# Virtuelle LED Kanäle

Virtuelle Kanäle eignen sich, um versteckte Operationen auszuführen, die man für andere Funktionen wieder abfragen kann.
So können beispielsweise eigene Zeitabläufe auf einem virtuellen LED-Kanal erfolgen, ohne dafür einen echten WS2811 in der LED-Kette zu belegen.
Den Zeitablauf kann man wiederum mit einer Variable abrufen und zum Schalten echter WS2811/12 auf jedem anderen Kanal verwenden.

## Einleitung

### Was kann man damit machen?

Das Thema hört sich im ersten Moment hochkomplex an und ohne Beispiel fehlt oft der Bezug zur Verwendbarkeit.
Daher startet dieser Beitrag mit einem Beispiel:

Im konkreten Fall sollten drei Abschnitte à acht Neonlampen eines Bahnsteigs mit kurzer Zeitverzögerung nacheinander angehen.
Starten wir mit dem Ergebnis:



Hier werden zwei Funktionen der MobaLedLib miteinander vereint.

1. Der **Pattern Configurator** gibt den zeitlichen Ablauf vor, indem er mit jeweils einer Sekunde Verzögerung die Kanäle Rot, Grün und Blau an einem virtuellen WS2811 einschaltet.
   Dieser Ablauf wird bewusst auf einen virtuellen Kanal ausgelagert, damit man diesen WS2811

nicht wirklich einlöten muss. Er existiert nur in der virtuellen Welt.
Die MobaLedLib kann im Programm jederzeit die Schaltzustände dieses virtuellen WS2811 abrufen, obwohl er physisch gar nicht existiert.

2. Das **belebte Haus** zündet innerhalb weniger Millisekunden die jeweils acht LEDs eines Bahnsteigdachs.

Vereint werden diese beiden Funktionen später mit dem Befehl LED-Werte als Variable.
Mit dieser Funktion kann man die Zustände anderer LEDs abfragen (auch die virtueller LEDs) und bei bestimmten Zuständen Aktionen auslösen. Dazu später mehr.

## Wie aktiviert man virtuelle Kanäle?

Virtuelle Kanäle kann man ganz einfach zusätzlich zu den echten Kanälen definieren. Das geht mit der Funktion Pins LED Bus definieren

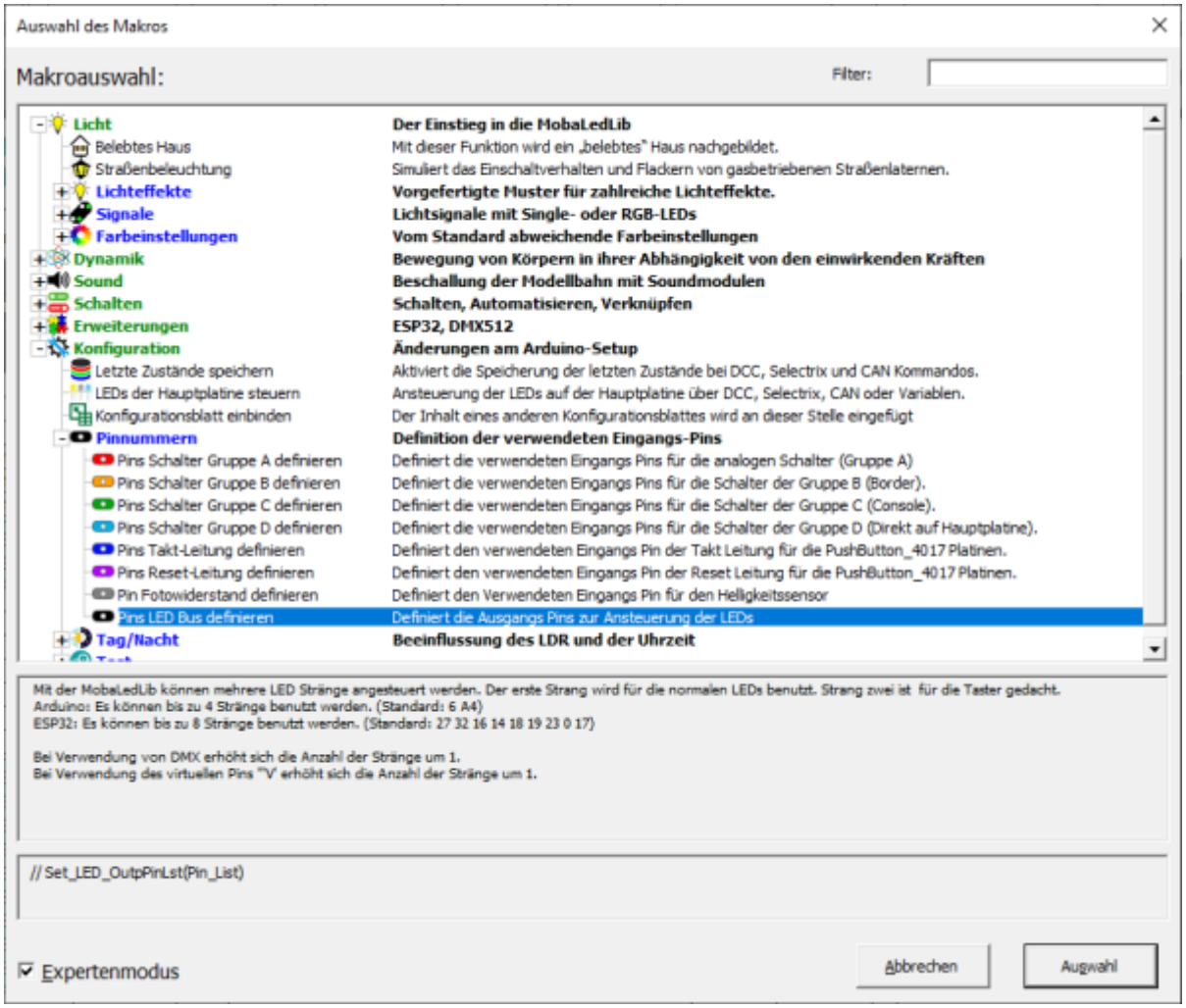
Zunächst wählt man die erste Zeile im aktuellen Excel-Sheet. Dort sollte der Befehl **Pins LED Bus definieren** stehen.

Beim **Arduino** sollte in der Spalte rechts daneben folgender Eintrag stehen: **Set_LED_OutpPinLst(6 A4)**
Die 6 steht für den digitalen Pin D6, A4 steht für den analogen Pin A4. Das sind die beiden Arduino Pins, an denen die Kanäle LED #0 und Push Button #1 der LichtMaschine Classic hängen.

Beim **ESP32/Pico** sollte in der Spalte rechts daneben folgender Eintrag stehen:
**Set_LED_OutpPinLst(27 32 16 14 18 19 23 0)**
Hier handelt es sich um die acht digitalen Pins, an denen die Kanäle LED #0 bis #7 der LichtMaschine Pro hängen.

Diese Zeile bitte per Doppelklick öffnen.

**Bei aktiviertem Expertenmodus** findet man unter Konfiguration > Pin-Nummern nun den Eintrag **Pins LED-Bus definieren**.




Für den Fall, dass alle vorhandenen LED-Kanäle verwendet werden sollen, muss man zunächst alle verwendeten oder einfach alle möglichen LED-Kanäle des jeweiligen Arduinos/ESPs definieren und zusätzlich den virtuellen Kanal mit einem V bezeichnen.
Schauen wir uns das am besten am Beispiel eines Arduinos an. Im folgenden Bild wurden der LED-Kanal 0 (Arduino-Pin: 6), der PushButtonKanal 1 (Arduino-Pin: A4), der LED-Kanal2 (Arduino-Pin: 2) und der virtuelle Kanal V extra definiert.




## Welcher Kanal ist der virtuelle Kanal?

Die Herleitung ist ganz einfach. Wie oben bereits beschrieben, liegt der LED-Kanal 0 am Arduino-Pin 6. Wenn wir im Programm Generator nach dem LED-Kanal gefragt werden, tragen wir dort standardmäßig die „0“ ein.

Für den Push-Button Kanal am Arduino-Pin A4 tragen wir die „1“ ein.
Für den zweiten LED-Kanal am Arduino-Pin 2 tragen wir die „2“ ein.

Der Programm Generator erwartet beim LED-Kanal also einfach die Nummer des Kanals, **NICHT** den Namen.
Wenn also wie in unserem Beispiel alle physischen Kanäle des Arduinos verwendet werden (LED #0, PushButton #1 & LED #2), dann ist der virtuelle Kanal eben Nummer #3.
Wird der LED-Kanal 2 des Arduinos nicht als echter Ausgang genutzt, dann wäre der virtuelle Kanal Nummer #2. Die Liste der Arduino Pins wäre dann „6 A4 V“. Ganz einfach.

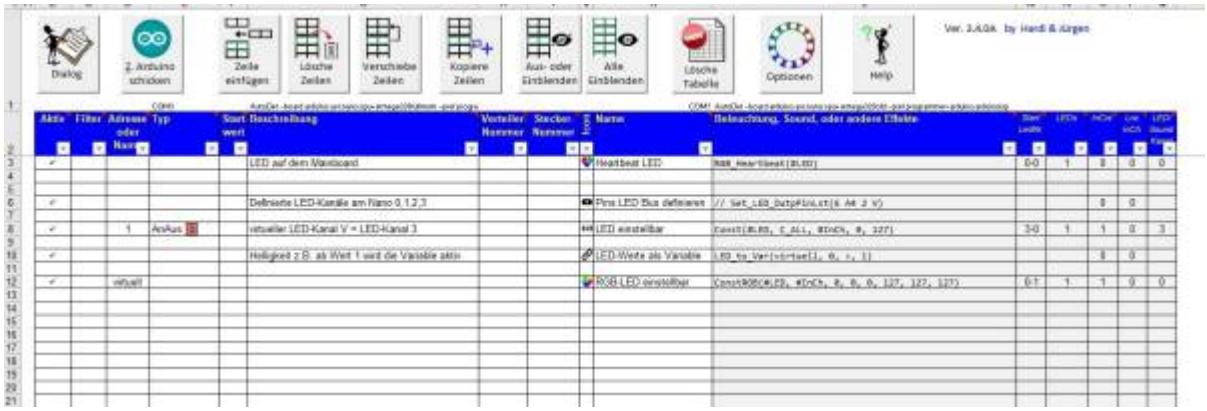
## Wie verwendet man den virtuellen Kanal?

Die Einrichtung liegt nun hinter uns. Jetzt kommt der virtuelle Kanal in einem sehr einfachen Beispiel als versteckter Schalter zum Einsatz. Es eignet sich sehr gut zum Üben, weil es zunächst auf den oben beschriebenen Zeitablauf über den Pattern Configurator verzichtet. Das Beispiel zeigt lediglich die Anwendung des virtuellen Kanals und die Verknüpfung mit dem reellen Kanal.

In der Programmierung ist der virtuelle Kanal wie oben genannt Kanal 3.
Ablauf: Mit der DCC Adresse 1 wird lediglich die virtuelle LED auf Kanal 3 eingeschaltet. Sobald der Helligkeitswert größer als 1 ist, wird die Variable „virtuell“ aktiv und die RGB-LED auf dem LED-Kanal 0 beginnt zu leuchten.

Es empfiehlt sich, dieses Beispiel einmal nachzubauen, bevor wir in den Ablauf des eingangs erwähnten Bahnsteig einsteigen.

Für den ESP gilt übrigens das Gleiche, hier müssen auch die verwendeten Kanäle und der zusätzliche V-Kanal extra definiert werden.




## Funktionen schrittweise ein- und ausschalten?

Die ganze Macht dieser Funktion wird erst sichtbar, wenn man ein Pattern mit den bekannten Effekten der MobaLedLib kombiniert, so wie beim eingangs erwähnten Bahnsteig.
Dabei ist selbst der Bahnsteig ein sehr einfaches Beispiel.

Ziel soll es sein, verschiedene Effekte (hier die Neonlampen eines belebten Hauses) in zeitlicher Abfolge (hier hintereinander) zu schalten.
Also bauen wir uns erstmal die zeitliche Abfolge:

### Schritt 1: Das Schrittschaltwerk im Pattern Configurator

Für den Bahnsteig benötigen wir drei Zustände, die ich hier zunächst in logischer Reihenfolge definiere.

Zustand 0: Der Bahnsteig ist aus. Diesen Zustand benötigen wir nur einmalig bei jedem Einschalten der Eisenbahnanlage.
Zustand 0: Ohne diesen Zustand würde der Bahnsteig bei jedem Start kurz komplett angehen und dann stufenweise ausgehen.
Zustand 1: Der Bahnsteig geht stufenweise an.
Zustand 2: Der Bahnsteig geht stufenweise aus.

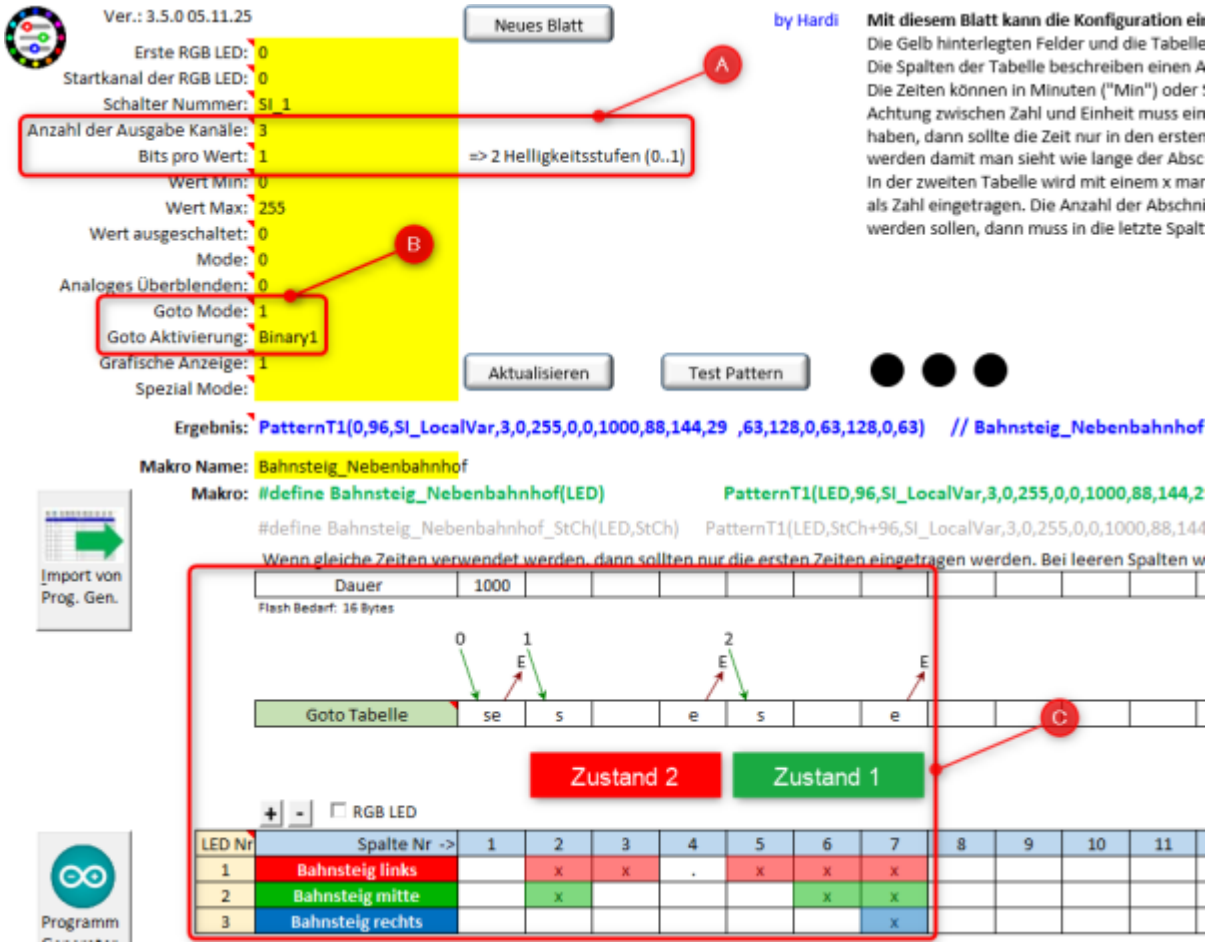
**A)** Die Anzahl der Ausgabekanäle setzen wir auf „3“, da wir drei Bahnsteigdächer schalten wollen. Dafür brauchen wir drei virtuelle Einzel-LEDs.
Bei Bits pro Wert reicht eine „1“. Das spart Speicher und wir wollen nur die Zustände „an“ und „aus“ erkennen.

**B)** Der Goto-Mode wird aktiviert, damit wir die drei oben beschriebenen Zustände per DCC-Adresse schalten können. Die Aktivierung ist Binary1. Diese unterscheidet sich von Binary dadurch, dass der Zustand 0 beim Schalten mit DCC ignoriert wird. Die DCC-Adresse schaltet mit ihren beiden Zuständen Rot und Grün nur zwischen Zustand 1 und Zustand 2 hin und her.

**C)** Wenn die DCC-Adresse auf Grün (an) geschaltet wird, wollen wir den Zustand 1 erreichen: Der Bahnsteig geht stufenweise an.
Wenn die DCC-Adresse auf Rot (aus) geschaltet wird, wollen wir den Zustand 2 erreichen: Der Bahnsteig geht stufenweise wieder aus.
Deswegen muss die Reihenfolge im Pattern Configurator wie folgt aussehen: Zustand 0 - Zustand 2 - Zustand 1 (An und Aus werden an das Verhalten von DCC angepasst).

Wenn alles richtig gemacht wurde, kann man jetzt mit dem Button „Test Pattern“ überprüfen, ob die LEDs Rot, Grün und Blau nacheinander angehen, wenn man auf die Taste „2“ klickt und ob sie nacheinander ausgehen, wenn man auf die Taste „1“ klickt.

WICHTIG ist jetzt beim Übertragen zum Programm Generator, dass man als LED-Kanal den virtuellen Kanal angibt. In unserem Beispiel also Kanal 3.

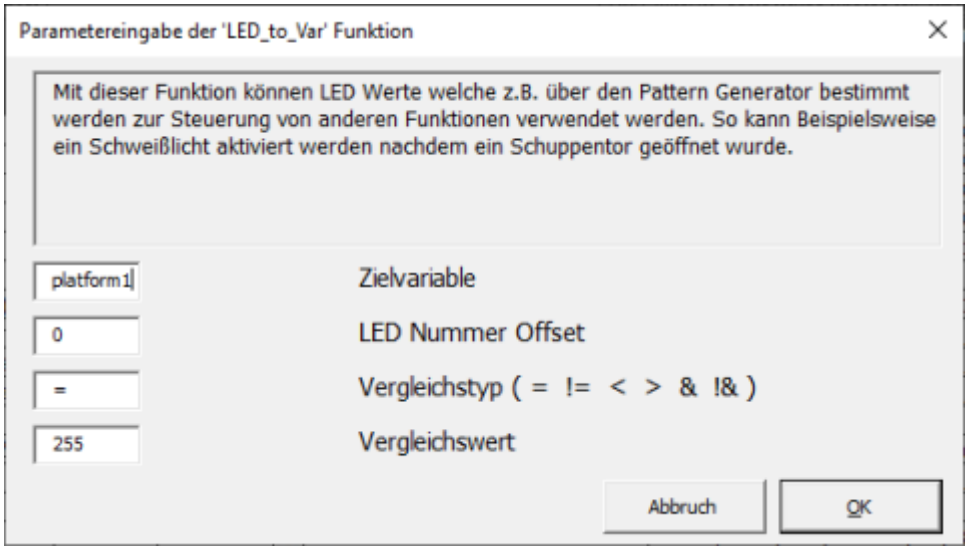



An der Stelle sind wir mit dem Pattern fertig. Das ist nun unser „Schrittschaltwerk“, dass die belebten Häuser steuert.

### Schritt 2: Die Verknüpfung zwischen virtuellem und reelem Kanal

Nun ist es an der Zeit, den Zustand der virtuellen LEDs abzufragen. Dazu bedienen wir uns der Funktion LED-Werte als Variable.
Wichtig ist, dass die Funktion direkt unter das Pattern des Bahnsteigs gesetzt wird. Die Bedienung ist dann echt einfach.
Mit dem Offset „0“ fragen wir den ersten Ausgang des vorangegangenen WS281x ab, also Rot. Mit dem Offset „1“ fragen wir den grünen Kanal ab und mit Offset „2“ den blauen.

Man kann alle möglichen Zustände abfragen (ist der Wert größer als null, ist der Wert gleich 255, ist der Wert ungleich null usw.).
Da wir im Pattern mit vollen Werten (x) gearbeitet haben, wurde in diesem Fall die Abfrage „ist gleich 255“ gewählt (größer null wäre auch gegangen).
Der Variablen-Name ist frei wählbar, er dient uns später als Ersatz für eine DCC-Adresse. Mit ihm wird das erste belebte Haus gezündet.




Die Funktion **LED-Werte als Variable** wird jetzt dreimal untereinander ausgeführt, weil alle drei Kanäle des vorangegangen (virtuellen) WS2811 ausgewertet werden sollen.
In der Tabelle kann man nun auch sehen, dass das Pattern auf Kanal 3 gelandet ist. Hier muss man aufpassen. Die Abfrage, auf welchen Kanal das Pattern programmiert werden soll , kommt nur einmal und zwar direkt beim ersten Übertragen an den Programm Generator.

| Adresse oder Name | Typ | Start-wert | Beschreibung | Verteiler-Nummer | Stecker-Nummer | Icon | Name | Beleuchtung, Sound, oder andere Ef | Start LedNr |
|---|---|---|---|---|---|---|---|---|---|
| 100 | AnAus 0 | | Bahnsteig_Nebenbahnhof (pc) | | | | Muster Pattern_Configurator | // Activation: Binary1Bin_InCh_to | 3-0 |
| | | | Rote LED als Referenz setzen | | | | LED-Werte als Variable | LED_to_Var(platform1, 0, =, 255) | |
| | | | Grüne LED als Referenz setzen | | | | LED-Werte als Variable | LED_to_Var(platform2, 1, =, 255) | |
| | | | Blaue LED als Referenz setzen | | | | LED-Werte als Variable | LED_to_Var(platform3, 2, =, 255) | |

### Schritt 3: Mit dem virtuellen Kanal schalten

Der letzte Schritt ist nun der einfachste. Es werden drei belebte Häuser mit jeweils acht Neonröhren angelegt.
Geschaltet werden die drei Häuser mit den zuvor festgelegten Variablen platform1, platform2 und platform3.
Die belebten Häuser werden selbstverständlich wieder auf Kanal 0 programmiert, da diese nicht virtuell sondern real sind.

| Adresse oder Name | Typ | Start-wert | Beschreibung | Verteiler-Nummer | Stecker-Nummer | Icon | Name | Beleuchtung, Sound, oder andere Ef | Start LedNr |
|---|---|---|---|---|---|---|---|---|---|
| platform1 | | | Bahnsteig links | | | | Belebtes Haus | HouseT(#LED, #InCh, 12, 12, 0, 0, | 0-10 |
| platform2 | | | Bahnsteig mitte | | | | Belebtes Haus | HouseT(#LED, #InCh, 12, 12, 0, 0, | 0-15 |
| platform3 | | | Bahnsteig rechts | | | | Belebtes Haus | HouseT(#LED, #InCh, 12, 12, 0, 0, | 0-19 |

Ich wünsche Euch viel Spaß und gutes Gelingen. Michael (raily74)

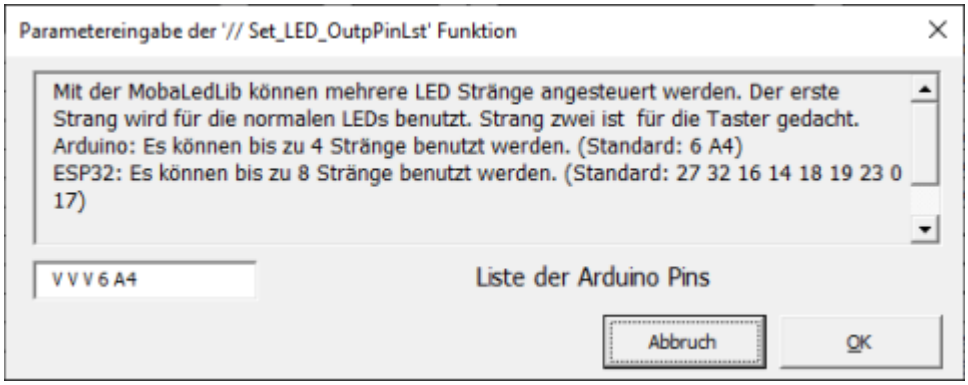
## Virtuelle Kanäle zur Manipulation der Kanal-Nummer

Virtuelle Kanäle lassen sich nicht nur hinter den physischen Kanälen platzieren sondern auch davor. Das ist eine hilfreiche Methode, wenn man beispielsweise in der Werkstatt eine Mini MLL Pro oder eine ausrangierte LichtMaschine ≤ 1.8.2 verwendet, um das aktuell gebaute Objekt direkt auf der Werkbank zu testen und an der Anlage eine LichtMaschine Pro mit acht Kanälen arbeitet.

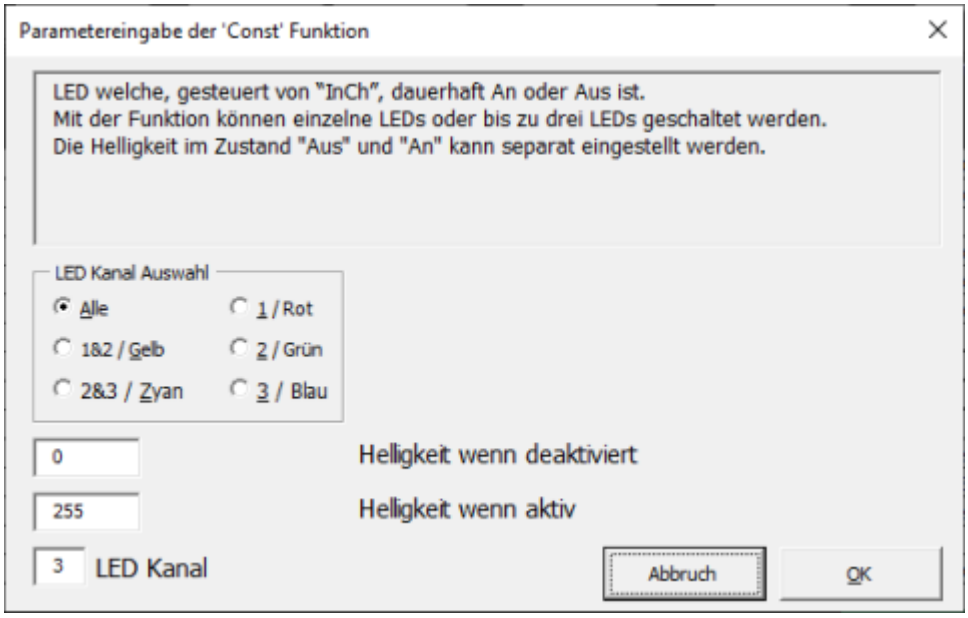
An allen Arduino-basierten Hauptplatinen kann als Kanal nur #0 oder #2 verwendet werden, die LichtMaschine Pro hingegen beherrscht acht Kanäle. Spätestens wenn das fertige Objekt von der Werkbank auf die Anlage umzieht, muss nun jeder Effekt händisch auf den gewünschten Kanal der LichtMaschine Pro angepasst werden.

Wenn man aber vorher weiß, dass man ein Haus für Kanal 3 bauen will, kann man einfach die Kanäle 0, 1 und 2 an der Arduino-basierten Hauptplatine überspringen. Da diese Kanäle aber physisch nicht existieren, muss man vor dem eigentlichen Pin 6 (Das ist Kanal 0 an der Werkstatt-Platine) drei virtuelle Kanäle setzen. Klingt kompliziert, ist aber super easy.

Wie oben beschrieben muss man nun zunächst die LED-Bus Pins einstellen. Dort gibt man jetzt für jeden Kanal, den man an der LichtMaschine Pro später überspringen will ein „V“ gefolgt von einem Leerzeichen ein, beginnend mit Kanal 0. Soll also Kanal 3 der LichtMaschine Pro verwendet werden, muss Kanal 0, 1 und 2 übersprungen werden. Das sind dann drei „V“irtuelle Kanäle. Erst dann folgen die Pins 6 (Kanal 3) und Pin A4 (Kanal 4). Die Push Buttons spricht man in dem Fall natürlich über Kanal 4 an. Baut man diese zeitgleich mit dem Haus, müsste man beim Umzug auf die Pro lediglich den Kanal des Buttons anpassen.




Die Werkbank-Platine belegt nun die Kanäle 0, 1 und 2 rein virtuell. Erst auf Kanal 3 lässt sie die Ausgabe am Pin D6 zu. Nun beginnt man mit der Programmierung des zu bauenden Objekts und definiert jeden Effekt auf Kanal 3, **beginnend mit der Heartbeat-LED der Hauptplatine!** Es folgt zum Beispiel eine konstante LED in weiß.




Obwohl man jetzt per Definition Kanal 3 verwendet, kann man in der Werkstatt alle Effekte am normalen LED-Kanal der Hauptplatine nutzen und somit testen, denn diese denkt ja, dass der

Wannenstecker Kanal 3 ist. Wenn man dann mit dem Bau des Häuschens und der Programmierung fertig ist, braucht man nur noch die entsprechenden Zeilen in das Programm der LichtMaschine Pro zu übertragen. Ohne Anpassungen, ohne Risiko und ohne zusätzliche Fehlerquellen.

From:
https://wiki.mobaledlib.de/ - **MobaLedLib Wiki**

Permanent link:
**https://wiki.mobaledlib.de/anleitungen/prog_gen/virtual?rev=1771834652**

Last update: **2026/02/23 08:17**