Keine Vorkenntnisse erforderlich!

Ihr Weg zum Arduino-Profi

c't Make:

Make:

ARDUINO SPECIAL

Grundlagen

- LEDs steuern
- Eingänge abfragen
- Spannungen messen
- Sensoren einlesen
- Motoren schalten
- Mikrofon anschließen
- Fernsehausgabe
- Töne erzeugen

4. Auflage

ANFÄNGER-PROJEKTE

Batterietester • Lärmampel
Temperaturanzeige • Synthesizer

Hardware & Software einfach erklärt

2/2017

www.make-magazin.de

Übersicht wichtiger Befehle und Funktionen

Struktur & Programmfluss

Grundsätzliche Sketch-Struktur

```
void setup() {
  // läuft nur einmal beim Start
}
void loop() {
  // wiederholt sich kontinuierlich
}
```

Kontroll-Strukturen

```
if (x < 5) { ... } else { ... }
while (x < 5) { ... }
for (int i = 0; i < 10; i++) { ... }
break;      // Schleife sofort beenden
continue;   // weiter mit nächster Iteration
switch (var) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  default:
    ...
}
return x;   // x ist der Rückgabewert
return;     // ohne Rückgabewert
```

Funktionen

```
<ret. type> <name>(<params>) { ... }
bspws. int double(int x) {return x*2;}
```

Operatoren

Allgemein

=	Zuweisung		
+	Addieren	-	Subtrahieren
*	Multiplizieren	/	Dividieren
%	Modulo		
==	gleich	!=	ungleich
<	kleiner als	>	größer als
<=	kleiner oder gleich		
>=	größer oder gleich		
&&	logisch UND	||	logisch ODER
!	NICHT		

Kombinierte Operatoren

++	Variable um 1 erhöhen
--	erniedrigen
+=	wie a = a + b
-=	wie a = a - b
*=	wie a = a * b
/=	wie a = a / b
&=	wie a = a & b
|=	wie a = a | b

Bitoperatoren

&	bitweises UND	|	bitweises ODER
^	bitweises ODER	~	bitweises NICHT
<<	um 1 Bit nach links schieben		
>>	um 1 Bit nach rechts schieben		

Pointer

&	Referenz: Adresse des Pointers
*	Dereferenzierer: Ziel des Pointers

Eingebaute Funktionen

Pin Input/Output

Digital I/O - Pins 0-13 A0-A5

```
pinMode(pin,
  [INPUT, OUTPUT, INPUT_PULLUP])
int digitalread(pin)
digitalWrite(pin, [HIGH, LOW])
```

Analog In - pins A0-A5

```
int analogRead(pin)
analogReference(
  [DEFAULT, INTERNAL, EXTERNAL])
```

PWM Out - pins 3 5 6 9 10 11

```
analogWrite(pin, value)
```

Advanced I/O

```
tone(pin, Freq_Hz)
tone(pin, Freq_Hz, Dauer_ms)
noTone(pin)
shiftOut(dataPin, clockPin,
  [MSBFIRST, LSBFIRST], WERT)
unsigned long pulseIn(pin,
  [HIGH, LOW])
```

Zeit

```
unsigned long millis()
  // läuft nach 50 Tagen über
unsigned long micros()
  // läuft nach 70 Minuten über
delay(msec)
delayMicroseconds(usec)
```

Mathe-Funktionen

```
min(x, y)   max(x, y)   abs(x)
sin(rad)    cos(rad)    tan(rad)
sqrt(x)     pow(Basis, Exponent)
constrain(x, Minl, Maxl)
map(val, fromL, fromH, toL, toH)
```

Zufallszahlen

```
randomSeed(seed)   // long oder int
long random(max)   // 0 bis max-1
long random(min, max)
```

Bits und Bytes

```
lowByte(x)   highByte(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn)   // bitn: 0=LSB 7=MSB
```

Konvertierung von Datentypen

```
char(Wert)   byte(Wert)
int(Wert)    word(Wert)
long(Wert)   float(Wert)
//int(Fließkommazahl) wandelt bspws
//Fließkommazahl in Integer-Wert um
```

Externe Interrupts

```
attachInterrupt(interrupt, func,
  [LOW, CHANGE, RISING, FALLING])
detachInterrupt(interrupt)
interrupts()
noInterrupts()
```

Variablen, Arrays und Daten

Datentypen

boolean	true	|	false
char	-128	-	127, 'a' '$' etc.
unsigned char	0	-	255
byte	0	-	255
int	-32768	-	32767
unsigned int	0	-	65535
word	0	-	65535
long	-2147483648	-	2147483647
unsigned long	0	-	4294967295
float	-3.4028e+38	-	3.4028e+38
double	wie float		
void	kein Rückgabewert/ unbestimmt		

Strings

```
char str1[8] =
  {'A','r','d','u','i','n','o','\0'};
  // manuelle \0 Null-Terminierung
char str2[8] =
  {'A','r','d','u','i','n','o'};
  // Compiler fügt Null-Terminierung hinzu
char str3[] = "Arduino";
char str4[8] = "Arduino";
```

Numerische Konstanten

123	dezimal
0b01111011	binär
0173	oktal (Basis 8)
0x7B	hexadezimal (Basis 16)
123U	erzwinge unsigned
123L	erzwinge long
123UL	erzwinge unsigned long
123.0	erzwinge floating point
1.23e6	1.23*10^6 = 1230000

Eigenschaften von Variablen

static	Variable bleibt über Aufrufe erhalten
volatile	Variable im RAM statt Register speichern
const	Konstante, nur lesbar
PROGMEM	im Flash ablegen

Arrays

```
int myPins[] = {2, 4, 8, 3, 6};
int myInts[6];     // Array mit 6 ints
myInts[0] = 42;  // Zuweisung der
                    ersten Stelle
myInts[6] = 12;  // Fehler, da 6. Variable
                    // den 5. Platz belegt
```

Bibliotheken

Serial – serielle Kommunikation mit PC oder via RX/TX

```
begin(long Baudrate)   // bis 115200
end()
int available()  // Anzahl empfangener Bytes
int read()       // -1 wenn keine Daten
int peek()       // Lesen ohne den Puffer
                    zu löschen
flush()   // Puffer löschen
print(data)     println(data)
write(byte)     write(char * string)
write(byte * data, size)
SerialEvent()   // wenn Daten fertig
```

SoftwareSerial.h – serielle Kommunikation über beliebige Pins

```
SoftwareSerial(rxPin, txPin)
begin(long Baudrate)   // bis 115200
listen()         // immer nur eine serielle
isListening()    // kann empfangen
read, peek, print, println, write
```

EEPROM.h – Zugriff auf nicht flüchtigen Speicher

```
byte read(Addr)
write(addr, byte)
EEPROM[index]
```

Servo.h – Servo-Motoren steuern

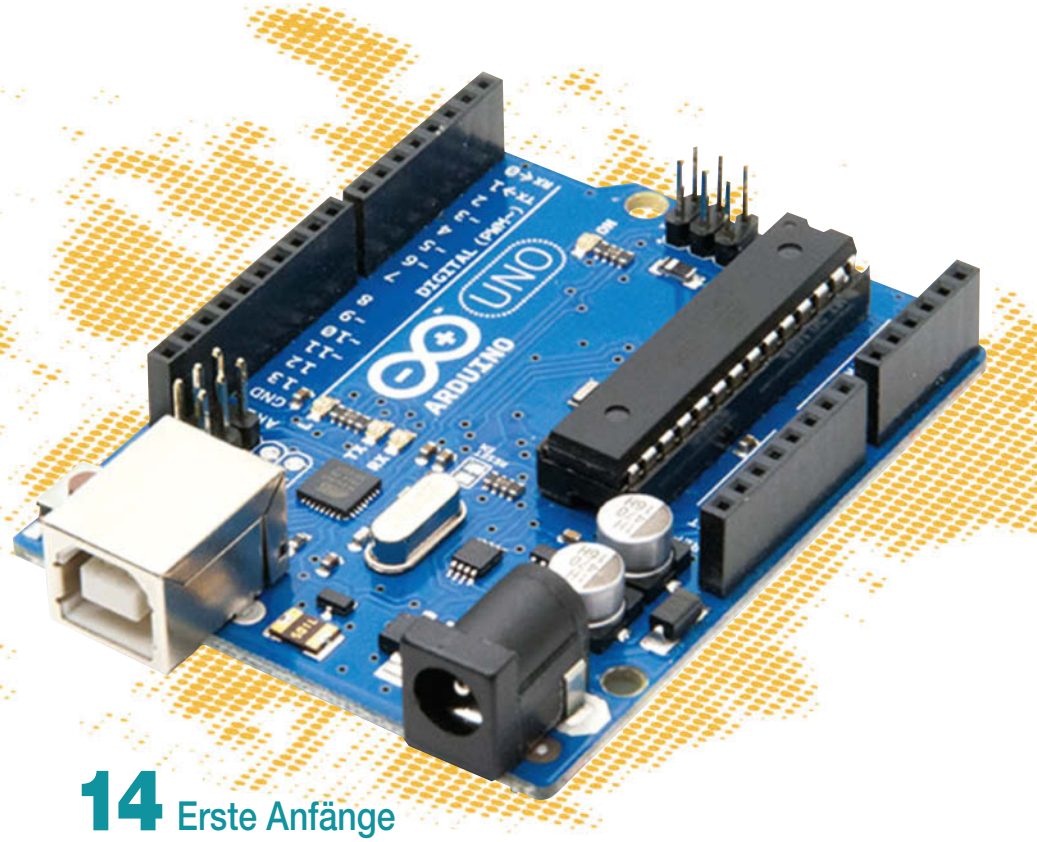
```
attach(pin, [min_uS, max_uS])
write(Winkel)   // 0 bis 180
writeMicroseconds(uS)
  // 1000-2000; 1500 ist Mitte
int read()      // 0 bis 180
bool attached()
detach()
```

Wire.h – I²C-Kommunikation

```
begin()
begin(Addr)   // Slave ansprechen
requestFrom(Addr, count)
beginTransmission(Addr)   // Schritt 1
send(byte)                // Schritt 2
send(char * string)
send(byte * data, size)
endTransmission()         // Schritt 3
int available()   // verfügbare Bytes
byte receive()    // nächstes Byte lesen
onReceive(handler)
onRequest(handler)
```

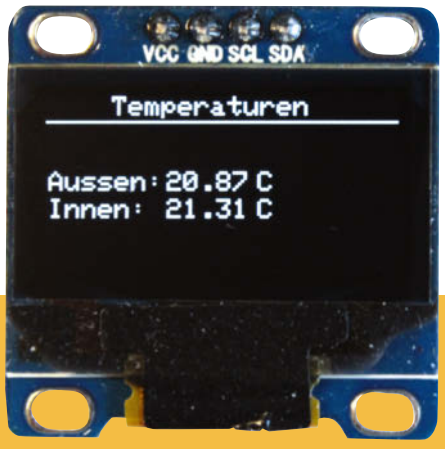
Weitere „Cheat Sheets“ finden Sie unter ▶ overapi.com

Inhalt



14 Erste Anfänge
Beispiele, die ohne Zusatzhardware sofort funktionieren.

29 Motoren steuern
Mit Motoren kommt Bewegung ins Spiel, um etwas anzutreiben oder einen Zeiger zu bewegen.



42 Mäusekino
Mit einem kleinen Display lässt sich der Arduino als mobiles Messgerät einsetzen.

Wozu eigentlich?

Mikrocontroller stecken hinter der Fassade vieler elektronischer Produkte. Mit dem Arduino kann man spielerisch Geräte nach eigenen Ideen bauen.

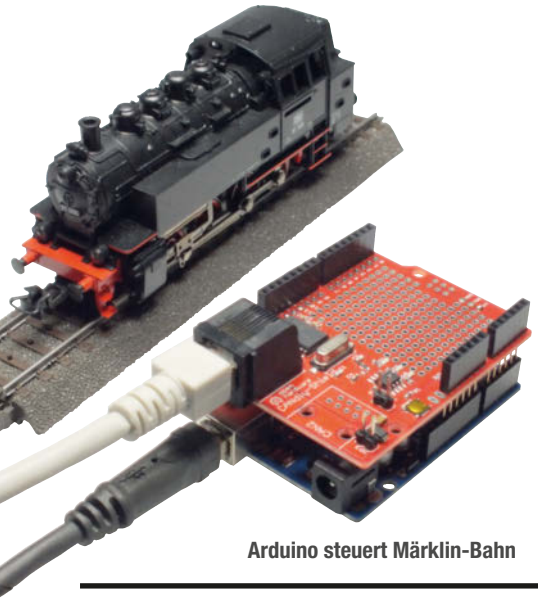
von Daniel Bachfeld

Arducopter

Jeder hat ihn, aber nur wenige wissen es: Mikrocontroller stecken überall drin. Ob im Auto, DSL-Router, Smartphone oder Tablet: Mikroprozessoren helfen Dinge automatisiert zu schalten, zu steuern, zu regeln und zu kontrollieren. Auch im nicht kommerziellen Bereich lassen sich Mikrocontroller kreativ und praktisch einsetzen, etwa um regelmäßig wiederkehrende Aufgaben erledigen zu lassen, statischen Dingen Leben einzuhauchen oder um etwa die Umwelt mit Sensoren zu erfassen und die Daten anzuzeigen – und das alles ohne einen vergleichsweise teuren und klobigen PC.

Dank des großen und günstigen Angebots an elektronischen Komponenten kann der Privatmann leicht Geräte entwickeln, die es so bislang gar nicht am Markt gibt oder sie passgenau auf seine Anforderungen zuschneidern. Aquarienfutterautomat, Gartenbewässerungssteuerung, Luftgütemessgerät, Füllstandsüberwachung, Abstandswarner, Smartwatch, Drohnen, Roboter, Lichtsteuerung, Heimautomation: Je nach Gusto übernimmt der Mikrocontroller verschiedene Aufgaben in Haus, Hof und Hobby.

Tetris



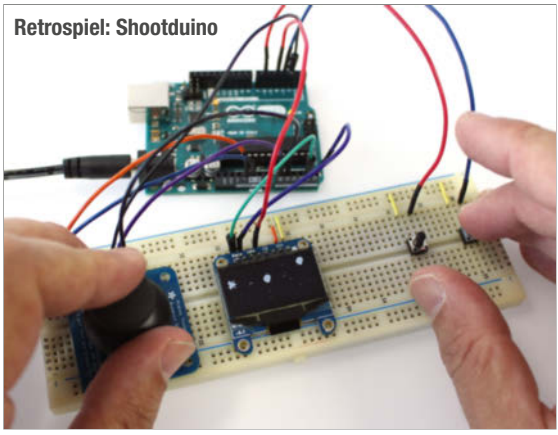
Arduino steuert Märklin-Bahn

Mit dem Arduino und der auch für Nicht-Informatiker leicht verständlichen Software-entwicklungsumgebung findet der Anfänger einen schnellen Einstieg in die Programmierung von Mikrocontrollern. Und obwohl der Arduino und seine Software ursprünglich für eher IT-ferne Anwender wie Künstler, Designer und Medienwissenschaftler entwickelt wurde, nutzen ihn mittlerweile auch professionelle Hardware-Entwickler für den Bau von Prototypen. Durch sein Stecksystem lässt sich der Arduino nämlich ohne Löten spielend leicht und schnell mit weiterer Hardware – sogenannten Shields – erweitern.

Schnell, einfach und günstig statt langwierig, kompliziert und teuer: Durch dieses Grundprinzip hat sich eine große Internet-Community um den Arduino geschart, die sich gegenseitig inspiriert und unter die Arme greift. Arduino gilt derzeit als erste Wahl, wenn es um die Umsetzung eigener Projekte geht. Deshalb bieten zahlreiche Hersteller Zusatzprodukte an, die sowohl hardware- als auch softwarekompatibel sind, mitunter aber mehr Rechenleistung liefern. Mit Intels briefmarkengroßem Mikrocontroller Edison beispielsweise können Arduino-Programme die Rechenleistung von Atom-Prozessoren nutzen und sogar Funktionen wie WLAN und Bluetooth nutzen, die der Original-Arduino nur durch zusätzliche Shields bietet.

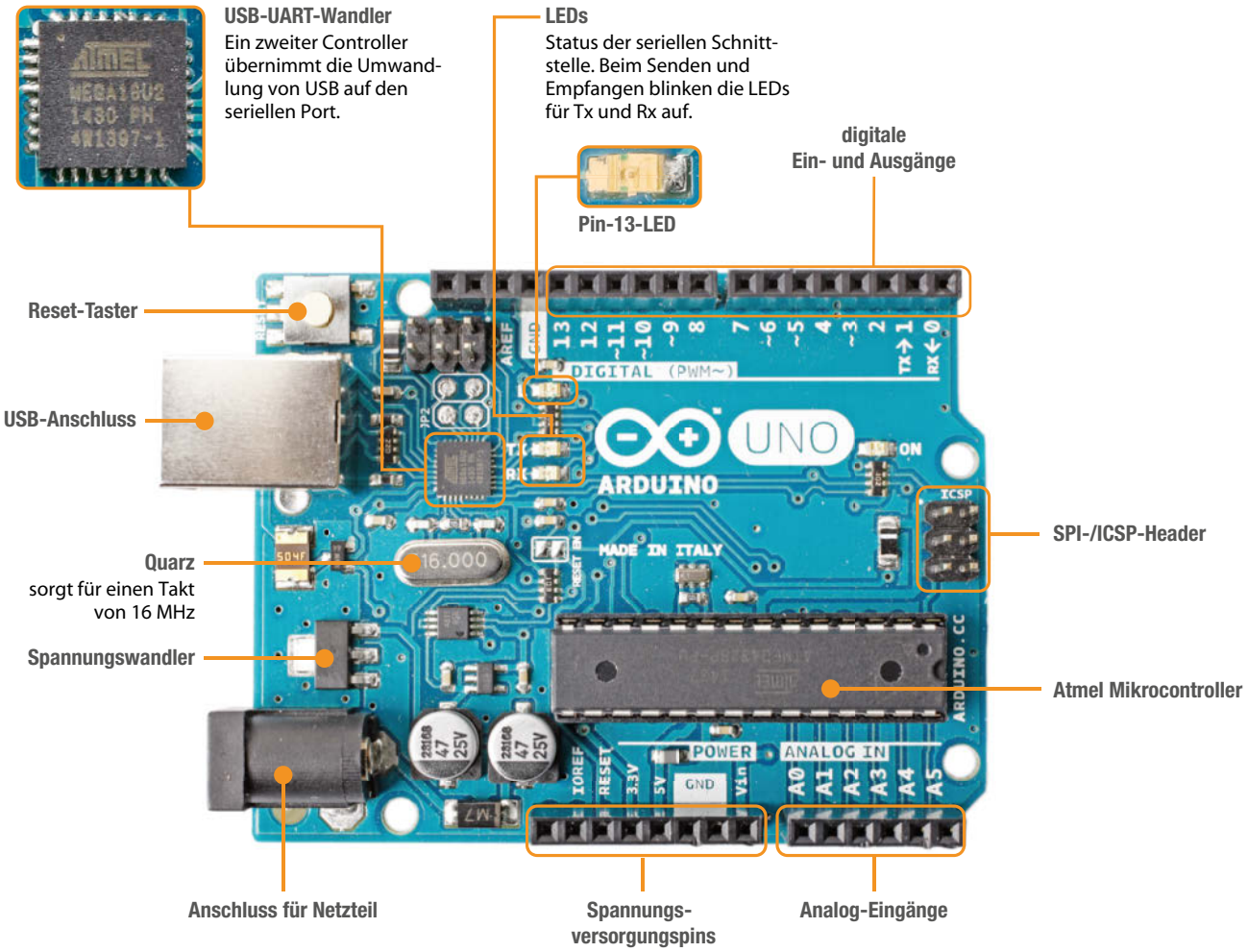
Mit diesem Make-Special „Arduino“ wollen wir einen leichten, kurzweiligen, aber langfristig fesselnden Einstieg in den Arduino und seine Möglichkeiten bieten. Alles, was Sie für die ersten Schritte benötigen, sind ein USB-Kabel und ein Stück Draht – zur Not tut es auch eine auseinandergebogene Büroklammer. Zur Programmierung steht die kostenlose Arduino-IDE unter arduino.cc im Internet bereit. Und nun viel Spaß!

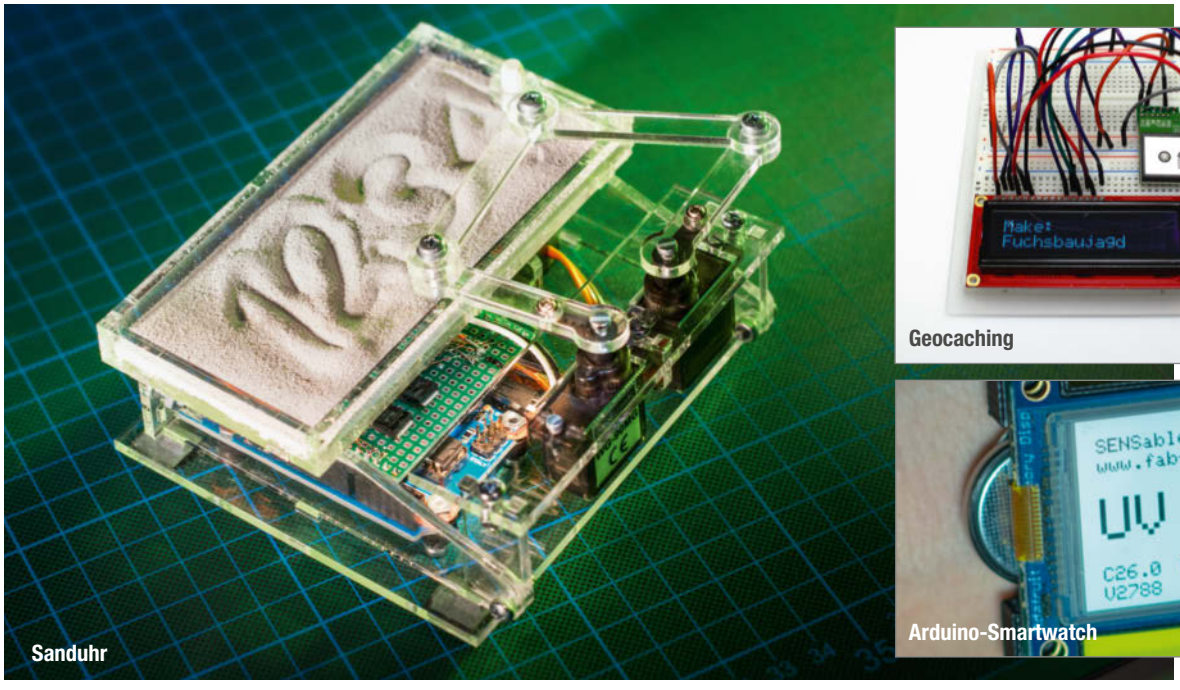
Links und Foren
make-magazin.de/x82g



Retrospiel: Shootduino

Bestandteile eines Arduino UNO (Revision 3)





Die Pins

Eine Übersicht über die Funktion der Arduino-Pins

Der im Arduino eingesetzte Prozessor hat im Inneren viele Funktionen für digitale Ein- und Ausgabe zu bieten, die er jedoch nicht alle gleichzeitig über seine externen Beinchen (Pins) dem Anwender zu Verfügung stellen kann. Mit einer ausgefeilten Logik kann man festlegen, welche interne Funktion mit der Außenwelt in Verbindung steht. So kann ein Pin mal ein analoger Eingang oder Vergleicher, mal ein digitaler Ausgang, mal ein Generator für Rechtecksignale oder in Kombination mit weiteren Pins sogar ein sogenannter Bus zum Datenaustausch mit anderen elektronischen Systemen sein.

Durch die Flexiblität des verwendeten Atmel-Prozessors lässt sich der Arduino für verschiedene Aufgaben einsetzen. Es genügt, die Verdrahtung der alten Komponenten abzunehmen, neue Komponenten anzuschließen und einen neuen Sketch zu installeren. Hier erklären wir die Bedeutung der Pins des Arduino, zuerst die mit Sonderfunktionen:

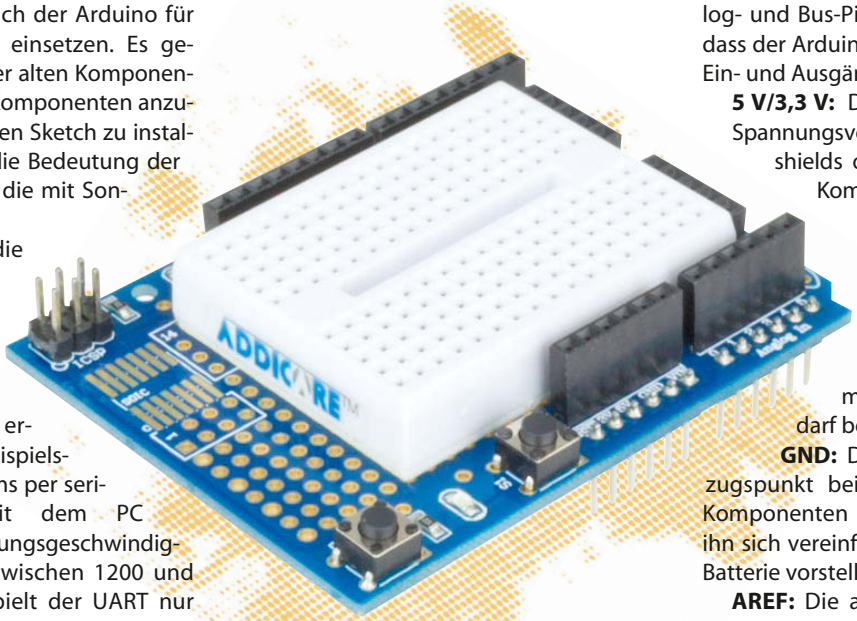
RX, TX: Das sind die Sende- und Empfangspins der seriellen Schnittstelle (UART, Universal Asynchronous Receiver Transmitter). Bevor der USB-Bus erfunden wurde, waren beispielsweise Mäuse und Modems per serieller Schnittstelle mit dem PC verbunden. Die Übertragungsgeschwindigkeit liegt in der Regel zwischen 1200 und 115 200 Baud. Heute spielt der UART nur noch im Entwicklerbereich eine Rolle.

SCL, SDA: Die zwei Pins gehören zum I²C-Bus (Inter IC Communication), über den der Arduino Daten mit Sensoren und anderer Elektronik austauschen kann. Im Unterschied zum UART gibt es nur eine Leitung für die Hin- und Rückrichtung (SDA, Serial Data), dank eines einheitlichen Protokolls und eines Taktsignals (SCL, Serial Clock) gibt es aber keine Konflikte auf dem Bus.

SPI: Das Serial Peripheral Interface ist ein synchroner serieller Bus, mit Hin-, Rück- und Taktleitung. Beim Arduino dient er dem Datenaustausch mit Sensoren und Steuermodulen, aber auch der Programmierung seines Flash-Speichers mit einem speziellen Gerät. Standardmäßig arbeitet der SPI auf dem Arduino mit 4 MHz.

Digital (PWM~): Die digitalen Pins lassen sich jeweils als Ein- oder Ausgang programmieren. Arbeiten sie als Eingang, kann man dem Arduino mit einer Spannung von 0 V eine logische 0 und mit 5 V eine logische 1 von außen signalisieren, auf die er reagieren kann. Ist ein Pin als Ausgang konfiguriert, kann er per Programm seiner Umwelt oder weiterer Elektronik eine logische 1 oder eine logische 0 signalisieren.

Daneben kann man die Ausgangsspannung von 5 V auch direkt benutzen, um eine Leuchtdiode (LED) zum Leuchten zu bringen oder ein Relais zu schalten, das wiederum ein anderes Gerät mit Batterieversorgung oder gar mit Netzspannung anschaltet. Zu beachten ist, dass ein Ausgang als Dauerlast nur einen Strom von 20 Milliampere liefert, kurzzeitig auch bis zu 40. Darüber droht ein dauerhafter Schaden des Pins. Um das zu verhindern, muss man den Strom mit einem Widerstand begrenzen. Wie man das macht, erfahren Sie im hinteren Teil des Hefts.



Auf Prototypingshields lassen sich alle Pins des Arduino für eigene Elektronik anzapfen.

Alternativ können einige der digitalen Pins (zu erkennen an der Tilde ~ vor der Nummer auf dem Board) ein sogenanntes PWM-Signal ausgeben. Neben der Leistungssteuerung von LEDs benötigt man dieses Signal in erster Linie zur Ansteuerung von Servo-Motoren, wie man sie im Modellbau einsetzt.

Analog In (A0–A5): Da die Welt eben nicht nur schwarz (5 V) und weiß (0 V) ist, benötigt man noch eine Möglichkeit, Spannungen irgendwo dazwischen zu messen. Ein Analog-Digital-Wander wandelt, wie beim Mikrofon-Eingang am PC, analoge Signale in digitale Werte zwischen 0 und 1023 (10 Bit). Der UNO hat eigentlich nur einen einzigen internen Wandler, mit einem eingebauten Umschalter kann er jedoch die einzelnen externen Eingänge auf den Eingang des A/D-Wandlers legen.

Grundsätzlich lassen sich auch alle Analog- und Bus-Pins als I/O-Pins festlegen, sodass der Arduino UNO insgesamt 20 digitale Ein- und Ausgänge befehligen kann.

5 V/3,3 V: Die beiden Pins dienen der Spannungsversorgung von Erweiterungsshields oder anderen elektronischen Komponenten.

VIn: Hier liegt die Spannung an, mit der der Arduino bei Verwendung eines externen Netzteils versorgt wird. Shields mit Motoren und anderen Verbrauchern mit höherem Spannungsbedarf benötigen diesen Pin.

GND: Das ist der gemeinsame Bezugspunkt beim Zusammenschalten von Komponenten mit dem Arduino. Man kann ihn sich vereinfacht wie den Minuspol einer Batterie vorstellen.

AREF: Die analogen Eingänge nehmen die Digitalisierung in Bezug auf eine bestimmte Spannung vor, standardmäßig 5 V. Man kann aber auch durch eine externe Spannungsreferenz einen anderen Wert vorgeben. Liefert beispielsweise ein Sensor nur eine maximale Spannung von 1 V, würde man bei AREF = 5 V nur ein Fünftel des Wertebereichs (0–205) nutzen. Sinnvoller wäre, AREF mit einer Referenzspannung von 1 V zu speisen.

IOREF: Damit signalisiert der Arduino einem Shield, mit welchem Spannungspegel er arbeitet. In der Regel spielt dieser Pin keine Rolle.

Reset: Neben dem manuellen Druck auf den Resetkopf kann auch eine externe Schaltung einen Neustart des Arduino auslösen.

—dab