HANSER


Leseprobe

Robert Jänisch, Jörn Donges

Mach was mit Arduino!

Einsteigen und durchstarten mit Drum Machine, Roboterauto & Co.

ISBN (Buch): 978-3-446-45128-5

ISBN (E-Book): 978-3-446-45258-9

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-45128-5

sowie im Buchhandel.

# Inhaltsverzeichnis

# 6 Eine Weltzeituhr mit Alarmfunktion

In diesem Kapitel werden wir ein Projekt verwirklichen, das einen ganz typischen Einsatz von Mikrocontrollern im Alltag zeigt. Wir bauen eine Digitaluhr mit dem Arduino und erweitern sie im nächsten Schritt um eine Weltzeituhr sowie eine Alarmfunktion. Dabei lernst du ein neues, sehr nützliches Bauteil kennen, das du auch in vielen anderen Projekten gut gebrauchen kannst: das LCD- Display HD44780. Softwareseitig wirst du ein paar neue Programmierkonzepte kennen lernen – allen voran den `String`-Datentyp und die Verarbeitung von Zeichenketten.

**Bild 6.1** Wir bauen eine Weltzeituhr mit Alarmfunktion.

## 6.1 Das LCD-Display HD44780 anschließen

Das LCD-Display HD44780 kann zwei Zeilen mit jeweils 16 Zeichen darstellen. Es ermöglicht dir, verschiedenste Daten und Werte anzuzeigen, und ist quasi so etwas wie ein kleiner Bildschirm für den Arduino. HD44780 ist eigentlich die Bezeichnung für den Control-

lerchip, der die Daten vom Arduino annimmt und damit die Anzeige ansteuert. Das Display hat 16 Eingangspins, von denen acht für die Übertragung der Anzeigedaten vorgesehen sind. Die restlichen acht Pins dienen der Steuerung, Spannungsversorgung und Beleuchtung des Displays.

Die Pinbelegung dieses Bauteils folgt einem weit verbreiteten Standard und wird von der Arduino-Programmiersprache bereits mit vielen Funktionen unterstützt, die das Darstellen von Daten auf dem Display zum Kinderspiel machen. Die meisten Displays, die du im Elektronikhandel für wenige Euro kaufen kannst, sind schon mit Bausteinen bestückt und können direkt am Arduino betrieben werden. Am besten besorgst du dir ein Modul mit angelöteter Stiftleiste, die du direkt ins Breadboard stecken kannst.

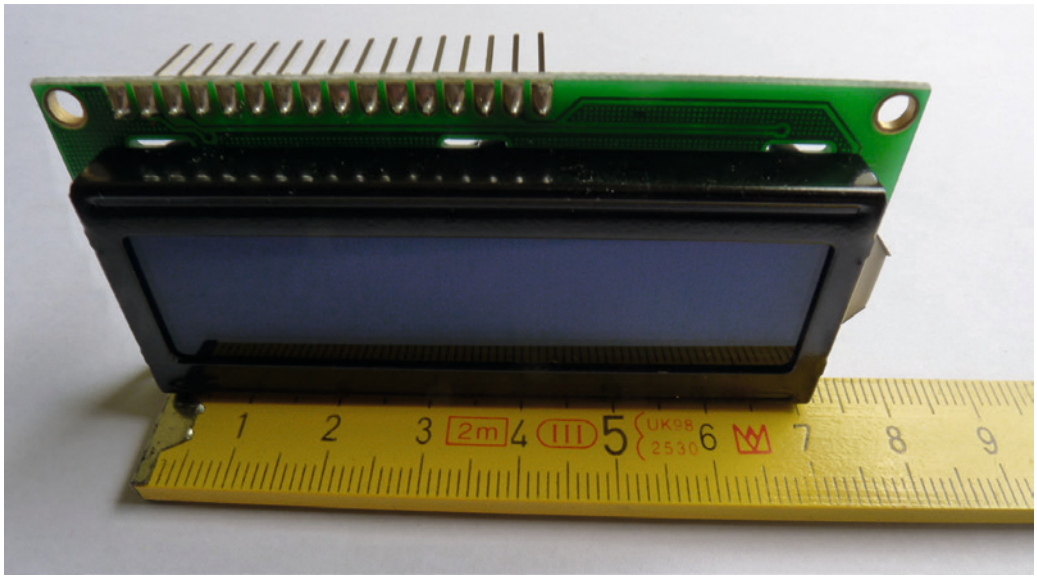


**Bild 6.2** LCD-Display HD44780

Um das LCD-Display HD44780 zu verdrahten und an den Arduino anzuschließen, brauchst du eine ganze Menge von Drahtstückchen. Bei voller Beschaltung belegt das Display neun Pins am Arduino. Außerdem musst du auf dem Breadboard noch einige Pins miteinander verbinden. Nun ist der Zeitpunkt gekommen, zu dem du aus einer Rolle Klingeldraht kurze Verbindungsstücke schneidest, so wie es in Kapitel 3 erklärt wurde. Es gibt im Elektronikhandel auch fertige Sets sogenannter Jumperkabel, die du verwenden kannst.

Die Pins am Display werden von links nach rechts gezählt. An die ersten beiden Pins wird die 5 V-Versorgungsspannung angelegt. Pin 3 ist für die Einstellung des Kontrasts verantwortlich. Manche Schaltungen, die du im Internet findest, sehen hier einen veränderbaren Widerstand vor, um den Kontrast der Anzeige einstellen zu können. Normalerweise reicht es aber aus, den Pin einfach auf GND zu legen. Damit erzielt man meist schon eine gute Darstellung. Solltest du dennoch Probleme mit einer zu blassen Anzeige haben, kannst du

über diesen Pin eingreifen. Die Pins 4 bis 6 steuern die Datenübertragung zwischen Arduino und LCD-Controller. Darauf wollen wir erst einmal nicht im Einzelnen eingehen, denn der Arduino-Compiler kümmert sich automatisch um die korrekte Ansteuerung des Displays.

Die Pins 7 bis 14 sind die Datenleitungen. Wir haben also 8 Bit zur Verfügung, um Anzeigedaten auf das Display zu schicken. Der LCD- Controller bietet auch einen „Light-Modus“, in dem nur vier Datenleitungen benötigt werden. Dadurch wird die Übertragung etwas langsamer laufen, aber die Geschwindigkeit reicht für unsere Zwecke dicke aus. Daher schließen wir hier nur vier Pins an.

Die Pins 15 und 16 dienen der Spannungsversorgung für die Hintergrundbeleuchtung des Displays. Wir schließen hier die +3.3 V-Ausgangsspannung des Arduino an.

6.3 zeigt dir, wie du die Schaltung aufbauen kannst, wenn du das LCD-Display mit der Stiftleiste ins Breadboard steckst.

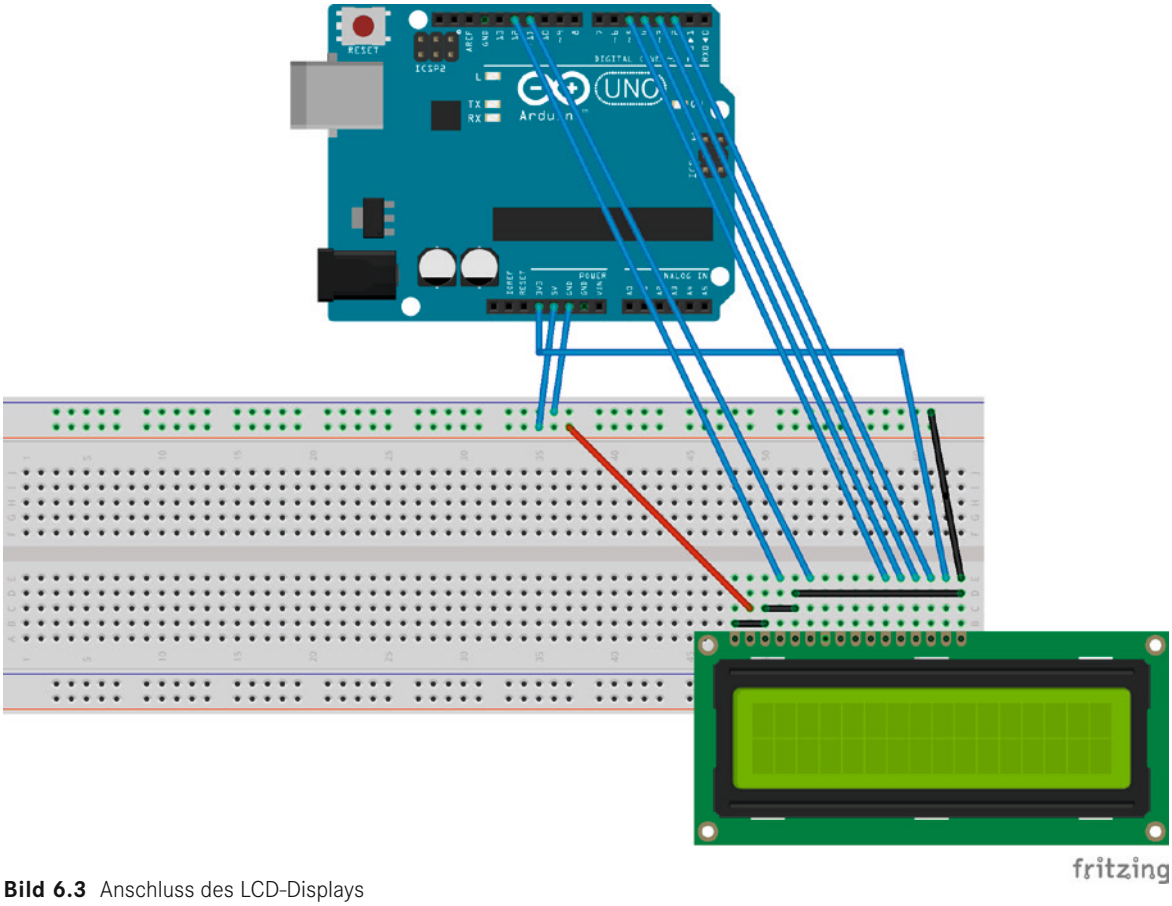


**Bild 6.3** Anschluss des LCD-Displays

Die folgende Tabelle zeigt die Verdrahtung noch einmal in der Übersicht.

| LCD-Pin | LCD-Bezeichnung | Arduino-Anschluss | Funktion |
|---|---|---|---|
| 1 | VSS | GND | Spannung – |
| 2 | VDD | + 5 V | Spannung + |
| 3 | V0 | GND | Kontrast (0 – 5V) |
| 4 | RS | DIGITAL 12 | Register Select |
| 5 | R/W | GND | Read/Write |
| 6 | E | DIGITAL 11 | Enable |
| 7–10 | DB0–DB3 | nicht verwendet | Datenpins |
| 11 | DB4 | DIGITAL 5 | Datenpins |
| 12 | DB5 | DIGITAL 4 | Datenpins |
| 13 | DB6 | DIGITAL 3 | Datenpins |
| 14 | DB7 | DIGITAL 2 | Datenpins |
| 15 | A | + 3.3 V | LED Backlight + |
| 16 | K | GND | LED Backlight – |

Bild 6.4 zeigt den Schaltplanaufbau für das LCD-Display.

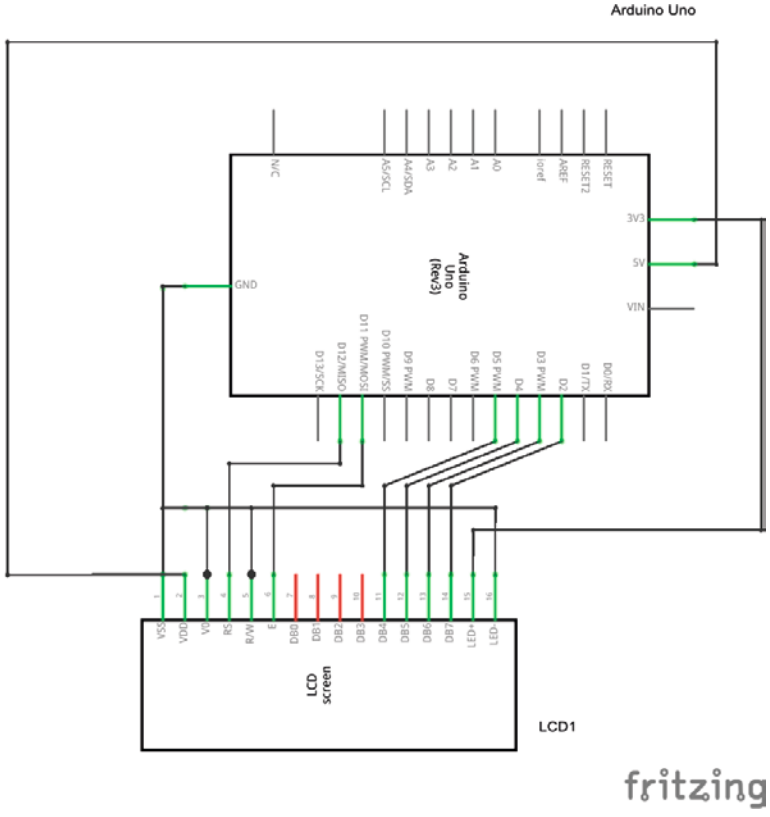


**Bild 6.4** Schaltplan für den LCD-Display-Aufbau

Bild 6.5 bis Bild 6.7 zeigen, wie du den Arduino an das LCD-Display anschließt.

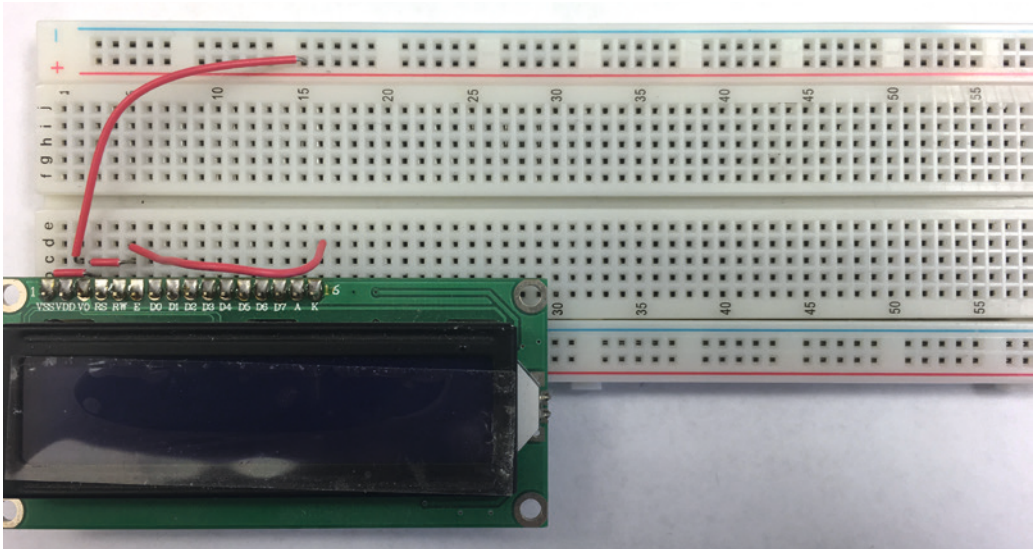

**Bild 6.5** Die ersten Anschlüsse auf dem LCD-Display

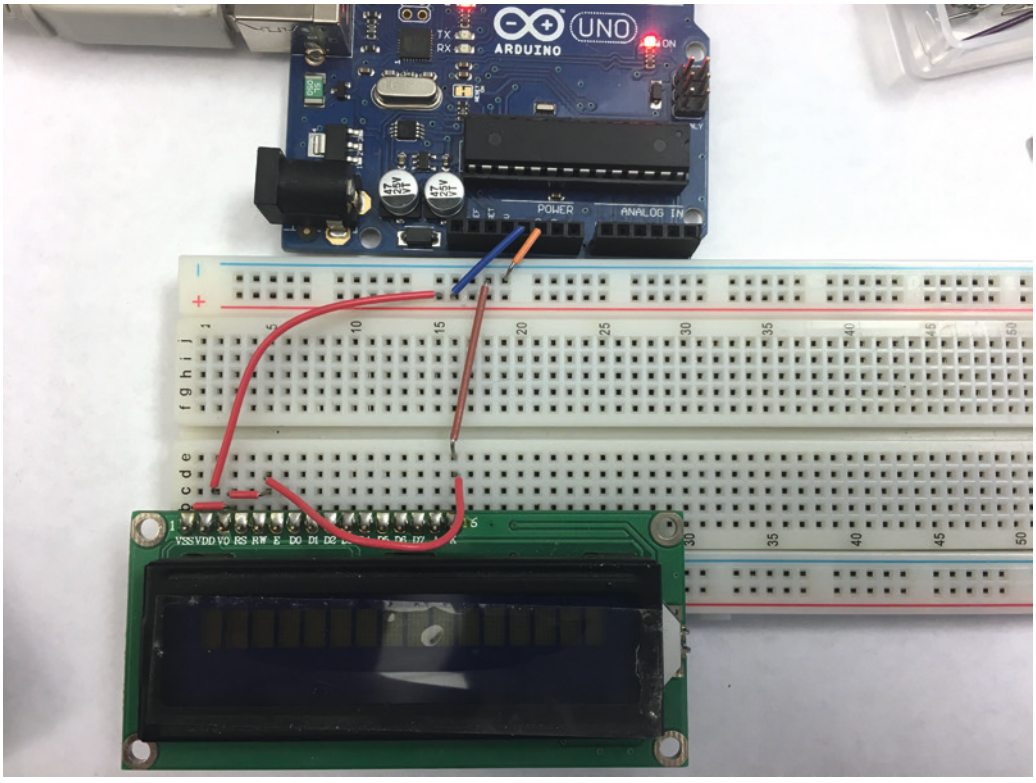


**Bild 6.6** Jetzt schließt du den Arduino an.

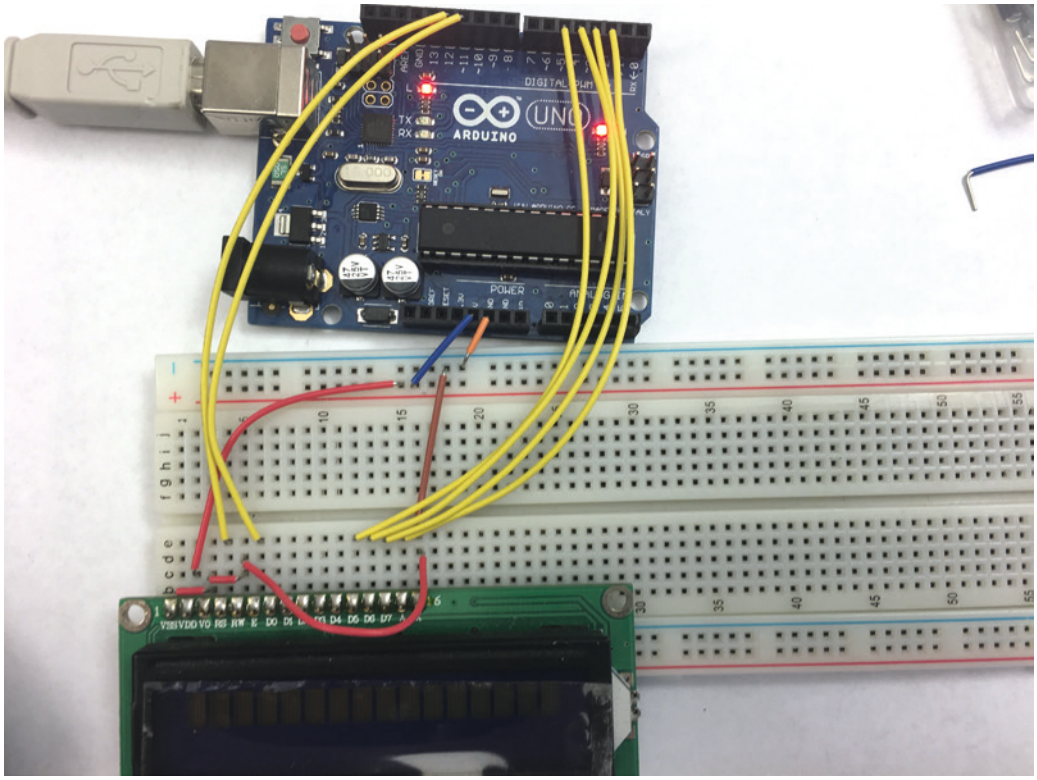


**Bild 6.7** Das LCD-Display ist nun komplett angeschlossen.

## 6.2 Text auf dem Display darstellen

Hast du das LCD-Display erfolgreich angeschlossen? Prima, dann können wir loslegen. Dein Aufbau wird wahrscheinlich so ähnlich aussehen wie in Bild 6.8. Mit Folgenden wollen wir nun das Display per Software ansteuern.

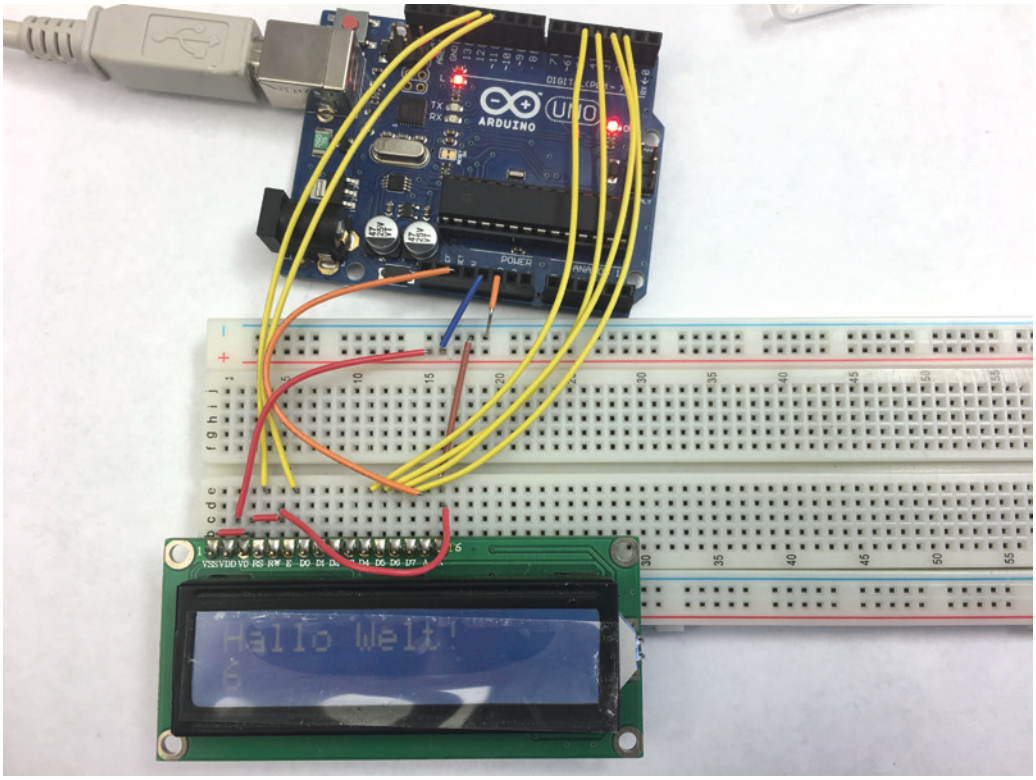


**Bild 6.8** „Hallo Welt!“

Bevor wir uns um die Funktionen der Uhr kümmern, testen wir das Display erst einmal mit einem Sketch, der eine Meldung auf dem Display ausgibt. Wir entscheiden uns für das klassische „Hallo Welt“, aber du kannst natürlich einen beliebigen Text wählen.

```
// Testsketch für LCD-Display
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
// LCD mit 16 Spalten und zwei Zeilen initialisieren
lcd.begin(16, 2);
// Text ausgeben
lcd.print("Hallo Welt!");
}
void loop() {
// Cursor in der zweiten Zeile an erster Stelle
positionieren
lcd.setCursor(0, 1);
// Gib die Anzahl der Sekunden seit dem letzten Reset
aus:
lcd.print(millis()/1000);
}
```

In diesem Code sind einige neue Elemente enthalten, die wir im Folgenden kurz durchgehen werden.

- **Zeile 2:** Die Anweisung `#include <LiquidCrystal.h>` dient dazu, den Inhalt einer anderen Datei in deinen Code einzubinden. Der Compiler behandelt das so, als ob der Text aus der Datei direkt an dieser Stelle stehen würde. Meist verwendet man diese Anweisung, um externe Programmbibliotheken einzubinden. In diesem Fall binden wir die Funktionen zur Ansteuerung des LCD-Displays ein, die in der Datei `LiquidCrystal.h` stehen.
- **Zeile 3:** Hier erzeugen wir ein Objekt vom Typ `LiquidCrystal` mit Namen `lcd`. Wir übergeben der Software mit `LiquidCrystal lcd(12, 11, 5, 4, 3, 2);` gleich das Pin-Layout unserer Schaltung. Immer wenn wir in Zukunft auf das Display schreiben, passiert das über dieses Objekt. Eine ausführliche Dokumentation findest du unter dem Stichwort „LiquidCrystal Library“ auf der Arduino-Website: *https://www.arduino.cc/en/pmwiki.php?n=Reference/LiquidCrystal*
- **Zeile 6:** Hier initialisieren wir das Display, d. h., wir legen die Zeilen- und Spaltenzahl fest und löschen die Anzeige.
- **Zeile 8:** Mit `lcd.print()` schicken wir eine Zeichenkette auf das Display.
- **Zeile 13:** Die Funktion `lcd.setCursor()` erhält Zeile und Spalte als Parameter und legt fest, an welcher Stelle auf dem Display der nächste `print`-Befehl ausgegeben wird.

Im `loop`-Block des Testsketches nutzen wir den Arduino schon als Timer: Die Funktion `millis()` liefert zurück, wie viele Millisekunden seit dem letzten Reset oder Einschalten des Arduino vergangen sind. Wir schicken diesen Wert aufs Display und teilen ihn durch 1000, das heißt, wir haben einen Sekundenzähler. Diesen Wert werden wir als Basis für das Programmieren der Uhr- Funktionen verwenden.

## 6.3 Strings oder Rechnen mit Wörtern: Der Arduino als Digitaluhr (Teil 1)

Nachdem das Display nun einsatzbereit ist, kommen wir zu unserem eigentlichen Vorhaben. Wir möchten zunächst einmal die aktuelle Uhrzeit auf dem Display darstellen – und zwar wie bei einer Digitaluhr üblich im Format HH:MM:SS (je zwei Stellen für Stunde, Minute und Sekunde, getrennt durch Doppelpunkte).

Dazu bedienen wir uns eines neuen Datentyps, der Zeichenketten abspeichern und verarbeiten kann: der `String`-Typ. Eine `String`-Variable speichert eine Zeichenkette. Du hast bereits in Kapitel 5 gelernt, dass man Zahlen als Variablen unter einem bestimmten Namen abspeichern kann und dann im Sketch unter diesem Namen auf sie zugreifen und

mit ihnen rechnen kann. Dazu haben wir den Datentyp `int` verwendet. Diese Abkürzung steht für `Integer`. Solche Variablen sind immer ganzzahlige Werte. Wir können Texte oder Zeichen aber auch als Variable abspeichern. Diese sind dann vom Typ `String`. Man kann sie – genauso wie Zahlen – in Funktionsaufrufen verwenden oder Operationen damit ausführen. Du kannst das ausprobieren, indem du im Testsketch den `lcd.print()`-Befehl ersetzt:

```
1  String Teil1="Hallo";
2  String Teil2="Welt!";
3  lcd.print(Teil1+" "+Teil2);
```

Im Ergebnis steht wieder „Hallo Welt!“ auf dem Display, doch wir haben die Ausgabe aus zwei `String`-Variablen und einer `String`-Konstanten zusammengesetzt. Mit dem +-Operator kannst du beliebige Ausgabezeilen aus konstanten Textstückchen und `String`-Variablen zusammensetzen. Das wollen wir in Abschnitt 6.4 für die Anzeige der Uhrzeit nutzen.

## 6.4 Was schlägt die Stunde? Der Arduino als Digitaluhr (Teil 2)

Im Folgenden wirst du den Sketch für die digitale Anzeige der Uhrzeit kennenlernen. Bevor wir die einzelnen Funktionen genauer durchgehen, kannst du den ganzen Code erst einmal per Copy & Paste in die Arduino-Entwicklungsumgebung kopieren und dort abspeichern.

Dann setzt du in Zeile 40 (siehe folgender Code) die aktuelle Uhrzeit ein und schickst den Sketch an den Arduino. Wir müssen dem Programm die Startzeit mitgeben, da der Arduino keine eingebaute Uhr hat wie zum Beispiel dein PC oder Laptop. Außerdem kannst du in Zeile 49 eintragen, was in der ersten Zeile angezeigt werden soll (zum Beispiel deine Stadt). Auf dem Display deines Arduino ist nun eine Zeitanzeige zu sehen (Bild 6.9).

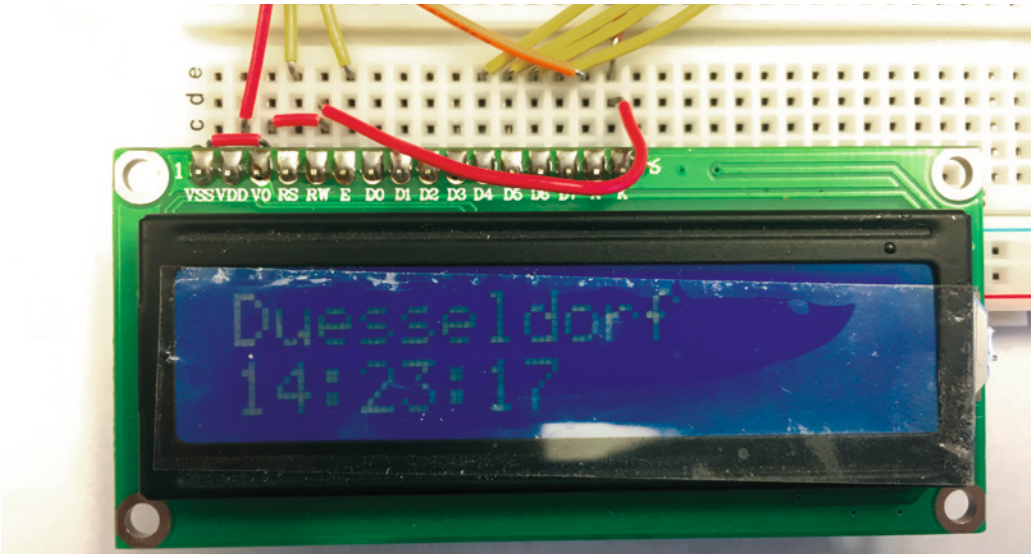


**Bild 6.9** Ort und Zeit werden auf dem LCD-Display angezeigt.

Dies ist der komplette Code für die Programmierung der Zeitanzeige:

```
// Arduino als Digitaluhr mit dem LCD-Display

#include <LiquidCrystal.h>

String number_to_string(long secs, int DIV, int MOD){
// Rechnet die Sekundenzahl um und wandelt das Ergebnis
// in einen String
// DIV=3600: Rechnet Sekunden in Stunden um
// DIV=60: Rechnet Sekunden in Minuten um
// Der Wert MOD gibt an, ab welcher Zahl die Anzeige
// wieder auf Null springt
// (Normalerweise 12 oder 24 beim Stundenzähler, 60 bei // Minuten und Sekunden

String Str;
unsigned long number=secs/DIV%MOD;

// Bei einstelliger Zahl noch eine Null voranstellen,
// sonst direkt umwandeln
if(number<10) Str='0'+String(number);
else Str=String(number);
return Str;
}

String timeString(unsigned long secs){
// Wandelt die Sekundenzahl secs um in einen String im
// Uhrzeit-Format
return
number_to_string(secs,60*60,24)+':'+number_to_string(secs,60,60)+':'+
}

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

```
32
33  // Unter diesem Kommentar die aktuelle Uhrzeit einsetzen.
34  // Die Variable Offset ist die Anzahl der Sekunden von 0
35  // Uhr bis zum gegenwärtigen Zeitpunkt.
36  // Sie wird zu den Sekunden seit Start addiert. Dadurch
37  // zeigt der Arduino beim Start nicht 0:00:00 an, sondern // Uhrzeit, die hier definiert
    wird:
38
39  // ---------------------UHRZEIT EINSETZEN-----------
40  long init_stunde=17; long init_minute=35; long init_sekunde= 0;
41  // ------------------------------------------------
42  unsigned long Offset=init_stunde*3600+init_minute*60+init_sekunde;
43
44
45  void setup() {
46  // LCD mit 16 Spalten und 2 Zeilen initialisieren
47  lcd.begin(16, 2);
48  // Text für erste Zeile ausgeben
49  lcd.print("Hamburg");
50  }
51
52  void loop() {
53  // Anzahl Sekunden seit dem letztem Reset berechnen
54  unsigned long secs=millis() /1000 ;
55
56  // Die mit der Funktion timeString aus den Sekunden und
57  // der Startzeit berechnete Uhrzeit ausgeben
58  lcd.setCursor(0, 1);
59  lcd.print(timeString(secs+Offset));
60  }
```

## 6.5 Bits und Bytes bis zum Überlaufen: Der Arduino als Digitaluhr (Teil 3)

Sehen wir uns nun den Code zur Programmierung der Zeitanzeige aus Abschnitt 6.4 einmal genauer an. Als Erstes fällt auf, dass wieder ein neuer Datentyp benutzt wird – und zwar `unsigned long`. Hierbei handelt es sich nur um ganze Zahlen, genau wie beim Typ `int`. Allerdings bedeutet `long`, dass doppelt so viel Speicherplatz für die Variable reserviert wird. Statt den 16 Bit einer `int`-Variable besitzt eine `long`-Variable 32 Bit. Wir benötigen die `long`-Variable, weil wir die Sekunden seit dem Start des Programms zählen wollen. Eine `int`-Variable kann maximal Werte bis $2^{16}$ = 65536 annehmen. Diese Anzahl von Sekunden ist bereits nach 18 Stunden gezählt. Dann würden die Variablen überlaufen und wieder auf null springen. Mit 32-Bit-Variablen passiert das erst nach 50 Tagen. Daher müssen wir für diese Anwendung die längeren, speicherintensiven `long`-Typen verwenden. Aus demselben Grund sagen wir dem Compiler mit dem Schlüsselwort `unsigned`, dass wir keine negativen Zahlen verwenden wollen, denn auch dadurch erweitert sich der Bereich bis zum Überlauf.

Leider können wir unsere Ausführungen zu Variablen und Datentypen an dieser Stelle nicht weiter vertiefen, da dies dem Rahmen des Buches sprengen würde. Nutze die Möglichkeit und vertiefe dein Wissen in Einstiegsbüchern oder Programmierkursen zu den Sprachen C und C++.

## 6.6 Wie Funktionen funktionieren: Der Arduino als Digitaluhr (Teil 4)

In Zeile 5 des Codes aus Abschnitt 6.4 definieren wir übrigens eine eigene Funktion. Das ist ein wichtiges Konzept bei der Programmierung. Bislang haben wir immer bereits vordefinierte Funktionen verwendet und aufgerufen. Wir können aber auch eigene Funktionen definieren und auf diese Weise sich wiederholende Aufgaben effizient von einem Funktionsaufruf erledigen lassen.

In unserem Falle definieren wir die Funktion `number_to_string()`, die eine Zahl (eine `int`- oder `long`-Variable) in einen `String` umwandelt. Daraus werden wir die Uhrzeit auf dem LCD-Display zusammensetzen. Es wird immer zuerst der Rückgabetyp genannt, dann der frei wählbare Funktionsname und in Klammern werden die Parameter angegeben, die an die Funktion übergeben werden sollen (immer mit Typangabe und Name). Das sieht wie folgt aus:

```
String number_to_string(long secs, int DIV, int MOD)
```

Danach folgt ein Block in geschweiften Klammern. Dort wird implementiert, was diese Funktion macht.

## 6.7 Ein wenig Zeitrechnung muss sein: Der Arduino als Digitaluhr (Teil 5)

Wir bleiben bei der Analyse des Codes aus Abschnitt 6.4. Für die Funktion `number_to_string` schauen wir uns erst einmal Zeile 15 an:

```
unsigned long number=secs/DIV%MOD;
```

Die Zeile sieht etwas kryptisch aus, aber wir werden sie dir Schritt für Schritt erklären. Wir nehmen die Sekundenzahl in der Variablen `secs` und teilen sie erst einmal durch den Wert `DIV`, der beim Aufruf der Funktion mit übergeben wurde. Damit kann man entweder Sekunden in Minuten oder Sekunden in Stunden umrechnen, je nachdem, was man für `DIV` einsetzt.

Danach folgt ein neuer Operator: Das Zeichen `%` nennt man Modulo- Operator. Das klingt vielleicht etwas kompliziert und mathematisch, aber eigentlich ist es ganz einfach. Die Zahl, die hinter dem `%` steht, ist wie eine Grenze, die nicht überschritten wird. Genauer gesagt ergibt die Modulo-Operation den Rest, der beim Teilen entsteht. Für den Sekundenzähler benutzen wir z. B. `%60`. Dies bewirkt, dass nach 59 Sekunden nicht 60 kommt, sondern der Zähler wieder bei 0 anfängt. Denn 60 geteilt durch 60 geht glatt auf und der Divisionsrest ist 0. Bei 120 passiert das Ganze wieder usw. Anstatt die Sekunden immer weiterzuzählen, springen wir also immer bei 60 wieder auf 0 zurück – genau wie eine Uhr eben. Das erledigt der Modulo-Operator ganz automatisch für uns.

In den Zeilen 19 und 20 sorgen wir nun dafür, dass der Rückgabe-String immer genau zwei Zeichen hat. Bei Zahlen kleiner als 10 ist er jedoch einstellig. Dann setzen wir einfach eine „0“ vorne dran.

Das in der Variablen `Str` gespeicherte Ergebnis wird dann in Zeile 21 an den Funktionsaufrufer zurückgegeben. Wir können nun überall im Code auf diese Funktion zurückgreifen, indem wir einfach `number_to_string()` mit den entsprechenden Werten aufrufen.

In Zeile 24 definieren wir noch eine weitere Funktion. Diese Funktion ist die wichtigste des ganzen Programms. Sie nimmt die absolute Sekundenanzahl als Grundlage und wandelt diese in eine Uhrzeit im HH:MM:SS-Format um. Dies erfolgt, indem ein `String` mit dem +-Operator zusammengesetzt wird, und dreimal die Funktion `number_to_string()` aufgerufen wird, jeweils mit den Parametern für Stunden, Minuten und Sekunden.

In Zeile 42 definieren wir einen Wert namens `Offset`, den wir noch auf die Sekundenzählung des Arduino aufschlagen. Da der Arduino mit 0 zu zählen beginnt, wäre die angezeigt Uhrzeit beim Start immer 00:00:00, was wir nicht möchten. Stattdessen soll hier die aktuelle Zeit angezeigt werden. Dazu behelfen wir uns mit einem Trick: Wir addieren die Zahl der Sekunden, die seit der Anzeige „0:00:00“ vergangen sind, zum Timer-Wert hinzu.

Den Rest wirst du jetzt sicherlich schnell verstehen. Im `loop`-Block nehmen wir die Laufzeit des Arduino aus der Funktion `millis()`, berechnen daraus die absolute Sekundenzahl, packen unseren `Offset` drauf und schicken das Ganze an die Funktion `timeString()`, die einen String im Uhrzeitformat daraus macht. Dieser String wird mittels `lcd.print()` ans LCD-Display geschickt.

# Stichwortverzeichnis

## K



## L



## M



## N



## O



## P



## R



## S

## T



## U



## V



## W



## Z