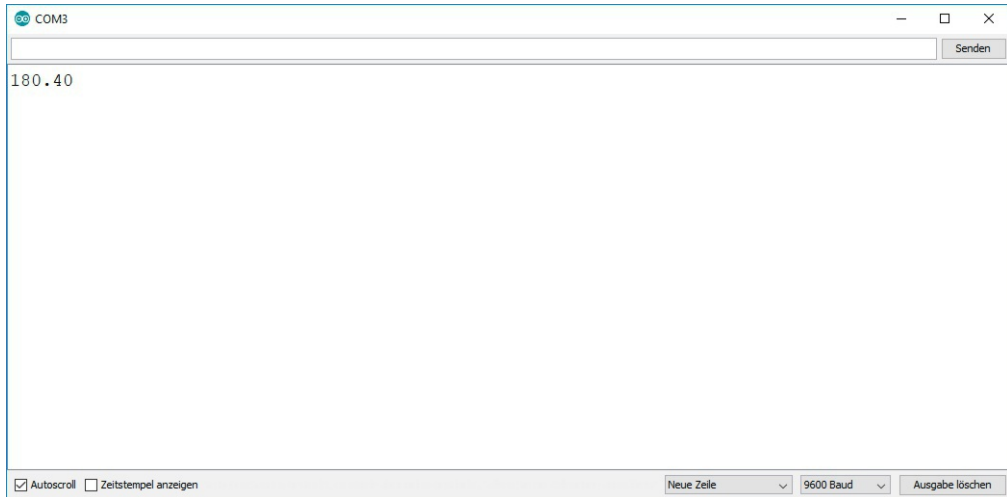
Wir drücken als Nächstes auf den zweiten Knopf in der Werkzeugleiste,  (Hochladen) und werden aufgefordert, einen Namen für unser Programm zu vergeben. Im Arduino-Slang spricht man von einem *Sketch* (Skizze, Entwurf) und nicht von einem Programm. Das Programm wird im Verzeichnis Arduino in einem neuen Verzeichnis mit dem gleichen Namen wie das Programm abgespeichert. Bei größeren Projekten werden weitere Dateien ebenfalls dort abgelegt. Als Namen vergeben wir Prog\_3.a. Die Namensweiterung .ino wird automatisch ergänzt. Programmnamen müssen mit einem Buchstaben oder einer Zahl beginnen, gefolgt von Buchstaben, Zahlen, Bindestrichen, Punkten und Unterstrichen. Die maximale Länge beträgt 63 Zeichen.

Das Programm wird übersetzt (compiliert), in den Speicher des Arduinos übertragen und dann gestartet. Auf dem Bildschirm wird im unteren, schwarzen Bereich der Entwicklungsumgebung folgende (Erfolgs-)Meldung ausgegeben:

```
Hochladen abgeschlossen
Der Sketch verwendet 1718 Bytes (5%) des Programmspeicherplatzes. Das Maximum sind 32256 Bytes.
Globale Variablen verwenden 184 Bytes (8%) des dynamischen Speichers, 1864 Bytes für lokale Variablen verbleiben. Das Maximum sind 2048 Bytes.
```

**Bild 3-8** Bildschirmausdruck des ersten Arduino-Programms

Um das Ergebnis ansehen zu können, starten wir den *Monitor*. Dieser empfängt vom Arduino Nachrichten über die USB-Schnittstelle. Ein Druck auf den ganz rechts in der Werkzeugleiste angeordneten Knopf „serieller Monitor“, , öffnet ein neues Fenster, das nach kurzer Zeit das Ergebnis der Berechnung anzeigt.



**Bild 3-9** Monitorausgabe des ersten Arduino-Programms

Die ausgegebene Zahl 180.40 ist das Rechenergebnis: Die eine einfache Laborausstattung kostet 180,40 €.

Was hat unser Programm nun gemacht? Da wir die Berechnung nur ein einziges Mal durchführen wollen, verwenden wir die Sektion `setup()`. Alles, was dann innerhalb der geschweiften Klammern folgt, `{ }`, wird einmal ausgeführt.

Als erstes initialisieren wir die Verbindung vom Arduino über dessen serielle Schnittstelle zum PC. Die Angabe (9600) definiert die Übertragungsgeschwindigkeit der Schnittstelle im Arduino. Der Vorgabewert für das PC-Monitorprogramm ist 9600, so dass wir am PC nichts umstellen müssen.

Dann „drucken“ wir über diese Schnittstelle auf den Monitor das Ergebnis der Berechnung

$(5 * 29.80 + 5 * 5.50 + 3.9)$  .

Obwohl wir den zweiten Teil eines Arduino-Programms, der mit `void loop` beginnt und der schließenden, geschweiften Klammer `}` endet, für die ersten Beispiele nicht benötigen, muss er im Programm vorhanden sein.

```
12 void loop() {  
13     // put your main code here, to run repeatedly:
```

```
14  
15 }
```

Wenn diese Zeilen fehlen, erscheint eine Fehlermeldung:

undefined reference to 'loop'

Wenn unser erstes Programm arbeitet, ist ein guter Teil des Weges bereits absolviert. Bei Automatisierungssystemen benötigt man häufig viel Arbeitszeit, bis das System zum ersten Mal funktioniert, und nicht nur Fehlermeldungen erscheinen.

### 3.2.2 Das zweite Programm

In die Sektion `setup()` schreiben wir jetzt folgende Anweisungen:

```
1 void setup() {  
2   // Serielle Schnittstelle im Arduino konfigurieren  
3   // Uebertragungsrate 9600,  
4   // muss mit der Einstellung im PC uebereinstimmen  
5   // 9600 Baud  
6   Serial.begin(9600);  
7  
8   // Variablen deklarieren und gleichzeitig Werte zuweisen  
9   int Anzahl = 5;  
10  float KostenBoard = 29.80;  
11  float KostenShield = 5.50;  
12  float KostenKabel = 3.9;  
13  float KostenGesamt;  
14  
15  KostenGesamt = Anzahl * (KostenBoard + KostenShield) + KostenKabel ;  
16  
17  Serial.print(KostenGesamt);  
18  }  
19  
20 void loop() {  
21   // put your main code here, to run repeatedly:  
21  }  
22 // Prog_3_b
```

Wir verwenden jetzt *Variablen*, um Informationen zu speichern. Die Anzahl der zu kaufenden Boards ist eine ganze Zahl. Daher verwenden wir dafür den Datentyp `int`. Da unsere Preise reelle Zahlen sind, benutzen wir den Datentyp `float`. Wir müssen Variablen mit ihrem Datentyp vor der ersten Benutzung *deklarieren*. Dabei können wir auch gleich einen Wert zuweisen; man spricht dann von *initialisieren*.

Die Gesamtkosten errechnen wir mit der Anweisung

`KostenGesamt = Anzahl * (KostenBoard + KostenShield) + KostenKabel;`

Die Gesamtkosten werden wieder mit der Anweisung `Serial.print` ausgegeben.

Bei derartigen Berechnungen hält sich der Arduino an die üblichen Rechenregeln, zum Beispiel, dass die Punktrechnung (Multiplikation) vor der Strichrechnung (Addition) ausgeführt wird. Es wird zuerst der Ausdruck rechts des *Zuweisungszeichens* = ausgewertet und dann das Ergebnis der Variablen links des = zugewiesen. Beim = handelt es sich *nicht* um ein Gleichheitszeichen im mathematischen Sinne! Andere Programmiersprachen, z. B. der Strukturierte Text für Speicherprogrammierbare Steuerungen verwenden daher := als Zuweisungszeichen, um jegliche Verwechslung auszuschließen.

Bei den Namen für Variablen sollten wir uns an einige Regeln halten. Wichtig ist ein sinnvoller Name, der auch etwas länger sein darf, um das Lesen und Verstehen des Programms zu erleichtern. Deutsche Sonderzeichen, wie ä, ö, ü oder ß sollte man beim Programmieren nicht verwenden, da die Programmierumgebungen aus englischsprachigen Ländern stammen und bei diesen Sonderzeichen häufig Probleme auftreten. Leerzeichen in Variablennamen oder Rechenzeichen sind ebenso verboten. Das erste Zeichen darf auch keine Ziffer sein.

Wenn beim Programmieren eine Zeile nicht ausreicht, können wir in der nächsten fortfahren. Die Anweisung wird erst durch ein Semikolon beendet. Man könnte also auch mehrere Anweisungen in eine Zeile schreiben. Da das Programm dadurch schlecht lesbar wird, sollte man das aber besser bleiben lassen.

### 3.2.3 Formatierte Ausgabe

Wir können die Ausgabe auch ein bisschen schöner gestalten. Dazu dient die Anweisung `Serial.println`, die einen Zeilenumbruch hinter der Ausgabe bewirkt. Texte können wir direkt in die Anweisung schreiben, wenn wir am Anfang und am Ende das Zollzeichen " verwenden: `Serial.print(" Boards, je ");` .

Bei einer Gleitkommazahl können wir die Anzahl der angezeigten Nachkommastellen vorgeben, indem wir nach dem Namen ein Komma setzen und die Anzahl anfügen. Integerzahlen können wir als Dezimalzahl, also als Zahl mit der Basis 10 ausgeben; Ergänzung DEC. Andere Basen sind ebenfalls möglich: Dualzahlen mit der Basis 2 werden mit BIN gekennzeichnet, Hexadezimalzahlen mit der Basis 16 mit HEX.

```
1 void setup() {  
2     // put your setup code here, to run once:  
3  
4     // Serielle Schnittstelle im Arduino konfigurieren  
5     // Uebertragungsrate 9600  
6     // muss mit der Einstellung im PC uebereinstimmen
```



```
7 // 9600 Baud
8 Serial.begin(9600);
9
10 // Variablen deklarieren und gleichzeitig Wert zuweisen
11 int Anzahl = 5;
12 float KostenBoard = 29.80;
13 float KostenShield = 5.50;
14 float KostenKabel = 3.9;
15 float KostenGesamt;
16
17 KostenGesamt = Anzahl * (KostenBoard + KostenShield) + KostenKabel ;
18
19 Serial.println("Kostenaufstellung");
20
21 Serial.print(Anzahl,DEC);
22 Serial.print(" Boards, je ");
23 Serial.print(KostenBoard,DEC);
24 Serial.print(" = ");
25 Serial.println(Anzahl * KostenBoard,DEC);
26
27 Serial.print(Anzahl,DEC);
28 Serial.print(" Shields, je ");
29 Serial.print(KostenShield,2); // 2 Nachkommastellen
30 Serial.print(" = ");
31 Serial.println(Anzahl * KostenShield,DEC);
32
33 Serial.print(1,DEC);
34 Serial.print(" Kabel, je ");
35 Serial.print(KostenKabel,4); // 4 Nachkommastellen
36 Serial.print(" = ");
37 Serial.println(1 * KostenKabel,DEC);
38 Serial.println("-----");
39 Serial.print(" ");
40 Serial.println(KostenGesamt,2); // 2 Nachkommastellen
41 Serial.println("=====");
42 }
43 void loop() {
44     // put your main code here, to run repeatedly:
45 }
46
47 // Prog_3_c
```

Im Monitor erhalten wir:

Kostenaufstellung

5 Boards, je 29.7999992370 = 149.0000000000

5 Shields, je 5.50 = 27.5000000000

1 Kabel, je 3.9000 = 3.9000000953

-----  
180.40  
=====

Der Arduino beherrscht die Grundrechenarten und noch einige andere Funktionen, z. B. die Quadratwurzel, trigonometrische Funktionen oder den natürlichen Logarithmus zur Basis e. Wenn wir weitere Funktionen benötigen, können wir die Bibliothek `math.h` laden. Im folgenden Programm berechnen wir aus einer gemessenen Spannung über einem Heißleiter die Temperatur. Dies benötigen wir in Kap. 7.

$$T(u) = \frac{1}{\frac{\log_e \left( \frac{R \cdot u}{R_{25} \cdot (U_{\text{ref}} - u)} \right)}{\beta_{\text{NTC}}} + \frac{1}{T_{25}}}$$

mit	T	Temperatur des Heißleiters in K
	R	Festwiderstand in der Reihenschaltung in $\Omega$ , hier 10 k $\Omega$
	u	Spannung über dem Heißleiter in V, liegt am Pin A0 an
	R <sub>25</sub>	Widerstand bei der Temperatur 25 °C in $\Omega$ , hier 10009 $\Omega$
	$\beta_{\text{NTC}}$	Kennwert in K, hier 3933 K
	T <sub>25</sub>	Bezugstemperatur 25 °C
	U <sub>ref</sub>	Spannung der Reihenschaltung, hier 5 V

```

1 void setup() {
2     // put your setup code here, to run once:
3     Serial.begin(9600);
4
5     // Variablen deklarieren und gleichzeitig Wert zuweisen
6     float T_mess;
7     const float R_Reihenwiderstand = 1e4;
8     const float U_ref = 5.0;
9     const float R_25 = 1e4;
10    const float betaNTC = 4200.0;
11    const float absNull = 273.15;
12    const float T_25 = 273.15 + 25.0;
13    const float u_NTC = 3.801;
14
15    T_mess = 1.0/( log( (R_Reihenwiderstand * u_NTC / (U_ref - u_NTC)) / R_25) / betaNTC +
16                  1.0 / T_25) - absNull;
17    Serial.print(" u_NTC = ");
18    Serial.print(u_NTC, 3); // 3 Nachkommastellen
19    Serial.println(" V ");
20    Serial.print(" T_mess = ");
21    Serial.print(T_mess,1); // 1 Nachkommastelle
22    Serial.println(" ° C ");
23 }
24 void loop() {
25     // put your main code here, to run repeatedly:
26 }
27 // Prog_3_d

```

Die serielle Schnittstelle des Arduinos ist keine Einbahnstraße. Wir können darüber auch Daten an den Arduino senden. Die entsprechende Anweisung für den Arduino, Daten von der Schnittstelle zu lesen, lautet `Serial.read()`; , s. Abs. 7.4

### 3.3 Boolesche Gleichungen

Wir wollen den Arduino in erster Linie für Automatisierungsaufgaben einsetzen. Dazu benötigen wir Boolesche Gleichungen. Das Boolesche UND steht als Anweisung `&&` ebenso zur Verfügung wie das Boolesche ODER als Anweisung `||` . Das folgende Programm zeigt einige Anwendungen.

```
1 void setup() {
2     // put your setup code here, to run once:
3
4     // Serielle Schnittstelle im Arduino konfigurieren
5     // Uebertragungsrate 9600
6     // muss mit der Einstellung im PC uebereinstimmen
7     // 9600 Baud
8     Serial.begin(9600);
9
10    // Variablen deklarieren und gleichzeitig Wert zuweisen
11    bool TasterLinks = true;
12    bool TasterRechts = false;
13    bool VerknuepfungUND;
14    bool VerknuepfungODER;
15    bool VerknuepfungXOR;
16    bool VerknuepfungOhneKlammern;
17
18    VerknuepfungUND = TasterLinks && TasterRechts;
19    VerknuepfungODER = TasterLinks || TasterRechts;
20    VerknuepfungXOR = (TasterLinks && (! TasterRechts) ) ||
21                      ( TasterRechts && (! TasterLinks) );
22    VerknuepfungOhneKlammern = TasterLinks && ! TasterRechts || ! TasterLinks &&
23                               TasterRechts;
24
25    Serial.println("Boolesche Algebra");
26
27    Serial.print(" TasterLinks = ");
28    Serial.println(TasterLinks);
29
30    Serial.print(" TasterRechts = ");
31    Serial.println(TasterRechts);
32
33    Serial.print(" TasterRechts UND TasterLinks = ");
34    Serial.println(VerknuepfungUND);
35
36    Serial.print(" TasterRechts ODER TasterLinks = ");
37    Serial.println(VerknuepfungODER);
38
39    Serial.print(" TasterRechts XOR TasterLinks = ");
40    Serial.println(VerknuepfungXOR);
41
42    Serial.print(" Taster XOR verknuepft ohne Klammern = ");
43    Serial.println(VerknuepfungOhneKlammern);
44
45 }
46
47
```

```
48 void loop() {  
49   // put your main code here, to run repeatedly:  
50 }  
51 // Prog_3_e
```

Im Monitor werden Boolesche Variablen mit 0 für false bzw. 1 für true ausgegeben. Wir erhalten für unser Beispiel:

```
Boolesche Algebra  
TasterLinks = 1  
TasterRechts = 0  
TasterRechts UND TasterLinks = 0  
TasterRechts ODER TasterLinks = 1  
TasterRechts XOR TasterLinks = 1  
Taster XOR verknuepft ohne Klammern = 1
```

Die Zeilen mit VerknuepfungUND und VerknuepfungODER sind nicht überraschend, da die jeweiligen Verknüpfungen im Befehlssatz des Arduinos enthalten sind. Für die Exklusiv-ODER-Verknüpfung gibt es im Programm zwei Versionen. Die erste verwendet runde Klammern, ( ), um die Reihenfolge der Abarbeitung festzulegen. Sie lässt sich einfacher verstehen, da beim Lesen die Rangfolge der einzelnen Operatoren keine Rolle spielt. In der zweiten Version muss man hingegen wissen, dass die Negation, !, vor einer UND-Verknüpfung, && , und diese vor einer ODER-Verknüpfung, || , ausgewertet wird. Generell ist es gut, das Programm so zu schreiben, dass es möglichst einfach zu lesen und zu verstehen ist.

### 3.4 Variablen und ihre Eigenschaften

Um den Preis unserer Laborausstattung zu berechnen, haben wir Variablen verwendet und mit diesen dann gerechnet.

```
void setup() {  
  // Variablen deklarieren und gleichzeitig Wert zuweisen  
  int Anzahl = 5;  
  float KostenBoard = 29.80;  
  float KostenShield = 5.50;  
  float KostenKabel = 3.9;  
  float KostenGesamt;  
  KostenGesamt = Anzahl * (KostenBoard + KostenShield) + KostenKabel ;  
}
```

Je nach Aufgabe, benötigen wir verschiedene Arten von Variablen. In diesem Buch werden wir folgende Datentypen verwenden:

**Tabelle 3.2** Datentypen

Datentyp	Größe in Bytes	Zahlenbereich	Kommentar
bool	1	true oder false	Boolesche Variable
byte	1	0 bis 255	positive Zahl
int	2	-32.768 bis 32.767	ganze Zahl
long	4	-2.147.483.648 bis 2.147.483.647	ganze Zahl
float	4	$\pm 3,4 \cdot 10^{38}$	Gleitkommazahl, ca. 7 Kommastellen Genauigkeit

Die Größe in Bytes gibt an, wie groß der Speicherplatz ist, den der Arduino für diese Variable zur Verfügung stellen muss. Ein Byte besteht aus 8 Bits und ist die kleinste Einheit, die im Speicher adressiert werden kann. Wenn unsere ganze Zahl nie negative Werte annehmen kann, verwenden wir das Schlüsselwort `unsigned` und vergrößern damit den darstellbaren Bereich auf fast das Doppelte. So verwendet die Systemfunktion `millis()` den Datentyp `unsigned long`, da die Anzahl der seit dem Systemstart vergangenen Millisekunden immer positiv ist, aber sehr groß werden kann.

Da der Speicherplatz bei einem Mikrocontroller sehr begrenzt ist, verwenden wir in der Regel Variablen mit kurzer Lebensdauer. Sie existieren nur während der Programmausführung in dem *Anweisungsblock*, in dem sie deklariert wurden. Im Beispiel oben ist das der Block `void setup() { }`. Wenn wir die Variable `KostenGesamt` davor oder danach verwenden, erhalten wir eine Fehlermeldung.

Programmtechnisch wird das so gelöst, dass ein Teil des Hardware-Speichers als Merkzettel für Variablen benutzt wird, der nach Bearbeitung des Anweisungsblocks überschrieben wird, der sog. *Stack*. Diese Variablen nennt man auch *lokale* Variablen. Sie werden vom Programmiersystem *nicht* zu Beginn mit 0 initialisiert; ihre Werte sind *völlig zufällig*, bis wir ihnen im Programm Werte zuweisen. Nach dem Compilieren zeigt die IDE im unteren, schwarzen Ausgabefeld an, wie viele Bytes noch zur Verfügung stehen und warnt, wenn der Platz knapp wird.

Es gibt aber Werte, auf die wir bei der weiteren Programmausführung noch zurückgreifen müssen. Mit dem Schlüsselwort `static` sorgen wir dafür, dass diese Variable an einer festen Speicherstelle im Hardwarespeicher abgelegt werden und auch nach Verlassen des Anweisungsblocks noch zur Verfügung stehen. Eine Besonderheit dieser statischen Variablen ist, dass sie vor dem Programmstart vom Programmiersystem mit 0 initialisiert werden.

Eine Variable ist immer in dem Anweisungsblock sichtbar, in dem sie deklariert wurde. Wenn wir Blöcke ineinander schachteln, ist die äußere Deklaration auch in den inneren Anweisungsblöcken sichtbar.

```
1 void setup() {
2   // put your setup code here, to run once:
3   // Serielle Schnittstelle im Arduino konfigurieren
4   // Uebertragungsrate 9600 muss mit der Einstellung
5   // im PC uebereinstimmen, 9600 Baud
6   Serial.begin(9600);
7
8   // Variablen deklarieren und gleichzeitig Wert zuweisen
9   int VarX = 5; // Initialisierung mit 5
10  Serial.print("Block 1: VarX = ");
11  Serial.println(VarX,DEC);
12
13  { int VarY; // keine Initialisierung
14    Serial.print("Block 2: VarX = ");
15    Serial.println(VarX,DEC);
16    Serial.print("Block 2: VarY = ");
17    Serial.println(VarY,DEC);
18
19    { int VarZ = 15; // Initialisierung mit 5
20      VarX = 7; // Zuweisung
21      VarY = 12; // Zuweisung
22      Serial.print("Block 3: VarX = ");
23      Serial.println(VarX,DEC);
24      Serial.print("Block 3: VarY = ");
25      Serial.println(VarY,DEC);
26      Serial.print("Block 3: VarZ = ");
27      Serial.println(VarZ,DEC);
28    }
29
30    Serial.print("Block 2 Fortsetzung: VarX = ");
31    Serial.println(VarX,DEC);
32    Serial.print("Block 2 Fortsetzung: VarY = ");
33    Serial.println(VarY,DEC);
34  }
35 }
36 void loop() {
37   // put your main code here, to run repeatedly:
38 }
39 // Prog_3_f
```

Wir erhalten folgende Ausgabe im Monitor:

Block 1: VarX = 5

Block 2: VarX = 5

Block 2: VarY = 0

Block 3: VarX = 7

Block 3: VarY = 12

Block 3: VarZ = 15

Block 2 Fortsetzung: VarX = 7

Block 2 Fortsetzung: VarY = 12

Neben den lokalen Variablen gibt es auch noch *globale* Variablen. Beim Arduino erzeugen wir diese globalen Variablen, indem wir die Deklaration ganz an den Anfang des Programms stellen. Diese globalen Variablen bleiben bis zum Abschalten des Arduinos erhalten. Sie werden vom Programmiersystem vor dem Programmstart auf 0 initialisiert. Für kurze, übersichtliche Programme kann man globale Variablen verwenden. Bei umfangreichen Aufgaben und entsprechend langen Programmen wird es aber schwierig, über alle globalen Variablen den Überblick zu behalten. Und da diese globalen Variablen in jedem Anweisungsblock geändert werden können, wird die Fehlersuche oder eine spätere Programmänderung mühsam. Es ist daher gute Praxis, globale Variablen nur für die Definition von Konstanten zu verwenden. Diese können im folgenden Programm nicht mehr geändert werden. Diese Deklaration erfolgt mit dem Schlüsselwort `const`.

Es ist möglich, einen Variablennamen in mehreren Deklarationen zu verwenden. Sogar die Deklaration lokaler und globaler Variablen mit identischen Namen ist erlaubt. Das Programmiersystem wird sich auch an die dafür geltenden Regeln halten. Für Programmierer ist das schon schwieriger, da es hohe Aufmerksamkeit erfordert. Für den (flüchtigen) Leser des Programms führt ein derartiger Programmierstil zum Chaos und macht das Programm fast nicht wartbar.

Es gibt leistungsfähigere Arduinoboards als den Uno. Diese unterstützen auch noch weitere Datentypen, z. B. längere Gleitkommazahlen. In der Programmiersprache C/C++, die von der Entwicklungsumgebung genutzt wird, gibt es darüber hinaus noch weitere Datentypen, z. B. `char` für Zeichen.



## 3.5 Funktionen

Beim Erstellen eines etwas größeren Programms ist es wichtig, den Überblick zu behalten. Daher ist es eine gute Praxis, einzelne Programmteile kurz zu halten. Eine Möglichkeit dazu bieten *Funktionen*, die in einem abgeschlossenen Programmteil Teilaufgaben bearbeiten.

Wenn wir die Exklusiv-ODER-Verknüpfung häufiger benötigen, können wir sie einmal als Funktion definieren und dann beliebig oft in unserem Programm darauf zurückgreifen.

Wir hatten oben folgende Anweisung:

```
VerknuepfungXOR = (TasterLinks && (! TasterRechts) ) || ( TasterRechts && (! TasterLinks) );
```

Etwas allgemeiner können wir schreiben:

```
A = (E1 && (! E2) ) || ( E2 && (! E1) );
```

Wir haben also eine Zuweisung mit den zwei Variablen E1 und E2 und dem Ergebnis A. Dafür definieren wir eine Funktion XOR. Diese besitzt ein Ausgangsargument vom Typ `bool` und zwei Eingangsargumente, ebenfalls vom Typ `bool`. Diese Information schreiben wir in die erste Zeile. Es folgt die Definition aller in der Funktion benötigten Variablen. Wir brauchen nur A, ebenfalls vom Typ `bool`. Dann schließen sich die Berechnungen an. In unserem Fall ist das nur eine Zeile. Die Anweisung

```
return A;
```

bewirkt zweierlei. Zum einen beendet sie alle Berechnungen in dieser Funktion. Sollten danach noch weitere Anweisung stehen, werden diese nicht ausgeführt. Zum anderen weist sie dem Ausgangsargument den Wert von A zu.

Damit die Funktion beim Lesen des Programms gefunden wird, schreiben wir sie vor den Block `void setup() { }`.

```
3 bool XOR (bool E1, bool E2) {  
4     bool A;  
5     A = (E1 && (! E2) ) || ( E2 && (! E1) );  
6     return A;  
7 }
```

Ein Anwendungsprogramm für unsere Funktion XOR könnte wie folgt aussehen:

```
1 const unsigned int Baud_Rate = 9600;  
2  
3 bool XOR (bool E1, bool E2) {  
4     bool A;  
5     A = (E1 && (! E2) ) || ( E2 && (! E1) );  
6     return A;
```

```
7  }
8
9  void setup() {
10     // put your setup code here, to run once:
11
12     // Serielle Schnittstelle im Arduino konfigurieren
13     // Uebertragungsrate 9600 muss mit der Einstellung
14     // im PC uebereinstimmen
15     // 9600 Baud
16     Serial.begin(Baud_Rate);
17
18     // Variablen deklarieren und gleichzeitig Wert zuweisen
19     bool  TasterLinks = true;
20     bool  TasterRechts = false;
21     bool  VerknuepfungXOR;
22
23     VerknuepfungXOR = XOR (TasterLinks, TasterRechts);
24
25     Serial.println(" Boolesche Algebra");
26
27     Serial.print(" TasterLinks  = ");
28     Serial.println(TasterLinks);
29
30     Serial.print(" TasterRechts = ");
31     Serial.println(TasterRechts);
32
33     Serial.print(" TasterRechts XOR TasterLinks = ");
34     Serial.println(VerknuepfungXOR);
35
36 }
37
38 void loop() {
39     // put your main code here, to run repeatedly:
40 }
41 // Prog_3_g
```

Als Ergebnis erhalten wir im Monitor

Boolesche Algebra

TasterLinks = 1

TasterRechts = 0

TasterRechts XOR TasterLinks = 1

Es gilt unter Programmierern als schlechter Stil, in einem Programm Zahlenwerte für Parameter zu verwenden. Wir hatten dies bisher getan, und die serielle Schnittstelle mit `Serial.begin(9600);` initialisiert. In einem großen Programm gibt es u. U. viele Parameter an vielen verschiedenen Stellen, so dass man bei einer Änderung fast mit Sicherheit eine Stelle übersieht und das System nicht mehr korrekt arbeitet. Besser ist es, derartige Konstanten am Programmanfang als globale Variablen zu definieren und sie auch mit dem Schlüsselwort `const` als unveränderlich zu kennzeichnen. Dann reicht ein Blick auf den Anfang des Programms, um alle Parameter zu sehen. Und auch nur an dieser Stelle müssen sie ggf. geändert werden. Für unsere serielle Schnittstelle können wir den Parameter, die Übertragungsrate, wie folgt festlegen:

```
1  const unsigned int Baud_Rate = 9600;
16 Serial.begin(Baud_Rate);
```

### 3.6 Exkurs zu den Anweisungen `&` und `|`

Der Arduino versteht neben der Anweisung `&&` für das Boolesche UND auch die Anweisung `&`. Hierbei handelt es sich um eine *bitweise* Operation.

Als Beispiel nehmen wir als erstes die Dezimalzahl 58, die in binärer Darstellung 0011 1010 lautet<sup>4</sup>, und wählen den Datentyp `byte`. Als zweite Zahl nehmen wir  $19_{\text{dec}} = 0001\ 0011_{\text{bin}}$  und verknüpfen die beiden Zahlen bitweise mit UND. Einen entsprechenden Befehl gibt es auch für die bitweise Verknüpfung mit ODER, `|`.

```
1  const unsigned int Baud_Rate = 9600;
2
3  void setup() {
4      // put your setup code here, to run once:
5
6      // Serielle Schnittstelle im Arduino konfigurieren
7      // Uebertragungsrate 9600
8      // muss mit der Einstellung im PC uebereinstimmen
9      // 9600 Baud
10     Serial.begin(Baud_Rate);
11
12     // Variablen deklarieren und ggf. Wert zuweisen
13     byte ersteZahl = 58;
14     byte zweiteZahl = 19;
```

---

<sup>4</sup> Mehr zu Zahlen und ihren Darstellungen steht im Kap. 10.

```
15 byte ErgebnisUND;
16 byte ErgebnisODER;
17 byte ErgebnisXOR;
18
19 ErgebnisUND = ersteZahl & zweiteZahl;
20 ErgebnisODER = ersteZahl | zweiteZahl;
21 ErgebnisXOR = ersteZahl ^ zweiteZahl;
22
23 Serial.println("Bitweise Verknüpfung ");
24
25 Serial.print("ersteZahl = ");
26 Serial.print(ersteZahl,DEC);
27 Serial.print(", binaer ersteZahl = ");
28 Serial.println(ersteZahl,BIN);
29
30 Serial.print("zweiteZahl = ");
31 Serial.print(zweiteZahl,DEC);
32 Serial.print(", binaer zweiteZahl = ");
33 Serial.println(zweiteZahl,BIN);
34
35 Serial.print("ErgebnisUND = ");
36 Serial.print(ErgebnisUND,DEC);
37 Serial.print(", binaer ErgebnisUND = ");
38 Serial.println(ErgebnisUND,BIN);
39
40 Serial.print("ErgebnisODER = ");
41 Serial.print(ErgebnisODER,DEC);
42 Serial.print(", binaer ErgebnisODER = ");
43 Serial.println(ErgebnisODER,BIN);
44
45
46 Serial.print("ErgebnisXOR = ");
47 Serial.print(ErgebnisXOR,DEC);
48 Serial.print(", binaer ErgebnisXOR = ");
49 Serial.println(ErgebnisXOR,BIN);
50 }
51
52 void loop() {
53     // put your main code here, to run repeatedly:
54 }
55 // Prog_3_h
```

Als Ergebnis sehen wir im Monitor

Bitweise Verknüpfung

ersteZahl = 58, binaer ersteZahl = 111010

zweiteZahl = 19, binaer zweiteZahl = 10011

ErgebnisUND = 18, binaer ErgebnisUND = 10010

ErgebnisODER = 59, binaer ErgebnisODER = 111011

ErgebnisXOR = 41, binaer ErgebnisXOR = 101001

Es wird das erste Bit der ersten Zahl mit dem ersten Bit der zweiten Zahl mit einem logischen UND verknüpft und dem ersten Bit des Ergebnisses zugewiesen. Dies wiederholt sich für die nächsten 7 Bits.

Führende nullen werden auf dem Monitor nicht ausgegeben. Im Programm wurden in die print-Anweisungen Leerzeichen eingefügt, um eine schöne Tabelle zu erhalten.

Eine Anwendung für die Anweisung & zeigt Abs. 9.4. Dabei wird eine 12-Bit-Zahl in eine 8-Bit-Zahl und eine 4-Bit-Zahl zerlegt, da der verwendete Bus nur 8-Bit-Zahlen übertragen kann.

Bei der Anweisung & ist der Wert der Variablen logisch 1, wenn mindestens ein Bit gesetzt ist. Beim Datentyp bool führt dies nicht zu Unklarheiten, da es sich nur um ein Bit handelt. Beim Datentyp byte ist das Ergebnis der Verknüpfung also nur dann 0, wenn in mindestens einem Operanden alle Bits 0 sind.

Die folgenden Zeilen

```
Serial.print("Bool &&: ");  
Serial.println(B1100 && B0111,BIN);  
Serial.print("Bool &: ");  
Serial.println(B1100 & B0111,BIN);
```

führen zu dieser Ausgabe auf dem Monitor:

Bool &&: 1

Bool &: 100

# A1 Programmieranweisungen

## A1.1 Datentypen und Variablen

**Tabelle A1.1** Datentypen

Datentyp	Größe in Byte	Zahlenbereich	Kommentar
bool	1	true oder false	Boolesche Variable
byte	1	0 bis 255	positive Zahl
int	2	-32.768 bis 32.767	ganze Zahl
long	4	-2.147.483.648 bis 2.147.483.647	ganze Zahl
float	4	$\pm 3,4 \cdot 10^{38}$	Gleitkommazahl, ca. 7 Kommastellen Genauigkeit

**Tabelle A1.2** Zusatz bei der Deklaration von Variablen

Schlüsselwort	Kommentar
unsigned	positive Zahl
static	permanent gespeicherte Variable
volatile	durch Interrupts geänderte Variable

## A1.2 Verknüpfungen

**Tabelle A1.3** Verknüpfungen

Operation	Kommentar
=	Zuweisung; rechte Seite des Zuweisungszeichens = auswerten und der linken Seite zuweisen
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo, d. h. Divisionsrest; $13 \% 5 = 3$

## A1.3 Funktionen

Die Syntax lautet:

```
DatentypRückgabe Funktionsname (Argumente) {  
    ... ausführbare Anweisungen ...  
    return Rückgabewert;  
}
```

Beispiel:

```
bool XOR (bool E1, bool E2) {  
    bool A;  
    A = (E1 && (! E2) ) || ( E2 && (! E1) );  
    return A;  
}
```

`void`      dieses Schlüsselwort gibt an, dass keine Daten übergeben werden.

## A1.4 if-Anweisung

Die Syntax lautet:

```
if(«Bedingung1»)
    {Anweisungsblock1}
else if («Bedingung2»)
    {Anweisungsblock2}
else if («Bedingung3»)
    {Anweisungsblock3}
else
    {Anweisungsblock4}
```

Für die Bedingungen können wir die Operatoren aus Tabelle A1.4 verwenden. Die Reihenfolge der Bedingungen ist gleichzeitig eine Rangfolge. Die erste Bedingung wird immer geprüft. Wenn sie erfüllt ist, wird der erste Anweisungsblock ausgeführt und dann die Programmbearbeitung nach dem letzten Anweisungsblock fortgesetzt. Die zweite Bedingung wird nur geprüft, wenn die erste nicht erfüllt war. Falls *keine* Bedingung erfüllt ist, wird der Anweisungsblock nach else ausgeführt. Dieser else-Zweig muss nicht vorhanden sein.

**Tabelle A1.4** Vergleichsoperatoren

Operator	Beschreibung
<code>x == y</code>	x ist gleich y
<code>x != y</code>	x ist ungleich y
<code>x &lt; y</code>	x ist kleiner als y
<code>x &gt; y</code>	x ist größer als y
<code>x &lt;= y</code>	x ist kleiner oder gleich y
<code>x &gt;= y</code>	x ist größer oder gleich y

In Abs. 4.2 verwenden wir die if-Anweisung.

```
27  Zaehler = Zaehler + 1;
28  if (Zaehler > 300)
29      {                                // neuer Durchlauf
30          Zaehler = 0;                // alles aus
31          analogWrite(ledRotPin, 0);
32          analogWrite(ledGelbPin, 0);
33          analogWrite(ledGruenPin, 0);
34      }
```



## A1.5 for-Anweisung

Die Syntax lautet:

```
for(Initialisierung; Bedingung; Update) {  
  Anweisungsblock1  
}
```

Beispiel:

```
for (int iMessen = 1; iMessen <= 4; iMessen = iMessen + 1) {  
  Temperatur_ist = Temperatur_ist + analogRead(AnalogEinPin2);  
                                     // jetzt von 0 bis 2047, 11 Bit, vier Werte  
  delay(25);                         // 25 ms pro Messung; 100 ms pro Schritt  
}
```

## A1.6 while-Anweisung

Die Syntax lautet:

```
while(«Bedingung») {  
  Anweisungsblock  
}
```

Der Anweisungsblock wird ausgeführt, wenn die Bedingung wahr ist. Unter Umständen wird der Anweisungsblock nie ausgeführt, da die Bedingung nie wahr wird.

Beispiel:

```
while (Serial.available() > 0) // Zeichen einzeln lesen und auf LCD schreiben  
{  
  lcd.write(Serial.read());  
}
```

## A1.7 switch-Anweisung

Die Syntax lautet:

```
switch( Ganzzahlige Variable ){
    case Ganzzahl: { Anweisungsblock}
        break;
    case Ganzzahl: { Anweisungsblock}
        break;
    ...           // ggf. weitere Fälle
    default: { Anweisungsblock}
}
```

Beispiel:

```
static int Schritt = 1 ; // Setzen Anfangsschritt
switch (Schritt) {
    case 1:    // Anfangsschritt 1
        digitalWrite(ledRotPin, LOW);    // Rot aus
        if (! digitalRead(Taster_1)) {    // Eingang lesen
            Schritt = 2;
        }
        break;    // Ende Schritt 1
    case 2:    // Schritt 2
        digitalWrite(ledRotPin, HIGH);    // Rot ein
        if (! digitalRead(Taster_2)) {    // Eingang lesen
            Schritt = 1;
        }
        break;    // Ende Schritt 2
}
```

## A1.8 Übertragung mit I<sup>2</sup>C-Bus und wire

**Tabelle A1.5** Schlüsselworte der seriellen Übertragung

Schlüsselwort	Kommentar
Wire.begin(eigene Adresse)	in setup(), ohne Adresse arbeitet der Arduino als Master
Wire.beginTransmission(HW-Adresse)	Start, öffnet HW-Adresse schreibend
Wire.write(Byte)	Schreibt Byte
Wire.endTransmission()	Stop
Wire.available()	Gibt HIGH zurück, wenn Bytes empfangen wurden

Wire.read()	Gibt das empfangene Byte zurück
Wire.requestFrom(HWAdresse, Anzahl)	Start, öffnet HW Adresse, Anzahl der angeforderten Bytes

## A1.9 Serielle Übertragung mit der USB-PC Schnittstelle und Serial

**Tabelle A1.6** Schlüsselworte der seriellen Übertragung

Schlüsselwort	Kommentar
Serial.begin(baudrate)	Initialisieren der seriellen Schnittstelle, baudrate hat 9600 als Vorgabe, Maximalwert ist 115200
Serial.print(Daten);	Ausgabe von Daten
Serial.print(Daten, Kodierung);	Ausgabe von Daten mit Formatierung
Serial.println( )	wie Serial.print, jedoch mit Zeilenvorschub danach
Serial.available()	liefert Anzahl der verfügbaren ungelesenen Bytes
int Serial.read()	ein Byte lesen
Serial.flush()	Lesespeicher löschen
Serial.end()	Beenden der seriellen Übertragung; D0 und D1 werden freigegeben

**Tabelle A1.7** Kodierungsmöglichkeiten der seriellen Übertragung

Schlüsselwort	Kommentar
DEC	dezimal, oder Angabe der Nachkommastellen, Vorgabe sind 2
HEX	hexadezimal
OCT	oktal
BIN	binär
BYTE	ASCII

## A1.10 Konfiguration der Pins

**Tabelle A1.8** Schlüsselworte der Pins

Schlüsselwort	Kommentar
pinMode(pin, INPUT)	Initialisieren der Digitalpins als Eingang; $0 \leq \text{pin} \leq 13$
pinMode(pin, OUTPUT)	Initialisieren der Digitalpins als Ausgang; $0 \leq \text{pin} \leq 13$
pinMode(pin, INPUT_PULLUP)	Initialisieren der Digitalpins als Eingang mit Pullup-Widerstand
digitalRead(pin)	Digitalpin pin einlesen; 5 V entsprechen HIGH
analogRead(pin)	Analogpin pin einlesen; 5 V entsprechen 1023, A0 bis A5
digitalWrite(pin, Wert)	Ausgabe der Booleschen Variablen Wert, HIGH entspricht 5 V
analogWrite(pin, Wert)	Ausgabe der ganzzahligen Variablen Wert mittels PWM-Modulation; 0 bedeutet immer aus, 255 bedeutet immer an

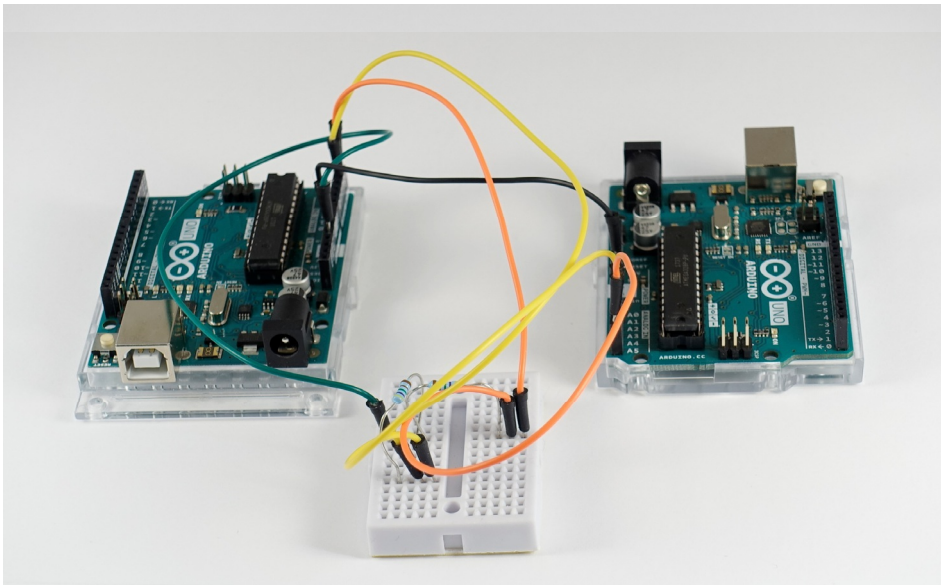
---

Die serielle Schnittstelle verwendet die Pins D0 und D1. Diese Pins können wir daher nicht einsetzen, wenn wir z. B. für den Monitor die serielle Schnittstelle initialisiert haben.

## A2 Mechanischer Aufbau

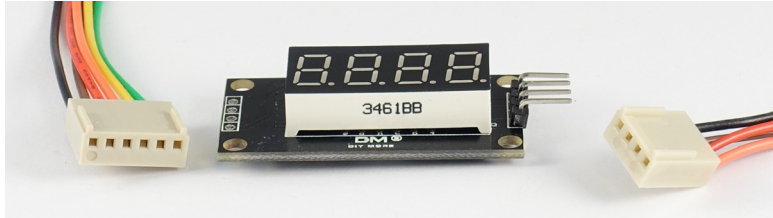
Wenn wir den Arduino nur zum Rechnen verwenden, können wir ihn auf den Schreibtisch legen und über ein USB-Kabel mit dem PC verbinden. Der originale Arduino wird mit einem Halter geliefert, so dass auch bei elektrisch leitenden Unterlagen keine Gefahr besteht. Ansonsten gibt es einfache Gehäuse aus Plexiglas, die den Arduino schützen, s. Bild A2-1.

Die einfachste Art, Signale von außen dem Arduino zuzuführen, besteht in Jumper-Kabeln und Steckbrett. Der Aufbau geht schnell, ist allerdings recht unübersichtlich, wenn die Zahl der Komponenten steigt. Bewährt hat sich auch farbiger Schalt draht mit 0,20 mm<sup>2</sup> Querschnitt. Bild A2-1 zeigt diese Möglichkeiten. Verbunden sind jeweils die Anschlüsse des I<sup>2</sup>C-Busses. Als Werkzeug empfiehlt sich ein kleiner Seitenschneider und eine Abisolierzange. Wenn man an den Widerständen rechte Winkel statt Rundbögen möchte, hilft eine Abbiegevorrichtung für Widerstände.



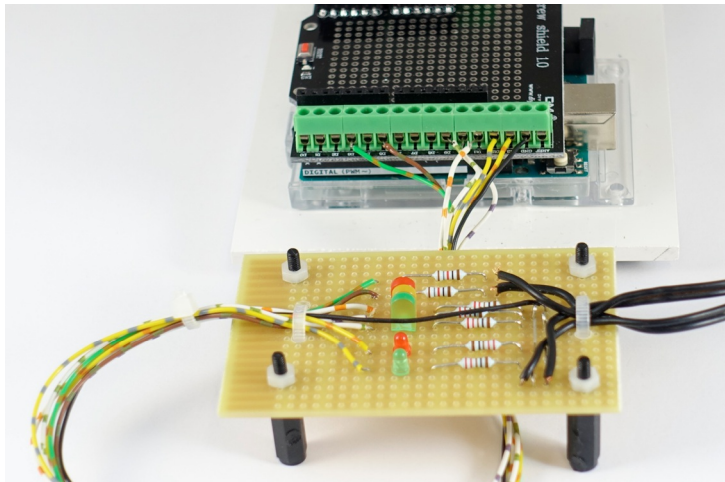
**Bild A2-1** Steckbrett und Kabel für den Arduino

Mechanisch robuster werden die Aufbauten, wenn man ein Prototypen-Shield mit Schraubklemmen verwendet. Zusammen mit einer 4-poligen Buchsenleiste mit Kabel kann man zusätzliche Baugruppen mit I<sup>2</sup>C-Bus schnell und sicher anschließen. Buchsenleisten mit Kabel gibt es auch für 6-polige Verbindungen, die z. B. beim SPI-Interface Verwendung finden.



**Bild A2-2** Shield mit Stiften und Buchsenleisten mit Kabel

Bei größeren Aufbauten führt an Lötverbindungen kein Weg vorbei. Bewährt hat sich, die Komponenten auf einer üblichen Lochstreifenrasterplatine aufzubauen und per Kabel an ein Schraubklemmen-Shield anzuschließen, s. Bild A2-3.



**Bild A2-3** Schraubklemmen-Shield und Aufbau auf Lochstreifenrasterplatine

Wenn es eine überschaubare Anzahl an Komponenten ist, kann man sie auch auf ein Prototypen-Shield löten, dass über Stifte direkt auf den Arduino aufgesetzt wird, siehe Bild 4-7. Handelsübliche Experimentierplatinen mit einem Lochabstand von ca. 2,54 mm können leider *nicht* verwendet werden, da der Abstand zwischen Pin D6 und Pin D7 ca. 3,8 mm beträgt.

## A3 ASCII-Tabelle

Die folgende Tabelle enthält ASCII Codes (American Standard Code for Information Interchange). Der ASCII-Code wurde ab den 1960er Jahren für Fernschreiber entwickelt und enthält daher auch Steuerzeichen, wie z. B. für den Wagenrücklauf (CR für carriage return) oder Zeilenvorschub (LF für line feed). Diese zwei Steuerzeichen werden automatisch als ASCII-Zeichen 13 und 10, am Ende einer Zeile eingefügt, wenn wir `Serial.println` verwenden.

Dez	ASCII	Dez	ASCII	Dez	ASCII	Dez	ASCII	Dez	ASCII
0	NUL	26	SUB	52	4	78	N	102	f
1	SOH	27	ESC	53	5	79	O	103	g
2	STX	28	FS	54	6	80	P	104	h
3	ETX	29	GS	55	7	81	Q	105	i
4	EOT	30	RS	56	8	82	R	106	j
5	ENQ	31	US	57	9	83	S	107	k
6	ACK	32	SP	58	:	84	T	108	l
7	BEL	33	!	59	;	85	U	109	m
8	BS	34	"	60	<	86	V	110	n
9	HT	35	#	61	=	87	W	111	o
10	LF	36	\$	62	>	88	X	112	p
11	VT	37	%	63	?	89	Y	113	q
12	FF	38	&	64	@	90	Z	114	r
13	CR	39	'	65	A	91	[	115	s
14	SO	40	(	66	B	92	\	116	t
15	SI	41	)	67	C	93	]	117	u
16	DLE	42	*	68	D	94	^	118	v
17	DC1	43	+	69	E	95	_	119	w
18	DC2	44	,	70	F	96	`	120	x
19	DC3	45	-	71	G	97	a	121	y
20	DC4	46	.	72	H	98	b	122	z
21	NAK	47	/	73	I	99	c	123	{
22	SYN	48	0	74	J	100	d	124	
23	ETB	49	1	75	K	101	e	125	}
24	CAN	50	2	76	L	102	f	126	~
25	EM	51	3	77	M	103	g	127	DEL

## A4 LCD-Display 16x2 mit Treiber HD44780

Um ein Display nutzen zu können, muss die entsprechende Bibliothek der IDE bekannt sein. Wenn ein Display mit I<sup>2</sup>C-Busanschluss verwendet wird, muss eine passende Bibliothek unter Sketch/Bibliothek einbinden/.ZIP Bibliothek hinzufügen eingebunden sein. Für dieses Buch habe ich die Bibliothek Newliquidcrystal\_1.3.5.zip verwendet; zusätzlich die Standardbibliothek wire.

Im Programm wird das Display am Anfang deklariert:

```
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
```

Die Adresse hängt vom Display ab. Es gibt Exemplare, bei denen die Adresse vom Nutzer mit Hilfe von Lötbrücken geändert werden kann. Dann kann man auch mehr als ein Display an einem Arduino betreiben.

Die folgenden Steuersequenzen für die Darstellung von Sonderzeichen arbeiten bei manchen HD44780-kompatiblen-LCD-Displays. Da es viele Versionen dieser Anzeige gibt, hilft im Einzelfall nur Ausprobieren oder ggf. Suchen im Datenblatt. Beispiel:

```
lcd.print("f\365r den Arduino"); //wird dargestellt als "für den Arduino"
```

**Tabelle A3.1** Darstellung von Sonderzeichen

Sequenz	Zeichen	Sequenz	Zeichen	Sequenz	Zeichen
\40	!	\74	<	\343	ε
\41	“	\75	=	\344	μ
\42	\$	\76	>	\350	√
\44	%	\77	?	\351	Hoch minus 1
\50	(	\100	@	\353	Hoch x
\51	)	\174		\357	ö
\52	*	\176	→	\363	∞
\53	+	\177	←	\364	Ω
\54	,	\260	-	\365	ü
\55	-	\333	Kastenrahmen	\366	Σ
\56	.	\337	° Zeichen	\367	π
\57	/	\340	α	\375	÷
\72	:	\341	ä	\377	alles ein
\73	;	\342	ß, β		



**Tabelle A3.2** Anweisungen

Anweisung	Kommentar
lcd.begin(16, 2);	Display wird initialisiert, 16 Zeichen, 2 Zeilen
lcd.home();	setzt den Cursor in die linke obere Ecke
lcd.setCursor(0, 1);	Textbeginn 1. Spalte in der 2. Zeile
lcd.clear();	löscht das Display
lcd.backlight();	schaltet die Hintergrundbeleuchtung ein
lcd.noBacklight();	schaltet die Hintergrundbeleuchtung aus
lcd.noBlink();	schaltet das Blinken des Cursors aus
lcd.blink();	schaltet das Blinken des Cursors ein
lcd.cursor();	schaltet das Anzeigen des Cursors ein
lcd.noCursor();	schaltet das Anzeigen des Cursors aus
lcd.scrollDisplayLeft();	rollt die Anzeige ein Leerzeichen nach links
lcd.scrollDisplayRight();	rollt die Anzeige ein Leerzeichen nach rechts
lcd.leftToRight();	setzt die Schreibrichtung auf dem LCD von links nach rechts
lcd.rightToLeft();	setzt die Schreibrichtung auf dem LCD von rechts nach links
lcd.moveCursorLeft();	bewegt den Cursor ein Leerzeichen nach links
lcd.moveCursorRight();	bewegt den Cursor ein Leerzeichen nach rechts
lcd.autoscroll();	schaltet das automatische Rollen des LCD ein
lcd.noAutoscroll();	schaltet das automatische Rollen des LCD aus
lcd.off();	schaltet das LCD aus
lcd.on();	schaltet das LCD an
lcd.write( ... );	interpretiert Daten als Binärdaten, " " für Text möglich
lcd.print( ... );	interpretiert Daten als ASCII, " " für Text möglich

## Sachverzeichnis

- 1-wire 115
- Ablaufauswahl 91
- Ablaufsprache 91, 100
- Ablaufsteuerung 85 ff
- Addition im Zweierkomplement 164
- Adressensuche 144
- Aktion 87 ff
- Analog/Digital-Wandler 104, 108, 139
- analogWrite 55
- Anfangsschritt 89
- Anschlüsse Arduino Uno R3 43
- Anweisungsblock 35 ff, 58, 94
- ARDUBlock 196
- Arduino Uno R3 21 ff
- ASCII code 155
- ASCII-Tabelle 212
- Assembler 181 ff
- AT.h 80
- AT.cpp 81
- Ausschaltverzögerungszeit 15
- Auswahlbetrieb 91
- Beharrungszustand 122
- Bestimmungszeichen 91
- Bibliothek für C++ 80
- Bibliothek einbinden in IDE 84
- Blockschaltbild 120
- Bootloader 185
- Boolesche Algebra 1-6
- Bus 139 ff
- Busadresse 144
- CAN-Bus 148, 159
- carry 164
- carry out 164
- case 94, 207
- Compiler 183
- const 37, 40
- Continuous Function Chart 20
- data direction register 176
- data register 176
- data memory 178
- Datenkapselung 194
- Datentypen 35, 203
  - bool 35
  - byte 35
  - int 35
  - long 35
  - float 35
- delay() 15
- Digital/Analog-Wandler 121
- digitale Anschlüsse 46
- Dualzahlen 162 ff
- EEPROM 180 ff
- Einschaltverzögerung 15, 66
- Exklusiv-ODER-Verknüpfung 5, 33, 38
- Exponent 169
- F() 180
- Farbkodierung 44
- Feldbusse 157
- Festwertregelung 119
- Flag 186 ff
- Flash Speicher 178 ff
- Flash-EEPROM 178 ff
- Flipflop 11 ff, 62 ff
- floating-point 168
- formatierte Ausgabe 29
- Führungsgröße 120
- Funktion 38, 64, 204
- Funktionsplan 85 ff
- Fußgängerampel 96
- Fritzing 48

- Gleitkommazahl 30, 35, 168
- Halbaddierer 9
- Header-Datei 80 ff
- Heileiter 103 ff
- Hexadezimalzahlen 162
- I-Regler 128
- I<sup>2</sup>C-Bus 139 ff
- IDE Entwicklungsumgebung 22 ff
- if-Anweisung 58, 205
- In-circuit Serial Programmer (ISP) 186
- INPUT\_PULLUP 46, 51
- Instanz 110, 196
- Integralregler 128
- Interrupt 142 ff, 188 ff
- Interrupt-Serviceroutine 143, 188
- ISR 188
- Kennlinie 123
- Klasse 195
- komplementieren 165, 177
- K<sub>PR</sub> Proportionalbeiwert 124 ff
- LCD-Display 109 ff, 139, 213
- Least Significant Bit 162
- Leuchtdiode LED 22, 44 ff
- Linienstruktur 159
- Linker 81, 195
- Ltkolben 119 ff
- LSB 162
- Mantisse 169
- Marke 187
- Maschinenkonstante 170
- Maschinensprache 181 ff
- Master 139 ff
- memory
  - data 178
  - program 178
- millis() 35, 67
- mnemonic 168, 183
- Modbus 159
- Monitor 27 ff
- MSB (most significant bit) 155
- NAND-Verknpfung 5
- Negation 4
- NOR-Verknpfung 4
- Normalisieren 169
- NTC-Widerstand 103
- ODER-Verknpfung 3
- objektorientiert 194 ff
- Objektorientiertes Programmieren 194 ff
- Oktalzahl 163
- overflow 169
- opcode 182
- Parallelbetrieb 91
- Parittsbit 155 ff
- permanent zyklischer Betrieb 61
- pinMode(pin, mode) 46
- port 176
- PULLUP Widerstand 46, 51
- Puls-Breiten-Modulation 54, 130
- Puls-Pause-Verhltnis 54
- Prfix 162
- Prioritt 6
- Profibus 158
- Proportionalbeiwert K<sub>PR</sub> 125
- Proportionalregler 126
- Protokoll 158
- P-Regler 126
- PI-Regler 128
- program memory 178
- Programmieranweisungen 203 ff
  - Datentypen und Variable 203
  - for-Anweisung 206
  - Funktionen 204
  - if-Anweisung 58, 205
  - switch-Anweisung 207
  - Verknpfungen 204
  - while-Anweisung 206
- Proportionalbeiwert 125
- Proportionalregler 125

- PWM 54
- Referenzvariable 64
- Regelung 119 ff
- Regeldifferenz 121
- Regelgröße 120
- Regelstrecke 121
- Register 176, 184
- Reglerverstärkung 125
- return 38
- RS-232-Schnittstelle 155 ff
- RS-Flipflop 11 ff, 62 ff
- rücksetzdominant 14, 62
- Rücksetz-Eingang 12
- Rundungsfehler 170
- Schritt 87 ff
- SCL-Leitung (Serial Clock Line) 109, 139
- SDA-Leitung (Serial Data Line) 109, 139
- Serielle Schnittstelle 155 ff
- Serial 25, 208
- setzdominant 14
- Setz-Eingang 12
- shield 44
- Signalflussplan 20
- Sketch 26
- Slave 139
- Sollwert 120
- Speicherglied 11 ff
- SPI-Bus 140
- Sprungantwort 121
- Stack 35, 180 ff
- static 35, 63
- stationärer Endwert 122
- Statusregister 166, 186
- Steckbrett 45, 210
- Stellgröße 121
- Störverhalten 121
- Strukturierter Text 19
- Subtraktion im Zweierkomplement 164
- switch 94 ff, 207
- UART 155
- Überlauf 165
- UND-Verknüpfung 1
- Underflow 169
- unsigned 35
- Unterlauf 169
- Variable 28, 34 ff
  - deklarieren 28
  - initialisieren 28
  - Lebensdauer 35
    - lokale 35
    - globale 37
- Vererbung 194
- Vergleichsoperator 59, 205
- Verknüpfungssteuerung 1 ff
- volatile 177
- Volladdierer 10
- watchdog timer 192
- Widerstand 44 ff
- wire 110, 139, 207
- Wirkungslinie 88
- Wirkungsplan 120
- Zeitglied 15, 66 ff, 74 ff
- Zuweisungszeichen 29
- Zweierkomplement 164
- Zwei-Hand-Steuerung 20, 76, 195
- .ZIP-Bibliothek 84
- && 2, 32
- & 40, 64
- ! 4
- = 29
- == 59
- ; 29
- | 40
- | | 32
- Transitionsbedingung 87

