CE

Impressum

www.elo-web.de
Autor: Uli Sommer

ISBN 978-3-645-10174-5

Produziert im Auftrag der Firma Conrad Electronic SE, Klaus-Conrad-Str. 1, 92240 Hirschau

Elektrische und elektronische Geräte dürfen nicht über den Hausmüll entsorgt werden!
Entsorgen Sie das Produkt am Ende seiner Lebensdauer gemäß den geltenden gesetzlichen Vorschriften. Zur Rückgabe sind Sammelstellen eingerichtet worden, an denen Sie Elektrogeräte kostenlos abgeben konnen. Ihre Kommune informiert Sie, wo sich solche Sammelstellen befinden.

CE

Dieses Produkt ist konform zu den einschlägigen CE-Richtlinien, soweit Sie es gemäß der beiliegenden Anleitung verwenden. Die Beschreibung gehört zum Produkt und muss mitgegeben werden, wenn Sie es weitergeben.

INHALT

ARDUINO™-LERNPAKET MIT LC-DISPLAY

Dieses Lernpaket richtet sich an die Arduino™-Programmierer, die bereits die Grundlagen von Arduino™ und der Mikrocontrollertechnik erlernt haben und nun mehr aus ihrem Arduino™-Mikrocontroller herausholen wollen.

Wir zeigen Ihnen anhand kleiner Beispiele mit ausführlichen Erklärungen, wie LC-Displays, kurz LCDs, funktionieren und Sie sie mit Arduino™ ansteuern und in der Praxis verwenden. Zudem erfahren Sie, wie Sie zwischen PC und Arduino™ mithilfe kleiner VB.NET-Programme kommunizieren können.

Dieses Lernpaket vermittelt Ihnen die Grundlagen der praxisnahen Arduino™-Programmierung und zeigt Ihnen einige Kniffe, die Sie bei Ihren eigenen Projekten anwenden können.

1 CD-ROM ZUM LERNPAKET

Diesem Lernpaket liegt eine CD-ROM bei, die verschiedene Programme und Beispiele enthält. Die CD-ROM erleichtert Ihnen das Arbeiten mit diesem Buch. Die hier abgedruckten Beispiele sind auf der CD-ROM enthalten.

1.1 | Inhalt der CD-ROM

- Arduino™-Entwicklungsumgebung (IDE)
- Beispielprogrammcode zum Lernpaket
- VB.NET-Programme

1.2 | GPL (General Public License)

Sie können Ihre eigenen Programme mit anderen Anwendern über das Internet austauschen. Die Beispielprogramme stehen unter der Open-Source-Lizenz GPL (General Public License). Daher sind Sie berechtigt, die Programme unter den Bedingungen der GPL zu modifizieren, zu veröffentlichen und anderen Anwendern zur Verfügung zu stellen, sofern Sie Ihre Programme dann ebenfalls unter die GPL stellen.

1.3 | Systemvoraussetzung

Ab Windows 7/8/8.1, 32 und 64 bit oder höher, Linux 32 und 64 bit, Mac OS X, CD-ROM-Laufwerk, Java

Hinweis
Dieses Lernpaket enthält VB.NET-Programme, die nur unter Windows funktionieren. Die grundlegenden Arduino™-Programme zu diesen Experimenten funktionieren auch auf anderen Betriebssystemen. Nur die .NET-PC-Programme benötigen zum Experimentieren ein Windows-Betriebssystem mit .NET Framework.

1.4 | Updates und Support

Arduino™ wird ständig weiterentwickelt. Updates können kostenlos von der Website *http://arduino.cc* heruntergeladen werden.

2 INHALT DES LERNPAKETS

Das Lernpaket enthält alle Bauteile, die Sie für die Experimente benötigen. Bitte kontrollieren Sie die Bauteile auf Vollständigkeit, bevor Sie mit den Experimenten beginnen!

Hinweis
Die Arduino™-UNO-Mikrocontrollerplatine ist nicht im Lieferumfang enthalten. Da es sich um ein Lernpaket handelt, das sich an den fortgeschrittenen Arduino™-Programmierer richtet, sollte ein Arduino™ UNO oder MEGA bereits auf dem Arbeitsplatz vorhanden sein. Die Boards gibt es kostengünstig bei Conrad Electronic, im Franzis Shop oder bei anderen Onlineversendern.

Bauteileliste
1 x Steckbrett Tiny, 1 x LCD 16 x 2 blau, 1 x Stiftleiste 16 Pins zum Einlöten, 1 x Taster, 1 x NTC 472, 1 x Fototransistor, 1 x 10 kΩ, 1 x 2,2 kΩ, 1 x 330 Ω, 14 x Steckbrücken

2.1 | Sicherheitshinweise

Die Arduino™-Platinen sowie das Display sind weitgehend gegen Fehler abgesichert, sodass es kaum möglich ist, den PC zu beschädigen. Die Anschlüsse der USB-Buchse sind auf der Platinenunterseite nicht isoliert. Wenn Sie die Platine auf einen metallischen Leiter stellen, kann es daher zu einem höheren Strom kommen, was den PC und die Platine beschädigen könnte.

Beachten Sie bitte die folgenden Sicherheitsregeln!

- Vermeiden Sie metallische Gegenstände unter der Platine oder isolieren Sie die gesamte Unterseite mit einer nicht leitenden Schutzplatte oder Isolierband.
- Halten Sie Netzteile, andere Spannungsquellen oder führende Leiter mit mehr als 5 Volt (V) von der Experimentierplatine fern.
- Schließen Sie die Platine nach Möglichkeit nicht direkt an den PC, sondern über einen Hub an. Dieser enthält meist eine zusätzliche wirksame Schutzschaltung. Wenn dennoch etwas passiert, wird im Normalfall nur der Hub und nicht der PC beschädigt.

3 DIE BAUTEILE UND IHRE FUNKTION

Die Bauteile des Lernpakets werden an dieser Stelle vorgestellt und die jeweilige Funktion wird kurz erklärt. Aber erst die folgenden Experimente vermitteln die praktischen Erfahrungen mit der Schaltungstechnik der Elektronik.

3.1 | Steckbrett

Auf dem Steckbrett (engl. Breadboard) können Sie Ihre Schaltungen ohne Lötarbeiten aufbauen. Unser Steckbrett besteht auch 17 Spalten und 5 Zeilen. Die Spalten mit je 5 Kontakten sind untereinander in einer Linie verbunden (von oben nach unten, siehe Abbildung). Der Trennsteg in der Mitte des Steckbretts kennzeichnet, dass keine Verbindung zum anderen 17 Spalten und 5 Zeilen großen Feld besteht. Es hat sich als hilfreich herausgestellt, die Anschlussdrähte der Bauteile zuvor schräg abzuzwicken, sodass eine Art Keil an den Drahtenden entsteht. Dadurch lassen sich die Bauteile leichter in das Steckbrett einstecken. Sollte das Einstecken der Bauteile doch einmal schwieriger sein, benutzen Sie am besten eine kleine Feinmechanikerflachzange, um das Bauteil mit etwas mehr Druck in das Steckbrett zu stecken.

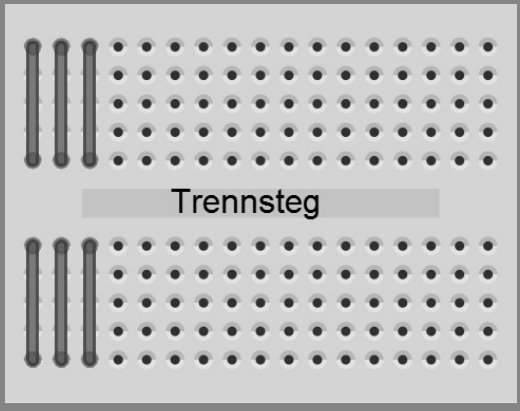

Abb. 3.1: Ministeckbrett

3.2 | Steckbrücken

Dem Lernpaket liegen mehrere fertig konfektionierte Steckbrücken bei. Mit ihnen werden die Verbindungen zwischen dem Steckbrett und der Arduino™-Platine hergestellt. Die Steckbrücken besitzen auf beiden Seiten einen kleinen Stift, der sich leicht in das Experimentierboard sowie in die Arduino™-Platine stecken lässt. Seien Sie aber trotzdem vorsichtig dabei, damit Sie keinen Stift versehentlich abbrechen oder verbiegen!

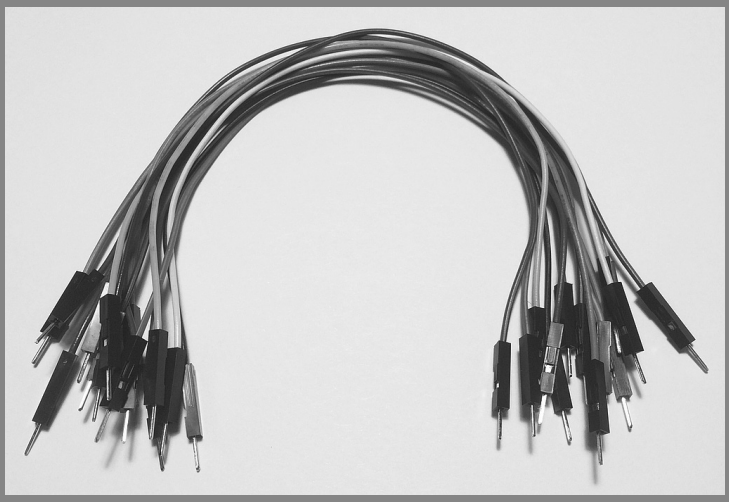

Abb. 3.2: Steckbrücken

3.3 | Taster

Ein Taster hat eine ähnliche Funktion wie ein Schalter. Einen Schalter kennen Sie vom Ein- oder Ausschalten des Lichts in der Wohnung. Drücken wir die Wippe nach unten, geht das Licht an. Drücken wir sie nach oben, geht das Licht wieder aus. Ein Schalter verharrt also in seiner Stellung. Bei einem Taster ist das etwas anders. Betätigen wir den Taster, wird der Stromkreis geschlossen. Er bleibt aber nur so lange geschlossen, wie wir den Knopf gedrückt halten. Lassen wir ihn los, öffnet er den Stromkreis wieder und das Licht erlischt. Der Taster geht über seine interne Mechanik automatisch in seinen Ur- oder Ruhezustand zurück.

Es gibt Taster, die bei Betätigung den Stromkreis schließen, und solche, die den Stromkreis öffnen. Man bezeichnet die Taster, die den Stromkreis schließen, oft mit »N. O.« (normaly open) und solche, die den Stromkreis öffnen, mit »N. C.« (normaly close). Die Abbildung zeigt einen Taster, der dem Lernpaket beiliegt. Er schließt bei Betätigung den Stromkreis und der Strom kann von Kontakt 1 nach 2 fließen. Die beiden anderen Kontakte sind jeweils miteinander verbunden.

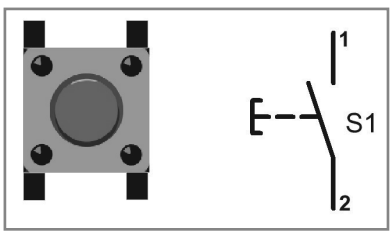

Abb. 3.3: Der Taster

3.4 | Widerstände

Widerstände braucht man zur Strombegrenzung und zum Einstellen von Arbeitspunkten sowie als Spannungsteiler in elektronischen Schaltungen. Die Einheit für den elektrischen Widerstand ist Ohm (Ω). Durch das Vorzeichen Kilo (k, Tausend) oder Mega (M, Million) erhält man eine abgekürzte Schreibweise für große Widerstandswerte.

1 kΩ = 1000 Ω

10 kΩ = 10.000 Ω

100 kΩ = 100.000 Ω

1 MΩ = 1.000.000 Ω

10 MΩ = 10.000.000 Ω

In Schaltplänen lässt man das Zeichen Ω meist weg und kürzt 1 kΩ mit 1 k ab. Der Wert des Widerstands ist in Form eines Farbcodes auf dem Widerstand aufgebracht. Meist sind es drei farbige Ringe und ein zusätzlicher vierter Ring, der die Genauigkeit des Widerstands angibt. Metallschichtwiderstände haben eine Toleranz von nur 1 %. Das wird durch einen braunen Toleranzring dargestellt, der etwas breiter aufgedruckt ist als die restlichen vier Farbringe. Dadurch soll eine Verwechslung mit einem normalen Wertring mit der Bedeutung »1« verhindert werden.

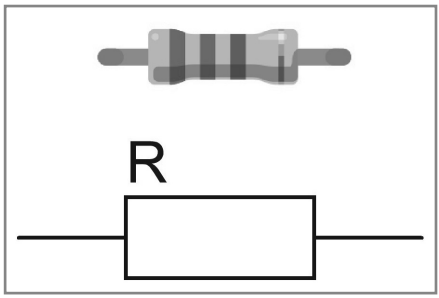

Abb. 3.4: Der Widerstand und sein Schaltzeichen

Widerstände mit einer Toleranz von +/-5 % gibt es in Werten der E24-Reihe, wobei jede Dekade 24 Werte mit etwa gleichmäßigem Abstand zum Nachbarwert enthält.

Die Widerstände der E24-Normreihe siedeln sich folgendermaßen an:
1,0 / 1,1 / 1,2 / 1,3 / 1,5 / 1,6 / 1,8 / 2,0 / 2,2 / 2,4 / 2,7 / 3,0 / 3,3 / 3,6 / 3,9 / 4,3 / 4,7 / 5,1 / 5,6 / 6,2 / 6,8 / 7,5 / 8,2 / 9,1

Der Farbcode wird ausgehend von dem Ring gelesen, der näher am Rand des Widerstands liegt. Die ersten beiden Ringe stehen für die zwei Ziffern, der dritte Ring für den Multiplikator des Widerstandswerts in Ohm. Ein vierter gibt die Toleranz an.

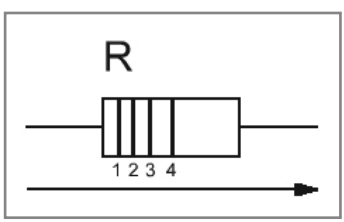

Abb. 3.5: So wird der Widerstandswert abgelesen.

Ein Widerstand mit den Farbringen Gelb, Violet, Braun und Gold hat den Wert 470 Ω bei einer Toleranz von +/-5 %. Versuchen Sie gleich einmal, die Widerstände des Lernpakets zu identifizieren.

3

Farbe	Ring 1	Ring 2	Ring 3 (Faktor)	Ring 4 (Toleranz)
Silber	-	-	1 x 10^{-2} = 0,01 Ω	+/- 10%
Gold	-	-	1 x 10^{-1} = 0,1 Ω	+/- 5%
Schwarz	0	0	1 x 10^{0} = 1 Ω	-
Braun	1	1	1 x 10^{1} = 10 Ω	+/- 1%
Rot	2	2	1 x 10^{2} = 100 Ω	+/- 2%
Orange	3	3	1 x 10^{3} = 1 kΩ	-
Gelb	4	4	1 x 10^{4} = 10 kΩ	-
Grün	5	5	1 x 10^{5} = 100 kΩ	+/- 0,5%
Blau	6	6	1 x 10^{6} = 1 MΩ	+/- 0,25%
Violett	7	7	1 x 10^{7} = 10 MΩ	+/- 0,1%
Grau	8	8	1 x 10^{8} = 100 MΩ	-
Weiß	9	9	1 x 10^{9} = 1000 MΩ	-

Abb. 3.6: Tabelle für Widerstände mit vier Farbringen

Tipp
Im Internet findet man unter dem Suchbegriff »Widerstandscode Rechner« viele Widerstandsfarbcoderechner, z. B. unter *http://www.ab-tools.com/de/software/widerstandsrechner/* oder *http://www.dieelektronikerseite.de/Tools/Widerstandsrechner.htm*.

Es gibt auch eine altmodische Variante, wie in der Abbildung unten zu sehen. Mit dieser »Widerstandlehre«, auch *Vitrometer* genannt, lässt sich schnell und ohne Computer der Widerstand durch einfaches Drehen der Farbräder bestimmen. So prägen sich die Farbcodes wesentlich schneller ins Gedächtnis ein als mit der Computervariante.

Abb. 3.7: Das Widerstandsvitrometer

3.5 | Temperatursensor

Zum Erfassen der Temperatur liegt ein NTC-Temperatursensor bei. Die Bezeichnung NTC steht für »negativer Temperaturkoeffizient« und besagt, dass der Widerstand bei zunehmender Erwärmung sinkt. Es handelt sich somit um einen Heißleiter. Der im Lernpaket enthaltene NTC besitzt einen Widerstand von 4,7 kΩ bei 25 °C/298,15 K (Kelvin).

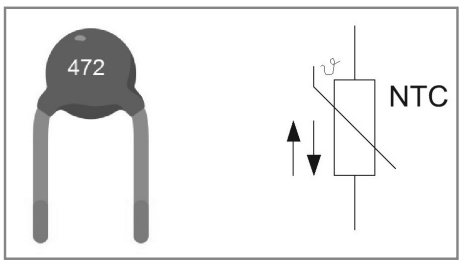

Abb. 3.8: Der NTC-Temperaturfühler und sein Schaltzeichen

3.6 | Fototransistor

Um die Helligkeit zu bestimmen, kommen in der modernen Elektronik häufig Fototransistoren zum Einsatz. Im Lernpaket finden Sie ein Bauteil vor, das einer weißen Leuchtdiode sehr ähnlich sieht, nur dass es sich um einen Fototransistor handelt. Er sieht nicht nur anders aus als ein normalen Bipolartransistor, sondern besitzt zudem keinen Basisanschluss. Die Basis, also der Eingang eines normalen Transistors, der für die Stromsteuerung zwischen Kollektor und Emitter zuständig ist, ist bei einen Fototransistor das in das Gehäuse einfallende Licht. Das Licht trifft dort auf das Silizium und lässt je nach Lichtstärke einen niedrigeren oder höheren Strom zwischen Kollektor und Emitter fließen.

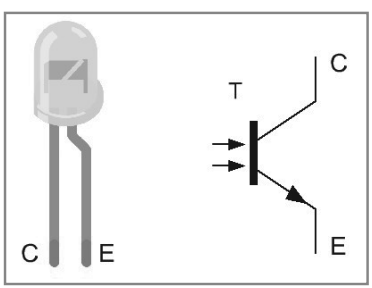

Abb. 3.9: Der Fototransistor und sein Schaltzeichen

3

3.7 | LC-Display

Der Hauptakteur dieses Lernpakets ist das blau-weiße LCD. Für das Lernpaket wurde ein LCD mit zwei Zeilen und 16 Spalten mit je 5 x 8 Dots (Punkten) verwendet. Diese Displays sind mittlerweile auch einzeln in jedem gutem Elektronikfachgeschäft oder Internetshop für wenige Euros erhältlich. Farblich gibt es sie in Grün, Blau, Bernsteinfarben, Gelb und ein paar Sonderfarben, die jedoch meist teurer sind. In unseren Fall ist ein blaues LCD verbaut. Als LCD-Controller ist ein KS0066/HD44780 verbaut, der von vielen Herstellern produziert wird – aber dazu später mehr.

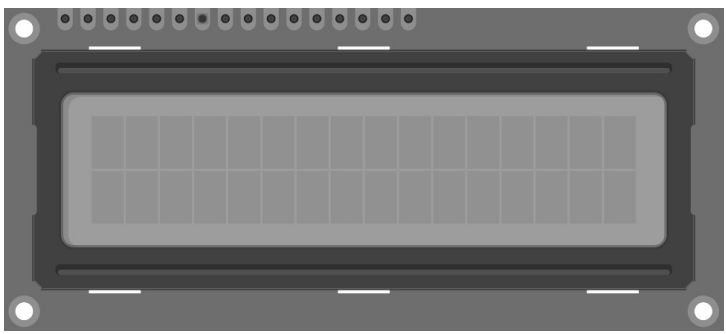

Abb. 3.10: Das LCD mit zwei Zeilen à 16 Zeichen

Bevor Sie das LCD in den Experimenten verwenden können, müssen Sie die beiliegende 16-polige Stiftleiste in die Kontakte des LCD einlöten. Dazu stecken Sie die Stiftleiste mit den kurzen Kontakten von hinten in das LCD und löten vorerst nur einen Kontakt an. So können Sie die Stiftleiste sauber im 90°-Winkel ausrichten. Wenn die Stiftleiste ausgerichtet wurde, können Sie die restlichen Pins anlöten. Sofern Sie noch keinen Lötkolben besitzen, besorgen Sie sich einen günstigen Handlötkolben mit einer Leistung zwischen 20 und 30 W und Elektroniklötzinn. Diese Investition lohnt sich auf jeden Fall, wenn man sich mit Arduino™ und Elektronik beschäftigt.

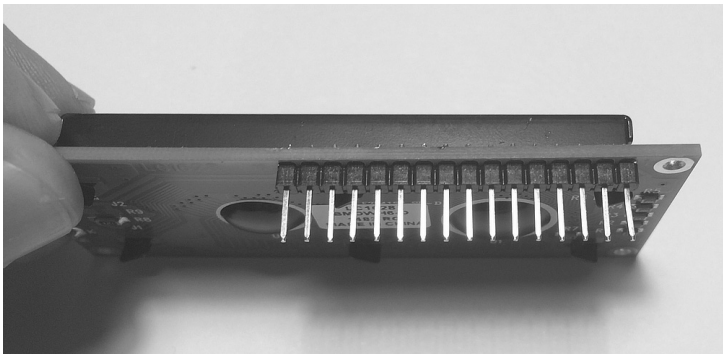

Abb. 3.11: Fertig eingelötete Stiftleiste

4 ERSTER FUNKTIONSTEST

Verdrahten Sie den ersten Versuch so, wie in der Abbildung gezeigt. Seien Sie dabei vorsichtig, um die Pins der Steckbrücken nicht zu verbiegen oder gar abzubrechen.

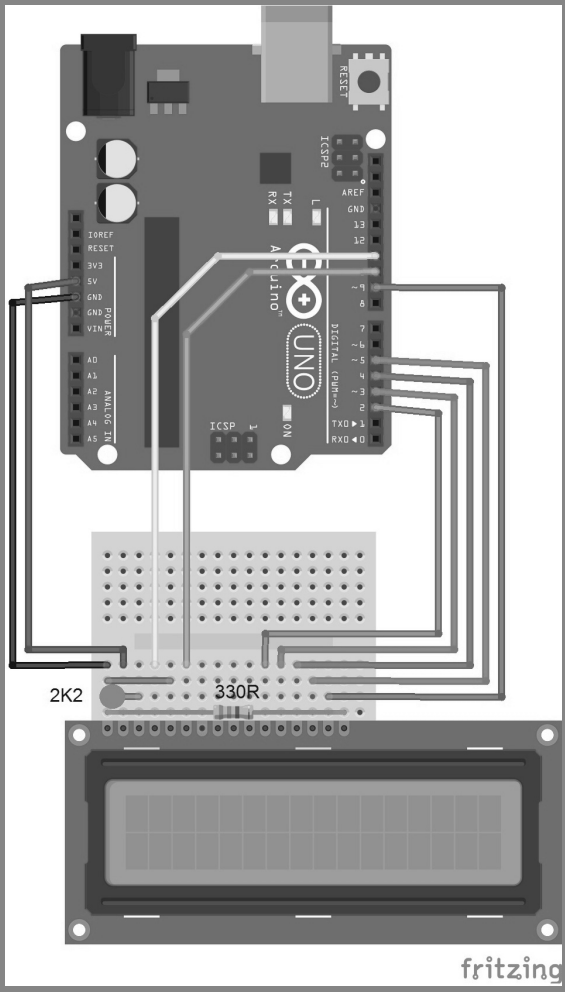

Abb. 4.1: Experimentier-Grundbeschaltung mit dem LCD

Kontrollieren Sie am Ende die Schaltung noch einmal sorgfältig auf Richtigkeit, um keine Bauteile zu beschädigen.

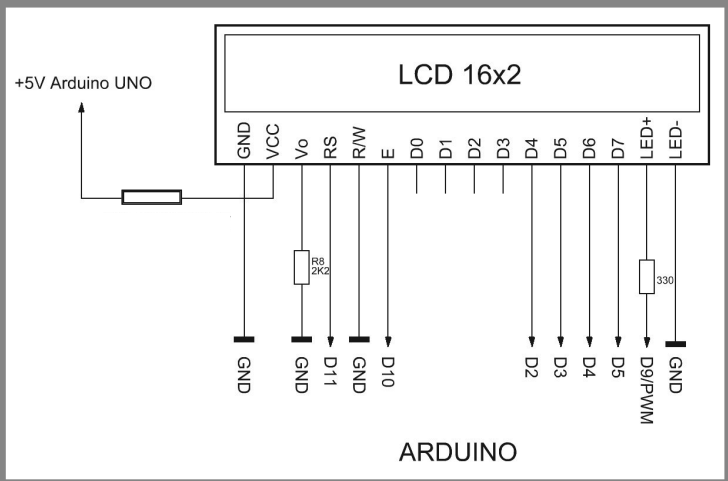

Abb. 4.2: Der Schaltplan zur LCD-Grundschaltung

Info
Sollten Sie zum ersten Mal mit Arduino™ arbeiten, müssen Sie zuvor die Arduino™-Entwicklungsumgebung herunterladen. Sie finden sie auf der offiziellen Arduino™-Website *http://www.arduino.cc*.
Hier können Sie Ihr Betriebssystem auswählen und bestimmen, ob Sie die Installer- oder die Zip-Version haben möchten. Bei der Installer-Version installieren Sie Arduino™ wie ein normales Standardprogramm. Bei der Zip-Version wird keine Installation benötigt. Hier entpacken Sie das Zip-File und speichern es an einen gewünschten Speicherort auf Ihren Rechner ab. Das hat den Vorteil, dass Sie den Arduino™ z. B. auf einem USB-Stick abspeichern und überall mit hinnehmen können.

Achtung
Speichern Sie Aduino nur dort ab, wo Sie alle Rechte zum Schreiben, Lesen u. Ä. besitzen!

Um einen ersten Funktionstest durchzuführen, laden Sie das folgende Programm auf das Arduino™-Board. Die Beispielprogramme finden Sie alle auf der mitgelieferten CD im Ordner *Beispiele*.

4

Das Programm lässt einen Text und eine Art Zähler auf dem LCD erscheinen und eignet sich, da es sehr klein und übersichtlich ist, gut als erster Funktionstest, um zu kontrollieren, ob alles ordnungsgemäß funktioniert.

Beispielcode: LCD

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 void setup()
009 {
010     // LED-Hintergrundbeleuchtung
011     analogWrite(9, 150);
012
013     // LCD Ausgabe
014     lcd.begin(16, 2);
015     lcd.setCursor(0, 0);
016     lcd.print("**ARDUINO LCD**");
017     lcd.setCursor(0, 1);
018     lcd.print("CNT:");
019 }
020
021 void loop()
022 {
023     lcd.setCursor(5, 1);
024     lcd.print(millis() / 1000);
025 }
```

Wir sehen in der ersten Zeile des Programms, das wir zum Betrieb des LCD die Arduino™ Library mit dem Namen *LiquidCrystal.h* einbinden müssen. In dieser steht der komplexere Code, der zur Ansteuerung des Displays benötigt wird. Sie können im Arduino™-Ordner unter *Arduino\libraries\LiquidCrystal* einen Blick in die *LiquidCrystal.h* und *LiquidCrystal.cpp* werfen, um sich einen Einblick in die Funktionalität der Library zu verschaffen. Zum Öffnen dieser Dateien empfiehlt es sich, z. B. das Programm Notepad++ zu verwenden. Sie können es unter *http://www.notepad-plus-plus.org* kostenlos herunterladen.

Sie werden sehen, dass diese Library sehr viel Arbeit abnimmt, die bereits andere Programmierer für Sie erledigt haben. In unserem Arduino™-Programm, binden wir mit der *LiquidCrystal.h* nur das sogenannte Hea-

derfile mit ein und Arduino™ kennt nun automatisch alle LCD-Funktionen.

In der nächsten Zeile teilen wir Arduino™ mit, mit welchen Pins das LCD an der Arduino™-Platine angeschlossen ist.

```
001 LiquidCrystal lcd(11, 10, 2, 3, 4, 5)
```

Mit dem nächsten Kommando bestimmen wir, wie hell unsere Displayhintergrundbeleuchtung leuchten soll. Die LED des LCD ist mit dem Arduino™-Digital/PWM-Port D9 verbunden. Dieser kann als einfacher Digitalport oder als PWM (Pulsweiten-modulierter Port) verwendet werden. In unseren Versuchen verwenden wir ihn als PWM-Port. Somit können wir die Helligkeit der Hintergrundbeleuchtung stufenlos einstellen. Der Wert 150 lässt die LED bereits ausreichend hell leuchten. Wird der PWM-Wert kleiner gewählt, leuchtet die LED dunkler. Versuchen Sie gleich einmal, den Wert zu ändern und beobachten Sie, was passiert.

```
001 analogWrite(9, 150)
```

Die Initialisierung wäre somit fast erledigt. Nun müssen Sie nur noch angeben, wie viele Spalten (engl. columns) und Zeilen (engl. rows) das LCD besitzt, und zwar 16 Spalten/einzelne Zeichen und 2 Zeilen.

```
001 lcd.begin(16, 2)
```

Die Grundinitialisierung ist also erledigt. Jetzt können wir mit *lcd.setCursor* die Position des Cursors und somit des auszugebenden Textes bestimmen.

```
001 lcd.setCursor(0, 0)
```

Der erste Parameter gibt die Position innerhalb der Zeile an, also bei uns 0 bis 15. Der zweite Parameter die Zeilennummer also 0 oder 1.

Nun können wir den Text an der vorgebenden Position am LCD mit dem Befehl *lcd.print* ausgeben.

```
001 lcd.print("**ARDUINO LCD**")
```

Wir sehen auch, dass wir für die LCD-Ausgabe immer vor der eigentlichen Funktion »lcd.« schreiben müssen. Das gibt an, dass wir die Klasse *lcd* verwenden, die wir mit *#include <LiquidCrystal.h>* eingebunden haben. Arduino™ weiß nun beim »Übersetzen«, in Fachkreisen auch »Kompi-

lieren« genannt, woher dieser Aufruf kommt und welche Klasse dafür zuständig ist.

Wenn Sie sich schon mit der Programmiersprache C++ beschäftigt haben, erkennen Sie an der Endung **.cpp*, dass es sich um C++-Klassen handelt. Arduino™ basiert im Grunde auf C++. Das ist eine gute Möglichkeit, eigene Klassen oder Librarys zu programmieren und anderen Arduino™-Anwendern zur Verfügung zu stellen.

Nach diesem kurzen C++-Ausflug wieder zurück zu unserem Beispiel. Bisher haben wir uns noch in der `Setup()`-Funktion befunden, die beim Programmstart immer einmal durchlaufen wird und hauptsächlich zur Startkonfiguration dient. In ihr können wir Variablen vor dem eigentlichen Programmstart vorinitialisieren und die Hardware vorkonfigurieren.

Die darauf folgende `Loop()`-Funktion ist eine sogenannte Endlosschleife, die nie beendet wird. Das ist zugleich die Arduino™-Hauptschleife für unser Programm. Hier rufen wir bei jedem Durchlauf die Laufzeit in Millisekunden mit der Funktion `millis()` ab. Durch die Division von 1.000 erhalten wir die Ausgabe in Sekunden. Wir geben auf dem LCD die Programmlaufzeit in Sekunden wieder.

```
001 lcd.setCursor(5, 1)
002 lcd.print(millis() / 1000)
```

Da die Funktion `millis()` sehr interessant ist, probieren wir noch ein Experiment aus, bevor wir uns näher mit dem LC-Display beschäftigen, denn die Funktion `millis()` kann auch zur Zeitmessung von Programmdurchläufen verwendet werden, wie das folgende Beispiel zeigt.

Beispielcode: TIME_DIFF

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 long time_diff, diff;
009
010 void setup()
011 {
012     // LED-Hintergrundbeleuchtung
013     analogWrite(9, 150);
```

```
014
015     // LCD Ausgabe
016     lcd.begin(16, 2);
017     lcd.setCursor(0, 0);
018     lcd.print("**ARDUINO LCD**");
019     lcd.setCursor(0, 1);
020     lcd.print("TIME DIFF: ");
021 }
022
023 void loop()
024 {
025     diff = millis();
026
027     // Mein Programm Start
028
029     lcd.setCursor(11, 1);
030     lcd.print(diff - time_diff);
031     delay(100);
032
033     // Mein Programm Ende
034     time_diff = diff;
035 }
```

Der Beispielcode zeigt, wie man die Programmdurchlaufzeit ermittelt. Wir lesen dazu den aktuellen Zählerstand von `millis()` bei jedem neuen Programmdurchlauf aus und subtrahieren den letzten Zählerstand, den wir beim Programmende gespeichert habe. Das `delay(100)` am Ende des Programms simuliert eine längere Programmdurchlaufzeit. Ihr Programmcode muss zwischen *// Mein Programm Start* und *// Mein Programm Ende* stehen, um die Durchlaufzeit Ihres Programmcodes zu ermitteln.

Die ersten Experimente und somit auch der Funktionstest sind hiermit erfolgreich abgeschlossen. Lassen Sie die Schaltung so aufgebaut. Sie werden sie in den nächsten Experimenten wieder benötigen und erweitern. Im folgenden Kapitel lernen Sie das LC-Display und seine Eigenschaften etwas genauer kennen.

5 AUFBAU UND FUNKTIONSWEISE DER LC-DISPLAYS

LCDs finden Verwendung in vielen elektronischen Geräten, etwa in der Unterhaltungselektronik, in Messgeräten, Mobiltelefonen, Digitaluhren und Taschenrechnern. Auch Head-up-Displays und Videoprojektoren arbeiten mit dieser Technik. In der folgenden Abbildung ist das LCD aus dem Lernpaket abgebildet. Es handelt sich um ein Standard-5x8-Dot-Matrix-Display mit 2 Zeilen zu je 16 Zeichen.

Abb. 5.1: Das 16x2-LC-Display im Betrieb

Ein LCD besteht grundsätzlich aus 2 einzelnen Glasscheiben und einer speziellen Flüssigkeit dazwischen. Das Besondere an der Flüssigkeit ist, dass sie die Polarisationsebene von Licht dreht. Dieser Effekt wird durch Anlegen eines elektrischen Feldes beeinflusst. Also bedampft man die beiden Glasplatten jeweils mit einer hauchdünnen Metallschicht. Um polarisiertes Licht zu erhalten, klebt man auf die obere Glasplatte eine Polarisationsfolie, den sogenannten Polarisator. Auf die untere Glasplatte muss nochmals eine solche Folie aufgebracht werden, allerdings mit um 90° gedrehter Polarisationsebene. Dies ist der Analysator.

Die Flüssigkeit dreht im Ruhezustand die Polarisationsebene des einfallenden Lichts um 90°, sodass es ungehindert den Analysator passieren kann. Das LCD ist somit durchsichtig. Legt man nun eine bestimmte Spannung an die aufgedampfte Metallschicht, drehen sich die Kristalle in der Flüssigkeit. Dadurch wird die Polarisationsebene des Lichts um

z. B. weitere 90° gedreht: Der Analysator versperrt dem Licht den Weg, das LCD ist undurchsichtig geworden.

5.1 | Polarisation von Displays

Unter Polarisation versteht man bei LC-Displays nicht die Polung der Spannungsversorgung, sondern es handelt sich hier um den Glas-, Flüssigkeits- und Filteraufbau des Displays. Die meisten LCDs sind TN-Displays (engl. Twisted-Nematic-Displays). Sie beinhalten eine Flüssigkeit, die die Polarisationseben des Lichts um 90° dreht. STNs (Super-Twisted-Nematics) drehen die Polarisationsebene des Lichts um mindestens 180°. Dadurch erreicht man einen besseren Kontrast der Anzeige. Allerdings erhält man in dieser Technik eine gewisse Färbung des Displays. Die gängigsten Farbgebungen nennt man *yellow-green* und *blue mode*. Ein sogenannter *gray mode* erscheint in der Praxis auch mehr blau als grau. Um den unerwünschten Farbeffekt zu kompensieren, verwendet man in der FSTN-Technik eine weitere Folie auf der Außenseite. Die dadurch entstehenden Lichtverluste machen diese Technik allerdings nur für beleuchtete Displays sinnvoll. Die verschiedenen Farben treten jedoch nur bei unbeleuchteten oder mit weißer Beleuchtung ausgestatteten Displays auf. Sobald die Beleuchtung eine Färbung aufweist (z. B. LED-Beleuchtung gelb-grün) tritt die jeweilige Displayfarbe in den Hintergrund. Ein Blue-mode-LCD mit gelb-grüner LED-Beleuchtung wird immer gelb-grün aussehen.

5.2 | Statische Ansteuerung, Multiplexbetrieb

Kleine Displays mit geringem Anzeigenumfang werden meist statisch angesteuert. Statische Displays haben den besten Kontrast und den größtmöglichen Blickwinkel. Die TN-Technologie erfüllt hier voll ihren Zweck (Schwarz-Weiß-Darstellung, kostengünstig). Werden die Displays allerdings größer, wären im statischen Betrieb immer mehr Leitungen nötig (z. B. Grafik 128 x 64 = 8.192 Segmente = 8.192 Leitungen). Da so viele Leitungen weder auf dem Display noch auf einem Ansteuer-IC Platz fänden, bedient man sich des Multiplexbetriebs. Das Display wird also in Zeilen und Spalten aufgeteilt und in jedem Kreuzungspunkt befindet sich ein Segment (128 + 64 = 192 Leitungen). Hier wird Zeile für Zeile abgescannt (64 x, d. h. Multiplexrate 1 : 64). Da immer nur 1 Zeile aktiv ist, leiden allerdings mit zunehmender Multiplexrate der Kontrast und auch der Blickwinkel.

5

5.3 | Blickwinkel 6 Uhr/12 Uhr

Jedes LC-Display besitzt eine sog. Vorzugsblickrichtung. Von dieser Richtung aus betrachtet hat das Display einen optimalen Kontrast. Die meisten Displays werden für den 6-Uhr-Blickwinkel (auch engl. bottom view, BV) produziert. Dieser Blickwinkel entspricht dem auf einem Taschenrechner, der flach auf dem Tisch liegt. 12-Uhr-Displays (engl. top view, TV) baut man am besten in die Frontseite eines Tischgeräts ein. Senkrecht von vorn lassen sich alle Displays lesen.

5.4 | Reflektiv, transflektiv, transmissiv

Reflektive (unbeleuchtete) Displays besitzen auf der Rückseite einen 100%-Reflektor. Eine Beleuchtung von der Rückseite ist nicht möglich. Transflektive Displays haben auf der Rückseite einen teildurchlässigen Reflektor. Sie lassen sich mit und ohne Beleuchtung ablesen. Dadurch sind sie unbeleuchtet aber etwas trüber als eine reflektive Version. Trotzdem ist das der wohl beste Kompromiss für beleuchtete LCDs. Transmissive Displays besitzen gar keinen Reflektor. Sie sind nur mit Beleuchtung ablesbar, dafür aber sehr hell. Das im Lernpaket enthaltene LCD ist ein transflektives LCD.

5.5 | Der Controller des LC-Displays

Dot-Matrix-Displays werden von vielen Herstellern in der ganzen Welt (und besonders in Taiwan) hergestellt. Neben Displays von Größen wie Data-Vision gibt es auch immer wieder Displays, deren Hersteller gar nicht zu ermitteln ist. Zum Glück sind Funktion und Anschluss der Displays immer gleich. Wir beschäftigen uns in diesem Lernpaket mit Displays, die einen Controller vom Typ HD44780 (oder kompatibel) verwenden, z. B. den KS0066.

Das einheitliche Verhalten aller Displays verdanken wir einem Controllerchip, der sich als Standard durchgesetzt hat und von allen Herstellern verbaut wird. Dabei handelt es sich um den HD44780 von Hitachi.

5.6 | So wird das Display vom Displaycontroller angesteuert

Die folgenden Abbildungen zeigen, wie der Displaycontroller (KS0066) mit dem Display verbunden ist. Diese Verschaltungen müssen Sie nicht vornehmen, da sie bereits auf dem LCD-Modul gegeben sind.

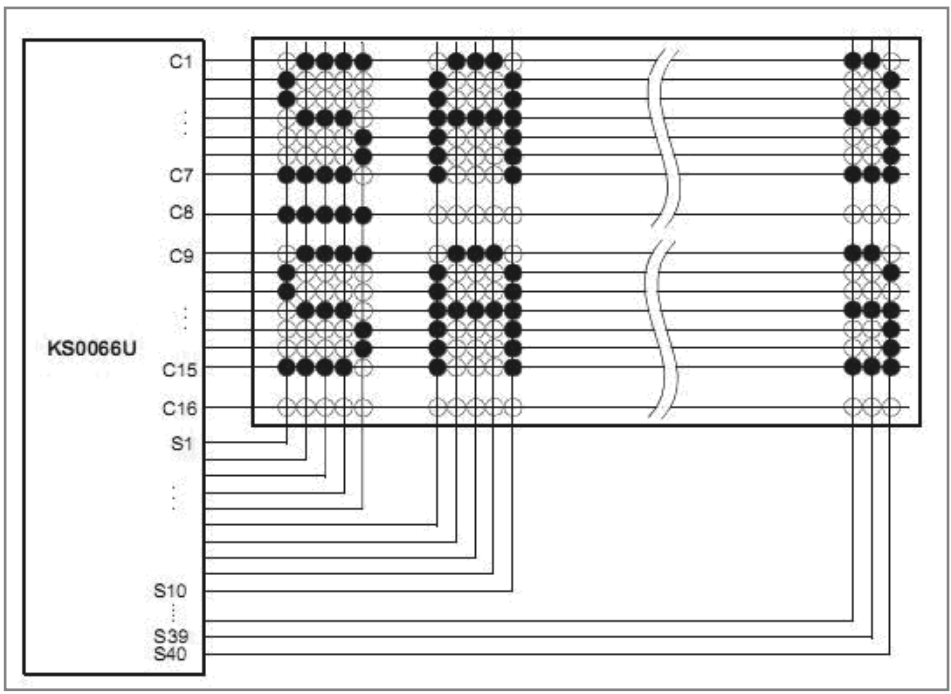

Abb. 5.2: Zweizeiliges 5x8-Dot-Display (Quelle: Datenblatt Samsung)

Die Controller sind mit den Displays teils unterschiedlich verbunden und je nach Hersteller auch anders beschaltet. So kann es sein, dass ein einzeiliges 16-Zeichen-Display aus 2 x 8 Zeichen besteht. Hier gilt es, wieder einen Blick in das Datenblatt zu werfen. Größere Displays verwenden auch häufig zwei Controllerchips, die über eine Chipselect- (CS) oder zwei Enable-Leitungen verfügen. Grund dafür ist, dass ein Controller nur einen Zeichenbuffer von 80 Zeichen besitzt. Durch Aneinanderschalten von mehren Displaycontrollern erhöht sich der Zeichenbuffer um zusätzlich je 80 Zeichen. Die Displays besitzen auch einen anderen Anschluss und sind relativ leicht von den Standard-LCD-Modulen zu unterscheiden. So

5

kann man davon ausgehen, dass ein Display ohne Beleuchtung 14 Pins und ein Display mit Beleuchtung 16 Pins besitzt.

5.7 | Die Kontrasteinstellung des Displays

Wie bei anderen Bildschirmen können wir auch bei den LCD-Modulen den Kontrast einstellen. Das geschieht z. B. mit einem 10-kΩ-Potenziometer, das als variabler Spannungsteiler verschaltet ist. Da die LCDs mittlerweile eine sehr kleine Streuung der elektrischen Eigenschaften besitzen, kann auch die im Lernpaket angewandte Technik mit einem einzigen Festwiderstand angewandt werden. Wir verwenden dazu einen 2,2-kΩ-Widerstand, der zwischen Masse und den Kontrastanschluss *Vee* des LCD liegt, um den Kontrast fest einzustellen.

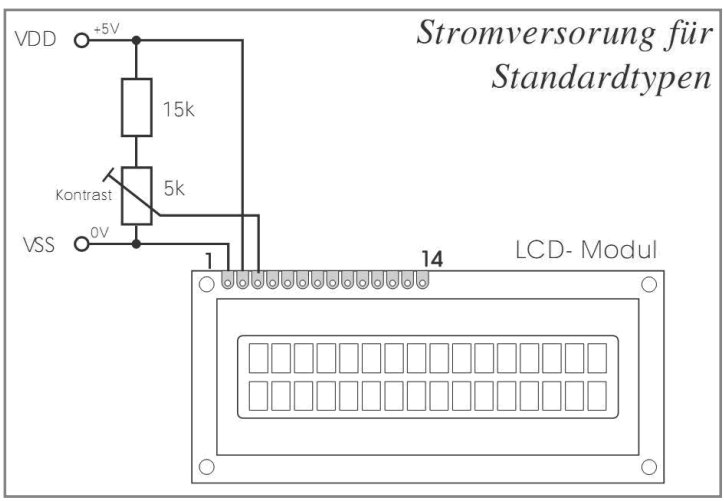

Abb. 5.3: Einfache Kontrasteinstellung mit Potenziometer (Quelle: Datenblatt Electronic Assembly)

Bei der Verwendung eines Potenziometers ohne Vorwiderstand ist der einstellbare Bereich, der sich auf den Kontrast auswirkt, jedoch sehr klein. Um eine bessere Spreizung des Kontrastbereichs zu erzielen, ist es ratsam, einen dementsprechenden Vorwiderstand zwischen Vcc (+5 V) und dem einen Ende des Potenziometers zu schalten.

Die Spannung am Pin Vee sollte zwischen 0 und 1,5 V einstellbar sein. Diese Schaltung eignet sich für eine Umgebungstemperatur von 0 bis 40 °C. Sollte der einstellbare Bereich nicht optimal sein (manche LCDs

weichen davon ab), müssen wir den Vorwiderstand etwas abändern. Praktische Werte liegen im Bereich zwischen 10 und 22 kΩ.

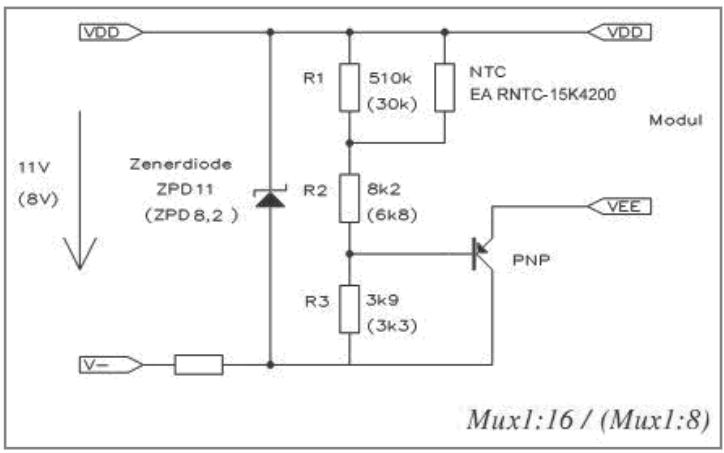

Abb. 5.4: Temperaturkompensierte Kontrasteinstellung (Quelle: Datenblatt Electronic Assembly)

Setzen wir das Display außerhalb des normalen Temperaturbereichs ein (0 bis 40 °C), ist es empfehlenswert, die Beschaltung wie im obigen Schaltplan durchzuführen. Diese Schaltung gleicht den Kontrast auf die Umgebungsbedingung an. Die Temperatur wird mit einem Temperaturfühler-NTC (engl. Negative Temperature Coefficient Thermistor) gemessen, der die Kontrastspannung über den PNP-Transistor verschiebt. Die LCD-Module haben die Eigenschaft, bei zu tiefen Temperaturen unter 0 °C nicht mehr richtig lesbar zu sein. Der Kontrast ist also temperaturabhängig.

5.8 | Der Zeichensatz

Die Displays besitzen einen Zeichensatz, der fest im Displaycontroller integriert ist. Indem wir die oberen und die unteren 4 Bit aneinanderreihen, erhalten wir das Datenbyte für das entsprechende ASCII-Zeichen. Beispiel für das ASCII-Zeichen A: 01000001

5

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	x	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		−	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	°	ö	

Abb. 5.5: Zeichensatz des LCD (Quelle: Datenblatt Samsung)

Machen wir ein Experiment mit dem LCD-Zeichensatz. Der folgende Programmcode zeigt, wie Sie die Zeichen der Zeichentabelle auf das Display schreiben können. Sonderzeichen wie das Gradsymbol oder das Ohmzeichen können nicht über eine Stringausgabe erfolgen. Da es sich um den erweiterten Zeichensatz des LCD handelt, müssen wir das über die Zeichentabelle erledigen.

```
001 lcd.write(B11110100)
```

Hier schreiben wir den binären Wert in den Displaycontroller, der das Omegazeichen ausgibt. Das B am Anfang der Zahlenfolge kennzeichnet, dass es sich um eine Zahlenfolge in binärer Schreibweise handelt.

So setzen sich die oberen und unteren 4 Bits für das Omega-Zeichen zusammen:

```
upper       =       1111
lower       =       0100
```

Versuchen Sie, auch andere Zeichen auszugeben, und sehen Sie dazu in der Zeichentabelle nach.

Upload

Das Experiment benötigt die LCD-Grundbeschaltung, wie wir Sie beim Funktionstest aufgebaut haben.

Beispielcode: ZEICHENSATZ

```
// LCD-Library einbinden
#include <LiquidCrystal.h>

// LCD-Pins festlegen
// RS, E, D4, D5, D6, D7
LiquidCrystal lcd(11, 10, 2, 3, 4, 5);

void setup()
{
    // LED-Hintergrundbeleuchtung
    analogWrite(9, 150);

    // LCD-Ausgabe
    lcd.begin(16, 2);
    lcd.setCursor(0, 0);
    lcd.write(B11110100);

}

void loop()
{
    // Nix zu tun ...
}
```

5.9 | Pinbelegung der gängigen LCDs

Die meisten Displays ohne Beleuchtung haben eine Pinbelegung wie in der folgenden Tabelle. Wenn Sie später einmal ein anderes LCD als das im

5

Lernpaket enthaltene verwenden, empfehlen wir, zuvor einen Blick in das dazugehörige Datenblatt zu werfen, um das LCD nicht zu beschädigen.

Pinbelegung			
Pin	**Symbol**	**Pegel**	**Beschreibung**
1	VSS	L	Versorgung 0V, GND
2	VDD	H	Versorgung +5V
3	VEE	-	Displayspg. 0..1,5V Kontrasteinstellung
4	RS	H / L	Register Select
5	R/W	H / L	H: Read / L: Write
6	E	H	Enable
7	D0	H / L	Datenleitung 0 (LSB)
8	D1	H / L	Datenleitung 1
9	D2	H / L	Datenleitung 2
10	D3	H / L	Datenleitung 3
11	D4	H / L	Datenleitung 4
12	D5	H / L	Datenleitung 5
13	D6	H / L	Datenleitung 6
14	D7	H / L	Datenleitung 7 (MSB)

Abb. 5.6: Gängige Pinbelegung von Standard-LCDs ohne Beleuchtung (Quelle: Datenblatt Electronic Assembly)

Bei LCD-Modulen mit Beleuchtung ist immer etwas Vorsicht geboten. Manche Hersteller legen die LED-Hintergrundbeleuchtungskontakte nicht an Pin 15 und 16, sondern an Pin 1 und 2. Auch hier sollten Sie zuvor einen Blick in das Datenblatt des Herstellers werfen, bevor Sie das LCD anschließen.

Info
Die LCD des Lernpakets hat die LED-Anschlüsse an Pin 15 (+ = Anode) und 16 (- = Kathode).

Sollten Sie zu den LCDs kein Datenblatt zur Hand haben, z. B. wenn Sie ein LCD auf einem Elektronikflohmarkt erworben haben, müssen Sie die Leiterbahnen nachverfolgen und die Hintergrundbeleuchtungsanschlüsse ausfindig machen. Diese sind meist etwas dicker ausgelegt als die anderen Leiterbahnen. Wenn Sie sicher sind, welche der Anschlüsse für die Beleuchtung zuständig sind, können Sie mit einem Multimeter, das

auf Diodentest eingestellt ist, die Polung ermitteln. In Durchlassrichtung müssen Sie dann eine Durchlassspannung zwischen 2 und 4 V messen. Eine weitere Möglichkeit wäre, die LED-Pins sowie die Polung zu identifizieren, wenn Sie mit einen Netzgerät oder einer Batterie von ca. 5 V und einem relativ großen Vorwiderstand (ca. 1–4,7 kΩ) die LED zum Leuchten bringen. Durch den großen Vorwiderstand ist die Gefahr einer Zerstörung des LCD relativ gering.

Pinbelegung			
Pin	**Symbol**	**Pegel**	**Beschreibung**
1	VSS	L	Versorgung 0V, GND
2	VDD	H	Versorgung +5V
3	VEE	-	Displayspannung 0..0,5V
4	RS	H / L	Register Select
5	R/W	H / L	H: Read / L: Write
6	E	H	Enable
7	D0	H / L	Datenleitung 0 (LSB)
8	D1	H / L	Datenleitung 1
9	D2	H / L	Datenleitung 2
10	D3	H / L	Datenleitung 3
11	D4	H / L	Datenleitung 4
12	D5	H / L	Datenleitung 5
13	D6	H / L	Datenleitung 6
14	D7	H / L	Datenleitung 7 (MSB)
15	LED +	-	LED-Versorgung Plus /Vorwiderstand!
16	LED -	-	LED-Versorgung Minus

Abb. 5.7: Gängige Pinbelegung von Standard-LCDs mit Beleuchtung (Quelle: Datenblatt Electronic Assembly)

6 DIE ARDUINO™ LIQUITCRYSTAL LIBRARY

Wie wir bereits beim Funktionstest erfahren haben, besitzt die Arduino™ LiquitCrystal Library eine ganze Reihe von Funktionen, die speziell für die Ausgabe auf dem LCD bestimmt sind. Nun lernen Sie die LCD-Funktionen genauer kennen.

6.1 | LiquidCrystal

LiquidCrystal legt fest, mit welchen Arduino™-Pins das LCD verbunden ist. Das LCD kann im 4- oder 8-Bit-Modus konfiguriert werden. Um es im 8-Bit-Modus zu verwenden, müssen Sie acht statt der vier Datenpins (D0 bis D7) angeben und diese mit der Arduino™-Platine verbinden.

Arduino™-Syntax

```
001 LiquidCrystal lcd(rs, enable, d4, d5, d6, d7)
002 LiquidCrystal lcd(rs, rw, enable, d4, d5, d6, d7)
003 LiquidCrystal lcd(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
004 LiquidCrystal lcd(rs, rw, enable, d0, d1, d2, d3, d4,
                                                  d5, d6, d7)
```

Bei unserem Lernpaket haben wir folgende Konfiguration:

```
001 // RS, E, D4, D5, D6, D7
002 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
```

RS= Arduino™-Pin D11

E = Arduino™-Pin D10

D4= Arduino™-Pin D2

D3= Arduino™-Pin D3

D2= Arduino™-Pin D4

D1= Arduino™-Pin D5

6.2 | .begin()

`.begin()` initialisiert das LCD mit den angebenden Zeilen und Spalten. Unser LCD besitzt 2 Zeilen und 16 Spalten. Die Parametrierung muss deshalb wie folgt sein:

Arduino™-Syntax

```
001 lcd.begin(16, 2)
```

6.3 | .clear()

`.clear()` löscht die angezeigten Zeichen und positioniert den Cursor in der Ecke links oben.

Arduino™-Syntax

```
001 lcd.clear()
```

6.4 | .home()

`.home()` positioniert nur den Cursor in der Ecke links oben. Es werden keine Zeichen gelöscht.

Arduino™-Syntax

```
001 lcd.home()
```

6.5 | .setCursor()

`.setCursor()` setzt den Cursor auf die angegebene Position. Hier wird, wie so oft in der Informatik, von Null aus gezählt. Die Position links oben, also das erste Zeichen in Zeile 1, ist wie folgt:

Arduino™-Syntax

```
001 lcd.setCursor(0, 0)
```

Der erster Parameter ist die Zeichenposition, der zweite Parameter die Zeile.

6

6.6 | .write()

`.write()` schreibt ein einzelnes Zeichen auf das LCD. Hiermit können wir auch Sonderzeichen aus der Zeichentabelle ausgeben oder den ASCII-Code für das Zeichen angeben.

Arduino™-Syntax

```
001 lcd.write(64)
```

Das Zeichen @ besitzt im ASCII-Code die Dezimalzahl 64.

Einzelne ASCII-Zeichen werden mit einen Apostroph gekennzeichnet. Wir könnten auch wie folgt schreiben:

Arduino™-Syntax

```
001 lcd.write('@')
```

6.7 | .print()

Mit `.print()` können wir ganze Zeichenketten sogenannte *Strings* ausgeben. Es ist auch möglich, damit Variablen auszugeben. Dazu gibt es eine Reihe von Formatierungsparametern (BASE), die man als zweiten Parameter angibt.

Arduino™-Syntax

```
001 lcd.print(data, BASE)
002
003 lcd.print("Arduino")  // es wird nur ein Text ausgeben
004
005 int variable1 = 100
006 lcd.print(variable1) // es wird der Wert der "Variablen1"
                                                    ausgeben
007
008 lcd.print(40 + 2)    // es wird die Summe aus 40 + 2
                                                  ausgegeben
009
010 lcd.print(3.1415, 2) // es wird nur 3.14 ausgeben
011
012 lcd.print(42, BIN)   // es wird die 42 binär ausgeben
```

6.8 | .cursor()

`.cursor()` schaltet den Cursor ein. Falls der Cursor ausgeschaltet war, wird er jetzt wieder sichtbar.

Arduino™-Syntax

```
001 lcd.cursor()
```

6.9 | .noCursor()

`.noCursor()` schaltet den Cursor aus (nicht sichtbar).

Arduino™-Syntax

```
001 lcd.noCursor()
```

6.10 | .blink()

`.blink()` schaltet den Cursor ein und lässt ihn blinken.

Arduino™-Syntax

```
001 lcd.blink()
```

6.11 | .noBlink()

`.noBlink()` schaltet den Cursor aus und beendet das Blinken.

Arduino™-Syntax

```
001 lcd.noBlink()
```

6.12 | .noDisplay()

`.noDisplay()` schaltet das Display aus. Es werden die Zeichen und die Cursorposition gespeichert.

Arduino™-Syntax

```
001 lcd.noDisplay()
```

6

6.13 | .display()

`.display()` schaltet das LCD nach einem `.noDisplay()` wieder ein. Die letzten Werte werden wiederhergestellt.

Arduino™-Syntax

```
001 lcd.display()
```

6.14 | .scrollDisplayLeft()

`.scrollDisplayLeft()` scrollt bei jedem Aufruf den Displayinhalt um ein Zeichen nach links.

Arduino™-Syntax

```
001 lcd.scrollDisplayLeft()
```

6.15 | .scrollDisplayRight()

`.scrollDisplayRight()` scrollt bei jedem Aufruf den Displayinhalt um ein Zeichen nach rechts.

Arduino™-Syntax

```
001 lcd.scrollDisplayRight()
```

6.16 | .autoscroll()

`.autoscroll()` scrollt den Displayinhalt automatisch von rechts nach links. Ist das Ende der Zeichenkette erreicht, wird die Scrollrichtung automatisch umgeschaltet. Es wird bei jeden Aufruf 1 x weitergeschoben (gescrollt).

Arduino™-Syntax

```
001 lcd.autoscroll()
```

6.17 | .noAutoscroll()

`.noAutoscroll()` beendet die `.autoscroll()`-Funktion.

Arduino™-Syntax

```
001 lcd.noAutoscroll()
```

6.18 | .leftToRight()

`.leftToRight()` legt die Ausgaberichtung der Zeichen fest. Es wird von links nach rechts geschrieben.

Arduino™-Syntax

```
001 lcd.leftToRight()
```

6.19 | .rightToLeft()

`.rightToLeft()` legt die Ausgaberichtung der Zeichen fest. Es wird von rechts nach links geschrieben.

Arduino™-Syntax

```
001 lcd.rightToLeft()
```

6.20 | .createChar()

Mit `.createChar()` können wir eigene Zeichen erstellen. Dazu müssen wir ein Array mit acht Datenfeldern erstellen, indem wir unser Zeichen definieren. Mit `lcd.createChar` geben wir unserem Zeichen mit dem ersten Parameter eine laufende Nummer und im zweiten Parameter übergeben wir den Namen des Arrays. Es können bis zu acht eigene Zeichen erzeugt werden, die von 0 bis 7 angesprochen werden.

Arduino™-Syntax

```
001 byte myChar[8] = {
002   B00000,
003   B10001,
004   B00000,
005   B00000,
006   B10001,
007   B01110,
008   B00000,
009 }
010
011 void setup()
012 {
013   lcd.createChar(0, myChar)
014   lcd.begin(16, 2)
015   lcd.write(byte(0));
016 }
```

7 LCD-FUNKTIONEN

Das folgende Beispiel fasst die zuletzt erklärten LCD-Funktionen in einem größeren Beispiel zusammen. Sehen Sie sich dazu den Programmcode an und ändern Sie ein paar der eben beschriebenen Parameter, um die Funktionsweise zu verinnerlichen.

Upload
Das Experiment benötigt die LCD-Grundbeschaltung, wie wir sie beim Funktionstest aufgebaut haben.

Beispielcode: FUNCTIONS

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 #define Backlight   9
009
010 int i;
011
012 void setup()
013 {
014     analogWrite(Backlight, 200);
015
016     lcd.begin(16, 2);
017     lcd.setCursor(0, 0);
018     lcd.print("ARDUINO LCD");
019     delay(1000);
020     lcd.clear();
021 }
022
023 void loop()
024 {
025     // Cursor blinken und Position wechseln
026     lcd.clear();
027     lcd.setCursor(0, 0);
028     lcd.print("blink/setCursor");
029     delay(1000);
030     lcd.clear();
031
```

```
032     lcd.setCursor(0, 0);
033     lcd.blink();
034     delay(1500);
035
036     lcd.setCursor(15, 0);
037     delay(1500);
038
039     lcd.setCursor(0, 1);
040     delay(1500);
041
042     lcd.setCursor(15, 1);
043     delay(1500);
044
045
046     // Cursor on/off
047     lcd.noBlink();
048     lcd.clear();
049     lcd.setCursor(0, 0);
050     lcd.print("cursor on/off");
051     delay(1000);
052     lcd.clear();
053     lcd.home();
054     lcd.cursor();
055
056     char txt[6] = {"HALLO"};
057      for(i = 0; i < 5; i++)
058     {
059         lcd.print(txt[i]);
060         delay(500);
061     }
062
063     lcd.noCursor();
064     delay(2000);
065
066
067     // Scroll LCD
068     lcd.clear();
069     lcd.noBlink();
070     lcd.setCursor(0, 0);
071     lcd.print("scroll LCD");
072     delay(1000);
073     lcd.setCursor(0, 0);
074
075     for(i = 0; i < 16; i++)
076     {
077         lcd.scrollDisplayLeft();
```

7

```
078          lcd.setCursor(0, 0);
079          lcd.print("FRANZIS ARDUINO IST MEGA SPITZE!");
080          delay(350);
081      }
082
083      delay(1500);
084
085      for(i = 0; i < 16; i++)
086      {
087          lcd.scrollDisplayRight();
088          lcd.setCursor(0, 0);
089          lcd.print("FRANZIS ARDUINO IST MEGA SPITZE!");
090          delay(350);
091      }
092
093      delay(1500);
094  }
```

Neu ist, dass wir den Pin für die Hintergrundbeleuchtung mit `#define Backlight` angeben. Das ist ein Präprozessor-Kommando, das alle im Quellcode vorkommenden Namen mit der Bezeichnung *Backlight* durch den Wert 9 ersetzt. Sie können somit sehr schnell Änderungen von Parametern durchführen, ohne den ganzen Quellcode durchsuchen zu müssen.

Tipp
Mehr zum Thema Präprozessor finden Sie unter:
http://www.mikrocontroller.net/articles/C-Pr%C3%A4prozessor

Eine Möglichkeit, Zeichen einzeln wie auf einer alten Schreibmaschine auszugeben, zeigt folgende Programmstelle:

```
001     char txt[6] = {"HALLO"};
002      for(i = 0; i < 5; i++)
003     {
004         lcd.print(txt[i]);
005         delay(500);
006     }
```

Hier wird ein Array mit 6 Zeichen angelegt und mit dem String »HALLO« vorbelegt. Wir müssen das Array immer um 1 größer anlegen, da automatisch eine unsichtbare Stringterminierung (\0) angehängt wird.

Die `For()`-Schleife gibt jedes einzelne Zeichen aus dem Array mit einer kurzen Pause aus. Damit das Ganze noch mehr nach Schreibmaschine aussieht, wird der Cursor dazu eingeschaltet.

8 EIGENE ZEICHEN ERSTELLEN

Das Erzeugen eigener Zeichen, wie eben bereits mit `.createChar()` beschrieben, wird oft bei der Verwendung von Dot-Matrix-LCDs benötigt, denn viele in der Praxis benötigte Zeichen sind in der Zeichentabelle des LCD nicht enthalten. Dafür gibt es aber die Möglichkeit, eigene Zeichen Punkt für Punkt zu erstellen und anzuzeigen. Wer z. B. einen Smiley benötigt, kann ihn sich über ein Array definieren und an das LCD senden.

Es beseht die Möglichkeit, bis zu acht eigene Zeichen im RAM (Speicher) des LCD abzulegen. Das Array für unser Sonderzeichen muss 8 Bytes groß sein und wird am besten wie im Beispielcode gezeigt geschrieben. Man kann das Zeichen z. B. schon zuvor auf einem karierten Zeichenblock entwerfen. Sie sehen auch, dass es aus 8 Zeilen zu je 5 Werten besteht, die unsere 5 x 8 Dots des LCD widerspiegeln. Überall dort, wo wir im Binärcode eine 1 setzen, wird später ein weißer Punkt angezeigt. Mit `lcd.write(byte(0))`; schreiben wir das Zeichen auf das LCD.

Das Beispiel verdeutlicht das Ganze noch genauer. Versuchen Sie, ein Akkusymbol oder ein Thermometer zu erzeugen.

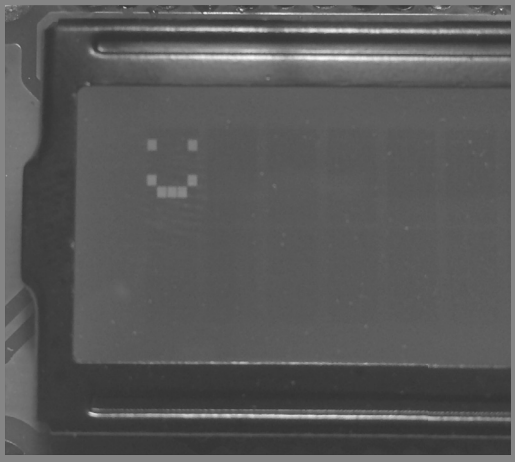

Abb. 6.1: Ein selbst »gezeichneter« Smiley :-]

Upload

Das Experiment benötigt die LCD-Grundbeschaltung, wie wir sie beim Funktionstest aufgebaut haben.

Beispielcode: EIGENE_ZEICHEN

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 byte myChar[8] = {
009   B00000,
010   B10001,
011   B00000,
012   B00000,
013   B10001,
014   B01110,
015   B00000,
016 };
017
018 void setup()
019 {
020     // LED-Hintergrundbeleuchtung
021     analogWrite(9, 150);
022
023     lcd.createChar(0, myChar);
024     lcd.begin(16, 2);
025     lcd.write(byte(0));
026 }
027
028 void loop()
029 {
030
031     // Nix zu tun ...
032
033 }
```

9 BACKLIGHT DIMMEN

Das folgende Experiment zeigt, wie wir die LCD-Beleuchtung automatisch heller oder dunkler einstellen können. Durch das Ändern des PWM-Werts an Pin D9 wird die Helligkeit der Hintergrundbeleuchtung stufenlos verstellt. Wird der PWM-Wert größer gewählt, leuchtet die LED heller. Wird er kleiner gewählt, wird die Beleuchtung dunkler. Indem wir den PWM-Wert ändern, verändern wir das Puls-Pausen-Verhältnis zwischen der Ein- und Ausschaltdauer des 5-V-Signals an D9. Die folgende Grafik verdeutlicht das.

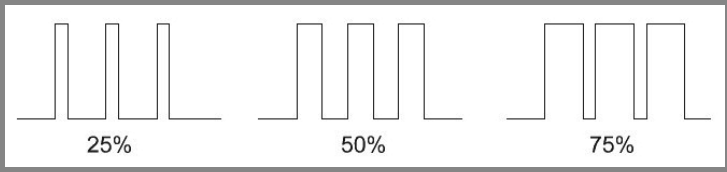

Abb. 9.1: Ein kleiner PWM-Wert bedeutet kürzere Einschaltzeiten und somit eine dunklere Hintergrundbeleuchtung. Je länger die Einschaltzeiten werden, desto heller wird die Beleuchtung. Bei einem niedrigen PWM-Wert sehen wir aber zudem, dass die LED langsam zu flackern beginnt, nur kurz eingeschaltet wird und dann für längere Zeit aus ist.

Übertragen Sie das Programm und beobachten Sie die LED. Das Ganze sieht schon fast so aus, als ob die Anzeige zum Leben erweckt wurde.

Upload

Das Experiment benötigt die LCD-Grundbeschaltung, wie wir sie beim Funktionstest aufgebaut haben.

Beispielcode: LCD_LED

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 #define Backlight   9
009
010 byte i = 0;
011 byte flag = 0;
012 unsigned long previousMillis = 0;
013 const long interval = 10;
014
015 void setup()
016 {
```

```
017     analogWrite(Backlight, 0);
018
019     lcd.begin(16, 2);
020     lcd.setCursor(0, 0);
021     lcd.print("**ARDUINO LCD**");
022 }
023
024 void loop()
025 {
026     unsigned long currentMillis = millis();
027     if(currentMillis - previousMillis >= interval)
028     {
029         if(flag==0)i++;
030         if(flag==1)i--;
031
032         if(i==255)flag=1;
033         else if(i==0)flag=0;
034
035         analogWrite(Backlight, i);
036
037         previousMillis = currentMillis;
038     }
039 }
```

Das Experiment zeigt auch, wie ein automatischer Up/Down-Zähler mit Begrenzung verwirklicht werden kann. Wichtig ist hier, dass die Variable mit dem Namen `flag` einen definierten Startwert von 0 bekommt. Beim Programmstart wird die Variable `i` bis 255 hochgezählt. Wollten wir mit der vollen Helligkeit beginnen, müssten wir die Variable `flag` mit 1 initialisieren und die Variable `i` mit 255.

Bei Erreichen der Zählerwerts `i` von 255 oder 0 wird immer die Variable `flag` von 0 auf 1 bzw. von 1 auf 0 gesetzt und kehrt die Zählerrichtung um (i++ erhört den Zählerstand, inkrementiert um 1, i-- verringert den Zählerstand, dekrementiert um 1). In der `Loop()`-Funktion benutzen wir dieses Mal keine Pause, sondern ermitteln die Zeit über die Funktion `millis()`. Nur wenn eine vorgegebene Differenz, die wir in der Variablen `previousMillis` deklariert haben, verstrichen ist, wird die Helligkeit um eine Stufe geändert. So läuft das Programm außerhalb der If()-Abfrage mit voller Geschwindigkeit weiter. Nur wenn der PWM-Wert verändert wird, erhöht sich die Durchlaufzeit etwas, da einige Funktionen aufgerufen werden, die eine gewisse Abarbeitungszeit benötigen. Hier könnten Sie auch versuchen, die unterschiedlichen Durchlaufzeiten mit der `Millis()`-Funktion zu ermitteln wie Sie es bereits kennengelernt haben.

10 DOT-MATRIX-LCD-UHR

In vielen Anwendungen wird eine Uhr zur Programmsteuerung benötigt – sei es eine einfache Zeitschaltuhr, eine Steuerung, die einen genauen zeitlichen Ablauf einhalten soll, oder ein Betriebsstundenzähler. Die Anwendungen, bei denen eine Uhr benötigt wird, sind zahlreich.

Das Experiment zeigt, wie wir uns eine sehr einfache Uhr selbst programmieren können. Das Programm läuft in der `Loop()`-Funktion endlich und zählt dabei im 10-ms-Takt den Zähler hoch. Ist der Zählerstand cnt = 100, wird die Zeit ausgegeben. Das erfolgt im Sekundentakt. Unsere User-LED *L* blinkt im Sekundentakt mit. Dadurch überwachen wir die Funktion des Programms, ob es auch noch läuft oder sich ein Fehler beim Programmieren eingeschlichen hat. Man bedenke jedoch, dass die Uhr nicht die Präzision einer echten Quarzuhr besitzt, da der Takt und die Abweichung des Mikrocontrollerquarzes mit 16 MHz viel größer sind als bei einem Uhrquarz im Kilohertzbereich (Uhrenquarz = 32,768 kHz). Abweichungen von mehr als einer Minute pro Tag sind keine Seltenheit. Die Genauigkeit ist zudem stark von der Umgebungstemperatur abhängig. Schwankt diese im Lauf der Zeit stark, wird auch die Uhr größere Zeitfehler aufweist. Wir können die Zeitabweichung mit `delay()` korrigieren. Alternativ bietet sich auch `delayMicroseconds()` an, um die Abweichung noch genauer zu korrigieren. Dazu sollten Sie einen digitalen Pin als Ausgang konfigurieren und ihn bei jedem Programmdurchlauf toggeln, also bei jedem Durchlauf einmal den Zustand wechseln. Dieses Signal können Sie mit einem Oszilloskop genau auf 10 ms Durchlaufzeit trimmen. Sie können auch die Abweichung über eine längere Zeit ermitteln, indem Sie die Zeit mithilfe einer anderen genauen Uhr, z. B. einer DCF-Uhr eine Weile (1 bis 2 Tage) vergleichen, die Differenz berechnen und dann die Abweichung mit `delayMicroseconds()` korrigieren.

Abb. 10.1: Die Uhr im Betrieb

Mit diesen Variablen stellen Sie die Uhr ein:

```
001 Sekunde = 12
002 Minute = 0
003 Stunde = 0
```

Tipp
Mehr zum Thema Uhrenquarz finden Sie unter: *http://de.wikipedia.org/wiki/Uhrenquarz*

Upload
Das Experiment benötigt die LCD-Grundbeschaltung, wie wir sie beim Funktionstest aufgebaut haben.

Beispielcode: RTC

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 #define Backlight   9
009
010 int cnt, Sekunde, Minute, Stunde=0;
011 int LED=13;
012
013 void setup()
014 {
015       pinMode(LED, OUTPUT);
016       analogWrite(Backlight, 200);
017       lcd.begin(16, 2);
018
019       // Zeitvorgabe
020       Sekunde = 12;
021       Minute = 0;
022       Stunde = 0;
023 }
024
025 void loop()
026 {
027
028   cnt++;
```

10

```
029   if(cnt == 50)digitalWrite(LED, LOW);
030
031   if(cnt == 100)
032   {
033       digitalWrite(LED, HIGH);
034
035       lcd.setCursor(3, 0);
036
037     if(Stunde < 10) lcd.print("0");
038     lcd.print(Stunde);
039     lcd.print(":");
040
041     if(Minute < 10) lcd.print("0");
042     lcd.print(Minute);
043     lcd.print(":");
044
045     if(Sekunde < 10) lcd.print("0");
046     lcd.print(Sekunde);
047
048     Sekunde++;
049     if(Sekunde == 60)
050     {
051         Sekunde = 0;
052         Minute++;
053         if(Minute == 60)
054         {
055             Minute = 0;
056             Stunde++;
057             if(Stunde == 24)
058             {
059                 Stunde = 0;
060             }
061         }
062     }
063     cnt = 0;
064   }
065
066   delay(10);
067 }
```

Würden wir die Zählerstände eins zu eins auf das Display ausgeben, wäre die Darstellung bei Zählerständen kleiner 10 etwas seltsam, da die führende Null nicht angezeigt würde. Damit die Uhr die bekannte »00:00:00«-Formatierung erhält, prüfen wir vor der Ausgabe, ob der Wert kleiner als 10 ist.

```
001 if(Sekunde < 10) lcd.print("0")
```

Ist der Wert kleiner als 10, wird mit einer einfachen Ausgabe nur z. B. »12:1:8« angezeigt werden. Wir prüfen jedoch, ob der Wert kleiner ist, und fügen ggf. eine »0« manuell hinzu, um die Zehnerstelle aufzufüllen.

11 KAPAZITÄTS-MESSGERÄT

Messgeräte mit geringsten Mitteln selbst zu bauen ist immer interessant und spannend. Mit Arduino™ können wir mit geringsten Kosten und Aufwand ein Kapazitätsmessgerät für kleine Kondensatoren im Bereich von 1 nF bis zu ca. 100 µF für unser Hobbylabor programmieren. So funktioniert unser Kapazitätsmesser mit Autorange-Funktion:

Zu Beginn der Messung wird die Variable `C_time` mit Null initialisiert. Der Port D12 wird als Ausgang konfiguriert und danach sofort auf LOW (GND) geschaltet, um den angeschlossenen Kondensator (Prüfling) vor der eigentlichen Messung zu entladen.

Nach einer kurzen Endladepause von 1 Sekunde wird der Port D12 als Eingang konfiguriert und der interne Pull-up-Widerstand aktiviert. Der Pull-up-Widerstand lädt nun den zu prüfenden Kondensator so weit auf, bis der Port D12 ein HIGH erkennt. Die Schwelle, ab wann der Digitalport ein HIGH erkennt, liegt bei ca. 3,5 V bei einer Betriebsspannung von 5 V. Dieser Pegel ist somit von der Betriebsspannung abhängig und wird im Datenblatt zum Mikrocontroller mit Vcc x 0,7 angegeben.

```
001 HIGH = 5V x 0,7
```

Bis ein HIGH-Pegel am Digitalport erkannt wird, verstreicht eine gewisse Zeit, die wir innerhalb der Do-while-Schleife mit der Variablen `C_time` messen. `C_time` ist annähernd proportional zur Kapazität des Kondensators, d. h., ist `C_time` sehr groß, ist auch die zu messende Kapazität sehr groß.

Um nun einen richtigen Messwert zu erhalten, müssen wir die Variable noch umrechen (`C_time` • `Faktor`). Der Wert (Faktor) muss experimentell mit »ein paar Kalibrierkondensatoren« ermittelt werden, da die Erkennung eines HIGH-Pegels von Controller zu Controller trotz der Angabe Vcc x 0,7 im Datenblatt leicht unterschiedlich ausfällt und die Oszillatorfrequenz von 16 MHz nicht zu 100 % bei jedem Board gleich ist (Quarztoleranzen). Zu guter Letzt wird der Messwert noch in Nanofarad (nF) und Mikrofarad (µF) mit einer einfachen If()-Abfrage aufgeteilt und auf dem LCD ausgegeben, bevor wieder eine neue Messung startet.

11.1 | Aufbau des Kondensatormessgeräts

An Pin D12 und GND wird der Kondensator eingesteckt. Achten Sie darauf, dass Sie den Kondensator vor dem Einstecken in die Arduino™-Platine zusätzlich entladen, indem Sie die beiden Drähte des Kondensators zusammenhalten. Auch wenn das Programm den Kondensator vor Beginn der Messung entlädt, kann es vorkommen, dass der von Ihnen ausgewählte Kondensator zuvor mit einer höheren Spannung als 5 V betrieben wurde. Das kann dazu führen, dass Ihr Arduino™-Board beschädigt wird.

11.2 | So kalibrieren Sie Ihr Kondensatormessgerät

Besorgen Sie sich ein paar neue Kondensatoren – auf jeden Fall sollten Sie Kondensatoren verwenden, bei denen Sie wissen, welche Kapazität genau in ihnen steckt. Evtl. können Sie sie bei einem Elektroniker im Labor vermessen lassen. Neue Kondensatoren haben aber je nach Typ meist den aufgedruckten Wert +/- 20 %.

Bei dieser Messung ist als Faktor 1 (`c_time = 1.0`) einzutragen. Stecken Sie einen dieser Kondensatoren in das Messgerät und lesen Sie den Wert nach dem Übertragen am Terminal ab. Dann teilen Sie den Kapazitätswert des eingesteckten Kondensators durch das Messergebnis, das auf dem LCD erscheint, und tragen das Ergebnis als Faktor ein, z. B. 1 µF / 19,55 µF = 0,0511.

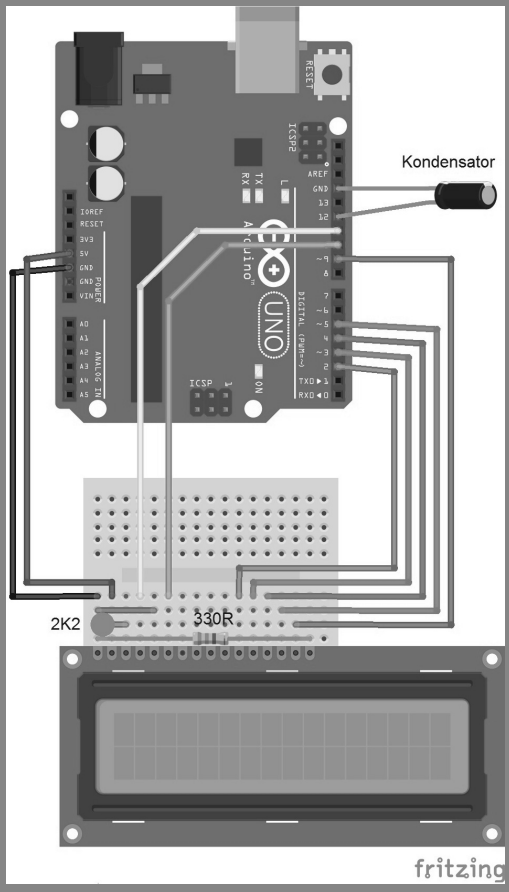

Abb. 11.1: Aufbau den Kondensatormessgeräts

11

Beispielcode: CAPA

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 #define Backlight   9
009
010 int messPort=12;
011 float c_time=0.0;
012 float kapazitaet=0.0;
013
014 void setup()
015 {
016     analogWrite(Backlight,200);
017
018     lcd.begin(16, 2);
019     lcd.setCursor(0,0);
020     lcd.print("C-MESSGERAET");
021 }
022
023 void loop()
024 {
025       // Entladen
026       pinMode(messPort,OUTPUT);
027       digitalWrite(messPort,LOW);
028       c_time=0.0;
029       delay(1000);
030
031       // Laden
032       pinMode(messPort,INPUT);
033       digitalWrite(messPort,HIGH);
034
035       // Messen
036       do
037       {
038         c_time++;
039       }while(!digitalRead(messPort));
040
041       // Umrechnen
042       kapazitaet=(c_time*0.06162)*10.0;
043
044       // Bereich
```

```
045         if(kapazitaet<999)
046         {
047             lcd.setCursor(0,1);
048             lcd.print(kapazitaet);
049             lcd.print("nF    ");
050         }
051         else
052         {
053             lcd.setCursor(0,1);
054             lcd.print(kapazitaet/1000);
055             lcd.print("uF    ");
056         }
057 
058         delay(1000);
059 }
```

12 ZUFALLSZAHLEN – DER LOTTOZAHLENGENERATOR

Beim Schreiben von Mess-, Steuer-, Regel- oder Spielprogrammen ist es oft von Nutzen, Zufallszahlen zu generieren, z. B. wenn in einem Haus zu unterschiedlichsten Zeiten die Lichter an- und ausgehen sollen, um einen Anwesenheitssimulator zu programmieren. Für diesen Zwecke kann man die Arduino™-random()-Funktion verwenden. So kann man auch auf einfache Weise einen Lottozahlengenerator aufbauen, der 6 aus 49 für Sie zieht und Ihnen das lästige Nachdenken abnimmt, welche Zahlen Sie nehmen sollen, wenn es gilt, den Lottoschein auszufüllen.

Für den Aufbau des Lottozahlengenerators benötigen Sie einen Taster und eine Antenne. Der Taster wird in der Software entprellt. Taster und Schalter neigen dazu, den Kontakt nicht sofort zu 100 % zu schließen, sondern nach dem Drücken noch mehrfach auszulösen. Das ist vergleichbar damit, wenn man einen Ball auf den Boden wirft. Er wird zuerst ein paar Mal hochspringen, bis er endgültig auf dem Boden liegen bleibt. Bei einen Taster geht das zwar deutlich schneller, aber dafür ist der Arduino™-Mikrocontroller so schnell, dass er auch diese im Millisekundenbereich liegenden »Hüpfer« noch erfasst. Um das zu vermeiden, wird der Taster entprellt, indem er zweimal hintereinander mit einer Pause von 50 ms abgefragt wird, was in der Praxis für eine Entprellroutine reicht. Erst wenn bei der zweiten Auswertung noch immer ein LOW am Eingang D7 erkannt wird, wird die Anweisung zwischen den Klammern ausgeführt.

Zu Beginn des Programms schalten wir den Port D7 auf INPUT und aktivieren den internen Pull-up-Widerstand, indem wir mit `digitalWrite()` eine 1 für HIGH auf den Eingang schreiben. Nun liegt im Ruhezustand eine Spannung von ca. 5 V am Eingang an. Jetzt können Sie den Eingang mit dem Taster gegen GND (Masse) ziehen. Dieser Pull-up-Widerstand ist im Mikrocontroller integriert und besitzt einen Wert von ca. 20 bis 50 kΩ. Von der Funktion her ist es das Gleiche, als wenn Sie einen externen Widerstand vom Eingang D7 auf +5 V legen würden. Im Ruhezustand wird somit im Programm immer ein HIGH am Eingang D7 erkannt und bei gedrücktem Taster ein LOW. Deshalb ist im Programm die Tasterabfrage auch mit einen Ausrufezeichen versehen. Das bezeichnet man in

der C-Programmierung als *NOT-Operator*. Da bekanntlich eine If()-Abfrage auf TRUE prüft, wird alles andere auf FALSE interpretiert. Würde ohne diesen Operator die If()-Abfrage durchgeführt, wäre die Bedingung immer TRUE, also »wahr«. Sie würde dann bereits ausgeführt, wenn der Taster noch gar nicht von Ihnen gedrückt wurde. Mit dem NOT-Operator kehren wir den Status des Tasters um. Aus 1 wird 0 und aus 0 wird 1 und die If()-Abfrage ist nun erst TRUE, wenn der Taster auch tatsächlich gedrückt wurde.

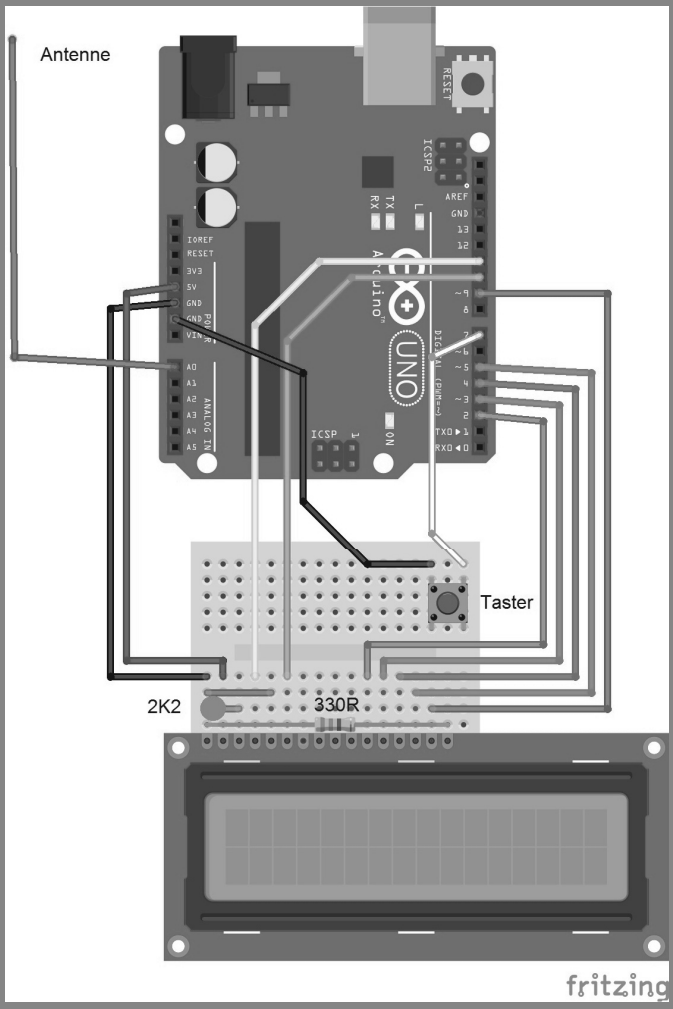

Abb. 12.1: Aufbau des Lottozahlengenerators; als Antennen können Sie eine Steckbrücke verwenden. Drücken Sie den Taster, um Ihre Gewinnzahlen zu ermitteln.

Abb. 12.2: Ihre »Gewinnzahlen« für kommenden Samstag

Beispielcode: LOTTO

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 #define Backlight   9
009 #define TASTER      7
010
011 int i, zahl=0;
012 int Anz = 6;
013
014 void setup()
015 {
016     analogWrite(Backlight, 200);
017
018     pinMode(TASTER, INPUT);
019     digitalWrite(TASTER, HIGH);
020
021     lcd.begin(16, 2);
022     lcd.setCursor(0, 0);
023     lcd.print("LOTTO 6 aus 49");
024     delay(1000);
025     lcd.clear();
026
027     randomSeed(analogRead(0)*5);
028
029 }
030
031 void loop()
032 {
033     lcd.clear();
034     lcd.setCursor(0, 0);
035     lcd.print("TASTER DRUECKEN");
036
037     while(digitalRead(TASTER));
038     {
```

```
039         if(!digitalRead(TASTER))
040         {
041             delay(50);
042             if(!digitalRead(TASTER));
043             {
044                 lcd.clear();
045                 lcd.setCursor(0, 0);
046                 lcd.print("IHRE LOTTOZAHLEN");
047
048                 lcd.setCursor(0, 1);
049                 for(i=0;i<Anz;i++)
050                 {
051                    zahl=random(49);
052                    zahl++;
053                    lcd.print(zahl);
054                    lcd.print(" ");
055                    delay(500);
056                 }
057
058                 delay(5000);
059             }
060         }
061     }
062 }
```

Zum Programmstart wird mit `randomSeed()` ein Wert als Ausgangspunkt für die `Random()`-Funktion generiert. Wenn man den Wert von `randomSeed()` ändert, werden immer andere Zufallszahlen erzeugt. Wäre der Wert von `randomSeed()` hingegen beim Programmstart immer gleich, würden auch immer die gleichen Zufallszahlenreihen erzeugt, was zum Lottospielen nicht gerade günstig wäre.

Hier kommt unsere Antenne zum Einsatz. Wir verwenden zum Erzeugen unterschiedlicher Werte mit der Funktion `randomSeed()` einen ADC-Eingang, der mit einer Steckbrücke verbunden ist und auf der anderen Seite offen bleibt. Das wirkt wie eine Antenne und erzeugt ein höheres Rauschen am Analogeingang und somit einen immer anderen Wert für `randomSeed()`. Am besten funktioniert das, wenn Sie Ihre Hand neben die Antenne halten oder die Antenne in die Nähe von elektrischen Geräten bringen. Das Ergebnis mit unterschiedlichen Zahlenreihen sehen Sie erst, wenn Sie den Resettaster auf der Arduino™-Platine drücken und dann die Zahlen ausgeben lassen. Sie können einmal statt analogRead() eine fixe Zahl eingeben, und Sie werden sehen, dass nach einem Reset immer die gleichen Zahlenfolgen erscheinen.

13 BARGRAFANZEIGE

Bargrafanzeigen werden oft in der Messtechnik verwendet. Man bezeichnet sie auch als Balkenanzeige. Sie stellen einen visuellen/tendenziellen Messwert dar. Bei Einstellarbeiten an elektronischen Schaltungen ist eine Bargrafanzeige eine deutliche Erleichterung, da hier die Tendenz auf max. oder min. einfacher abzulesen ist, als das mit der digitalen Zahlenwertanzeige der Fall ist. Auch bei Computerprogrammen kennen wir diese Anzeige als Fortschrittsanzeige, z. B. bei der Installation eines Programms. Hier wird über den Balken angezeigt, wie weit die Installation bereits fortgeschritten ist. Im Grunde handelt es sich bei der im Experiment gezeigten Bargrafanzeige um eine analoge Anzeige auf digitaler Basis. Bereits in der früheren Elektronik wurden elektromechanische Bargrafanzeigen verwendet. Unsere Bargrafanzeige verwendet jedoch ein modernes LCD und einen Mikrocontroller.

Im einfachsten Fall könnten wir für jeden Anzeigeschritt ein ganzes Zeichen einblenden (5 x 8 Dots). Wir können dann aber nur eine sehr grobe Anzeige von 0 bis 16 realisieren. Schöner wäre es, wenn Sie die einzelnen fünf Spalten jedes einzelnen Zeichens nutzen könnten. Da wir acht eigene Zeichen generieren können, ist es ein Leichtes, eine Bargrafanzeige mit 5 x 16 = 80 Zeichen/Zuständen zu programmieren.

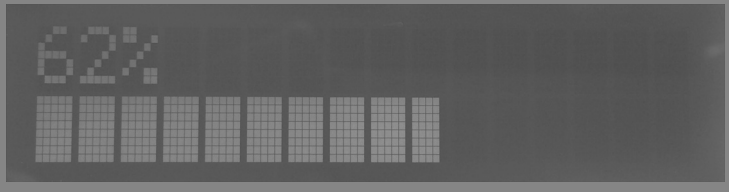

Abb. 13.1: Die Bargrafanzeige in Aktion

Upload

Das Experiment benötigt die LCD-Grundbeschaltung, wie wir sie beim Funktionstest aufgebaut haben.

Beispielcode: BARGRAPH

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
```

```
007
008 #define LCD_LENGHT 16.0
009
010 int value;
011 byte flag = 0;
012
013 byte MyChar0[8] = {
014 B00000,
015 B00000,
016 B00000,
017 B00000,
018 B00000,
019 B00000,
020 B00000,
021 B00000,
022 };
023
024 byte MyChar1[8] = {
025 B10000,
026 B10000,
027 B10000,
028 B10000,
029 B10000,
030 B10000,
031 B10000,
032 B10000,
033 };
034
035 byte MyChar2[8] = {
036 B11000,
037 B11000,
038 B11000,
039 B11000,
040 B11000,
041 B11000,
042 B11000,
043 B11000,
044 };
045
046 byte MyChar3[8] = {
047 B11100,
048 B11100,
049 B11100,
050 B11100,
051 B11100,
052 B11100,
```

13

```
053 B11100,
054 B11100,
055 };
056
057 byte MyChar4[8] = {
058 B11110,
059 B11110,
060 B11110,
061 B11110,
062 B11110,
063 B11110,
064 B11110,
065 B11110,
066 };
067
068 byte MyChar5[8] = {
069 B11111,
070 B11111,
071 B11111,
072 B11111,
073 B11111,
074 B11111,
075 B11111,
076 B11111,
077 };
078
079 void draw_bargraph(byte percent)
080 {
081   byte i, c1, c2;
082
083   lcd.setCursor(0, 0);
084   lcd.print(percent);
085   lcd.print("%  ");
086
087   lcd.setCursor(0, 1);
088
089   percent = map(percent, 0, 100, 0, 80);
090
091   c1 = percent / 5;
092   c2 = percent % 5;
093
094   for(i = 0; i < c1; ++i)
095   {
096     lcd.write(byte(5));
097     lcd.write(c2);
098   }
```

```
099
100   for(i = 0; i < 16 - (c1 + (c2 ? 1 : 0)); ++i)
101   {
102     lcd.write(byte(0));
103   }
104 }
105
106 void setup()
107 {
108     analogWrite(9,200);
109
110     lcd.createChar(0, MyChar0);
111     lcd.createChar(1, MyChar1);
112     lcd.createChar(2, MyChar2);
113     lcd.createChar(3, MyChar3);
114     lcd.createChar(4, MyChar4);
115     lcd.createChar(5, MyChar5);
116
117     lcd.begin(16, 2);
118 }
119
120 void loop()
121 {
122     double percent;
123
124     if(flag == 0)value++;
125     if(flag == 1)value--;
126     if(value > 1024)flag=1;
127     else if(value == 0)flag=0;
128
129     percent = value / 1024.0 * 100.0;
130     draw_bargraph(percent);
131     delay(10);
132 }
```

Die einzelnen Segmente der Bargrafanzeige werden mit den Arrays `MyChar0` bis `MyChar5` vorgegeben. Man sieht bereits im Programmcode, wie sich die Segmente aufsteigend Array für Array mit Einsen auffüllen. In der `Setup()`-Funktion werden die Zeichen mit `lcd.createChar()` erzeugt.

In der `Loop()`-Funktion wird wieder ein Up/Down-Zähler verwendet, den wir aus vorhergehenden Experimenten kennen. Dieser zählt jedoch dieses Mal von 0 bis 1.024. So würde sich der Zähler leicht durch einen Arduino™-Analogeingang ersetzen lassen, der einen Wertebereich von 0

13

bis 1.023 abdeckt. Der Zählerwert wird in Prozent umgerechnet und der Funktion `draw_bargraph()` übergeben.

In der `draw_bargraph()`-Funktion wird es spannend. Hier erfolgt das Zusammensetzen und Anzeigen der Bargrafanzeige. Bei jedem Aufruf der Funktion wird der Cursor an die Position (0, 0) gesetzt. Das ist die Startposition, um die Anzeige neu zu zeichnen. In der ersten Zeile geben wir den prozentualen Wert der Variablen `percent` aus und schreiben ein Prozentzeichen dahinter. Hinter dem Prozentzeichen folgen zwei Leerzeichen, um die Anzeige bei einem Zehner oder Hunderterversatz zu bereinigen. Die digitale Werteausgabe in Prozent ist damit schon mal fertig.

Dann setzen wir den Cursor mit `setCursor(0, 1)` in die untere, also die zweite Zeile des LCD. Damit wir den prozentualen Wert, der von 0 bis 100 % reicht, auf die 80 Einzelzeichen (Pixel) aufteilen können, verwenden wir die `Map()`-Funktion. Diese skaliert den Eingangswert `percent`, der von 0 bis 100 reicht, auf den Ausgangswert von 0 bis 80. In der Variablen `percent` steht danach ein Wert zwischen 0 und 80, abhängig vom Eingangswert `percent`.

Nun ermitteln wir, indem wir den Wert von `percent` durch 5 dividieren, wie viele Kästen komplett gefüllt werden müssen, und schreiben das Ergebnis in die Variable `c1`. Mithilfe der Modulo-Operation % ermitteln wir den Rest oder die Teilfüllung und schreiben den Wert in die Variable `c2`. Nun wissen wir, wie viele Kästen komplett gefüllt werden müssen und wie viele nur zum Teil gefüllt sind.

In der darauf folgenden `For()`-Schleife zählen wir so weit hoch, bis alle ausgefüllten Kästchen erreicht sind, und schreiben bei jedem Schleifendurchlauf ein volles Kästen auf das LCD. Da bei jeder aufeinanderfolgenden Ausgabe am LCD die Zeichen automatisch um eins weitergeschoben werden, erhalten wir am Ende der Schleife einen Balken mit vollen Kästen. Wir sehen an dieser Programmstelle auch, dass keine geschweiften Klammern bei der `For()`-Schleife verwendet werden. Der Compiler nimmt für die `For()`-Schleife nur die darauffolgende Zeile mit in die Schleife auf. Das wäre in diesem Fall der Aufruf von `lcd.write(byte(5))`. Nach Beenden dieser ersten Schleife wird das Kästchen mit der Teilfüllung auf das LCD geschrieben. So erhalten wir einen Balken, der sich Pixel für Pixel auf dem LCD aufbaut.

Beispiel

Es soll der prozentuale Wert 43 dargestellt werden. Wir dividieren die Zahl 43 durch 5, das ergibt 8,6. Da die Variable c1 als Byte deklariert ist, wird

in ihr nur eine 8 abgespeichert. Modulo 5 aus 43 ergibt 3, was drei Teilstriche bedeutet.

Wenn der Wert wieder kleiner wird, müssen wir die überflüssigen Zeichen vom LCD löschen. Es kommt eine neue Operation hinzu, die sich *bedingter Ausdruck* oder auch *ternärer Auswahloperator* nennt.

Im Grunde können wir das Ganze wie eine If-Else-Anweisung betrachten, es stellt lediglich eine verkürzte C-Schreibweise dar.

Der Syntax der bedingten Anweisung wäre: Bedingung ? Ausdruck1 : Ausdruck2

Das kommt Ihnen bekannt vor oder? Im Grunde ist es nichts anderes als:

```
001 if(Bedingung)
002 {
003       // Ausdruck1
004 }
005 else
006 {
007       // Ausdruck2
008 }
009
010 for(i = 0; i < 16 - (c1 + (c2 ? 1 : 0)); ++i)
011 lcd.write(byte(0))
```

Die Schleife löscht die überflüssigen Zeichen aus dem LCD, indem sie ermittelt, wie lang der nicht verwendete Bereich ist, und ihn mit Leerzeichen überschreibt.

Ein wenig Know-how steckt durchaus in der Bargrafanzeige, aber einmal verstanden, lässt sie sich in vielen Applikationen einfach anwenden.

14 LICHTMESSER – DAS FOTOMETER

Ein Fotometer ist ein Messgerät zum Ermitteln der Leuchtdichte oder Lichtstärke. Es wird z. B. bei Fotografen als Belichtungsmesser verwendet oder in der Astronomie zur Ermittlung der Helligkeit der Himmelskörper eingesetzt.

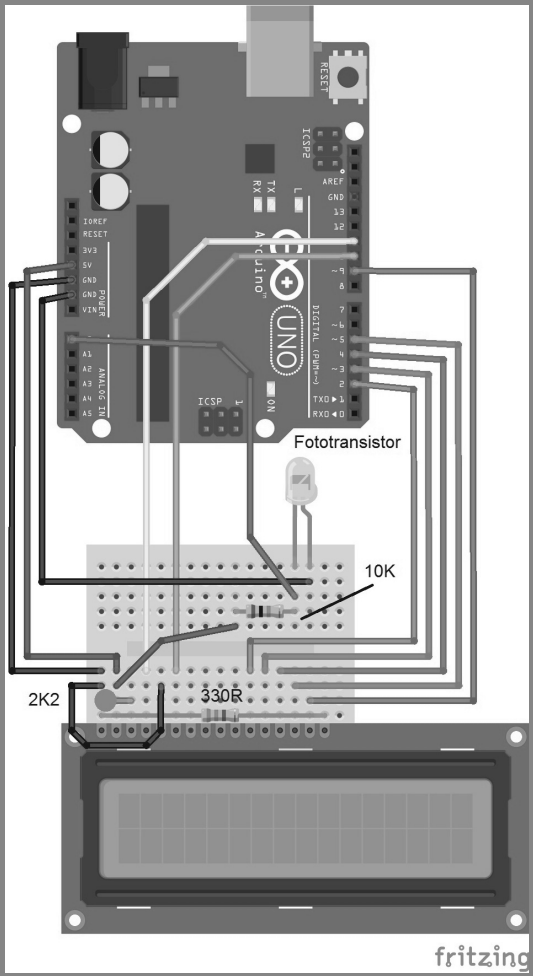

Abb. 14.1: Aufbau des Fotometers; die LCD-Beschaltung ist so aufgebaut, wie sie bereits in der Grundschaltung verwendet wird. Die Schaltung wurde um einen Fototransistor und einen 10-kΩ-Widerstand erweitert.

In der Chemie wird es zur Bestimmung von Konzentrationen verwendet. Haben wir im letzten Experiment eine Bargrafanzeige programmiert, können wir nun statt des Up/Down-Zählers eine echte physikalische Größe übergeben und uns einen einfachen Lichtmesser programmieren.

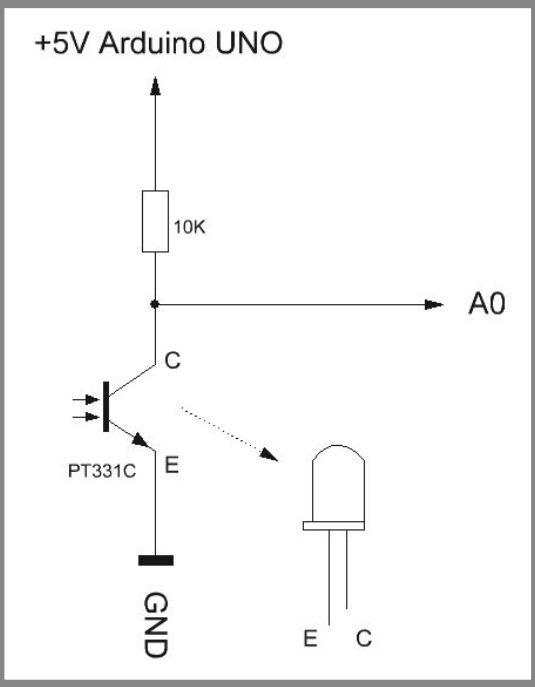

Abb. 14.2: Schaltplan zum Betrieb eines Fototransistors des Typs PT331C am Arduino™-Board

Der Schaltplan zeigt, wie der Fototransistor am Analogeingang (ADC) des Arduino™-Boards angeschlossen ist. Der Fototransistor ist nicht leicht von einer normalen LED zu unterscheiden. Er besitzt zwar ein klares durchsichtiges Gehäuse, aber das besitzen manche LEDs auch. Wenn Sie sich nicht sicher sind, ob es sich um eine LED oder einen Fototransistor handelt, können Sie das Verhalten mit einem Multimeter, das auf Widerstandsmessung eingestellt ist, ermitteln. Klemmen Sie dazu den Kollektor (C – abgeflachte Seite des Gehäuses) an die Plusleitung und den Emitter (E) an die Minusleitung des Messgeräts. Wenn Sie nun einen Fototransistor verdunkeln, ändert sich der Widerstandswert enorm, bei einer LED hingegen ist die Auswirkung nur sehr gering. Ideal sind hier Multimeter mit Autorange-Funktion, da sich der Messwert zwischen wenigen Kiloohm und einigen Megaohm befindet.

14

Beispielcode: PHOTOMETER

```
// LCD-Library einbinden
#include <LiquidCrystal.h>

// LCD-Pins festlegen
// RS, E, D4, D5, D6, D7
LiquidCrystal lcd(11, 10, 2, 3, 4, 5);

#define LCD_LENGHT 16.0

const int numReadings = 10;
unsigned int readings[numReadings];
int index = 0;
unsigned int total = 0;
int average = 0;

byte MyChar0[8] = {
B00000,
B00000,
B00000,
B00000,
B00000,
B00000,
B00000,
B00000,
};

byte MyChar1[8] = {
B10000,
B10000,
B10000,
B10000,
B10000,
B10000,
B10000,
B10000,
};

byte MyChar2[8] = {
B11000,
B11000,
B11000,
B11000,
B11000,
B11000,
B11000,
B11000,
```

```
};

byte MyChar3[8] = {
B11100,
B11100,
B11100,
B11100,
B11100,
B11100,
B11100,
B11100,
};

byte MyChar4[8] = {
B11110,
B11110,
B11110,
B11110,
B11110,
B11110,
B11110,
B11110,
};

byte MyChar5[8] = {
B11111,
B11111,
B11111,
B11111,
B11111,
B11111,
B11111,
B11111,
};

int adc_AVG(byte channel)
{
    total= total - readings[index];
    readings[index] = analogRead(channel);
    total= total + readings[index];
    index = index + 1;

    if (index >= numReadings)index = 0;

    average = total / numReadings;
    return average / 1024.0 * 100.0;
}
```

14

```
094
095 void draw_bargraph(byte percent)
096 {
097     byte i, c1, c2;
098
099     lcd.setCursor(0, 0);
100     lcd.print("Brightness: ");
101     lcd.print(percent);
102     lcd.print("%  ");
103
104     lcd.setCursor(0, 1);
105
106     percent = map(percent, 0, 100, 0, 80);
107
108     c1 = percent / 5;
109     c2 = percent % 5;
110
111     for(i = 0; i < c1; ++i)
112     lcd.write(byte(5));
113
114     lcd.write(c2);
115
116     for(i = 0; i < 16 - (c1 + (c2 ? 1 : 0)); ++i)
117     lcd.write(byte(0));
118 }
119
120 void setup()
121 {
122     analogWrite(9,200);
123
124     lcd.createChar(0, MyChar0);
125     lcd.createChar(1, MyChar1);
126     lcd.createChar(2, MyChar2);
127     lcd.createChar(3, MyChar3);
128     lcd.createChar(4, MyChar4);
129     lcd.createChar(5, MyChar5);
130
131     lcd.begin(16, 2);
132 }
133
134 void loop()
135 {
136     int raw_adc = adc_AVG(0);
137     draw_bargraph(100 - raw_adc);
138     delay(20);
139 }
```

Wenn Sie die Schaltung aufgebaut und den Beispielcode auf das Arduino™-Board übertragen haben, wird bei heller Umgebung ein Wert nahe der 100 % auf dem LCD angezeigt und der Balken des Bargrafen füllt die untere Zeile des LCD fast aus.

Verdunkeln Sie nun den Fototransistor, wird der Wert niedriger bis nahe Null. Sehen Sie sich dazu die Schaltung genauer an. Der Fototransistor ist am Kollektor mit dem 10-kΩ-Widerstand verbunden, der auf +5 V liegt, und am Emitter mit GND (Masse) verbunden. Der Analogeingang 0 der Arduino™-Platine ist mit dem Knotenpunkt des Kollektors und des Widerstands verbunden. Wird der Fototransistor nun angeleuchtet, wird er leitend und der Spannungsabfall zwischen Kollektor und Emitter sinkt. Wir messen also eine sehr kleine Spannung. Wird der Fototransistor abgedunkelt, fließt fast kein Strom, der Fototransistor sperrt und die Kollektor-Emitter-Spannung steigt an. Wir messen jetzt fast die kompletten 5 V.

Zwischen den beiden Extremwerten verhält sich der Fototransistor sehr dynamisch und reagiert auch auf kleinste Lichtschwankungen. Da so die Anzeige genau verkehrt herum funktionieren würde – also sehr hell wäre ein kleiner Wert und dunkel ein sehr großer –, müssen wir den Messwert anpassen. Dazu subtrahieren wir den Messwert von 100 %, um das gewünschte Ergebnis zu erhalten. Wir invertieren also unseren analogen Messwert.

Damit die Bargrafanzeige bei kleinsten Lichtänderungen nicht zu sehr »zappelt", wurde der Bargrafanzeigefunktion eine Mittelwertbildung hinzugefügt. Sie glättet die analogen Messwerte und berechnet daraus einen gleitenden Mittelwert, auch AVG (engl. Avarage) genannt.

Dazu wird bei jeden Durchlauf der aktuelle Messwert einem Array hinzuaddiert und, je nachdem, wie hoch der Zählerstand in dieser Funktion gerade ist, durch ihn dividiert. Damit erhalten wir immer den aktuellen Mittelwert. Die Anzahl der Messreihen zur Mittelwertbildung wird mit der Variablen `numReadings` festgelegt. Je höher die Anzahl der zu mittelnden Werte ist, desto genauer, aber auch träger wird die Anzeige. Hier können Sie mit den Werten experimentieren. Als sinnvoll haben sich Werte zwischen 8 und 64 erwiesen. Am Ende der AVG-Funktion wird der Eingangswert (0 bis 1.023) noch auf einen Prozentwert für die Bargraffunktion berechnet.

Solch ein Lichtmesser kann auch zum automatischen Ein- und Ausschalten einer Beleuchtung umprogrammiert oder, wie im nächsten Experiment gezeigt, zu einer Alarmanlage umfunktioniert werden.

15 ALARMANLAGE

Das Fotometer lässt sich auch als Alarmanlage verwenden, die schon auf kleinste Lichtänderungen reagiert. Zu Beginn des Alarmanlagenprogramms wird die aktuelle Lichtstärke am Analogeingang A0 ermittelt, der als Referenzpunkt für die Messung dient. Wird der Spannungswert bei der fortlaufenden Messung durch eine Lichtänderung (z. B. eine vorbeigehende Person) größer oder kleiner und somit die vorgegebene Schwelle unter- oder überschritten, löst der Alarm aus.

Da sich die Helligkeit im Raum bedingt durch den Tagesverlauf ändert, wird alle 10 Sekunden automatisch ein neuer Referenzwert (aktuelle Spannung des Fotometers) ermittelt, der als neuer Anhaltspunkt für die fortlaufende Messung dient.

Wir vergleichen somit einen festen Lichtwert, der nur alle 10 Sekunden neu gemessen wird, mit dem Lichtwert der fortlaufenden Messung. Durch die definierte Schwelle wird der Alarm nur dann ausgelöst, wenn der aktuelle Lichtwert +/- die Schwelle über- oder unterschreitet.

```
001 cnt++;
002 if(cnt > 2000)
003 {
004     cnt = 0;
005     value = analogRead(PHOTOTRANSITOR)
006 }
```

Ist die Variable `cnt` größer 2.000, wird ein neuer Wert vom Analogeingang A0 gelesen. Durch eine eingefügte Pause, mit der die Programmdurchlaufgeschwindigkeit beeinflusst wird, wird nur alle 5 ms die Variable `cnt` um 1 inkrementiert. Daraus ergibt sich ein Wert von 5 ms x 2.000 = 10.000 ms = 10 Sekunden.

```
001 Schwelle = 25
002 if(analogRead(PHOTOTRANSITOR) > (value + Schwelle) ||
          analogRead(PHOTOTRANSITOR) < (value - Schwelle))
```

Hier wird der Wert der Variablen mit dem Namen `Schwelle` eingestellt. Sie ist für die Auslöseempfindlichkeit zuständig und sollte nicht zu hoch und nicht zu niedrig eingestellt werden, da sonst die Alarmanlage fast gar nicht reagiert oder zu viele Fehlalarme auslöst. Die Auswertung des

fortlaufenden Messwerts, der alle 5 ms aufgenommen wird, wird in diesen Programmzeilen vorgenommen.

Zum Testen können Sie in einem normal beleuchteten Zimmer mit der Hand in einer Entfernung von ca. 50 cm über den Fototransistor streifen und der Alarm wird auslösen. Das Ganze können Sie auch sehr schnell durchführen. Der Detektor wird Sie sofort erfassen, solange die Verdunkelung durch Ihre Hand mehr als 5 ms beträgt.

Man könnte die Alarmanlage auch in den Kühlschrank legen und zur Alarmmeldung eine Zählervariable hinzufügen, um damit die Öffnungszyklen des Kühlschranks zu erfassen. Sie werden bei dem Kühlschrankexperiment feststellen, dass die LCD-Anzeige ihren Kontrast ändert und zudem sehr träge wird. Probieren Sie es einfach einmal aus.

Upload
Wir verwenden denselben Aufbau wie beim Fotometerlichtmesser!

Beispielcode: ALARM

```
001  // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003 
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007 
008 #define Backlight   9
009 
010 int PHOTOTRANSITOR = 0;
011 int cnt = 0;
012 int value, Schwelle;
013 
014 void setup()
015 {
016     analogWrite(Backlight, 200);
017     lcd.begin(16, 2);
018     lcd.setCursor(1, 0);
019     lcd.print("ALARMANLAGE!!!");
020     value = analogRead(PHOTOTRANSITOR);
021 }
022 
023 void loop()
024 {
025 
026   Schwelle = 25;
```

15

```
027
028   cnt++;
029   if(cnt > 2000)
030   {
031     cnt = 0;
032     value=analogRead(PHOTOTRANSITOR);
033   }
034
035
036   if(analogRead(PHOTOTRANSITOR) > (value + Schwelle) ||
            analogRead(PHOTOTRANSITOR) < (value - Schwelle))
037   {
038       lcd.setCursor(1, 1);
039       lcd.print("<<< ALARM >>>");
040       delay(2000);
041       value=analogRead(PHOTOTRANSITOR);
042   }
043   else
044   {
045       lcd.setCursor(0, 1);
046       lcd.print("               ");
047   }
048
049   delay(5);
050
051 }
```

16 DIGITALVOLTMETER MIT BARGRAFANZEIGE UND USB-SCHNITTSTELLE

Mit dem bereits Erlernten können Sie nun einen digitales Voltmeter mit analoger Bargrafanzeige programmieren. Die Bargrafanzeige leistet wertvolle Dienste bei Einstellarbeiten. So erkennt man auf einer analogen Anzeige wesentlich genauer, wo z. B. das Maximum oder das Minimum ist als auf einer digitalen Anzeige mit einer reinen Zahlendarstellung. Als Besonderheit ergänzen wir das Programm mit einer seriellen Ausgabe, die Ihre Messdaten über die USB-Schnittstelle an den PC sendet. Hierzu nutzen wir die USB-Schnittstelle, die bereits auf dem Arduino™-UNO-Board vorhanden ist und die wir bereits zum Programmieren verwenden.

Die Widerstände R1 und R2 werden bei diesem Experiment nicht benötigt und liegen dem Lernpaket nicht bei! Sie werden in diesem Kapitel aber weiter erklärt und können, falls benötigt, später im Elektronikfachhandel erworben werden.

Sie können mit der Schaltung ohne die beiden Widerstände R1/R2 mit dem Analogeingang A0 sehr genau Spannungen zwischen 0 und 5 V messen. Achten Sie aber darauf, dass Sie keine höheren Spannungen an den Anschluss geben, da sonst das Arduino™-Board beschädigt würde. Sie können mit der Schaltung bereits die Spannung von einer oder zwei Mignonzellen (AA) oder Mikrozellen (AAA) äußerst genau messen. Das Beispiel ähnelt sehr dem des Fotometers, jedoch mit ein paar kleinen Feinheiten. Wir verwenden dieses Mal zusätzlich die serielle Schnittstelle (UART = engl. Universal Asynchronous Receiver Transmitter) des Arduino™-Mikrocontrollers. Die Messdaten werden über die serielle Schnittstelle des Mikrocontrollers (UART) an den UART-zu-USB-Konverter auf der Arduino™-Platine gesendet, der sie an den PC weitergibt. Die seriellen Anschlüsse D0/RX und D1/TX sind bereits fest mit dem Konverter verbunden und es müssen hierzu keinen weiteren Verdrahtungs-

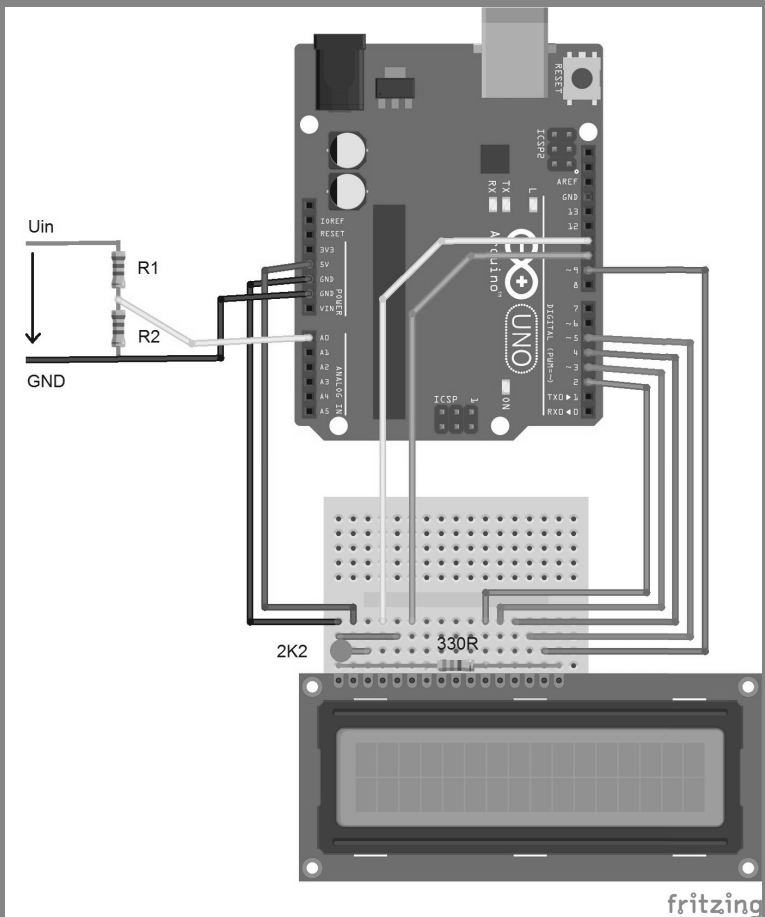

Abb. 16.1: Der Aufbau des Digitalvoltmeters

arbeiten vorgenommen werden. Auf PC-Seite wird schon bei der Installation der Arduino™-Platine ein virtueller Comport erzeugt. Dieser wird bereits zum Programmieren benutzt. Wir können ihn nun auch ganz einfach zur Datenübertragung an den PC verwenden. Dazu müssen wir im Programm nur die UART-Schnittstelle initialisieren. Diese wird mit `Serial.begin()` konfiguriert. Der Parameter `19200` zwischen den Klammern steht für die Übertragungsgeschwindigkeit. Der Initialisierung muss nur einmal bei Programmstart in der `Setup()`-Funktion ausgeführt werden.

```
001 Serial.begin(19200)
```

16

Baud ist die Einheit für die Symbolrate in der Nachrichten- und Fernmeldetechnik. *19200 Baud* bedeutet, dass 19.200 Symbole pro Sekunde übertragen werden. Die Symbolrate kann je nach Codierung unterschiedlich viele Bits enthalten und muss auf der Sender- und der Empfängerseite gleich eingestellt werden, um eine Übertragung zu ermöglichen.

Mit folgenden Zeilen senden wir nun das Messergebnis des ADC (0 bis 1013) ohne vorhergehende Umrechnung direkt an den PC. Die Umrechnung in Volt erfolgt im PC-Programm, da wir so nur zwei einzelne Bytes an den PC senden müssen, die sich wesentlich einfacher auswerten lassen als ein String (ASCII-Zeichenkette).

Nun wird der Puffer der UART-Schnittstelle mit `flush` geleert.

```
001 Serial.flush()
```

Jetzt zerlegen wir den analogen Messwert, der von 0 bis 1023 reicht, in ein High- und ein Lowbyte. Das Highbyte erhalten wir, indem wir den Messwert durch 256 dividieren.

```
001 highbyte = adc_raw / 256
```

Das Lowbyte erhalten wir mit der Modulo-Operation 256.

```
001 lowbyte = adc_raw % 256
```

Danach senden wir als Erstes das Highbyte und danach das Lowbyte an den PC.

```
001 Serial.write(highbyte)
002 Serial.write(lowbyte)
```

Um zu prüfen, ob die Werte korrekt übertragen wurden, senden wir zum Abschluss der Übertragung eine Checksumme, die aus einer fixen Zahl und der XOR-Bildung aus dieser sowie den beiden Bytes gebildet wird.

```
001 crc = 170^highbyte^lowbyte
002 Serial.write(crc)
```

Info
Das Programm funktioniert auch ohne PC-Programm und kann als Stand-alone-Spannungsmesser verwendet werden.

Beispielcode: VOLTMETER

```
// LCD-Library einbinden
#include <LiquidCrystal.h>

// LCD-Pins festlegen
// RS, E, D4, D5, D6, D7
LiquidCrystal lcd(11, 10, 2, 3, 4, 5);

#define LCD_LENGHT  16.0
#define ADC_CHANNEL 0

const int numReadings = 20;
unsigned int readings[numReadings];
int index = 0;
unsigned int total = 0;

byte MyChar0[8] = {
B00000,
B00000,
B00000,
B00000,
B00000,
B00000,
B00000,
B00000,
};

byte MyChar1[8] = {
B10000,
B10000,
B10000,
B10000,
B10000,
B10000,
B10000,
B10000,
};

byte MyChar2[8] = {
B11000,
B11000,
B11000,
B11000,
B11000,
B11000,
```

16

```
045 B11000,
046 B11000,
047 };
048
049 byte MyChar3[8] = {
050 B11100,
051 B11100,
052 B11100,
053 B11100,
054 B11100,
055 B11100,
056 B11100,
057 B11100,
058 };
059
060 byte MyChar4[8] = {
061 B11110,
062 B11110,
063 B11110,
064 B11110,
065 B11110,
066 B11110,
067 B11110,
068 B11110,
069 };
070
071 byte MyChar5[8] = {
072 B11111,
073 B11111,
074 B11111,
075 B11111,
076 B11111,
077 B11111,
078 B11111,
079 B11111,
080 };
081
082 int adc_AVG(byte channel)
083 {
084     total= total - readings[index];
085     readings[index] = analogRead(channel);
086     total= total + readings[index];
087     index = index + 1;
088     if (index >= numReadings)index = 0;
089     return total / numReadings;
090 }
```

```
091
092 void draw_bargraph(byte percent)
093 {
094   byte i, c1, c2;
095
096   lcd.setCursor(0, 1);
097
098   percent = map(percent, 0, 100, 0, 80);
099
100   c1 = percent / 5;
101   c2 = percent % 5;
102
103   for(i = 0; i < c1; ++i)
104   lcd.write(byte(5));
105
106   lcd.write(c2);
107
108   for(i = 0; i < 16 - (c1 + (c2 ? 1 : 0)); ++i)
109   lcd.write(byte(0));
110 }
111
112 void setup()
113 {
114     analogWrite(9, 200);
115
116     lcd.createChar(0, MyChar0);
117     lcd.createChar(1, MyChar1);
118     lcd.createChar(2, MyChar2);
119     lcd.createChar(3, MyChar3);
120     lcd.createChar(4, MyChar4);
121     lcd.createChar(5, MyChar5);
122     lcd.begin(16, 2);
123
124     Serial.begin(19200);
125 }
126
127 void loop()
128 {
129     double percent;
130     float voltage;
131     byte highbyte, lowbyte, crc;
132     int adc_raw = adc_AVG(ADC_CHANNEL);
133
134     voltage = (5.0 / 1024.0) * adc_raw;
135
136     lcd.setCursor(0, 0);
```

16

```
137     lcd.print(voltage, 2);
138     lcd.print(" V   ");
139
140     percent = voltage / 5.0 * 100.0;
141     draw_bargraph(percent);
142
143     Serial.flush();
144     highbyte=adc_raw/256;
145     lowbyte=adc_raw%256;
146     Serial.write(highbyte);
147     Serial.write(lowbyte);
148     crc=170^highbyte^lowbyte;
149     Serial.write(crc);
150
151     delay(20);
152 }
```

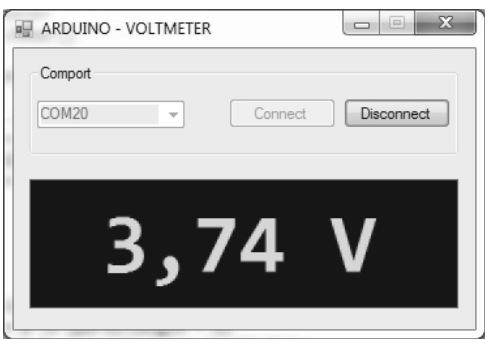

Abb. 16.2: Das VB.Net-Programm in Aktion

Um das PC-Programm zu starten, müssen Sie nur im Ordner *...\VOLTMETER\vb.net\bin\Release* die EXE-Datei ausführen. Danach wählen Sie den Comport aus, der identisch mit dem ist, den Sie bereits in der Arduino™-IDE für die Programmübertragung eingestellt haben. Wenn Sie nun auf *Connect* klicken, wird die Spannung im PC-Programm angezeigt.

Das VB.NET-Programm liegt als Quellcode bei und kann als Grundlage für eigenen Experimente verwendet werden. Dazu müssen Sie sich die kostenlose Visual-Basic-Express-Version von Microsoft herunterladen. Sie finden sie unter *https://www.visualstudio.com/downloads/download-visual-studio-vs*. Wenn Sie das Programm mit Visual Basic geöffnet haben, sehen Sie den Quellcode und den Designer vor sich. Im Designer werden die visuellen Steuerelemente wie Buttons, Textfelder etc. angezeigt. Werfen Sie nun einen Blick in den Quellcode des Voltmeterprogramms, indem Sie auf die Quellcodeansicht umschalten.

```
001 Imports System.IO.Ports.SerialPort
002 Imports System.Text.Encoding
```

Zuerst importieren wir die für die serielle Schnittstelle und die Codierung benötigten Funktionen. Ohne den Import der Encoding Library können wir z. B. keine Werte größer als 128 auswerten, da wir die Schnittstelle sonst nicht auf das Format UTF-8 umschalten können. Unser Messergebnis wäre daher falsch!

```
001 Dim input_data(10) As Byte
```

Mit `input_data(10)` legen wir ein Array an, das bis zu 10 Bytes aufnehmen kann. Darin werden die empfangenen Bytes der seriellen Schnittstelle später abgelegt.

```
001 Private Sub Form1_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles MyBase.Load
```

In der Funktion `Form1_load()` wird nun nach den vorhanden Comports gesucht. Diese werden in der Combobox aufgelistet. Die `Form1_Load()`-Funktion wird automatisch zuerst beim Programmstart aufgerufen – ähnlich wie die `Setup()`-Funktion unseres Arduino™-Programms.

```
001 Private Sub Button_Connect_Click(ByVal sender As System.
            Object, ByVal e As System.EventArgs) Handles
            Button_Connect.Click
```

In der Funktion `Button_Connect_Click()` konfigurieren und öffnen Sie zugleich die serielle Schnittstelle. Diese Funktion wird beim Anklicken des Connect-Buttons aufgerufen.

```
001 SerialPort1.PortName = ComboBox_Comport.Text
002 SerialPort1.BaudRate = 19200
003 SerialPort1.Encoding = System.Text.Encoding.UTF8
004 SerialPort1.Open()
```

Wir legen hier den Comport, die Baudrate und das Encoding fest und öffnen mit `SerialPort1.Open()` die Schnittstelle, die darauf hin sende- und empfangsbereit ist.

```
001 Private Sub Button_Disconnect_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        Button_Disconnect.Click
```

16

Mit der Funktion `Button_Disconnect_Click()` schließen wir die Schnittstelle wieder. Sie steht dann wieder für andere Programme wie die Arduino™-IDE zur Verfügung. Wenn Sie also Programmänderungen oder andere Programme auf Ihr Arduino™-Board übertragen wollen, müssen Sie das Programm zuvor beenden oder *Disconnect* klicken, um die Schnittstelle wieder freizugeben.

```
001 Private Sub SerialPort1_DataReceived(sender As Object
        , e As System.IO.Ports.SerialDataReceivedEventArgs)
        Handles SerialPort1.DataReceived
002
003         Dim cnt As Byte
004         Dim in_bytes As Byte
005         Dim HighByte As Byte
006         Dim LowByte As Byte
007         Dim crc As Byte
008         Dim crc_ok As Byte
009         Dim data_Word As Integer
010         Dim voltage As Single
011
012         Try
013
014             ' Hier werden die Daten empfangen
015             If SerialPort1.IsOpen Then
016
017                 Control.CheckForIllegalCrossThreadCalls
                                                    = False
018
019                 ' wie viele Bytes sind im Puffer
020                 in_bytes = SerialPort1.BytesToRead
021
022                 ' Alle Bytes holen
023                 For cnt = 1 To (in_bytes)
024                   input_data(cnt) = SerialPort1.ReadByte
025                 Next
026
027                 ' Puffer leeren
028                 SerialPort1.DiscardInBuffer()
029
030                 HighByte = input_data(1)
031                 LowByte = input_data(2)
032                 crc = input_data(3)
033
034                 ' Checksumme
035                crc_ok = 170 Xor input_data(1) Xor input_
```

```
                                                        data(2)
036
037                 If crc = crc_ok Then
038
039                     ' High und Low Byte wieder
                                                    zusammensetzen
040                  data_Word = ((HighByte * 256) + LowByte)
041                     voltage = data_Word * (5.0 / 1024.0)
042
043                     ' RAW Wert umrechen und als Spannung
                                                          Anzeigen
044                  Label1.Text = Format(voltage, "0.00 V")
045
046                 End If
047
048             End If
049
050         Catch ex As Exception
051         End Try
052
053     End Sub
```

In der Funktion `SerialPort1_DataReceived()` erfolgt nun der interessanteste Teil des VB.NET-Programms. Diese Funktion wird immer dann aufgerufen, wenn Daten von der seriellen Schnittstelle empfangen werden. Hier lesen wir die Bytes ein, die unser Arduino™-Programm sendet, und verarbeiten sie dort gleich. Wir prüfen vor dem Einlesen und Verarbeiten immer zuerst, ob der Anschluss geöffnet ist, und sehen dann nach, wie viel Bytes im Empfangspuffer zur Verfügung stehen. Danach lesen wir die Bytes in das `input_data()`-Array ein und ordnen die empfangenen Werte den Variablen `HighByte`, `LowByte` und `crc` zu. Zuletzt berechnen wir noch die Checksumme, indem wir die gleiche Prozedur anwenden wie in unserem Arduino™-Programm. Ist die berechnete und empfangene Checksumme gleich, ist kein Übertragungsfehler aufgetreten. Jetzt müssen wir nur noch die Messwertberechnung durchführen. Um eine gewisse Formatierung zu erhalten, verwenden wir die `Format()`-Funktion von VB.NET und geben den formatierten Wert auf das `Label1` aus.

16.1 | Erweitern des Messbereichs

Möchten Sie größere Spannungen messen, benötigen Sie einen Vorspannungsteiler, bestehend aus den Widerständen R1 und R2. Mit ihnen kön-

16

nen Sie den Eingangsspannungsbereich beliebig erweitern. Man beachte aber, dass mit zunehmendem Eingangsspannungsbereich auch die Auflösung sinkt.

Bei unserem Experiment, das für eine Eingangsspannung von 5 V ausgelegt ist, haben wir eine Auflösung von 0,00488 V oder 4,88 mV pro Wandlungsschritt. Unser digitaler Wert des analogen Eingangs kann 1.024 Schritte auflösen, da er eine digitale Auflösung von 10 Bit besitzt.

Wandlungsschritte (Steps): $1.024 = 2^{10}$
Auflösung pro Digit = U_{ADC} / Steps
5 V / 1.024 = 0,00488 V = 4,88 mV

Würden wir den Eingangsspannungsbereich auf 30 V anheben, würde sich die Auflösung um das Fünffache verschlechtern ((30 V – 5 V) / 5 V = 5). Wir würden »nur« noch mit einer Auflösung von 0,0244 V = 24,4 mV arbeiten.

Ermitteln wir nun den Spannungsteiler für einen Messbereich bis 30 V. Wir wissen bereits, dass wir den Eingangsbereich von 5 V auf 30 V erweitern möchten, was dem Faktor 5 entspricht. Der Messeingang für Spannungsmessungen darf aber nicht zu niederohmig sein und sollte mindestens 100 kΩ betragen. Moderne Spannungsmesser hingegen besitzen einen Eingangswiderstand von 10 MΩ, um die Spannungsquelle bei der Messung möglichst wenig zu belasten und das Messergebnis so wenig wie möglich zu verfälschen.

Gehen wir von einen Eingangswiderstand von ca. 100 kΩ aus und definieren den Widerstand R1 mit 100 kΩ. In einer Serienschaltung ist das Verhältnis der Spannungen identisch mit dem Verhältnis der Widerstände zueinander. Der Widerstand R2 muss also nur 1/5 so groß sein wie der Widerstand von R1. Wir rechnen 100 kΩ / 5 und erhalten für R2 den Wert 20 kΩ.

Ermitteln wir nun den Strom, der in der Schaltung fließt. In einer Serienschaltung ist der Storm durch die Widerstände gleich und die Spannungen teilen sich auf. Der Strom durch die Widerstände berechnet sich folgendermaßen:

$I = U / R_{Gesamt}$
30 V / 120 kΩ = 0,25 mA

An R2 würde dann folgende Spannung abfallen:

U = R2 x I
20 kΩ x 0,25 mA = 5 V

Bei einer angelegten Spannung von 15 V wäre der Wert am ADC:

15 V / 120 kΩ = 0,125 mA
20 kΩ x 0,125 mA = 2,5 V

Berechnen wir nun die Verlustleistung über die Widerstände bei maximaler Eingangsspannung:

P = U x I
30 V x 0,25 mA = 7,5 mW

Sie können sich nun selbst Ihren passenden Spannungsteiler berechnen und den Messwert im Programm anpassen, indem Sie ihn mit dem Spannungsteilerfaktor multiplizieren. Der Messfehler lässt sich experimentell mit einem genauen Spannungsmesser bestimmen und wird in den Multiplikationsfaktor einberechnet. Achten Sie auf die Widerstandswerte der E-Reihen und versuchen Sie nicht, die mathematisch ermittelten Werte in eine Schaltung zu übernehmen, indem Sie auf unübliche E-Reihen zurückgreifen oder sogar versuchen, Widerstände in Serie oder parallel zu verschalten, um genau den berechneten Widerstandswert zu erhalten.

Diese Berechnung lässt zudem den Eingangswiderstand des ADC außer Betracht. Dieser liegt bei dem Arduino™-UNO ATmega328 bei ca. 100 kΩ. Wir sollten somit nicht höher als 100 kΩ mit dem Eingangsspannungsteiler gehen, um noch brauchbare Messergebnisse zu erhalten. Um genauere Ergebnisse zu erhalten, könnten Sie die Berechnung für einen belasteten Spannungsteiler durchführen und zu R2 den Widerstand des ADC einberechnen.

Bedenken Sie auch, dass sich die Toleranz bei einer Serienschaltung addiert. Wenn Sie z. B. bei der gezeigten Schaltung zwei Widerstände mit je 5 % Toleranz verwenden, liegt die Gesamttoleranz schon bei 10 %.

Als ideal haben sich die Werte der E12-Reihe herauskristallisiert. Sie sind in jedem Elektronikshop erhältlich und zählen zur Standardreihe. Beim Bau von Messgeräten sollten Sie aber die Toleranz der Komponenten so niedrig wie möglich halten, um genaue Messergebnis zu bekommen.

Tipp
Unter folgendem Link finden Sie ein Onlinetool, mit dem Sie Spannungsteiler berechnen können: *http://www.peacesoftware.de/einigewerte/spannungsteiler.html*

17 TEMPERATUR-ANZEIGE IN GRAD UND FAHRENHEIT

Dieses Experiment zeigt, wie Sie mit einem kostengünstigen Temperaturwiderstand wie dem hier verwendeten NTC (engl. Negative Temperature Coefficient Thermistor) ein einfaches LCD-Thermometer programmieren.

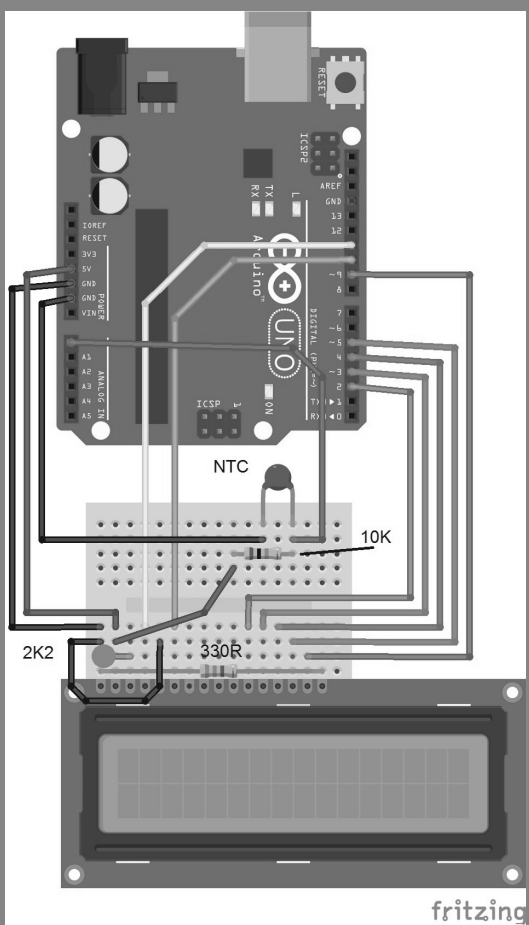

Abb. 17.1: Aufbau des NTC-LCD-Thermometers; die Schaltung ist ähnlich wie die des Fotometers, nur dass statt des Fototransistors ein NTC verwendet wird.

Ein NTC ist ein Widerstand, der seinen Widerstandswert abhängig von seiner Temperatur verändert. Es handelt sich bei einem NTC um einen sogenannten Heißleiter. Das bedeutet, dass der Widerstand bei Erhöhung der Temperatur abnimmt.

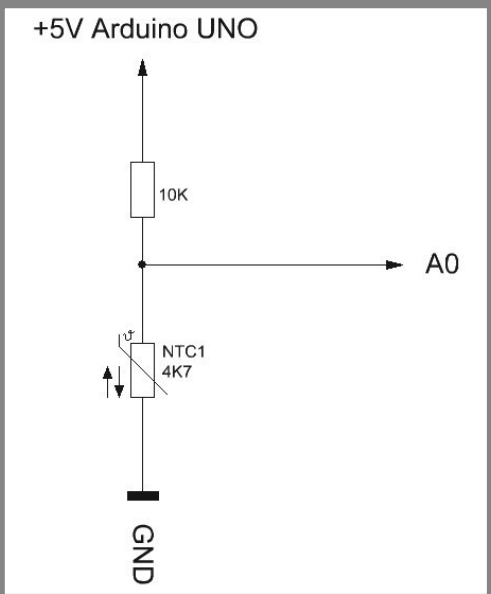

Abb. 17.2: Schaltbild der NTC-Temperaturmessschaltung

Das Schaltbild zeigt den Aufbau genauer. Es handelt sich wieder um einen variablen Spannungsteiler, bestehend aus einem 10-kΩ-Festwiderstand und dem variablen NTC-Widerstand. Sinkt die Temperatur, steigt der Widerstand des NTC und dadurch auch die Spannung am Analogeingang A0 an.

Beispielcode: LCD THERMO

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 #define Backlight   9
009 #define ADC_NTC     0
010
```

17

```
011 float temp_celsius, temp_fahrenheit;
012 int ADC_raw;
013
014 float Grad_to_Fahrenheit(float grad)
015 {
016     return (9.0 / 5.0) * grad + 32;
017 }
018
019 void setup()
020 {
021     analogWrite(Backlight, 200);
022     lcd.begin(16, 2);
023     lcd.setCursor(0, 0);
024     lcd.print("THERMO - ARDUINO");
025     Serial.begin(9600);
026     delay(2000);
027     lcd.clear();
028 }
029
030 void loop()
031 {
032     ADC_raw = analogRead(ADC_NTC);
033     temp_celsius = (580.0 - ADC_raw) / 10;
034     temp_fahrenheit = Grad_to_Fahrenheit(temp_celsius);
035
036     lcd.setCursor(0, 0);
037     lcd.print(temp_celsius, 1);
038     lcd.write(223);
039     lcd.print("C  ");
040
041     lcd.setCursor(0, 1);
042     lcd.print(temp_fahrenheit, 1);
043     lcd.write(223);
044     lcd.print("F  ");
045
046     Serial.print("Temperatur = ");
047     Serial.print(temp_celsius);
048     Serial.print(" °C");
049
050     Serial.print(" | ");
051     Serial.print(temp_fahrenheit);
052     Serial.println(" °F");
053
054     delay(1000);
055 }
```

Die Widerstandskurve des NTC ist nicht gerade linear und muss durch eine Berechnung angepasst werden.

```
001 temp_celsius = (580.0 - ADC_raw) / 10
```

Um die Ausgabe nicht nur in Grad Celsius zu erhalten, wird der Wert in Fahrenheit umgerechnet und auf dem LCD angezeigt.

```
001 float Grad_to_Fahrenheit(float grad)
002 {
003     return (9.0 / 5.0) * grad + 32;
004 }
```

Sie sehen, wir können nach dem Befehl *return* direkt Berechnungen schreiben und müssen diese nicht zwingend zuvor einer Variablen übergeben. Diese Funktion berechnet aus Grad Celsius den Wert in Grad Fahrenheit, der im amerikanischen System verwendet wird.

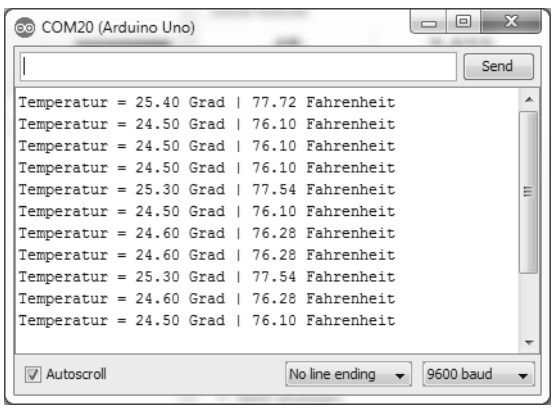

Abb. 17.3: Terminalausgabe der Temperatur

In diesem Programm wurde auch die serielle Schnittstelle verwendet, und der gleiche Wert auf dem LCD wird zusätzlich als ASCII-String über die UART-Schnittstelle ausgegeben. Wenn Sie das Arduino™-eigene Terminalprogramm öffnen und dessen Schnittstelle auf 9.600 Baud stellen, erhalten Sie die Messwerte in Klartext im Terminalprogramm.

18 TEMPERATUR-PLOTTER MIT USB-SCHNITTSTELLE

Erweitern wir das Thermometer um ein VB.NET-Programm und ändern wir den Programmcode so ab, dass wir statt der Klartextausgabe wie im Vorgängerprogramm nur den Temperaturwert an den PC senden. Die Schaltung bleibt gleich, nur das Programm ändern wir wie folgt:

```
001 Serial.flush()
002 highbyte=ADC_raw/256
003 lowbyte=ADC_raw%256
004 Serial.write(highbyte)
005 Serial.write(lowbyte)
006 crc=170^highbyte^lowbyte
007 Serial.write(crc)
```

Sie sehen, wir übertragen den Temperaturwert wie den Messwert der Spannung beim digitalen USB-Voltmeter. Das komplette Programm finden Sie auf der beiliegenden CD-ROM. Es trägt den Namen »TEMP_PLOT". Eine weitere Auflistung des Programmcodes ist nicht nötig, da es dem Vorgängerprogramm entspricht.

Jedoch hat sich im VB.NET-Programm einiges geändert. Es wurde um eine grafische Ausgabe erweitert. Dazu wurde ein Steuerelement mit den Namen »AutoRedraw« programmiert, das eine zusammenhängende Line in X- und Y-Richtung, abhängig von der Eingangsvariable, zeichnet und als Temperaturplotter dient. Dieses Programm liegt als Sourcecode bei und kann für eigene Experimente verwendet werden. Eine detaillierte Beschreibung zu den Zeichenfunktionen würde jedoch den Rahmen dieses Lernpakets sprengen. Einschlägige Internetseiten wie z. B. www.vb-paradise.de und VB.NET-Fachbücher ermöglichen es, sich in die VB.NET-Programmierung zu vertiefen.

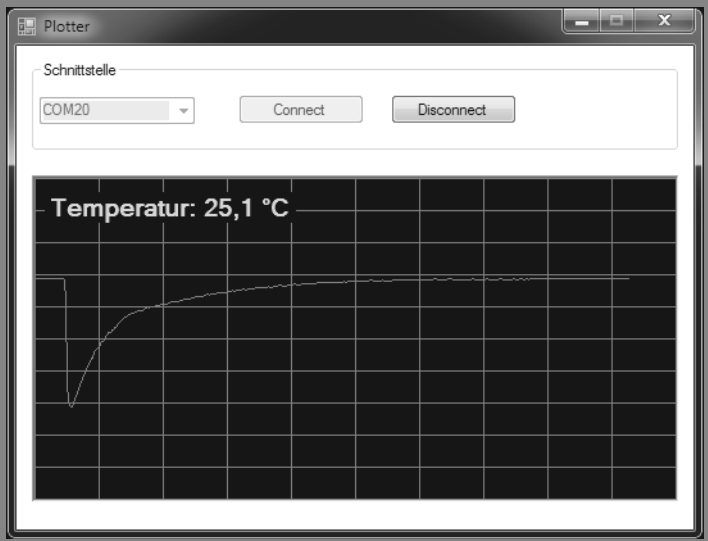

Abb. 18.1: Der Temperaturplotter im Einsatz; der Temperaturabfall wurde mit Kältespray erzeugt und die Kurve zeigt deutlich, wie danach die Temperatur wieder auf ca. 25 °C ansteigt.

19 WEBSYNCHRONE UHR

Eine Uhr haben wir in diesem Lernpaket bereits programmiert. Da sie nicht sonderlich genau ist und im Lauf der Betriebszeit einer sehr großen Abweichung unterliegt, programmieren wir nun mit den Kenntnissen um die serielle Übertragung zwischen PC und Arduino™ eine websynchrone Uhr. Websynchron deshalb, da standardmäßig die Windows-Uhrzeit mit einem Internetzeitserver automatisch im Hintergrund abgeglichen wird. In diesem VB.NET-Programm werden wir nun die Zeit des PCs an Arduino™ senden und auf dem LCD ausgeben. Das VB.NET-Programm liegt als ausführbare EXE-Datei und als Quellcode bei.

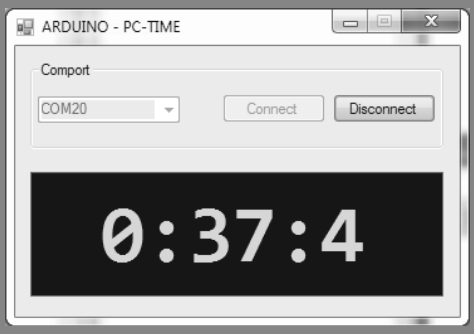

Abb. 19.1: Der PC sendet seine Uhrzeit an die Arduino™-Platine.

Abb. 19.2: Arduino™ mit genauer Zeitausgabe

Upload

Das Experiment benötigt die LCD-Grundbeschaltung, wie wir sie beim Funktionstest aufgebaut haben.

Beispielcode: PC Time

```
001 // LCD-Library einbinden
002 #include <LiquidCrystal.h>
003
004 // LCD-Pins festlegen
005 // RS, E, D4, D5, D6, D7
```

```
006 LiquidCrystal lcd(11, 10, 2, 3, 4, 5);
007
008 #define Backlight   9
009
010 byte Stunde, Minute, Sekunde;
011
012 void setup()
013 {
014     Serial.begin(9600);
015
016     analogWrite(Backlight, 200);
017
018     lcd.begin(16, 2);
019     lcd.clear();
020     lcd.setCursor(0, 0);
021     lcd.print("ARDUINO PC-UHR");
022
023     Stunde = 0;
024     Minute = 0;
025     Sekunde = 0;
026 }
027
028 void loop()
029 {
030
031     if(Serial.available()>3)
032     {
033         Stunde = Serial.read();
034         Minute = Serial.read();
035         Sekunde = Serial.read();
036
037         lcd.setCursor(0, 1);
038         lcd.print("NOW: ");
039
040         if(Stunde < 24)
041         {
042             if(Stunde < 10) lcd.print("0");
043             lcd.print(Stunde);
044             lcd.print(":");
045         }
046         if(Minute < 60)
047         {
048             if(Minute < 10) lcd.print("0");
049             lcd.print(Minute);
050             lcd.print(":");
051         }
```

19

```
052         if(Sekunde < 60)
053         {
054             if(Sekunde < 10) lcd.print("0");
055             lcd.print(Sekunde);
056         }
057     }
058
059     Serial.flush();
060     delay(100);
061 }
```

Die Grundstruktur des Programms entspricht der der normalen Uhr (RTC), die wir am Anfang des Lernpakets bereits kennengelernt haben.

Diesmal empfangen wir im VB.NET-Programm keine Daten, sondern senden Daten vom PC an den Arduino™. Die Daten entsprechen Stunden, Minuten und Sekunden in Form von Bytes. Diese lesen wir mit Arduino™ immer ein, wenn `Serial.available()` größer drei ist. Das kennzeichnet, dass drei Bytes im Empfangspuffer anstehen. Mit `Serial.read()` lesen wir systematisch die drei ankommenden Bytes ein und ordnen sie den Arduino™-Variablen für Stunde, Minute und Sekunde zu. Das war's auch schon auf der Arduino™-Seite. Betrachten wir noch die Sendefunktion im VB.NET-Programm. Hierzu verwenden wir einen Timer, der auf 500 ms eingestellt ist.

```
001 Private Sub Timer1_Tick(sender As System.Object, e As
                    System.EventArgs) Handles Timer1.Tick
002
003         If SerialPort1.IsOpen Then
004
005             SerialPort1.Write(Chr(Now.Hour) & Chr(Now.
                                Minute) & Chr(Now.Second))
006             Label1.Text = Now.Hour & ":" & Now.Minute &
                                        ":" & Now.Second
007
008         End If
009 End Sub
```

Der Timer könnte auch auf 1.000 ms eingestellt werden. Es kann dann aber vorkommen, dass die Sekundenausgabe auf dem LCD stellenweise ruckelt, da sich die Daten überschneiden. Besser ist, die Updatezeit etwas schneller oder eben auf genau die Hälfte einzustellen, um die Ausgabe flüssiger erscheinen zu lassen.

In der VB.NET-Timer-Funktion wird vor der eigentlichen Ausgabe sicherheitshalber geprüft, ob der serielle Anschluss geöffnet ist. Ist das der Fall, werden mit `SerialPort1.Write()` die Zeitdaten übertragen. Dazu verwenden wir die Funktion `Now()`. Sie enthält die Zeit und das Datum und kann mit den Parametern `Hour`, `Minute` und `Second` angewiesen werden, nur die gewünschten Stellen auszugeben. Um die Werte für die Übertragung in Bytes zu wandeln, wird jeder Wert mit `Char()` in ein Byte konvertiert.

Unsere PC-Uhr zeigt uns nun die genaue Zeit auf dem LCD an.

20 ANHANG

20.1 | Elektrische Einheiten

Man unterscheidet zwischen Spannung, Strom, Widerstand und den Einheiten, in denen die Größen gemessen werden (z. B. Volt oder Ampere). Jede Einheit hat eine Abkürzung, die in den Formeln verwendet wird. Die Abkürzungen ermöglichen eine kurze und übersichtliche Schreibweise. So schreibt man statt Strom gleich 1 Ampere nur I = 1 A.

In allen Formeln im Buch werden diese Abkürzungen verwendet.

Größe	Abkürzung	Einheit	Abkürzung
Spannung	U	Volt	V
Strom	I	Ampere	A
Widerstand	R	Ohm	Ω
Leistung	P	Watt	W
Frequenz	f	Hertz	Hz
Zeit	t	Sekunde	s
Wellenlänge	Λ (Lambda)	Meter	m
Induktivität	L	Henry	H
Kapazität	C	Farad	F
Fläche	A	Quadratmeter	m^2

20.2 | ASCII-Tabelle

Zeichen	dezimal	hexadezimal	binär	Beschreibung
NUL	000	000	00000000	Null Character
SOH	001	001	00000001	Start of Header
STX	002	002	00000010	Start of Text
ETX	003	003	00000011	End of Text
EOT	004	004	00000100	End of Transmission
ENQ	005	005	00000101	Enquiry
ACK	006	006	00000110	Acknowledgment
BEL	007	007	00000111	Bell
BS	008	008	00001000	Backspace
HAT	009	009	00001001	Horizontal TAB
LF	010	00A	00001010	Line Feed
VT	011	00B	00001011	Vertical TAB
FF	012	00C	00001100	Form Feed
CR	013	00D	00001101	Carriage Return
SO	014	00E	00001110	Shift out
SI	015	00F	00001111	Shift in
DLE	016	010	00010000	Data Link Escape
DC1	017	011	00010001	Device Control 1

Zeichen	dezimal	hexadezimal	binär	Beschreibung
DC2	018	012	00010010	Device Control 2
DC3	019	013	00010011	Device Control 3
DC4	020	014	00010100	Device Control 4
NAK	021	015	00010101	Negative Acknowledgment
SYN	022	016	00010110	Synchronous Idle
ETB	023	017	00010111	End of Transmission Block
CAN	024	018	00011000	Cancel
EM	025	019	00011001	End of Medium
SUB	026	01A	00011010	Substitute
ESC	027	01B	00011011	Escape
FS	028	01C	00011100	File Separator
GS	029	01D	00011101	Group Separator
RS	030	01E	00011110	Request to Send, Record Separator
US	031	01F	00011111	Unit Separator
SP	032	020	00100000	Space
!	033	021	00100001	Exclamation Mark
«	034	022	00100010	Double Quote
#	035	023	00100011	Number Sign
$	036	024	00100100	Dollar Sign
%	037	025	00100101	Percent
&	038	026	00100110	Ampersand
‘	039	027	00100111	Single Quote
(	040	028	00101000	Left Opening Parenthesis
)	041	029	00101001	Right Closing Parenthesis
*	042	02A	00101010	Asterisk
+	043	02B	00101011	Plus
,	044	02C	00101100	Comma
-	045	02D	00101101	Minus or Dash
.	046	02E	00101110	Dot
/	047	02F	00101111	Forward Slash
0	048	030	00110000	
1	049	031	00110001	
2	050	032	00110010	
3	051	033	00110011	
4	052	034	00110100	
5	053	035	00110101	
6	054	036	00110110	
7	055	037	00110111	
8	056	038	00111000	
9	057	039	00111001	
:	058	03A	00111010	Colon
;	059	03B	00111011	Semi-Colon
<	060	03C	00111100	Less Than
=	061	03D	00111101	Equal
>	062	03E	00111110	Greater Than
?	063	03F	00111111	Question Mark
@	064	040	01000000	AT Symbol
A	065	041	01000001	
B	066	042	01000010	
C	067	043	01000011	
D	068	044	01000100	
E	069	045	01000101	
F	070	046	01000110	
G	071	047	01000111	
H	072	048	01001000	
I	073	049	01001001	

20

Zeichen	dezimal	hexadezimal	binär	Beschreibung
J	074	04A	01001010	
K	075	04B	01001011	
L	076	04C	01001100	
M	077	04D	01001101	
N	078	04E	01001110	
O	079	04F	01001111	
P	080	050	01010000	
Q	081	051	01010001	
R	082	052	01010010	
S	083	053	01010011	
T	084	054	01010100	
U	085	055	01010101	
V	086	056	01010110	
W	087	057	01010111	
X	088	058	01011000	
Y	089	059	01011001	
Z	090	05A	01011010	
[	091	05B	01011011	Left opening Bracket
\	092	05C	01011100	Back Slash
]	093	05D	01011101	Right closing Bracket
^	094	05E	01011110	Caret
_	095	05F	01011111	Underscore
`	096	060	01100000	
a	097	061	01100001	
b	098	062	01100010	
c	099	063	01100011	
d	100	064	01100100	
e	101	065	01100101	
f	102	066	01100110	
g	103	067	01100111	
h	104	068	01101000	
i	105	069	01101001	
j	106	06A	01101010	
k	107	06B	01101011	
l	108	06C	01101100	
m	109	06D	01101101	
n	110	06E	01101110	
o	111	06F	01101111	
p	112	070	01110000	
q	113	071	01110001	
r	114	072	01110010	
s	115	073	01110011	
t	116	074	01110100	
u	117	075	01110101	
v	118	076	01110110	
w	119	077	01110111	
x	120	078	01111000	
y	121	079	01111001	
z	122	07A	01111010	
{	123	07B	01111011	Left opening Brace
|	124	07C	01111100	Vertical Bar
}	125	07D	01111101	Right closing Brace
~	126	07E	01111110	Tilde
DEL	127	07F	01111111	Delete

21 BEZUGSQUELLEN

Conrad Electronic SE
Klaus-Conrad-Straße 1

92240 Hirschau

www.conrad.de

Electronic Assembly GmbH
Zeppelinstraße 19

82205 Gilching bei München

www.lcd-module.de