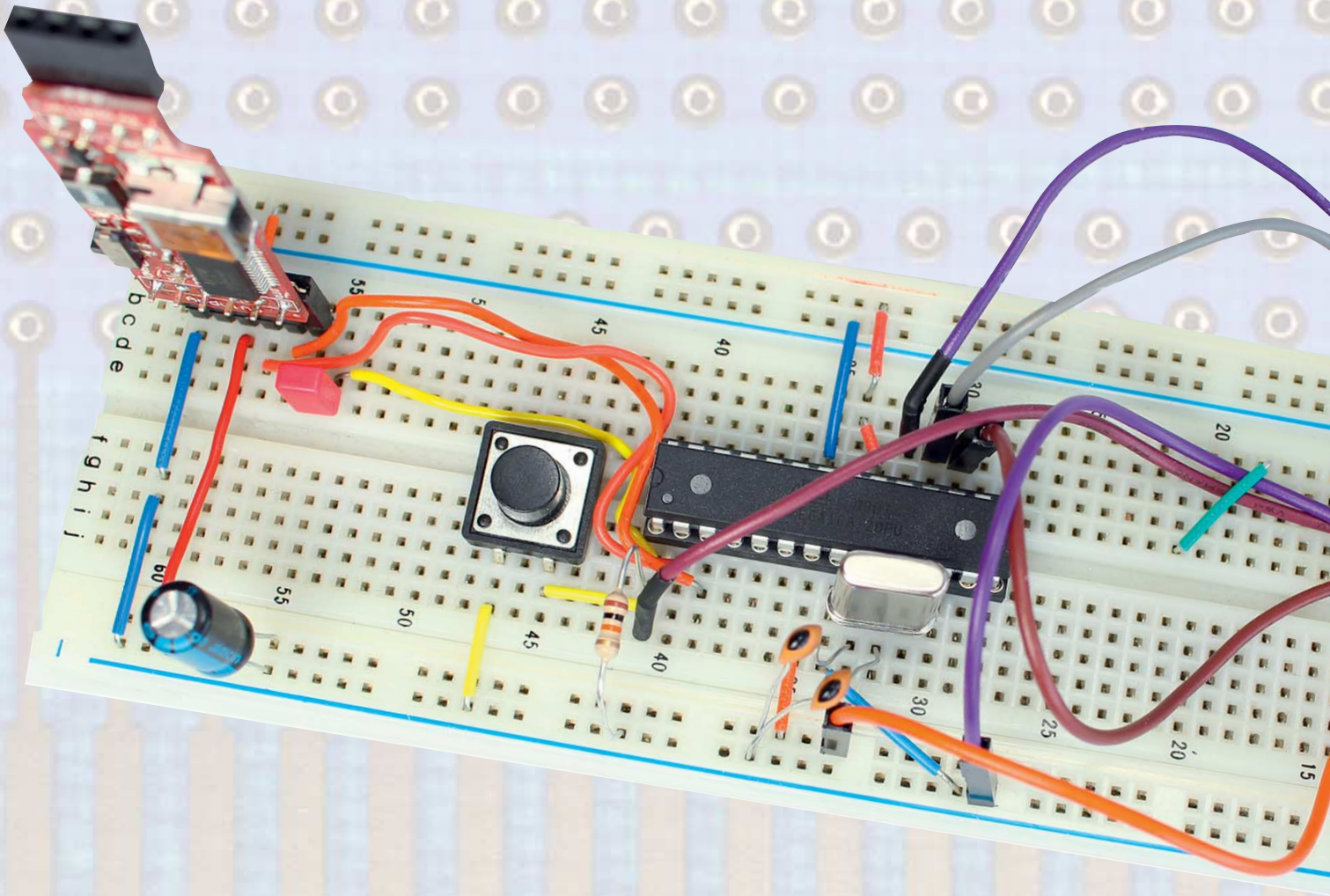
# Arduino im Eigenbau

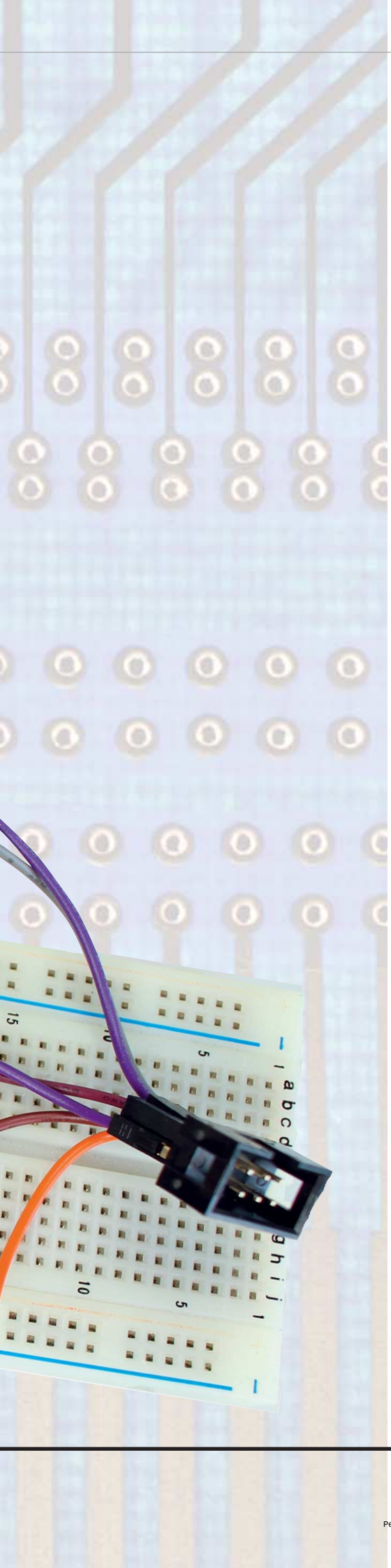
**Mit Arduino-Boards kann man zahllose Elektronik-Projekte umsetzen. Doch warum jedes Mal 20 Euro für einen Arduino verschwenden, wenn es auch viel günstiger geht? Wir zeigen Ihnen, wie Sie einen Arduino mit wenigen Bauteilen nachbauen.**

von Artur Kriening und Ralf Stoffels





© Copyright by Maker Media GmbH
Persönliches PDF für alle Leser des Make Magazins aus 30625 Hannover



Dank ihrer flexiblen Hardware, der gut dokumentierten Software und der leicht zu bedienenden Bibliotheken erleichtern Arduinos die Entwicklung eigener Schaltungen erheblich. Zudem benötigt man für die Arbeit mit ihnen nicht einmal einen zusätzlichen Programmer. Doch bemerkt man schnell, dass Arduino-Boards für den gewünschten Einsatz meist üppig überdimensioniert sind. Und ist man mit einem Projekt fertig und will das nächste aufbauen, stellt sich die Frage: Jetzt das Alte wieder abbauen oder erneut mindestens 20 Euro für den Controller investieren?

Um dieses Problem zu lösen, kann man mit wenig Aufwand eine ATmega-Minimalschaltung bauen, die Arduino-kompatibel ist und sich leicht mit den anderen Bauteilen wie Porterweiterungs-ICs, MOSFETS oder ähnlichem in die eigene Lochrasterplatine integrieren lässt. Die Schlüsselkomponente für den Nachbau ist der 8-Bit-ATmega Prozessor, der sich in verschiedenen Ausführungen auf gängingen Arduino-Boards befindet und den man mit der frei erhältlichen Arduino-IDE programmieren kann.

Die Arduino-IDE verfügt bereits über eine große Anzahl an Bibliotheken sowie Beispielcode und stellt zudem einen leicht zu bedienenden Programmer für die Hardware bereit. Der Clou an den ATmega-Prozessoren ist, dass sie mit Hilfe eines Bootloaders über eine serielle Schnittstelle mit der Arduino-IDE kommunizieren können. So kann man die eigenen Sketches, auch ohne zusätzliche Hardware wie einen ISP-Programmer, hochladen. Wie dies funktioniert, zeigen wir Ihnen anhand eines Beispiels mit einem ATmega328P-Prozessor, der zum Beispiel im Arduino Uno verbaut ist.

## Das ATmega-Einmaleins

Der ATmega-Prozessor verfügt zwar ab Werk über eine serielle UART-Schnittstelle (Universal Asynchronus Receiver and Transmitter) – über diese ist er standardmäßig jedoch nicht in der Lage, Befehle oder Sketches entgegenzunehmen. Damit unser ATmega wie ein klassischer Arduino über die serielle Schnittstelle programmierbar wird, benötigt er einen Bootloader. Der Bootloader ermöglicht es, serielle Datenbytes an den ATmega zu senden und von ihm zu empfangen. Er wird in einen vordefinierten Speicherbereich auf dem ATmega geladen. Sobald der ATmega mit Strom versorgt wird, zeigt der Program Counter auf den Anfang des Bootloaders und startet diesen. Der Bootloader führt einen Code aus, welcher innerhalb eines vordefinierten Zeitfensters von wenigen Sekunden Befehle über die serielle Schnittstelle erwartet.

Erhält er den Befehl zum Schreiben, fängt er an, die empfangenen Daten in den Flashspeicher ab Adresse 0x00 zu schreiben. Falls er keine Anweisungen durch die serielle Schnittstelle erhält, setzt er den Program Counter auf die Adresse 0x00 und führt den dort gespeicherten Programmcode aus. Damit der ATmega erkennt, dass ein Bootloader installiert ist, muss die Fuse BOOTRST aktiv sein und die Größe des Bootloaders mit den Fuses BOOTSZ0 und BOOTSZ1 definiert werden. Weitere Hinweise zu Fuses finden Sie im Kasten „Fuses mit Avrdude“ auf Seite 99. Wie der Bootloader mit wenig Aufwand installiert wird, erklären wir im nächsten Abschnitt.

**Links und Foren**
*make-magazin.de/xcv3*

## Kurzinfo

**Zeitaufwand:**
2 Stunden

**Kosten:**
ab 20 Euro

**Programmieren:**
Arduino-IDE

**Schwierigkeitsgrad**
leicht – schwer

## Material

- 1 × ATmega328P
- 1 × 16 MHz Quarzoszillator
- 2 × 22pF Kondensatoren
- 1 × 100nF Kondensator
- 1 × Elko, 10µF
- 1 × 10 kOhm Widerstand
- 1 × FTDI-USB-Seriell Breakout
- 1 × ISP-Programmer oder Arduino
- Kabel
- Breadboard

© Copyright by Maker Media GmbH
Persönliches PDF für alle Leser des Make Magazins aus 30625 Hannover

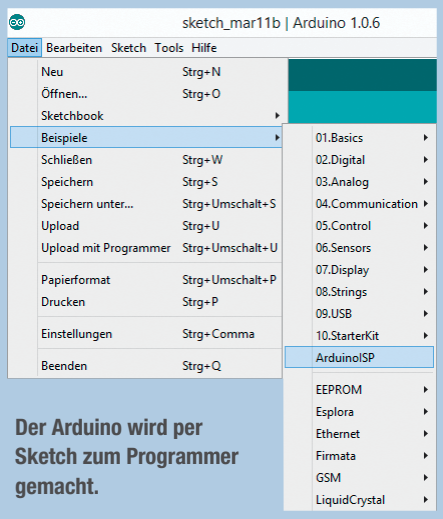
# ISP-PROGRAMMING

Um den Bootloader zu installieren, muss man ihn einmalig in den Flash schreiben. Atmel hat hierfür eine ISP-Schnittstelle entwickelt, die es ermöglicht, einen nackten ATmega-Prozessor über das Serial Peripheral Inferface (SPI) zu programmieren. Da eine serielle RS232-Schnittstelle ihre Datenbits asynchron überträgt, ist sie zum synchronen SPI-Interface inkompatibel und benötigt einen Programmer, der das Signal übersetzt. Hierzu ersteht man entweder einen ISP-Programmer für ATmega Mikrocontroller zu Preisen zwischen 10 und 30 Euro oder nutzt einen Arduino, den man mit wenig Mühe als ISP-Programmer zweckentfremden kann.

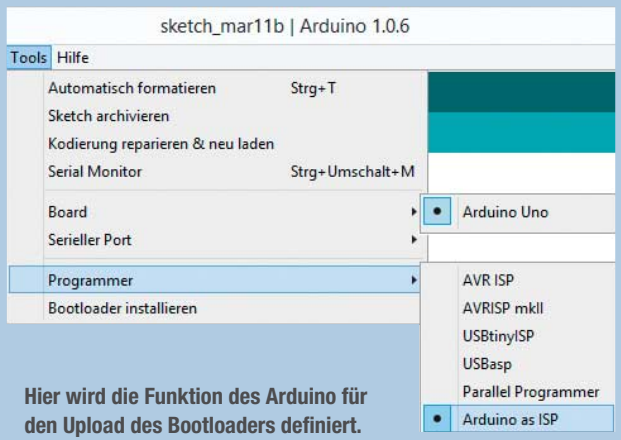
Zuerst muss man hierfür – falls man nicht schon über sie verfügt – die aktuelle Arduino-Entwicklungsumgebung von www.arduino.cc herunterladen und sie installieren. Die Arduino-IDE stellt alle benötigten Werkzeuge und Dateien für die Programmierung des ATmega bereit. Wir setzen sie sowohl für den Upload des Bootloaders als auch die Programmierung der Sketches über die serielle Schnittstelle ein.

Unser erster Schritt innerhalb der Software ist es, den ISP-Sketch, der in der Arduino-IDE im Menü Datei unter der Option Beispiele zur Verfügung gestellt wird, auf unseren Arduino zu laden. So wird der Mikrocontroller zum ISP-Programmer umfunktioniert. Danach kann man den Programmer auf „Arduino as ISP" festlegen, um später den Bootloader über ISP hochladen zu können. Außerdem benötigt die Arduino-IDE zum Upload des Bootloaders Informationen über den zu beschreibenden Mikroprozessor, der in unserem Fall ein ATmega328P ist und sich auf den gängigen Arduino UNOs wiederfindet. Daher stellen wir „Arduino UNO" unter dem Menüpunkt „Board" ein.

Um den Bootloader auf den ATmega zu laden, muss man jetzt nur noch im Menü „Tools" die Option „Bootloader installieren" anklicken. Dabei muss man vor dem Klick auf „Upload" die Shift-Taste drücken und halten. So erkennt die Arduino-IDE, dass der Arduino als Programmer genutzt werden soll. Im Hintergrund besteht das Installieren des Bootloaders in der Arduino-IDE aus folgenden Schritten: Zuerst setzt das Programm Unlock-Bits auf dem ATmega, damit der Speicherbereich des Bootloaders freigegeben wird. Danach setzt es die High, Low und Extended Fuses und lädt im Anschluss den Bootloader hoch. Zu guter Letzt schützt es den Speicherbereich des Bootloaders durch das Setzen eines Lock-Bits vor versehentlichem späteren Überschreiben. Von all dem bemerkt man nichts, erkennt aber an der Meldung in der Statusleiste, ob der Schreibvorgang erfolgreich war. Jetzt verbindet man die Datenleitungen MOSI, MISO, SCK und Reset des ATmega Prozessors mit den Pins 10-13 des Arduino-Boards (siehe Bild). Die Anschlüsse VCC und GND des ATmega-Prozessors können an den 5V-Ausgang und Masse des Arduinos angeschlossen werden. Zudem legt man einen Kondensator (Elko mit 10 µF) zwischen GND und Reset des Arduinos – er fängt den Reset-Impuls des USB-Seriell-Wandlers auf dem Arduino ab. Dieser Reset-Impuls würde nämlich im Normalfall dazu führen, dass der Arduino zuerst seinen eigenen Bootloader zur Neuprogrammierung startet, anstatt den ISP-Sketch zu starten. Das führt aber beim Programmieren anderer ATmegas zu Problemen.

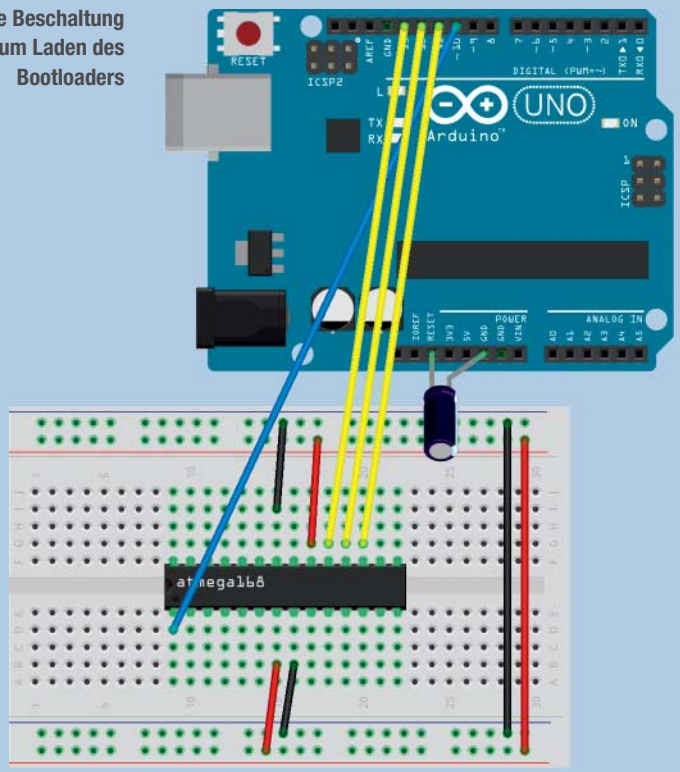



Der Arduino wird per Sketch zum Programmer gemacht.




Hier wird die Funktion des Arduino für den Upload des Bootloaders definiert.

Die Beschaltung zum Laden des Bootloaders

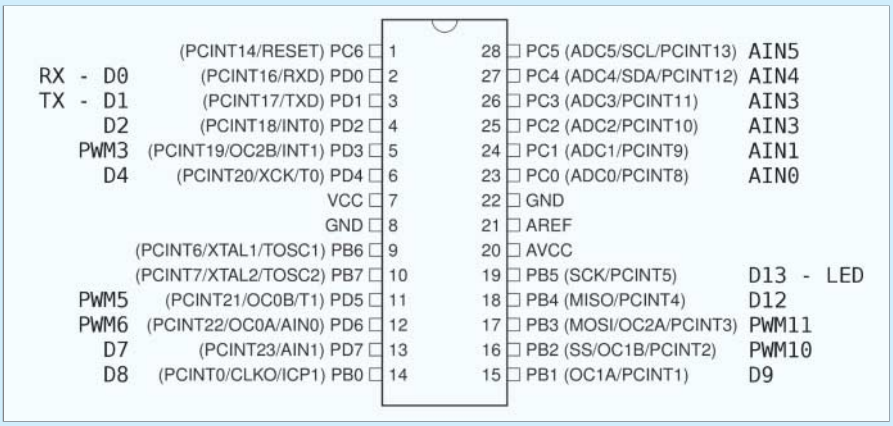




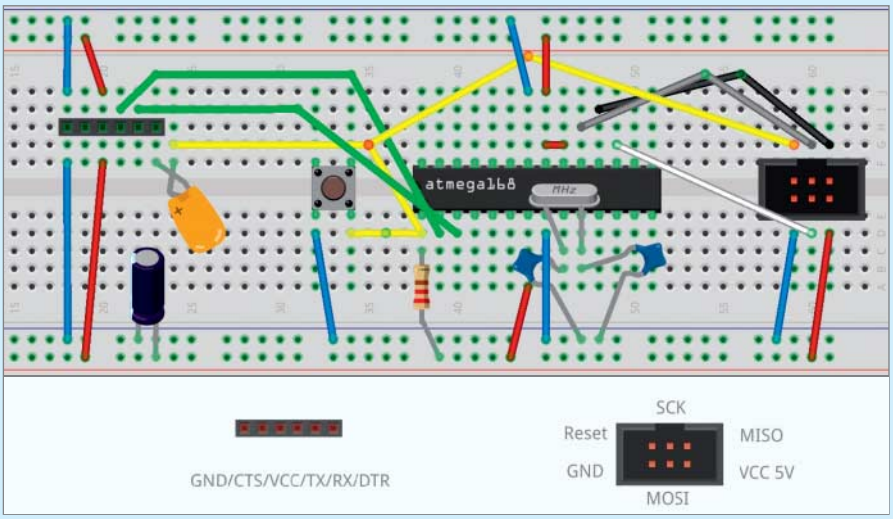

© Copyright by Maker Media GmbH
Persönliches PDF für alle Leser des Make Magazins aus 30625 Hannover

# AUFBAU

Nachdem der Bootloader installiert ist, können wir damit beginnen, den ATmega in einen Arduino-Nachbau zu verwandeln. Die Fuses sind nun so gesetzt, dass der ATmega einen externen 16-MHz-Quarz-Ozillator erwartet. Den schließen wir an XTAL1 und XTAL2 an und legen ihn mit zwei 22-pF-Kondensatoren auf Masse. Am Reset-Eingang schließen wir einen 10-kOhm-Pull-Up-Widerstand sowie einen optionalen Reset-Taster an. Auf der linken Seite befindet sich ein 6-Pin-Anschluss für den Seriell-USB-Wandler mit einem 100-nF-Kondensator an DTR, den wir Dank des Bootloaders zum Upload der Sketches verwenden können. DTR wird in diesem Fall als Reset-Trigger genutzt. Wer nicht auf einen ISP-Anschluss verzichten möchte, kann diesen ebenfalls mit unterbringen.




**Die Übersicht der Pinbelegung des ATmega328P**




**Unten links ist der Anschluss für den Seriell-USB-Wandler, unten rechts der ISP-Anschluss.**

Anzeige

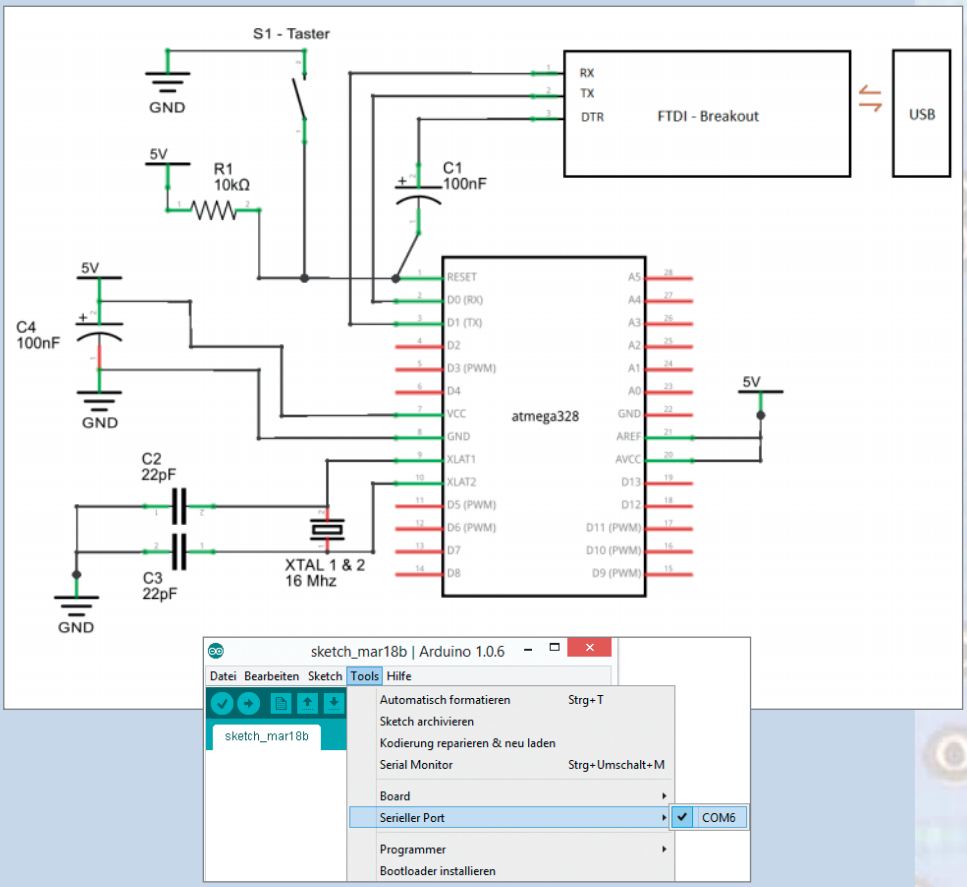
© Copyright by Maker Media GmbH
Persönliches PDF für alle Leser des Make Magazins aus 30625 Hannover

# SERIELLE PROGRAMMIERUNG

Da unser ATmega nun in der Lage ist, Befehle über die serielle Schnittstelle entgegenzunehmen, können wir die Sketches mit Hilfe eines USB-Seriell-Wandlers hochladen. Es gibt ab circa 5 Euro eine Vielzahl günstiger USB-TTL-Breakboards. Empfehlenswert sind solche, die mit einem FTDI-Chip ausgestattet sind und dadurch keine zusätzlichen Treiber benötigen. Um sie mit dem ATmega zu verbinden, werden lediglich drei Datenleitungen benötigt: RX, TX und DTR.

Hier ist zu beachten, dass RX und TX über Kreuz (RX→TX und TX→RX) angeschlossen werden müssen und der DTR-Ausgang des FTDI-Breakoutboards über einen 100-nF-Kondensator an den Reset-Eingang des ATmegas gelegt wird.

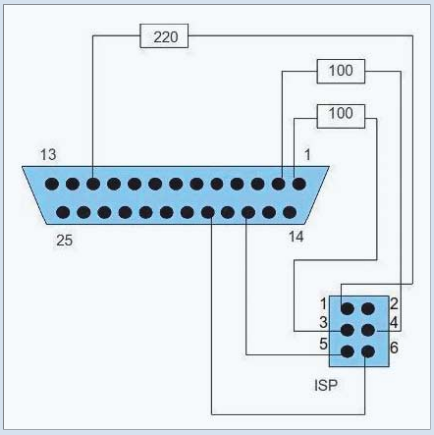
Sobald das FTDI-Breakoutboard über USB mit dem PC verbunden ist, wird ihm ein COM-Port zugewiesen, den man in der Arduino-IDE auswählt. Jetzt kann der eigene Sketch hochgeladen werden. Im Anschluss startet er automatisch.




# ALTERNATIVMODELL PARALLELPORT (DB25)

Wer noch über einen einen PC mit klassischer paralleler Druckerschnittstelle, also einen alten Parallelport verfügt, kann als Alternative zu unserem Vorschlage eine minimalistische Lösung mit drei Widerständen und einem DB25 Stecker nutzen. Der ATmega muss dabei mit dem ISP Kabel an den Parallelport (Drucker) des PCs angeschlossen sein. Das Kabel zwischen PC und dieser Platine sollte nicht länger als ein Meter sein, um eine sichere Datenübertragung zu gewährleisten.

Während des Programmiervorgangs muss der ATmega328P hier mit 5V versorgt werden. Bei einem käuflichen ISP Programmer geschieht das über die ISP-Schnittstelle (Pin 2), in dieser Minimalschaltung muss dies ein Steckernetzteil tun, da die Druckerschnittstelle keinen 5V-Ausgang hat. Nutzt man einen Windows PC, muss man den Giveio-Parallelport-Treiber installieren, wofür es eine Batch-Datei im Installationsverzeichnis von Avrdude gibt : install_giveio.bat.




Ein Beispiel mit einem parallelen Dapa-Programmer:

```
Avrdude -i 40 -c dapa -p m328p -F
-U lfuse:w:0xFF:m -U hfuse:w:0xDD:m
```

| ISP | ATMega | Gehäuse Pin (DIP 28) |
|---|---|---|
| 1 | MISO2 | 18 |
| 2 | +5V | 7 |
| 3 | SCK2 | 19 |
| 4 | MOSI2 | 17 |
| 5 | RESET2 | 24 |
| 6 | GND | 8 |

**Links: Die hier gezeigte Beschaltung ist ein sogenannter Dapa-Programmer (Direct AVR parallel Access).**



© Copyright by Maker Media GmbH
Persönliches PDF für alle Leser des Make Magazins aus 30625 Hannover

# FUSES MIT AVRDUDE

Man kann den ATmega-Prozessor an seine Arbeitsumgebung anpassen und so zum Beispiel die Taktquelle für die Datenübertragung und die angelegten Spannungen definieren. Das geschieht durch das Setzen sogenannter Fuses. Dabei handelt es sich um programmierbare Registerbits, die man mit dem Open-Source-Tool Avrdude bearbeitet. Durch das Brennen des Bootloaders über die Arduino-IDE werden zwar standardmäßig die wichtigen Fuses gesetzt, können aber für die eine oder andere Schaltung noch etwas Feintuning vertragen. Im Auslieferungszustand des ATmega steht beispielsweise 0xDF in dem hfuse Register geschrieben, was bedeutet, dass die Brown out Detection deaktiviert ist und der ATmega sehr empfindlich auf Spannungseinbrüche reagiert. Dies kann man unverändert lassen oder die Brown out Detection mit dem Wert 0xDD einschalten, falls man wünscht, dass der ATmega sich erst bei einer Spannung von 2.7 V zurücksetzt.

Zur Ermittlung der richtigen Fuses kann man Programme wie den AVR8 Burn-O-Mat oder einen webbasierten Fuse Calculator auf http://www.engbedded.com/fusecalc/ oder http://eleccelerator.com/fusecalc/fusecalc.php nutzen. Achtung: Die Fuses werden je nach verwendetem Atmel-Chiptyp unterschiedlich berechnet. Um die Fuses auf den ATmega zu brennen, muss dieser mit dem ISP-Programmer verbunden sein. Im Anschluss rufen wir das Kommandozeilentool auf, wechseln in das Verzeichnis in dem sich Avrdude.exe befindet und rufen dies mit folgenden Parametern auf:

```
Avrdude -P com5 -b 19200 -c avrisp
-p m328p -v -e -U hfuse:w:0xDD:m
-U lfuse:w:0xFF:m
```

## ZUSÄTZLICHER I/O-PORT FÜR ATTINY

Ist man in Besitz eines achtbeinigen ATtinys und benötigt einen weiteren I/O-Port, kann man die RSTDISBL-Fuse aktivieren, um diesen freizugeben. Dies hat aber den Nachteil, dass man den Chip nicht mehr über ISP flashen kann. Abhilfe schafft in diesem Falle das sogenannte High Voltage Programming (HPV), das man in Verbindung mit einem STK500/600-Programmer zur Rettung verwenden kann.

Avrdude – Das Werkzeug zum Brennen der Programme und Fuses befindet sich im Hardware-Ordner der Daten der Arduino-IDE. Ganz konkret finden Sie es unter:

Linux (sudo): /usr/share/arduino/hardware/tools/Avrdude

Windows: c:\programme\arduino\hardware\tools\avr\bin\Avrdude.exe

Weitere Informatione zu Fuses finden Sie im Artikel „Arduino Uno als In-System-Programmer“ von Daniel Bachfeld. Sie finden ihn online unter den Links zu diesem Artikel und in der c’t Hacks 1/14 auf Seite 138. *—esk*

### Diese Optionen bedeuten

| Funktion | Bedeutung |
|---|---|
| -c | Auswahl des Programmers – avrisp |
| -p | Verwendeter Prozessor – m328p |
| -P | den richtigen Port des Programmers definieren – Comport (bei Programmern wie dem AVRisp mkII wird dieser auch mit „usb“ definiert) |
| -b | Baudrate 19 200 Bit/s |
| -v | Verbose – Ausgabe aller Informationen über den Vorgang in der Kommandozeile |
| -e | löscht Speicherbereich im Zielprozessor (chip erase) |
| -U | spezifische Speicheroperation. w löst einen Schreibvorgang aus. |

Anzeige



© Copyright by Maker Media GmbH
Persönliches PDF für alle Leser des Make Magazins aus 30625 Hannover