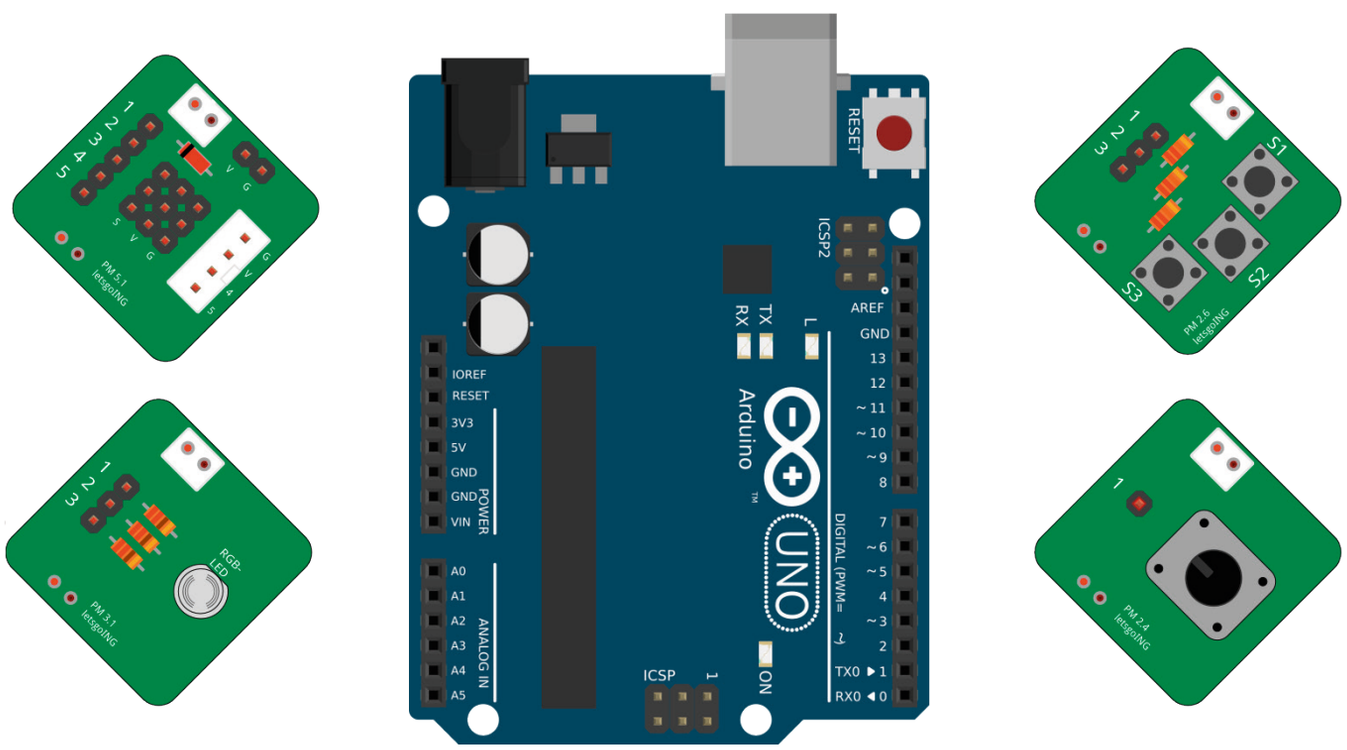
letsgoING

Mechatronik Hochschule Reutlingen

3 - Analoge Signale und Variablen



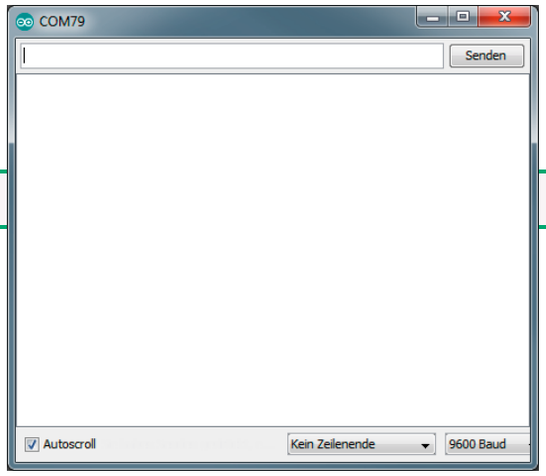
Serial Monitor • analoge Ein- & Ausgänge • analoge Variablen • PWM • Einschränken & Zuordnen • Programm • Solange

gefördert von der vector Stiftung

3.1 Der Serial Monitor

Was ist der Serial Monitor?

Der Serial Monitor ist ein Text-Ausgabefenster in der Arduino-IDE. Damit kann man sich Informationen vom Mikrocontroller anzeigen lassen und Nachrichten an ihn schicken. Der Arduino kommuniziert dabei mit seiner seriellen Schnittstelle (diese belegt dann Pin0 und Pin1) über den USB-Anschluß mit deinem PC.
Wenn Daten übertragen werden, kannst du das an den TX- (Sende-) und RX- (Empfangs-) LEDs auf deinem Arduino-Board erkennen.



Für was brauche ich den Serial Monitor?

Der Serial Monitor hat drei Hauptfunktionen:
1. das Anzeigen von Messwerten oder Programmzuständen/-ergebnissen
2. das Fehlersuchen (Troubleshooting) in deinem Programm
3. das Eingeben von Daten durch einen Nutzer

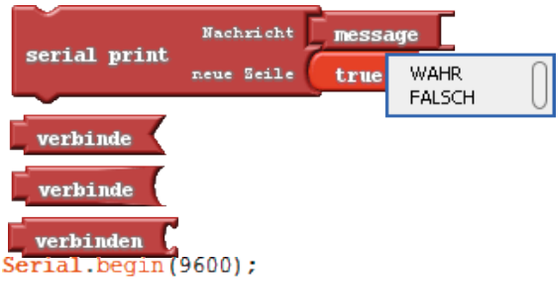
Die wichtigste Funktion des Serial Monitor in unserem Kurs ist die Fehlersuche (2.).
Um Fehler in deinem Programm zu finden, kannst du dir z.B.
- Messwerte anzeigen lassen
- Zwischenergebnisse anzeigen lassen
- ausgeben lassen, an welcher Stelle sich das Programm gerade befindet

Da das Verarbeiten von eingegebenen Daten (3.) mit dem Arduino sehr komplex ist, wird das in den Grundlagen nicht erklärt.

Wie verwende ich den Serial Monitor?

Den Block „serial print“ findest du im braunen Menü „Kommunikation“. Mit diesem Block kannst du Text und andere Informationen an den Serial Monitor übertragen. Der Block hat zwei Anschlüsse
- Nachricht → hier kannst du Text anfügen (message) und mit den „verbinde“-Blöcken auch Werte, Variablen oder Eingänge (Form beachten)
- neue Zeile → hier kannst du einstellen, ob nach der Nachricht eine neue Zeile gestartet werden soll

Um die Daten am PC angezeigt zu bekommen, musst du im ArduBlock noch auf den „Seriellmonitor“-Knopf drücken. Dann öffnet sich das Textfenster. Hier musst du noch überprüfen, ob die Übertragungsgeschwindigkeit (Baudrate) auf 9600 Baud eingestellt ist (rechts unten). Ist der Haken bei Autoscroll gesetzt, wandert die Anzeige immer mit den aktuellsten Daten.

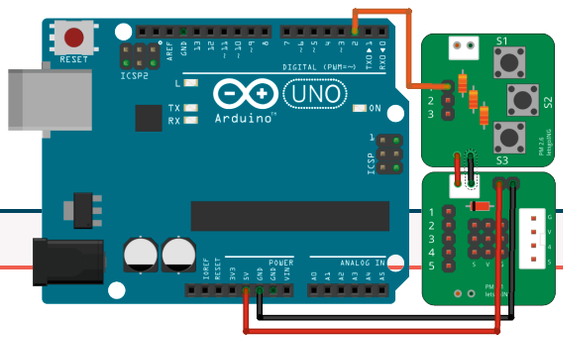



```
Serial.begin(9600);
Serial.print("message");
Serial.println();
```

Übungsaufgabe

Übung

Schreibe ein Programm, das einen Taster einliest und den Wert des Tasters auf dem Serial Monitor ausgibt.
Dabei soll in der Ausgabe erkennbar sein, was angezeigt wird (Zustand des Tasters) und die Anzeigegeschwindigkeit sollte nicht zu hoch sein.



Troubleshooting

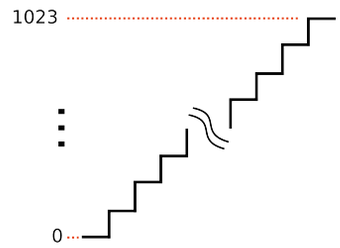
Der Serial Monitor lässt sich nicht öffnen
→ ist der Arduino angeschlossen?
→ ist der richtige COM-Port eingestellt?

Die Daten werden zu schnell angezeigt
→ Pause zwischen den „serial-Print“-Befehlen einfügen (min. 100 ms)

Es werden nur Sonderzeichen oder Leerzeichen angezeigt
→ Baudrate überprüfen → auf 9600 Baud einstellen

3.2 Der analoge Eingang

Was ist ein analoger Eingang?



Im Gegensatz zum digitalen Signal (zwei Zustände) hat ein analoges Signal unendlich viele Zwischenwerte.
In der Praxis kann ein analoger Eingang aber nicht unendlich viele Zwischenwerte erkennen. Je nach verwendetem Mikrokontroller können die meisten analogen Eingänge zwischen 256 und 65.536 Zwischenwerte unterscheiden.
Unser Arduino erkennt z.B. 1024 Zwischenwerte.

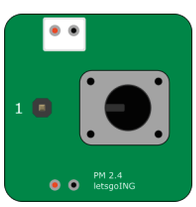
Für was brauche ich einen analogen Eingang?

Mit einem analogen Eingang kannst du nicht nur überprüfen, ob etwas an oder aus ist, sondern auch welchen Wert etwas hat. Man kann damit also etwas messen:
- wie hell ist das Licht
- wie hoch ist die Raumtemperatur
- wie weit ist etwas entfernt

Wie verwende ich einen analogen Eingang?

Um einen analogen Eingang zu lesen, nimmst du den Block „analogRead“ aus dem hellblauen Menü „Input“. Der Block gibt dann den Wert des Eingangs an einen anderen Block weiter. Dazu musst du
- den einzulesenden PIN auswählen (0-5 am Arduino → A0-A5) und
- den „analogRead“-Block an einen geeigneten Block andocken (Form beachten)

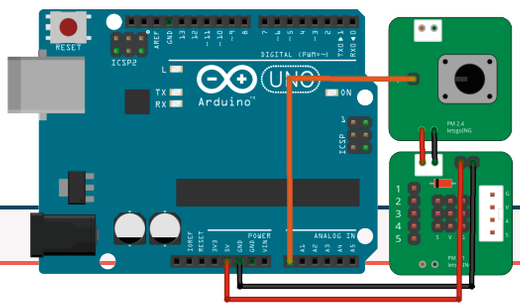
Das Potentiometer-Modul



Das Potentiometer:
Auf dem Modul befindet sich ein Potentiometer, welches als linearer Drehgeber angeschlossen ist. Das bedeutet, dass es deinem Arduino einen analogen Messwert zwischen 0 und 1023 in Abhängigkeit zum Drehwinkel zurückgibt.
- Anschlag im Uhrzeigersinn → 0
- Anschlag gegen den Uhrzeigersinn → 1023

Das Modul (PM 2.4):
Das Modul wird über das Adaptermodul mit Strom versorgt. Du kannst den Messwert an dem Pin1 abgreifen.

Übungsaufgabe



Schreibe ein Programm, welches den Wert des Potentiometers einliest und den Wert auf dem Serial Monitor ausgibt. Dabei soll in der Ausgabe erkennbar sein, was angezeigt wird (Wert des Potis) und die Anzeigegeschwindigkeit sollte nicht zu hoch sein.

Troubleshooting

Der angezeigte Wert ändert sich nicht
→ hast du im Programm denselben Pin verwendet wie am Arduino?
→ hast du das Poti-Modul an einem analogen Pin angeschlossen (A0-A5)?
ACHTUNG: analogRead(0) in ArduBlock entspricht Pin A0 am Arduino

3.3 Die analoge Variable

Was ist eine analoge Variable?

Eine analoge Variable ist ein Platzhalter, der während der Laufzeit eines Programms analoge Werte einnehmen kann. „Analog“ bedeutet, wie bei den analogen Eingängen, dass es viele Zwischenwerte gibt. Eine analoge Variable kann beim Arduino Werte von -32768 bis 32767 speichern.

Du kannst den analogen Variablen beliebige Namen geben. Mit sinnvollen Namen („Abstand“ statt „xyz“) versteht man ein Programm viel besser (→ Fehlersuche).

Für was brauche ich eine analoge Variable?

Eine analoge Variable brauchst du immer dann, wenn du einen analogen Wert zwischenspeichern willst, um ihn später weiterzuverarbeiten (z.B. für Berechnungen) oder an einer anderen Stelle im Programm auszuwerten (z.B. Vergleichen mit einem Schwellwert).
Beispiel „Automatische Straßenbeleuchtung“:
Bei einer automatischen Straßenbeleuchtung wird das Licht ab einer gewissen Helligkeit ausgeschaltet (morgens). Wird diese Helligkeit unterschritten (abends), wird die Beleuchtung wieder eingeschaltet. Dazu wird

- die Helligkeit analog gemessen
- der Messwert in eine Einheit für Licht (z.B. Lux) umgerechnet und in einer analogen Variable gespeichert
- die Variable mit einer zweiten Variable verglichen, die die Schaltschwelle enthält

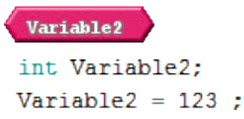
Wie verwende ich eine analoge Variable?

Wenn du einer analoge Variable einen Wert zuweisen willst, brauchst du den Block „setze analoge Variable“ aus dem roten Menü „Variablen/Konstanten“. Dort kannst du

- den Namen der Variablen angeben (violett)
- den Wert der Variablen festlegen (rot)
  oder z.B. auf den Wert eines analogen Eingangs setzen

Willst du den Wert der Variable lesen (oder vergleichen), kannst du den violetten Block klonen oder aus dem selben Menü holen. Wichtig ist dabei, dass

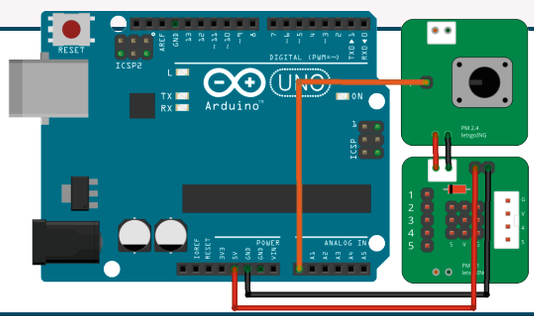
- die Namen identisch sind
- die Namen nicht nur aus einer Zahl bestehen und
- die Namen keine Leerzeichen oder Sonderzeichen enthalten



Übungsaufgabe

Übung

Schreibe ein Programm, das den Wert des Potentiometers einliest und den Wert in einer Variable speichert. Der Wert der Variable soll dann in Millivolt umgerechnet werden. Auf dem Serial Monitor sollen beide Werte ausgegeben werden.
TIPP: Messwert × 5 ≈ mV
Dabei soll in der Ausgabe erkennbar sein, was angezeigt wird (Wert des Potis und Millivolt) und die Anzeigegeschwindigkeit sollte nicht zu hoch sein.



Troubleshooting

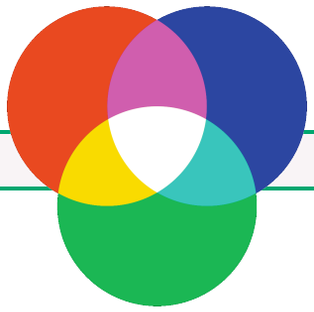
Die Werte verändern sich nicht / werden nicht übernommen

→ hast du überall exakt denselben Variablennamen verwendet? **TIPP:** Variablen-Block (violett) klonen
→ hast du immer die richtigen Blöcke verwendet? rot → Werte / violett → Variablen (-namen)

3.4 Übungsprojekt

Projektbeschreibung

In diesem Projekt soll die Bedienung der RGB-Beleuchtung aus dem Projekt 2.7 vereinfacht werden. Dabei sollen die 7 Farben über das Potentiometer eingestellt werden können und die eingestellte Farbe soll dem Nutzer angezeigt werden.



Aufgabenstellung

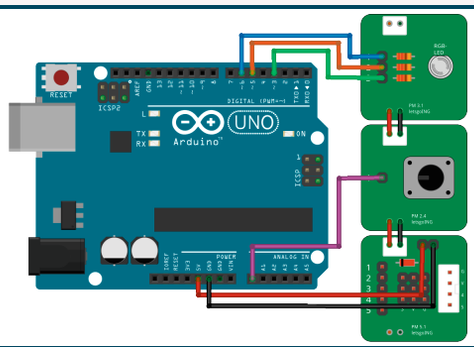
Schreibe ein Programm welches den Wert des Potentiometers ausliest und diesen Wert durch 128 teilt. Dadurch kannst du mit dem Poti die Werte 0-7 einstellen. Gebe den berechnten Wert auf dem SerialMonitor aus und teile die Faben wie folgt ein:

Farbe 0 → Beleuchtung aus
Farbe 1 → Rot
Farbe 2 → Gelb
Farbe 3 → Grün
Farbe 4 → Cyan (Türkis)
Farbe 5 → Blau
Farbe 6 → Magenta (Violett)
Farbe 7 → Weiß

TIPP: Wenn du eine Variable (z.B. „Farbe“) verwendest, kannst du mit „Falls“-Blöcken unterscheiden welcher Wert gerade eingestellt ist. → falls Farbe == 0, dann alle LEDs aus

Versuchsaufbau

Du brauchst für dieses Übungsprojekt deinen Arduino, das Adapter-Modul, das Poti-Modul und das RGB-Modul.
Verwende das Adapter-Modul um das RGB- und das Poti-Modul mit Strom zu versorgen.
Schließe jetzt alle LEDs und das Potentiometer an.
LED1 (blau) → Pin2
LED2 (rot) → Pin3
LED3 (grün) → Pin4
Poti → Pin A0



Troubleshooting

LED geht nicht an / LED geht nicht aus
→ hast du im Programm dieselben Pins für die LEDs verwendet wie am Arduino?
→ hast du die LEDs an und aus geschaltet (HIGH/LOW)

Die Poti-Werte verändern sich nicht / stimmen nicht
→ hast du im Programm denselben Pin für das Poti verwendet wie am Arduino?
→ hast du überall exakt denselben Variablennamen verwendet? TIPP: Variablen-Block (violett) klonen
→ hast du immer die richtigen Blöcke verwendet? rot → Werte / violett → Variablen (-namen)
→ stimmen die Umrechnungen der Werte? (Potiwert / 128)

Programm wird nicht hochgeladen
→ sind alle Einstellungen in der IDE richtig? (siehe Kapitel 1.3 Arduino IDE)
→ ist der Arduino eingesteckt?

Wissensfragen

1. Erkläre warum es sinnvoll ist die berechneten Werte in einer Variablen zu speichern?
2. Wie kannst du überprüfen ob eine analoge Variable einen bestimmten Wert hat?
3. Auf was musst du achten, wenn du Namen für Variablen erstellst?
4. Wie kannst du sicherstellen, dass man die Daten auf dem SerialMonitor gutlesen kann?

3.5 Die Pulsweitenmodulation

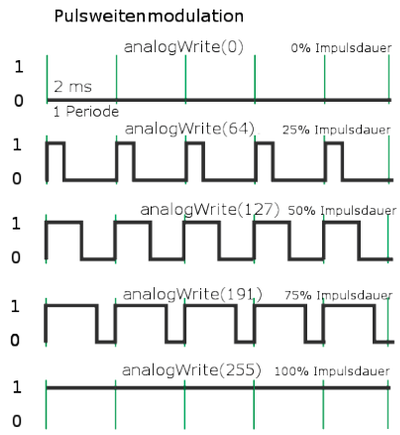
Was ist eine Pulsweitenmodulation?

Der Arduino hat nur digitale Ausgänge. Wenn du etwas nicht nur an- und aus-schalten willst, z.B. eine LED dimmen, brauchst du aber analoge Ausgänge. Die Pulsweitenmodulation, kurz PWM, ist eine Technik um mit digitalen Ausgängen einen Ausgangswert zu erzeugen, der sich ähnlich wie ein Analogwert verhält. Dabei wird der Ausgang sehr schnell an- und ausgeschaltet. Über das Verhältnis wie lange der Ausgang an und wie lange er aus ist, kann man die Zwischenwerte des Ausgangs einstellen.

Beispiele:

- Eine mit PWM angesteuerte LED wirkt durch die Trägheit unseres Auges dunkler.
  kurz an, lang aus → dunkler / lang an, kurz aus → heller
- Ein mit PWM angesteuerter Elektromotor dreht sich durch seine Massenträgheit langsamer.
  kurz an, lang aus → langsamer / lang an, kurz aus → schneller

Der Arduino hat eine Schaltfrequenz von 500Hz (2ms) und kann über die PWM 256 Zwischenwerte erzeugen.



Für was brauche ich eine PWM?

Mit Hilfe der Pulsweitenmodulation kannst du z.B. eine LED nicht nur an- und ausschalten, du kannst sie auch auf einen Zwischenwert dimmen.
Durch das sehr schnelle Ein- und Ausschalten wird dein Auge getäuscht: die LED wirkt dunkler.
Mit Hilfe von PWM kannst du auch die Geschwindigkeit eines Motors steuern.

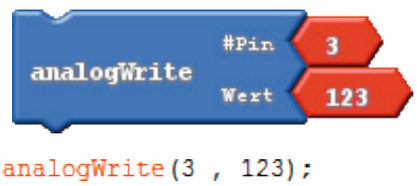
Wie setze ich einen PWM-Ausgang?

Um einen PWM-Ausgang zu setzen, nimmst du den Block „analogWrite“ aus dem dunkelblauen Menü „Output“. Dabei musst du

- den PIN auswählen (3, 5, 6, 9, 10 oder 11)
- den Wert festlegen (0 - 255) (hier kannst du auch Variablen verwenden)

Der Pin bleibt dann solange in dem eingestellten Zustand bis du ihn änderst.

ACHTUNG: nur die am Arduino mit einer Tilde (~) gekennzeichneten Pins, können als PWM-Pins verwendet werden (siehe auch oben).

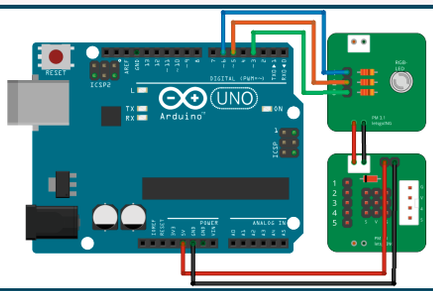



```
analogWrite(3 , 123);
```

Übungsaufgabe

Übung

Schreibe ein Programm, das alle drei LEDs in unterschiedlicher Helligkeit (PWM-Wert) ansteuert. Lasse die RGB-LED in verschiedenen Farben blinken.
Schließe die LEDs dazu so an:
LED1 (blau) → Pin6
LED2 (rot) → Pin5
LED3 (grün) → Pin3



Troubleshooting

Die LED lässt sich nicht dimmen / leuchtet nicht

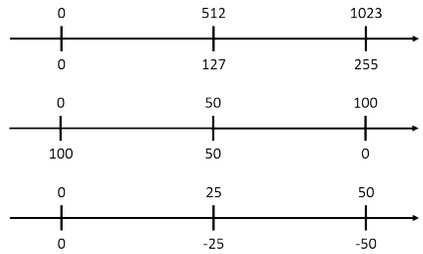
→ hast du im Programm den selben Pin verwendet wie am Arduino?
→ hast du PWM-fähige Pins verwendet? (Tilde-Symbol am Pin?)
→ liegt der eingestellte PWM-Wert zwischen 0 und 255?

3.6 Zuordnen und Einschränken

Was macht ein zuordnen-Block?

Mit dem „zuordnen“-Block kannst du Werte aus einem Wertebereich auf einen anderen skalieren. Man kann aber damit auch Wertebereiche invertieren oder negieren.

Beispiel Analoger Eingang auf PWM-Ausgang: Eingabe 0 – 1023 → Ausgabe 0 – 255
Beispiel Wertebereich invertieren: Eingabe 0 – 100 → Ausgabe 100 – 0
Beispiel Wertebereich negieren: Eingabe 0 – 50 → Ausgabe 0 – -50

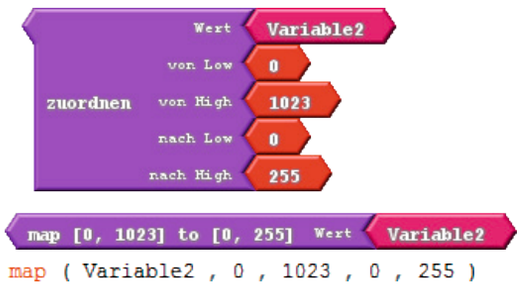


Wie verwende ich einen zuordnen-Block ?

Den „zuordnen“-Block findest du im violetten Menü „Math. Operatoren“. Wenn du ihn verwenden willst, musst du

- die Variable (den Wert) festlegen
- die untere / obere Grenze des Eingangsbereichs festlegen (von Low, von High)
- die untere / obere Grenze des Ausgangsbereichs festlegen (nach Low, nach High)

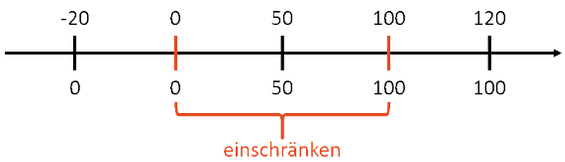
Der „map“-Block ist eine Sonderversion des „zuordnen“-Blocks. Er hat die Werte zur Umwandlung von analogen Eingängen zu PWM-Ausgängen fest eingestellt.



Was macht ein einschränken-Block?

Der „einschränken“-Block legt einen Maximalwert und einen Minimalwert fest. Der Wert bleibt innerhalb des festgelegten Bereichs unverändert. Ist er größer oder kleiner, wird der Maximal- bzw. Minimalwert zurückgegeben.

Beispiel: Wertebereich von 0 bis 100
Eingabewert 55 → Ergebnis 55
Eingabewert 120 → Ergebnis 100
Eingabewert -7 → Ergebnis 0



Wie verwende ich einen einschränken-Block?

Den „einschränken“-Block findest du im violetten Menü „Math. Operatoren“. Wenn du ihn verwenden willst, musst du

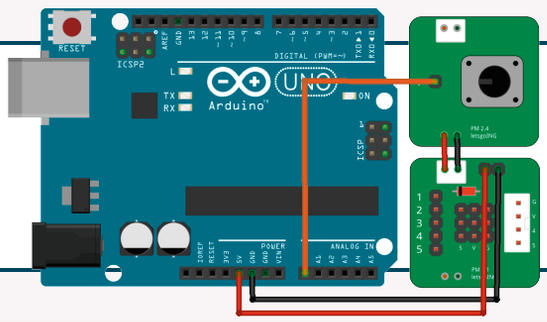
- die Variable (den Wert) festlegen
- die untere und obere Grenze des Bereichs festlegen (unterer, höherer)

TIPP:
Man kann die Blöcke auch kombinieren, sodass das Ergebnis des „zuordnen“-Blocks direkt an den „einschränken“-Block übergeben wird. Damit kann man verhindern, dass bei der Zuordung die Bereiche überschritten werden.

Übungsaufgabe

Übung

Schreibe ein Programm, das den Wert des Potentiometers einliest. Der eingelesene Wert soll dann im Bereich von 312 bis 712 in Prozent (0-100%) umgerechnet und auf dem Serial Monitor ausgegeben werden. Dabei soll in der Ausgabe erkennbar sein, was angezeigt wird (Wert des Potis und Prozent).



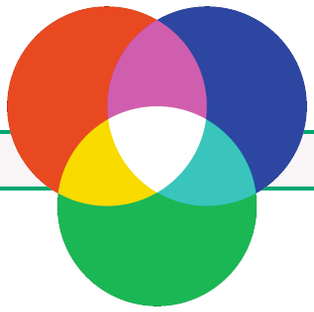
Troubleshooting

Die Werte werden falsch berechnet
→ Wird der Eingangsbereich beim „zuordnen“-Block über-/unterschritten, passiert das auch im Ausgangsbereich?

3.7 Übungsprojekt

Projektbeschreibung

Für eine RGB-Hintergrundbeleuchtung soll eine einfache Steuerung entworfen werden, mit der zwei Farben stufenlos gemischt werden können.
Dazu soll der Benutzer über ein Potentiometer die gewünschte Farbe einstellen können.



Aufgabenstellung

Schreibe ein Programm welches den Wert des Potentiometers ausliest und zwei LED-Farben deiner Wahl entsprechend ansteuert. Dabei soll:

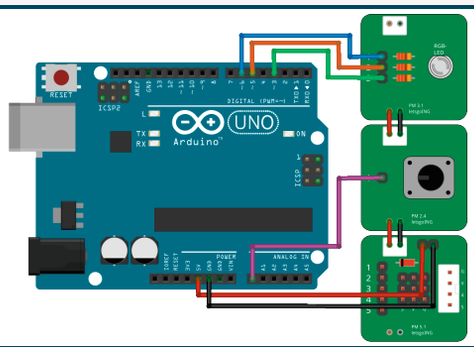
- Poti ganz links → Farbe 1 voll
- Poti ganz rechts → Farbe 2 voll
- Poti mittig → beide Farben halbe Helligkeit

TIPP:
Farbe1 = 255 - Wert_Poti_skaliert
Farbe2 = Wert_Poti_skaliert

Zusatzaufgabe:
Versuche das Programm so zu erweitern, dass alle drei Farben verwendet werden.
Dabei sollte nun in der Mittelstellung Fabe 2 voll leuchten und dann in Farbe 3 übergehen.

Versuchsaufbau

Du brauchst für dieses Übungsprojekt deinen Arduino, das Adapter-Modul, das Poti-Modul und das RGB-Modul.
Verwende das Adapter-Modul um das RGB- und das Poti-Modul mit Strom zu versorgen.
Schließe jetzt alle LEDs und das Potentiometer an.
LED1 (blau) → Pin6 Poti → Pin A0
LED2 (rot) → Pin5
LED3 (grün) → Pin3



Troubleshooting

LED geht nicht an / LED geht nicht aus
→ hast du im Programm dieselben Pins für die LEDs verwendet wie am Arduino?
→ hast du PWM-fähige Pins verwendet? (Tilde-Symbol am Pin?)
→ liegt der eingestellte PWM-Wert zwischen 0 und 255?

Die Poti-Werte verändern sich nicht / stimmen nicht
→ hast du im Programm denselben Pin für das Poti verwendet wie am Arduino?
→ hast du überall exakt denselben Variablennamen verwendet? **TIPP:** Variablen-Block (violett) klonen
→ hast du immer die richtigen Blöcke verwendet? rot → Werte / violett → Variablen (-namen)
→ stimmen die Umrechnungen der Werte? (0-1023 und 0-255)

Programm wird nicht hochgeladen
→ sind alle Einstellungen in der IDE richtig? (siehe Kapitel 1.3 Arduino IDE)
→ ist der Arduino eingesteckt?

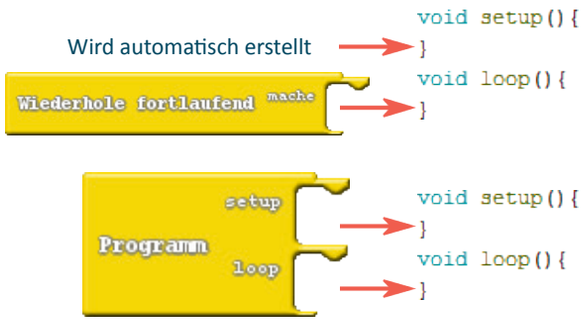
Wissensfragen

1. Erkläre warum der SerialMonitor bei der Fehlersuche so hilfreich ist.
2. Erläutere den Unterschied zwischen einem digitalen und einem analogen Eingang.
3. Was ist der Unterschied zwischen einem PWM-Ausgang und einem echten analogen Ausgang?
4. Beschreibe die Funktion des „zuordnen“-Blocks mit eigenen Worten.

3.8 Kontrollstrukturen: Programm

Was macht der Programm-Block?

Wenn du ein neues Ardublock-Programm erstellst, startest du automatisch mit dem „Wiederhole fortlaufend“-Block. Dieser Block enthält die Hauptschleife des Programms (engl. „loop“). Alle sich darin befindenden Programmteile werden immer wieder nacheinander ausgeführt.
Willst du, dass etwas zum Programmstart nur einmal ausgeführt wird, brauchst du einen zweiten Programmteil. Dieser Teil nennt sich „setup“ (vergl. Programm in der Arduino IDE). Dieser wird von dem „Wiederhole fortlaufend“-Block automatisch erstellt. Willst du selbst Programmteile in den „setup“-Teil einfügen, kannst du den „Wiederhole fortlaufend“-Block durch den „Programm“-Block ersetzen.



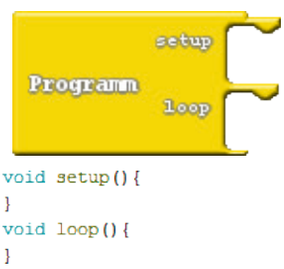
Was kann ich mit dem Programm-Block machen?

Mit dem „Programm“-Block kannst du neben dem Hauptprogramm auch den „setup“-Teil programmieren. Das musst du machen, wenn du z.B.
- Variablen einen Startwert übergeben willst
- Variablen als Konstanten für dein Programm anlegen willst (z.B. Pi = 3,14)
- deinen Pins, mit Variablen, Namen geben willst (z.B. LED_Rot = 6, Poti_Pin = 0)
- komplexere Bauteile verwenden willst, die bestimmte Einstellungen benötigen

Wie verwende ich den Programm-Block ?

Den „Programm“-Block findest du in dem gelben Menü „Steuerung“.
Der Block hat zwei Bereiche für
- „setup“ → Blöcke die einmalig (bei Programmstart) ausgeführt werden sollen
- „loop“ → das eigentliche Programm, welches sich immer wiederholt

Die Blöcke die du unter „setup“ einbaust, werden nur einmal ausgeführt. Die Blöcke unter „loop“ werden wiederholt, bis das Programm neu gestartet oder überschrieben wird.

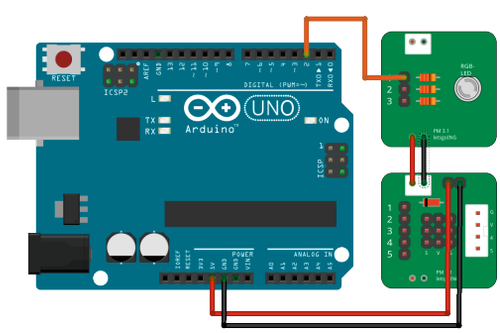


Übungsaufgabe

Übung

Schreibe ein Programm, bei dem eine LED zum Programmstart für 2s dauerhaft leuchtet und anschließend im Sekundentakt blinkt.

TIPP:
Wenn du den „Reset“-Taster am Arduino drückst, wird das Programm neu gestartet. So kannst du dein Programm mehrfach testen, ohne es jedes mal neu hochladen zu müssen.



Troubleshooting

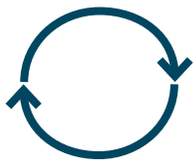
Die LED geht nicht an / geht nicht aus
→ hast du im Programm denselben Pin verwendet wie am Modul?

Die LED blinkt nur einmal im Sekundentakt, dann leuchtet sie wieder für 2s
→ hast du die Blöcke an der richtigen Position eingefügt? („setup“ / „loop“)
→ hast du den richtigen Haupt-Block verwendet? („Wiederhole fortlaufend“ oder „Programm“)

3.9 Kontrollstrukturen: Solange

Was macht ein solange-Block?

Mit dem „solange“-Block kannst du, wie mit dem „wiederhole“-Block, Programmteile wiederholen. Im Gegensatz zum „wiederhole“-Block, kannst du hier aber ein Ereignis als Bedingung angeben (z.B. Wiederhole solange, bis der Taster gedrückt wird). Man nennt einen solchen Block auch „solange“-Schleife (engl. „while-loop“).



Was kann ich mit einem solange-Block machen?

Mit dem „solange“-Block kannst du z.B.
- auf einen Tastendruck warten
- etwas ausführen solange ein Taster gedrückt ist
- ein Programmteil wiederholen, bis ein bestimmtes Ereignis passiert

Die Bedingung für die „solange“-Schleife muss digital („true“/“false“) sein und kann z.B. ein Vergleich zweier analoger Werte sein oder die Abfragen eines digitalen Eingangs.

Wie verwende ich einen solange-Block ?

Den „solange“-Block findest du in dem gelben Menü „Steuerung“. Wenn du ihn verwenden willst, brauchst du
- eine digitale Prüfbedingung (teste)
- einen Programmteil, der wiederholt wird, wenn Bedingung „true“ ist (Befehle)

Die Blöcke die du unter „Befehle“ einbaust, werden also solange wiederholt, bis die Bedingung unter „teste“ nicht mehr „true“ ist.

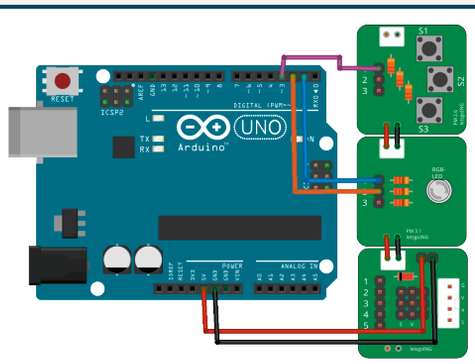

```
while (  ){
}
```

Übungsaufgabe

Übung

Schreibe ein Programm, bei dem eine LED in einer Farbe leuchtet (z.B. blau). Nachdem der Taster das erste mal gedrückt wurde, soll die LED dauerhaft blinken.

TIPP: Wiederhole solange der Taster nicht gedrückt ist.



Troubleshooting

Die LED geht nicht an / geht nicht aus

→ hast du im Programm denselben Pin verwendet wie am Modul?

Die LED blinkt nur einmal, dann muss der Taster erneut gedrückt werden

→ hast du die Blöcke an der richtigen Position eingefügt? („setup“ / „loop“)

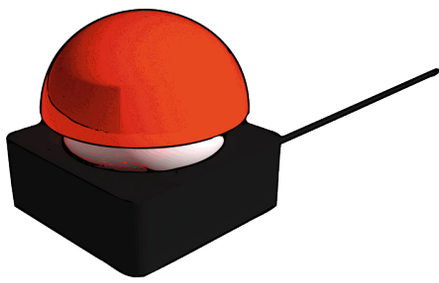
→ hast du den richtigen Haupt-Block verwendet? („Wiederhole fortlaufend“ oder „Programm“)

3.10 Grundlagenprojekt

Projektbeschreibung

Für einen Schulversuch soll die Reaktionszeit auf optische Reize gemessen werden. Dazu soll ein Aufbau erstellt werden, der folgende Funktionen beinhaltet:

- Einstellung des Schwierigkeitsgrades mit dem Potentiometer
- Start der Messung mit einem Taster
- Erzeugung einer zufälligen Wartezeit
- Wartesignal
  - blaue LED an → warten auf Zeitmessung
  - blaue LED aus → Zeitmessung gestartet
- Messung der Zeit mit einem zweiten Taster
- optische Ausgabe entsprechend der Reaktionszeit
  - grün → Reaktionszeit <= 150 ms
  - rot → Reaktionszeit >= 350 ms
- Anzeige der Reaktionszeit und des Spielstatus (Start, Messung beendet, Drücke Taster...) über den SerialMonitor

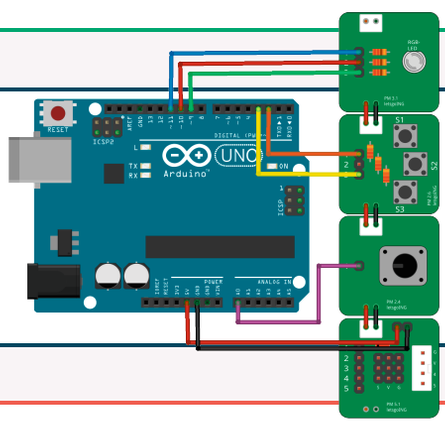


Die genaue Aufgabenbeschreibung findest du auf der nächsten Seite.

Versuchsaufbau

Du brauchst für das Grundlagenprojekt deinen Arduino, das Adapter-Modul, das Poti-Modul, das Taster-Modul und das RGB-Modul.

LED1 (blau)	→ Pin11	Poti	→ Pin A0
LED2 (rot)	→ Pin10	Taster1	→ Pin 2
LED3 (grün)	→ Pin9	Taster3	→ Pin 3



Troubleshooting

Wie kannst du überprüfen, ob alle Werte / Rechnungen korrekt sind?

→ lasse dir die Werte der Rechnungen oder Variablen auf dem SerialMonitor ausgeben
→ verwende testweise feste Werte, anstatt eingelesener Werte (z.B. analogRead) oder berechneter Werte
Beispiel: 0, 512 und 1024 anstatt analogRead
0, 127 und 255 bei analogWrite anstatt berechneter Werte

Wie kannst du überprüfen, ob der Programmablauf richtig ist?

→ gebe die jeweilige Stelle im Programm auf dem SerialMonitor aus (z.B. „Start“ oder „Erstelle Zufallszahl“, ...)
→ nutze LEDs für dich als Information (Bsp.: FALLS a==1 → LED = rot SONST → LED = grün) → geht natürlich auch mit der Onboard-LED (AN/AUS)
→ Vergleiche deinen Programmablauf anschließend mit der Aufgabenstellung
TIPP: Wenn du dir den Ablauf laut vorsagst, erkennst du selbtst leichter ob dein Ablauf stimmt

Wie kannst du überprüfen ob die Ausgaben funktionieren

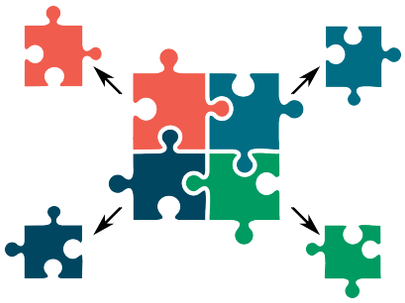
→ verwende wie oben feste Werte um die Ausgaben zu testen
TIPP: wenn du selbst Werte festlegst, ist es sinnvoll sowohl die Grenzen als auch den Hauptbereich zu testen

Wissensfragen

1. Erkläre wie du einen Programmteil nur einmalig bei Programmstart ausführen kannst.
2. Erkläre mit eigenen Worten, was in den Teilprogrammen 1-4 passiert.
3. Wodurch ändert sich die Schwierigkeit wenn man an dem Poti dreht?
4. Im Teilprogramm 3 „Zeitmessung mit einem Taster“ werden nach der gemessenen Zeit viele Nullen ausgegeben. Erkläre woher diese Ausgabe kommt und ob man den Taster hier (mit der gelernten Methode) sinnvoll entprellen kann.
5. Weshalb ist es sinnvoll erst Teilprogramme zu erstellen und einzeln zu testen?
6. Sollten die einzelnen Programmteile alle auf einmal zusammen gesetzt werden oder nacheinander? Begründe deine Entscheidung.
7. Mit welchem Block kannst du erkennen wie lange dein Programm schon läuft? Welchen Wert gibt er zurück?

3.10 Grundlagenprojekt

Vorgehensweise 1 (sehr wichtig!)



Ab einer gewissen Anzahl von Funktionen kannst du ein Projekt nicht mehr sinnvoll auf einmal umsetzen. Dann gehst du wie folgt vor:

1. Du unterteilst das Programm in überschaubare Teilfunktionen.
2. Diese setzt du um und testet sie.
3. Wenn alle getestet sind, setzt du das Programm wieder zusammen

TIPP: Du kannst die getesteten Teile aus der Hauptschleife ziehen. Dadurch werden diese Teile nicht mehr ausgeführt. So hast du am Schluss alle Teilprogramme zusammen.
→ man kann bei ArduBlock nicht aus anderen Programmen kopieren

1. Einstellen der Schwierigkeit

1. Einlesen des Poti-Werts solange Taster 1 nicht gedrückt
2. Zuordnen auf 200 – 3000
3. Speichern in einer Variable
4. Ausgabe im SerialMonitor (zum Testen)

2. Erzeugung einer zufälligen Wartezeit

1. Zufallszahl*[1] erzeugen
   (2000 als max-Wert zum Testen, im Gesamtprogramm durch Variable von Teilprogramm 1 ersetzen)
2. Zufallszahl + Mindestwartezeit (1 s) in Variable speichern
3. LED mit Wartezeit blinken lassen (zum Testen)
4. Wartezeit auf SerialMonitor ausgeben (zum Testen)

*[1] Den „Zufallszahl“-Block findest du im Menü „Math. Operatoren“. Er gibt eine zufällige Zahl zwischen 0 und dem Maximalwert (analoger Wert an max.) zurück.

3. Zeitmessung mit einem Taster

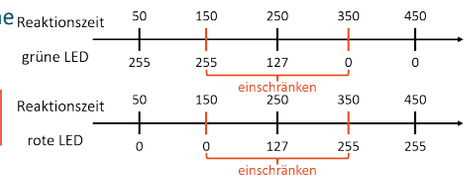
1. Variable *Startzeit* auf aktuelle Zeit*[2] setzen
2. warten solange Taster 3 nicht gedrückt
3. nach Tastendruck *Reaktionszeit = aktuelle Zeit - Startzeit*
4. Werte auf SerialMonitor ausgeben (zum Testen)

*[2] Den „millis“-Block findest du im Menü „Variablen/Konstanten“. Dieser Block gibt dir die Millisekunden seit dem Programmstart als analogen Wert zurück → aktuelle Zeit.

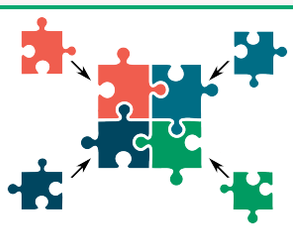
4. Reaktionszeit optisch darstellen

1. grüne LED auf *Reaktionszeit* mappen und einschränken (0-255)
2. rote LED auf *Reaktionszeit* mappen und einschränken (0-255)
3. verschiedene Werte für *Reaktionszeit* testen

(ersetzt gemessene Reaktionszeit)



Vorgehensweise 2 (sehr wichtig!)



Auch wenn man die einzelnen Funktionen schon getestet hat, ist es wichtig nicht alle Teilfunktionen sofort zusammen zu setzen.
→ Immer nur eine neue Teilfunktion hinzufügen und dann das Programm erneut testen
→ So behältst du den Überblick und erkennst besser wann/wo ein Problem/Fehler auftritt

Aufgabenstellung Teil 2: Programm zusammensetzen

Nachdem du jetzt alle komplizierteren Teile des Programms einzeln programmiert und getestet hast, kannst du jetzt das gesamte Programm zusammen bauen. Dabei sollst du folgenden Ablauf programmieren:

1. Einlesen des Schwierigkeitsgrades mit Poti, bis Taster 1 das erste mal gedrückt wird
2. Einschalten des Wartesignals (blaue LED)
3. zufällige Wartezeit erstellen und warten
4. Ausschalten des Wartesignals (blaue LED) und Zeitmessung durchführen
5. optische Ausgabe Reaktionszeit
6. warten bis Messung mit Taster 1 neu gestartet wird
7. zurück zu 2.

Ergänze den Programmablauf um die Ausgaben auf dem SerialMonitor. Zeichne dazu den Programmablaufplan rechts nach und füge die Ausgaben an den passenden Stellen ein.