FRANZIS

MACH'S EINFACH

MUSIK MIT DEM ARDUINO® UND MIDI

Spielend leicht zum selbstgebauten MIDI-Mischpult

JOHANNES WRONKA
TOBIAS NAUMANN

MACH'S EINFACH

MUSIK MIT DEM ARDUINO® UND MIDI

Spielend leicht zum selbstgebauten MIDI-Mischpult

Die Autoren

Johannes Wronka und Tobias Naumann studierten beide im selben Jahrgang 2008 Medientechnologie an der Technischen Universität in Ilmenau. Ihre Begeisterung für elektronische Projekte mit Mikrocontrollern sowie das gemeinsame Interesse an der Musik und der praktischen Informatik brachte beide zu diesem Buchprojekt. Die Autoren leben in Berlin bzw. Halle und sind in den Bereichen Medienproduktion und Backend-Entwicklung tätig.

FRANZIS

MACH'S EINFACH

MUSIK MIT DEM ARDUINO® UND MIDI

Spielend leicht zum selbstgebauten MIDI-Mischpult

JOHANNES WRONKA
TOBIAS NAUMANN

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Daten sind im Internet über *http://dnb.ddb.de* abrufbar.

Lektorat: Ulrich Dorn
Satz: Nelli Ferderer (nelli@ferderer.de)
Covergestaltung: Julia Harrer
ISBN 978-3-645-20658-7

DANKSAGUNG

Dieses Buch entstand als Erstlingswerk nach diversen Überlegungen, es zu schreiben oder die Zeit lieber anderweitig zu nutzen. Es hat sich für die Autoren als spannende Herausforderung entwickelt und war teilweise mit Kopfzerbrechen und mehreren späten Arbeitsstunden verbunden. Ein großer Dank geht an unsere Familien für die Unterstützung und das Verständnis dafür, doch ein bisschen häufiger keine Zeit für das Familienleben oder generell gesellschaftliches Miteinander zu haben. Dank geht auch an den FRANZIS Verlag für das Angebot, dieses Werk zu schreiben, und an Ulrich Dorn für die Unterstützung – und die zwei Wochen Aufschub.

Ich, Tobias Naumann, danke meiner baldigen Frau Christin, die selbst bei fehlendem Verständnis immer Begeisterung für unsere Projekte vorgaukeln kann und mir nach frustrationsreichen Programmierabenden stets tröstend zur Seite stand. Zudem möchte ich Johannes danken, der es geschafft hat, mich von einem »Ja, ich lese dein Buch dann mal Korrektur« in einen unterschriebenen Autorenvertrag rutschen zu lassen. Er ist für mich der Inbegriff eines Bastlers und steckt mit seiner Begeisterung für solche Themen ausgesprochen schnell an. Mein Dank gilt auch Maik und Andreas, die meine Faszination für Technik und Programmierung weiter geschürt haben. Mit ihnen ist spaßiges Arbeiten möglich, dennoch lassen sie einem die Nerven für das Entwickeln eines MIDI-Controllers und das Schreiben eines Buchs in der Freizeit.

Ich, Johannes Wronka, danke meiner Familie und meiner zauberhaften Claudia für ihre liebevolle Unterstützung besonders in Durststrecken, für leckeres Essen und ihre wunderbare Fürsorge. Großer Dank geht auch an meinen Vater Klaus und Onkel Ben. Beide haben mir auf ihre ganz eigene Art gezeigt, wie kreativ man die Welt sehen kann, und damit maßgeblich meinen beruflichen Werdegang bestimmt. Tobias bekommt besonderen Dank für sein wesentliches Mitwirken in diesem Buch, dessen Entstehung ohne ihn so nicht möglich gewesen wäre. Wenn einer soliden Code schreiben und nachts im Sitzen einschlafen kann, dann er. Ines Zimzinski möchte ich danken für ihre Geduld und ihre Gabe, mir, so gut sie konnte, alle Hörbuch-Stolpersteine aus dem Weg zu räumen. Außerdem möchte ich mich sehr bei Christoph Hahn und Constantin Wiedemann für die von Ihnen geschossenen Technikfotos bedanken sowie Markus Baartz und dem Team von »Just Music« in Berlin für ihre freundliche Unterstützung.

INHALT

2 WAS IST EIGENTLICH MIDI?

MIDI – diese vier Buchstaben haben die Musikwelt in den letzten 30 Jahren maßgeblich mitgeprägt. Es gibt kaum einen Standard, der sich in der Industrie so langjährig gehalten und trotzdem nichts von seiner Aktualität eingebüßt hat. Doch was versteht man eigentlich darunter? Viele, die sich das erste Mal mit der Thematik des Musikmachens und Aufnehmens auseinandersetzen, stolpern früher oder später über diesen Begriff in seinen verschiedenen Formen. Und in der Tat, MIDI ist wandelbar und in einer Vielzahl von Hard- oder Software anzutreffen. Aber eins nach dem anderen.

Der MIDI-Standard stellt sich vor

MIDI ist die Kurzform für *Musical Instrument Digital Interface* und beschreibt einen Standard, um zwischen elektronischen Geräten Steuerdaten zu senden. 1981 wird die Technik von Dave Smith in dem wissenschaftlichen Artikel »Universal Synthesizer Interface« eingeführt und spezifiziert. In Zusammenarbeit mit der Firma Roland sowie anderen Musikgeräteherstellern wird er dann auf der NAMM-Show 1983 in Anaheim, USA, vorgestellt. Dort wird demonstriert, wie ein Prophet-600-Synthesizer von Sequential Circuits (dem Unternehmen, in dem Dave Smith damals arbeitete) mit einem Jupiter 6 von Roland über ein kleines Kabel kommuniziert. Wird das eine Gerät angesteuert, reagiert das andere darauf und beginnt ebenfalls zu spielen.

Die Menschen sind begeistert davon, dass zwei Musikgeräte über eine derartig einfache Steckverbindung quasi via Plug-and-play zusammen funktionieren, denn bisher gilt die Zusammenschaltung von elektronischen Instrumenten als schwierig. Heraus kommt stets ein großer Kabelsalat mit zweifelhafter Funktionalität. Nicht so bei MIDI. Mit seiner einfachen Verbindung bietet sich damit die Möglichkeit, verschiedene elektronische Klangerzeuger zu kombinieren und komplexe Klänge nur durch ein einziges Keyboard zu erschaffen. Und auch wenn es bereits zur Entstehungszeit von MIDI technisch bessere Ansätze gibt, so ist es am Ende doch die kompromisslose Einfachheit, die MIDI den Siegeszug und seine Daseinsberechtigung bescheren wird. 1983 wird der MIDI-Standard von der MMA (*MIDI Manufacturers Association*) festgelegt. Die Einhaltung des Standards kontrolliert bis heute die IMA (*Internal MIDI Association*).

MIDI ist in kurzer Zusammenfassung ein Datenprotokoll, das in Paketen von jeweils drei Bytes Parameter zur Kontrolle eines Geräts sendet. Diese Parameter können entweder bestimmte Noten mit einer Anschlagstärke sein oder auch klangverändernde Elemente wie z. B. ein Rad zur Variation der Tonhöhe – ein sogenanntes Pitch Wheel. Wichtig zu verstehen ist dabei, dass man mit MIDI selbst keine Töne oder Musiksignale überträgt, sondern eben reine Steuersignale, die ein anderes Gerät interpretieren und dann mit seinen eigenen klangerzeugenden Fähigkeiten wiedergeben kann. MIDI ist also eher eine Art Notationssystem, und genauso arbeitet man damit auch. Es kann in Geräten wie Samplern, Synthesizern oder Expandern genutzt werden, aber auch eine Verbindung mit oder zwischen Computern ist zumeist problemlos möglich.

Dass MIDI bereits frühzeitig auch in Computern zur Anwendung kam, ist besonders dem Commodore-64 und dem Atari ST zu verdanken. Der Commodore-64 verfügte über einen zur damaligen Zeit überragenden Soundchip. Der Atari ST wiederum beinhaltete bereits ab Werk MIDI-Buchsen. Beide Geräte wurden dadurch für die computergestützte Musikproduktion essenziell, und eine Reihe von heute bekannten Musikprogrammen (wie z. B. Cubase) wurde auf den damaligen Computern entwickelt.

MIDI wird auch in anderen Anwendungsgebieten genutzt. So verwendet man für die Steuerung von Lichttechnik ebenfalls MIDI-Befehle. Außerdem kann ein sogenannter *MIDI-Timecode* (oder besser MTC) gesendet werden, um miteinander verbundene Hard- und Software durch einen Zeitstempel synchron zu halten. Es soll Musiker und Studios geben, die bis heute ihren Atari ST als MTC-Clock zur Synchronisation verwenden.

Bild 2.1: Der KORG Volca Beats: ein kleiner Step Sequencer mit MIDI-Steuerfunktion.

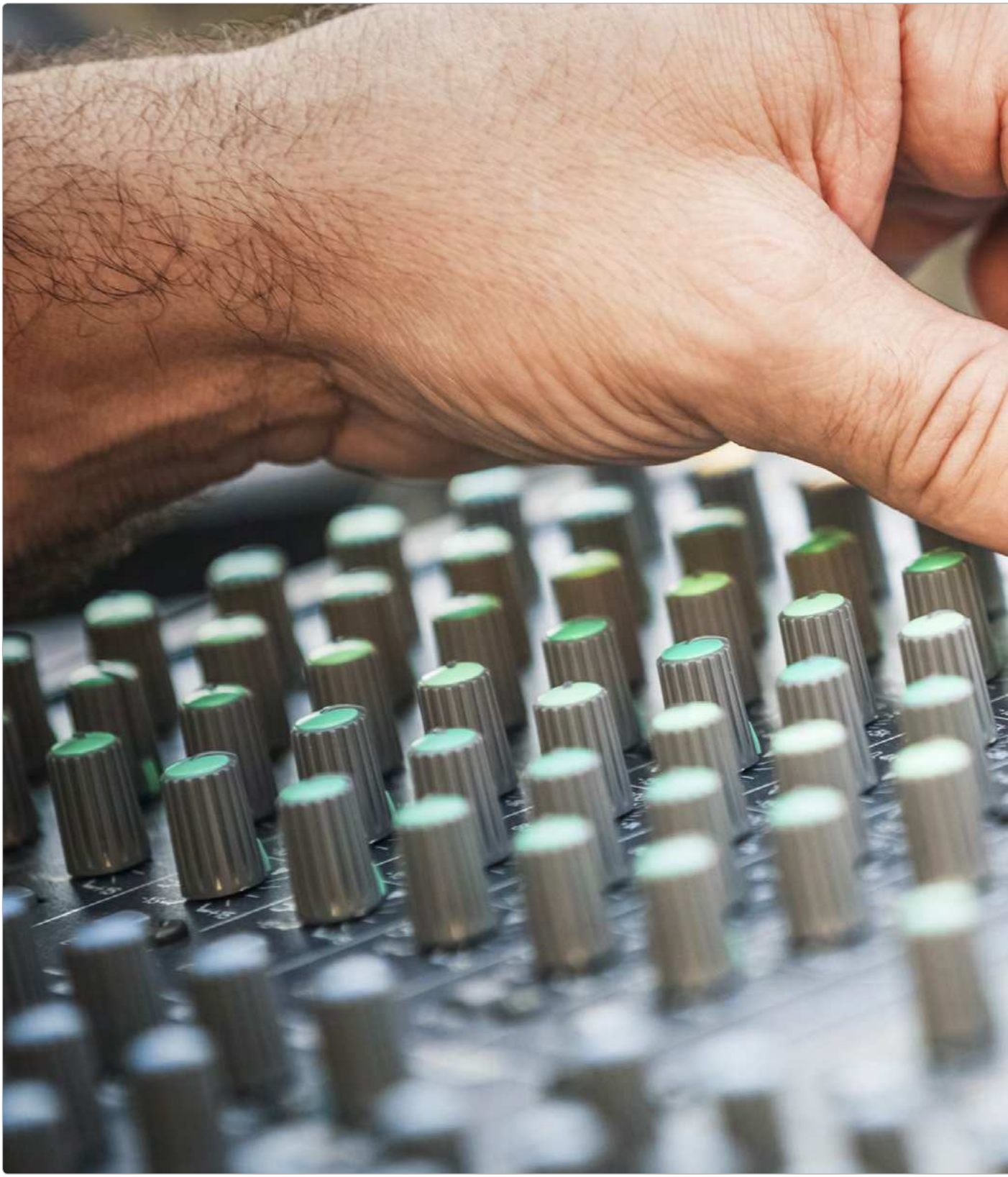
MIDI – technische Aspekte

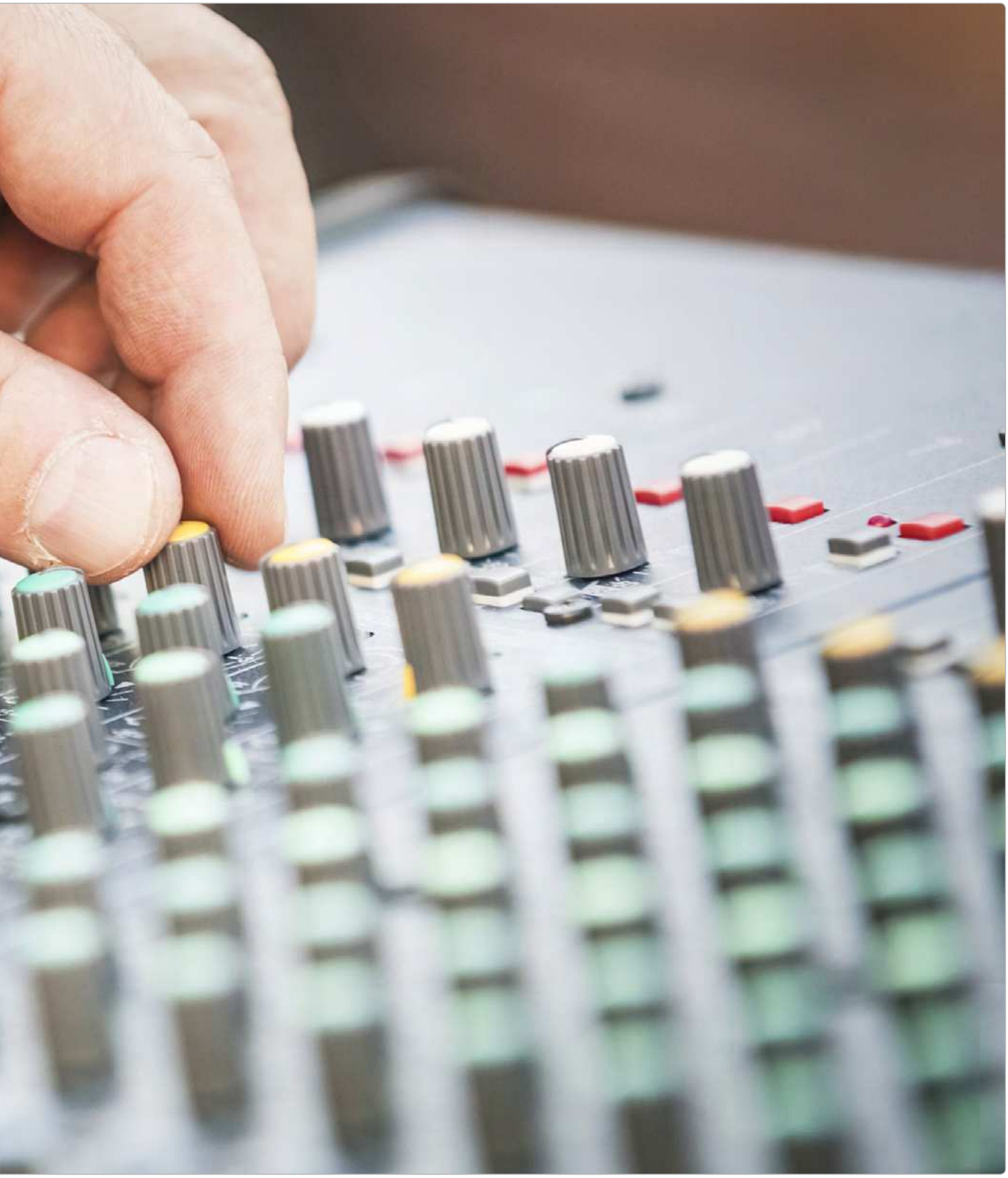
MIDI ist ein Protokoll zur seriellen Übertragung (also ein Befehl nach dem anderen) von Datenpaketen mit jeweils drei Bytes. Diese Übertragung findet unidirektional bei einer Geschwindigkeit von 31.250 Zeichen pro Sekunde statt. Unidirektional meint, dass eine Leitung nur für die Übertragung in eine Richtung genutzt werden kann, während für die Übertragung in die andere Richtung eine zweite Leitung notwendig ist.

Das erste Byte ist dabei das sogenannte Statusbyte. Dessen erstes Bit, das auch *Most Significant Bit* (MSB) genannt wird, beginnt immer mit einer 1. Das Statusbyte enthält den anzusteuernden Parameter sowie den Kanal, auf dem die Übertragung stattfindet. Insgesamt können 16 Kanäle genutzt werden. Darauf folgen zwei Datenbytes, die zusätzliche Informationen zum Statusbyte beinhalten und immer mit einer 0 beginnen.


TUNE
OSCILLATOR MODULATION
ON
GLIDE
MODULATION MIX
OSC. 3
NOISE
CONTROLLERS
RANGE
OSCILLATOR –1 FREQUENCY
WAVEFORM
OSCILLATOR –2
OSCILLATOR –3
OSC. 3 CONTROL
LO
OSCILLATOR BANK
VOLUME
ON
MIXE


EXTERNAL
NPUT VOLUME
OVERLOAD
OISE VOLUME
WHITE
PINK
ER
FILTER
MODULATION
ON
ON
KEYBOARD
CONTROL
ON
CUTOFF FREQUENCY
FILTER
EMPHASIS
AMOUNT
OF CONTOUR
ATTACK TIME
DECAY TIME
SUSTAIN LEVEL
M-SEC.
SEC.
LOUDNESS CONTOUR
ATTACK TIME
DECAY TIME
SUSTAIN LEVEL
MODIFIERS
VOLUME
MAIN OUTPUT
ON
A-440
ON
VOLUME
PHONES
OUTPUT
POWE
ON





Durch das Festlegen des MSB beider Bytetypen von 1 und 0 wird außerdem gewährleistet, dass ein abreißender Datenstrom wieder aufgenommen werden kann. Ein Beispiel verdeutlicht die MIDI-Übertagung und wird uns auch gleich einen ersten Einblick in unsere spätere Arbeit geben. Ein MIDI-Datenpaket könnte wie folgt aussehen:

1. Statusbyte: Spiele eine Note auf Kanal 1 an!
2. Datenbyte 1: Note
3. Datenbyte 2: Anschlagstärke der Note

Das Statusbyte beginnt mit Werten ab `128` bis maximal `255`. Binär ausgedrückt, sind das die Werte von `10000000` (`128`) bis `11111111` (`255`). Das MSB ist immer eine `1`. Die ersten vier Bits im Statusbyte legen fest, welcher Parameter angesteuert wird:

- `1000: Note Off`
- `1001: Note On`
- `1010: Aftertouch`
- `1011: Control Change`
- `1100: Program Change`
- `1101: Channel Pressure`
- `1110: Pitch Wheel`
- `1111: SysEx`

`Note Off` und `Note On` legen fest, ob ein Tastenanschlag zum Spielen einer Note erfolgt oder die Taste losgelassen wird. Dementsprechend wird bei `Note On` eine Note so lange gespielt, bis der `Note Off`-Befehl diesen Vorgang wieder stoppt. `Aftertouch` regelt die Funktionalität für die Druckstärke, die auf eine einzelne Taste ausgeübt wird. Mit diesem Parameter können zusätzliche Effekte definiert werden, etwa ein Tremolo-Effekt – ein Klangeffekt, erzeugt durch schnelle Signalunterbrechungen. Wenn man also eine Taste auf einem MIDI-Keyboard leicht drückt, kann dies bewirken, dass der zu spielende Ton einen leicht hörbaren Tremolo-Effekt beinhaltet. Wird die Taste hingegen mit viel Druck betätigt, wird der Effekt sehr deutlich zu hören sein. Wichtig zu wissen ist, dass der `Aftertouch`-Parameter innerhalb eines MIDI-Kanals für jede Taste einzeln gilt.

Der `Channel Pressure`-Parameter hingegen regelt den Parameter zur Tastendruckstärke für einen ganzen MIDI-Kanal. Werden dort also z. B. mehrere Tasten unterschiedlich stark betätigt, wird ein gemittelter Tastendruckwert aller Tasten übertragen. `Control Change`, auch »Controller« genannt, beinhaltet Informationen zur Klangveränderung mit verschiedenen Controllerparametern, z. B. durch Regler oder Fußpedale. Mittels `Program Change` kann der Klang eines MIDI-Geräts geändert werden. Zwischen 128 verfügbaren Klängen kann also gewechselt werden. Mit dem `Pitch Wheel`-Parameter wird der Wert zur Änderung der Tonhöhe verändert.

Entsprechende Regler (oft in Form eines Rads) befinden sich an einer Vielzahl von MIDI-Instrumenten. Werden sie in die obere Richtung bewegt, erhöhen sie die Tonhöhe der gespielten Note. Eine Bewegung nach unten verringert die Tonhöhe entsprechend. `SysEx`-Befehle dienen der Übertragung nicht musikalischer Nachrichten und können große Datenmengen enthalten. Sie werden oftmals genutzt, um herstellerspezifische Informationen zwischen MIDI-Geräten auszutauschen.

Die letzten vier Bit des Statusbytes legen fest, auf welchem MIDI-Kanal die entsprechenden Daten übertragen werden sollen. Dabei stehen wie bereits beschrieben 16 Kanäle zur Verfügung – ausgehend von Kanal 1 (`0000`) bis Kanal 16 (`1111`). Bezogen auf das im Beispiel oben genannte Statusbyte »Spiele eine Note auf Kanal 1 an!«, würde sich also folgender Wert ergeben: `10010000` (`144`).

Wegen der 0 als MSB reichen die Datenbytes von `00000000` (`0`) bis `01111111` (`127`). Wie bereits beschrieben, spezifizieren sie den Parameter des Statusbytes mit jeweils zwei Bytes. In unserem Beispiel würden sie einmal die zu übertragende Note definieren und dann die Anschlagstärke der Note selbst. Für das tiefste C müsste der Wert `00000000` (`0`) übertragen werden. Wird die Note mit maximaler Anschlagstärke übertragen, ist zusätzlich den Wert `01111111` (`127`) erforderlich. Es würde sich also folgender kompletter MIDI-Befehl für unser Beispiel ergeben:

1. Statusbyte: Spiele eine Note auf Kanal 1 an! : 10010000 = 144
2. Datenbyte 1: Note: tiefstes C : 00000000 = 0
3. Datenbyte 2: maximale Anschlagstärke der Note : 01111111 = 127

MIDI-Daten können einfach in einer Standard-MIDI-Datei gespeichert werden. Das Format gewährleistet die einheitliche MIDI-Verarbeitung auf unterschiedlichsten Geräten und Computerplattformen. Mit dem *General-MIDI-Standard* (kurz GM), eingeführt 1991 durch die MMA, kann man die gespeicherten MIDI-Daten an verschiedenen Geräten abspielen. Der Standard ist mindestens 24-stimmig und ordnet 128 Instrumente bestimmten Klangnummern zu, sodass beispielsweise ein Klavierpart auch von einem Klavierklang wiedergegeben wird. Damit ist gewährleistet, dass abgespeicherte Stücke immer von den korrekten Klängen bzw. Instrumenten abgespielt werden.

Außerdem beinhaltet der Standard ein Drum-Kit, das aus 30 Einzelklängen besteht. Neben GM entwickelten sich weitere neue Standards wie GS der Firma Roland oder XG Mitte der 1990er-Jahre von Yamaha. Sie besitzen ein größeres Spektrum an Klangmöglichkeiten, sind aber trotzdem zu GS abwärtskompatibel. Der XG-Standard beispielsweise bietet in der Level-3-Variante sogar 1.149 Instrumente, 35 Drum-Kits (die jeweils 72 Klänge pro Kit beinhalten) und zwei Effekt-Kits.

MIDI – Anschlüsse und Verkabelung

MIDI-Geräte besitzen klassisch drei Anschlüsse in Form von fünfpoligen DIN-Buchsen:

MIDI-IN	Anschluss für ankommende MIDI-Daten.
MIDI-OUT	Anschluss für ausgehende MIDI-Daten.
MIDI-THROUGH	Anschluss zum direkten Durchschleifen von ankommenden MIDI-Daten an ein anderes MIDI-Gerät.

Besonders computerunabhängige Geräte wie Synthesizer besitzen diese Buchsen heute noch, jedoch verzichten Hersteller gern mal auf den MIDI-THROUGH-Anschluss. Frühe MIDI-Controller besaßen ebenfalls fünfpolige DIN-Buchsen. Durch die Verbreitung von USB wurde dieser Anschluss jedoch zunehmend verdrängt. Von Vorteil ist, dass man über USB den Hin- und Rückkanal in einem gemeinsamen Anschluss vereinen kann, der zudem noch um einiges kleiner ist. MIDI kann außerdem über kabelgebundene oder kabellose Netzwerkverbindungen zwischen verschiedenen Geräten übertragen werden. Ein Smartphone kann über eine WLAN-Verbindung ein ebenso interessanter MIDI-Controller sein wie ein USB-betriebenes MIDI-Keyboard.

MIDI – Instrumente und Controller

Dass man Klangerzeuger mit MIDI verbinden und fernsteuern kann, wissen wir bereits. Mit dem Aufkommen immer stärkerer und vor allem erschwinglicher Rechenleistung wurde MIDI zunehmend auch für die komplexe Fernsteuerung von Computern interessant. Darüber hinaus ergab sich durch das Aufkommen virtueller Instrumente die Möglichkeit, verschiedene Klangerzeuger am Computer zu simulieren. Mit einem entsprechenden Controller ist es möglich, auch das Spielgefühl des jeweiligen virtuellen Instruments nachzuahmen.

Bild 2.2: Ein MIDI-Instrument mit zusätzlichen Controllerelementen der Firma AKAI eignet sich perfekt zur Steuerung virtueller Instrumente und Musikprogramme.

Verschiedenste Klangerzeuger nachbilden

Heute können reine MIDI-Tastaturen in Verbindung mit einem Computer verschiedenste Klangerzeuger nachbilden. Sie beinhalten eine Fülle von Tasten-, Saiten-, Schlag- und Blasinstrumenten. Mit einem MIDI-Keyboard kann man beispielsweise Klavier spielen. Aber man kann damit auch ausprobieren, wie gut sich Gitarrenklänge über Tasten einspielen lassen. Wir sprechen hierbei von sogenannten MIDI-Instrumenten als Teil der MIDI-Controller. Diese Geräte können nicht als selbstständige Klangerzeuger genutzt werden, aber sie können Funktionen am Computer steuern, die sonst mittels Maus oder Tastatur bedient werden müssten.

Man kann noch einen Schritt weiter gehen und ganze Musikprogramme am Rechner mit MIDI-Controllern steuern. Parameter wie die Lautstärkeregelung oder die Einstellung der Aufnahme an einzelnen Kanälen können so über haptische Eingabegeräte kontrolliert werden. Oftmals sehen diese Geräte äußerlich wie Mischpulte aus, nur dass sie eben keinen Ton verarbeiten, sondern die Parameter im Computer.

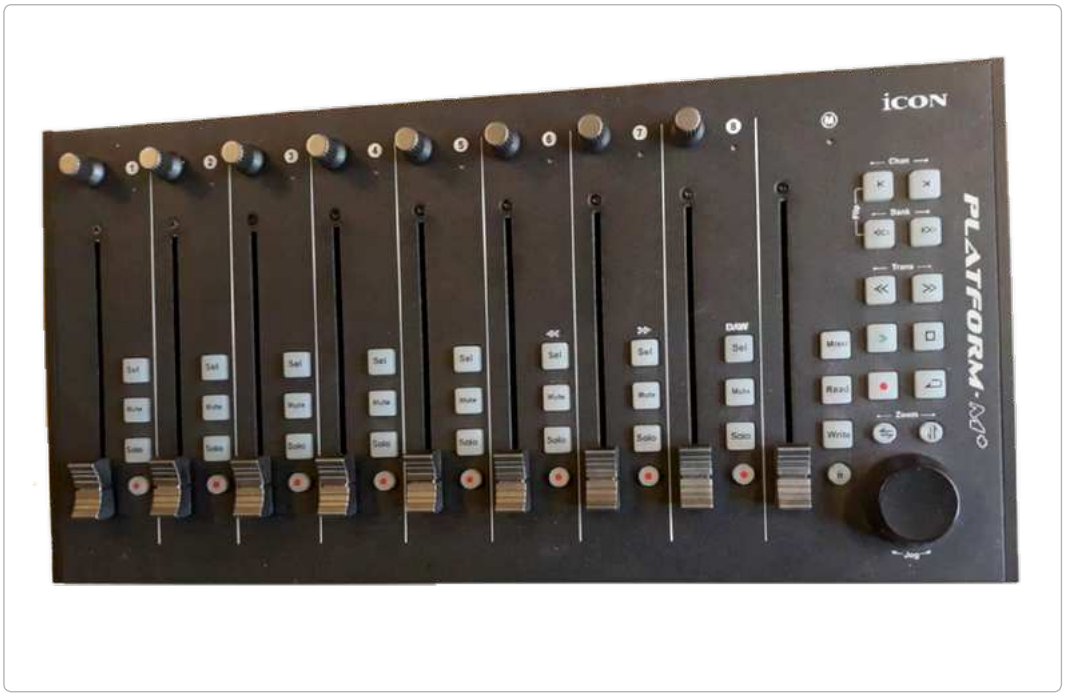


Bild 2.3: Der Icon Platform M+ ist ein einfacher MIDI-Controller zur Steuerung virtueller Mischpulte. (Vielen Dank an Constantin Wiedemann für die Bereitstellung des Fotos.)

Ein kurzer Ausblick auf MIDI 2.0

Im Januar 2019 wurde mit Version 2.0 erstmals seit ca. 30 Jahren ein offizieller Nachfolger für das MIDI-Protokoll vorgestellt. Das Thema ist derzeit (also zum Erscheinungstermin dieses Buchs) brandaktuell und noch in der Entwicklung. Veröffentlichungen zu dem Standard selbst wird es wahrscheinlich erst in den kommenden Monaten geben, genauso wie neue Geräte. Vorangetrieben wird MIDI 2.0 von der MMA sowie der japanischen *Association of Music Electronics Industry* (kurz AMEI).

Die Neuerungen im Protokoll umfassen hauptsächlich eine feinere Auflösung und Genauigkeit für bereits bestehende Funktionen. Werte wie »Pitch Wheel« und die Anschlagdynamik sollen künftig mit 16 Bit (also 65.536 Abstufungen) übertragen werden können. Aktuell unterstützt MIDI 1.0 das nur mit sieben Bit, also vergleichsweise mageren 128 Abstufungen. Controllerwerte erhalten zukünftig gar 32 Bit und damit über vier Milliarden Abstufungen. Im Vordergrund der Entwicklung steht, eine maximale Kompatibilität zu bereits bestehenden Geräten zu ermöglichen. Alte Geräte können demnach theoretisch sehr viel feiner auflösen als bisher – mit den vorhandenen 128 Abstufungen ein großer Vorteil, wenn man z. B. virtuelle Fader oder Filterkurven von Synthesizern steuern will, denn das geht bisher nur recht grob.

Damit diese Unmenge von Daten bewältigt werden kann, braucht es jedoch auch Neuerungen in der Übertragung selbst. MIDI 2.0 soll deshalb mit höherer Übertragungsrate arbeiten, und Daten werden künftig bidirektional gesendet. Auch wenn der neue Standard zum alten maximal kompatibel sein soll, werden natürlich nur aktuelle Geräte von allen Vorteilen profitieren können. Wann wir damit rechnen können, bleibt abzuwarten. Es lässt sich jedoch sagen, dass Verbesserungen im Standard selbst nach über 30 Jahren vielen Anwendungsgebieten sehr zugute kommen werden und sogar zwingend notwendig sind. Lassen wir uns überraschen, wie es mit MIDI 2.0 in den kommenden Jahren weitergehen wird.

3 ARDUINOS – KLEINE MIKRO-CONTROLLER-WUNDER

Viele Menschen haben heutzutage einen PC oder Laptop zu Hause oder am Arbeitsplatz stehen. Wer einmal einen Blick in einen Computer geworfen hat, der weiß, wie viel verbaute Hardware für das Funktionieren dieser Systeme verantwortlich ist. Vermutlich besitzen viele Leser auch ein Tablet oder ein Smartphone und damit ein zwar kleineres, aber dennoch sehr leistungsfähiges Rechensystem. Doch lassen wir den Blick noch ein Stück weiter schweifen. Heutzutage nutzen wir, ohne viel darüber nachzudenken, im Alltag eine Vielzahl von Geräten, die durch mal mehr und mal weniger Rechenpower in der Lage sind, logische Operationen durchzuführen, Signale zu verarbeiten und Geräteteile zu steuern.

So lassen sich bei neueren Digitalkameras Fotos über Wi-Fi direkt am Smartphone betrachten, über programmierte Logik in Reaktion auf die Wetterlage Heizungsanlagen steuern oder ganz ohne weiteres Zutun die wöchentlichen Folgen der Lieblingsserie auf der TV-Box automatisch aufzeichnen. Es stellt sich im Wesentlichen die Frage, wie all diese Dinge mit im Vergleich zum PC vereinfachter Systemarchitektur funktionieren können oder – um den Kreis zu schließen: Wie kann so ein MIDI-Controller eigentlich funktionieren?

Bei den genannten Beispielen bewegen wir uns im Bereich der »embedded«, also der eingebetteten Systeme, in denen reduzierte Recheneinheiten in einem Gerät oder Bauteil integriert sind und darin beispielsweise Steuerungslogik oder spezifische Teilaufgaben eines komplexeren Systems übernehmen. Hier kommen ganz andere Anforderungen an die verbaute Hardware zum Tragen als in einem PC, ohne dass man auf Funktionalität verzichten muss. In kleinen Geräten eingebaut, muss sie möglichst platzsparend sein. Dank dauerhaften Betriebs sollte sie energiesparend und ausdauernd sein, und trotz alledem muss eine hohe Funktionalität im Austausch mit weiterer Hardware gewährleistet sein. Es muss Peripherie geben, um Sensoren auslesen, Geräteteile steuern oder mit anderer Hardware kommunizieren zu können. Mikrocontroller werden diesen Anforderungen in den meisten Fällen gerecht. Die kleinen Minirechner beschränken sich dabei auf das Wesentliche und sind inzwischen gar nicht mal so schwierig zu handhaben. In den letzten Jahren bzw. bereits Jahrzehnten hat sich rund um die Mikrocontroller viel getan, sodass sich z. B. in der Hobby- und Bastlerszene umfangreiche Projekte umsetzen lassen, ohne dass die Motivation an zu hohen Einstiegshürden wie Kosten, Einarbeitungsaufwand und hoher Komplexität in Hard- und Software verloren geht. Dieser Umstand kommt uns zugute, und aus dieser Motivation heraus werfen wir einen genaueren Blick auf die grundlegenden technischen Hintergründe der Mikrocontroller.

Was ist ein Mikrocontroller?

Ein Mikrocontroller ist ein *integrierter Schaltkreis* (IC), der die wichtigsten Dinge, die das Innenleben eines PCs ausmachen, auf einem Chip vereint. Daher werden Mikrocontroller oftmals auch Ein-Chip-Computer genannt.

Was ist damit im Einzelnen gemeint? Zum einen das rechnende Herz des Mikrocontrollers: der Prozessor. Den kennt jeder aus den technischen Daten seines PCs oder Smartphones. Anzahl und Dichte von verbauten Transistoren und damit die Prozessorleistung übertreffen sich mit

jeder neuen Generation am Markt, und die Leistung ist beim direkten Vergleich im Werbeprospekt oftmals ausschlaggebend für die Leistungsfähigkeit eines Computers. Die deutlichsten Unterscheidungen zeigen sich in der Prozessorarchitektur, die wiederum maßgeblich Einfluss auf die Möglichkeiten und Umstände der Programmierung hat.

Während in neuen PCs heutzutage vorwiegend 64-Bit-Prozessoren zum Einsatz kommen, werden Mikrocontroller hauptsächlich in 8- oder 16-Bit-, vereinzelt auch in 32-Bit-Architektur verbaut. Vereinfacht gesagt, verrät uns die Bitzahl die Größe der Dateneinheiten, die in einem Rechenschritt verarbeitet werden können. Theoretisch sind mit einer höheren Bitarchitektur schnellere Berechnungen möglich. Dieser Vorteil wird aber durch den ebenso steigenden Stromverbrauch erkauft, weshalb sparsamere 8- bzw. 16-Bit-Mikrocontroller durchaus noch ihre Daseinsberechtigung haben und in den meisten Embedded-Anwendungsgebieten ausreichend sind.

Architektur von Mikrocontroller-Chips

Wäre das bereits alles, dann wäre der Mikrocontroller nur ein Mikroprozessor. Tatsächlich beinhaltet der Mikrocontroller-Chip weitere Vorteile, die ihn von normalen Prozessoren abhebt. Ein wichtiger Vorteil ist die Peripherie. Damit sind alle Möglichkeiten des Chips gemeint, mit der Umwelt zu interagieren. So besitzt ein Mikrocontroller diverse I/O-Anschlüsse. I/O steht für *Input* und *Output*. Deren Funktion ist größtenteils frei programmierbar und kann z. B. für das Auslesen von Sensoren oder die Steuerung von Motoren oder LEDs genutzt werden.

Dazu integrieren Mikrocontroller je nach Controllerfamilie und Typ verschiedene Schnittstellen zur Kommunikation und zur Interaktion mit anderen Bauteilen wie UART/USART, SPI, I^2C und weiteren. Auf diverse Schnittstellen werden wir im weiteren Verlauf des Buchs einen genaueren Blick werfen.

Den Takt, also die Geschwindigkeit, gibt im Mikrocontroller ein Oszillator vor. Hierbei handelt es sich um ein einzelnes Bauteil oder eine Bauteilgruppe im Schaltkreis zur Erzeugung von Spannungsschwingungen. Sie geben die Wechsel von Spannungshoch und Spannungstief vor und legen per Wiederholungen pro Sekunde (also ihrer Frequenz) die Anzahl von Rechenoperationen in einem Zeitraum fest – je höher die Frequenz, desto schneller können Operationen ausgeführt werden. In Mikrocontrollern finden sich häufig RC-Oszillatoren. Das sind Baugruppen, die hauptsächlich aus Widerständen (R) und Kondensatoren (C) bestehen. Sie sind relativ ungenau, was die Gleichförmigkeit der Schwingungen über eine gewisse Zeit angeht.

Allerdings lassen sie sich aufgrund ihrer Größe einfach in Chips integrieren. Mikrocontroller unterstützen darüber hinaus in der Regel die Taktvorgabe durch externe Oszillatoren, wodurch sich abhängig von den eingesetzten Bauteilen sowohl die Frequenz als auch die Genauigkeit der Schwingungen im Vergleich zu den vorhandenen internen Oszillatoren deutlich steigern lassen. So können sie beispielsweise auch mit dem Takt eines Quarzoszillators arbeiten. Das sind Bauteile mit einem Schwingquarz, die eine Frequenz von sehr hoher Genauigkeit erzeugen.

Des Weiteren beinhalten Mikrocontroller Speicher. Das ist zum einen ein persistenter Speicher, der häufig frei programmierbar ist und selbst nach dem Ausschalten der Spannungsversorgung erhalten bleibt. Darauf befindet sich der Programmcode, der die Logik, die Aufgaben und die Reihenfolge von Operationen beinhaltet. Zum anderen ist auch ein flüchtiger Speicher vorhanden, in dem während des Programmablaufs temporäre Daten gesichert und abgerufen werden können. Wie der Name schon sagt, gehen diese Daten mit dem Entfernen der Spannungsversorgung verloren, sie sind flüchtig. Ohne besonders in die Tiefe gehen zu wollen, ist es interessant, zu wissen, dass es verschiedene Arten von Speichern gibt.

Beispielhaft seien hier ROM (*Read Only Memory*), EPROM (*Erasable Programmable Read Only Memory*), EEPROM (*Electrically Erasable Programmable Read Only Memory*) und Flash-EEPROM (kurz Flash) genannt. Diese Aufzählung stellt nur einen Ausschnitt der vorhandenen Datenspeicher dar. Hauptsächlich unterscheiden sich die Arten in ihrer Schreib- und Lesegeschwindigkeit, in der Persistenz – das ist der Erhalt der Daten auch ohne Spannung – sowie in der Lebensdauer und in der Art bzw. Möglichkeit, den Speicher zu löschen und erneut zu beschreiben. So ist es anders als zur Anfangszeit der Evolution der Mikrocontroller heute möglich, mittels EEPROM oder Flash-Programmspeicher Mikrocontroller selbst und mehrmalig zu beschreiben, was einen herstellerunabhängigen Einsatz beispielsweise im Prototyping oder im Hobby- und Bastelbereich ermöglicht. Dagegen wurden am Anfang hauptsächlich ROM-Speicher eingesetzt, die im Herstellungsprozess nur einmalig beschrieben werden konnten.

Mikrocontroller, oftmals auch unter der Kurzbezeichnung »µC« zu finden, existieren wie viele andere IC-Chips in verschiedenen Bauformen. Es wird dabei zwischen THT (*Through Hole Technology*) und SMD (*Surface Mounted Device*) unterschieden. Neben SMD wird uns der Begriff DIP (*Dual In-Line Package*) als Vertreter der THT-Bauweise am häufigsten begegnen, bei dem Pinanschlüsse an zwei Seiten entlang eines länglichen Gehäuses verlaufen. Die Begriffe verraten bereits, was die Bauformen ausmacht.

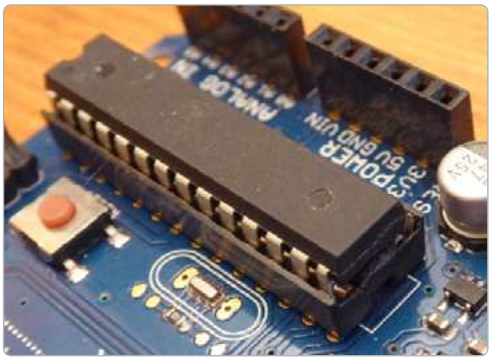


Bild 3.1: Ein steckbarer Mikrocontroller in DIP-Bauform.

Die nach unten abgehenden Anschlüsse von THT-Chips lassen sich durch Leiterplatten hindurchstecken, wo sie auf der Unter- bzw. der Rückseite fest verlötet werden können. Spezielle Gegenstücke auf einem Bauteil erlauben zusätzlich das Stecken und wieder Abnehmen des Mikrocontrollers, wodurch ein unkomplizierter Austausch des Controllers stattfinden kann, etwa wenn das Bauteil einen Defekt aufweist. Außerdem erleichtert es die Programmierung über externe Programmiergeräte.

Dennoch handelt es sich hierbei um eine vergleichsweise grobe Bauform. Deutlich kompakter kommen Mikrocontroller in SMD-Bauform daher. Sie sind oben auf der Leiterplatte verlötet. Entsprechende Bauteile sind deutlich kleiner und erlauben somit sehr kompakte und flache Schaltkreise. Allerdings ist das Verlöten um einiges anspruchsvoller und für Anfänger und Hobbybastler zu Beginn eher ungeeignet. Man schaue sich nur einmal kleine Widerstände in SMD-Bauform an. Ohne Lupe und Pinzette ist ein Erkennen der Beschriftung oder ein Hantieren damit oftmals mehr Glücksspiel als ernsthaftes Arbeiten.

Bild 3.2: Ein Mikrocontroller in SMD-Form.

Diversität von Controllerfamilien und -Herstellern

So klein und unscheinbar Mikrocontroller daherkommen, so groß ist die Diversität an Controllerfamilien und -Herstellern. In diesem Buch wird es hauptsächlich um AVR-Mikrocontroller gehen, die von der Firma Microchip Technology Inc. herstellt werden. Ursprünglich wurden die Mikrocontroller von der Firma Atmel produziert, bis diese übernommen wurde. Der Name spiegelt sich in den Bezeichnungen einzelner Controllerreihen wider, wie etwa ATmega oder ATtiny.

Die Namen der Reihen verraten schon die hauptsächlichen Unterschiede – diese liegen in Größe und Ausstattung. Von einzelnen Ausreißern abgesehen, gehen mit dem ATtiny z. B. eine geringere Leistungsfähigkeit und ein kleinerer Speicher einher. Die geringere Größe und die reduzierte Anzahl von Anschlüssen qualifizieren den ATtiny allerdings für kompaktere Schaltkreise und können den Einsatz für spezifische, einfache Aufgaben erforderlich machen.

Nicht vergessen werden sollte der Vorteil der geringeren Kosten, die allerdings auch beim ATmega nicht der Rede wert sind. Der Unterschied reicht vom Cent-Bereich bis zu wenigen Euro. Einzelne Vertreter der ATtiny-Reihe lassen sich sogar schon mit der Spannung einer einzelnen Batterie versorgen. Controller der Reihe ATmega hingegen besitzen mehr Speicher, mehr Anschlüsse und eignen sich somit für verschiedene Einsatzgebiete und auch für mehrere Aufgaben gleichzeitig in einem komplexeren Projekt.

Anno 2005: Es darf programmiert werden

So mächtig der Einsatz von Mikrocontrollern ist, die Verwendung in Schaltkreisen und die Programmierung erfordern sehr viel Einarbeitungszeit, eine gute technische Auffassungsgabe, und zur Programmierung werden Kenntnisse in der hardwarenahen maschinenorientierten Programmiersprache Assembler oder einer höheren Programmiersprachen wie C oder BASIC benötigt. Dazu ist mindestens ein ausführliches Studium der Datenblätter vonnöten sowie das Verständnis der Elektronik und des Speicheraufbaus auf der Ebene von Speicherregistern, um verrückte Sachen zu machen, wie Fuse-Bits zu ändern oder zwischen externen Oszillatoren oder stromsparenden, internen Low-Frequency-Oszillatoren umzuschalten.

Auch der Vorgang der Manipulation und des Auslesens von Bits in die verschiedenen Speicherregister DDRx, PORTx sowie PINx, um Pins als Ein- oder Ausgang zu definieren und Werte lesen oder schreiben zu können, klingt nicht so, als würde er locker von der Hand gehen. Selbst mit der erwähnten positiven Entwicklung von Mikrocontrollern, hin zum Einsatz im Privatbereich, beispielsweise durch die Verwendung von Flashspeichern, erfordert die bloße Nutzung weiterhin ein intensives Studium aller Komponenten.

Mikrocontroller-Boards kommen ins Spiel

Ruft man sich das Erscheinungsbild eines Mikrocontrollers in Erinnerung – ein Mikrochip mit mehreren Pins –, fragt man sich unweigerlich, wie man einen Einstieg in die Entwicklung, beispielsweise ins Prototyping, mit schnellen ersten Erfolgen, bewerkstelligen kann. Hier kommen Mikrocontroller-Boards ins Spiel. Dabei handelt es sich um fertige Schaltkreise auf einer Leiterplatte, die den Mikrocontroller um eine Spannungsversorgung sowie weitere externe Bauteile und Schaltglieder, wie Taktgeber oder Speicher, erweitert, um einfacheres Arbeiten in der Mikrocontroller-Programmierung und -Modellierung zu ermöglichen.

Solche Boards gibt es bereits seit einiger Zeit und in verschiedenen Ausführungen, ausgerichtet auf diverse Anwendungszwecke. Für den Einstieg in die Mikrocontroller-Welt bzw. für erste Entwicklungsarbeiten vertreibt beispielsweise der Hersteller von AVR-Mikrocontrollern Einstiegs- und Entwicklungsboards mit den Namen STK200, STK500 und STK600.

Die Boards sind im Aufbau relativ groß und umfangreich. Wir selbst haben mit solchen Boards bisher keine Erfahrung sammeln können, der Einarbeitungs- und Programmieraufwand dürfte jedoch relativ hoch sein. Auch der Preis, angefangen bei 60 Euro bis mehr als 200 Euro für neue Boards, kann als Einstiegshürde gelten. Die Zielgruppe dürfte jedenfalls eine andere sein,

und so sind wir als Mikrocontroller-Neulinge mit dem Wunsch, einen DIY-MIDI-Controller zu bauen, mit diesen Boards schlichtweg schlecht beraten.

Sucht man nach anderen Controllerfamilien und -herstellern, ist eigentlich fast immer die Rede vom Arduino. Er genießt eine enorme Verbreitung und Beliebtheit sowohl in der Maker- und DIY-Szene als auch in der Forschung und Entwicklung an Universitäten und in Unternehmen. Es gibt zu fast jedem DIY-Mikrocontroller-Projekt eine Arduino-Umsetzung, und ein Großteil der Projekte ist gut dokumentiert im Internet zu finden. Dieser Siegeszug fand seinen Anfang im Jahr 2005 in Italien. Dort gründeten Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino sowie David Mellis mit Arduino eine Open-Source-Plattform, bestehend aus einem Mikrocontroller-Board sowie einer Softwarekomponente zur Programmierung dieser Boards.

Beschäftigt man sich intensiver mit der Gründungshistorie, begegnen einem etliche konträre Aussagen, was durchaus in Verwirrung enden kann. Tatsächlich beginnt die Geschichte schon etwas früher, nämlich im Jahr 2003, als im Rahmen der Masterarbeit von Hernando Barragán eine Plattform namens *Wiring* entstand. Dort wurde das grundlegende Schema von Hard- und Software entwickelt mit dem Ziel, eine Entwicklungs- und Experimentierplattform zu schaffen, die kostengünstig ist und mit der sich auch ohne großartige Programmierkenntnisse spannende Projekte umsetzen lassen.

2005 wurde auf dieser Grundlage unter dem Namen Arduino eine etwas weiterentwickelte Version als Abspaltung von Wiring (man spricht hier von einem Fork) veröffentlicht. Der Open-Source-Charakter blieb zwangsläufig bestehen, und so kann jeder das Schema des Boards oder den Code der Software für sich nutzen, vervielfältigen und weiterentwickeln. Lediglich die Verwendung des Markennamens »Arduino« ist geschützt. So öffnete sich im Laufe der Geschichte, in der der Arduino schnell weltweit als beliebte Entwicklungsplattform auf dem Vormarsch war, ein weiteres Kapitel, in dem es unter anderem um die Nutzung der Markenrechte ging.

Die beiden Firmen Arduino SRL (exklusiver Boardhersteller) und Arduino LLC (Gründergruppe) begaben sich in einen Rechtsstreit um Markenrechte und Lizenzgebühren, was dazu führte, dass es zwischenzeitlich zwei offizielle Webauftritte gab und dass das offizielle Arduino-Board in Teilen der Welt als Genuino-Board vertrieben wurde. Inzwischen stellen viele verschiedene Hersteller die offiziellen Arduino-Boards her, und wegen der öffentlich verfügbaren Boardarchitektur existieren inzwischen zahlreiche Derivate und Abwandlungen unter verschiedenen Namen, wie Freeduino oder Seeduino. Größtenteils funktionieren diese Boards genauso wie die Originale und sind oftmals kompatibel zur Arduino-Programmiersoftware.

Während bei anderen Mikrocontroller-Boards wie den erwähnten STK200 oder STK500 ein großes Augenmerk auf den Mikrocontroller liegt, steht er bei Arduino durch Architektur und verwendeter Programmiersprache eher im Hintergrund. Der Arduino lässt sich durch seinen Aufbau als kompaktes Ganzes, als abstrahiertes, aber mächtiges Bauteil bzw. als Blackbox betrachten und benötigt keinen Blick auf das spezifische Innenleben, wie etwa Taktgeber oder Speicher. Die Software lässt zwar eine hardwarenahe Programmierung zu, im Standardfall nutzt man jedoch die vereinfachte und oftmals selbstbeschreibende Form einer höheren Programmiersprache, ohne sich mit spezifischen Eigenheiten wie beispielsweise der Speicherbelegung des Mikrocontrollers beschäftigen zu müssen.

Auch gibt es in der IDE für etliche Anwendungsszenarien bereits integrierte oder nachladbare Programmbibliotheken (Libraries), die komplexe Programmiermuster auf wenige bis einzelne Zeilen reduzieren. So lassen sich mit Arduinos auch ohne Vorwissen sehr schnell Erfolge erzielen. Und war es bei den ersten Boards noch anders, kann man heutzutage Arduinos einfach mit einem USB-Kabel an den PC stecken und zu entwickeln beginnen. Dazu ist inzwischen noch nicht einmal eine installierte Software notwendig, da die Programmierschnittstelle – die Arduino-IDE – nun auch als Browserversion angeboten wird.

Was Arduino-Boards alles können

Werfen wir einen Blick auf die Boards selbst – was sind ihre Vorteile gegenüber allein stehenden Mikrocontrollern, und wie vereinfacht das die Arbeit mit ihnen? Bringen Mikrocontroller als Taktgeber meist einen internen RC-Oszillator mit, wird auf einem Arduino-Board für die Taktgebung ein Quarzkristall (oftmals auch nur Quarz genannt) als separates Bauteil genutzt. Der interne Oszillator eines ATmega328P-AVR-Mikrocontrollers erzeugt nur eine vergleichsweise geringe Frequenz von 8 MHz. Im Vergleich zur Taktfrequenz eines Prozessors im PC, wo wir uns schon im Gigahertz-Bereich befinden, erscheint das ziemlich gering.

Da Mikrocontroller keine Allzweckwaffe sein sollen und nur für spezifische Teilaufgaben eingesetzt werden, reicht das jedoch in vielen Fällen aus. Der genannte Mikrocontroller unterstützt maximal eine Frequenz von 20 MHz aus einer externen Taktquelle. Auf einem Arduino Uno, der diesen Mikrocontroller verwendet, ist ein Quarz verbaut, der eine Taktfrequenz von 16 MHz vorgibt. Dadurch ist ein größeres Einsatzspektrum möglich, da diese Frequenz eine deutlich schnellere Datenverarbeitung gestattet.

Doch nicht nur die höhere Frequenz gehört zu den Vorteilen des externen Taktgebers. Wie bereits erwähnt, werden Quarzoszillatoren für ihre hohe Genauigkeit geschätzt. Während der interne Oszillator laut Datenblatt ab Werk eine Genauigkeit von ±10 % haben kann, ist dieser Wert bei Quarzkristallen mit einem Genauigkeitsbereich von unter ±0,001 % deutlich geringer. Statt % (Prozent) nutzt man in diesem Bereich eher die Einheit ppm (*parts per million*). Damit sind Operationen mit Anforderungen an eine hohe Frequenzstabilität möglich, wie beispielsweise die Kommunikation über eine serielle Schnittstelle mit einem PC.

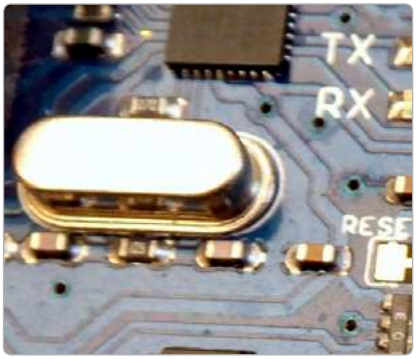


Bild 3.3: Mithilfe eines Quarzoszillators wird die Leistung des Mikrocontrollers verdoppelt.

Mikrocontroller sind klein und besitzen diverse Anschlusspins. Ohne viel zu löten und die entsprechende Feinarbeit ist es gar nicht möglich, selbst einfache Schaltkreise zu bauen und einen Mikrocontroller darin einzugliedern, indem man Schnittstellen zu seinen Pins herstellt. Mit den größeren Beinchen, die sich in Platinen stecken und verlöten lassen, ist die DIP-Bauform im Privatbereich zwar etwas einfacher handhabbar als die meist kleineren Mikrocontroller in SMD-Bauform, dennoch kann man einen Schaltkreis nicht eben mal umstecken, falls einem ein logischer Fehler auffällt oder man Erweiterungen vornehmen möchte.

Schaltkreise ohne aufwendige Lötarbeit

Man benötigt in diesen Fällen doch wieder einen Lötkolben. Dieser Umstand ist für Experimente oder für Einsteiger mit wenig Elektronikerfahrung ganz offensichtlich ein Hindernis. Deshalb sind die Pins des verlöteten Mikrocontrollers mit gut erreichbaren Anschlüssen am Board verbunden, mit denen sich in der Grundform über einen Plastikaufsatz mittels Steckverbindung Kabel einfach verbinden lassen. Zusammen mit steckbaren Elektronikbauteilen und einem Breadboard lassen sich so sehr schnell Schaltkreise ohne Lötarbeit zusammenstellen und ändern.

Auf diese Pins lohnt es sich, genauer zu schauen. Es stehen diverse Funktionen zur Verfügung. Zum einen gibt es Pins, die als digitaler Input oder Output (I/O) nutzbar sind, und dann gibt es noch die analogen Input-Pins. Was diese können, kann man sich wahrscheinlich bereits denken. Die digitalen Pins arbeiten mit den Zuständen Spannung (1) und keine Spannung (0). So ist es möglich, Schalter-/Tasterzustände auszulesen, LEDs zu beleuchten oder Schaltungen zu steuern. Dank eingebauter Pull-up-Widerstände lassen sich je nach programmierter Konfiguration die Zustände der eingelesenen Werte vertauschen, beispielsweise liegt am Mikrocontroller-Eingang keine Spannung an, wenn ein Taster gedrückt wurde.

Analoge Input-Pins eignen sich hervorragend zum Auslesen von Sensoren, etwa Temperatur- oder Feuchtigkeitssensoren oder lichtempfindlichen Sensoren. Echte analoge Ausgänge gibt es nicht. Einige Pins können dennoch quasi-analoge Signale ausgeben. Das gelingt mittels Pulsweitenmodulation, kurz PWM. Programmiert werden diese Pins so, als würde man ein analoges Signal ausgeben wollen, und die interne Logik moduliert ein digitales Signal (Spannung und keine Spannung) in der Frequenz – das heißt, die Spannungswechsel variieren in ihrer Pulsdauer. Daran angeschlossene elektronische Komponenten, wie etwa Motoren oder LEDs, reagieren aufgrund ihrer Trägheit bzw. der des menschlichen Auges auf die veränderte Pulsdauer so, wie sie auf unterschiedliche Spannungswerte eines analogen Ausgangssignals reagieren würden.

Mit diesen Voraussetzungen sind einem fast keine Grenzen im Auslesen von Signalen und im Steuern von Geräten gesetzt. Aber eben nur fast, denn nicht jeder Motor lässt sich über einen Output-Pin des Arduino-Boards betreiben. Ein einzelner Pin hat eine maximale Ausgangsstromstärke von 40 mA (A = Ampere, mA = Milliampere). Zusammen mit der anliegenden Spannung von 5 V (Volt) ergibt das eine Leistung von 0,2 W (Watt). Es gibt sicher Mini-Motoren, denen diese Leistung ausreicht, für die meisten anderen muss man jedoch in die Trickkiste greifen und mit dem Output-Pin etwa ein Relais oder eine Transistorschaltung ansteuern.

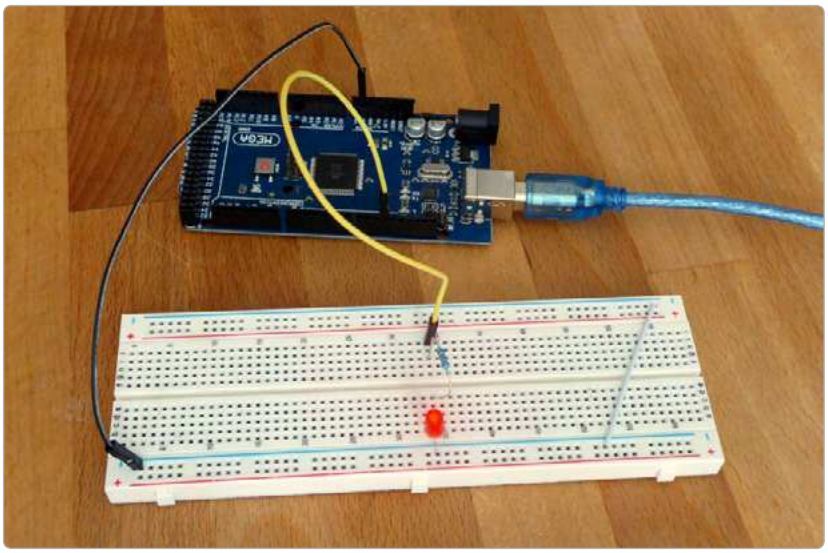


Bild 3.4: So sieht ein einfaches Beispiel für eine Arduino-Schaltung aus. Wir haben eine LED angeschlossen und bringen sie zum Leuchten.

Arduino-Boards programmieren

Anfangs benötigte man zur Programmierung von Mikrocontroller-Boards noch ein externes Programmiergerät. Inzwischen lassen sich Arduino-Boards direkt über die USB-Schnittstelle von einem PC aus programmieren. Zu diesem Zweck ist ein kleines Skript namens Bootloader auf dem Board vorinstalliert, das dafür sorgt, dass eingehende Programmierversuche den Programmspeicher des Controllers überschreiben können.

Doch auch ohne Bootloader ist der Mikrocontroller programmierbar, da das Board über entsprechende ISP- (*In-System-Programming*) oder ICSP-Header (*In-Circuit-Serial-Programming*) verfügt, an denen sich ganz klassisch externe Programmiergeräte anschließen lassen. Mit einem solchen externen Programmierer verzichtet man zwar auf die Plug-and-program-Bequemlichkeit durch Löschen des Bootloaders, kann jedoch in vielen Fällen den zusätzlichen Speicher, den vorher der Bootloader belegte, für eigene Zwecke nutzen.

Was Arduino-Boards nicht können

Ebenfalls im DIY-Bereich bekannt und beliebt ist der Raspberry Pi. Beide sind kleine, einsteigerfreundliche Systeme, deren Boards mit programmierbaren Pins (GPIO) bestückt sind und deren Ursprung und Zweck in kostengünstigen Systemen zum Programmieren und Experimentieren liegt. Jedoch unterscheiden sie sich in ihrer Leistungsfähigkeit und ihren Anwendungsgebieten. Damit kommen wir zu der Frage, was Arduino-Boards alles nicht können.

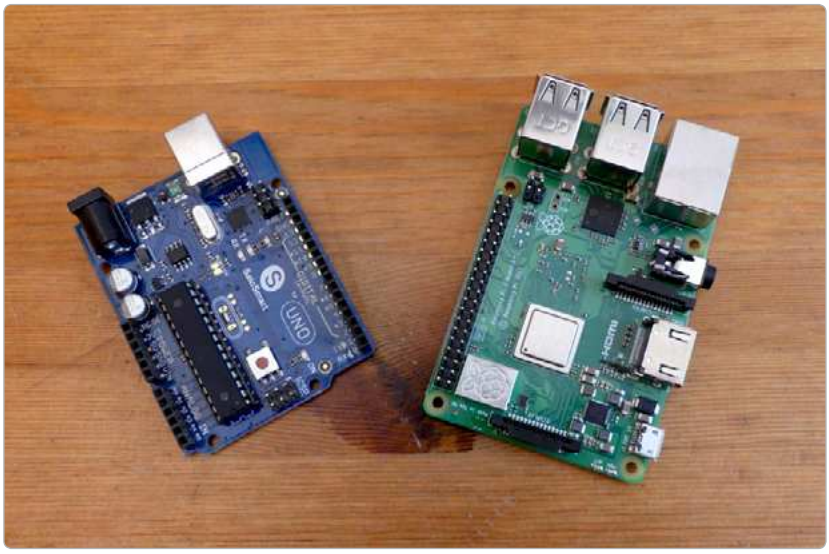


Bild 3.5: Arduino und Raspberry Pi (rechts) sehen ähnlich aus, unterscheiden sich jedoch fundamental in ihren Anwendungsgebieten.

Während der Raspberry Pi mit viel mehr Rechenleistung im Gigahertzbereich, einem Betriebssystem und somit potenziell auch mit einer grafischen Oberfläche daherkommt, besteht der Arduino tatsächlich nur aus dem Mikrocontroller, der über eine Schnittstelle programmiert werden muss. Trotz aller genannten Vorteile ist die Einstiegshürde beim Arduino also eher etwas höher, was für motivierte Einsteiger allerdings dank der simplen Entwicklungsumgebung und der einfachen seriellen Verbindung über USB ein eher vernachlässigbarer Punkt sein dürfte.

Die geläufigsten Arduino-Boards sind in den meisten Fällen tatsächlich nur zur Steuerung von elektrischen Bauteilen, Sensoren oder Aktoren geeignet. Möchte man komplexere Aufgaben wie die Kommunikation über WLAN oder Bluetooth realisieren, sollte man zunächst doch lieber zum Raspberry Pi greifen. WLAN- oder Bluetooth-Dongles sind hier an den USB-Buchsen ansteckbar und können mithilfe geeigneter Treiber für das jeweilige Betriebssystem (in den meisten Fällen Linux) nachinstalliert werden. Oder sie sind bereits integriert und damit vergleichsweise schnell eingerichtet. Neuere Geräte, wie der Raspberry Pi 3, haben sogar bereits ein entsprechendes Modul integriert.

Auch andere Schnittstellen bringt der Raspberry Pi mit, wie das *Display Serial Interface* (DSI), HDMI für den Anschluss von Bildschirmen oder das *Camera Serial Interface* (CSI) für Kameras. Wer nun denkt, dass es keinen Weg gäbe, solche Dinge mit dem Arduino umzusetzen, dem sei gesagt: Es gibt Lösungen. Das ist der Punkt, an dem Arduino-Shields zum Einsatz kommen. Das sind Module, die auf die GPIO-Pinleisten gesteckt werden und das Board um Bauteile und Funktionen erweitern. Geläufige Shields geben einem Arduino-Board beispielsweise die Möglichkeit, über WLAN zu kommunizieren (Wi-Fi-Shield) oder mittels SIM-Karte SMS und Anrufe

zu tätigen und über GPRS zu kommunizieren (GPRS-/GSM-Shield). Ein Proto-Shield erweitert das Board um eine Lochrasterplatine, auf der sich Schaltkreise mit weiteren elektrischen Bauteilen platzsparend und übersichtlich auflöten lassen. Manche Funktionen, wie z. B. Bluetooth, sind mittlerweile in einigen Arduinos sogar von Haus aus enthalten. Der Arduino BT beispielsweise benötigt kein Shield mehr, um eine zusätzliche Bluetooth-Schnittstelle bereitzustellen.

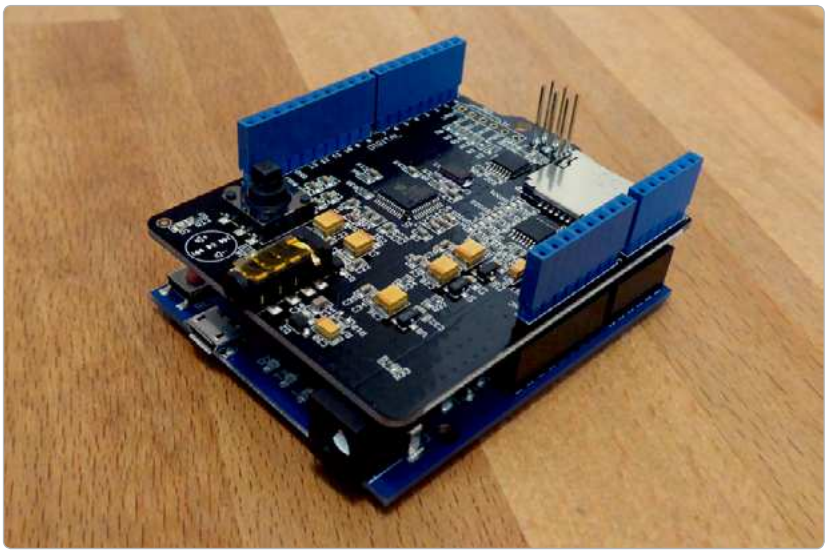


Bild 3.6: Mit einem Shield können wir den Arduino um Funktionen erweitern. Das Music Shield V2.2 kann beispielsweise MIDI- und Musikdaten auf dem Mikrocontroller abspielen.

Einblick in die Welt der Mikrocontroller-Boards

Die Mikrocontroller-Welt ist vielfältig, und auch Arduino-Boards kommen in etlichen Aufmachungen daher. Sowohl die Software als auch die Boards selbst sind unter Open-Source-Lizenzen öffentlich und somit auch kommerziell verwendbar, lediglich die Marke Arduino bzw. Genuino ist geschützt. So gibt es neben den originalen Arduino-Boards eine gewisse Auswahl an baugleichen Boards von anderen Marken wie etwa Adafruit. Man muss lediglich ab und zu einen Blick auf die Verarbeitungsqualität werfen, da wirklich jeder heutzutage ein entsprechendes Board herstellen darf. Doch wie kam es eigentlich dazu?

Zuerst müssen wir etwas ausholen, um den Hintergrund des Erfolgs des Arduino und die damit einhergehende enorme Bereicherung für die Open-Source-Welt näher zu beleuchten. Mikrocontroller sind mit ihrem Funktionsumfang auf Low-Level-Ebene bestens dafür geeignet, Projekte wie unsere umzusetzen, da sie nur einen bestimmten Zweck erfüllen müssen – wie etwa die Verarbeitung von Tastendrücken und das Senden und Empfangen von Signalen an den PC zur Steuerung einer Software. Dazu ist kein funktioneller Overhead nötig, wie ihn eine Ent-

wicklungsplattform wie der Raspberry Pi mit sich bringt. Seine Funktion nimmt der Arduino nahezu sofort nach Beginn der Spannungsversorgung auf, ohne langwierig ein Betriebssystem starten zu müssen. Auf dieser minimalen Ebene ist er im Betrieb zudem stromsparender. Mit der Arduino-Plattform ist es nun auch technischen Laien möglich, mit einem Mikrocontroller zu entwickeln.

Als wäre die Bandbreite der verfügbaren Mikrocontroller-Boards aus dem Hause Arduino mit ihren diversen Formen und Größen und den unterschiedlichen Hardwarekonfigurationen nicht schon groß genug, gibt es eine Menge fremder Produkte, die in Teilen Arduino-kompatibel sind (und somit beispielsweise mit der Arduino-IDE programmiert werden können), angepasst an diverse Einsatzzwecke. Um sich das Universum rund um Mikrocontroller-Boards etwas besser vorstellen zu können, werden im Folgenden ein paar Arduino-Boards bzw. Arduino-kompatible Geräte mit ihren Unterschieden vorgestellt. Die Liste ist bei Weitem nicht vollständig, bietet aber einen Einstieg in die wichtigsten Vertreter. Es gibt noch andere, exotischere Arduino-Derivate und Boards. Wer mag, kann sich hierzu im Internet gern umsehen.

- **Arduino Uno – der Klassiker unter den Arduinos**
  Der Arduino Uno ist quasi das Einstiegsgerät unter den Arduino-Boards. Er kommt im Vergleich zu anderen Geräten mit Standardspezifikationen daher: Seine Maße entsprechen in etwa denen einer Zigarettenschachtel oder einer Chipkarte. Der Uno ist bestückt mit einem ATmega328 und bietet 14 digitale I/O- (als Input und Output verwendbare) sowie sechs analoge Input-Pins. Zwar hat er als Arduino-Board alles, was für eine Plug-and-play-Programmierung benötigt wird, und eignet sich damit für eine Vielzahl von Projekten, trotzdem kommt man bei etwas mehr Komplexität auch schnell an seine Grenzen. Ein großer Vorteil ist jedoch, dass man seinen Chip ganz einfach austauschen kann. Für kleine Projekte ist der Arduino Uno die perfekte Wahl.

Bild 3.7: Der Arduino Uno – das Einsteigerboard.

- **Arduino Mega – die unsinkbare Titanic**
  Der Arduino Mega (2560) ist ein Peripheriewunder. Dank des verbauten ATmega2560 kann er deutlich mehr GPIO-Pins bedienen als viele andere Boards. Zur Verfügung stehen 16 analoge Input-Pins sowie ganze 54 digitale I/O-Pins. Dadurch und durch den etwas größeren Speicher des ATmega2560 lassen sich selbst komplexe Projekte mit nur einem einzigen Board verwirklichen. Sucht man nach Projekten, die den Mega verwenden, stößt man recht häufig auf den Bereich der 3-D-Drucker oder der Robotik. Hinsichtlich der weiteren Schnittstellen unterscheidet sich das Board kaum von anderen Boards, etwa dem Uno. Es lässt sich ebenso leicht programmieren und hat zusätzliche Schnittstellen zur seriellen Kommunikation. Wir behalten den Mega für später im Auge.

Bild 3.8: Der Arduino Mega 2560 – das Peripheriewunder.

- **Arduino Leonardo – die USB-Büchse der Pandora**
  Der Arduino Leonardo hat einen ATmega32U4-Mikrocontroller verbaut. Von den Maßen entspricht er in etwa einem Arduino Uno, er besitzt jedoch mit 20 digitalen I/O- sowie zwölf analogen Input-Pins etwas mehr Spielraum. Anders als bei anderen Boards, bei denen für gewöhnlich ein FTDI-Chip die Kommunikation über eine USB-Schnittstelle übernimmt, kann der ATmega32U4 diese Rolle selbst übernehmen. An einen PC angeschlossen, wird der Leonardo sofort als interaktionsfähiges USB-Gerät erkannt und kann als Human-Interface-Device genutzt werden. Auf anderen Boards müsste man dafür Mehraufwand betreiben und den Chip flashen, der für die USB-Schnittstelle verantwortlich ist.

  Mit dieser Besonderheit wird der Leonardo mit Vorliebe in Projekten eingesetzt, in denen mit einem interaktiven USB-Gerät, etwa simulierte Maus, Tastatur oder Joystick, Eingaben an einem PC vorgenommen und zurückgegebene Signale ausgewertet werden sollen. Auch als MIDI-Controller kann der Leonardo eingesetzt werden. Daher werden wir uns dieses Board ebenfalls merken und kommen im späteren Teil des Buchs noch einmal darauf

zurück. Anders als andere Arduinos verwendet der Leonardo einen Micro-USB-Anschluss. Das kann man sowohl als Vorteil – kleine Buchse und damit platzsparender – als auch als Nachteil werten – empfindlicherer Kabelanschluss.

Bild 3.9: Der Arduino Leonardo – die USB-Referenz.

- **Arduino Micro – klein, aber fein**
  Der Arduino Micro ist hinsichtlich seiner Größe quasi das Gegenstück zum Arduino Mega und eine Zwergenversion des Arduino Leonardo. Der Name sagt es schon, der Sinn dieses Boards liegt darin, die gesamte Architektur eines Mikrocontroller-Boards auf möglichst kleinem Raum unterzubringen. Der Micro ist kaum größer als eine Streichholzschachtel und eignet sich besonders gut für Projekte, in denen platzsparende Hardware benötigt wird, ohne groß auf Funktionalität verzichten zu müssen. Das Mikrocontroller-Board nutzt einen ATmega32U4, womit er ebenfalls, als interaktives USB-Gerät eingesetzt werden kann. Den geringen Maßen zum Trotz stehen 20 digitale I/O-Pins zur Verfügung. Daneben gibt es noch den Arduino Pro Mini. Er unterscheidet eigentlich nur darin, dass er mit dem ATmega328P ausgestattet ist.

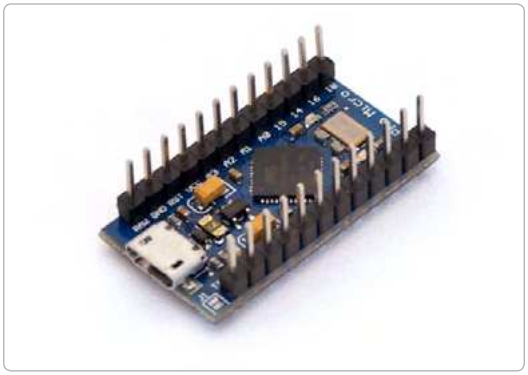


Bild 3.10: Der Arduino Pro Micro – ein Board so groß wie eine Streichholzschachtel.

- **Digispark – geschaffen für Minimalisten**
  Der Digispark ist ein Mikrocontroller-Board mit einem ATtiny 85. Die offensichtlichsten Vorteile dieses Boards erkennt man im Bild sofort: Es ist deutlicher kleiner als die meisten Boards (selbst der Arduino Micro ist größer) und kann durch seine Bauweise mit integriertem USB-Stecker direkt in entsprechende Buchsen gesteckt werden. Dort wird der Digispark für gewöhnlich auch sofort als Tastatureingabegerät erkannt. Zwar hat er nur sechs GPIO-Pins, allerdings erlaubt die minimale Ausstattung einen im Vergleich sehr geringen Anschaffungspreis.

  Aktuell ist er am Markt für wenige Euro zu haben. Dennoch wird der Digispark von der Arduino-IDE unterstützt, es sind allerdings eventuell weitere Softwarebibliotheken notwendig, die sich in der IDE aber installieren lassen. Dank I^2C- und SPI-Schnittstellen eignet er sich sogar zur Kommunikation mit anderen Geräten. Eingesetzt wird er in Projekten, in denen es darauf ankommt, Platz zu sparen, und die nur wenige Pins benötigen. Dank seiner USB-Tastatureigenschaften wird er dem einen oder anderen vermutlich auch als Hackertool bekannt sein, mit dem man beispielsweise unentdeckt am USB-Slot eines PCs Tastatureingaben mitloggen oder Befehle eingeben kann. Ein Schelm, wer Böses dabei denkt!

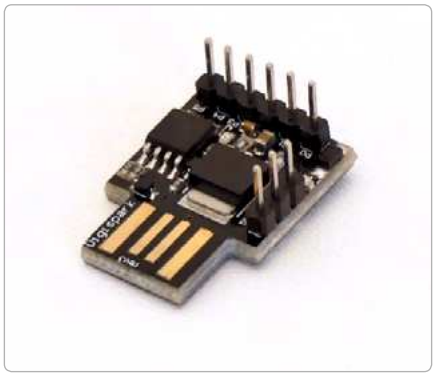


Bild 3.11: Der Digispark rev3 – ein Arduino-kompatibles Board in Mini-Größe.

Arduino-IDE – Programmierung für jedermann

Arduino ist kein reines Hardwareprojekt. Zwar haben die Open-Source-Boards einen großen Anteil an der Verbreitung und der Beliebtheit der Marke Arduino bzw. der Boards unter den zahlreichen anderen Namen, die Hardware allein macht jedoch noch keine einfache Handhabung der Mikrocontroller aus, wenn sie selbst nach langer Einarbeitungszeit nur umständlich oder mit komplizierten Softwaretools als sogenannte Toolchain programmierbar sind. Hier kommt die Softwarekomponente von Arduino ins Spiel. Mit der Arduino-IDE lassen sich die Boards bereits mit wenigen Zeilen Code und nur ein paar Mausklicks programmieren. Eine Vielzahl der Modelle wird unterstützt. Doch zuerst etwas zu den Grundlagen.

Programmieren kann im Grunde jeder

Zum Programmieren braucht es nicht einmal komplexe Werkzeuge. Für den Anfang reicht ein einfacher Texteditor wie der Microsoft-Editor oder Notepad++. Wer mag, kann natürlich ein Stück weiter gehen und ganz auf eine grafische Oberfläche verzichten, indem er in einer Kommandozeile Editoren wie nano oder vi nutzt. Das ist uns natürlich nicht genug und bringt uns zur Grundfunktionalität einer IDE und dem Anfang eines Programms: Es muss Text zu Papier gebracht werden. Mit diesem Stück Text, dem sogenannten Quellcode, kann man noch nicht viel anfangen. An diesem Punkt stellt sich die Frage, wie der Code auf das Arduino-Board kommt. Und wie kann dieser Text in einen maschinenlesbaren Code umgewandelt werden? Denn auch wenn ein Mensch die Befehle lesen und interpretieren kann, ein Mikrocontroller kann mit dieser Form nichts anfangen. Er kann nur den für normale Menschen unlesbaren Maschinencode (bei AVR-Mikrocontrollern als Hex-File vorliegend) ausführen, deshalb benötigen wir einen Übersetzer. Dieser wird Compiler genannt und kann als eigenständiges Programm ausgeführt werden oder integrierter Bestandteil einer IDE sein. Zu guter Letzt wird ein Tool benötigt, um den übersetzten Code auf den Mikrocontroller zu bringen. Damit haben wir schon die grundlegenden Bestandteile, die eine IDE für die Mikrocontroller-Programmierung benötigt.

Was ist eine IDE?

Die IDE, das *Integrated Development Environment*, auf Deutsch »integrierte Entwicklungsumgebung« ist ein Bündel von verschiedenen Elementen, Werkzeugen und Hilfsprogrammen, mit denen man Software entwickeln kann. Die Arduino-IDE setzt die AVR Toolchain ein, eine Abwandlung der freien Softwaresammlung *GNU Compiler Collection* (GCC). Neben dem erwähnten Compiler, hier *gcc* (klein geschrieben für GNU C Compiler), und dem Tool zur Übertragung auf den Mikrocontroller, *avrdude*, beinhaltet diese Toolchain diverse weitere Dinge, etwa einen Linker und Libraries, deren Funktion wir an der Stelle nicht weiter vertiefen wollen.

Bild 3.12: Die Arduino-IDE mag zwar eine rudimentäre Entwicklungsumgebung sein, aber dafür ist sie sehr komfortabel und auf den Punkt gebracht.

Die Funktionalität der Arduino-IDE beschränkt sich auf das Wesentliche. Gimmicks professioneller IDEs, wie automatische Codevervollständigung oder integrierter Code-Sniffer zur Einhaltung von Codestyle-Regeln, existieren hier nicht, sind unvollständig oder werden erst in neueren Betaversionen getestet. Man kann sich darüber streiten, ob das ein Vorteil für Einsteiger oder ein Nachteil für fortgeschrittene Entwickler ist. Letztere können bei Missfallen allerdings auch auf andere etablierte IDEs wie Visual Studio ausweichen.

Die von der Arduino-IDE verwendete Programmiersprache ist eine leichte Abwandlung von C und C++. Sie wurde für den Nutzer so vereinfacht, dass der C++-spezifische Umgang mit Programm- (*.cpp*) und Header-Files (*.h*) entfällt. Es wird lediglich mit Sketchen gearbeitet, im Editor dargestellt als einzelner oder mehrere Tabs. Der Haupt-Tab enthält die grundlegende Programmroutine, alle weiteren Tabs, mit ausgelagerten Funktionen, werden von der IDE automatisch in den Sketch integriert. Im Dateisystem werden die einzelnen Tabs mit der Dateiendung *.ino* abgelegt. Die Arduino-spezifischen Programmbefehle, wie `pinMode()` oder `digitalWrite()`, vereinfachen umfangreichere Methoden zu selbstbeschreibenden Einzeilern. Neben der vereinfachten Programmiersprache unterstützt die Arduino-IDE das Programmieren unter anderem mit ausführlichem C++ oder dem maschinennahen Assembler. Die Arduino-IDE wurde in der plattformunabhängigen Programmiersprache Java entwickelt, sodass sie sowohl auf Windows und macOS als auch auf Linux-Betriebssystemen lauffähig ist. Neuerdings existiert außerdem ein Web-Editor. So kann man sein Arduino-Board auch ohne Installation der IDE programmieren und spart sich bei neuen Releases den Patch auf eine aktuelle Version der IDE.

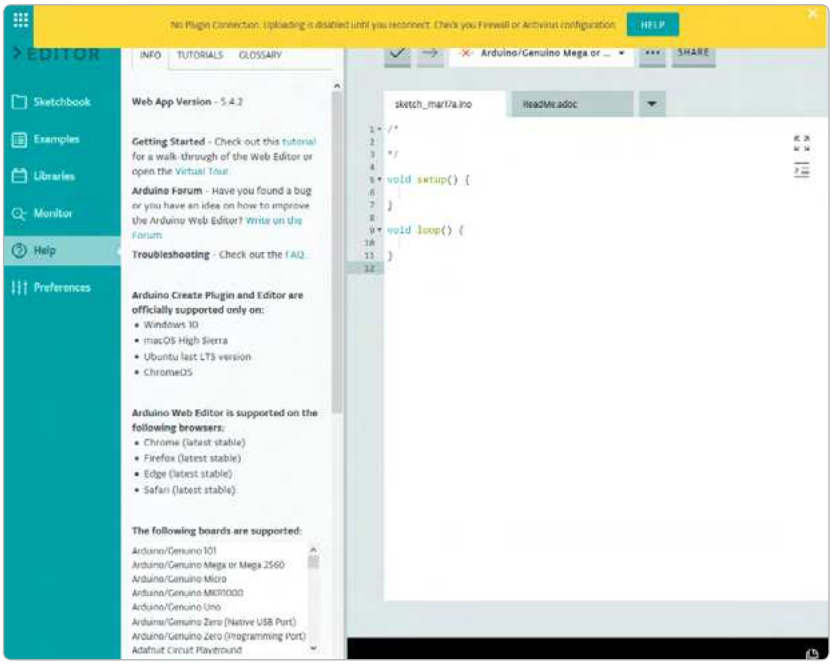


Bild 3.13: Der Arduino-Web-Editor eignet sich ideal als Alternative zur lokalen Entwicklungsumgebung.

Skript überprüfen und übertragen sowie der serielle Monitor

Sehen wir uns nun die grundsätzliche Bedienung der IDE an – von der Erstellung eines Sketchs bis zur Übertragung auf ein Arduino-Board und dem Auslesen von Informationen. Hat man ein neues, originalverpacktes Board vor sich liegen, hat es in vielen Fällen bereits ab Werk einen Programmcode aufgespielt. Er lässt eine auf dem Board integrierte LED blinken, die intern mit dem digitalen Pin #13 verbunden ist. Wir nutzen im Folgenden einen Sketch, der sowohl diese LED blinken lässt als auch über die serielle Schnittstelle eines Arduino den Satz »Hello World!« sendet. Warum? Wir wollen uns den traditionellen Einstieg in eine neue Programmiersprache mit »Hello World!« nicht nehmen lassen. Dieser Sketch beinhaltet mit der Ansteuerung eines Pins sowie der Bedienung der seriellen Schnittstelle schon zwei grundlegende Punkte der Arduino-Programmierung, wir werden an dieser Stelle jedoch nicht weiter auf das Skript und dessen Funktionsweise eingehen. Dazu soll das nächste Kapitel dienen. Hier nehmen wir es zur Demonstration der IDE-Funktionsweise vorerst als gegeben an. Als Erstes laden wir uns die Entwicklungsumgebung der Arduino-Website (*https://www.arduino.cc/*) herunter und installieren sie für unser jeweiliges Betriebssystem.

Für Windows gibt es zusätzlich noch eine portable Version, die keine Installation erfordert. Die Zeilennummern in den Codebeispielen dieses Buchs sind möglicherweise nach der Installation der IDE standardmäßig abgeschaltet. Wer dennoch damit arbeiten möchte, kann sie sich in den Voreinstellungen mit angehakter Checkbox anzeigen lassen, erreichbar im Menü unter *Datei/Voreinstellungen/Zeilennummern anzeigen*.

HelloWorld §

```
1  const int LED = 13;
2
3  void setup() {
4    pinMode(LED, OUTPUT);
5    Serial.begin(9600);
6  }
7
8  void loop() {
9    digitalWrite(LED, HIGH);
10   Serial.println("Hello World!");
11   delay(100);
12   digitalWrite(LED, LOW);
13   delay(3000);
14 }
```

Bild 3.14: Das Beispiel *HelloWorld* dient der Einführung in die Arbeit mit der Arduino-IDE.

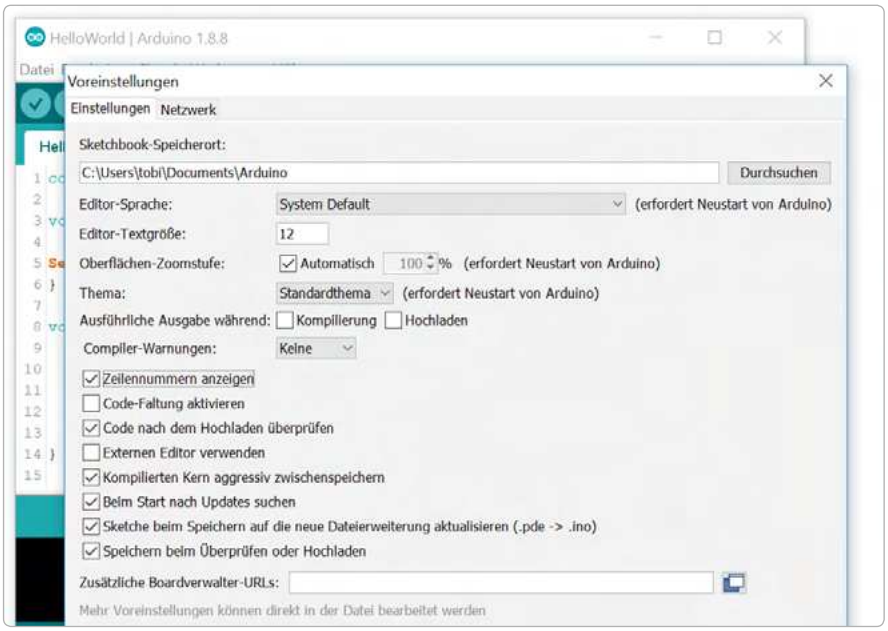


Bild 3.15: In den Voreinstellungen lassen sich Aussehen und Verhalten der IDE individualisieren.

Bevor der Programmcode kompiliert werden kann

Bevor wir den Programmcode kompilieren lassen können, müssen wir der IDE Bescheid geben, für welches Arduino-Board der Compiler den Code übersetzen soll. Wegen architekturbedingter Abweichungen zwischen den vielen verschiedenen Boards und dem verfügbaren Speicherplatz kann es Unterschiede bei der Übersetzung geben.

1. Im Menü gehen wir auf *Werkzeuge/Board* und wählen in der Liste das gewünschte Arduino-Board aus. Die IDE stellt bereits eine große Auswahl zur Verfügung.
2. Da wir für das Ziel dieses Buchs, die Erstellung eines MIDI-Controllers, unter anderem einen Arduino Mega verwenden werden, wählen wir an dieser Stelle im Menü *Arduino/Genuino Mega or Mega 2560* aus.
3. Je nach Typ und Bauweise müssen wir gegebenenfalls noch den richtigen Prozessor angeben. Der Arduino Mega beispielsweise wird mit einem ATmega1280 oder einem ATmega2560 ausgeliefert. Wir wählen passend zu unserem Board unter *Werkzeuge/Prozessor* den *ATmega2560 (Mega 2560)* aus. Wer dieses Beispiel mit einem anderen Arduino testen möchte, muss logischerweise sein Modell auswählen.
4. Da die IDE unser Board nun kennt, können wir den Programmcode kompilieren lassen. Hierzu klicken wir entweder auf das Häkchen links in der Symbolleiste, oder wir gehen im Menü auf *Sketch/Überprüfen/Kompilieren* bzw. nutzen das entsprechende Tastaturkürzel – für Windows: Strg + R.

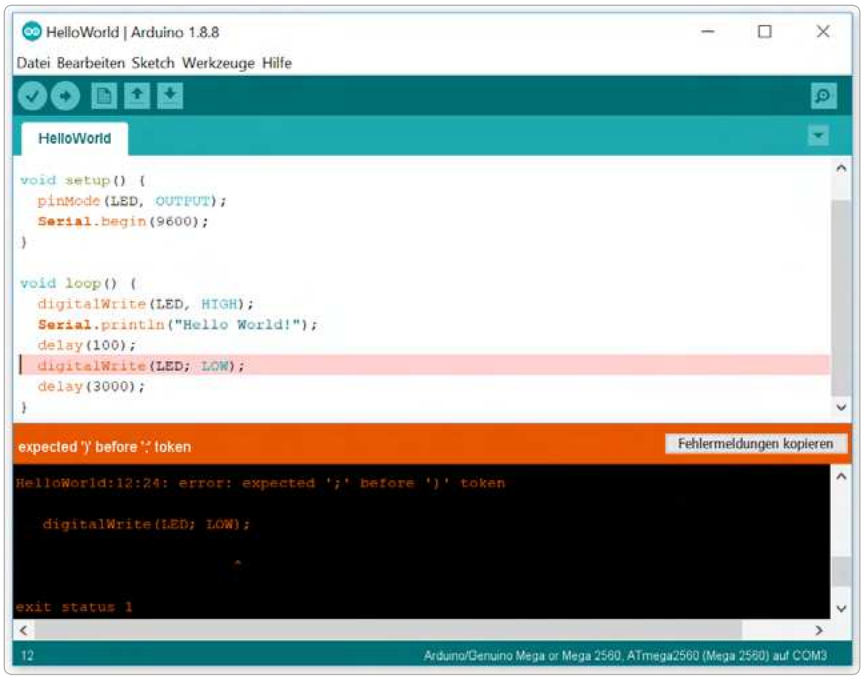


Bild 3.16: Der Compiler weist auf einen Syntaxfehler hin.

Zum Glück ist der Compiler schlau und erkennt sofort einen Syntaxfehler, der sich eingeschlichen hat. Im unteren Bereich der IDE wird der Fehler mal mehr, mal weniger spezifisch angezeigt. Hilfreichere Fehlermeldungen erhalten wir, wenn in der IDE eine ausführlichere Ausgabe aktiviert ist. Das ist möglich unter *Datei/Voreinstellungen/Ausführliche Ausgabe während Kompilierung/Hochladen*.

5. An der Fehlermeldung ist erkennbar, dass wir in Zeile 11 an Position 24 fälschlicherweise ein Semikolon anstelle des richtigen Kommas verwenden. Wir korrigieren den Fehler und kompilieren den Programmcode erneut. Siehe da, es wurden keine weiteren Fehler gefunden, und der sehnsüchtig erwartete Schriftzug *Kompilieren abgeschlossen* taucht auf. Im nächsten Schritt wollen wir den erfolgreich kompilierten Code zum Arduino übertragen.

6. Dazu verbinden wir ihn über die USB-Schnittstelle mit dem Rechner und wählen in der IDE unter *Werkzeuge/Port* den korrekten Port aus. In Windows werden USB-Geräte über COM-Ports dargestellt. Wenn das Board richtig erkannt wird, findet sich in der Liste unser Arduino-Board mit einem entsprechenden COM-Port und der Typbezeichnung, beispielsweise *COM3 Arduino/Genuino Mega or Mega 2560*.

Bild 3.17: Der Programmcode wurde erfolgreich kompiliert und kann jetzt auf den Arduino übertragen werden.

❼ Nun haben wir alles Notwendige eingestellt und können mit der Übertragung beginnen. Dazu kann das Pfeilsymbol in der Symbolleiste rechts neben *Überprüfen*, der Menüeintrag *Sketch/Hochladen* oder das Tastaturkürzel (Windows: Strg + U) genutzt werden. Der Sketch wird erneut kompiliert, und etwaige Fehler werden direkt angezeigt. Sollte alles fehlerfrei sein, wird der kompilierte Code übertragen, und im Erfolgsfall wird nun in der IDE der Status *Hochladen abgeschlossen* angezeigt. Ab jetzt wird die integrierte LED auf dem Board periodisch blinken.

Die Funktionen *Kompilieren* und *Übertragen* gehen in diesem Beispiel schnell vonstatten, können jedoch je nach Komplexität etwas länger dauern. Auch verschiedene Fehler bei der Übertragung ziehen diesen Schritt unter Umständen in die Länge. Meistens werden im unteren Bereich der IDE entsprechende Fehler angezeigt. Sollte das Übertragen eines Sketchs nach mehrmaligem Durchlauf kein Ende finden, hilft in manchen Fällen das Betätigen des Reset-Tasters auf dem Board vor erneutem Hochladen.

VORSICHT! DER PORT DES BOARDS

Manchmal ändert sich der Port des Boards, ohne uns Bescheid zu geben. Dann erscheint beim Übertragen ein Fehler, bis wir den aktuellen Port im Menü erneut auswählen. Zudem ist uns gelegentlich ein bizarres Verhalten nach dem Board-Reset aufgefallen. Dann taucht dieses Board nämlich in der Portliste mehrfach auf.

MERKE: PROBLEME BEIM HOCHLADEN

Auch hier macht die Technik keine Ausnahme. Irgendwann stößt man immer auf Fehler, deren Ursachen vielfältig sind. Ist die Ursache nicht gleich ersichtlich, hilft ein Blick auf die offizielle Seite:

https://www.arduino.cc/en/Guide/Troubleshooting#upload

Serieller Monitor als Debugging-Hilfe

Wir sehen bis hierhin nur das Blinken der LED. Doch wo steckt der Schriftzug »Hello World!«? Und es stellt sich generell die Frage: Wie kann man ohne LEDs oder sonstiges direktes physikalisches Feedback sehen, was der Arduino momentan anstellt oder welche Stelle im Programmcode er aktuell abarbeitet? Da hilft unter anderem der serielle Monitor weiter. Es zeigt sich erneut, wie einsteigerfreundlich die Arduino-Welt sein kann.

Wir benötigen weder zusätzliche Kabelverbindungen an Pins noch muss zusätzliche Software bzw. eine Library installiert werden. Für das einfache Debuggen reicht bereits aus, dass der Arduino per USB mit dem Computer verbunden ist und dass die Arduino-IDE die Serial-Bibliothek und den seriellen Monitor mit sich bringt. Auf den Programmcode gehen wir später ein. Den seriellen Monitor öffnen wir über *Werkzeuge/Serieller Monitor* oder über das entsprechende Tastaturkürzel (Windows: Strg + ⇧ + M).

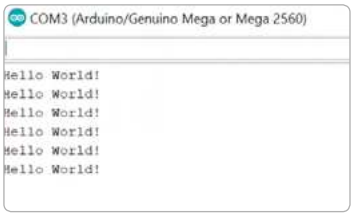


Bild 3.18: Der serielle Monitor begrüßt mit: »Hello World!«.

Nun sehen wir, wie in periodischen Zeitabständen jeweils eine Zeile mit dem Schriftzug »Hello World!« im Monitor erscheint. Sollte das nicht der Fall sein, hat das hauptsächlich zwei Ursachen: Sollten mehrere Arduinos über USB mit dem Rechner verbunden sein, müssen wir erneut über *Werkzeuge/Port* die entsprechende Verbindung auswählen. Hier sehen wir auch, wenn keine Verbindung existiert. Der zweite Punkt betrifft die Übertragungsrate. Im unteren Bereich des neu erschienenen Fensters können wir in einem Drop-down-Menü die Übertragungsrate in Baud auswählen. Wenn sie nicht mit dem Arduino übereinstimmt, sehen wir nichts. Wir schauen kurz im Programmcode nach und erkennen, dass der Arduino mit 9.600 Baud überträgt. Dementsprechend passen wir den Wert im seriellen Monitor an.

Alternative Programmierung und Bootloader installieren

Die Programmierung eines Arduino-Boards erfolgt in den meisten Fällen mittels Bootloader. Doch wie funktioniert die Übertragung des Sketchs in den anderen Fällen, und was machen wir, wenn gar kein Bootloader vorinstalliert ist?

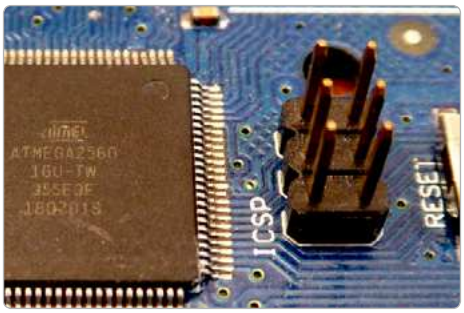


Bild 3.19: Der ICSP-Header ist unser Tor in eine Bootloader-lose Welt.

Uns bleibt eine weitere Möglichkeit, die glücklicherweise von der Arduino-IDE unterstützt wird. Es gäbe sogar einen Grund, freiwillig auf einen Bootloader zu verzichten und nach alternativen Programmiermöglichkeiten Ausschau zu halten: Der Bootloader nimmt Platz weg. Es mag zwar nur wenig Platz sein, dennoch könnte man bei fortgeschrittenen Projekten auf die Idee kommen, die vorhandenen Ressourcen zu optimieren und Speicherplatz einzusparen, wo es nur geht. Wenn auf dem Board also kein Bootloader installiert ist, kommen die bereits erwähnten ISP- oder ICSP-Schnittstellen ins Spiel. Davon kann es unter Umständen zwei geben, deshalb müssen wir vorsichtig sein. Auf einigen Boards, wie dem Arduino Mega oder dem Uno, gibt es zwei Mikrocontroller mit jeweils eigenem ICSP-Header.

Bild 3.20: Der USB-Tiny-ISP-Programmer programmiert unseren Arduino ganz ohne Bootloader.

Zuständig für die Übersetzung von USB zu seriell (RX, TX) ist der kleinere ATmega16U2 oder der ATmega8U2. Wenn er dieser nicht fehlerhaft arbeitet, fassen wir ihn auch nicht an. Für den Bootloader interessiert uns das Herzstück des Boards, also beim Arduino Mega der ATmega2560 oder beim Uno der ATmega238. An den ICSP-Header schließen wir mit einem IDC-Flachbandkabel einen externen Programmer an. An externen Programmern gibt es eine

gewisse Auswahl. Deshalb ist es beim Kauf wichtig, vorher einen Blick in die Liste der unterstützten Geräte zu werfen, sich vom Verkaufspersonal beraten zu lassen oder beim Kauf im Netz Kundenrezensionen zu durchforsten. Damit kann man sich eine Menge Arbeit ersparen.

Arbeiten mit dem USB-Tiny-ISP-Programmer

Besonders Neulinge verzweifeln an inkompatiblen Geräten oder an der Einrichtung der entsprechenden Treiber. Im Beispiel verwenden wir einen *USB Tiny ISP* und wählen ihn in der IDE unter *Werkzeuge/Programmer:/USBtinyISP* aus.

Da es sich um einen häufig verwendeten Programmer handelt, findet man den entsprechenden Treiber auch ziemlich schnell durch eine kurze Internetrecherche. Der Programmer wird über USB an den PC angeschlossen. Für die Verbindung zum Arduino-Board stellt der Programmer entweder einen sechspoligen, einen zehnpoligen oder sogar beide ISP-Header zur Verfügung. Der Arduino Mega im Beispiel benötigt ein sechspoliges Kabel. Für abweichende Programmer existieren Kabeladapter.

Das Kabel wird folgendermaßen auf den ISP-Header gesteckt: Pin #1 sollte am Stecker mit einem Pfeil markiert sein. Häufig ist das entsprechende Kabel zu Pin #1 im Kabelverbund rot gefärbt. Auf der anderen Seite ist bei einem originalen Arduino-Board an Pin #1 ein kleiner weißer Punkt auf das Board gedruckt. Bei anderen Boards kann wiederum ein Blick ins Datenblatt Aufschluss geben.

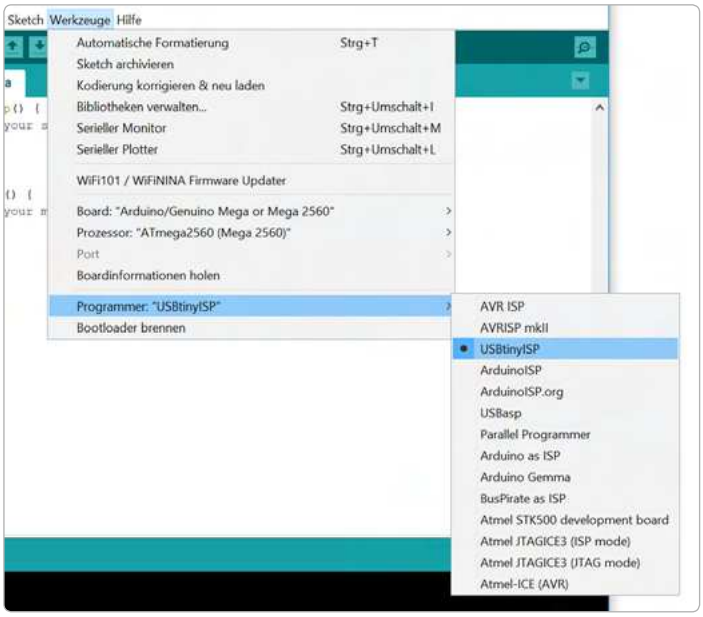


Bild 3.21: In der IDE wird der Programmer zum Hochladen ausgewählt.

Wenn alles wie beschrieben miteinander verbunden und der entsprechende Programmer in der Arduino-IDE ausgewählt ist, müssen wir nur den hochzuladenden Sketch öffnen und das zu beschreibende Board auswählen.

Der weitere Workflow unterscheidet sich nur noch im Hochladen, denn jetzt gehen wir über *Sketch/Hochladen mit Programmer* (Tastenkürzel Strg + ⇧ + U) und sehen im Fall des Erfolgs die entsprechende Meldung. Man sollte sich allerdings vorher der Tatsache bewusst sein, dass der Bootloader nach dem Hochladen über einen Programmer auf dem Mikrocontroller überschrieben wird. Es ist somit nicht mehr möglich, das Arduino-Board auf dem Standardweg über die USB-Verbindung zu programmieren. Allerdings kann man den Bootloader mittels Programmer relativ schnell wieder auf den Mikrocontroller aufspielen.

WARNUNG: HOCHLADEN MIT PROGRAMMER

Beim Hochladen mit dem Programmer wird der Bootloader auf dem Mikrocontroller überschrieben!

So könnte man nun problemlos weiterarbeiten. Insbesondere wenn das Board fest in einer Bastelei verbaut ist und selten wieder angefasst wird, benötigt man unter Umständen keinen Bootloader mehr. Wenn das Board allerdings beispielsweise für schnelllebige Experimente, Prototyping oder zum Debugging genutzt wird, ist es deutlich bequemer, das Board direkt mit einem USB-Kabel am Rechner anzuschließen, über diese Schnittstelle abwechselnd Sketche hochzuladen und die Ausgaben des seriellen Monitors zu analysieren. Deshalb schauen wir uns jetzt noch an, wie wir auf einem überschriebenen oder leeren Mikrocontroller wieder einen Bootloader installieren können. Hierfür werden Arduino-Board und Programmer wie im vorherigen Beispiel miteinander verbunden.

Die Einstellungen in der IDE sind vorerst ebenfalls dieselben: Informationen zu Board und Prozessor über *Werkzeuge* einstellen und den verwendeten Programmer ebendort auswählen. Das war es auch schon, denn jetzt müssen wir nur noch den Bootloader hochladen und über *Werkzeuge/Bootloader brennen* genau das tun. Nun ist es wieder möglich, das Arduino-Board über die USB-Schnittstelle zu programmieren. Zu guter Letzt sei noch erwähnt, dass man auch einen Arduino als Programmer nutzen kann. So beschreibt ein Gerät das andere, und man spart sich die Anschaffung eines richtigen Programmers. Inzwischen sind diese jedoch nicht mehr so teuer in der Anschaffung, der Kostenfaktor ist also kein Argument mehr für diese Vorgehensweise. Wer bereits zwei Arduino-Boards besitzt, kann sich auch mit dieser Variante der Programmierung auseinandersetzen. Im Netz gibt es genügend Anleitungen.

Arduino und MIDI – eine spannende Verbindung

Aus einem Arduino ein MIDI-fähiges Gerät zu bauen ist nicht nur möglich, es geht sogar relativ einfach. Die Plattform bringt nahezu alle Voraussetzungen mit, die für den Bau eines eigenen MIDI-Geräts notwendig sind, und es gibt auch im Internet viele Anleitungen und selbst gebaute MIDI-Instrumente von genialen Tüftlern. Arduinos eignen sich aus verschiedenen Gründen sehr gut für unser Vorhaben.

Zum einen haben sie genug Speicher und Rechenleistung für die Übertragung komplexer MIDI-Befehle, sind aber nicht so komplex wie beispielsweise ein Raspberry Pi und benötigen kein Betriebssystem. Sie funktionieren praktisch »out of the box«. Es gibt genügend Schnittstellen, um die verschiedensten Ein- und Ausgabegeräte zu realisieren. Wenn das nicht ausreichen sollte, kann man den Arduino auch noch um Schnittstellen erweitern. Die Programmierung ist vergleichsweise simpel und lässt doch kaum Wünsche offen. Die Umsetzung von MIDI-Befehlen ist sowohl über DIN-Buchsen als auch via USB (zumindest bei den neueren Geräten) möglich. Und schließlich das Killerargument: Arduinos sind vergleichsweise günstig zu haben. Es braucht also nicht viel Geld, um einen einfachen eigenen MIDI-Controller zu realisieren. Es gibt Anleitungen zum MIDI-Anschluss und zur Übertragung von MIDI-Noten. Die Arduino-IDE beinhaltet auch bereits vorgefertigte Bibliotheken, um die Arbeit mit MIDI-Daten zu vereinfachen. Viel wichtiger wird für uns die Frage sein, was man genau umsetzen möchte. Wie soll unser MIDI-Controller aussehen?

Wir werden uns auf den kommenden Seiten mit der Frage auseinandersetzen, wie wir unseren eigenen MIDI-Controller nach Mackie-HUI-Vorbild gestalten können. Wir werden zeigen, wie einfach man simple elektronische Bauteile mit dem Arduino verbinden kann, um darüber später MIDI-Befehle zu senden. Ferner werden wir zeigen, wie man MIDI-Daten übertragen kann. Dabei werden wir sowohl auf die klassische Variante mittels DIN-Steckern eingehen als auch auf die Möglichkeiten, den Transport über USB zu realisieren.

Bill Gates wird folgender Spruch nachgesagt: *»Your most unhappy customers are your greatest source of learning.«*

Wir hoffen, dass dieses Buch für so viele Leser wie nur möglich eine Bereicherung sein wird und sie Spaß am Nachbau des hier vorgestellten Projekts hatten bzw. haben. Die Arbeit selbst hat sicherlich manchmal öde Momente – wir haben es zumindest ab und an so erlebt –, doch der Moment, in dem die ersten LEDs leuchten und Taster virtuelle Knöpfe am Computer drücken, ist es allemal wert. Wir wünschen uns, mehr für die Glücksgefühle (und weniger für die frustrierenden Momente) beim Nachbauen verantwortlich zu sein. Ferner freuen wir uns sehr über Anregungen oder auch Kritik, die zur Verbesserung dieses Werks beitragen können, und wünschen allen Lesern tolle Momente in der faszinierenden Arduino-Welt.

Tobias Naumann und Johannes Wronka

Symbole

A

B

C

D

U

V

W

MACH'S EINFACH

MUSIK MIT DEM ARDUINO® UND MIDI

Musiker aufgepasst!
Spielend leicht zum selbstgebauten MIDI-Mischpult

Einen MIDI-Controller selbst bauen, das große Mischpult des Musikprogramms endlich ohne Maus und Tastatur steuern – das wünscht sich so mancher Musiker und Elektronikbegeisterte. Mit den Open-Source-Mikro-Controllern der Arduino®-Familie geht das spielend leicht und macht eine Menge Spaß.

Dieses Buch bietet den perfekten Einstieg in die Arbeit mit MIDI-Controllern auf Arduino®-Basis, egal ob Sie Bastelneuling sind oder bereits schwungvoll mit dem Lötkolben umgehen.

Anhand praktischer Beispiele und vieler Grafi ken lernen Sie die Grundlagen der Elektronik und der Arduino®-Programmierung kennen. Vom Löten bis zur Verkabelung fi nden Sie in dieser Do-it-yourself-Anleitung alle erforderlichen Schritte detailliert beschrieben: Das Ziel: Ihr erstes selbst gebautes MIDI-Mischpult.

IN DIESEM BUCH GEHT ES UM:

- Das eigene Homerecording-Studio
- **MIDI:** Anschlüsse und Verkabelung
- **MIDI:** Instrumente und Controller
- **Arduino®-IDE:** Programmierung ganz einfach
- Schaltkreise ohne aufwendige Lötarbeit
- Übersetzung von USB zu seriell
- Den Programmcode kompilieren
- Vom Blinken bis zur seriellen Ausgabe
- **Taster und Schalter:** Es gibt viel zu drücken
- Wenn ein Arduino®-Board zum anderen spricht
- **Kommunikation:** MIDI-Signale übertragen
- **Mackie HUI:** Vorlage für einen Controller
- MIDI-Einrichtung im Musikprogramm
- USB-MIDI mit dem Arduino® Leonardo
- Bau eines MIDI-Controllers von A-Z
- **Bohren und sägen:** Das Gehäuse für den Einbau

FRANZIS

Besuchen Sie unsere Webseite:
www.franzis.de