

Aufgabenblatt 3 Termine: KW 17, KW 18

Gruppe	
Name(n)	Matrikelnummer(n)

Ein Hauptmerkmal eingebetteter Systeme ist die Fähigkeit zur Protokoll-gesteuerten Kommunikation mit extern angeschlossener Hardware, die über primitives Betrachten des Signalpegels der Kommunikationsleitungen hinausgeht. Sensoren, Aktuatoren oder andere eingebettete Systeme kommen dabei als Kommunikationspartner in Frage.

Häufig sind Mikrocontroller mit unterschiedlichen Kommunikationsschnittstellen ausgestattet. Besonders häufig handelt es sich dabei um **serielle Kommunikationsschnittstellen**. Besonders breit etabliert sind:

- **UART**: Universal Asynchronous Receiver/Transmitter
Beispielsweise **RS-232**: Ein Kommunikationspartner pro Schnittstelle, asynchroner Datentransfer
- **I²C**: Inter-Integrated Circuit
Serieller Bus, (multi-)**Master/Slave** Kommunikation, synchroner Datentransfer
- **SPI**: Serial Peripheral Interface
Serieller Bus, **Master/Slave** Kommunikation, synchroner Datentransfer

Der Mikrocontroller des Arduino Due enthält $4 \times \text{UART}$, $2 \times \text{I}^2\text{C}$ und $2 \times \text{SPI}$ Schnittstellen. Setzen Sie sich noch einmal mit der **Arduino Due Pinbelegung** auseinander, um die Anschlusspins der jeweiligen Schnittstellen identifizieren zu können. In der folgenden Aufgabe sollen Sie sich zunächst ausschließlich mit serieller Kommunikation über die **UART**-Schnittstelle befassen.

ACHTUNG: Die RX und TX Anschlusspins der ersten seriellen UART-Schnittstelle am Arduino Due (Anschlusspins 0 und 1) sind gleichzeitig an den ATmega16U2 USB-to-TTL Wandler angeschlossen. Dieser Wandler ermöglicht die serielle Kommunikation mit dem angeschlossenen Rechner. Der Wandler wird jedoch zusätzlich für das Programmieren des Arduino Due verwendet. Sollten die Anschlusspins zum Zeitpunkt des Programmierens belegt sein wird der Vorgang fehlschlagen!

Aufgabe 3.1

Benutzen Sie für diese Aufgabe den in die Arduino IDE integrierten **seriellen Monitor**. Diesen können Sie entweder durch Anklicken des Lupen-Symbols (rechts, oben im Fenster der Arduino IDE), über das Menü (Werkzeuge -- Serieller Monitor) oder mit einem Keyboard-Shortcut (Strg+Shift+M) aufrufen. Mit dem seriellen Monitor haben Sie die Möglichkeit Daten an den Mikrocontroller zu senden und die vom Mikrocontroller an die erste serielle UART-Schnittstelle gesendeten Daten zu empfangen.

Für die Lösung der Aufgabe wird nahegelegt den Versuchsaufbau gemäß Abbildung 1 zu verdrahten.

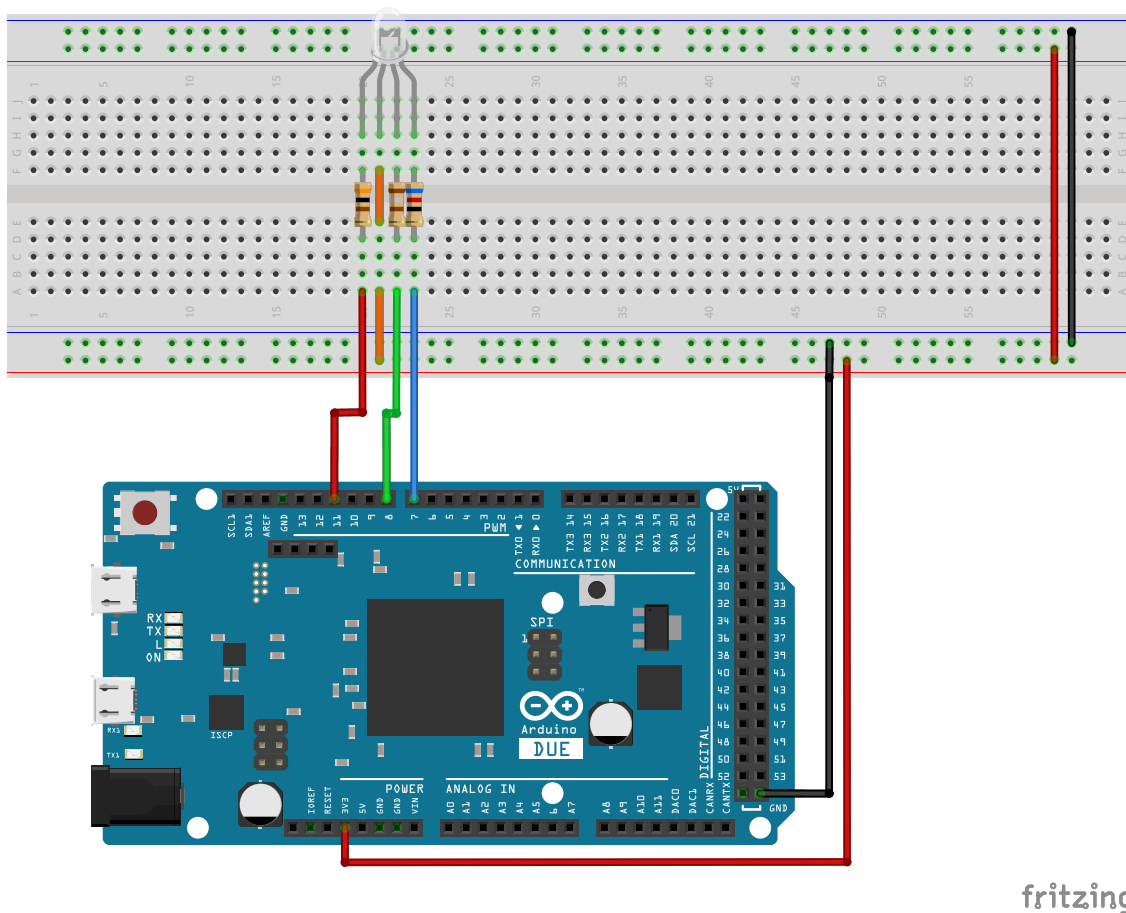


Abbildung 1: Vorschlag für die Verdrahtung des Versuchsaufbaus.

Beachten Sie die Anordnung der Anschlüsse der **RGB-LED** (siehe Abbildung 2). Die Farbkanäle (Red, Green, Blue) der LED besitzen eine **gemeinsame Anode** (der „Pluspol“). Schließen Sie diese an VCC (3,3 V) an. Schließen Sie jeden einzelnen Kanal über den zugehörigen Vorwiderstand an einen **PWM-fähigen** Anschlusspin des Arduino Due an.

Welche Auswirkung hat die gemeinsame Anode der RGB-LED auf die Ansteuerung der drei Farbkanäle mit der `analogWrite()` Funktion?

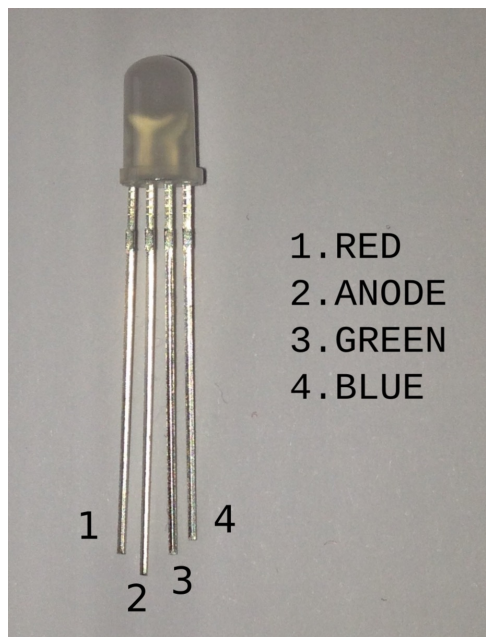


Abbildung 2: Anordnung der Anschlüsse der RGB-LED.

Benutzen Sie für den Anschluss der RGB-LED folgende Vorwiderstände:¹

Red : 300 Ω (Durchlassspannung: 1,6 V)

Green : 180 Ω (Durchlassspannung: 2,4 V)

Blue : 62 Ω (Durchlassspannung: 3,0 V)

Das Arduino Framework macht Ihnen die Verwendung der seriellen UART-Schnittstellen einfach. Betrachten Sie die Dokumentation der Bibliothek **Serial** und verschaffen Sie sich einen Überblick über den Ihnen zur Verfügung stehenden Umfang an Funktionen.

Die Bibliothek stellt viele nützliche Funktionen zur Verfügung (die Sie gerne verwenden können), eine funktionsfähige Lösung der Aufgabe lässt sich bereits mit folgenden grundlegenden Funktionen erstellen:

* <code>Serial.begin(<speed>)</code>	→ <code>Serial.begin</code>
* <code>Serial.available()</code>	→ <code>Serial.available</code>
* <code>Serial.read()</code>	→ <code>Serial.read</code>
* <code>Serial.write(<arg>)</code>	→ <code>Serial.write</code>
* <code>Serial.print(<arg>)</code>	→ <code>Serial.print</code>
* <code>Serial.println(<arg>)</code>	→ <code>Serial.println</code>

¹Die Widerstände sind notwendig, da sonst die LEDs durchbrennen können!

Folgender Beispielcode skizziert die Verwendung der Serial-Bibliothek:

```
void setup()
{
  // Initialisierung der ersten seriellen UART-Schnittstelle (9600 Baud)
  Serial.begin(9600);
}

void loop()
{
  // Sofern Daten zum Lesen vorliegen ...
  while (Serial.available() > 0)
  {
    // ... 1. Lese jedes Zeichen
    char ch = Serial.read();
    // ... 2. Sende Zeichen zurück an den Sender (ECHO)
    Serial.write(ch);
  }
}
```

Entwickeln Sie einen kleinen String-Parser, welcher ein über das Eingabefeld des seriellen Monitors abgeschicktes Kommando entgegennimmt, dieses auf Korrektheit überprüft und die übermittelten Argumente als Parameter zur Steuerung einer RGB-LED verwendet.

Das Arduino Framework bietet eine eigene String-Klasse, die Sie gerne verwenden können um Ihre Lösung zu entwickeln `String()` Sie können jedoch gerne auf C (www.cplusplus.com/reference/cstring) oder C++ (www.cplusplus.com/reference/string) Funktionen zurückgreifen.

Entwerfen Sie zunächst ein Programm, das Ihnen ein Empfangen von newline-terminierten Strings über die erste serielle UART-Schnittstelle ermöglicht. Erweitern Sie Ihr Programm um Funktionalität zum Parsen des empfangenen Strings. Ihr String-Parser soll ausschließlich die beiden folgenden Befehle akzeptieren:

1). `setRGB(<red value>, <green value>, <blue value>)`

Die Parameter `<red value>`, `<green value>` und `<blue value>` sollen dabei die gewünschte Intensität für den jeweiligen Kanal darstellen und sich im Intervall von `0.0... 1.0` bewegen (`0%... 100%`).

2). `RGBon(), RGBoff()`

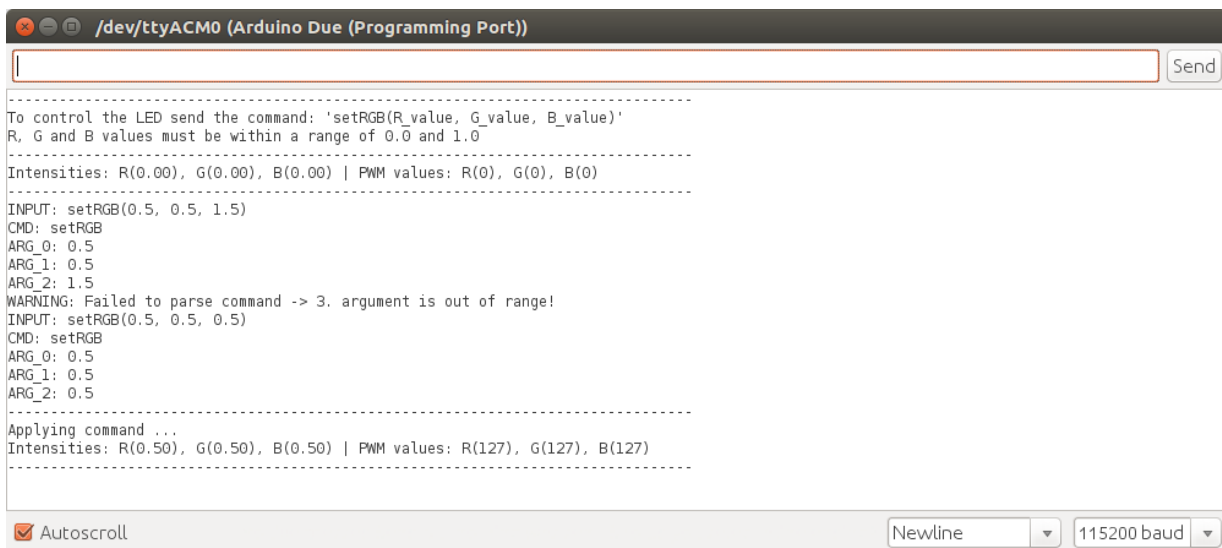
Diese beiden Befehle aktivieren die LED mit den voreingestellten Intensitäten bzw. deaktivieren sie.

Überprüfen Sie die Gültigkeit/Korrektheit der empfangenen Befehle. Achten Sie dabei auf folgende Fälle:

- Entspricht der Name des Befehls dem akzeptierten Befehl?
- Hat der Befehl die richtige Form (Klammern, Trennzeichen für die Argumente)?
- Enthält der Befehl die akzeptierte Anzahl an Argumenten?
- Befinden sich die Argumente innerhalb des spezifizierten Intervalls?

Sollte die Syntax des Befehls nicht stimmen, so benachrichtigen Sie den Benutzer **möglichst ausführlich** über den erkannten Fehler, indem Sie eine entsprechende Nachricht über die erste serielle UART-Schnittstelle an die Ausgabe des seriellen Monitors schicken.

Ist der Befehl von Ihrem String-Parser als korrekt bestätigt worden, wandeln Sie die übermittelten Argumente entsprechend um und setzen sie mittels der bereits bekannten Funktion `analogWrite()` die Intensitäten aller drei Farbkanäle. In Abbildung 3 finden Sie ein Beispiel dafür wie die Ausgabe einer Lösung aussehen könnte.



```
-----  
To control the LED send the command: 'setRGB(R_value, G_value, B_value)'  
R, G and B values must be within a range of 0.0 and 1.0  
-----  
Intensities: R(0.00), G(0.00), B(0.00) | PWM values: R(0), G(0), B(0)  
-----  
INPUT: setRGB(0.5, 0.5, 1.5)  
CMD: setRGB  
ARG_0: 0.5  
ARG_1: 0.5  
ARG_2: 1.5  
WARNING: Failed to parse command -> 3. argument is out of range!  
INPUT: setRGB(0.5, 0.5, 0.5)  
CMD: setRGB  
ARG_0: 0.5  
ARG_1: 0.5  
ARG_2: 0.5  
-----  
Applying command ...  
Intensities: R(0.50), G(0.50), B(0.50) | PWM values: R(127), G(127), B(127)  
-----
```

Abbildung 3: Serielle Kommunikation über den seriellen Monitor der Arduino IDE.