

# Arduino Microcontroller

10. Mai 2022

# 1 Digitale Messdatenerfassung

Das **digitale Messen** einer analogen Größe beinhaltet eine Diskretisierung der Messwerte. Soll z.B. ein Messbereich von 0,0V bis 5,0V abgedeckt und dies über einen Analog-Digital-Umwandler mit 8 bit (256 digit) diskretisiert werden, ist die maximale Auflösung der Messung  $5,0\text{V}/256\text{ digit} = 0,02\text{V/digit}$ . Die Diskretisierung bestimmt also den kleinsten auflösbaren Schritt der Messgröße.

**Automatisiertes Messen** ist maschinelles Übertragen von Messwerten in eine verwertbare Form. Das Verwerten kann ein Eintragen in eine Liste sein, um später ausgewertet zu werden, oder auch das Überprüfen durch einen weiteren Algorithmus, um z.B. Notaus auszulösen. Die Messungen sind dadurch wiederholbar und vergleichbar. Für häufig wiederkehrende, gleiche Messungen kann somit der Aufwand im laufenden Betrieb maßgeblich vermindert werden.

# 2 Arduino Microcontroller

Arduino ist eine Open-Source-Elektronikplattform, die auf vergleichsweise einfach zu bedienender Hardware und Software basiert, insofern die vorgefertigte Entwicklungsumgebung genutzt wird. Arduino-Boards sind in der Lage, Eingaben zu lesen (z.B. Licht an einem Sensor oder das Aktivieren mechanischer Schalter) und sie in eine Ausgabe umzuwandeln (z.B. einen Motor zu aktivieren oder eine LED einzuschalten). Das Board kann je nach Bestückung und der aufgespielten Software unterschiedliche Aufgaben erledigen: von einfachen Logik-Verknüpfungen bis hin zu komplexeren Berechnungen wie Regelung einer Prozessgröße, können diverse Einsatzfelder bedient werden. Dabei ist die begrenzte Anzahl digitaler Ein- und Ausgänge, sowie die geringe Schaltleistung (5 V, 20 mA) bei der Planung einer Schaltung zu berücksichtigen. Der Leistungsbereich kann z.B. durch Transistor-Schaltungen erweitert werden. Durch sogenannte *Shields* ist es möglich, die Funktionalität des Boards schnell und modular zu verändern oder zu erweitern.

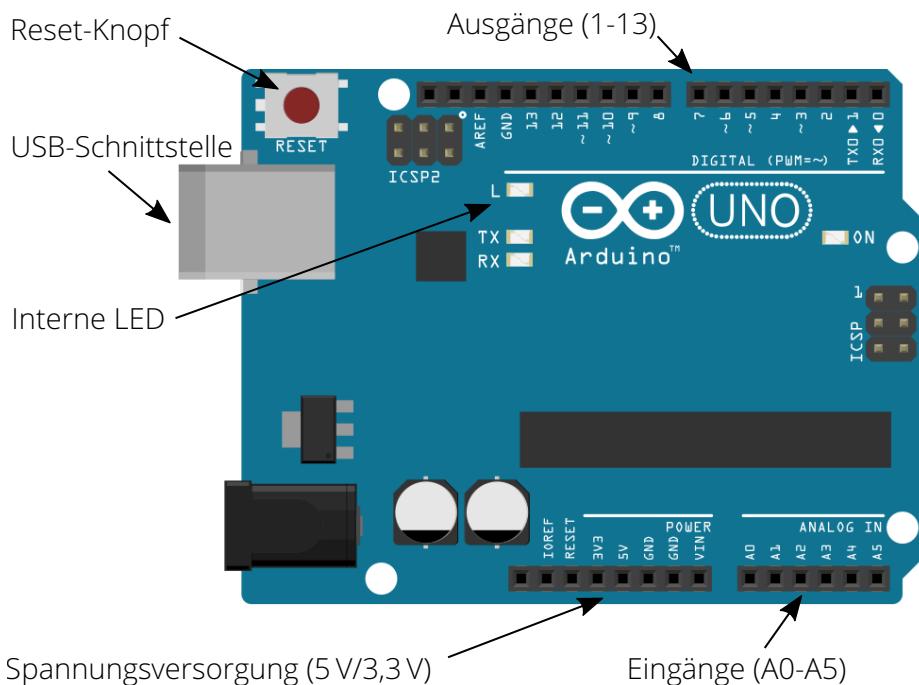


Abbildung 2.1: Übersicht der wichtigsten Schnittstellen des Arduino Uno Rev. 3

Zum Aufspielen der Software wird die Arduino-Programmiersprache (basierend auf C++, <https://www.arduino.cc/reference/de/>), und die Arduino-Entwicklungsumgebung (IDE) verwendet. Durch den Arduino (bzw. baugleicher Alternativ-Produkte) soll in den Praktika die Funktionsweise eines Microcontrollers, sowie die damit umgesetzte automatisierte Messdatenerfassung vermittelt werden. Eine Übersicht der wichtigen Schnittstellen des Arduino-Boards (Modell Uno Rev. 3) ist in Abbildung 2.1 dargestellt.

### 3 Arduino Entwicklungsumgebung

Die Arduino IDE (Integrated Development Environment) ist für alle gängigen Betriebssysteme kostenfrei unter <https://www.arduino.cc/en/software> beziehbar. Die in der Arduino IDE genutzte Programmiersprache ist an C++ angelehnt, außerdem gibt es spezielle Befehle für die Steuerung der Ein- und Ausgänge (Pins). Der geschriebene Programm-Code wird abschließend in eine für den Microcontroller lesbare Sprache übersetzt (kompiliert) und über USB aufgespielt. Eine Übersicht der Oberfläche ist in Abbildung 3.1 ersichtlich.

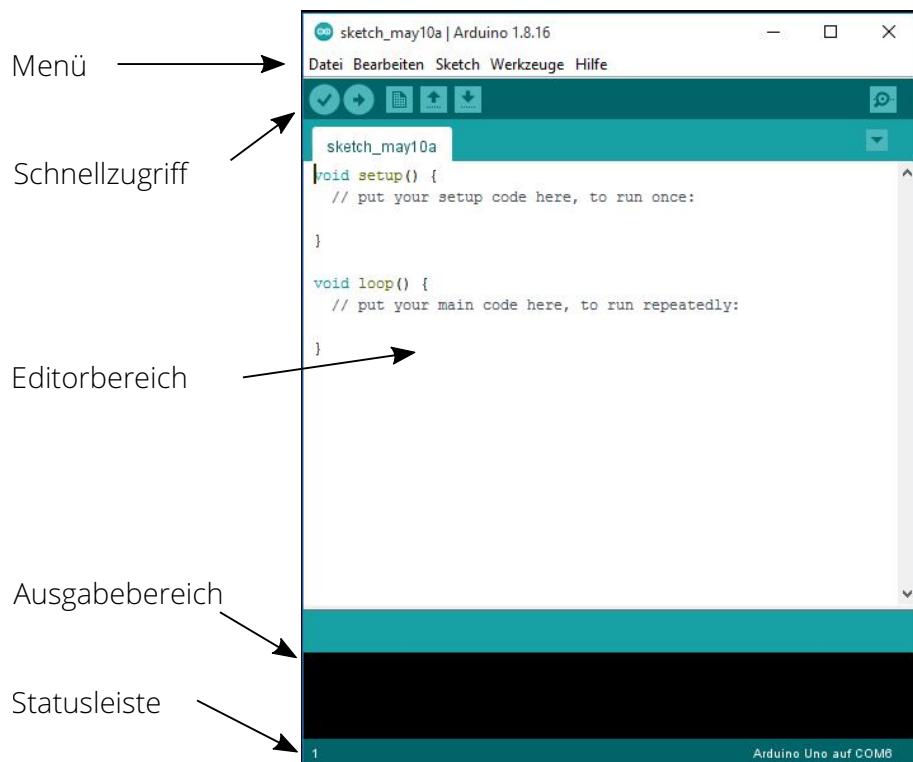


Abbildung 3.1: Arduino Entwicklungsumgebung (IDE)

Im **Menü** unter dem Menüpunkt *Datei* können neue Arduino-Skripte (Sketches) erstellt und gespeichert werden. Es können unter anderem auch Beispieldateien geöffnet werden. Eine solche Datei wird im Kapitel 5 näher betrachtet. Außerdem können die Voreinstellungen bearbeitet werden. Der Menüpunkt *Bearbeiten* bezieht sich auf den Editor-Bereich und beinhaltet Funktionen wie Suche und Kommentieren sowie das Anpassen der Schriftgröße. Der Menüeintrag *Sketch* ermöglicht das Überprüfen des geschriebenen Codes, das Hochladen des Codes auf ein via USB angeschlossenes Board sowie das Einbinden externer Bibliotheken. Im Menüeintrag *Werkzeuge* ist es möglich, die externen Bibliotheken über das Internet herunterzuladen und zu aktualisieren, sowie diese auch wieder zu löschen. Das Öffnen des seriellen

Monitors sowie des seriellen Plotters erfolgt ebenfalls über diesen Menüeintrag. Ist der Arduino mit dem Computer via USB verbunden, können außerdem das erkannte Board und Verbindungseigenschaften eingesehen werden: *Board*: „Arduino Uno“ und *Port*: „/dev/ttyUSB0“ (Ubuntu) oder *Port*: „COM1“ (Windows).

Die **Schnellzugriffe** beinhalten die Quellcode-Aktionen *Kompilieren*, *Hochladen*, sowie Dateioperationen wie *Neu*, *Öffnen* und *Speichern*.

Im **Editorbereich** wird der Quellcode entwickelt. Dieser ist einen Setup-Bereich und einen Schleifen-Bereich aufgeteilt. Das Setup wird nur zu Beginn, also kurz nach dem Zeitpunkt der Stromversorgung, ausgeführt – die Schleife hingegen wird solange ausgeführt, bis dem Schaltkreis die Energieversorgung entzogen wird.

Der **Ausgabebereich** zeigt Meldungen, welche z.B. beim Kompilieren oder Hochladen aufkommen.

Die **Statusleiste** zeigt die aktuelle Zeilennummer sowie die verbundene Hardware.

## 4 Aufbau einer einfachen Schaltung

Die Beispialschaltung ist das simple Ein- und Ausschalten einer LED in einem definierten Zeitabstand. Außerdem soll der aktuelle Zustand der LED im seriellen Monitor angezeigt werden. Für den Aufbau der Schaltung werden folgende Komponenten benötigt:

- 1x Arduino Uno Rev. 3 inkl. USB-Kabel
- 1x Computer mit installierter Arduino IDE
- 1x Steckplatine (Breadboard)
- 1x LED (Farbe beliebig, nicht im Koffer enthalten)
- 1x 100 Ohm Widerstand
- 2x Kabel (beidseitig Stecker)

Die LED benötigt nur eine geringe Leistung, sodass sie direkt über einen Ausgang des Arduino-Boards betrieben werden kann. Zusätzlich ist zur Strombegrenzung ein korrekter Vorwiderstand notwendig, da sonst die Diode oder der Ausgang am Arduino zerstört wird. Der Schaltplan ist Abbildung 4.2 zu entnehmen. Zu Beachten ist dabei die Sperr- bzw. Durchlassrichtung der Diode. Alternativ kann der Versuch auch ohne die externe LED und ausschließlich über auf dem Arduino-Board befindliche LED durchgeführt werden.

Die Steckplatine (Breadboard) ist in vier Bereiche unterteilt. Die zwei Bereiche am oberen und unteren Rand, bestehend aus zwei getrennten Leitungen und markiert mit roten und blauen Strichen, verlaufen horizontal. Diese werden häufig verwendet, um die Versorgungsspannung und die Erdung über ein breiteres Board besser zugänglich zu machen. In diesem einfachen Fall wurden diese Bereiche nicht verwendet. In der Mitte der Steckplatine befinden sich zwei weitere Bereiche, welche horizontal durch eine Vertiefung voneinander getrennt sind. Darin verlaufen die Leitungen vertikal bis zur Trennung in der Mitte. Größere Boards, wie z.B. in Abbildung 4.1 sind üblicherweise noch einmal vertikal getrennt. Die horizontal ausgerichteten Bereiche werden dann an dieser Stelle unterbrochen. In den Koffern sind die großen Boards (vgl. Abbildung 4.1), aber auch kleinere zu finden. Die kleinen Steckplatten sind nur halb so groß, weshalb die vertikale Trennung des Boards entfällt.

## 4 Aufbau einer einfachen Schaltung

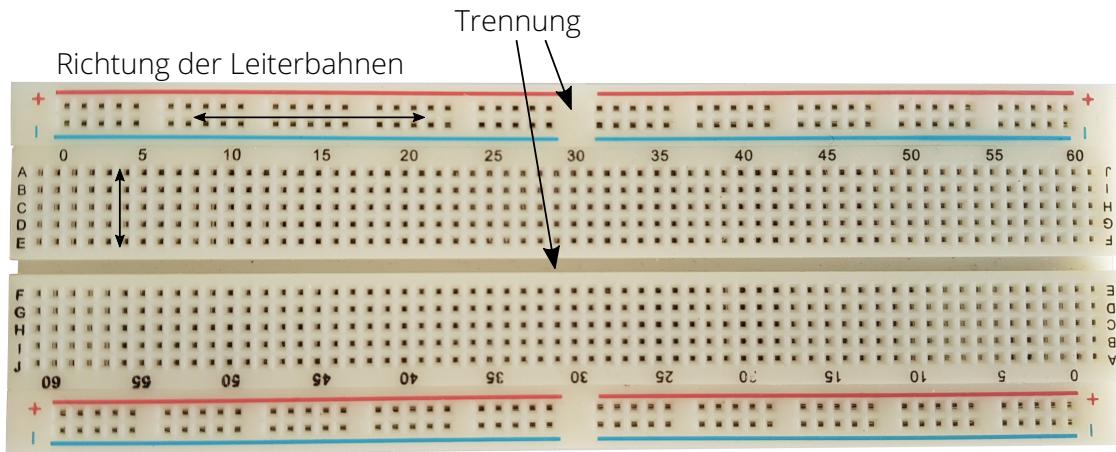


Abbildung 4.1: Aufbau eines großen Breadboards

Die Leitungen sind in der Abbildung 4.2 an den vertikalen, grünen Markierungen erkennbar. Die Spannungsversorgung verläuft also vom Pin 13, über den Vorwiderstand, durch die LED und von dort wieder aufs Arduino-Bord zum GND-Pin (Ground/Erdung).

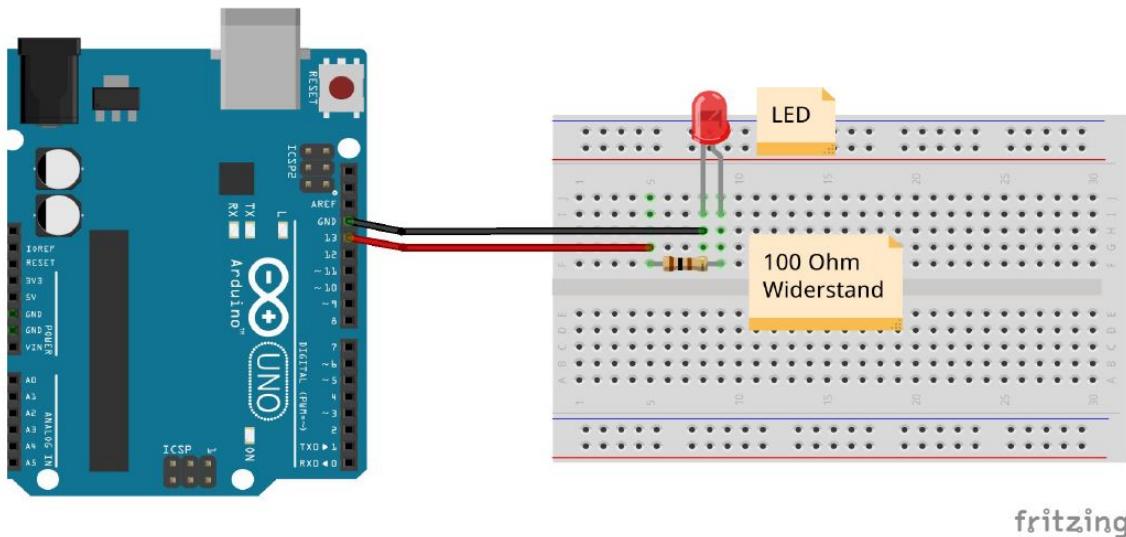


Abbildung 4.2: Schaltplan des Beispiel-Sketches „Blink“

Der Widerstandswert in Ohm kann über den darauf befindlichen Farbcoden bestimmt werden. Der Widerstand wird mittels Tabelle 1 und Gleichung (4.1) gebildet. Ein Beispiel für einen  $100\Omega$  Widerstand (braun, schwarz, braun, gold) ist in Abbildung 4.2 ersichtlich. Dieser wird in Anwendung der Gleichung (4.1) in Gleichung (4.2) gebildet.

$$(10A + B) \pm D \quad (4.1)$$

$$(10 \cdot 1 + 0) \cdot 10 \pm 5\% = 100 \pm 5\% \quad (4.2)$$

Tabelle 1: Widerstandsbestimmung (4 Ringe) bei Kohleschichtwiderständen

| Ringfarbe | 1. Ring (A) | 2. Ring (B) | 3. Ring (Multiplikator) (C) | 4. Ring (Toleranz) (D) |
|-----------|-------------|-------------|-----------------------------|------------------------|
| schwarz   | 0           | 0           | -                           | -                      |
| braun     | 1           | 1           | x 10                        | 1,00 %                 |
| rot       | 2           | 2           | x 100                       | 2,00 %                 |
| orange    | 3           | 3           | x 1.000                     | -                      |
| gelb      | 4           | 4           | x 10.000                    | -                      |
| grün      | 5           | 5           | x 100.000                   | 0,50 %                 |
| blau      | 6           | 6           | x 1.000.000                 | 0,25 %                 |
| violett   | 7           | 7           | x 10.000.000                | 0,10 %                 |
| grau      | 8           | 8           | -                           | -                      |
| weiß      | 9           | 9           | -                           | -                      |
| gold      | -           | -           | x 0,1                       | 5,00 %                 |
| silber    | -           | -           | x 0,01                      | 10,00 %                |

## 5 Programmierung eines Arduino-Sketches

Der Quelltext für den Beispiel-Sketch kann prinzipiell in jedem Texteditor entwickelt werden. Die Arduino IDE bietet aber wesentliche Vorteile gegenüber den meisten anderen Editoren: Syntax-Highlighting, Einbinden und Verwalten von externen Bibliotheken, Überprüfen des Quelltextes auf (Syntax-)Fehler und letztendlich wird die Software benötigt, um den Quellcode zu Kompilieren sowie auf das Arduino-Board aufzuspielen.

Der Quelltext, welcher in Tabelle 2 zu sehen ist, wird nachfolgend erläutert. Zunächst werden die verwendeten Variablen initialisiert. Dies erfolgt in der Zeile 02 in der Form *Typ(Integer) Name(LED\_PIN) = Wert(13)*. In Zeile 05 beginnt das Setup, dieses wird zu Beginn jeweils nur ein Mal ausgeführt. Hier werden Vorbereitungen für den Betrieb getroffen. In Zeile 07 wird die interne LED dem Namen *LED\_BUILTIN* zugewiesen. Das Starten der Ausgabe des seriellen Monitors mit der Baud-Rate 9600 (Übertragungsfrequenz) erfolgt in Zeile 09. Ein Beispiel für eine Textausgabe ist in Zeile 11 ersichtlich.

Die Schleife beginnt in Zeile 15. Die anschließenden Anweisungen werden bis zur Unterbrechung der Stromversorgung ausgeführt. Nach einem Reset beginnt das Programm bei Zeile 01. Mit dem Befehl *digitalWrite(PIN\_NR(Integer), STATUS(HIGH, LOW))* werden in den Zeilen 17 & 18 sowie 23 & 24 die Spannungssignale auf ein oder aus bzw. auf hoch oder niedrig gesetzt. Die Nummer der Pins wurde zuvor Variablen zugewiesen, siehe Zeilen 02 & 07. Somit kann das Schalten der Pins darüber erfolgen: für die interne LED wird die Variable *LED\_BUILTIN* und für die externe LED wird *LED\_PIN* verwendet. Die Zuordnung der Pin-Zustände erfolgt mittels der vordefinierten Zustände *HIGH* und *LOW*. *HIGH* ist vergleichbar mit Einschalten, also ein hohes Spannungssignal - *LOW* ist vergleichbar mit Ausschalten, also ein niedriges Spannungssignal. Unterbrechen des Programmcodes für eine bestimmte Zeit (Angabe in Millisekunden) wird mittels *delay(TIME)* in den Zeilen 21 & 27 umgesetzt. Kommentare können mit „// Kommentar“ für den Rest der Zeile oder „/\* Kommentar \*/“ für alles, was zwischen den Markierungen steht, umgesetzt werden. Jede Anweisung wird außerdem mit einem Semikolon beendet.

Tabelle 2: Quelltext „send\_blink“ auf Basis des modifizierten Beispiel-Sketches „blink“

|    |  |
|----|--|
| 01 | // Festlegen des Pin 13 fuer die externe LED                           |
| 02 | int LED_PIN = 13;  |
| 03 |  |
| 04 | // Das Setup wird einmalig nach dem (Neu-)Start des Boards ausgefuehrt |
| 05 | void setup() {   |
| 06 | // Festlegen der eingebauten LED (LED_BUILTIN) als Ausgabe             |
| 07 | pinMode(LED_BUILTIN, OUTPUT);  |
| 08 | // Starten der Ausgabe im seriellen Monitor mit Baud 9600              |
| 09 | Serial.begin(9600)   |
| 10 | // Textausgabe ueber den seriellen Monitor, "\t"= Zeilenumbruch        |
| 11 | Serial.print("Status der LED: \n");                                    |
| 12 | }  |
| 13 |  |
| 14 | // Wird ausgefuehrt bis die Stromversorgung unterbrochen wird          |
| 15 | void loop() {  |
| 16 | // LEDs einschalten (Spannungslevel ist HIGH)                          |
| 17 | digitalWrite(LED_BUILTIN, HIGH);                                       |
| 18 | digitalWrite(LED_PIN, HIGH);   |
| 19 | Serial.print("LED Status: EIN \n");                                    |
| 20 | // 3 Sekunden warten   |
| 21 | delay(3000);   |
| 22 | // LEDs ausschalten (Spannungslevel ist LOW)                           |
| 23 | digitalWrite(LED_BUILTIN, LOW);  |
| 24 | digitalWrite(LED_PIN, LOW);  |
| 25 | Serial.print("LED Status: AUS \n");                                    |
| 26 | // 3 Sekunden warten   |
| 27 | delay(3000);   |
| 28 | }  |

## 6 Verwendung des Arduino und Ausgabe an eine Konsole

Nachdem der Quellcode in die IDE geladen und Überprüft wurde, kann dieser an einen per USB angeschlossenen Arduino aufgespielt werden. Das Board benötigt einige Sekunden für das Laden des neuen Codes und führt diesen dann unverzüglich aus. Nach dem Bereitstellen der Energieversorgung via USB am programmierten Arduino sollte in einem Abstand von drei Sekunden sowohl auf dem Arduino-Board als auch auf dem wie in Abbildung 4.2 bestückten Breadboard das Ein- und Ausschalten der LEDs zu beobachten sein. Über die Arduino IDE kann über den Menüpunkt *Werkzeuge* sowie über die Tastatur via *Strg+Umschalt+M* der serielle Monitor geöffnet werden. Dieser ist in Abbildung 6.1 nach einigen Malen Ein- und Ausschalten der LEDs dargestellt. Über das Auswahlfeld *Zeitstempel anzeigen* ist es möglich die aktuelle Computer-Zeit zum Zeitpunkt der Ausgabe anzeigen zu lassen. Die Option Autoscroll ermöglicht durch Ausschalten auch im laufenden Betrieb Daten zu markieren und kopieren. Alternativ kann auch die USB-Verbindung zum Arduino getrennt werden, um die Energieversorgung zu stoppen. Die Ausgabe im seriellen Monitor bleibt dabei erhalten. Es empfiehlt sich, die kopierten Daten in einen Texteditor ihrer Wahl einzufügen und als *CSV-Datei* (comma separated values) zu speichern. Diese Datei kann mit Angabe der Werttrennzeichen (z.B. Leerzeichen, Tab, Komma, Semikolon) sowie der Dezimaltrennzeichen (Punkt oder Komma) in diversen Datenverarbeitungsprogrammen eingelesen werden.

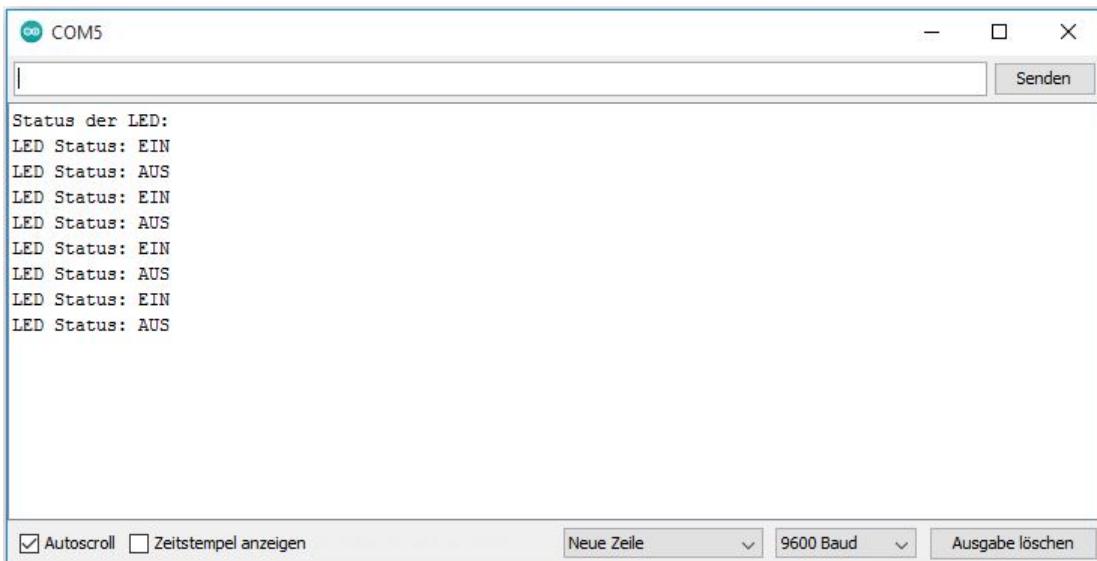


Abbildung 6.1: Ausgabe des seriellen Monitors (Sketch „send\_blink“)