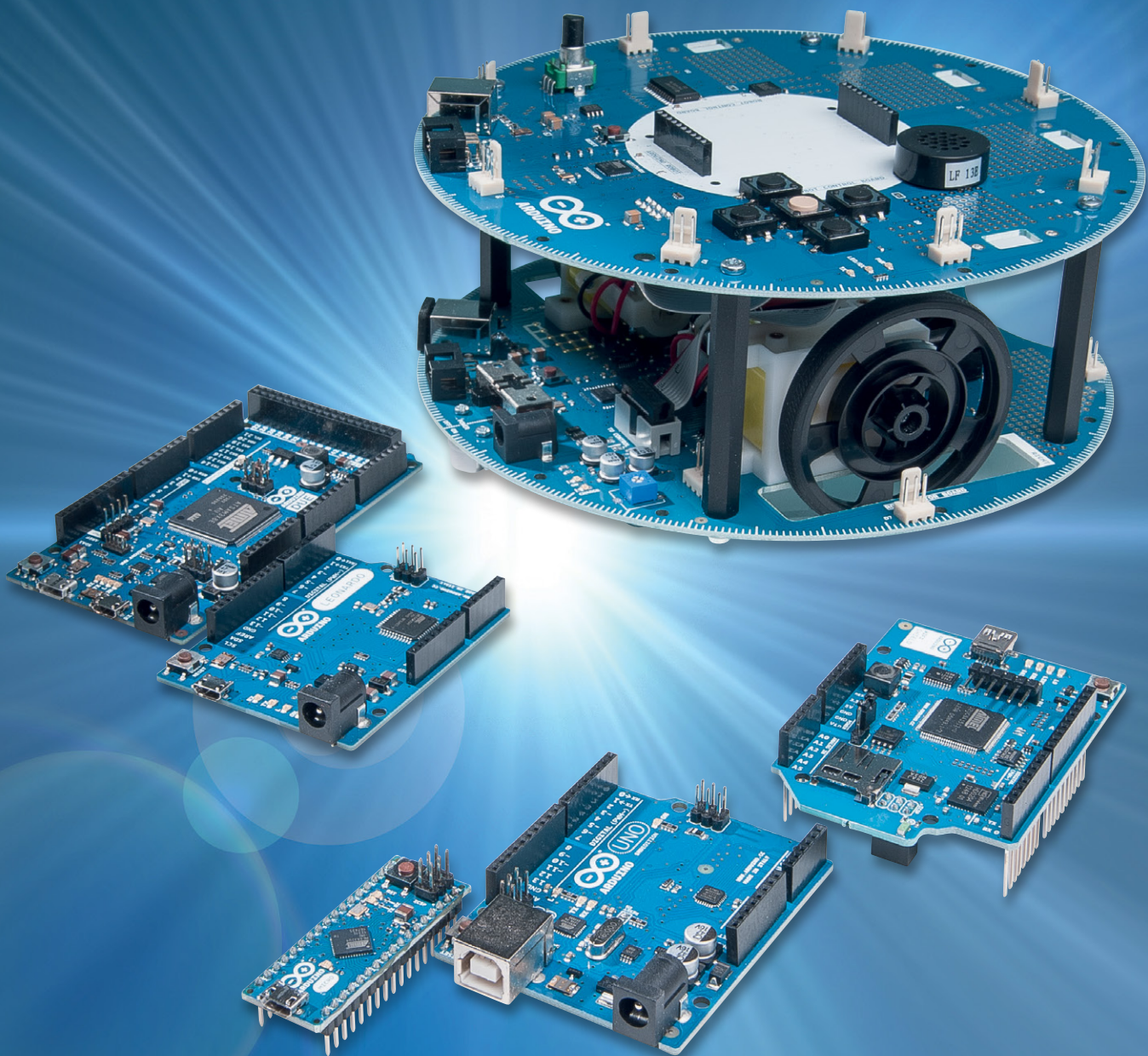




Arduino verstehen und anwenden

Teil 2: Die Programmierumgebung





Im zweiten Teil der Beitragsreihe zum Thema Arduino wird die IDE (Integrated Development Environment), also die sogenannte Entwicklungsumgebung, genauer betrachtet.

In den Abschnitten

- Die Entwicklungsumgebung auf dem PC,
- „Arduino-Processing“ und „C“: ein Vergleich,
- Die Funktionen der IDE,
- Die Standard-Beispielprogramme der IDE,
- Fehlerbeseitigung

werden zunächst alle wichtigen Eigenschaften dieser Entwicklungsumgebung dargestellt.

Abschließend entstehen dann mit den Abschnitten

- Der Alarmanlagensimulator,
 - Intuitive Programmierung: der Arduino sendet Morsesignale
- erste eigene Programme.

Die Entwicklungsumgebung auf dem PC

Die Programmierung des Arduino erfolgt über eine speziell für die Arduino-Boards entwickelte IDE. Diese Entwicklungsumgebung ist mit Sicherheit einer der wesentlichen Erfolgsfaktoren des Arduino-Projektes.

Die Oberfläche kann einerseits sehr intuitiv bedient werden und erleichtert daher den Einstieg in die Welt der Mikrocontrollerprogrammierung ganz erheblich. Andererseits basiert sie auf der im Controllerbereich am weitesten verbreiteten Programmiersprache „C“.

Anders als beispielsweise beim Einsatz von BASCOM kann der Anwender später somit problemlos auf professionelle IDEs wie etwa das ATMEL-Studio umsteigen. Die mit der Arduino-Umgebung erlernten Befehle und Strukturen können dort weitestgehend wiederverwendet werden. Das bereits erlernte Programmierfachwissen ist also nicht verloren.

Für die Programmierung der Arduino-Boards kommt „Processing“ zum Einsatz. Dabei handelt es sich aber eben nicht um eine spezielle Anfänger-Programmiersprache, sondern prinzipiell um klassisches „C“. Um den Einstieg zu erleichtern, wurden aber einige spezielle Prozeduren und Routinen implementiert.

Die bei vielen Einsteigern gefürchtete „Unlesbarkeit“ von C-Code wurde so wesentlich entschärft.

„Arduino-Processing“ und „C“: ein Vergleich

Der direkte Vergleich zeigt die Ähnlichkeit von Processing mit C. Es wird aber auch deutlich, dass Processing-Programme für Einsteiger wesentlich leichter zu lesen sind. Die in C so beliebten Abkürzungen werden vermieden, sie werden durch wesentlich leichter verständliche Anweisungen ersetzt.

So ist ein einfaches Arduino-Programm mit Grundkenntnissen in Englisch bereits recht gut lesbar:

```
// blink

int led = 13;

void setup()
{ pinMode(led, OUTPUT);
}

void loop()
{ digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Hier kann man ohne große Vorkenntnisse schon vermuten, dass eine Leuchtdiode (LED) in einem bestimmten Rhythmus geschaltet wird. Ein entsprechendes Standard-C-Programm sieht da schon wesentlich kryptischer aus:

```
// blink

#define F_CPU 16e6
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{   DDRB = 0b1;
    while (1)
    {   PORTB|=(1<<0);
        _delay_ms(1000);
        PORTB&=~(1<<0);
        _delay_ms(1000);
    }
    return 0;
}
```

Insbesondere die für C typischen Bitmanipulationen und die logischen Operatoren (|, &, ~, << etc.) lassen den embedded C-Code schwer lesbar erscheinen. Andererseits wird aber auch bereits bei diesem einfachen Programmbeispiel die enge Verwandtschaft zwischen Arduino-Processing und C deutlich. In der Tat ist es sogar so, dass sich im Hintergrund der einfachen Arduino-IDE ein kompletter AVR-GCC-Compiler verbirgt.

Die simplen „Einstiegsanweisungen“ wie „digitalWrite()“ sind dabei nur vorgefertigte Funktionen und Unterprogramme von der gleichen Art, wie sie in späteren Abschnitten zu dieser Artikelreihe noch näher erläutert werden.

Die Funktionen der IDE

Wie bereits im ersten Teil dieser Artikelserie erwähnt, kann die jeweils aktuelle Version der Entwicklungsumgebung kostenlos unter

<http://arduino.cc/en/Main/Software> heruntergeladen werden [1].

Nach dem Download liegt das komplette Programmpaket als komprimiertes ZIP-Archiv vor und kann in ein beliebiges Verzeichnis extrahiert werden (Bild 1).

Das Verzeichnis enthält dann alle zur Programmierung des Arduinos erforderlichen Komponenten.

**TIPP I:**

Beachten Sie, dass die IDE-Releases oft nicht abwärtskompatibel sind. Das heißt, ein Sketch, der mit Version 1.0.3 tadellos funktionierte, kann beispielsweise mit der Version 1.0.5 Probleme bereiten. Man sollte also immer im Auge behalten, welcher Sketch mit welcher IDE-Release erstellt wurde.

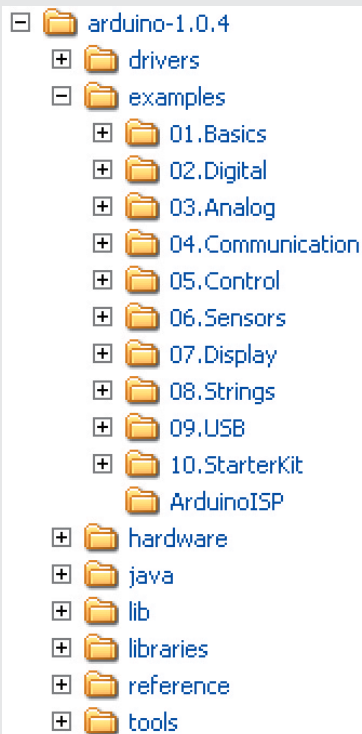


Bild 1: Das Arduino-Verzeichnis

Darüber hinaus sind unter „examples“ noch verschiedene Beispielprogramme verfügbar.

Das Unterverzeichnis „libraries“ enthält nützliche Programmbibliotheken, die unter anderem die Ansteuerung von

- Modellbauservos.
- Schrittmotoren.
- LCD-Anzeigen.
- externen EEPROM-Speichern
- und vielem mehr ...

unterstützen. Dieses Verzeichnis ist für den Anwender auch deshalb von besonderer Bedeutung, da hier später weitere nützliche Software-Bibliotheken eingestellt werden können. Weitere Details dazu finden sich z. B. im Mikrocontroller-Onlinekurs [2].

Der Start der IDE erfolgt über die Datei arduino.exe. Natürlich kann man sich dazu auch eine Verknüpfung auf dem Desktop erstellen.

Nach dem Starten meldet sich das Programm mit einem Infobild (Bild 2).

Danach wird die Programmierumgebung angezeigt (Bild 3). Da das erste, noch leere Programm noch keinen Namen hat, wird ein Default-Wert verwendet:

sketch_MonatTagx

X steht dabei für einen Buchstaben aus der fortlaufenden Buchstabenreihe x = a, b, c ... für den 1., 2., 3. ... neuen Sketch des aktuellen Tages.

Zur Info:

Anwendungsprogramme werden im Arduino-Umfeld auch als „Sketch“ bezeichnet.

Die für die Programmierung eines Arduino erforderlichen nächsten Schritte sind bereits aus dem ersten Beitrag bekannt:

1. Auswahl der richtigen Arduino-Board-Version
2. Auswahl des verwendeten virtuellen COM-Ports

Sollte es beim Start der IDE unerwartete Probleme geben, so kann man im Arduino-Online-Forum kompetente Hilfe erhalten [3].

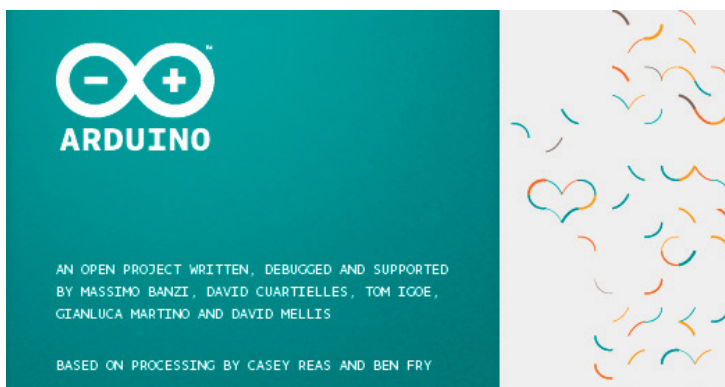


Bild 2: Start der Programmierumgebung mit den Credentials



Bild 3: Programmierumgebung mit Default-Einstellungen

Die Funktionen der IDE

In Ergänzung zum ersten Beitrag sollen hier nun auch die anderen Funktionen der IDE detaillierter dargestellt werden.

Die Bedeutungen der Symbole am oberen Rand der IDE können Bild 4 entnommen werden.

Die zum Teil bereits im ersten Beitrag verwendeten Icons haben die folgenden Funktionen:

- **Prüfen/Compilieren**

Mit dieser Schaltfläche kann der aktuell geöffnete Sketch compiliert, d. h. in Maschinensprache übersetzt werden. Dies ist insbesondere nützlich, wenn man ein Programm z. B. auf Syntaxfehler hin testen möchte, ohne dass ein Arduino-Board mit dem PC verbunden ist.

- **Aktuellen Sketch zum Controller senden**

Dieses Icon compiliert und sendet den aktuellen Sketch direkt zum Arduino. Ein vorheriges, separates Übersetzen ist also nicht erforderlich.

- **Neuer Sketch**

Hier kann ein neuer Sketch begonnen werden.

- **Sketch öffnen**

Mit diesem Icon wird ein bereits abgespeicherter Sketch in die IDE geladen.

- **Aktuellen Sketch speichern**

Das Speichern eines Sketches erfolgt mit diesem Icon.



· Seriellen Datenmonitor starten

Hier kann der serielle Datenmonitor gestartet werden. Er dient zur Kommunikation zwischen Controller und PC und wird häufig auch zu Test- und Prüfzwecken eingesetzt.

Die Standard-Beispielprogramme der IDE

Die Standard-IDE enthält bereits eine umfangreiche Sammlung von Beispielprogrammen (Bild 5).

Es ist sicher immer eine gute Idee, diese Beispielprogramme genauer unter die Lupe zu nehmen. Sehr häufig findet sich hier bereits eine geeignete Basis für neue Projekte. Die Erfahrung zeigt sogar, dass auch der fortgeschrittene Programmierer bei der Entwicklung komplexer Projekte häufig einen Einstieg über diese Beispielprogramme findet.

Die nachfolgenden ersten eigenen Programme werden auch aus diesen Standardbeispielen abgeleitet werden. Wie Bild 1 zeigt, kann man die Beispielprogramme auch problemlos in der Verzeichnisstruktur wiederfinden.

Fehlerbeseitigung

Bereits im ersten Beitrag wurde gezeigt, wie man einen Sketch auf den Arduino lädt. Im Erfolgsfall wird dies mit einer Meldung (Bild 6) quittiert.

Obwohl die Arduino-IDE sehr benutzerfreundlich ist, können natürlich Probleme nicht immer ausgeschlossen werden. Der häufigste Fehler ist die Auswahl des falschen seriellen Ports. In diesem Fall wird eine Fehlermeldung ausgegeben (Bild 7).

An zweiter Stelle der Fehlerstatistik steht die Auswahl des falschen Boardtyps. In diesem Fall sieht die Fehlermeldung so wie in Bild 8 aus.

Man erkennt, dass die Fehlermeldungen oft nicht sehr spezifisch sind. Allerdings geben sie in den meisten Fällen doch recht brauchbare Hinweise auf die Ursache des Problems.

Neben diesen beiden häufigsten Fehlern treten in der Praxis öfter auch „seltsame“ Fehler ohne definierte Ursache auf. Oft hilft es dann, den zuletzt ausgeführten Vorgang noch mal zu starten.

TIPP II

Falls der Arduino einmal nicht mehr ansprechbar sein sollte, hilft es oft, die USB-Verbindung zu lösen und das Board wieder neu mit dem Computer zu verbinden.

Weitere Tipps und Tricks zur Arbeit mit der IDE sind u. a. in der Literatur zum Arduino-Projekt zu finden [4]. Tieferegehende Informationen zur Programmierung und zum Controller selbst finden sich in [5].

Der Alarmanlagensimulator

Das Arbeiten mit der IDE sollte nun keine Schwierigkeiten mehr bereiten. Es ist jetzt also an der Zeit, erste eigene Sketche zu entwickeln.

Hierfür bietet sich der Aufbau eines Alarmanlagensimulators an. Solche Geräte werden durchaus auch in der Praxis eingesetzt. Sie bestehen letztendlich nur aus einer LED (Bild 9), die in regelmäßigen

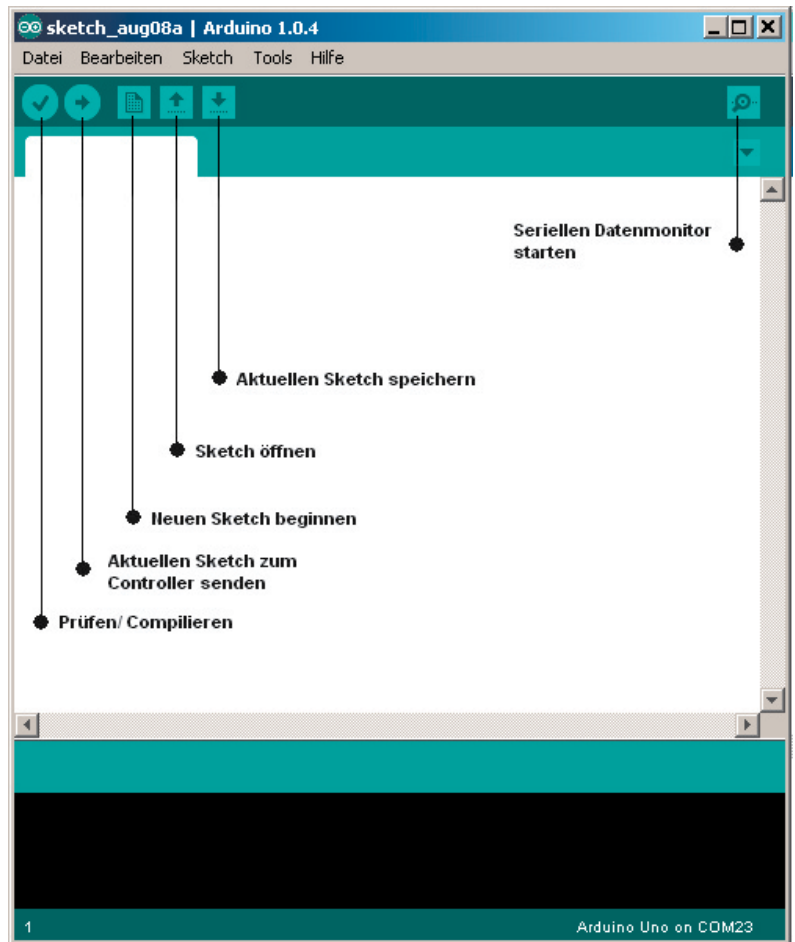


Bild 4: Symbole der IDE

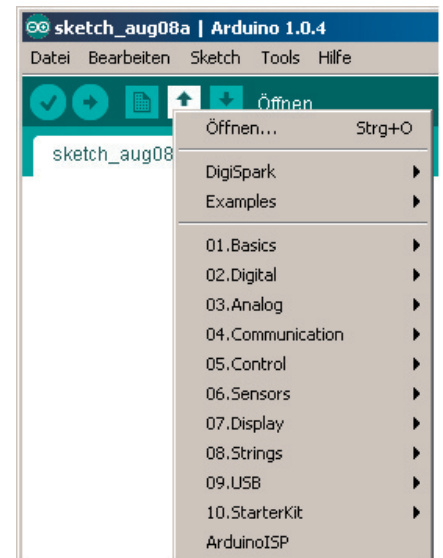


Bild 5: Die Standard-Beispielprogramme

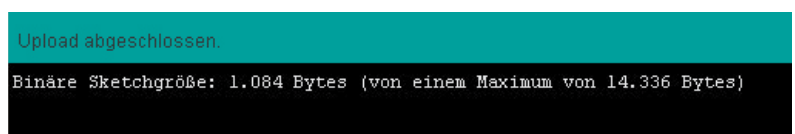


Bild 6: Upload o. k.

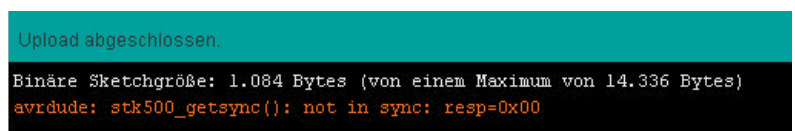


Bild 7: Fehlermeldung bei falschem Port



Es wurde ein falscher Microcontroller gefunden. Haben Sie den richtigen aus dem Menü Tools > Board ausgewählt?

Binäre Sketchgröße: 882 Bytes (von einem Maximum von 7.168 Bytes)
 avrdude: Expected signature for ATMEGA8 is 1E 93 07
 Double check chip, or use -F to override this check.

Bild 8: Fehlermeldung bei falsch gewählter Boardversion



Bild 9: Geeignete LEDs für Mikrocontroller-Schaltungen

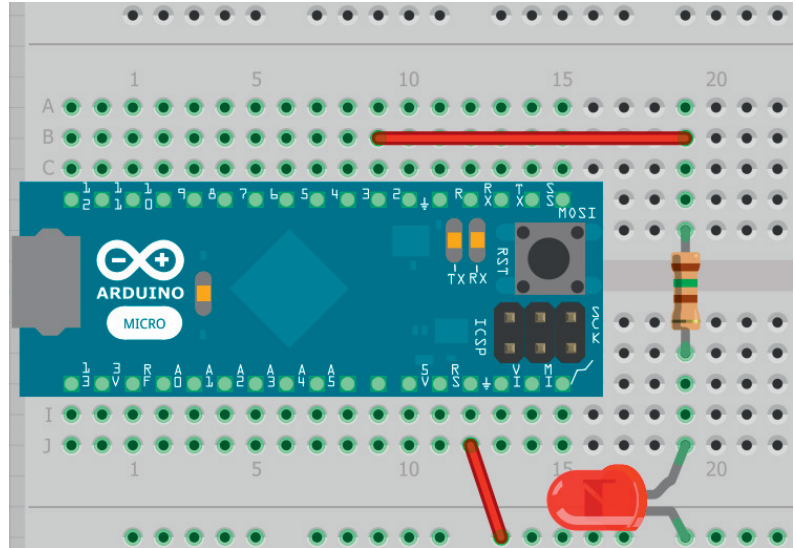


Bild 10: Aufbau zum Alarmanlagensimulator

Abständen kurz aufblitzt. Wird ein solches Gerät an geeigneter Stelle eingebaut, beispielsweise in einem Fahrzeug, so entsteht der Eindruck einer scharf geschalteten Alarmanlage. Zumindest Gelegenheitsdiebe lassen sich damit von einem Einbruch abschrecken.

Neben einem Arduino Micro werden hierfür noch vier weitere Bauelemente benötigt:

1. Rote LED
2. 150-Ω-Widerstand
3. Ca. 5 cm Klingeldraht
4. Ein lötfreies Steckbrett

Der Aufbau der kleinen Schaltung ist in [Bild 10](#) zu sehen.

Das zugehörige Programm sieht so aus:

```
// Alarmanlagensimulator
```

```
int led = 2;
```

```
void setup()
{ pinMode(led, OUTPUT);
}
```

```
void loop() {
  digitalWrite(led, HIGH);
  delay(100);
  digitalWrite(led, LOW);
  delay(3000);
}
```

Man kann das Programm in ein leeres Sketchfenster eintippen oder man modifiziert den bereits bekannten Blink-Sketch. Die Unterschiede zum Blink-Sketch bestehen in den folgenden Änderungen:

1. Die Kommentarzeilen, welche mit „//“ beginnen, wurden angepasst.

2. Der Pin für die LED wurde von 13 auf 2 geändert. Damit ist nun die dort angeschlossene externe LED aktiv.
3. Die Werte in der Funktion „delay“ wurden von jeweils 1000 auf 100 bzw. 3000 geändert. Damit wird aus dem regelmäßigen Blinken ein kurzes Aufblitzen.

Wenn die Schaltung korrekt aufgebaut wurde und der neue Sketch geladen wird, blitzt die angeschlossene LED im Abstand von jeweils 3 Sekunden kurz auf. Der Alarmanlagensimulator ist einsatzbereit ([Bild 11](#)).

Will man die elektronischen Komponenten nicht einzeln besorgen, so kann man beispielsweise auf das Lernpaket „Mikrocontroller in C programmieren“ [6] zurückgreifen. In diesem Lernpaket sind neben einem Arduino auch alle weiteren für den Einstieg in die Mikrocontroller-Welt erforderlichen Bauelemente (Widerstände, Kondensatoren, LEDs etc.) enthalten. Darüber hinaus finden sich in diesem Lernpaket weitere interessante Komponenten wie etwa ein 4-stelliges 7-Segment-Display oder eine LED-Punktmatrix für die mikrocontrollergesteuerte Anzeige von Zahlen und Zeichen ([Bild 12](#)).

Intuitive Programmierung: der Arduino sendet Morsesignale

Hier können die bereits erworbenen Kenntnisse praxisnah eingesetzt werden. Die Aufgabe besteht darin, den Alarmanlagensimulator in eine automatische SOS-Bake umzubauen. Das heißt, die LED soll zunächst 3x kurz, dann 3x lang und schließlich wieder 3x kurz aufblitzen. Dann soll sich die Signalfolge wiederholen.

Versuchen Sie zunächst, den Code für den Alarmanlagensimulator so abzuändern, dass die LED das gewünschte Signal sendet. Anschließend können Sie dann Ihre Lösung mit dem folgenden Musterprogramm vergleichen.

```
// SOS
```

```
int led = 2;
```

```
void setup()
{ pinMode(led, OUTPUT);
```



```

}

void loop()
{ digitalWrite(led, HIGH); delay(100);
  digitalWrite(led, LOW); delay(500);
  digitalWrite(led, HIGH); delay(100);
  digitalWrite(led, LOW); delay(500);
  digitalWrite(led, HIGH); delay(100);
  digitalWrite(led, LOW); delay(1000);

  digitalWrite(led, HIGH); delay(400);
  digitalWrite(led, LOW); delay(500);
  digitalWrite(led, HIGH); delay(400);
  digitalWrite(led, LOW); delay(500);
  digitalWrite(led, HIGH); delay(400);
  digitalWrite(led, LOW); delay(1000);

  digitalWrite(led, HIGH); delay(100);
  digitalWrite(led, LOW); delay(500);
  digitalWrite(led, HIGH); delay(100);
  digitalWrite(led, LOW); delay(500);
  digitalWrite(led, HIGH); delay(100);
  digitalWrite(led, LOW); delay(1000);
  delay(2000);
}

```

Natürlich ist dies nicht die eleganteste Methode, aber man sieht, dass so bereits ohne genaue Kenntnis der verwendeten Programmiersprache eigene Projekte realisierbar sind. In späteren Beiträgen zu dieser Reihe wird natürlich noch genauer darauf eingegangen, wie man ein derartiges Programm wesentlich effizienter implementieren kann.

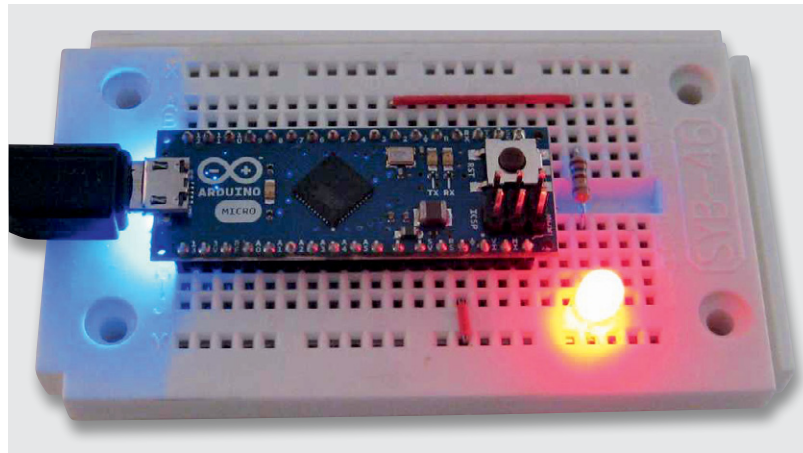


Bild 11: Der Alarmanlagensimulator ist scharf geschaltet.

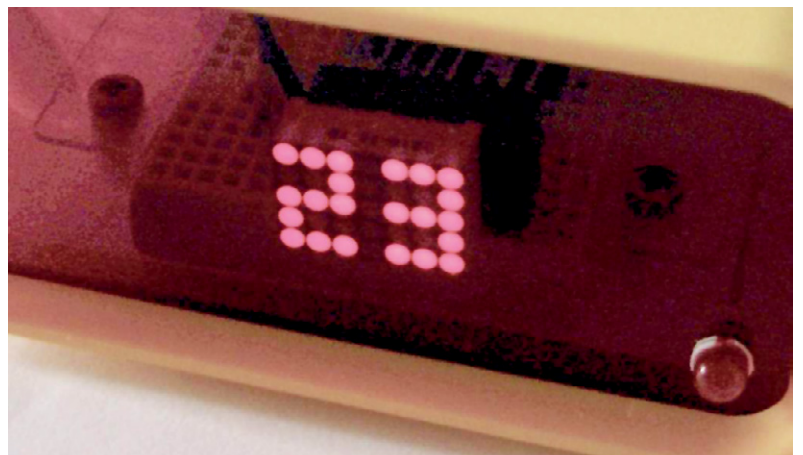


Bild 12: Zahlendarstellung mit dem Dot-Matrix-Display aus dem Lernpaket „AVR-Mikrocontroller in C programmieren“ [6]

Empfohlene Produkte/Bauteile:

	Best.-Nr.	Preis
Arduino Micro	JQ-10 97 74	€ 24,95
Mikrocontroller-Onlinekurs	JQ-10 20 44	€ 99,-
LEDs, z. B. Kemo Leuchtdioden, ca. 30 Stück	JQ-10 66 60	€ 1,65
Widerstände, Kemo-Sortiment, ca. 200 Stück	JQ-10 66 57	€ 1,85

Steckbretter sind in verschiedenen Franzis Lernpaketen enthalten oder im Web-Shop unter www.elv.de/experimentierboards.html erhältlich.


Alle ARDUINO-Produkte wie Mikrocontroller-Platinen, Shields, Fachbücher und Zubehör finden Sie unter: www.arduino.elv.de

Preisstellung November 2013 – aktuelle Preise im Web-Shop.

Dennoch erfüllt das Programm seinen Zweck. Verwendet man als LED einen besonders lichtstarken Typ, so kann die Schaltung beispielsweise in See- oder Bergnot durchaus nützlich sein.

Weitere Anwendungen und Projekte mit dem Arduino wie verschiedene LED-Effekte etc. finden sich beispielsweise in [2] und [6].

Ausblick

Nachdem nun die Arbeit mit der Arduino-IDE problemlos von der Hand geht, wird im nächsten Beitrag zu dieser Reihe die eigentliche Programmiersprache im Vordergrund stehen. Dabei sollen dann neben einigen grundlegenden Betrachtungen zur Programmstruktur insbesondere die Arduino-spezifischen Befehle besprochen werden. 



Weitere Infos:

- [1] <http://arduino.cc> ist die offizielle Homepage des Arduino-Projektes
- [2] Mikrocontroller-Onlinekurs, Franzis-Verlag, exklusiv für ELV, 2011, Best.-Nr.: JQ-10 20 44
- [3] <http://forum.arduino.cc>
- [4] G. Spanner: Arduino – Schaltungsprojekte für Profis, Elektor-Verlag, 2012, Best.-Nr.: JQ-10 94 45
- [5] G. Spanner: AVR-Mikrocontroller in C programmieren, Franzis-Verlag, 2010, Best.-Nr.: JQ-09 73 52
- [6] Lernpaket „AVR-Mikrocontroller in C programmieren“, Franzis-Verlag, 2012, Best.-Nr.: JQ-10 68 46