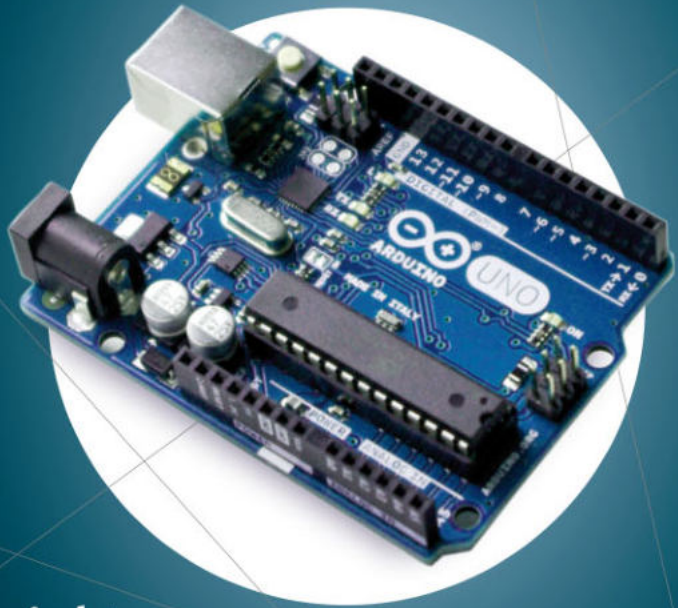


Erik Bartmann

Der
Arduino-
Bestseller in
4. Auflage

Mit
Arduino
die elektronische Welt
entdecken



48 Bastelprojekte
mit dem Arduino


bombini
verlag

Die Informationen in diesem Buch wurden mit größter Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Verlag, Autoren und Übersetzer übernehmen keine juristische Verantwortung oder irgendeine Haftung für eventuell verbliebene Fehler und deren Folgen.

Alle Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen. Der Verlag richtet sich im wesentlichen nach den Schreibweisen der Hersteller. Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Alle Rechte vorbehalten einschließlich der Vervielfältigung, Übersetzung, Mikroverfilmung sowie Einspeicherung und Verarbeitung in elektronischen Systemen. Kommentare und Fragen können Sie gerne an uns richten:

Bombini Verlags GmbH
Kaiserstraße 235
53113 Bonn
E-Mail: service@bombini-verlag.de

Copyright:
© 2021 by Bombini Verlag

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Umschlaggestaltung: Michael Oreal, Köln (www.oreal.de)
Satz: III-satz, Husby (www.drei-satz.de)

ISBN 978-3-946496-29-8

Die Siebensegmentanzeige

Wenn wir logische Zustände (*wahr* oder *falsch*) oder Daten (14, 2.5, »Hallo User«) in irgendeiner Form visualisieren wollten, müssen wir im ersten Fall LEDs ansteuern und im zweiten auf den Serial Monitor zurückgreifen. In der Elektronik gibt es neben LEDs weitere Anzeigeelemente, eines davon ist die Siebensegmentanzeige. Wie der Name schon vermuten lässt, besteht diese Anzeige aus sieben einzelnen Elementen, die in einer bestimmten Form angeordnet sind, um Ziffern und in beschränktem Maße auch Zeichen darstellen zu können. Der Einsatz dieser Anzeigeelemente ist weit verbreitet, deshalb ist es sinnvoll, sich mit der Funktionsweise und der Ansteuerung dieser Siebensegmentanzeige zu beschäftigen. Das Anzeigeelement ist bei den E-Bastlern sehr beliebt, nicht zuletzt deswegen, weil es sich dabei um sehr preiswerte Bauteile handelt.

Die Siebensegmentanzeige genau erklärt

In der nachfolgenden Abbildung ist der Aufbau einer solchen Anzeige schematisch dargestellt:

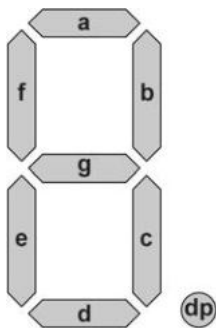


Abb. 1: Eine Siebensegmentanzeige mit Dezimalpunkt

Du kannst erkennen, dass jedes der sieben Segmente mit einem kleinen Buchstaben versehen wurde. Die Reihenfolge spielt zwar keine unmittelbare Rolle, doch die hier gezeigte Form hat sich etabliert und wird fast überall verwendet. Darum werden wir sie auch bei-

behalten. Wenn wir die einzelnen Segmente geschickt ansteuern, können wir unsere Ziffern von 0 bis 9 gut abbilden. Es sind auch Buchstaben möglich, auf die wir etwas später zu sprechen kommen werden. Du bist im Alltag bestimmt schon vielen dieser Siebensegmentanzeigen begegnet, ohne dass du weiter darüber nachgedacht hast. Du kannst beim nächsten Stadtbummel ja einmal auf diese Anzeigen achten. Du wirst schnell merken, an wie vielen Stellen sie zum Einsatz kommen. Hier eine kleine Liste der Einsatzmöglichkeiten.

- Preisanzeige an Tankstellen (sie zeigen irgendwie immer zu viel an ...)
- Zeitanzeige an hohen Gebäuden
- Temperaturanzeige
- Digitaluhren
- Blutdruckmessgeräte
- Elektronische Fieberthermometer
- Die Anzeigetafeln, mit denen bei Fußballspielen die nachzuspielenden Minuten angezeigt werden

In der folgenden Tabelle wollen wir für die zukünftige Programmierung festhalten, bei welchen Ziffern welches der sieben Segmente angesteuert werden muss:


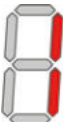

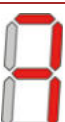



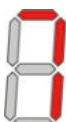


| Anzeige | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
|  | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
|  | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
|  | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
|  | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
|  | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Anzeige | a | b | c | d | e | f | g |
|  | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
|  | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
|  | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|  | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

Tabelle 1: Ansteuerungstabelle der sieben Segmente, um Ziffern darzustellen

Der Wert 1 in unserer Tabelle bedeutet nicht unbedingt *HIGH*-Pegel, sondern es handelt sich um die Ansteuerung des betreffenden Segments. Das kann entweder mit dem schon genannten *HIGH*-Pegel (+5V inklusive Widerstand) oder auch mit einem *LOW*-Pegel (0V) erfolgen. Du fragst dich jetzt bestimmt, wovon das abhängt, denn für eine Ansteuerung muss man sich ja entscheiden. Die Entscheidung wird uns durch den Typ der Siebensegmentanzeige abgenommen. Es gibt zwei unterschiedliche Ansätze, den der gemeinsamen Kathode und den der gemeinsamen Anode.

Bei einer gemeinsamen Kathode sind alle Kathoden der einzelnen LEDs einer Siebensegmentanzeige intern zusammengeführt und werden extern mit Masse verbunden. Die Ansteuerung der einzelnen Segmente erfolgt über Widerstände, die zum Aufleuchten entsprechend mit *HIGH*-Pegel verbunden werden. Wir verwenden in unserem folgenden Beispiel aber eine Siebensegmentanzeige mit einer gemeinsamen Anode. Hier ist es genau andersherum als beim vorherigen Typ. Alle Anoden der einzelnen LEDs sind intern miteinander verbunden und werden für das Aufleuchten mit *HIGH*-Pegel verbunden. Die Ansteuerung erfolgt über entsprechend dimensionierte Widerstände der einzelnen Kathoden der LEDs, die nach außen geführt werden. Die nachfolgende Abbildung zeigt beide Varianten:

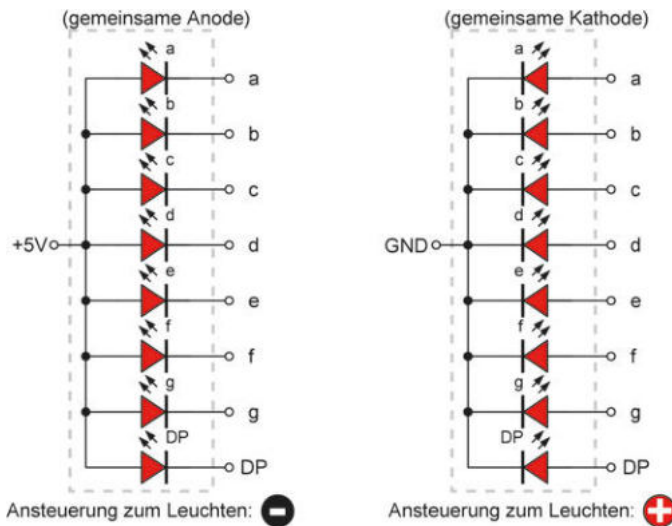


Abb. 2: Die Siebensegmentanzeige mit gemeinsamer Anode und Kathode

Auf der nächsten Abbildung siehst du, dass alle Anoden der Siebensegmentanzeige mit gemeinsamer Versorgungsspannung von +5V verbunden sind. Die Kathoden werden später mit den digitalen Ausgängen deines Arduino-Boards verbunden und entsprechend der eben gezeigten Ansteuerungstabelle mit unterschiedlichen Spannungspegeln versorgt. Wir verwenden in unserem Versuchsaufbau eine Siebensegmentanzeige mit gemeinsamer Anode des Typs SA 39-11 SRWA. Ich habe die Pinbelegung dieser Anzeige in der folgenden Abbildung aufgezeigt:

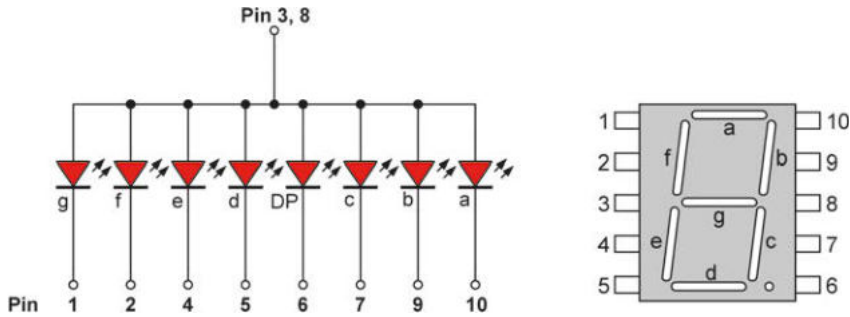


Abb. 3: Die Ansteuerung der Siebensegmentanzeige vom Typ SA 39-11 SRWA

In der linken Grafik sind die verwendeten Pins der Siebensegmentanzeige zu sehen, in der rechten Grafik ist die Pinbelegung des verwendeten Typs dargestellt. Die Bezeichnung *DP* ist übrigens die Abkürzung für Dezimalpunkt.

Weitere nützliche Hinweise zu Siebensegmentanzeigen

Siebensegmentanzeigen gibt es in schier unüberschaubaren Varianten. Sie sind in unterschiedlichen Farben erhältlich, beispielsweise in:

- Gelb
- Grün
- Rot
- super helles Rot

Beim Kauf immer auf die Anschlussvarianten achten: gemeinsame Anode *oder* gemeinsame Kathode!

Was wir brauchen

Für dieses Bastelprojekt benötigen wir die folgenden Bauteile:


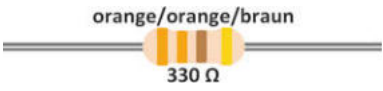
| Bauteil | Bild |
|--|---|
| Siebensegmentanzeige (zum Beispiel Typ SA 39-11 SRWA mit gemeinsamer Anode) 1x |  |
| Widerstand 330Ω 7x |  |

Tabelle 2: Bauteilliste

Der Schaltplan

Der Schaltplan ähnelt wieder der Ansteuerung unseres Lauflichts (siehe [Bastelprojekt 6](#)), doch die einzelnen digitalen Ausgänge werden natürlich nicht nacheinander angesteuert,

sondern gleichzeitig in einer bestimmten Kombination, denn sie sollen ja eine erkennbare Ziffer anzeigen:

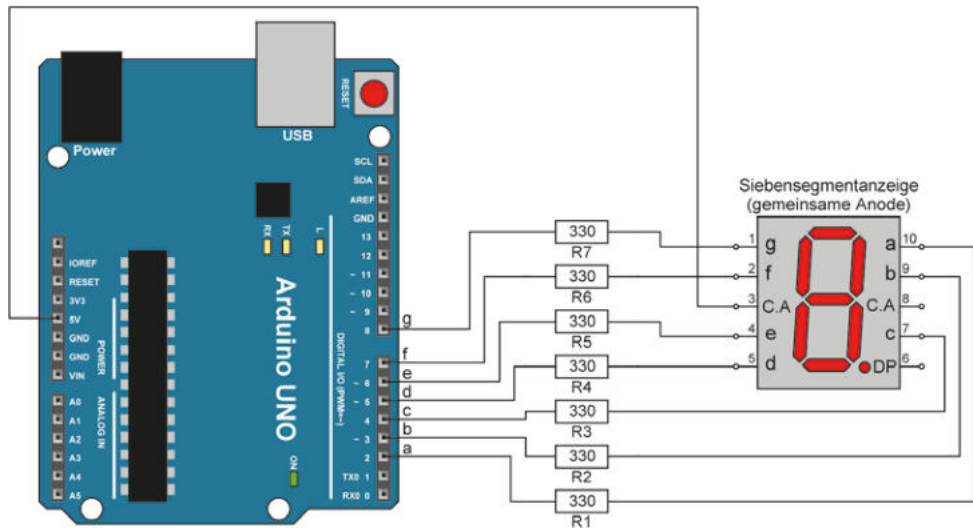


Abb. 4: Der Schaltplan zur Ansteuerung der Siebensegmentanzeige

Der Schaltungsaufbau

Der Schaltungsaufbau zeigt auf dem Arduino Discoveryboard die sieben Widerstände auf dem kleinen Breadboard und die eigentliche Siebensegmentanzeige am unteren Rand (die Ausrichtung des Arduino Discoveryboards ist diesmal um 90 Grad gedreht, da ansonsten die Anzeige etwas schwerer abzulesen wäre):

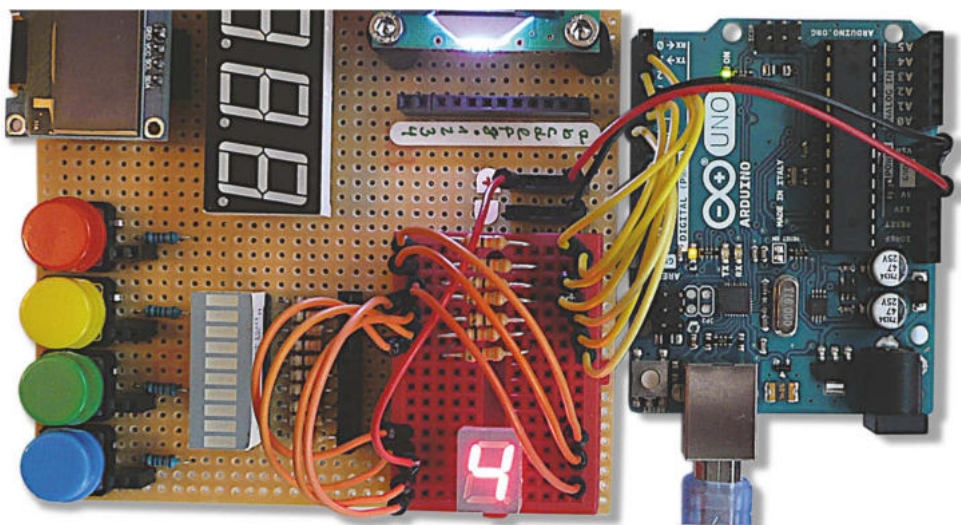


Abb. 5: Der Schaltungsaufbau zur Ansteuerung der Siebensegmentanzeige

Du fragst dich sicherlich, warum ich eine separate Siebensegmentanzeige auf dem kleinen Breadboard platziert habe, obwohl das Arduino Discoveryboard von Hause aus über eine vierstellige Siebensegmentanzeige verfügt. Dazu aber später mehr, denn es handelt sich dabei um vier zusammenschaltete Segmente, die anders angesprochen werden müssen. Mit diesen technischen Grundlagen wenden wir uns dem Sketch zu.

Der Arduino-Sketch

Der folgende Sketch vereinfacht die Ansteuerung der sieben Segmente derart, dass wir wieder ein Array zu Hilfe nehmen:

```
int segmente[10][7] = {{1, 1, 1, 1, 1, 1, 0}, // 0
                      {0, 1, 1, 0, 0, 0, 0}, // 1
                      {1, 1, 0, 1, 1, 0, 1}, // 2
                      {1, 1, 1, 1, 0, 0, 1}, // 3
                      {0, 1, 1, 0, 0, 1, 1}, // 4
                      {1, 0, 1, 1, 0, 1, 1}, // 5
                      {1, 0, 1, 1, 1, 1, 1}, // 6
                      {1, 1, 1, 0, 0, 0, 0}, // 7
                      {1, 1, 1, 1, 1, 1, 1}, // 8
                      {1, 1, 1, 1, 0, 1, 1}}; // 9

int pinArray[] = {2, 3, 4, 5, 6, 7, 8};

void setup() {
  for(int i = 0; i < 7; i++)
    pinMode(pinArray[i], OUTPUT);
}

void loop() {
  for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 7; j++)
      digitalWrite(pinArray[j], (segmente[i][j]==1)?LOW:HIGH);
    delay(1000); // Pause von 1 Sekunde
  }
}
```

Den Code verstehen

Da für jede einzelne Ziffer von 0 bis 9 die Informationen über die anzusteuernenden Segmente gespeichert sein müssen, bietet sich ein zweidimensionales Array an. Diese Werte werden in der globalen Variable *segmente* zu Beginn des Sketches gespeichert:

```
int segmente[10][7] = {{...},
                      ...
                      {...}};
```

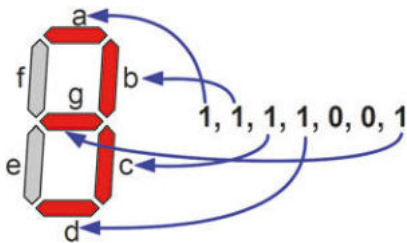

Das Array umfasst 10 x 7 Speicherplätze, wobei jedes einzelne über die Koordinaten angesprochen werden kann:

```
segmente[x][y]
```

Die x-Koordinate steht für alle Ziffern von 0 bis 9 (entspricht zehn Speicherplätzen) und die y-Koordinate für alle Segmente a bis g (entspricht 7 Speicherplätzen). Wenn wir beispielsweise die anzusteuernenden Segmente der Ziffer 3 ermitteln möchten, lässt sich das durch die folgende Zeile bewerkstelligen:

```
segmente[3][y]
```

Für die Variable müssen y die Werte von 0 bis 6 über eine *for*-Schleife eingesetzt werden. Die Segmentdaten lauten dann wie folgt:



Ich habe hier der besseren Übersicht halber nur die Pfeile der angesteuerten Segmente eingezeichnet. Aber halt! Hatten wir nicht gesagt, dass dieser Typ der Siebensegmentanzeige eine gemeinsame Anode hat?! Jetzt steht aber im *segment*-Array an der Stelle eine 1, an der eigentlich eine Ansteuerung mit Masse erfolgen sollte. Ist das nicht so? Der ersten Aussage kann ich vollkommen zustimmen. Bei der zweiten haben wir etwas übersehen. Ich hatte erwähnt, dass eine 1 nicht unbedingt *HIGH*-Pegel bedeutet, sondern lediglich, dass dieses Segment anzusteuern ist. Bei einer Siebensegmentanzeige mit gemeinsamer Kathode wird mit *HIGH*-Pegel angesteuert, um das gewünschte Segment leuchten zu lassen, bei einer Siebensegmentanzeige mit gemeinsamer Anode erfolgt das mittels *LOW*-Pegel. Und genau dazu dient die folgende Zeile:

```
digitalWrite(pinArray[j], (segmente[i][j]==1)?LOW:HIGH);
```

Ist die Information eine 1, dann wird *LOW* als Argument an die *digitalWrite*-Funktion übergeben, andernfalls *HIGH*. Bei *LOW* wird das entsprechende Segment leuchten, wobei es bei einem *HIGH* so gesteuert wird, dass es dunkel bleibt. Unser Sketch zeigt im Sekundentakt alle Ziffern von 0 bis 9 an. Dazu wird folgender Code verwendet:

```
for(int i = 0; i < 10; i++) {
  for(int j = 0; j < 7; j++)
    digitalWrite(pinArray[j], (segmente[i][j]==1)?LOW:HIGH);
}
```

```

    delay(1000); // Pause von 1 Sekunde
}

```

Die äußere Schleife mit der Laufvariablen *i* wählt die anzuzeigende Ziffer im Array aus und die innere Schleife mit der Laufvariablen *j* die anzusteuernden Segmente.

Wir verbessern den Sketch

Die Ansteuerung der einzelnen Segmente pro Ziffer erfolgte über ein zweidimensionales Array, bei dem die erste Dimension zur Selektion der gewünschten Ziffer und die zweite Dimension für die einzelnen Segmente herangezogen wurde. Im folgenden Sketch wollen wir das Ganze mit einem eindimensionalen Array realisieren. Wie das funktionieren soll? Nun, das ist recht simpel, denn du kennst dich ja mittlerweile ganz gut mit Bits und Bytes aus. Die Segmentinformation speichern wir jetzt in einem einzigen Wert ab. Welcher Datentyp würde sich für dieses Vorhaben anbieten? Du hast es mit einer Siebensegmentanzeige plus einem einzigen Dezimalpunkt zu tun, den wir aber außen vor lassen wollen. Das wären dann sieben Bits, die wunderbar in einem einzigen Byte mit acht Bits Platz fänden. Jedes einzelne Bit weisen wir einfach einem Segment zu und können so mit einem einzigen Byte-Wert alle benötigten Segmente ansteuern. An dieser Stelle möchte ich dir noch eine interessante Möglichkeit zeigen, direkt über eine Bit-Kombination eine Variable zu initialisieren:

```

void setup(){
    Serial.begin(9600);
    byte a = B10001011;    // Variable deklarieren + initialisieren
    Serial.println(a, BIN); // Als Binärwert ausgeben
    Serial.println(a, HEX); // Als Hexwert ausgeben
    Serial.println(a, DEC); // Als Dezimalwert ausgeben
}

void loop(){ /* leer */ }

```

Die entscheidende Zeile ist diese:

```
byte a = B10001011;
```

Das Merkwürdige oder eigentlich Geniale daran ist die Tatsache, dass du über den vorangestellten Buchstaben *B* eine Bit-Kombination angeben kannst, die der Variablen zur Linken zugewiesen wird. Das vereinfacht die Sache ungemein, wenn du zum Beispiel eine Bit-Kombination kennst und sie speichern möchtest. Andernfalls hättest du erst den Binärwert in eine Dezimalzahl umwandeln müssen, um sie zu speichern. Dieser Zwischenschritt entfällt jetzt. Hier ist aber etwas sehr merkwürdig! Der Datentyp *byte* ist doch eine Ganzzahl. Datentyp und Ganzzahlen bestehen jedoch immer aus Ziffern von 0 bis 9. Warum kann man jetzt den Buchstaben *B* voranstellen und eine Bit-Kombination folgen lassen? Oder haben wir es hier vielleicht mit einer Zeichenkette zu tun?

Der Datentyp *byte* ist ein Ganzzahldatentyp. Damit hast du vollkommen recht. Unrecht hast du jedoch mit deiner Vermutung, dass es sich um eine Zeichenkette handelt. Sie würde auch mit doppelten Anführungszeichen umschlossen. Es muss sich also um etwas anderes handeln. Irgendeine Idee? Ich sage nur *#define*. Na, klingelt es? Okay, schau mal. Es gibt eine Datei in den Tiefen des Arduino, die sich *binary.h* nennt und sich im Verzeichnis ...*Arduino\hardware\arduino\avr\cores\arduino* befindet. Ich zeige dir einen kurzen Ausschnitt dieser Datei, denn sie enthält sehr viele Zeilen; sie alle zu zeigen, wäre Platzverschwendung:

```

20 #ifndef Binary_h
21 #define Binary_h
22
23 #define B0 0
24 #define B00 0
25 #define B000 0
26 #define B0000 0
27 #define B00000 0
28 #define B000000 0
29 #define B0000000 0
30 #define B00000000 0
31 #define B1 1
32 #define B01 1
33 #define B001 1
34 #define B0001 1
35 #define B00001 1
36 #define B000001 1
37 #define B0000001 1
38 #define B00000001 1

```

In dieser Datei befinden sich alle möglichen Bitkombinationen für die Werte von 0 bis 255, die dort als symbolische Konstanten definiert wurden. Bevor ich zum eigentlichen Thema zurückkomme, möchte ich noch die folgenden Zeilen erläutern:

```

Serial.println(a, BIN); // Als Binärwert ausgeben
Serial.println(a, HEX); // Als Hexwert ausgeben
Serial.println(a, DEC); // Als Dezimalwert ausgeben

```

Die *println*-Funktion kann noch ein weiteres Argument neben dem auszugebenden Wert entgegennehmen, der durch ein Komma getrennt angegeben werden kann. Ich habe die drei wichtigsten angeführt. Weitere findest du auf der Befehlsreferenzseite von Arduino im Internet. Die Erläuterungen finden sich als Kommentare hinter den Befehlszeilen. Die Ausgabe im Serial Monitor ist dann Folgende:

```

10001011
8B
139

```

Kommen wir jetzt endlich zur Ansteuerung der Siebensegmentanzeige über das eindimensionale Array. Ich zeige dir vorab wieder den kompletten Sketch, den wir gleich analysieren werden:

```

byte segmente[10]= {B01111110, // 0
                    B00110000, // 1
                    B01101101, // 2
                    B01111001, // 3
                    B00110011, // 4
                    B01011011, // 5
                    B01011111, // 6
                    B01110000, // 7
                    B01111111, // 8
                    B01111011}; // 9

int pinArray[] = {2, 3, 4, 5, 6, 7, 8};

void setup() {
  for(int i = 0; i < 7; i++)
    pinMode(pinArray[i], OUTPUT);
}

void loop() {
  for(int i = 0; i < 10; i++) { // Ansteuern der Ziffer
    for(int j = 6; j >= 0; j--){ // Abfragen der Bits für die Segmente
      digitalWrite(pinArray[6 - j], bitRead(segmente[i], j) == 1?LOW:HIGH);
    }
    delay(500); // Eine halbe Sekunde warten
  }
}

```

In der folgenden Abbildung kannst du sehr gut erkennen, welches Bit innerhalb des Bytes für welches Segment verantwortlich ist:

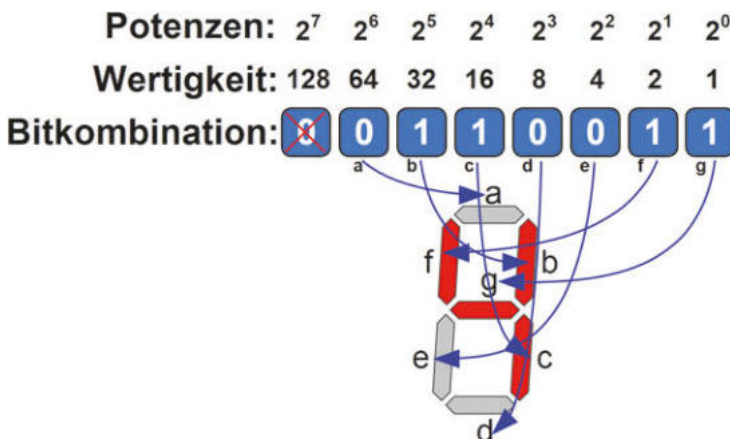


Abb. 6: Ein Byte steuert die Segmente der Anzeige (hier das Beispiel für die Ziffer 4).

Da wir lediglich sieben Segmente anzusteuern haben und ich den Dezimalpunkt nicht berücksichtige, habe ich das MSB (erinnere dich, MSB = Bit mit dem höchsten Wert) konstant bei allen Array-Elementen mit dem Wert 0 versehen. Das Entscheidende geschieht wieder – wie sollte es anders sein – innerhalb der *loop*-Funktion. Werfen wir einen genaueren Blick darauf:

```
void loop() {
  for(int i = 0; i < 10; i++) { // Ansteuern der Ziffer
    for(int j = 6; j >= 0; j--){ // Abfragen der Bits für die Segmente
      digitalWrite(pinArray[6 - j], bitRead(segmente[i], j) == 1?LOW:HIGH);
    }
    delay(500); // Eine halbe Sekunde warten
  }
}
```

Die äußere *for*-Schleife mit der Laufvariablen *i* steuert die einzelnen Ziffern von 0 bis 9 an. Das war auch in der ersten Lösung so realisiert worden. Jetzt kommt jedoch der abweichende Code. Die innere *for*-Schleife mit der Laufvariablen *j* ist für das Auswählen des einzelnen Bits innerhalb der selektierten Ziffer zuständig. Ich fange dabei auf der linken Seite mit Position 6 an, die für das Segment *a* zuständig ist. Da jedoch das Pin-Array an Indexposition 6 den Pin 8 für Segment *g* verwaltet, muss die Ansteuerung entgegengesetzt laufen. Das geschieht mittels der Subtraktion von der Zahl 6, da ich das Pin-Array aus dem ersten Beispiel so übernehmen wollte:

```
pinArray[6 - j]
```

Nun kommen wir zu einer interessanten Funktion, die es uns erlaubt, ein einzelnes Bit in einem Byte abzufragen. Sie lautet wie folgt:

Befehl
Wert
Bitposition

bitRead (139, 3) ;

Abb. 7: Der Befehl bitRead

In diesem Beispiel wird für den dezimalen Wert 139 (binär: 10001011) das Bit an Position 3 ermittelt. Die Zählung beginnt bei Index 0 am *LSB* (Least Significant Bit) auf der rechten Seite. Der Rückgabewert wäre demnach 1. Durch die folgende Befehlszeile wird überprüft, ob die selektierte Bit-Position 1 zurückgibt:

```
digitalWrite(pinArray[6 - j], bitRead(segmente[i], j) == 1?LOW:HIGH);
```

Falls dies der Fall ist, wird der ausgewählte Pin mit *LOW*-Pegel angesteuert, was bedeutet, dass das Segment leuchtet. Nicht vergessen: *Gemeinsame Anode!* Kannst du den Unter-

schied zwischen beiden Lösungen einmal formulieren? Er könnte wie folgt lauten: Okay, in der ersten Version mit dem zweidimensionalen Array wurde durch die erste Dimension die anzuzeigende Ziffer ausgewählt und über die zweite die anzusteuernenden Segmente. Diese Information steckte in den einzelnen Array-Elementen. Bei der zweiten Version wurde ebenfalls die anzuzeigende Ziffer über die erste Dimension ausgewählt. Da es sich um ein eindimensionales Array handelt, ist dies jedoch die einzige Dimension. Die Information zur Ansteuerung der Segmente ist jetzt in den einzelnen Byte-Werten enthalten. Was vorher durch die Array-Elemente der zweiten Dimension erfolgte, wird jetzt über die Bits eines Wertes gelöst.

Wie kann es weitergehen?

Nun, wenn man also eine einzige Siebensegmentanzeige ansteuern möchte – und ich lasse den Dezimalpunkt einmal außen vor – dann sind in Summe sieben digitale Leitungen erforderlich. Eine einzige Stelle ist jedoch sicherlich etwas wenig, um einen mehrstelligen Wert anzuzeigen. Bei einer zweistelligen Anzeige müssten also 14 Leitungen, bei einer dreistelligen – kurz überlegen – 21 Leitungen, und so weiter zur Verfügung gestellt werden. Das ist in der Praxis nicht ohne immensen Aufwand zu realisieren. Okay, es gibt da zum Beispiel den Arduino MEGA 2560 Rev3, der über 54 digitale Pins verfügt. Wow, das ist doch fantastisch! Doch warum mit Kanonen auf Spatzen schießen, wenn es auch eleganter geht? Erinnern wir uns an das Schieberegister, das kaskadierbar ist, was eine Hintereinanderschaltung beziehungsweise Verkettung mehrerer Schieberegister bedeutet. Das ist doch ein sehr guter Ansatz. Die zweite Möglichkeit besteht im sogenannten *Multiplexing*, das bei der vierstelligen Siebensegmentanzeige auf dem Arduino Discoveryboard zur Anwendung kommt.



Was bedeutet Multiplexing in der Elektronik?

Das Multiplexing ist eine besondere Methode der Signalübertragung, bei dem mehrere Signale zusammengefasst und simultan über ein Medium übertragen werden. Was hat das mit einer mehrstelligen Siebensegmentanzeige zu tun? Bei mehrstelligen Siebensegmentanzeigen bedeutet Multiplexing, dass nicht alle Anzeigen gleichzeitig eingeschaltet, also sichtbar sind, sondern immer nur eine einzige für eine sehr kurze Zeit. Erfolgt dieser Wechsel zwischen dem Anzeigen der vorhandenen Stellen jedoch schneller, als das menschliche Auge wahrnehmen kann, dann scheinen alle Anzeigen gleichzeitig sichtbar zu sein, obwohl immer nur eine einzige für eine kurze Zeit angesteuert wird und somit aufleuchtet.

Dazu mehr im folgenden **Bastelprojekt 18**.

Troubleshooting

Falls die Anzeige nicht den Ziffern von 0 bis 9 entspricht oder unsinnige Kombinationen angezeigt werden, überprüf Folgendes:

- Überprüf deine Steckverbindungen auf dem Breadboard, ob sie wirklich der Schaltung entsprechen.
- Achte auf mögliche Kurzschlüsse.

- Überprüf noch einmal den Sketch-Code auf Richtigkeit.
- Wenn unsinnige Zeichen in der Anzeige erscheinen, hast du möglicherweise die Steuerleitungen der einzelnen Segmente vertauscht. Kontrolliere noch einmal die Verdrahtung anhand des Schaltplans oder des Datenblatts der Siebensegmentanzeige.
- Hast du das *segmente*-Array mit den richtigen Werten initialisiert?

Was haben wir gelernt?

- In diesem Bastelprojekt bist du mit den Grundlagen der Ansteuerung einer Siebensegmentanzeige vertraut gemacht worden.
- Über die Initialisierung eines Arrays hast du die einzelnen Segmente der Anzeige definiert, um sie später komfortabel ansteuern zu können.
- Die Header-Datei *binary.h* beinhaltet viele symbolische Konstanten, die du in deinen Sketches verwenden kannst.
- Du hast gesehen, dass über die *println*-Methode durch Anfügen eines weiteren Arguments (*BIN*, *HEX* oder *DEC*) ein auszugebender Wert in eine Zahl einer anderen Zahlenbasis konvertiert werden kann.
- Mit der Funktion *bitRead* kannst du einzelne Bits eines Wert auf ihren Zustand abfragen.

Workshop zur Siebensegmentanzeige

Erweitere die Programmierung des Sketches so, dass in der Anzeige neben den Ziffern von 0 bis 9 auch bestimmte Buchstaben angezeigt werden können. Dies ist zwar nicht für das gesamte Alphabet möglich, doch überlege einmal, welche Buchstaben sich hierfür eignen könnten. Es folgen ein paar Beispiele:

