edition Make:

Arduino

Ein schneller Einstieg in die Microcontroller-Entwicklung

von
Maik Schmidt

2., akt. u. erw. Aufl.

dpunkt.verlag 2015

Verlag C.H. Beck im Internet:
www.beck.de
ISBN 978 3 86490 126 3

Zu Inhaltsverzeichnis

2 Umfangreichere Arduino-Projekte

Was Sie im letzten Kapitel über die Arduino-IDE gelernt haben, ist für einfache Anwendungen ausreichend. Ihre Projekte werden aber schnell größer werden, und dann ist es bequemer, sie in einzelne Dateien aufzuteilen, die Sie als Ganzes verwalten können. In diesem Kapitel lernen Sie daher, wie Sie auch mit größeren Projekten in der Arduino-IDE umgehen.

Normalerweise bestehen größere Projekte nicht nur aus mehr Software, sondern auch aus mehr Hardware, und Sie werden nur selten ein Arduino-Board allein betreiben. Vielmehr werden Sie mehr Sensoren verwenden, als Sie sich haben träumen lassen, und Sie müssen die erfassten Daten zurück auf Ihren Computer übertragen. Um Daten mit Arduino auszutauschen, verwenden Sie die serielle Schnittstelle. In diesem Kapitel erfahren Sie daher alles über die serielle Datenübertragung. Damit das alles anschaulicher wird, zeige ich Ihnen, wie Sie aus Ihrem Computer einen sehr teuren Lichtschalter machen, indem Sie eine LED über Ihre Tastatur steuern.

2.1 Was Sie benötigen

Für die Beispiele in diesem Kapitel benötigen Sie:

1. ein Arduino-Board wie das Uno, Duemilanove oder Diecimila.
2. ein USB-Kabel, um Arduino mit Ihrem Computer zu verbinden
3. eine LED (optional)
4. eine Terminalsoftware wie Putty (für Windows) oder Screen (für Linux und Mac OS X), optional

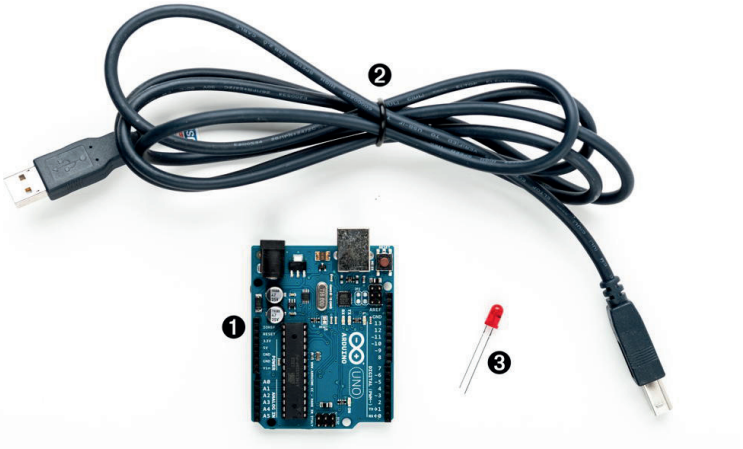

Abb. 2–1 *Alle Teile, die Sie in diesem Kapitel benötigen*

2.2 Projekte und Programme verwalten

Moderne Softwareentwickler können aus einer Reihe von Entwicklungswerkzeugen auswählen, die wiederkehrende und langweilige Aufgaben übernehmen. Das trifft auch auf Arduino zu, und Sie können Ihre Programme mit einer integrierten Entwicklungsumgebung (IDE) verwalten. Die bekannteste wurde vom Arduino-Team selbst entwickelt.

Die Arduino-IDE verwaltet alle zu einem Projekt gehörenden Dateien. Sie bietet einen einfachen Zugriff auf die benötigten Werkzeuge, um Binärdateien zu erzeugen, die auf Ihrem Board ausgeführt werden. Einfach und unaufdringlich.

Alle Dateien eines Projekts automatisch beieinander zu halten, ist eine der wichtigsten Funktionen einer IDE. Intern legt die Arduino-IDE für jedes Projekt ein Verzeichnis an und speichert darin alle zum Projekt gehörenden Dateien. Um dem Projekt neue Dateien hinzuzufügen, klicken Sie auf die *Tabs*-Schaltfläche, um ein Popup-Menü zu öffnen, und wählen dann *New Tab* (siehe Abbildung 2–2). Um eine vorhandene Datei zu öffnen, verwenden Sie *Sketch* > *Add File*.

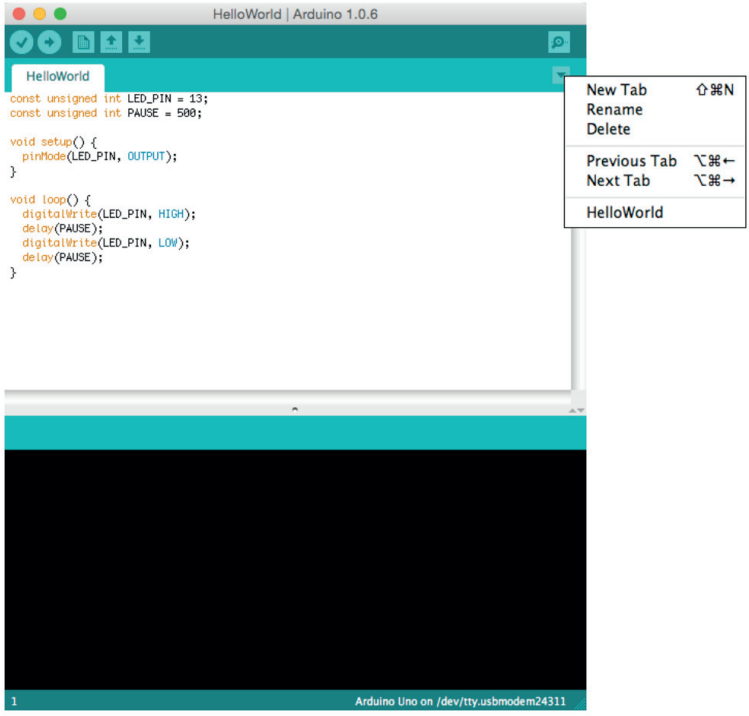

Abb. 2–2 *Das Tabs-Menü*

Wie Sie vielleicht schon anhand der Menüs erraten haben, bezeichnet die Arduino-IDE Projekte als *Sketches*. Wenn Sie einen neuen Sketch erstellen, gibt die IDE ihm einen Namen, der mit *sketch_* beginnt. Sie können den Namen jederzeit mit dem Befehl *Save as (Speichern unter)* ändern. Wenn Sie ein Sketch nicht explizit speichern, legt die IDE es in einem vordefinierten Ordner ab, den Sie im Menü *Preferences* einstellen können. Wenn Sie sich nicht mehr zurechtfinden, können Sie den Ordner des aktuellen Sketch im Menü *Sketch > Show Sketch Folder* nachschlagen.

Seit Arduino 1.0 tragen Sketche die Endung *ino*, wohingegen in älteren IDE-Versionen *pde* verwendet wurde. Arduino 1.0 kann immer noch *pde*-Dateien lesen, wandelt sie beim Speichern des Sketches aber in *ino*-Dateien um. (Dieses Verhalten können Sie im Menü *Preferences* abstellen.)

In der IDE können Sie nicht nur eigene Sketche erstellen. Im Lieferumfang sind auch viele fertige Beispielsketche enthalten, die Sie als Grundlage für Ihre eigenen Experimente verwenden können. Zugriff darauf haben Sie über *File > Examples*. Nehmen Sie sich etwas Zeit, um sie sich anzusehen, auch wenn Ihnen noch nicht klar ist, was Sie damit anfangen können.

Auch viele Bibliotheken werden mit Beispielen geliefert. Wenn Sie eine neue Bibliothek installieren (wie das geht, erfahren Sie weiter hinten), sollten Sie sich

erneut *File* > *Examples* ansehen. Wahrscheinlich sind dort jetzt neue Einträge verzeichnet.

Die Arduino-IDE erleichtert Ihnen die Arbeit durch sinnvolle Vorgaben für viele Einstellungen. Im nächsten Abschnitt sehen Sie, wie Sie die meisten dieser Einstellungen anpassen können.

2.3 Voreinstellungen ändern

Für die ersten Projekte sind die IDE-Voreinstellungen in Ordnung, früher oder später wollen Sie jedoch Anpassungen vornehmen. Wie Sie in Abbildung 2–3 sehen, lassen sich allerdings nur wenige Werte direkt verändern. Das Dialogfenster verweist auf eine Datei namens *preferences.txt* mit weiteren Voreinstellungen. Es handelt sich um eine Datei mit Schlüssel-Wert-Paaren. Einige davon sind:

```
...
preproc.web_colors=true
editor.font.macosx=Monaco,plain,10
update.check=true
build.verbose=true
upload.verbose=true
...
```

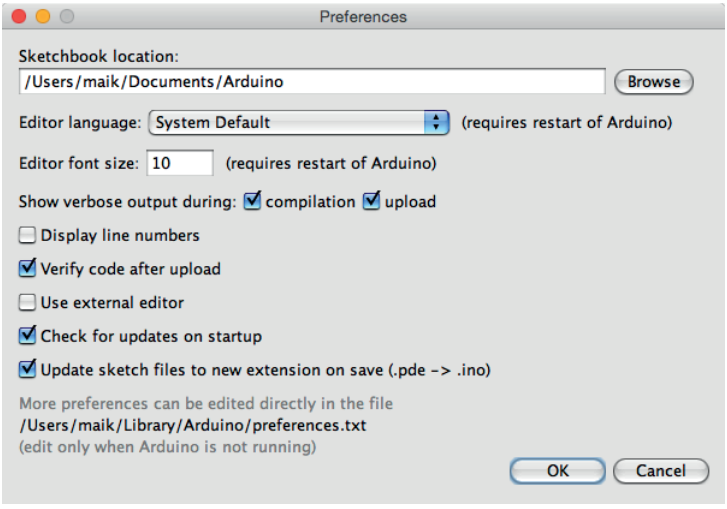

Abb. 2–3 *In der IDE lassen sich einige Voreinstellungen ändern.*

Die meisten dieser Voreinstellungen beeinflussen die Benutzerschnittstelle, also die Schriftart, Farben usw. Sie können aber auch das Verhalten der Anwendung steuern. Sie können umfangreichere Meldungen für Operationen wie Kompilierung oder das Hochladen eines Programms einstellen. Vor Arduino 1.0 mussten Sie dazu *preferences.txt* bearbeiten und *build.verbose* und *upload.verbose* auf *true*

setzen. Heute können Sie das im Voreinstellungsfenster erledigen. Sorgen Sie dafür, dass für Kompilierung und Upload die ausführliche Ausgabe eingerichtet ist.

Laden Sie dann das Programm für die blinkende LED aus Kapitel 1, *Willkommen bei Arduino*, und kompilieren Sie es erneut. Die Ausgabe sollte jetzt wie die in Abbildung 2–4 aussehen.

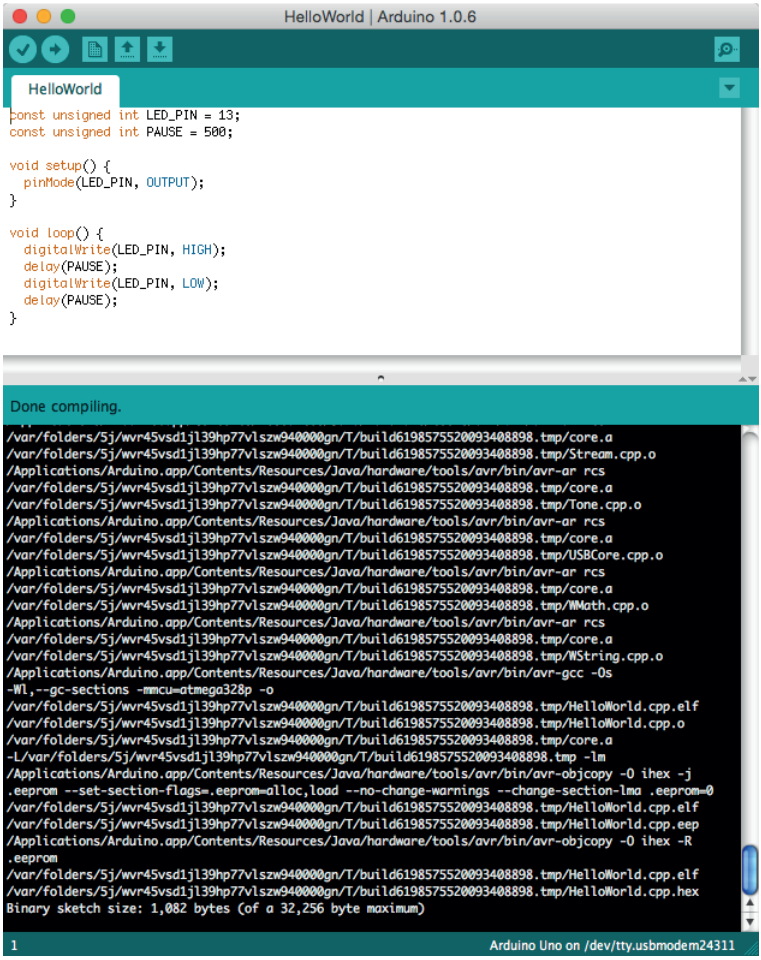

Abb. 2–4 *Ausführliche Meldungen der IDE bei Kommandozeilen-Tools*

Die IDE aktualisiert einige der Voreinstellungen erst, wenn sie beendet wird. Bevor Sie also Änderungen an der Datei *preferences.txt* vornehmen, müssen Sie die Arduino-IDE beenden.

Da Sie jetzt mit der Arduino-IDE vertraut sind, fangen wir mit der Programmierung an. Wir lassen Arduino mit der Außenwelt kommunizieren.

Die Arduino-Programmiersprache

Neulinge sind oft irritiert, wenn es um die Programmiersprache geht, in der ein Arduino programmiert wird. Das liegt daran, dass viele Beispielprogramme so aussehen, als wären sie in einer speziell für Arduino entwickelten Programmiersprache geschrieben. Das stimmt aber nicht ganz, denn Arduinos werden in C++ programmiert, und das bedeutet, dass sie auch in C programmiert werden können.

Die meisten Arduino-Boards basieren auf einem von der Firma Atmel entwickelten AVR-Microcontroller. (Atmel behauptet, der Name AVR stünde für nichts Besonderes.) Diese Microcontroller sind sehr beliebt und werden in vielen Hardware-Projekten verwendet. Einer der Gründe für ihre Beliebtheit ist die umfangreiche Palette an Werkzeugen, die zur Verfügung steht. Sie basieren auf dem GNU C++-Compiler und wurden für die Codeerstellung für AVR-Microcontroller optimiert.

Sie übergeben dem Compiler also C++-Code, der nicht in Maschinencode für Ihren Computer, sondern für den AVR-Microcontroller umgesetzt wird. Dieses Verfahren wird »Cross-Compiling« genannt und ist bei der Programmierung von Embedded-Systems üblich.

2.4 Serielle Schnittstellen einsetzen

Mit Arduino sind Stand-alone-Anwendungen möglich, also Projekte, die ohne einen Computer auskommen. In diesen Fällen wird Arduino nur an den Computer angeschlossen, um die Software hochzuladen. Danach wird nur noch eine (vom Computer unabhängige) Stromversorgung benötigt. Häufig verwenden Anwender Arduino aber, weil sie so die Fähigkeiten des Computers um Sensoren und zusätzliche Hardware erweitern können. Normalerweise steuern sie dann externe Hardware über eine serielle Schnittstelle. Es ist daher sinnvoll, dass wir uns mit der Datenübertragung zu Arduino beschäftigen.

Obwohl sich die Standards für die serielle Datenübertragung in den letzten Jahren verändert haben (heute haben wir USB und keine RS 232-Schnittstellen mehr), ist das Grundprinzip nach wie vor unverändert. Im einfachsten Fall verbinden wir zwei Geräte mit nur drei Leitungen: einer gemeinsamen Masseverbindung, einer Leitung zum Senden (TX) und einer zum Empfangen (RX) von Daten.