

Schalter und Taster einlesen mit der MobaLedLib und weitere Verbesserungen

1 Einleitung

Mit der MobaLedLib kann man Schalter oder Taster auf vier verschiedene Weisen einlesen.

Schalter ist eigentlich ein Sammelbegriff der Kippschalter und Tastschalter umfasst. Der Kippschalter ist ein Schalter der seine Position beibehält. Er ist An- oder Ausgeschaltet. Ein Taster besitzt eine Feder mit der er in die Ruheposition zurück springt, wenn er losgelassen wird. Der Kontakt ist hier nur so lange geschlossen wie der Taster betätigt wird.

In diesem Dokument bezieht sich „Schalter“ in der Regel auf beide Varianten.

Von den vier verschiedenen Varianten zum einlesen von Schaltern können drei Kippschalter und Taster einlesen. Dabei ist es egal wie viele Schalter gleichzeitig aktiv sind.

Bei der ersten und einfachsten Variante kann immer nur ein Taster (innerhalb einer Gruppe) gleichzeitig erkannt werden. Wenn mehrere Taster betätigt werden falschen Taster zurückgemeldet. Darum können mit diesem Verfahren auch nur Taster eingelesen werden.

Bei den ersten drei Verfahren können mit wenigen Leitungen viele Schalter ausgewertet werden. Das ist zum einen wichtig, weil der Arduino nur sehr wenige Anschlüsse hat. Zum anderen soll die Verkabelung der Eingabeelemente möglichst einfach sein.

Die erste und zweite Methode ist für das einlesen von Tastern am Anlagenrand gedacht. Damit können die aus dem Miniatur Wunderland bekannten „Knopfdruck Aktionen“ realisiert werden. Bei der Methode B sind die Beleuchtungen der Taster bereits mit vorgesehen. Grundsätzlich können aber alle Varianten mit beleuchteten Schaltern betrieben werden. Die LEDs der Schalter können dazu genutzt werden den aktuellen Zustand einer Funktion über Farben oder Blinkmuster zu visualisieren.

Alle vier Verfahren können gleichzeitig benutzt werden.

Die Anzahl der verwendeten Schalter kann fast beliebig erweitert werden. So ist die Auswertung von über hundert Schaltern problemlos möglich.

Alle Schalter können komfortabel mit dem Prog_Generator benutzt werden. Dazu wird einfach der Name des Schalters in der Eingangsspalte der Tabelle eingetragen. Ein Schalter kann auch mehrere Aktionen auslösen. Ein Schalter kann mehrfach bei verschiedenen Funktionen verwendet werden. Über die „Logic“ Funktion können die Schalter auch mit DCC, Selectrix oder CAN Signalen verknüpft werden so dass eine Aktion entweder vom Taster oder von der Zentrale gestartet werden kann.

Anstelle von manuell betätigten Schaltern können auf diese Weise auch Ereignisse oder Zustände auf der Moba eingelesen werden. So kann man z.B. über einen Reed Kontakt, oder eine Lichtschranke erkennen, wenn ein Zug an einer bestimmten Position ist. Wichtig ist, dass die Ereignisse einen potentialfreien Ausgang haben. Das ist bei einem Reed-Kontakt oder einem Relais immer der Fall. Bei anderen Schaltungen lässt sich die Potentialtrennung einfach über einen Optokoppler oder ein Relais nachrüsten.

2 Die verschiedenen Methoden:

Es gibt vier verschiedene Verfahren mit denen Schalter eingelesen werden:

A. **Analoge Taster:**

Die einfachste Variante ist die Kodierung von Tastern über verschiedene Widerstände. So können über zwei Leitungen und einen analogen Eingang des Arduinos 10 Taster eingelesen werden. Werden mehr Taster benötigt können weitere analoge Eingänge verwendet werden. Auf diese Weise könnten bis zu 80 Taster erfasst werden.

B. **Taster (oder Kippschalter) am Anlagenrand:**

Über eine PushButton_4017 Platine können 10 Taster eingelesen werden. Dabei können gleichzeitig betätigte Schalter erkannt werden. Mehrere dieser Platinen können entlang der Anlagenkante platziert werden. Dadurch kann die Anzahl der Taster erhöht werden und die Leitungen zwischen Tastern und Platine kurz gehalten werden. Die Anzahl der möglichen Taster ist (eigentlich) nicht begrenzt. Es können über 100 Taster eingelesen werden.

Als Eselsbrücke kann man sich „B“ = Border (Englisch Rand) merken.

C. **Weichenstellpult:**

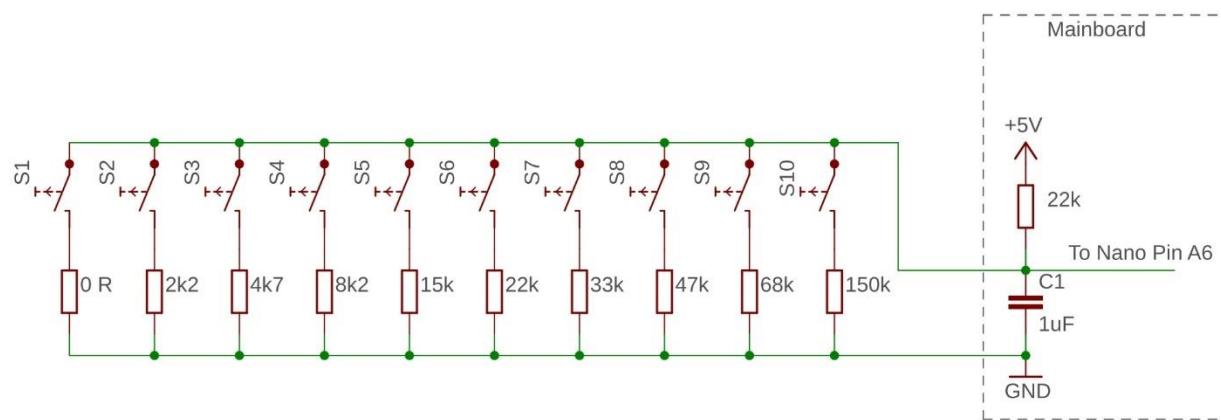
Wenn man viele Schalter an einer Stelle einlesen will, dann kann man die Variante C wählen. Hier können mit einer PushButton_4017 Platine 80 Schalter verarbeitet werden. Auch hier können mehrere Platinen kaskadiert werden. Bei dieser Variante werden 8 oder mehr Schalter gleichzeitig erfasst. Dadurch wird weniger Rechenzeit benötigt.

D. **Direkt angeschlossene Schalter:**

Diese Methode kommt ganz ohne zusätzliche Bauteile aus. (Eselsbrücke „D“ = Direkt). Sie nutzt die drei auf der Hauptplatine vorhandenen Taster. Es ist aber auch möglich externe Schalter über die vorhandenen Stecker anzuschließen. Die Anzahl der direkt an die Hauptplatine angeschlossenen Schalter kann auch erhöht werden. Allerdings wird hier für jeden Schalter ein eigener Pin des Arduinos belegt.

2.1 Methode A: Analog Taster

Bis zu zehn Taster können über eine zwei polige Leitung mit dem Arduino verbunden werden. Die beiden Leitungen werden dabei einfach von Taster zu Taster geführt. An jedem Taster sitzt ein anderer Widerstand über den der Arduino den Taster identifizieren kann. Die Bibliothek misst dazu einfach den entsprechenden Widerstand. Die Widerstände müssen entsprechend dem unten gezeigten Schaltbild gewählt werden. Der Taster 1 hat keinen Widerstand (Im Bild als 0 Ohm Widerstand dargestellt).



Zur Messung wird noch ein 22K Widerstand auf der Hauptplatine und ein $1\mu\text{F}$ Keramik Kondensator benötigt. Diese Bauteile sind bei der aktuellen Version der Platine noch nicht vorhanden und müssen über das Lochraster Feld und Kabelbrücken nachgerüstet werden:

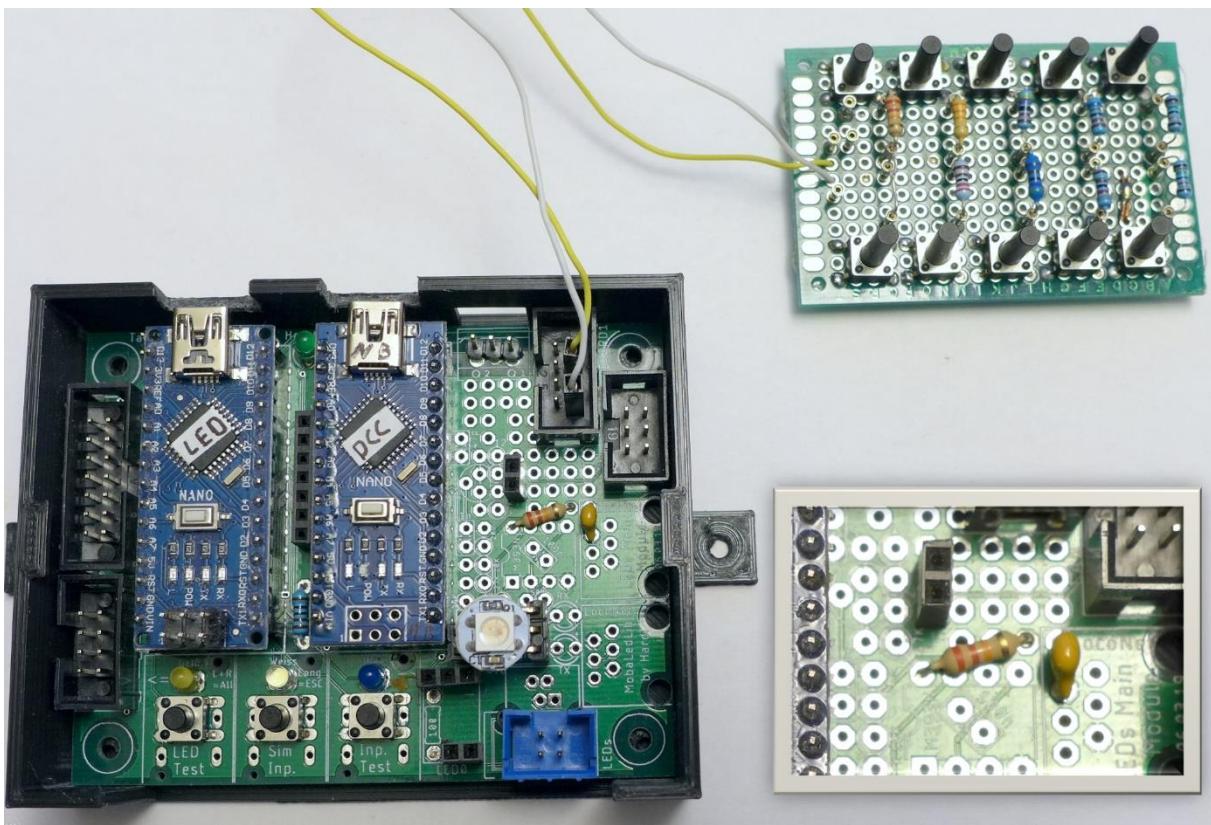


Bild: Zusätzlicher 1µF Kondensator und 22K Widerstand

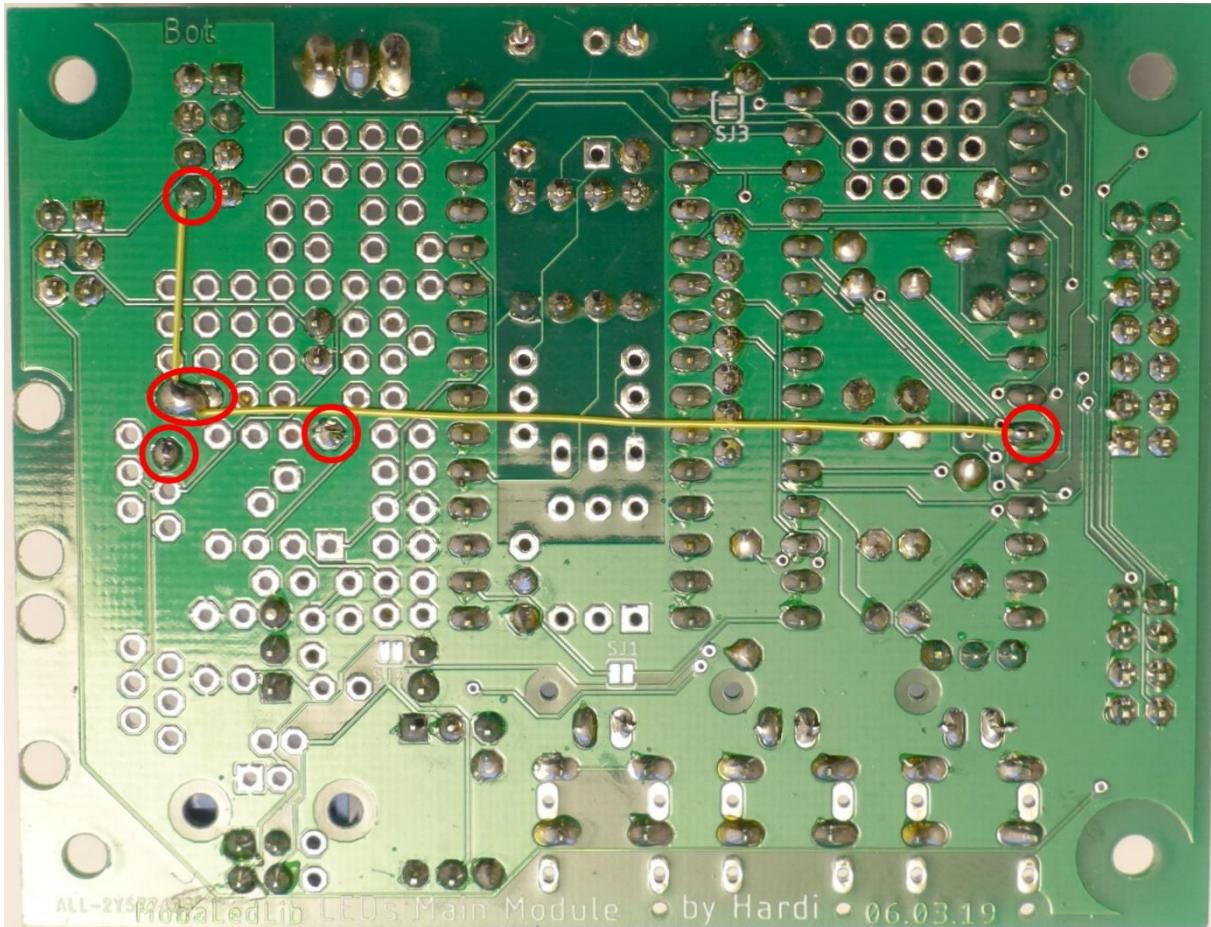


Bild: Kabelbrücken auf der Rückseite

2.1.1 Mögliche Störungen

Die Messung der Widerstandswerte kann insbesondere bei großen Werten durch elektrische Störungen anderer Leitungen beeinflusst werden. Darum sollten die Leitungen nicht zu lang sein und nicht über längere Stecken parallel zu anderen Leitungen geführt werden. Ein verdrillen der Leitungen reduziert die Störeinflüsse ebenfalls. Der 1 μF Kondensator auf der Platine dient als Filter. Er verzögert die Messung aber auch ein wenig so dass die analogen Taster nicht ganz so schnell reagieren wie die im folgenden beschriebenen Schalter. In der Praxis stört das aber nicht.

2.1.2 Verwendung im Excel Programm

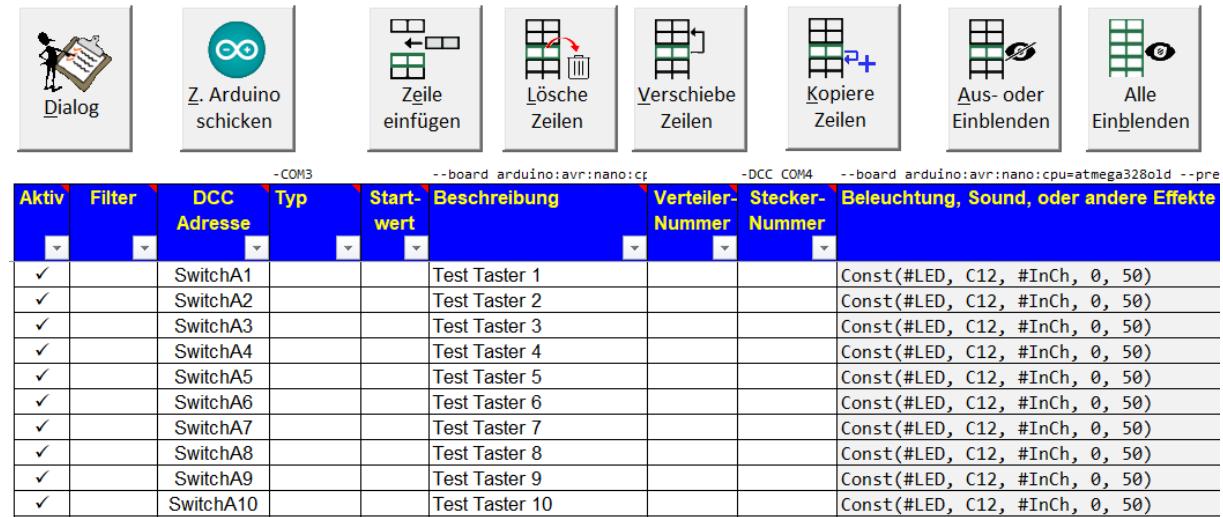
Im Excel Programm werden die analogen Schalter über den Namen „SwitchA“ gefolgt von einer Nummer angesprochen. Der erste Schalter hat dabei die Nummer 1.

Beispiel: SwitchA1

Wenn mehr als 10 Schalter verwendet werden sollen, dann können weitere analoge Eingänge des Arduinos zum einlesen von Tastern genutzt werden. Dazu müssen der Widerstand und Kondensator nachgerüstet werden. Über den Befehl „Set_SwitchA_InpLst()“ welcher im Expert Modus des Prog_Generators verfügbar ist können weitere analoge Eingänge definiert werden.

Beispiel: // Set_SwitchA_InpLst(A6, A5, A4)

Mit dieser Konfiguration können die Taster getestet werden:



Aktiv	Filter	DCC Adresse	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchA1			Test Taster 1			Const(#LED, C12, #InCh, 0, 50)
✓		SwitchA2			Test Taster 2			Const(#LED, C12, #InCh, 0, 50)
✓		SwitchA3			Test Taster 3			Const(#LED, C12, #InCh, 0, 50)
✓		SwitchA4			Test Taster 4			Const(#LED, C12, #InCh, 0, 50)
✓		SwitchA5			Test Taster 5			Const(#LED, C12, #InCh, 0, 50)
✓		SwitchA6			Test Taster 6			Const(#LED, C12, #InCh, 0, 50)
✓		SwitchA7			Test Taster 7			Const(#LED, C12, #InCh, 0, 50)
✓		SwitchA8			Test Taster 8			Const(#LED, C12, #InCh, 0, 50)
✓		SwitchA9			Test Taster 9			Const(#LED, C12, #InCh, 0, 50)
✓		SwitchA10			Test Taster 10			Const(#LED, C12, #InCh, 0, 50)

Damit wird jeden Taster eine LED zugewiesen. Die LED leuchtet solange der Taster betätigt ist und verlischt sobald der los gelassen wird. Diesen Test sollte man unbedingt zu Beginn einmal durchführen damit sichergestellt ist, dass alle Schalter funktionieren und die richtigen Widerstandswerte verwendet wurden.

Anstelle der LED könnte man hier auch einen Sound Befehl verwenden. So könnte man verschiedene Geräusche per Taster abrufen.

Zur Aktivierung von länger andauernden Effekten steuert man mit dem Taster z.B. eine „PushButton“ Funktion. Diese werden in einem Eigenen Kapitel erklärt.

2.2 Methode B: Schalter am Anlagenrand

Im Miniatur Wunderland findet man an vielen Stellen Taster mit denen der Besucher bestimmte Aktionen per Knopfdruck auslösen kann. Das macht allen Besuchern viel Spaß. Dieses Konzept kann

mit der MobaLedLib einfach auf der eigenen Anlage umgesetzt werden. Die Taster können mit LEDs beleuchtet werden welche über Blinksequenz oder Farben den Zustand wiedergeben.

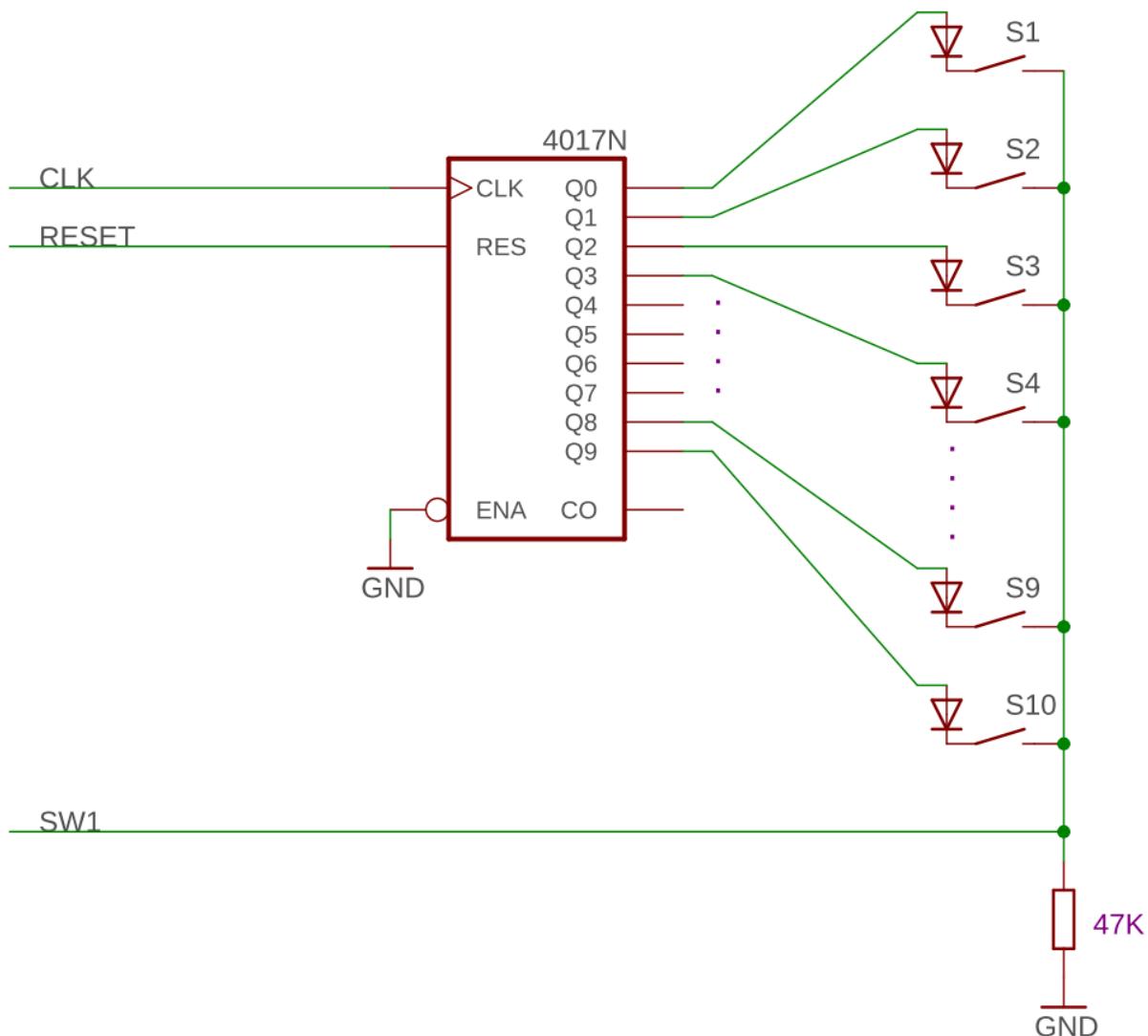
Über nur acht Kabel können fast beliebig viele Taster ausgewertet werden. Über das gleiche Flachkabel werden auch die LEDs der Taster angesteuert. Bis zu 10 Taster werden an eine PushButton_4017 Platinen angeschlossen. Mehrere dieser Platinen können hintereinandergeschaltet werden, wenn mehr Taster benötigt werden.

Wenn die Hauptplatine zentral positioniert ist, können zwei Flachkabel Stränge verwendet werden. Einer deckt die linke Seite der Anlage ab, der zweite die rechte Seite. Dazu wird der erste Strang an den Stecker „KEYBRD“ und der zweite Strang an „KEYBRD1“ angeschlossen.

2.2.1 Prinzip der Schaltung

Das einlesen vieler Schalter über wenige Leitungen wird dadurch erreicht, dass die Schalter zeitlich nacheinander abgefragt werden. Das geschieht aber so schnell, dass eine Reaktion sofort erfolgt, wenn ein Schalter betätigt wird. Das Programm fragt dazu alle angeschlossenen Kanäle innerhalb von 100 Millisekunden einmal ab. Die Abfragefrequenz wird dabei automatisch an die Anzahl der Taster angepasst.

Zur Auswahl des abzufragenden Schalters wird das IC CD4017 verwendet.



Dieser Baustein ist ein Zähler welcher seine Ausgänge nacheinander aktiviert. Zu Beginn liegt am Ausgang Q0 +5V an. Wenn Schalter S1 aktiv ist, dann fließt über die Diode und den Schalter ein Strom und am Ausgang der Schaltung SW1 liegt ebenfalls 5V an. Das wird von Arduino eingelesen und abgespeichert.

Im nächsten Schritt wird Q1 aktiviert. Q0 ist jetzt wieder abgeschaltet. Jetzt kann S2 erfasst werden. Wenn auch dieser Schalter geschlossen ist, dann fließt über seine Diode ein Strom und das kann wiederrum vom Arduino erkannt werden. Die Dioden verhindern dabei, dass das Signal von einem Ausgang rückwärts in einen anderen Ausgang fließen kann. Das würde passieren, wenn zwei Schalter gleichzeitig angeschaltet sind.

Den Befehl zum Umschalten schickt der Arduino über die „CLK“ Leitung zum Zähler (4017). Der Reset Anschluss wird dazu verwendet, dass der Zähler nach wieder bei 0 beginnt. Das Programm aktiviert diesen Ausgang zu Beginn einer Messung.

Die Erfassung aller Schalter geschieht so schnell, dass der Benutzer keine Verzögerung beim Betätigen eines Schalters bemerkt. Dazu werden alle angeschlossenen Schalter innerhalb von 100 Millisekunden einmal abgefragt.

Mit dieser Schaltung kann man über drei Anschlüsse eines Arduinos zehn Schalter einlesen. Dabei können beliebig viele Schalter gleichzeitig eingeschaltet sein.

2.2.2 LEDs in den Tastern

Der Besucher soll erkennen können ob eine Aktion gestartet werden kann oder bereits läuft. Im Miniatur Wunderland befinden sich neben den Tastern zwei LEDs über die der Zustand der Aktion angezeigt wird. Das ist gerade bei länger andauernden Aktionen nützlich, weil der Besucher daran erkennen kann ob er den Taster drücken kann oder ob er noch warten muss bis die Attraktion wieder bereit ist. Die Anzeige ist als Feedback wichtig.

Bei der MobaLedLib können die LEDs noch mehr. Sie können so konfiguriert werden, dass sie beliebige Farben, Helligkeiten und Blinkmuster wiedergeben. Das kann Beispielsweise so genutzt werden:

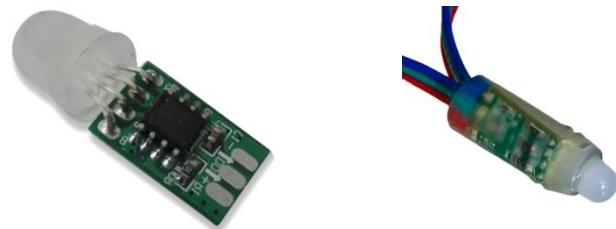
- Im Ausgeschalteten Zustand Leuchtet die LED schwach damit man den Schalter im abgedunkelten Raum findet
- Wenn die Aktion aktiv ist dann Blinkt die LED
- Bei Aktionen mit mehreren Zuständen blinkt die LED unterschiedlich oft. Wenn die Taste einmal gedrückt wird leuchtet die LED einmal kurz gefolgt von einer längeren Pause. Mit dem zweiten Druck auf den Taster Blinkt sie zweimal gefolgt von einer Pause. Auf diese Weise können bis zu 5 Zustände visualisiert werden.

Dieses Verhalten wird in der Konfiguration über den Befehl „PushButton_w_LED_BL_0_x“ generiert. „x“ steht dabei für eine Zahl zwischen 1 und 5. Die Zahl entspricht der Anzahl der Zustände.

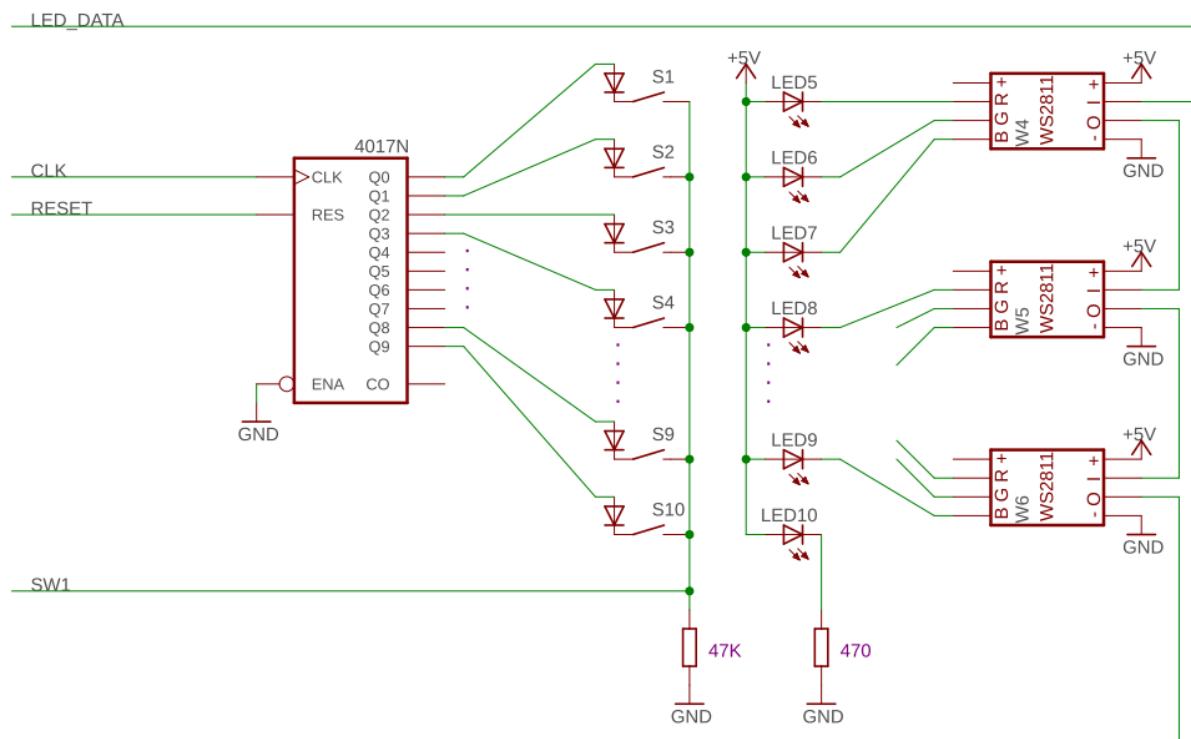
Zur Anzeige können entweder Taster mit eingebauter LED verwendet werden:



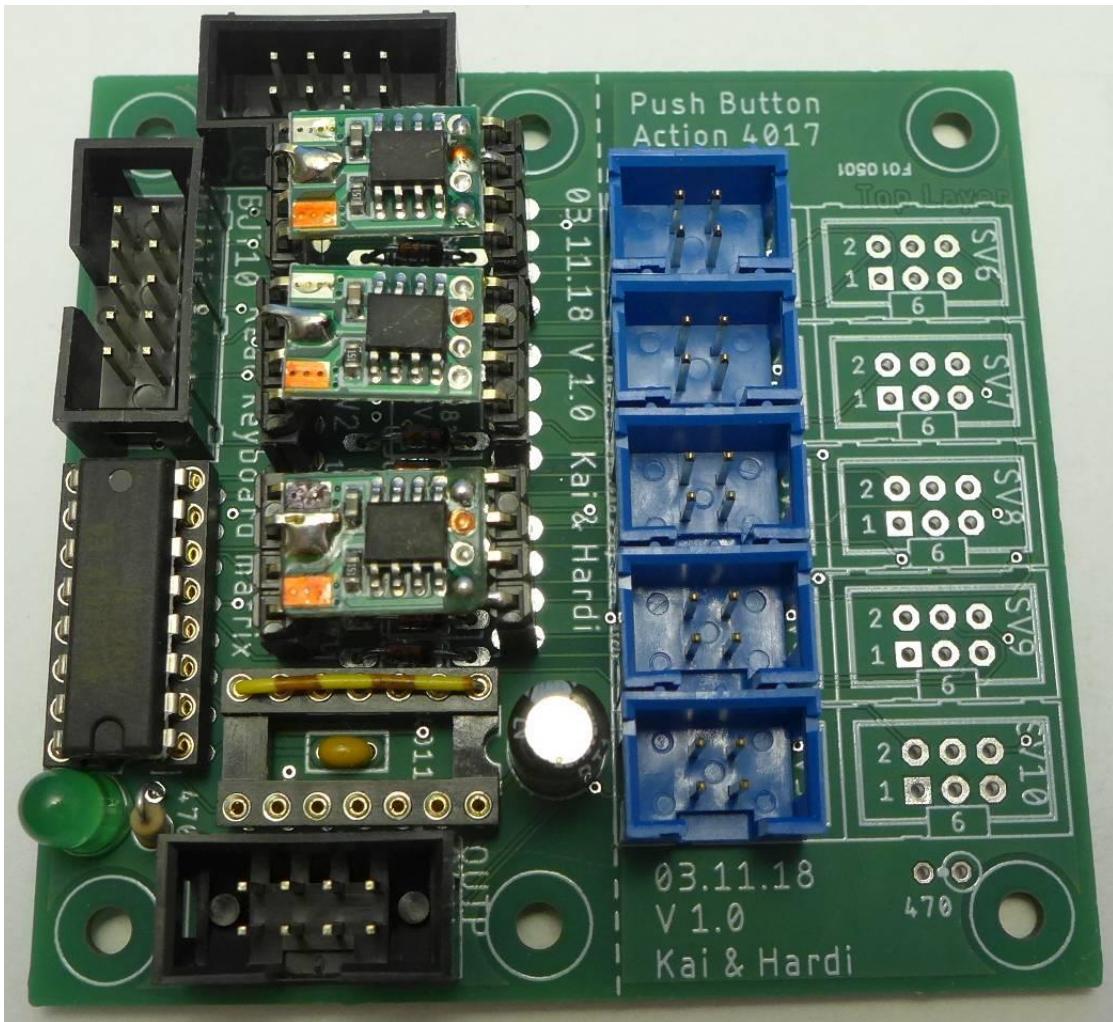
Oder separate (RGB) LEDs verwendet werden:



Auf der PushButton_4017 Platine sind bereits Steckplätze für 3 WS2811 Module vorgesehen. Im Schaltbild sieht das so aus:



Damit werden jetzt 4 Signal Leitungen zum Arduino benötigt. Dazu kommen noch zwei Leitungen für die Versorgungsspannung. Auf der Platine werden 8-polige Stecker verwendet welche oben und unten auf der folgenden Platine zu sehen sind:



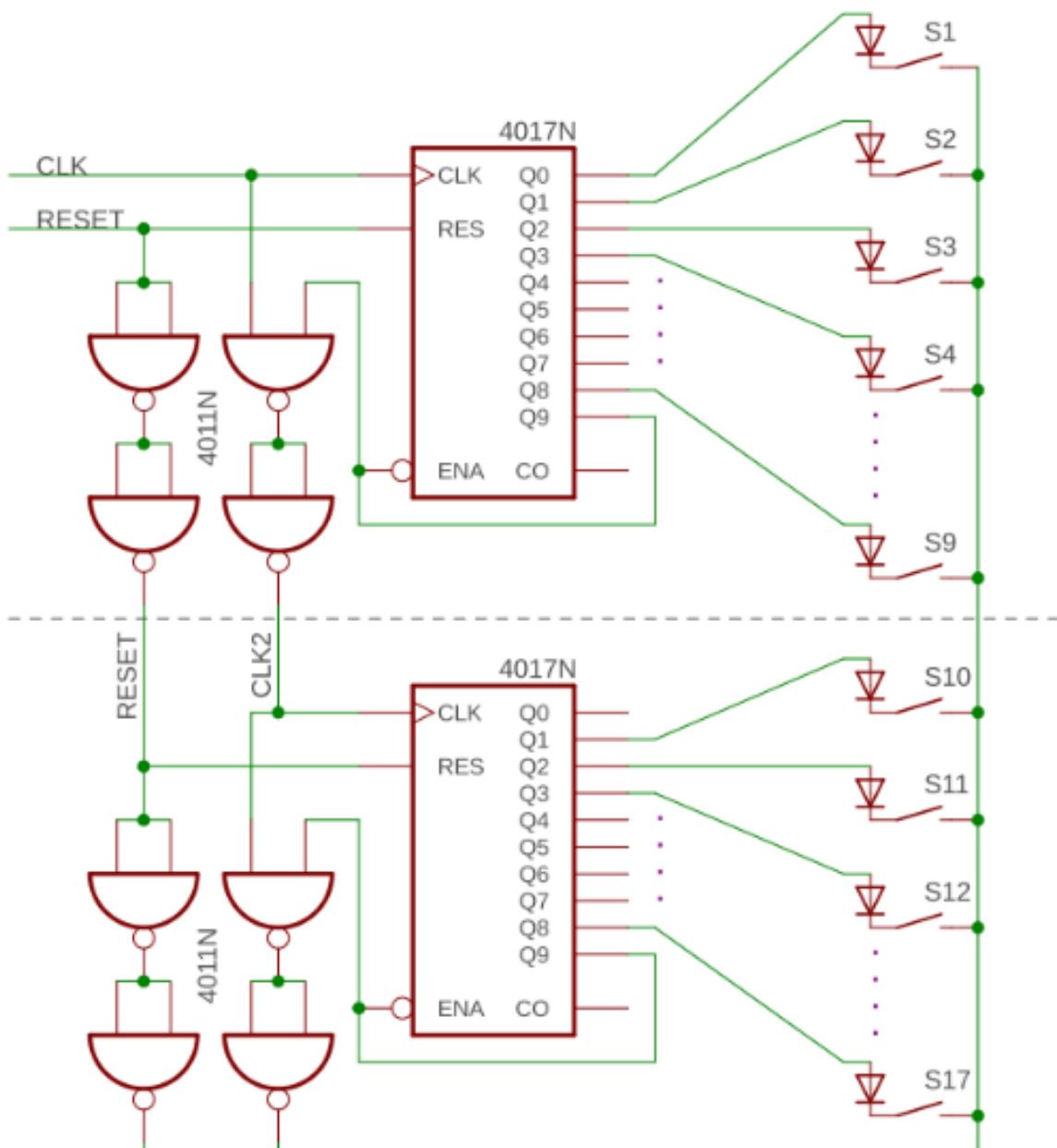
Der Stecker auf der oben ist der Eingang. Er wird über ein Flachkabel mit der Hauptplatine verbunden. An den Stecker an unteren Rand (OUTP) kann eine weitere PushButton_4017 Platine angeschlossen werden. Damit hat kann man die Anzahl der Taster (fast) beliebig erweitern. Das ist ein ganz besonderes Feature der Schaltung.

In dem Beispiel sind 5 der insgesamt 10 möglichen Stecker zum Anschluss der Taster Bestückt. Die Idee dahinter ist, dass man diese Platinen an verschiedenen Stellen am Rand der Modellbahn platziert und über ein 8-poliges Kabel untereinander verbindet. An jede Platine werden dann mit kurzen Kabeln die Taster angeschlossen. Die Taster können über 4-polige oder 6-polige Buchsen angesteckt werden. Die 4-Polige Variante kann verwendet werden, wenn man nur eine (im Schalter eingebaute) LED verwendet. Sollen RGB LEDs verwendet werden, dann muss ein 6-poliger Wannenstecker benutzt werden.

2.2.3 Kaskadierung der Taster Platinen

Zur Verwendung mehrere PushButton_4017 Platinen wird ein weiteres IC (CD4011) benötigt. Dieser Baustein enthält vier NAND-Gatter. Das sind Komponenten mit denen zwei Eingängen UND Verknüpft werden und das Ergebnis anschließend Invertiert wird. Damit wird das Takt Signal vom Arduino erst dann weitergegeben wie Schalter auf der ersten Platine eingelesen sind.

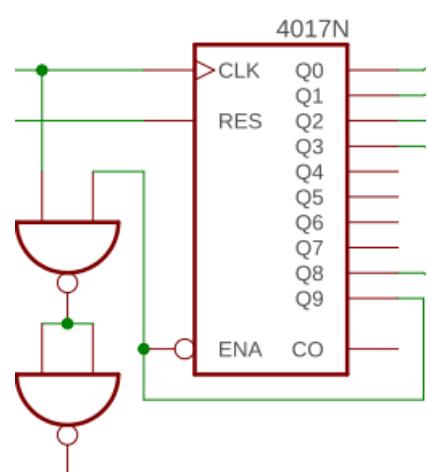
Im Schaltbild sieht das so aus:



2.2.4 Funktionsweise:

Das erste NAND-Gatter verknüpft das Taktsignal mit dem Ausgang Q9. Ein UND Gatter liefert erst dann eine 1 an seinem Ausgang, wenn beide Eingänge 1 sind. Damit wird das Taktsignal erst dann weitergeleitet, wenn Q9 1 ist. Das ist nach 9 Impulsen am Eingang des Zählers der Fall. Die Rückführung des Signals Q9 auf den Enable (ENA) Eingang des Zählers sorgt dafür, dass der Zähler gestoppt wird, wenn Q9 aktiv wird. Das Taktsignal wird über ein zweites Gatter des Bausteins invertiert und dann an den nächsten Zähler weitergegeben.

Die ersten 9 Kanäle werden damit vom ersten 4017 abgedeckt. Danach übernimmt der zweite 4017. Der Ausgang Q9 des ersten Bausteins bleibt so lange High bis die Zähler über den Reset Eingang zurückgesetzt werden. Das



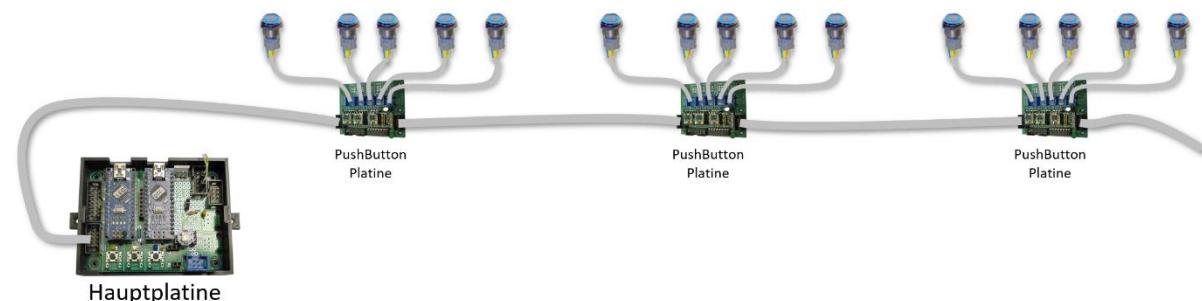
bedeutet aber auch, das mit der ersten Platine nur 9 Schalter eingelesen werden können, wenn ein zweites Modul folgt. Darum ist es auch nicht weiter schlimm, dass die PushButton_4017 Platine nur 9 steuerbare LED-Ausgänge hat (Aufmerksamen Lesern ist vielleicht aufgefallen, dass die LED10 direkt über einen 470 Ohm Widerstand mit Masse verbunden ist und nicht über einen WS2811 gesteuert werden kann.).

Die beiden linken NAND-Gatter invertieren das Reset Signal zweimal und verstärken das Signal dadurch lediglich.

Beim folgenden Zähler kann der erste Kanal ebenfalls nicht genutzt werden, weil dieser so lange aktiv ist bis die Schalter des vorangegangenen Moduls eingelesen sind. Erst mit dem 10. Taktimpuls wird der Ausgang Q1 des folgenden 4017 aktiviert. Im Schaltplan erkennt man, dass der Taster S10 am Pin Q1 des Zählers angeschlossen wird.

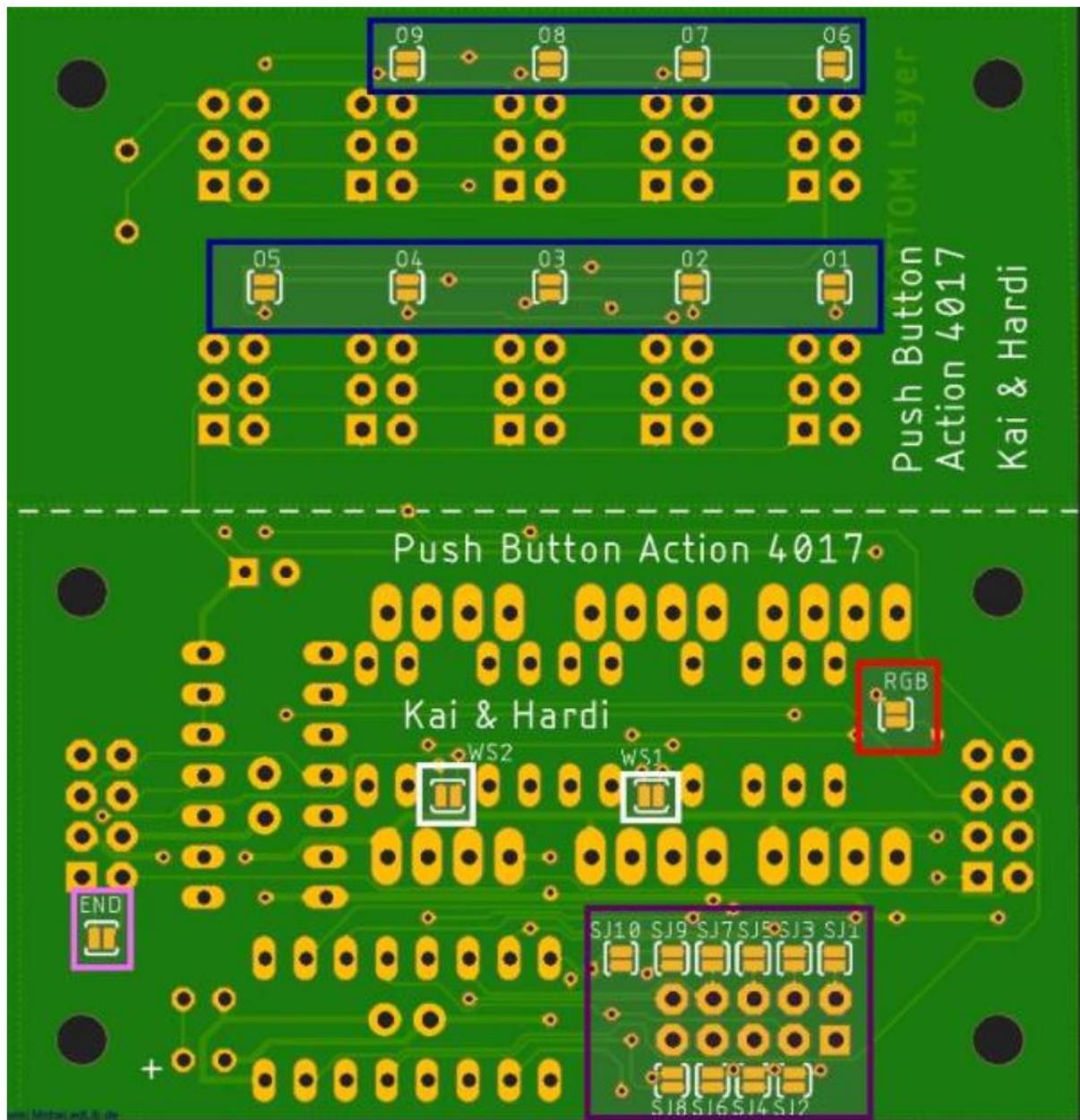
Wenn wie rechts gezeigt eine weitere Platine folgt dann entfällt auch der 9. Ausgang. Eine Mittlere Platine kann darum nur 8 Schalter auswerten. Am besten man bestückt die erste und letzte Taster Buchse nicht damit Fehlbedienungen ausgeschlossen sind. Wenn diese Kanäle versehentlich benutzt werden dann werden fälschlicherweise mehrere Taster als gedrückt gemeldet (Bei Q9 werden alle folgenden Taster gemeldet, Bei Q0 alle Vorangegangenen.)

Die einzelnen Platinen und die Schalter werden ganz einfach über Flachkabel und Schneid/Klemm Stecker miteinander verbunden.



2.2.5 Taster Anzahl reduzieren

Wenn man nicht alle verfügbaren Kanäle einer Platine benutzen will, weil keine weiteren „Knopf Druck Aktionen“ in der Nähe vorgesehen sind kann man die Anzahl der abgefragten Schalter über Lötbrücken auf der Unterseite der Platine begrenzen. Das hat den Vorteil, dass Rechenzeit des Prozessors und Eingangskanäle gespart werden. Dazu setzt man einen der Lötjumper SJ1 – SJ10:



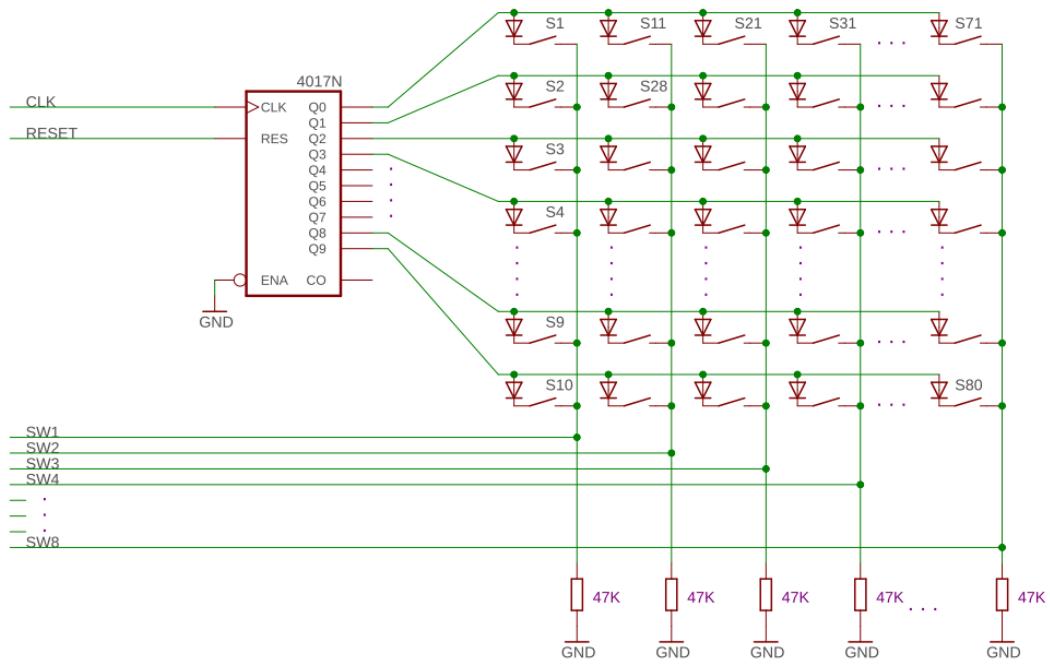
Achtung: Es muss immer mindestens ein Jumper gesetzt werden. Auch wenn nur eine Platine verwendet wird. Bei der ersten Version der Platine (V1.0) wurde der Jumper SJ10 vergessen. Dieser ist in dem ersten Bild der Platine den gestreiften Draht ersetzt. Er ist nur in dieser Version nötig.

Im Wiki findet man eine ausführliche Anleitung dazu:

https://wiki.mobaledlib.de/anleitungen/bauanleitungen/push_button_action_300de#beispiele_fuer_die_loetjumper_sj1_-_sj10

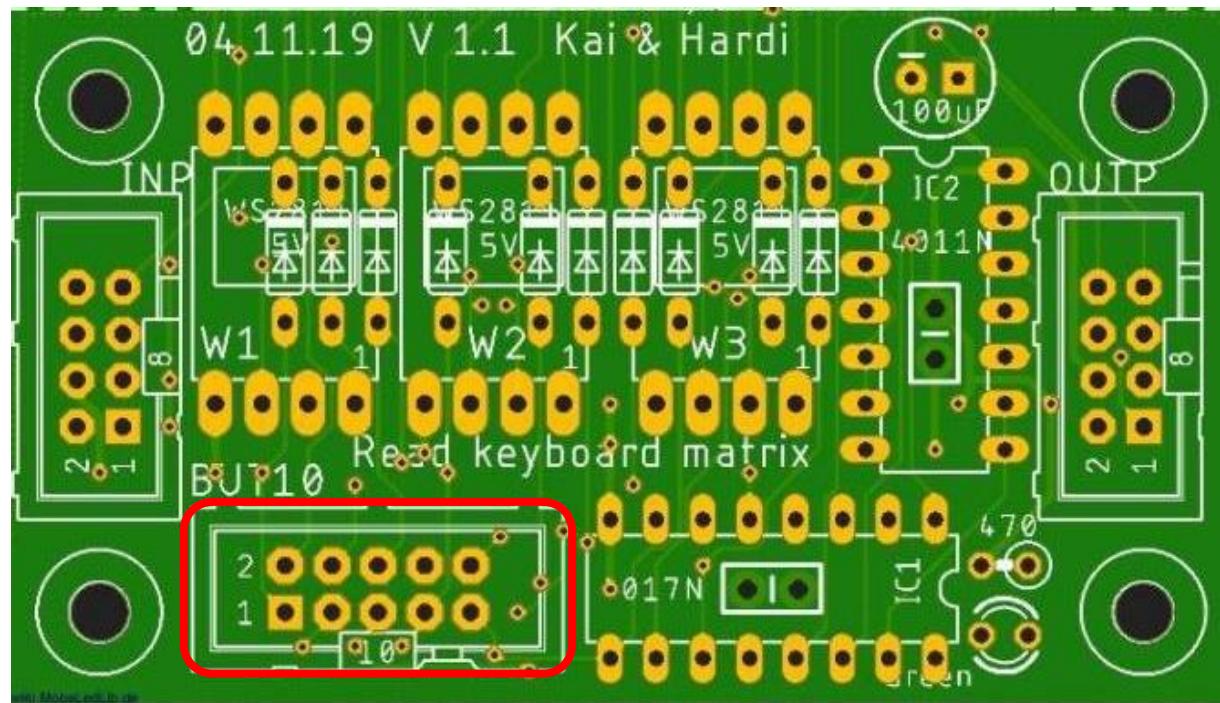
2.3 Methode C: Viele Schalter für ein Weichenstellpult

Das Verfahren zum einlesen von Tastern kann man noch erweitern. Dazu wird nicht nur eine Leitung zum einlesen der aktiven Schalter, sondern mehrere Leitungen verwendet. Wenn man acht Eingänge am Arduino benutzt (WS1 – SW8) können 80 Schalter eingelesen werden.

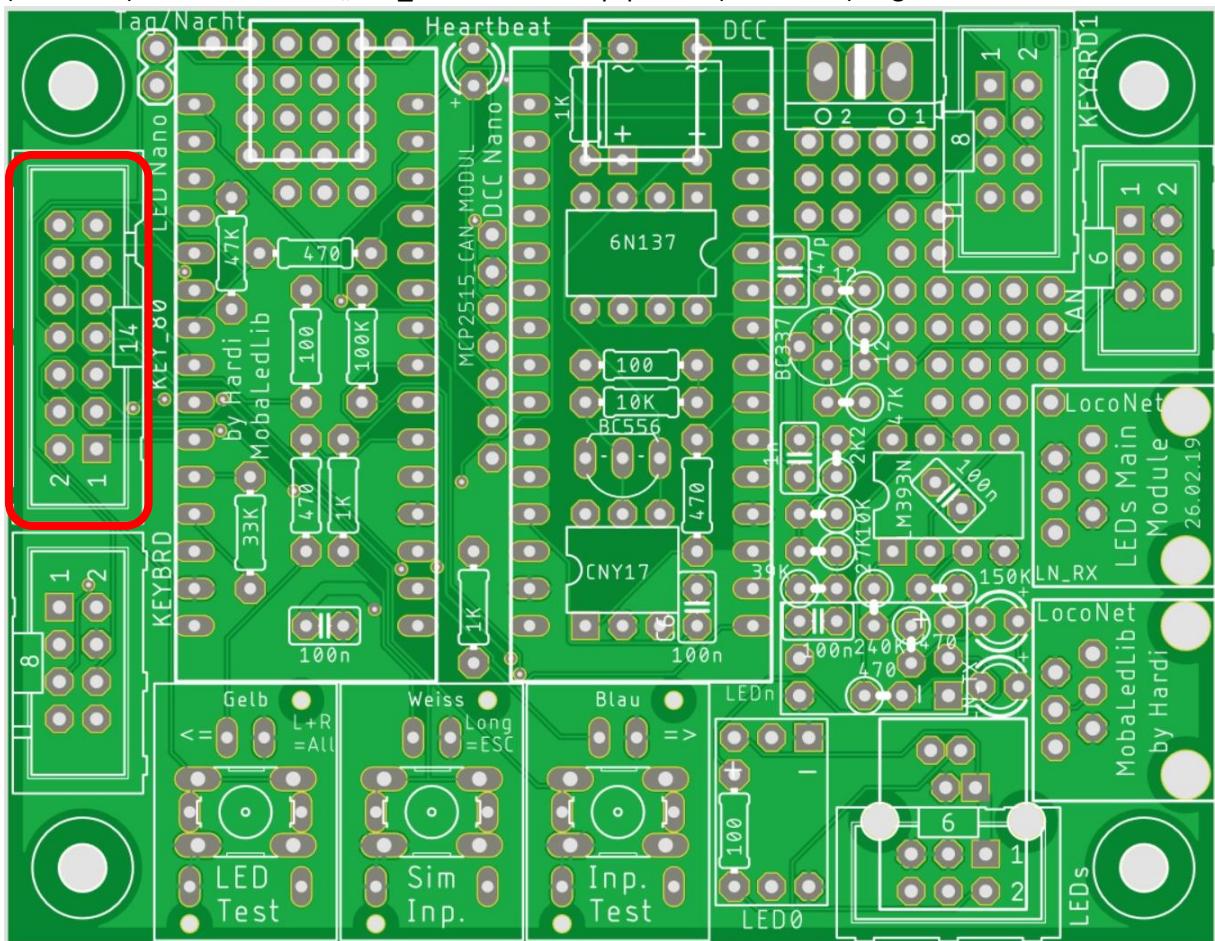


Diese Variante ist zum einlesen von Schaltern und Tastern in einem Weichenstellpult gedacht.

Dazu wird die PushButton_4017 Platine an der dafür vorgesehenen Stelle getrennt:



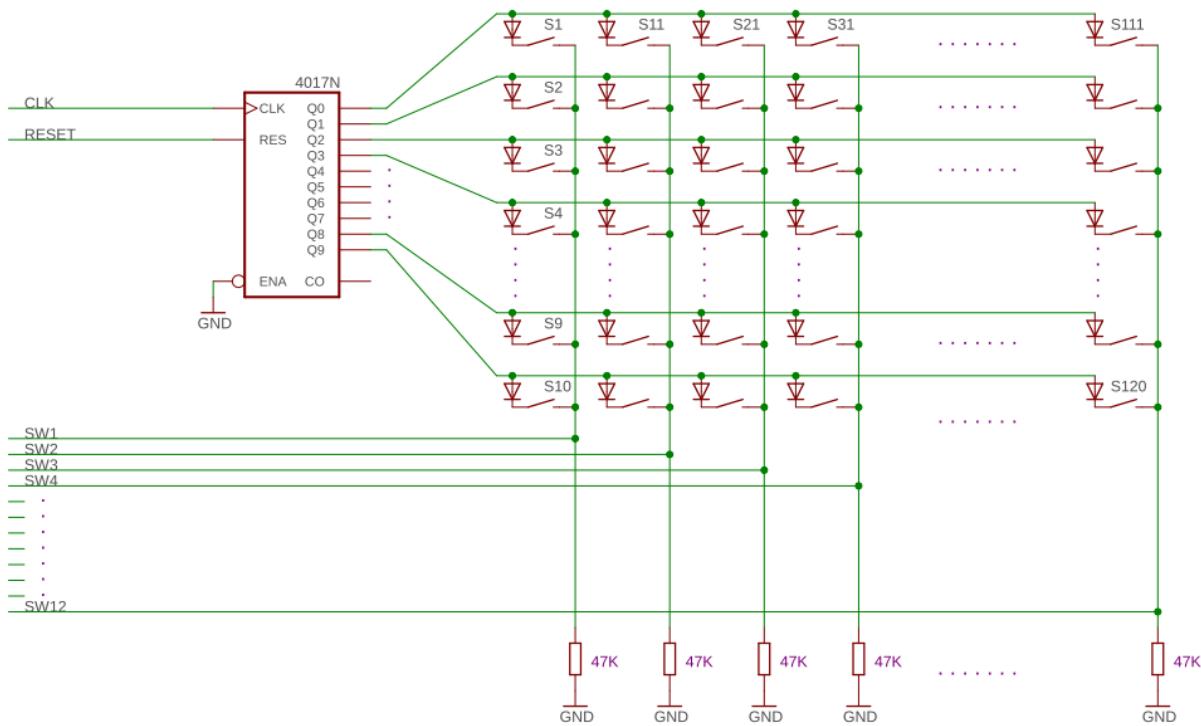
Die Schalter werden jetzt über den 10-poligen Stecker „BUT10“ auf der PushButton_4017 Platine (Bild oben) und den Stecker „KEY_80“ auf der Hauptplatine (Bild unten) angeschlossen.



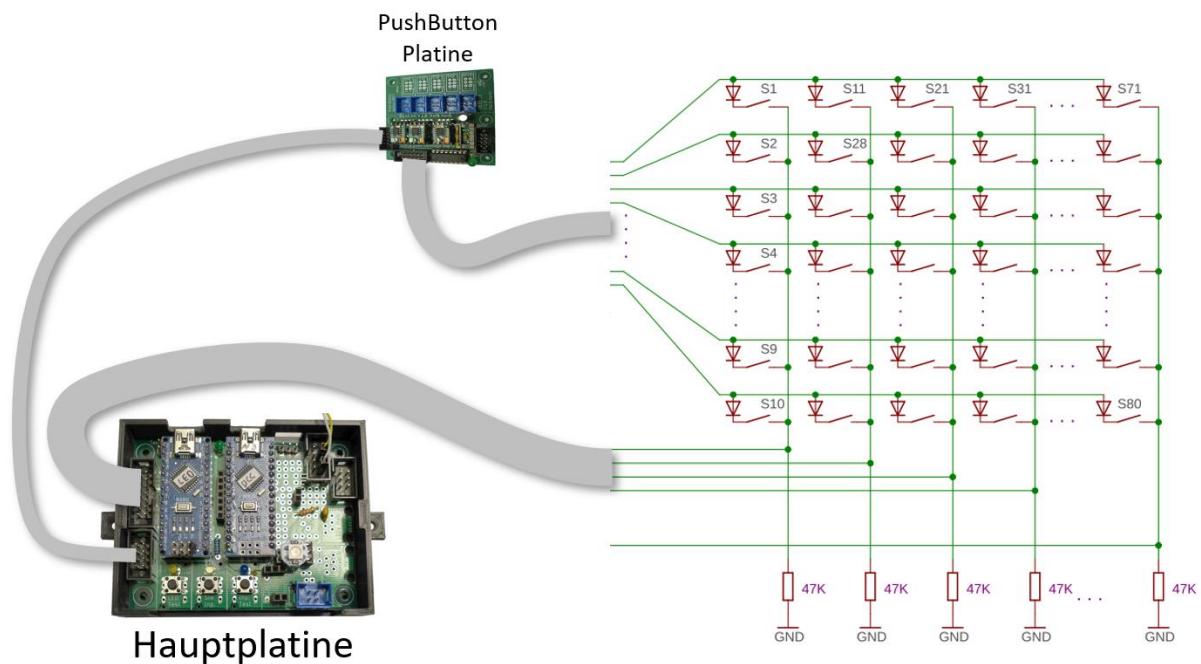
Über den 14-poligen Stecker auf der Hauptplatine können bis zu 10 Eingänge des Arduinos zum einlesen von Schaltern verwendet werden. Damit können 100 Schalter ausgewertet werden.

Wichtig ist, dass jede „Schalter“ Leitung zum Arduino mit einen 47K Widerstand auf Masse gezogen wird, wenn kein Taster betätigt wird. Diese Widerstände sind bis auf einen (Leitung A2) nicht auf der Hauptplatine vorhanden.

Da der Arduino relativ wenige Anschlüsse hat werden mache Pins mehrfach benutzt. Man muss sich entscheiden welche Pins für das Einlesen der Schalter verwenden will. Damit wird bestimmt welche Methoden zum einlesen von Schaltern verfügbar sind und wie viele Schalter eingelesen werden können. Im Folgenden wird darauf noch mal eingegangen.



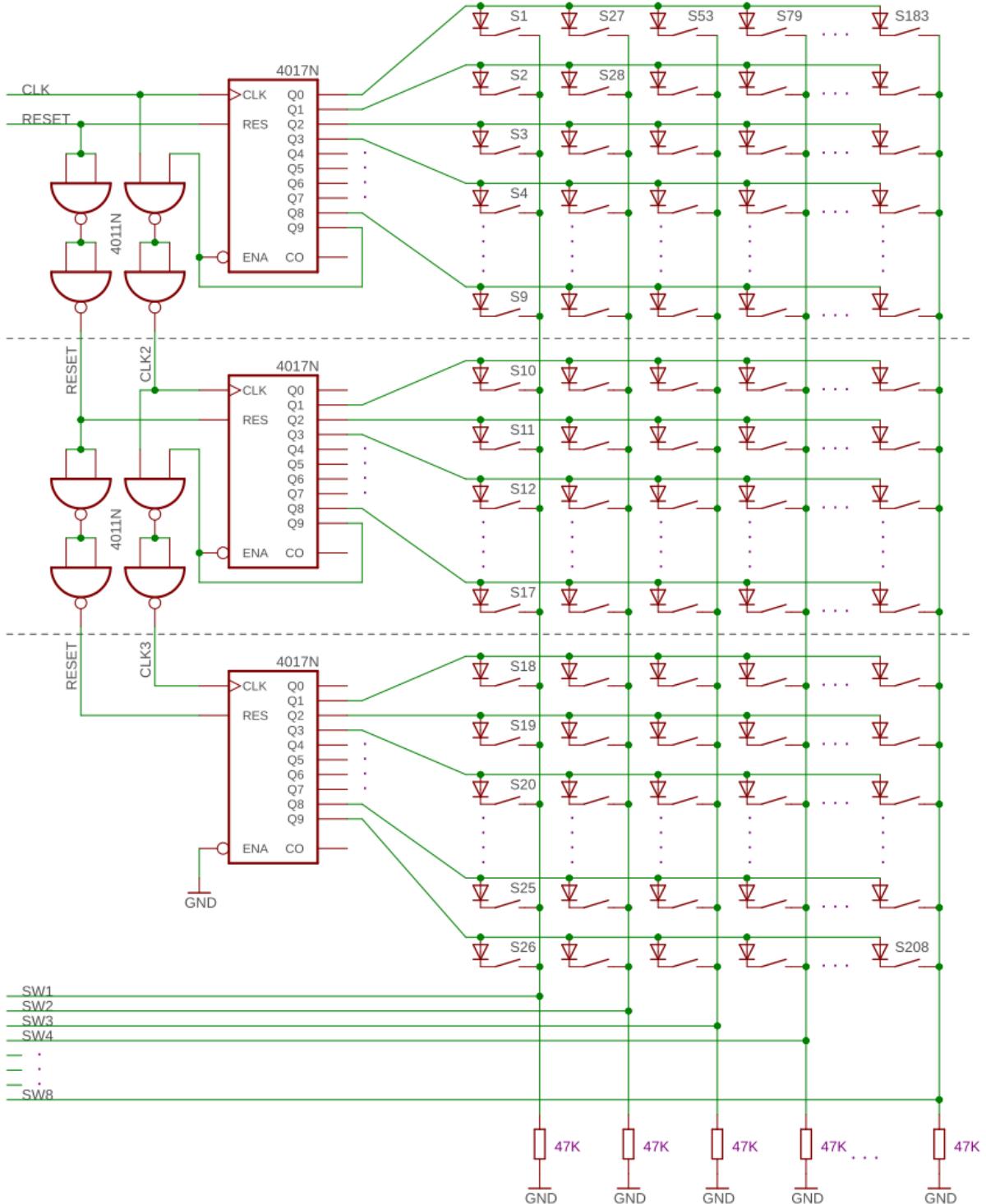
Die Verbindungen zwischen den Platinen werden über Flachkabel gemacht:



Bei dieser Variante ist es genau so wie bei der Methode B möglich mehrere Platinen hintereinander zu schalten. Damit kann man entweder noch mehr Schalter einlesen oder man benötigt weniger Eingangskanäle am Arduino und hat dadurch Pins für andere Aufgaben zur Verfügung. Auch hier verliert man wieder durch das Zusammenschalten der Zähler einige Kanäle. Trotzdem ist es wie das folgende Bild zeigt mit 8 Eingängen am Arduino möglich über 200 Schalter auszuwerten.

Theoretisch könnte man noch mehr Platinen verwenden aber dann stößt man an die Grenzen des im Arduino verfügbaren Speichers. Über die MobaLedLib können maximal 250 Schalter eingelesen werden. Allerdings hat man dann keine Eingangsvariablen mehr für andere Aufgaben zur Verfügung.

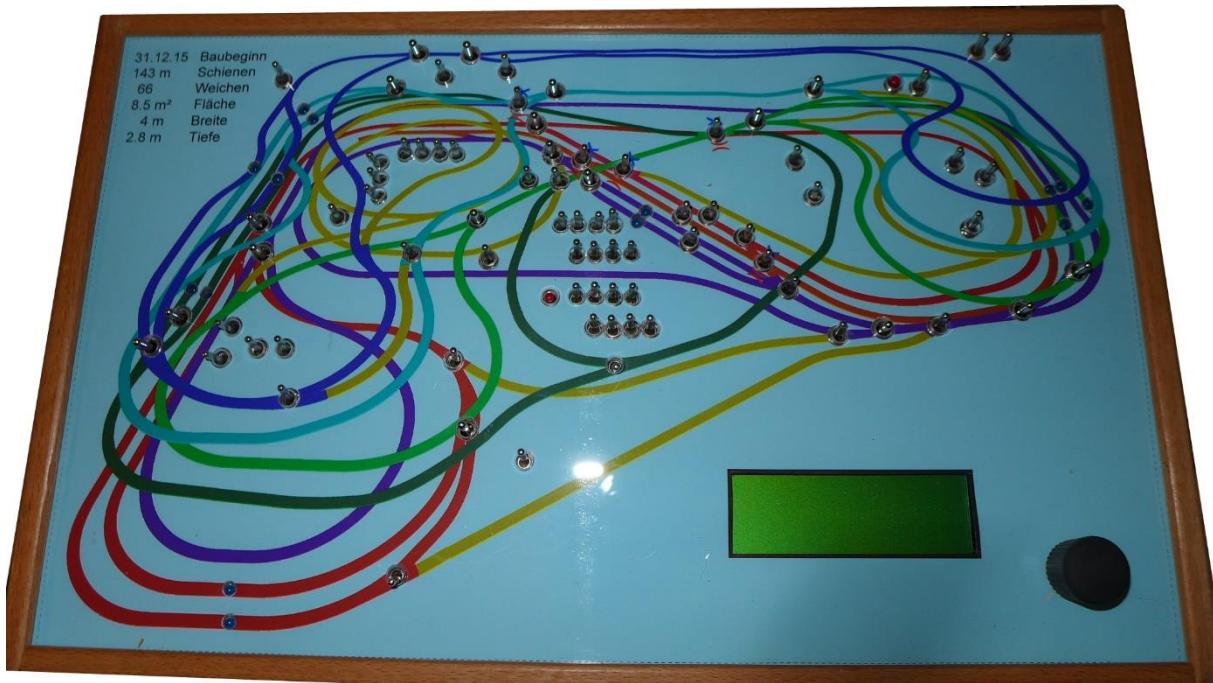
Generell ist das einlesen von so vielen Schaltern möglich, aber dann wird es besser sein, wenn man die Daten über zusätzliche C++ Funktionen auswertet. Hier kann man die Schalter dann über geeignete Schleifen im Programm verarbeiten und muss nicht für jeden Schalter eine eigene Zeile im Prog_Generator anlegen. Das Einbinden eigener C++ Funktionen in Kombination mit der automatisch per Excel generierten Konfiguration ist jeder Zeit möglich.



Dieses Verfahren verwenden wir auch auf unserer Anlage. Hier allerdings noch ohne die PushButton_4017 Platine. Auf dem Weichenstellpult werden zwei 4017 verwendet. Damit können 144 Schalter eingelesen werden.

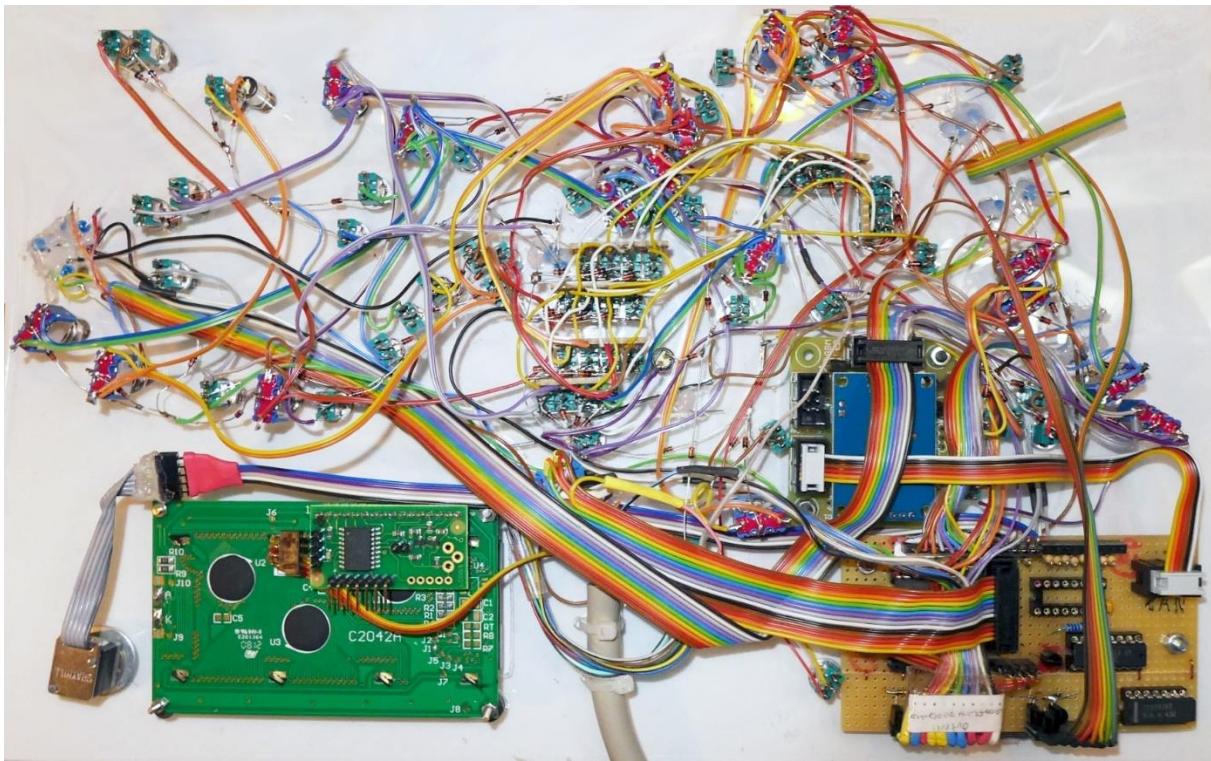
Zum Teil werden auch Schalter mit Mittelstellung verwendet. Das wird bei Drei-Weg-Weichen und bei Weichen welche Automatisch gesteuert werden eingesetzt. Die mittlere Stellung entspricht dem Automatik Modus. Schalter mit drei Stellungen belegen zwei Kanäle der Schaltermatrix.

In dem Bild unten sieht man einige Schalter zwischen den Trassen. Sie sind zur Steuerung der Beleuchtungen vorgesehen.



Das Weichenstellpult besteht aus einem 4 cm hohen Rahmen aus Holzleisten. Die Frontplatte besteht aus zwei Plexiglasscheiben. In der unteren sind die Schalter montiert. Darüber liegt ein bedruckter Karton welcher durch die zweite Scheibe geschützt wird. Bei diesem Aufbau sind die Schrauben der Schalter unsichtbar. Beide Platten werden zusammen in einen Schlitz im Ramen eingeschoben. Dazu lässt sich die untere Leiste abschrauben. So ist in meinen Augen ein recht schönes Schaltpult entstanden. Durch die Verwendung der Schaltermatrix müssen nur sehr wenige Kabel herausgeführt werden.

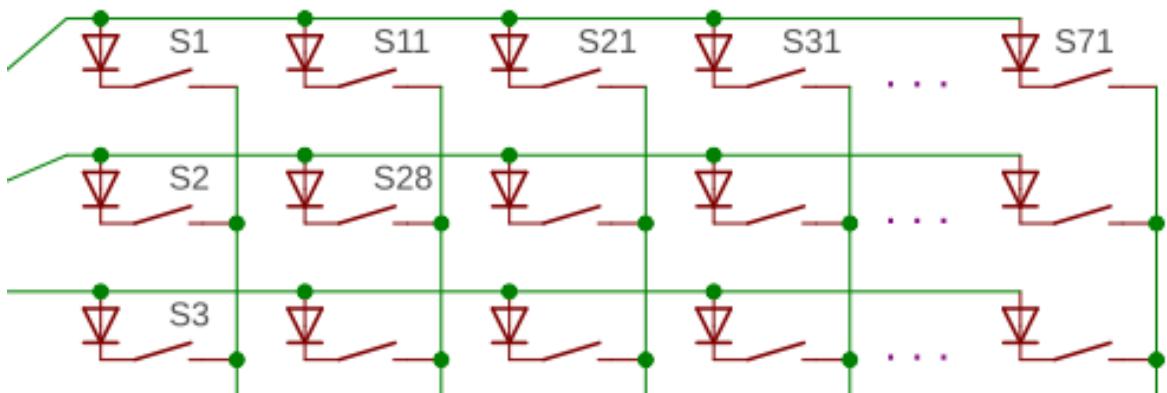
Im Inneren sieht es aber weniger Aufgeräumt aus:



Die Platine rechts unten ist die Vorgängerplatine des PushButton_4017 Moduls. Die blaue Schaltung darüber ist ein CAN Modul mit der die Schalterleitungen zum Prozessor geführt werden. Die grüne Platine ist das LCD-Display. Daneben der Dreh/Drück Schalter über den verschiedene Sonderfunktionen abgerufen werden können.

Durch die Verwendung farbiger Leitungen ist es aber gar nicht so schlimm. Im nächsten Bild sieht man einen Ausschnitt des oberen linken Bereichs. Hier erkennt man schön die Dioden welche einfach frei fliegend an die Schalter gelötet sind. Acht Dioden sind zu einer Zeilenleitung (Siehe Schaltplan unten) zusammengefasst welche zum 4017 führt. Im Bild die blaue Leitung. Anhand der Farbe erkennt man das es die Zeile 6 ist. Jeder Schalter hängt an einer eigenen Spaltenleitung. Die beiden Schalter links oben an der Spalte 1 (Rot) und 0 (Braun). Der Taster daneben ist mit Spalte 3 (Orange) und der Schalter mit Spalte 4 (Gelb) verbunden. Die Kabelfarben entsprechen der Nummerierung der Farbringe bei Widerständen.

Beim Schalter rechts oben sieht man eine Besonderheit. Es ist ein Kippschalter mit Mittelstellung. Darum ist er an zwei Spaltenleitungen (Lila = 7 und Grau = 8) angeschlossen. Beide Kontakte Teilen sich eine Diode. Das ist möglich, weil bei einem Umschalter immer nur ein Kontakt mit dem mittleren Anschluss verbunden ist.



2.4 Methode D: Direkt auf der Hauptplatine vorhandene Taster

Die letzte und einfachste Methode zum einlesen von Tastern mit der MobaLedLib benötigt zunächst keine zusätzliche Hardware. Die drei auf der Hauptplatine vorhandenen Taster können über die Namen „SwitchD1“ bis „SwitchD3“ direkt im Prog_Generator verwendet werden.

Auch hier kann die Anzahl der Eingänge erweitert werden. Dazu müssen der Bibliothek nur die benutzten Pins mitgeteilt werden. Die Anschlüsse welche für die drei Taster verwendet werden sind zusätzlich am Stecker „Key_80“ verfügbar. Damit können anstelle der Taster auf der Hauptplatine auch externe Schalter angeschlossen werden. Diese können auch parallel benutzt werden. Auf diese Weise könnten bis zu 12 externe Schalter an die Platine angeschlossen werden. Aber das ist eine Extrem „teure“ Variante zum Lesen von Schaltern, da jeder Schalter einen eigenen der raren Pins belegt. Damit können dann nur noch 10 analoge Schalter (Methode A) eingelesen werden.

Beim direkten Einlesen der Schalter können aber auch wie bei „B“ und „C“ alle Schalter gleichzeitig betätigt werden.

3 Verwendung der Schalter im Prog_Generator

Die Schalter werden über den Namen „Switch“ (Englisch Schalter) gefolgt von einem Buchstaben welcher den Typ beschreibt (A – D) und einer Nummer angesprochen.

Beispiel: „SwitchB7“

Es gibt folgende Schalter Typen:

- „SwitchA“ = Analoge Schalter
- „SwitchB“ = Schalter am Anlagenrand (B = Border in Englisch)
- „SwitchC“ = Schalter im Weichenstellpult (C = Console in Englisch)
- „SwitchD“ = Direkt auf der Hauptplatine vorhandene Schalter

Auf den Namen folgt eine Nummer zwischen 1 und N. Die maximale Anzahl der möglichen Schalter einer Gruppe hängt von deren Konfiguration ab. Das wird im Abschnitt “Konfiguration der benutzen Eingänge“ beschrieben.

Diese Namen können ganz einfach im Prog_Generator verwendet werden. Momentan trägt man die gewünschte Schalter Bezeichnung einfach „von Hand“ in die Tabelle ein. Vielleicht wird das in einer Zukünftigen Version auch über einen Dialog möglich sein.

Die Schalter können an zwei verschiedenen Stellen verwendet werden.

1. In der Spalte „Adresse oder Name“

Anstelle einer (DCC) Adresse kann in der Spalte auch der Name eines Schalters angegeben werden. Hier werden die drei Taster auf der Hauptplatine zum Test mit jeweils einer LED verknüpft:

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1			Test LED Rot			Const(#LED, C1, #InCh, 0, 50)
✓		SwitchD2			Test LED Grün			Const(#LED, C2, #InCh, 0, 50)
✓		SwitchD3			Test LED Blau			Const(#LED, C3, #InCh, 0, 50)

Achtung dabei handelt es sich nicht um die LEDs bei den Tastern, sondern um die drei Farben einer WS2812 RGB LED. Damit kann die Funktionsweise der Schalter schnell überprüft werden.

2. Die Schalter können aber auch in Spalte „Beleuchtung, Sound, oder andere Effekte“ benutzt werden. Hier gibt es einige Funktionen welche mehrere Eingänge haben. Ein Beispiel dafür ist die „Logic“ Funktion. Sie ist ein mächtiges Werkzeug mit dem man beliebige logische Verknüpfungen implementieren kann (Sie erscheint im Auswahl Dialog wen der Experten Mode aktiviert ist). Das kann man dazu nutzen eine Funktion per DCC **oder** per Taster zu steuern:

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		1	Rot	■	ODER Verknüpfung von DCC und Taster			Logic(LogicRes, #InCh OR SwitchD1)
✓		LogicRes			LED mit Ergebnis ansteuern			Const(#LED, C12, #InCh, 0, 50)

Hier werden gleich drei neue Features der MobaLedLib gezeigt.

Zunächst einmal die Verwendung des Schalters „SwitchD1“ in der Effekt Spalte. Dieser Schalter wird „Oder“ verknüpft mit dem Eingang der Funktion. Er wird über den speziellen Namen „#InCh“ abgerufen. Im Arduino wird er durch den DCC Kanal 1 / Rot ersetzt.

Die dritte Neuerung ist die Verwendung von eigenen Variablen. Das Ergebnis der „Logic“ Funktion wird in die Benutzer definierte Variable „LogicRes“ geschrieben. Der Name ist frei wählbar, muss aber den C++ Konventionen entsprechen und darf noch nicht im Programm vorhanden sein. Wenn man Beispielsweise als Ergebnis Variable den Namen „Logic“

verwenden würde dann erscheint diese Fehlermeldung:

```
LEDs_AutoProg.h:66:0: warning: "Logic" redefined
```

3.1 Beispiele mit Schaltern

In diesen Abschnitt werden einige kleine Beispiele aus der Praxis vorgestellt. Die Beispiele verwenden die Tasten auf der Hauptplatine „SwitchD1“ bis „SwitchD3“, weil diese Tasten immer verfügbar sind. Wenn man über die entsprechenden Zusatztasten und Platinen verfügt, dann kann man genauso jeden anderen Tastentyp verwenden.

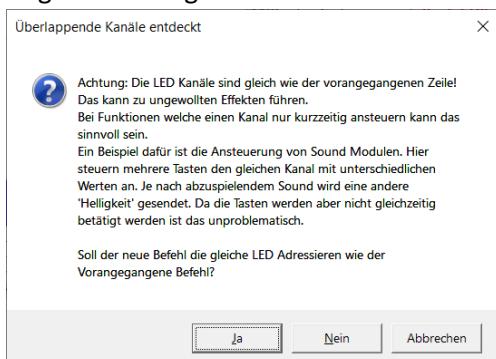
Durch das abtippen der Beispiele versteht man sehr schnell die wie Schalter mit dem Prog_Generator verwendet werden können. Darum werden die entsprechenden Dateien Bewusst nicht bereitgestellt.

3.1.1 Abrufen eines Sounds per Taster

Anstelle der in den vorangegangenen Beispielen verwendeten LED kann man den Sound Befehl verwenden. So kann der Benutzer per Knopfdruck einen Sound abrufen.

Aktiv	Filter	Adresse oder Name	Typ	Start- wert	Beschreibung	Verteiler- Nummer	Stecker- Nummer	Beleuchtung, Sound, oder andere Effekte	Start LedNr	LEDs
✓		SwitchD1			Sound 1 abrufen			Sound_Seq1(#LED, #InCh)	1	C1-2
✓		SwitchD2			Sound 2 abrufen			Sound_Seq2(#LED, #InCh)	1	^ C1-2
✓		SwitchD3			Zufällige Sound abrufen			Sound_PlayRandom(#LED, #InCh, 4)	1	^ C1-2

Wenn man so wie in dem Beispiel mehrere Sound Befehle nacheinander eingibt, dann kommt die folgende Abfrage



welche man mit „Ja“ beantwortet. Das sorgt dafür, dass sich alle drei Zeilen auf den gleichen LED-Kanal beziehen. Damit ist „Start LedNr“ bei allen drei Zeilen gleich und in Spalte „LEDs“ wird ein „^“ eingetragen was bedeutet, dass die gleiche LED-Nummer wie die letzte Zeile benutzt wird.

3.1.2 Zeitschalter

In der MobaLedLib gibt es verschiedene Methoden Zeitschalter zu nutzen. Die einfachste ist der „Button“ Befehl. Damit kann man eine LED für eine bestimmte Zeit anschalten. Und man kann sie mit einem zweiten Druck auf den Taster wieder ausgeschaltet werden. Gedacht ist die Funktion für einen Rauchgenerator in einem Haus. Im Dialog kann ausgewählt werden ob man alle LEDs einer RGB LED oder nur eine einzelne LED ansprechen will. Mit dem Befehl können keine anderen Makros wie z.B. „House()“ Makro gesteuert werden.

Aktiv	Filter	Adresse oder Name	Typ	Start- wert	Beschreibung	Verteiler- Nummer	Stecker- Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1			Beleuchtung, Sound, oder andere Effekte			Button(#LED, C_ALL, #InCh, 5 Sec, 0, 127)
✓		SwitchD2			Treppenhauslicht			ButtonFunc(AktAndreas, #InCh, 5 Sec)
✓		AktAndreas			Steuert ein Andreaskreuz			AndreaskrRGB(#LED, #InCh)
✓		SwitchD3			MonoFlop (Flanken getriggert)			MonoFlop(AktWelding, #InCh, 5 Sec)
✓		AktWelding			Steuert ein Schweißlicht			Welding(#LED, #InCh)

Der „ButtonFunc“ Befehl ist etwas flexibler damit kann eine beliebige andere Funktion für eine bestimmte Zeit aktiviert werden. Dazu gibt man im Dialog eine Zielvariable an welche nach einem Tastendruck so lange 1 ist wie gewünscht. Diese Variable benutzt man zum steuern anderer Funktionen in den folgenden Zeilen. Im Beispiel ist sie „ActAndreas“ genannt und aktiviert ein Andreaskreuz für eine bestimmte Zeit. Die Zeit der „ButtonFunc“ startet erst nachdem der Taster

losgelassen wird. Und kann verlängert werden, wenn der Taster innerhalb der Zeitspanne noch mal gedrückt wird. Wenn man anstelle eines Tasters einen Kontakt verwendet welcher von Zug gesteuert wird, dann würde das Andreaskreuz automatisch an gehen, wenn ein Zug kommt und nachdem der Zug vorbei gefahren ist noch eine gewisse Zeit weiter blinken. Durch die Zeitverzögerung können kurze Unterbrechungen des Kontaktes überbrückt werden.

Das dritte Beispiel verwendet die „MonoFlop“ Funktion. Diese kann nicht wie beim vorangegangenen Befehl durch halten der Taste verlängert werden. Hier startet die Zeit mit dem Moment wann der Taster betätigt wird. Die Zeit kann hier verlängert werden, wenn der Taster innerhalb der gegebenen Zeit erneut gedrückt wird.

3.1.3 Taster als Ein- und Ausschalter

Das nächste Beispiel zeigt wie man ein Haus mit zwei Tastern schaltet. Dazu wird ein RS-Flip-Flop verwendet. Ein Flip-Flop wird auch als bistabile Kippstufe bezeichnet. Es ist eine Schaltung welche zwei verschiedene Zustände annehmen kann. Ihr Ausgang kann entweder ein oder ausgeschaltet sein. Bei einem RS-Flip-Flop wird der Zustand mit zwei Impulseingängen umgeschaltet. Der „Set“ Eingang schaltet das Flip-Flop an, der „Reset“ Eingang schaltet es aus. Dazu werden zwei Taster (SwitchD1 und SwitchD2) benutzt. SwitchD2 wird als erste Parameter in der Zeile unten angegeben, weil er zum Abschalten (R) verwendet werden soll. Das Ergebnis des Flip-Flops wird in eine Variable geschrieben welche dann von der folgenden Funktion ausgewertet wird. Hier wird damit ein Haus geschaltet. Die in der „House“ Funktion integrierte Steuerung sorgt dafür, dass beim Einschalten nur ein „Zimmer“ aktiviert wird. Die anderen folgen zufällig. Die Anzahl der beleuchteten Zimmer wird dann gesteuert, dass sie zufällig zwischen den vorgegebenen minimalen und maximalen Anzahl liegt. So entsteht der Eindruck eines belebten Hauses. Bei abschalten mit dem zweiten Taster geht sofort ein zufälliges Licht aus. Die anderen folgen wiederum nach zufälligen Zeiten. Im Beispiel unten ist zusätzlich noch eine Ausschaltzeit von 2 Minuten eingestellt. Diese „Strom spar“ Funktion kann mit der Zeit 0 deaktiviert werden.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD2			Ein- Ausschalter mit zwei Tasten			RS_FlipFlopTimeout(HausNeon, #InCh, SwitchD1, 2 Min)
✓		HausNeon			Haus mit 3 Neon Lichtern			House(#LED, #InCh, 1, 2, NEON_LIGHTD, NEON_LIGHTD, NEON_LIGHTD)

Wenn nur ein Taster zum Ein- und Ausschalten benutzt werden soll, dann kann die „T_FlipFlopResetTimeout“ Funktion eingesetzt werden. Hier werden damit die Straßenlaternen gesteuert. Mit dem Einschaltbefehl wird zunächst eine Laterne gezündet. Sie ist nicht sofort hell, weil der Glühstrumpf erst heiß werden muss. Die anderen Straßenlaternen folgen zeitlich versetzt. Damit werden die unterschiedlichen Einstellungen der Zeitschalter in den alten Gaslaternen simuliert. Genau so wird beim Ausschalten über den Taster wird zunächst auch nur eine Lampe ausgehen. Hier keine automatische Abschaltung eingestellt (Abschaltzeit 0 Sec). Die Zeitangaben können in „ms“, „Sec“, „Sek“ oder „Min“ angegeben werden (Das „Sec“ stammt aus der englischen Abkürzung“). Es sind auch Kommazahlen wie „1.6 Min“ möglich.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1			"Stromstoß Relais"			T_FlipFlopResetTimeout(StrLaternen, #InCh, SI_0, 0 Sec)
✓		StrLaternen			Ansteuerung eines Verbrauchers			GasLights(#LED, #InCh, GAS_LIGHTD, GAS_LIGHTD, GAS_LIGHTD, GAS_LIGHTD, GAS_LIGHTD)

3.1.4 Knopf Druck Aktion

Komfortabler als die vorangegangenen Funktionen zum einlesen von Tastern kann man Aktionen mit der „PushButton_w_LED_BL_0_<Nr>“ schalten. Dabei können mehreren Zuständen benutzt werden.

Die „<Nr>“ im Namen des Befehls bestimmt die Anzahl aktiven Zustände. Mit der „3“ wie unten gezeigt gibt es drei aktive und einen nicht aktiven Zustand. Letzterer hat die Nummer 0. Die aktiven Zustände die Nummern 1 bis 3. Das „LED“ im Namen beschreibt, dass diese Funktion außerdem eine Status LED ansteuern kann. Diese LED blinkt im Muster des aktiven Zustands. Sie ist einmal an gefolgt von einer längeren Pause an, wenn die Taste einmal betätigt wurde. Mit dem zweiten Tastendruck wird Zustand Zwei ausgewählt. Die LED Leuchtet hier zweimal. Entsprechend aktiviert der dritte Druck auf die Taste den dritten Zustand welcher mit drei Lichtpulsen gefolgt von einer Pause signalisiert wird. Danach wird wieder mit Zustand 1 begonnen. Dieses Verhalten kann im Dialog konfiguriert werden.

Das „BL“ im Namen steht für „Back Light“, dem englischen Ausdruck für Hintergrund Beleuchtung. Dieses Feature sorgt für eine schwache Beleuchtung des Schalters im ausgeschalteten Zustand damit dieser im abgedunkelten Raum zu finden ist.

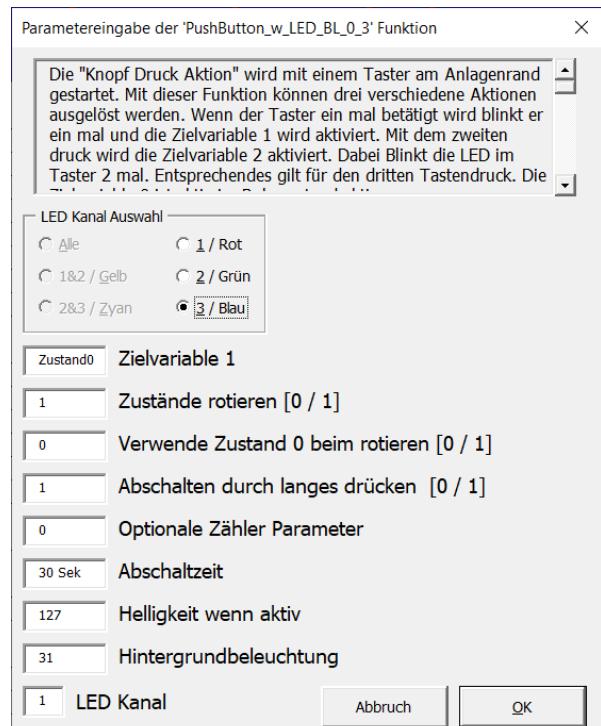
Die Parameter der „PushButton_w_LED_BL_0_3“ Funktion zeigt das Bild Rechts.

Über das „LED Kanal Auswahl“ Feld wird der verwendete LED Kanal bestimmt. Die Farbe „Rot“, „Grün“ und „Blau“ steht dabei für den Anschluss des WS2811 Chips über den die LEDs angesteuert werden. Sie hat nichts mit der tatsächlichen Farbe der LED zu tun. Die Taster auf der PushButton_4017 Platine verwenden die Kanäle in der Reihenfolge RGB. Das bedeutet, dass der Erste Taster den LED Kanal 1 = „Rot“ verwendet. Der zweite Taster benutzt die LED 2 = „Grün“, ...

In dem Feld „Zielvariable 1“ wird ein möglichst Aussagekräftiger Name einer Variable angegeben. Über diese Variable werden die Aktionen angesteuert. Die Funktion steuert über vier Variablen welche impliziert aus der angegebene Zielvariable „Zustand0“ generiert werden im Beispiel unten 4 LEDs an. Das ist zum Verständnis der Funktion hilfreich. In den folgenden Beispielen werden mit den Variablen komplexere Aktionen ausgelöst.

Die verschiedenen Zustände „Knopf Druck Aktion“ werden durch wiederholtes drücken auf den Taster nacheinander aktiviert. Mit der „1“ im Feld „Zustände rotieren“ beginnt der Zyklus nach dem letzten Zustand wieder mit Zustand 1. Der Benutzer kann die Zustände beliebig aktivieren. Wird im Feld „Zustände rotieren“ eine „0“ eingetragen, dann können die Zustände nur einmal nacheinander abgerufen werden. Nach dem letzten Zustand beginnt es nicht wieder mit Zustand 1.

Der nächste Parameter „Verwende Zustand 0 beim rotieren“ gibt an ob der Zustand 0 benutzt werden soll, wenn die Taste mehrfach nacheinander betätigt wird. Dieser Schalter kann nur dann genutzt werden wenn auch „Zustände rotieren“ aktiviert ist. Er ist gedacht für Aktionen welche mit einem Tastendruck An und mit dem nächsten Tastendruck wieder Ausgeschaltet werden. Also bei



Aktionen mit einem aktiven Zustand („PushButton_w_LED_BL_0_1“). Er kann aber auch bei Funktionen mit mehreren Zuständen benutzt werden.

Der Parameter „Abschalten durch langes drücken“ aktiviert eine alternative Methode zum Abschalten einer „Knopf Druck Aktion“. Damit kann die Aktion abgebrochen werden, wenn der Taster länger als 1.5 Sekunden betätigt wird.

Über das Feld „Optionale Zähler Parameter“ können weitere Parameter für den Zähler der die Funktion steuert definiert werden.

Folgende Flags sind möglich: CF_INV_INPUT, CF_BINARY, CF_PINGPONG, CF_RANDOM. Sie werden über '|' getrennt angegeben. Da diese Flags nur für Experten gedacht sind wird hier auf eine ausführliche Beschreibung verzichtet.

Mit der „Abschaltzeit“ wird bestimmt nach welcher Zeit sich die Aktion automatisch deaktiviert. Die Zeit wird in Millisekunden angegeben. Es können aber auch „Sec“ oder „Min“ angehängt werden. Die Maximale Zeit beträgt 17 Minuten (1048560 ms).

Wie hell die LED im Taster leuchtet bestimmt man über die Angabe „Helligkeit wenn aktiv“. Hier werden Zahlen zwischen 0 und 255 akzeptiert.

Mit dem Wert „Hintergrundbeleuchtung“ wird angegeben wie Hell die LED im Taster ist, wenn die Funktion deaktiviert ist. Damit soll der Taster auch im Abgedunkelten Raum zu finden sein.

Der „LED Kanal“ an welchen LED-Strang die LED des Tasters angeschlossen ist. Mit der neuen Version der MobaLedLib können bis zu 4 verschiedene Arduino Pins zur Ansteuerung unabhängiger LED-Stränge verwendet werden. Der Kanal 1 wird normalerweise für die Taster am Anlagenrand benutzt. Er wird über den 8-polige „KEYBRD“ Stecker der Hauptplatine an die PushButton_4017 Platine geführt. Der LED-Kanal kann aber auch ohne die PushButton_4017 Platine z.B. in Verbindung mit den analogen Tastern genutzt werden.

Achtung: Wenn man keinen Zusätzlichen LED-Kanal verwendet, dann muss bei „LED Kanal“ eine 0 eingetragen werden sonst sieht man die Status LED nicht.

Begriff „LED-Kanal“ wird in verschiedenen Kontexten verwendet. Er wird auch in Bezug auf einzelne LEDs eines WS281x Chips benutzt.

Beispiel:

Mit diesem ersten Beispiel wird die Funktionsweise der „PushButton“ Funktion gezeigt. Wenn man sich damit beschäftigen will, dann sollte man zunächst dieses Beispiel testen. Das ist auch zur Überprüfung der Hardware dringend zu empfehlen, wenn man einen neuen Schaltertyp einsetzt.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteller-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		PushButton1			Knopf Druck Aktion mit 3 Zuständen			PushButton_w_LED_BL_0_3(#LED, C3, #InCh, Zustand0, 1, 0, 1, 0, 30 Sek, 127, 31)
✓		Zustand0			Zustand 0			ConstRGB(#LED, #InCh, 0, 0, 0, 50, 0, 0)
✓		Zustand1			Zustand 1			ConstRGB(#LED, #InCh, 0, 0, 0, 50, 50, 0)
✓		Zustand2			Zustand 2			ConstRGB(#LED, #InCh, 0, 0, 0, 0, 50, 0)
✓		Zustand3			Zustand 3			ConstRGB(#LED, #InCh, 0, 0, 0, 50, 0, 50)

Windrad:

Das Windrad ist ein ganz einfaches Beispiel für die „PushButton“ Funktion. Es hat zwei verschiedene aktive Zustände. Im ersten Zustand blinkt nur das Positionslicht, im zweiten Zustand dreht sich zusätzlich noch das Windrad. Wie das aussieht kann man hier im Video betrachten:

Stummi Forum MobaLedLib #33:

<https://www.stummiforum.de/viewtopic.php?f=7&t=165060&sd=a&start=32>

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nr.	Stecker-Nr.	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1			Windrad mit 2 Zuständen			PushButton_w_LED_BL_0_2(#LED, C3, #InCh, Windrad0, 1, 0, 1, 0, 5 Min, 127, 31)
✓		Windrad0			Windrad0 invertieren für Leuchtfieber			Logic(LeuchtF, NOT #InCh)
✓		LeuchtF			Leuchtfieber ist an wenn Windrad0 aus ist (1&2)			Leuchtfieber(#LED, C2, #InCh)
✓		Windrad2			Motor des Windrads			Const(#LED, C1, #InCh, 0, 50)

Das Leuchtfieber soll leuchten, wenn Zustand 1 oder Zustand 2 aktiv ist. Dazu wird die „Logic“ Funktion verwendet. Naheliegend wäre eine ODER Verknüpfung von Windrad1 und Windrad2. Etwas einfacher und schneller ist es, wenn man das Leuchtfieber immer dann an macht, wenn der Zustand 0 nicht aktiv ist. Dadurch muss nur eine Variable gelesen werden. Diese Invertierung wird in Zeile zwei gemacht. Ob das tatsächlich einen messbaren Geschwindigkeitsvorteil bringt ist ungewiss... Das Ergebnis der logischen Verknüpfung wird über die Variable „LeuchtF“ an die dritte Zeile übergeben mit der dann das Leuchtfieber gesteuert wird. Mit der letzten Zeile wird schließlich der Motor im Windrad geschaltet. Die 50 in der „Const“ Funktion kann zur Einstellung der Drehzahl des Propellers angepasst werden.

U-Bahn Beleuchtung:

In unserer Anlage gibt es ein Fenster mit dem man die „unterirdisch“ verlaufenden Gleise betrachten kann. Über einen Taster am Anlagenrand kann man die „Unterwelt“ beleuchten. Mit dem ersten Tastendruck wird der Untergrund Blau beleuchtet. Dabei wird das Licht langsam heller was viel schöner ist als ein Abruptes einschalten der Beleuchtung. Mit dem zweiten Tastendruck wird von Blau nach Weiß übergeblendet. Wenn die Taste ein weiteres Mal betätigt wird dann wird wieder das blaue Licht aktiviert. Die Beleuchtung deaktiviert sich selbstständig nach 5 Minuten oder über einen längeren Tastendruck.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nr.	Stecker-Nr.	Beleuchtung, Sound, oder andere Effekte	Start_LedNr	LEDs	InCnt	
✓		SwitchD1			U-Bahn Beleuchtung: Knopf Druck Aktion mit 2 Zuständen			PushButton_w_LED_BL_0_2(#LED, C3, #InCh, U_Bahn0, 1, 0, 0, 5 Min, 127, 31)	1	1	C3-3	1
✓		U_Bahn0			Zustand 0: LEDs aus			XPatternT1(#LED, 160, #InCh, 51, 0, 255, 0, PM_SEQUENZ_STOP, 2, 17, 1)	19	-17	0	
✓		U_Bahn1			LED Numer für folgende Zeile wie oben			XPatternT1(#LED, 160, #InCh, 51, 0, 255, 0, PM_SEQUENZ_STOP, 2, 17, 1)	19	-17	0	
✓		U_Bahn2			Zustand 1: LEDs blau			XPatternT1(#LED, 160, #InCh, 51, 0, 255, 0, PM_SEQUENZ_STOP, 2, 17, 1)	19	-17	0	
					LED Numer für folgende Zeile wie oben			XPatternT1(#LED, 160, #InCh, 51, 0, 255, 0, PM_SEQUENZ_STOP, 2, 17, 1)	19	-17	0	
					Zustand 2: LEDs weiß			XPatternT1(#LED, 160, #InCh, 51, 0, 255, 0, PM_SEQUENZ_STOP, 2, 17, 1)	19	-17	0	

Hier die drei verwendeten Pattern Zeilen:

```
XPatternT1(#LED, 160, #InCh, 51, 0, 255, 0, PM_SEQUENZ_STOP, 1 Sek, 0, 0, 0, 0, 0, 0)
XPatternT1(#LED, 160, #InCh, 51, 0, 255, 0, PM_SEQUENZ_STOP, 1 Sek, 36, 73, 146, 36, 73, 146, 4)
XPatternT1(#LED, 160, #InCh, 51, 0, 255, 0, PM_SEQUENZ_STOP, 1 Sek, 255, 255, 255, 255, 255, 255, 7)
```

Achtung: Bei der Eingabe der XPattern Zeilen muss die 17 in der LEDs Spalte und der „InCnt“ Wert ebenfalls eingetragen werden. Dazu klickt man zunächst rechts neben die Tabelle und dann in das entsprechende Feld sonst wird der Cursor immer wieder aus dem Feld heraus bewegt. Diese Funktionalität soll verhindern, dass man die Einträge versehentlich verändert. Normalerweise werden diese Angaben automatisch vom Programm gemacht, wenn ein Muster aus dem Pattern_Configurator übernommen wird.

Burg Illumination:

Im Stummi Forum zur MobaLedLib findet man im Beitrag #224 ein Video unserer Burg Beleuchtung: <https://www.stummiforum.de/viewtopic.php?f=7&t=165060&sd=a&start=223>.

Hier sieht man das die verschiedenen Strahler mit dem ersten Tastendruck nacheinander aktiviert werden. Beim zweiten Druck auf die Taste wird der Farbmodus aktiviert. Die „Scheinwerfer“

verändern dabei ganz langsam ihre Farbe. Jeder Strahler mit einer etwas anderen Geschwindigkeit. Dadurch ergibt sich ein schön kitschiges Farbenspiel.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte	Start LedNr	LEDs	InCnt
✓		SwitchD1			Burg Illumination mit 2 Zuständen			PushButton_w_LED_BL_0_2(#LED, C1, #InCh, Castle0, 1, 0, 1, 0, 5 Min, 127, 31)	1	C1-1	1
✓		Castle0			Zustand 0: LEDs langsam ausblenden			XPatternT1(#LED,192,#InCh,18,0,255,0,PM_SEQUENZ_STOP,2 Sek,0,0,0)	2	6	1
✓		Castle1			LED Numer für folgende Zeile wie oben			// Next_LED(-6)	8	-6	0
✓		Castle1			Zustand 1: Weiße LEDs nacheinander auflenden			XPatternT1(#LED,128,#InCh,18,0,255,0,PM_SEQUENZ_STOP,2 Sek,0,0,0)	2	6	1
✓		Castle2			LED Numer für folgende Zeile wie oben			// Next_LED(-3)	8	-6	0
✓		Castle2			Zustand 2: Farbige LEDs			APatternT6(#LED+0,12, #InCh,3,0,255,0,PM_HSV,3 Sek,	2	6	1

Hier die drei verwendeten Pattern Zeilen:

```
XPatternT1(#LED,192,#InCh,18,0,255,0,PM_SEQUENZ_STOP,2 Sek,0,0,0)
XPatternT1(#LED,128,#InCh,18,0,255,0,PM_SEQUENZ_STOP,500 ms,7,0,252,0,240,31,192,255,3,255,127,252,255,15)
```

```
APatternT6(#LED+0,12, #InCh,3,0,255,0,PM_HSV,3 Sek, 30 Sek,40 Sek,38 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+1,12, #InCh,3,0,255,0,PM_HSV,3 Sek+100,30 Sek,40 Sek,40 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+2,12, #InCh,3,0,255,0,PM_HSV,3 Sek+200,30 Sek,40 Sek,30 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+3,12, #InCh,3,0,255,0,PM_HSV,3 Sek+300,30 Sek,40 Sek,40 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+4,12, #InCh,3,0,255,0,PM_HSV,9 Sek+300,31 Sek,40 Sek,45 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+5,12, #InCh,3,0,255,0,PM_HSV,8 Sek+300,35 Sek,40 Sek,5 Sek,500,15,255,255,240,255,255,0,0,0)
```

Die dritte „Zeile“ besteht aus 6 einzelnen Zeilen welche in eine Zelle kopiert werden (F2 und dann Strg+V). Jede „APattern“ Zeile ist für eine LED verantwortlich. Man erkennt, dass verschiedene Zeiten verwendet werden was zu dem scheinbar zufälligen Farbenspiel führt.

Achtung: Die 6 in der LEDs Spalte und der „InCnt“ Wert muss ebenfalls eingetragen werden. Dazu klickt man zunächst rechts neben die Tabelle und dann in das entsprechende Feld sonst wird der Cursor immer wieder aus dem Feld heraus bewegt. Diese Funktionalität soll verhindern, dass man die Einträge versehentlich verändert. Normalerweise werden diese Angaben automatisch vom Programm gemacht, wenn ein Muster aus dem Pattern_Configurator übernommen wird.

Höhlen Beleuchtung:

Die Beleuchtung in einer Höhle wird nur dann angeschaltet, wenn Besucher da sind damit das Pflanzenwachstum nicht unnötig angeregt wird. Mit der folgenden „Knopf Druck Aktion“ kann man das umsetzen. Ein Video dazu findet man im Stummi Forum unter MobaLedlib #29:

<https://www.stummiforum.de/viewtopic.php?f=7&t=165060&sd=a&start=28>

Es handelt sich dabei um eine der unglaublich seltenen Diamanten Höhlen in denen geschliffene Edelsteine zu finden sind.

Die Höhle hat 3 verschiedene Modis welche nacheinander per Taster aktiviert werden:

1. Ein oder zwei Lagerfeuer (Zufällig)
2. Lagerfeuer und eine dunkle Illumination
3. Lagerfeuer und eine helle Illumination

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte	Start LedNr	LEDs	InCnt
✓		SwitchD1			Höhlenbeleuchtung mit 3 Zuständen			PushButton_w_LED_BL_0_3(#LED, C3, #InCh, Hoehle0, 1, 0, 1, 0, 5 Min, 127, 31)	1	C3-3	1
✓		Hoehle0			Hoehle0 invertieren für Lagerfeuer			Logic(Feuer, NOT #InCh)			1
✓		Feuer			Lagerfeuer ist an wenn Hoehle0 aus ist			House(#LED, #InCh, 1, 2, FIRE, FIRED)	2	2	1
✓		Hoehle0			Alle LEDs herunter dimmen			XPatternT1(#LED,128,#InCh,36,0,255,0,PM_SEQUENZ_STOP,4 Sek,0,0,0)	4	12	1
✓		Hoehle1			LED Numer für folgende Zeile wie oben			// Next_LED(-12)	16	-12	0
✓		Hoehle1			Alle LEDs herunter dimmen			XPatternT1(#LED,128,#InCh,36,0,255,0,PM_SEQUENZ_STOP,4 Sek,0,0,0)	4	12	1
✓		Hoehle2			LED Numer für folgende Zeile wie oben			// Next_LED(-12)	16	-12	0
✓		Hoehle2			Makro Definitionen			#define Changing_Hue(LED, InNr, Period)			
✓		Hoehle3			Zustand 1: Lagerfeuer und eine dunkle Illumination			Illumination(#LED+0, #InCh, 70 Sec, 0 Sek, 5 Sek)	4	12	1
✓		Hoehle3			LED Numer für folgende Zeile wie oben			// Next_LED(-12)	16	-12	0
✓		Hoehle3			Zustand 2: Lagerfeuer und eine helle Illumination			APatternT6(#LED+0,12, #InCh,3,0,255,0,PM_HSV,3 Sek,	4	12	1

Die „Pattern“ und „#define“ Zeilen sind recht komplex:

Alle LEDs herunter dimmen: (2x)

```
XPatternT1(#LED,128,#InCh,36,0,255,0,PM_SEQUENZ_STOP,2 Sek,0,0,0,0,0)
```

Makro Definitionen:

```

#define Changing_Hue( LED, InNr, Period) \
    APatternT2(LED,192,InNr,1,0,255,0,PM_HSV|PF_SLOW,Period/16, 0 ms,1) \
\\

#define Pulsating_Val(LED, InNr, Delay, RampTime, HoldTime, Pause) \
    APatternT5(LED, 98,InNr,1,0,255,0,PM_HSV|PF_EASEINOUT,Delay,RampTime, HoldTime,RampTime,Pause,6) \
\\

#define Illumination(LED, InNr, HuePeriod, Delay, RampTime, HoldTime, Pause) \
    New_HSV_Group() \
    Changing_Hue( LED, InNr, HuePeriod) \
    Pulsating_Val(LED, InNr, Delay, HoldTime, RampTime, Pause)

```

Zustand 1: Lagerfeuer und eine dunkle Illumination:

```

Illumination(#LED+0, #InCh, 70 Sec, 0 Sek, 5 Sek, 2 Sek, 10 Sek) \
Illumination(#LED+1, #InCh, 30 Sec, 6 Sek, 5 Sek, 2 Sek, 19 Sek) \
Illumination(#LED+2, #InCh, 40 Sec, 11 Sek, 5 Sek, 4 Sek, 23 Sek) \
Illumination(#LED+3, #InCh, 65 Sec, 24 Sek, 8 Sek, 2 Sek, 1 Sek) \
Illumination(#LED+4, #InCh, 60 Sec, 30 Sek, 5 Sek, 2 Sek, 0 Sek) \
Illumination(#LED+5, #InCh, 50 Sec, 33 Sek, 5 Sek, 3 Sek, 2 Sek) \
Illumination(#LED+6, #InCh, 75 Sec, 28 Sek, 10 Sek, 2 Sek, 1 Sek) \
Illumination(#LED+7, #InCh, 25 Sec, 4 Sek, 5 Sek, 5 Sek, 4 Sek) \
Illumination(#LED+8, #InCh, 45 Sec, 13 Sek, 15 Sek, 9 Sek, 7 Sek) \
Illumination(#LED+9, #InCh, 55 Sec, 21 Sek, 5 Sek, 3 Sek, 21 Sek) \
Illumination(#LED+10, #InCh, 65 Sec, 38 Sek, 5 Sek, 2 Sek, 2 Sek) \
Illumination(#LED+11, #InCh, 50 Sec, 1 Sek, 9 Sek, 2 Sek, 32 Sek)

```

Zustand 2: Lagerfeuer und eine helle Illumination:

```

APatternT6(#LED+0,12, #InCh,3,0,255,0,PM_HSV,3 Sek,3 Sek,10 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+1,12, #InCh,3,0,255,0,PM_HSV,3 Sek+100,3 Sek,10 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+2,12, #InCh,3,0,255,0,PM_HSV,3 Sek+200,3 Sek,10 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+3,12, #InCh,3,0,255,0,PM_HSV,3 Sek+300,3 Sek,10 Sek,10 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+4,12, #InCh,3,0,255,0,PM_HSV,9 Sek+300,4 Sek,10 Sek,10 Sek,2 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+5,12, #InCh,3,0,255,0,PM_HSV,8 Sek+300,6 Sek,10 Sek,10 Sek,5 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+6,12, #InCh,3,0,255,0,PM_HSV,7 Sek+300,7 Sek,10 Sek,10 Sek,7 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+7,12, #InCh,3,0,255,0,PM_HSV,300,8 Sek,10 Sek,8 Sek,8 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+8,12, #InCh,3,0,255,0,PM_HSV,300,8 Sek,11 Sek,11 Sek,9 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+9,12, #InCh,3,0,255,0,PM_HSV,300,8 Sek,12 Sek,12 Sek,10 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+10,12, #InCh,3,0,255,0,PM_HSV,300,8 Sek,13 Sek,13 Sek,11 Sek,500,15,255,255,240,255,255,0,0,0) \
APatternT6(#LED+11,12, #InCh,3,0,255,0,PM_HSV,300,8 Sek,14 Sek,14 Sek,12 Sek,500,15,255,255,240,255,255,0,0,0)

```

3.1.5 Schalten per DCC oder Taster

Zum Schalten von Aktionen über DCC, Selectrix oder CAN und Taster gibt es verschiedene Varianten. In Folgenden wird Stellvertretend für alle drei von einer Zentrale kommenden Befehle DCC verwendet.

Die einfachste Variante funktioniert sowie die Lichter Zuhause welche man mit verschiedenen Tastern schalten kann. So kann man einen Verbraucher entweder per DCC oder per Taster steuern.

Dazu wird mit der „Logic“ Funktion eine ODER Verknüpfung der beiden Eingabemethoden implementiert. Die Ergebnisvariable Steuert das oben beschrieben T-Flip-Flop. Der DCC Eingang wird dazu als Taster („Rot“ oder „Grün“) konfiguriert.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		1	Grün █		DCC Taster ODER HW Taster D1			Logic(DCCuHW_T, #InCh OR SwitchD1)
✓		DCCuHW_T			„Stromstoß Relais“			T_FlipFlopResetTimeout(DCCuHW_T_Res, #InCh, SI_0, 0 Sec)
✓		DCCuHW_T_Res			Ansteuerung eines Verbrauchers			Const(#LED, C2, #InCh, 0, 127)

Das SI_0 in der „T_FlipFlopReset“ Funktion ist eine spezielle Eingangsvariable (Englisch Special Input) welche immer 0 ist. Sie wird ist bei dem Parameter „Eingangsvar. Reset“ der Funktion eingetragen. Das bedeutet, dass der Reset-Eingang der Funktion nie betätigt wird.

Anstelle von SI_0 in der zweiten Zeile könnte man noch einen anderen Eingang "anschließen" mit dem man den Ausgang per Tastendruck abschalten kann. Damit könnte man auch einen globalen Ausschalter für alle Funktionen definieren. Dazu muss der zweite DCC Taster über die „Define Input()“ Funktion definiert werden damit man anschließend auf die Variable „INCH_DCC_1_RED“ zugreifen kann.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		1	Grün		DCC Taster ODER HW Taster D1			Logic(DCCuHW_T, #InCh OR SwitchD1)
✓		1	Rot		DCC Eingang definieren für folgende Zeile			// Define Input()
✓		DCCuHW_T			"Stromstoß Relais"			T_FlipFlopResetTimeout(DCCuHW_T_Res, #InCh, INCH_DCC_1_RED, 0 Sec)
✓		DCCuHW_T_Res			Ansteuerung eines Verbrauchers			Const(#LED, C2, #InCh, 0, 127)

Zwei Taster (Rot/Grün) bei DCC und einen Hardware Taster:

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1			HW Taster steuert "Stromstoß Relais"			T_FlipFlopReset(FF_Res, #InCh, SI_0)
✓		1	AnAus	0	DCC Schalter ODER "Stromstoß Relais"			Logic(DCC_OR_FF, #InCh OR FF_Res)
✓		DCC_OR_FF			Ansteuerung eines Verbrauchers			Const(#LED, C2, #InCh, 0, 127)

Bei dieser Variante ist man sicher, dass die Funktion eingeschaltet wird, wenn man auf Grün drückt. Allerdings kann man den die Funktion auch über der HW Taster einschalten und hat dann keine Möglichkeit mehr zum Abschalten per DCC

Man kann auch einen Kreuzschalter implementieren. Dazu benötigt man eine XOR Logik.

XOR: (A AND NOT B) OR (B AND NOT A)

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1			HW Taster steuert "Stromstoß Relais"			T_FlipFlopReset(FF_Res, #InCh, SI_0)
✓		1	AnAus	0	XOR Verknüpfung			Logic(DCC_XOR_FF, #InCh AND NOT FF_Res OR FF_Res AND NOT #InCh)
✓		DCC_XOR_FF			Ansteuerung eines Verbrauchers			Const(#LED, C2, #InCh, 0, 127)

Hier kann man den Verbraucher mit beiden Eingabemethoden umschalten. Allerdings sieht man am DCC Zustand nicht ob der Verbraucher Ein- oder Ausgeschaltet ist.

Natürlich kann man DCC Kommandos und Hardware Taster auch zusammen mit den PushButton Funktionen benutzen. Das hat den Vorteil einer Status LED welche anzeigt in welchen Zustand sich die Aktion gerade befindet.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		1	Grün		DCC Taster ODER HW Taster D1			Logic(DCCuHW_T, #InCh OR SwitchD1)
✓		DCCuHW_T			"Stromstoß Relais"			PushButton_w_LED_BL_0_1(#LED, C1, #InCh, DCCuHW_T_Res0, 1, 1, 0, 0, 30 Sek, 127, 31)
✓		DCCuHW_T_Res1			Ansteuerung eines Verbrauchers			Const(#LED, C2, #InCh, 0, 127)

3.1.6 Signal Ansteuerung

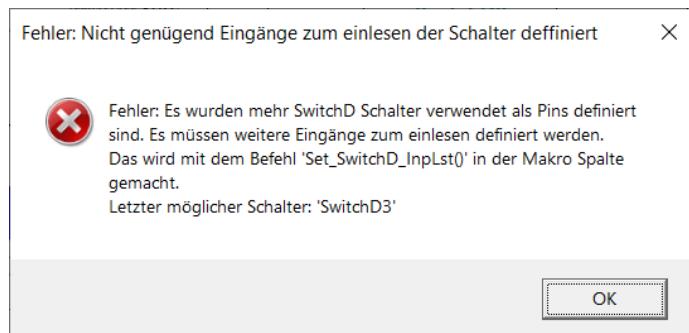
Mit den Tastern können auch Komplexere Funktionen mit mehreren Eingängen angesteuert werden. Signale benötigen meistens mehrere Eingänge. Das Einfahrtssignal mit drei Zuständen („EntrySignal3_RGB“) wird über drei Eingänge gesteuert.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1			Einfahrtssignal mit drei Zuständen			EntrySignal3_RGB(#LED, #InCh)

Da in der Spalte „Adresse oder Name“ nur ein Schalter angegeben werden kann benutzt das Programm automatisch die folgenden Eingänge. Das Signal im Beispiel oben wird über den Schalter „SwitchD1“ bis „SwitchD3“ gesteuert. Der erste Taster aktiviert den Zustand „Hp0“ (rotes Licht), der Zweite den Zustand „Hp1“ (grünes Licht) und der Dritte den Zustand „Hp2“ (grünes und gelbes Licht).

Die Funktion „EntrySignal3_RGB“ nur für Tests mit RGB LEDs gedacht. Zur Ansteuerung „richtiger“ Signale auf der Anlage wird die Funktion ohne den Zusatz „RGB“ verwendet welche einzelne LEDs über WS2811 Module ansteuert.

Wenn man im Beispiel oben die Funktion „DepSignal4_RGB“ verwendet, dann erkennt das Programm, dass die Anzahl verfügbaren Schalter nicht ausreicht und generiert die folgende Meldung, weil für dieses Signal 4 Eingänge benötigt werden.



Signal Ansteuerung mit DCC oder Tastern

Wenn ein Signal entweder per DCC Signal oder per Taster gesteuert werden soll, dann kann man die beiden Eingangskanäle über die „Logic“ Funktion ODER verknüpfen:

Aktiv	Filter	Adresse oder Name	Typ	Start- wert	Beschreibung	Verteiler- Nummer	Stecker- Nummer	Beleuchtung, Sound, oder andere Effekte
✓		1	Rot	■	Oder Verknüpfung DCC, SwitchD1			Logic(Sig0 , #InCh OR SwitchD1)
✓		1	Grün	■	Oder Verknüpfung DCC, SwitchD2			Logic(Sig1 , #InCh OR SwitchD2)
✓		2	Rot	■	Oder Verknüpfung DCC, SwitchD3			Logic(Sig2 , #InCh OR SwitchD3)
		Sig0			Einfahrt Signal			EntrySignal3_RGB(#LED, #InCh)

Etwas einfacher geht es mit dem folgenden Trick:

Aktiv	Filter	Adresse oder Name	Typ	Start- wert	Beschreibung	Verteiler- Nummer	Stecker- Nummer	Beleuchtung, Sound, oder andere Effekte	Start LedNr	LEDs
✓		5 - 6	Rot	■ ■ ■ ■	Einfahrt signal per DCC gesteuert LED Nr. -3 => Nächste Zeile gleiche LEDs			EntrySignal3_RGB(#LED, #InCh) // Next_LED(-3)	1	3
✓		SwitchD1			Einfahrt signal per Taster gesteuert			EntrySignal3_RGB(#LED, #InCh)	4	-3
									1	3

Hier wird die Start LedNr der nächsten LED mit dem Mako „Next_LED(-3)“ zurückgesetzt. Damit benutzt die zweite „EntrySignal3_RGB(#LED, #InCh)“ Zeile die gleichen LEDs wie die erste Zeile. Man erkennt das an der Spalte „Start LedNr“. Beide Einfahrtssignale starten mit LED-Nummer 1. Dieser Trick ist möglich, weil die MobaLedLib eine LED nur dann neu schreibt, wenn sich etwas an der entsprechenden LED verändert hat. Das ist ein ganz wichtiges Konzept mit dem die Auslastung des winzigen Arduino Prozessors klein gehalten wird.

3.2 Konfiguration der benutzen Eingänge

Die Arduino Anschlüsse welche zum einlesen der Schalter verwendet werden können frei konfiguriert werden. Damit kann man das Programm an die eigenen Bedürfnisse anpassen. So können die Anzahlen der in den 4 verschiedenen Gruppen verfügbaren Schalter individuell bestimmt werden.

Ein Arduino Nano hat nur eine beschränkte Anzahl von Anschlüssen darum muss man sich überlegen

- wie viele Schalter man benötigt
- wo diese Schalter eingebaut werden
- ob es möglich sein soll, dass mehrere Schalter gleichzeitig betätigt werden.
- ob die Anzahl der Schalter später erweitert werden soll

Alle 4 Varianten haben Vor- und Nachteile.

- A. Analoge Taster:
 - + Extrem einfache da keine zusätzliche Platine benötigt wird
 - + 10 Taster pro Arduino Pin
 - Nur für Taste, Gleichzeitige Betätigung nicht möglich
 - Störanfälliger
- B. Taster (oder Kippschalter) am Anlagenrand:
 - + Beliebig erweiterbar
 - + Zustandsanzeige per (RGB) LED-Beleuchtung bereits vorgesehen
 - + Gleichzeitige Betätigung möglich => Kippschalter oder Taster möglich
 - + Erweiterung durch zusätzliche Platinen möglich, ohne das weitere Pins benötigt werden
 - Zusätzliche Platine nötig
- C. Weichenstellpult:
 - + Sehr viele Schalter möglich
 - + Erweiterung durch zusätzliche Platinen möglich, ohne das weitere Pins benötigt werden
 - + Gleichzeitige Betätigung möglich => Kippschalter oder Taster möglich
 - Zusätzliche Platine nötig
- D. Schalter der Hauptplatine:
 - + Aller einfachste Lösung (Schalter bereits auf der Hauptplatine vorhanden)
 - + Gleichzeitige Betätigung möglich => Kippschalter oder Taster möglich
 - Nur ein Schalter pro Arduino Pin

Der Arduino Nano verfügt nur über 8 analoge Eingänge. Einer davon (A7) ist für die Messung der Umgebungshelligkeit vorgesehen. Der Eingang A1 wird zur Steuerung der Kommunikation zwischen LED Arduino und DCC>Selectrix Arduino benötigt. Die anderen 6 Anschlüsse können für verschiedene Aufgaben verwendet werden. Insbesondere für das einlesen der verschiedenen Schalter.

Man muss ich entscheiden welche Anschlüsse den benötigten Funktionen zugeordnet werden.

Das Einlesen der Schalter mit der MobaLedLib kann frei Konfiguriert werden. Dazu existieren eine Reihe von „Set“ Befehlen mit denen die Anschlusspins definiert werden können:

```
Set_SwitchA_InpLst(A6)
Set_SwitchB_InpLst(A2)
Set_SwitchC_InpLst(2 10 11 12 A5)
Set_SwitchD_InpLst(7 8 9)
Set_CLK_Pin_Number(A0)
Set_RST_Pin_Number(A3)
Set_LDR_Pin_Number(A7)
Set_LED_OutpPinLst(6 A4)
```

Mit den „..InpLst“ Kommandos definiert man eine Liste von Arduino Pins welche für die jeweilige Gruppe (A-D) verwendet werden sollen.

Die Beispiele in den Klammern entsprechen den Standard Einstellungen. Sie müssen nicht extra definiert werden. Die Zeilen müssen nur dann verwendet werden, wenn man davon abweichende Nummern verwenden will.

3.2.1 Konfiguration der analogen Taster

Wenn man den Befehl „Set_SwitchA_InpLst(A6 A7)“ benutzt, dann werden zwei analoge Leitungen zum Einlesen von Tastern bereitgestellt. Damit können bis zu 20 Taster eingelesen werden.

Allerdings kann der Helligkeitssensor (LDR) dann nicht gleichzeitig benutzt werden. Alternativ könnte auch Pin A5 benutzt werden. Diesem Anschluss ist keine bestimmte Funktion zugeordnet. Dem Befehl „Set_SwitchA_InpLst(A6 A7)“ können natürlich nur analoge Kanäle des Arduinos zugewiesen werden. Maximal können 8 analoge Kanäle benutzt werden.

Eine Liste der Arduino Pins und deren möglichen Funktionen bei der MobaLedLib findet man im Abschnitt „Tabelle: Nutzung der Arduino Pins“.

Alle Eingänge benötigen einen „Pull-Up“ Widerstand geben +5V und einen 1uF Kondensator gegen Masse. Am Eingang A3 kann Heartbeat LED für den CAN Mode angeschlossen werden. Diese darf nicht bestückt sein wenn der Eingang zum einlesen von Tastern benutzt wird.

Die MobaLedLib verwendet nicht Standard analogRead() Funktion von Arduino weil diese relativ viel Zeit benötigt (100µs). Stattdessen wird die „AnalogScanner“ Bibliothek eingesetzt welche die Kanäle per Interrupt abfragt.

3.2.2 Methode B und C

Die Methode B und C sind sehr ähnlich. Sie teilen sie sich auch die Takt- und Reset Leitung. Sie unterscheiden sich zunächst einmal nur in der Anzahl der verwendeten Kanäle zum einlesen der Schalter. Mit dem Kommando „Set_SwitchB_InpLst(A2)“ und „Set_SwitchC_InpLst(2 10 11 12 A5)“ werden die Anschlüsse zum einlesen der Schalter definiert. Die Takt und die Reset Leitung werden mit dem Befehlen „Set_CLK_Pin_Number(A0)“ und „Set_RST_Pin_Number(A3)“ bestimmt. Wenn die Hauptplatine der MobaLedLib verwendet wird, dann müssen diese Zeilen nicht angegeben werden da diese Pins standardmäßig verwendet werden.

Die Schaltergruppe B ist nicht zwangsläufig auf einen Schalter Anschluss beschränkt. Es können genauso wie beim Typ C mehrere Anschlussnummern angegeben werden. Damit könnte man zwei verschiedene Schaltpulte mit mehreren Tasten verwenden indem man die verfügbaren Anschlüsse auf zwei Gruppen aufteilt:

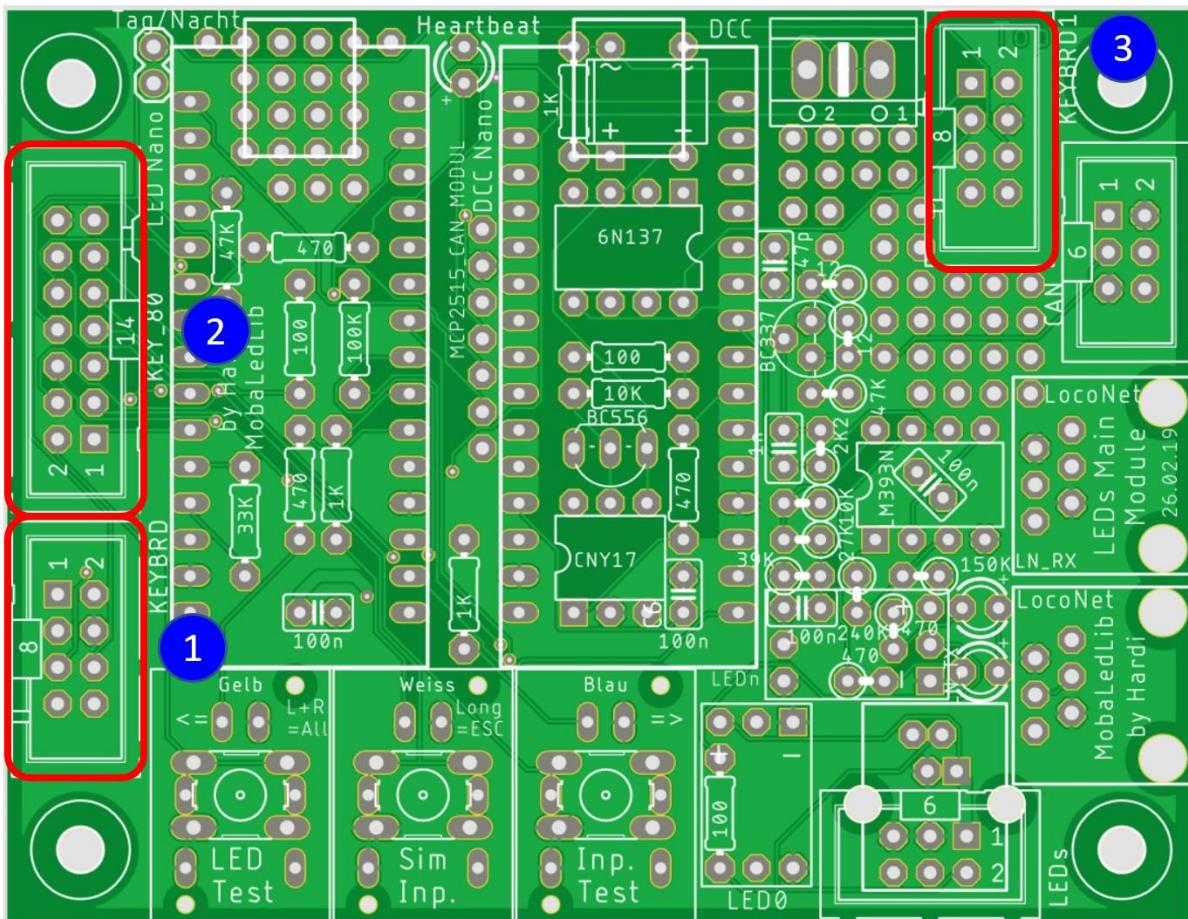
```
Set_SwitchB_InpLst(7 8 9 10 11 12)  
Set_SwitchC_InpLst(2 A2 A5 A6)
```

So kann ein Schaltpult mit 60 Schaltern und eins mit 50 Schaltern realisiert werden. In jedem Pult wird dabei ein 4017 verwendet. Durch Kaskadierung mehrerer PushButton_4017 Platinen können die Schalteranzahlen erhöht werden.

Wichtig ist, dass jede „Schalter“ Leitung zum Arduino mit einen 47K Widerstand auf Masse gezogen wird, wenn kein Taster betätigt wird. Diese Widerstände sind bis auf einen (Leitung A2) nicht auf der Hauptplatine vorhanden und müssen im Weichenstellpult verbaut werden.

Zwei Stränge für Taster am Anlagenrand

Auf der Hauptplatine befinden sich drei Stecker für den Anschluss der Schalter. Die Schalter am Anlagenrand (Variante B) werden über den Stecker 1 angeschlossen. Wenn man eine größere Anlage besitzt und die Hauptplatine zentral positioniert hat, dann kann man den Kabelstrang zu den Schaltern in zwei Abschnitte unterteilen. Der eine Strang wird an den Stecker 1 angeschlossen, der zweite Strang an den Stecker 3. So kann ein Strang die linke Seite der Anlage abdecken und der zweite Kabelstrang die rechte Seite. Bei der letzten PushButton_4017 Platine des ersten Strangs muss der Lötjumper „END“ verbunden werden damit die LEDs des zweiten Strangs funktionieren. Außerdem muss der Pin 5 und 8 des „OUTP“ Steckers gebrückt werden. Damit wird das Taktsignal weitergegeben. Hierfür existiert noch kein Lötjumper.



Wenn die Schalter eines Weichenstellpults eingelesen werden sollen, dann benutzt man die Stecker 1 und 2. Der erste Stecker wird mit der PushButton_4017 Platine verbinden. Der Stecker zwei geht zu den Rückleitungen der Schalter und der 47K Pull Down Widerstände. Die Dioden werden vom Stecker „BUT10“ der PushButton_4017 Platine versorgt.

Weichenstellpult und Taster am Anlagenrand

Wenn gleichzeitig Schalter am Anlagenrand und ein Weichenstellpult benutzt werden sollen, dann werden zwei Kabel am Stecker 1 verwendet. Dazu quetscht man einfach ein Flachkabel so an den Stecker, dass auf beiden Seiten ein Kabelende herauskommt. Das eine führt zum Weichenstellpult, das andere Kabelende zur ersten Platine der Schalter am Anlagenrand.

Alternativ könnte der Stecker 3 auch zum Weichenstellpult geführt werden. Dann müssen allerdings die Takteleitungen (Pin 5) der Stecker 1 und 3 verbunden werden. Dazu existiert auch noch kein Lötjumper.

3.3 Konfiguration der Taster direkt auf der Hauptplatine

Mit der vierten Methode werden Schalter abgefragt welche direkt an der Hauptplatine angeschlossen sind. Gleichzeitig aktive Schalter können erkannt werden.

Zunächst mal sind das die drei Taster auf der Platine. Sie werden von links nach rechts mit „SwitchD1“, „SwitchD2“ und „SwitchD3“ abgefragt. Aber auch diese Gruppe kann über den „Set_SwitchD_InpLst()“ Befehl konfiguriert werden, wenn mehr oder weniger Schalter abgefragt werden sollen. Dieses Verfahren kann mit fast jeder Arduino Pin benutzt werden. Lediglich bei dem Pin A6 gibt es eine kleine Einschränkung: Hier muss ein zusätzlicher Pull-Up Widerstand (22K)

verwendet werden wie das bei den analogen Tastern gezeigt ist. Das gilt aber nur für die Platinen Version von 2019. In der nächsten Version wird für dieses Bauteil ein Platz vorhanden sein. Bei A7 ist dieser Pull-Up Widerstand bereits vorhanden.

Die Pins der drei auf dem Main Board vorhandenen Taster sind auch auf dem Stecker „KEY_80“ (2) zu finden. Außerdem liegen dort noch 9 weitere Anschlüsse des Arduinos welche man zum einlesen von Schaltern verwenden könnte. In Summe kann man sehr einfach 12 Schalter oder Taster direkt an die Hauptplatine anschließen. Allerdings können dann keine weiteren Schalter mehr benutzt werden. Lediglich zwei Gruppen analoger Schalter an Pin A6 und A7 sind noch möglich. Wobei A7 auch nur dann verfügbar ist, wenn kein Helligkeitssensor benutzt wird.

Die Schalter werden einfach die entsprechenden Pins angeschlossen. Es sind keine weiteren Bauteile nötig. Der zweite Anschluss der Schalter wird mit Masse (Pin 14 des „KEY_80“ Steckers) verbunden. Man darf allerdings nicht vergessen, dass sie über den Befehl „Set_SwitchD_InpLst()“ aktiviert werden müssen.

3.4 Konfiguration weiterer Arduino Pins

Mit dem „Set_LDR_Pin_Number“ wird bestimmt welcher analoger Eingang zur Messung der Umgebungshelligkeit benutzt wird. Standardmäßig wird dazu der Anschluss A7 benutzt.

Über den Befehl „Set_LED_OutpPinLst(6 A4)“ wird festgelegt mit welchen Ausgängen des Arduinos die WS281x LEDs angesteuert werden.

3.5 Tabelle: Nutzung der Arduino Pins

Die folgende Tabelle zeigt die Nutzungsmöglichkeiten der verschiedenen Arduino Pins zum einlesen der Schalter. Die ersten 4 Zeilen enthalten ein „X“ für die Arduino Anschlüsse mit denen man die Entsprechende Funktion nutzen kann. In der Zeile „Analoge Taste“ erkennt man, dass man dafür nur die analogen Eingänge A0-A7 nutzen kann. Das ist eigentlich klar.

Die Einträge in Blau sind die Standard Funktionen. Eigentlich sollte in Jeder Spalte nur ein blaues Zeichen stehen. Das bedeutet, dass diese Funktionen nicht gleichzeitig genutzt werden können.

Das „#“ Zeichen markiert Pins welche benutzt werden müssen wenn die nebenstehende Funktion verwendet wird. Das gilt z.B. für die Pins A0 und A3. A0 generiert den Takt für die PushButton_4017 Platine und A3 das Reset Signal. Solange man die MobaLedLib Hauptplatine verwendet sind die Pin Nummern vorgegeben. Die Einträge sind in Zeile B und C vorhanden, weil die Anschlüsse von beiden Gruppen benötigt werden.

Mit „(x)“ werden die Pins markiert welche für die Funktionalität genutzt werden kommen, bei denen es aber Einschränkungen gibt. Dabei wurde nicht zwischen großen und kleinen Einschränkungen unterschieden da das schwierig zu beurteilen ist. Es muss anhand des Schaltplans geklärt werden ob eine Einschränkung wichtig ist oder nicht.

Gruppe	Funktion	Pin	0	1	2	3	4	5	6	7	8	9	10	11	12	13	A0	A1	A2	A3	A4	A5	A6	A7
A	Analoge Taste															X	X	X	X	X	X	X	X	
B	Schalter am Anlagenrand				(X)	#	(X)	X	#	(X)	(X)	(X)	(X)											
C	Weichenstellpult				X	(X)	(X)	(X)	(X)	(X)	(X)	X	X	X	(X)	#	X	X	#	X	X	(X)	(X)	
D	Schalter auf Hauptplatine		(X)	X	(X)	(X)	(X)	(X)	(X)	X	X	X	X	X	X	X	X	X	(X)	X	X	(X)	(X)	
	RGB LEDs auf der Anlage									#														
	RGB LEDs Schalter/Taster																						#	
	Kommunikation LED/DCC Nano							#												#				
	Helligkeitssensor																							#
	CAN Bus / SPI Kommunikation															#	#	#	#					
	Stecker 1: KEYBOARD																		5	7	6	1		
	Stecker 2: KEY_80															2	3	4	5	6	7			
	Stecker 3: KEYBOARD1																		8	9	10	11	12	
	Stecker Tag/Nacht																				6		(8)	
																							2	

Muss vorhanden sein wenn die Funktion genutzt wird

X Möglich für die entsprechende Funktion

(X) eingeschränkt nutzbar (Siehe Text)

Blau Standard Pin

Takt B, C

Reset B, C

3.5.1 Eingeschränkt nutzbare Pins

Die Pins 0 und 1 werden von der seriellen Schnittstelle verwendet. Pin 1 könnten nur dann verwendet werden, wenn kein zweiter Arduino (DCC oder Selectrix) verwendet wird.

Die LEDs der Taster auf der Hauptplatine sind an den Anschlüssen 3-5 angeschlossen. Darum können die Kanäle nicht verwendet werden, wenn die LEDs vorhanden sind.

An Pin 13 ist die LED „L“ auf dem Nano angeschlossen. Darum kann dieser Anschluss nur dann mit Gruppe B oder C als Eingang verwendet werden, wenn die LED auf dem Nano entfernt wird.

Die Leitung A3 steuert die Heartbeat LED im CAN Betrieb an. Wenn die LED bestückt ist, dann kann die Leitung auch dann nicht zum einlesen eines Tasters verwendet werden, weil die LED wie ein geschlossener Schalter wirkt. Darum darf sie nicht bestückt werden, wenn ein Schalter direkt an A3 angeschlossen werden soll.

4 Eigene Variablen

Eine wichtige Neuerung in der Version 1.1.0 der Bibliothek sind die selber definierten Variablen welche in dem Prog_Generator verwendet werden können. Bisher musste man die Variablen per #define Definieren und ihnen eine Nummer zuweisen. Jetzt macht das der Prog_Generator automatisch.

Eine Variable ist ein Speicher in dem Zustände gespeichert werden können. In einer Variable wird das Ergebnis einer Funktion gespeichert. Dieses Ergebnis kann dann an einer anderen Stelle als Eingang benutzt werden.

Beispiel:

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteller-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓	SwitchD1			-	Taster aktiviert Zeitschalter für 1 Minute			ButtonFunc(Haus7, #InCh, 1 Min)
✓		Haus7		-	Variable "Haus7" steuert das Haus an			HouseT(#LED, #InCh, 1, 2, 1, 2, ROOM_BRIGHT, ROOM_WARM_W)

Bei der „ButtonFunc()“ wird ein kurzer Tastendruck für eine einstellbare Zeit gespeichert. Während dieser Zeit ist die Ergebnisvariable gesetzt (1). Damit kann man dann die Beleuchtung eines Hauses ansteuern. Die Variable zur Eingangsvariable für die „House()“ Funktion. Solange die Variable gesetzt ist werden die Lichter im Haus angeschaltet.

Eine Variablenname besteht aus einem beliebigen „Wort“ welches mit einem Buchstaben beginnt und nur aus Buchstaben, Zahlen und dem „_“ besteht. Sonderzeichen wie „äöü &!.,+“ ... und

Leerzeichen dürfen **nicht** verwendet werden.

Variablennamen sollten möglichst Aussagekräftig sein. Der Name soll beschreiben was die Variable macht. „Eckhaus_Waldstr_7“ wäre beispielsweise ein geeigneter Name.

Die Variablennamen dürfen aber nicht an anderer Stelle im C Programm oder in der Konfiguration vorhanden sein. So ist z.B. der Name „Logic“ nicht erlaubt, weil es ein Makro mit diesem Namen gibt. Genau so wenig ist der Variablenname „new“ möglich da das ein Befehl in C++ ist. Der Name „New“ dagegen ist erlaubt. Das liegt daran, dass C++ zwischen Groß- und Kleinschreibung unterscheidet. Das bedeutet aber auch, dass man genau auf die Schreibweise einer Variable achten muss. Die Gültigkeit eines Namens kann innerhalb von Excel nur sehr schwer geprüft werden darum kann es passieren, dass ein Fehler erst beim schicken zum Arduino erkannt wird.

Variablen können im Prog_Generator an zwei Stellen verwendet werden. Zum einen in der Spalte „Adresse oder Name“, zum anderen in der Spalte „Beleuchtung, Sound, oder andere Effekte“:

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1			Ein- Ausschalter mit zwei Tasten			RS_FlipFlopTimeout(HausNeon, SwitchD2, #InCh, 2 Min)
✓		HausNeon			Haus mit 3 Neon Lichtern			House(#LED, #InCh, 1, 2, NEON_LIGHTD, NEON_LIGHTD)

In dem Beispiel sind die Variablen in Grün dargestellt. Die Variable „HausNeon“ wird in der Spalte „Beleuchtung, Sound, oder andere Effekte“ geschrieben und in der darauffolgenden Zeile in Spalte „Adresse und Name“ wieder gelesen um damit das Haus zu steuern. Die Variablen links werden als Eingang für die Funktion der Zeile benutzt.

Auf Variablen kann auch in der rechten Spalte lesender Weise zugegriffen werden. Das ist z.B. bei der „Logic“ Funktion oder bei der „RS_FlipFlop“ Funktion der Fall.

Es gibt Funktionen welche mehrere Variablen beschreiben. Hier gibt es verschiedene Varianten:

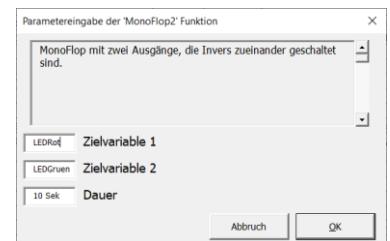
1. Eine oder zwei Zielvariablen

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1						MonoFlop2(LEDRot, LEDGruen, #InCh, 10 Sek)
✓		LEDRot						Const(#LED, C1, #InCh, 0, 127)
✓		LEDGruen						Const(#LED, C2, #InCh, 0, 127)

Ein „MonoFlop“ ist wie die „ButtonFunc()“ ein Zeitschalter. Die Zeit beim „MonoFlop“ wird aber im Gegensatz zur „ButtonFunc()“ im Moment der Betätigung des Tasters gestartet. Bei der „ButtonFunc()“ läuft die Zeit erst wenn der Taster losgelassen wird.

Bei der „ButtonFunc()“ gibt es nur eine Ausgangsvariable. Der „MonoFlop“ dagegen hat zwei Ausgänge. Die den Ausgängen zugewiesenen Variablen schalten invers zueinander. Wenn der eine Ausgang aus ist dann ist der andere an.

Bei Funktionen mit einer oder zwei Ausgangsvariablen werden die Namen der Variablen direkt angegeben.



2. Array von Zielvariablen mit fester Anzahl

Bei Funktionen welche mehrere Ausgänge haben soll nicht jede Variable einzeln angegeben werden. Hier wird nur die erste Variable angegeben. Das erleichtert die Eingabe und spart Speicherplatz im Arduino. Der Name der Variablen muss mit einer Zahl enden. Im folgenden Beispiel wird „PushBRes0“ verwendet.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓		SwitchD1			Knopf Druck Aktion mit 3 Zuständen			PushButton_w_LED_BL_0_3(#LED, C3, #InCh, PushBRes0, 1, 1, 0, 30 Sek, 127, 31)
✓		PushBRes0			Zustand 0			ConstRGB(#LED, #InCh, 0, 0, 0, 50, 0, 0)
✓		PushBRes1			Zustand 1			ConstRGB(#LED, #InCh, 0, 0, 0, 50, 50, 0)
✓		PushBRes2			Zustand 2			ConstRGB(#LED, #InCh, 0, 0, 0, 0, 50, 0)
✓		PushBRes3			Zustand 3			ConstRGB(#LED, #InCh, 0, 0, 0, 50, 0, 50)

Die hier verwendete „PushButton“ Funktion hat 3 aktive Zustände. Dazu kommt noch der Ausschaltzustand. Die Zahl im Variablennamen wird mit jedem Zustand hochgezählt. Die Nummer der übergebenen Zielvariablen bekommt der Ausschaltzustand. Darum ist es Sinnvoll, wenn man mit der Nummer 0 beginnt. Die Folgenden Zustände bekommen die Nummern 1 bis 3. Dass diese Funktion 4 Ausgangsvariablen benutzt erkennt man am Namen der Funktion: „PushButton_w_LED_BL_0_3“. Die Anzahl der Zielvariablen ist fest mit der Funktion verknüpft.

3. Array mit Start und End Variable

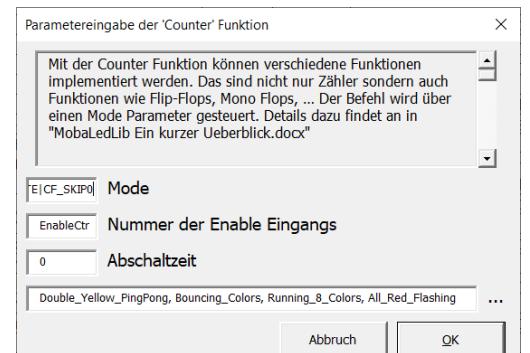
Bei der „RandMux“ Funktion ist die Anzahl der Variablen nicht über die Funktion bestimmt. Damit können beliebig viele Ausgangsvariablen zufällig aktiviert werden. Bestimmt wird die Anzahl der Variablen über die erste und letzte zu verwendende Variable. Man gibt Beispielsweise „Effekt0“ als erste Variable und „Effekt4“ als letzte Variable an. Die Erste Variable ist aktiv, wenn die „RandMux“ Funktion abgeschaltet ist. Das wird in dem folgenden Beispiel nicht gemacht. Die Variablen „Effekt1“ bis „Effekt4“ werden nach einer Zufälligen Zeit zufällig aktiviert.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓					Misha's Kirmes Effekt			#define Brightness 50
✓								RandMux(Effekt0, Effekt4, #InCh, RM_NORMAL, 5 Sek, 10 Sek)
✓		Effekt1			Double_Yellow_PingPong			PatternT1(#LED,4,#InCh,24,0,Brightness,0,PM_NORMAL,101 ms,1
✓					LED Nr -8 => Nächste Zeile gleiche LEDs			// Next_LED(-8)
✓		Effekt2			Bouncing_Colors			PatternT1(#LED,4,#InCh,24,0,Brightness,0,PM_NORMAL,105 ms,1
✓					LED Nr -8 => Nächste Zeile gleiche LEDs			// Next_LED(-8)
✓		Effekt3			Running_8_Colors			APatternT1(#LED,8,#InCh,24,0,Brightness,0,PM_NORMAL,107 ms,
✓					LED Nr -8 => Nächste Zeile gleiche LEDs			// Next_LED(-8)
✓		Effekt4			All_Red_Flashing			PatternT1(#LED,4,#InCh,24,0,Brightness,0,PM_NORMAL,104 ms,0

4. Liste von Zielvariablen

Die „Counter“ Funktion ist ein Beispiel für eine Funktion bei der die Anzahl der Ausgänge variabel ist. Hier bestimmt man die Anzahl der Ausgänge indem man für jede Zielvariable einen eigenen Namen im Dialog eingibt. Die Zahl der Variablen ist hier von der Eingabe des Benutzers abhängig.

In dem Beispiel wird mit jedem Tastendruck der Zähler um eins erhöht. Damit werden nacheinander die verschiedenen Kirmes Effekte aktiviert.



Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte
✓					Misha's Kirmes Effekt			#define Brightness 50
✓		SwitchD1			Zähler der nacheinander verschiedene Effekte aktiviert			Counter(CF_ROTATE, #InCh, SI_1, 5 Sek, Double_Yellow_PingPong, Bouncing_Colors, Running_8_Colors, All_Red_Flashing)
✓		Double_Yellow_PingPon			Double_Yellow_PingPong			PatternT1(#LED,4,#InCh,24,0,Brightness,0,PM_NORMAL,101
✓		Bouncing_Colors			Startposition der LEDs zurücksetzen			// Next_LED(-8)
✓		Running_8_Colors			Bouncing_Colors			PatternT1(#LED,4,#InCh,24,0,Brightness,0,PM_NORMAL,105
✓					Startposition der LEDs zurücksetzen			// Next_LED(-8)
✓		All_Red_Flashing			Running_8_Colors			APatternT1(#LED,8,#InCh,24,0,Brightness,0,PM_NORMAL,107
✓					Startposition der LEDs zurücksetzen			// Next_LED(-8)
✓					All_Red_Flashing			PatternT1(#LED,4,#InCh,24,0,Brightness,0,PM_NORMAL,104

5 Zusätzliche LED-Stränge

Bisher konnte man einen LED-Strang an die Hauptplatine anschließen. Alle angeschlossenen LEDs, Servos, Sound Module und sonstigen Verbraucher wurden von einem einzigen Arduino Pin angesteuert. Die LEDs werden dabei in einer Kette nacheinander angesprochen.

Durch das Konzept der Rückführung der Datenleitung zur Verteilerplatine ist es möglich, dass trotzdem eine beliebige Verkabelung mit Verzweigungen und Sternpunkten verwendet werden kann.

Wenn man zusätzlich WS281x LEDs zur Statusanzeige der „Knopf Druck Aktionen“ oder zur Zustandsanzeige im Weichenstellpult verwenden will, dann ist es viel praktischer, wenn man mehrere unabhängige LED-Ketten verwenden kann. Die eine Kette verbindet die einzelnen Häuser, Straßenlaternen, Servos ... auf der Anlage miteinander. Eine zweiter LED-Strang läuft entlang der Anlagenkante und steuert die LEDs der Taster an.

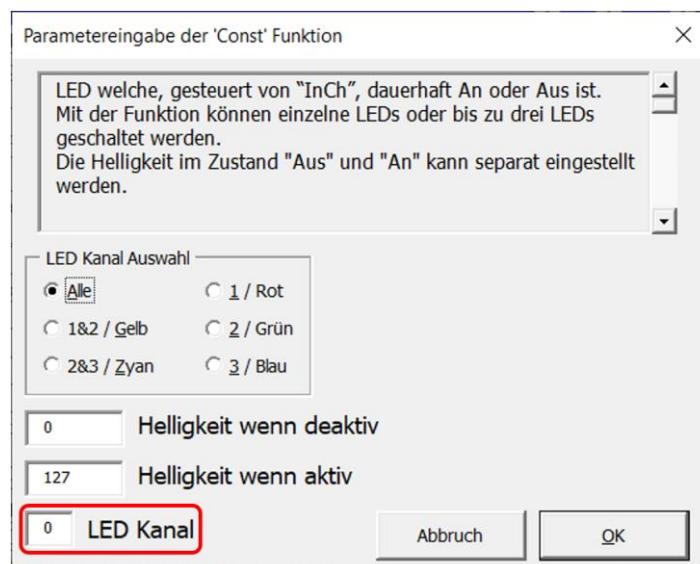
Mit der neuen Version der MobaLedLib können jetzt bis zu 4 unabhängige LED-Ketten betrieben werden. Zu „normalen“ LED-Kette kommt eine weiterer LED-Strang hinzu welcher zu den PushButton_4017 Platinen führt. Er wird von dem Arduino Pin A4 angesteuert und ist bereits in dem 8-poligen „KEYBRD“ Stecker vorhanden. Mit einem entsprechenden Kabel kann dieser LED Ausgang natürlich auch für die anderen Schalter Varianten benutzt werden.

Dieser separate Strang ist auch aus der Sicht der Konfiguration der LEDs nötig. Die Zeilen innerhalb des Prog_Generators müssen so angeordnet sein wie die LEDs im Strang. Durch einen weiteren Strang können die LEDs der „Knopf Druck Aktionen“ unabhängig von den LEDs der „normalen“ Verbraucher angeordnet werden.

Die Taster mit ihren LEDs werden am Anfang der Konfiguration eingetragen. Danach kommen die darüber angesteuerten LEDs des normalen Strangs.

Die Taster geben ihre Befehle über Variablen weiter. Die Variablen werden vom Schalter auf 0 oder 1 gesetzt und dann weiter unten von dem Haus oder einer anderen Zeile gelesen.

Jeder Makro Dialog hat ein zusätzliches Eingabefeld bekommen mit dem der LED-Kanal angegeben werden kann:



Kanal 0 ist der normale Kanal welcher am 4/6-poligen Stecker der Hauptplatine herausgeführt wird.

Der Kanal 1 ist für die LEDs der Taster vorgesehen.

Kanal 2 und 3 haben keine feste Zuordnung. Es sind auch noch keine Arduino Anschlüsse dafür reserviert. Wenn man diese Kanäle benutzen will muss man die zu verwendenden Pins über den Befehl „Set_LED_OutpPinLst(6 A4 x y)“ hinzufügen. „x“ und „y“ werden durch Arduino Pins ersetzt. Dabei kann ein Anschluss nicht für mehrere Funktionen benutzt werden. Er kann nicht gleichzeitig zum einlesen der Tastatur und zur Ansteuerung der LEDs verwendet werden.

5.1 Beispiel mit eigenem LED-Strang für die Taster

In den vorangegangenen „Knopf Druck“ Beispielen wurden der zusätzliche LED-Kanal nicht angesprochen. Das wird hier nachgeholt.

5.2 Zusätzliche Spalten

Zur Verwaltung der zusätzlichen LED-Stränge sind die Spalten „LED Kanal“, „Start Tast LED“, „Start LED G2“ und „Start LED G3“ zu der Tabelle hinzugefügt worden:

Start LedNr	LEDs	InCnt	Loc InCh	LED Kanal	Start Tast LED	Start LED G2	Start LED G3
		1	0				
	3	1	0	1	0		
1	C1-1	1	0	0			
2	C1-1	1	0	0			
3	C1-1	1	0	0			

Diese Spalten werden automatisch vom Programm verwaltet und sollten nicht manuell verändert werden. Sie enthalten Informationen über LEDs und die benutzen Variablen. Die Spalten wurden aus Kompatibilitätsgründen und weil es die Benutzer so gewohnt sind hinten angehängt.

Über die neue Spalte „LED Kanal“ wird der verwendete Kanal bestimmt. Der Kanal 0 ist der Standard Anschluss. Wenn dieser Kanal verwendet wird, dann findet man die Nummer der ersten LED dieser Zeile wie gewohnt in der Spalte „Start LedNr“.

Im Bild oben sieht man, dass in Zeile zwei der neue „LED Kanal“ 1 benutzt wird. Das entspricht dem Strang für die Taster. Darum findet man die Start Nummer der ersten LED dieser Zeile in der Spalte „Start Tast LED“. Hier ist es die 0, da es die erste Zeile der Konfiguration ist welche den LED-Kanal 1 benutzt. Die Zeilen 3 bis 5 nutzen LED-Kanal 0. Hier findet man die Startnummern 1, 2 und 3.

Die Anzahl der benutzten LEDs steht in Spalte „LEDs“. Im Beispiel oben werden von der zweiten Zeile 3 LEDs verwendet. Die Angabe „C1-1“ in den folgenden „LEDs“ Spalten bedeutet, dass diese Makros nur eine (die rote LED) der RGB LEDs eines WS281x Moduls benutzen. Das „C“ steht für Channel = Kanal in Englisch. Dieser Kanal bezieht sich auf die drei einzelnen LEDs einer RGB LED. 1 = Rot, 2 = Grün, 3 = Blau. Die Angabe „C1-2“ würde bedeuten, dass zwei einzelne LEDs verwendet werden (Rot und Grün).

5.3 RGB LED zur Visualisierung des Push Button Zustands

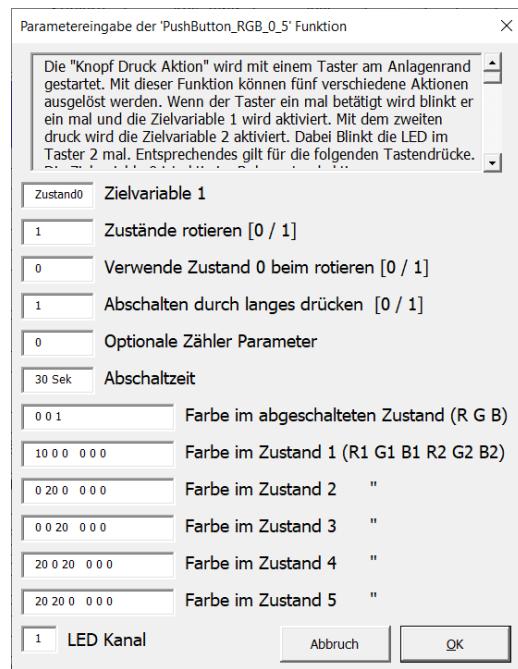
Mit der Funktion „PushButton_RGB_0_5“ können „Knopf Druck Aktionen“ mit fünf Zuständen und einer RGB LED als Zustandsanzeige genutzt werden. Die Funktion gibt es in verschiedenen Ausführungen mit einem bis fünf Zuständen was über die letzte Zahl bestimmt wird. „PushButton_RGB_0_2“ hat beispielsweise zwei aktive Zustände.

Der Nebenstehende Dialog zeigt die Konfigurationsmöglichkeiten.

Die ersten 6 Zeilen entsprechen den Eingaben der PushButton Funktion für einzelne LEDs (Siehe oben).

Wenn RGB LEDs zur Zustandsanzeige verwendet werden hat man viel mehr Möglichkeiten zur Unterscheidung der Zustände. Hier kann ein Zustand über eine beliebige Farbe und Helligkeit visualisiert werden. Parallel dazu kann die LED auch noch blinken.

Mit dem Makro „PushButton_RGB_0_...“ wird ein Teil der vielfältigen Möglichkeiten genutzt. Jeder Zustand außer dem Ruhezustand kann über zwei Farben angezeigt werden. Diese Farben werden im Wechsel mit einer Periode von 1 Hz angezeigt. Wenn die LED nicht blinken soll, dann verwendet man zweimal die gleiche Farbe. Die Farben werden als RGB Werte im Bereich von 0 bis 255 angegeben. Im Beispiel oben Blinkt die LED schwach Rot, wenn Zustand 1 aktiv ist. Im zweiten Zustand blinkt sie Grün. Danach Blau, Lila und Gelb.



5.4 Individuelle Zustandsanzeige mit dem Pattern_Configurator

Die Zustandsanzeige der „PushButton“ kann ganz beliebig über den Pattern_Configurator erstellt werden. Damit ist jede Beliebige Kombination aus Farben, Blinken und verschiedenen LEDs möglich. Man könnte damit einen LED Ring ansteuern welcher über unterschiedlich schnell rotierende LEDs den Zustand einer besonderen Funktion anzeigt. In dem Folgenden Beispiel werden nur drei LEDs verwendet welche im Zustand 1 als grünes Lauflich und im Zustand 2 als Lila Lauflicht in gegenrichtung angesteuert werden. Die Funktion benutzt dazu das weiche Überblenden der Helligkeiten. In der Excel Tabelle kommt dazu nur die Zeile mit dem LED Muster („XPatternT1“) hinzu. Anstelle der bisher benutzten „PushButton“ Funktionen wird hier eine Variante ohne integrierte LED ansteuerung verwendet.

Aktiv	Filter	Adresse oder Name	Typ	Start-wert	Beschreibung	Verteiler-Nummer	Stecker-Nummer	Beleuchtung, Sound, oder andere Effekte	Start-LedNr	LEDs	InCh	Loc-InCh	LED-Kanal	Start-Tast-Led
✓		SwitchD1			Knopf Druck Aktion mit 2 Zuständen ohne integrierte Status LEDs Eigene Status Anzeige über 3 RGB LEDs			PushButton_0_2(#InCh, Hallo0, 1, 1, 1, 0, 30 Sek)				1	0	
✓					Zustand 0			XPatternT1(#LED,140,SI_LocalVar,9,0,255,0,0,0.5 Sec,0)		3	1	0	1	0
✓		Hallo0			Zustand 1			Const#LED_C1,#InCh,0,127)		1	C1-1	1	0	0
✓		Hallo1			Zustand 2			Const#LED_C1,#InCh,0,127)		2	C1-1	1	0	0
✓		Hallo2						Const#LED_C1,#InCh,0,127)		3	C1-1	1	0	0

Das zugehörige LED-Muster zur Ansteuerung der LEDs ist schnell über den Pattern_Configurator erstellt.

Für die Aufgabe werden 9 einzelne LEDs benötigt. Diese trägt man in dem Feld „Anzahl der Ausgabe Kanäle“ ein. Daraufhin erscheint unten eine leere Tabelle in welche eingetragen wird wann eine einzelne LED leuchten soll. Ein x bedeutet Maximale Helligkeit. Die „1“ steht für die geringste Intensität. Der tatsächliche Wert ergibt sich aus der Anzahl der gewählten Helligkeitsstufen und den Min und Max Werten.

Wichtig für die Aktivierung der einzelnen Bereiche passend zu den Zuständen der „PushButton“ Funktion passend ist die „Goto Tabelle“. Sie wird eingeblendet, wenn im Feld „Goto Mode“ eine 1 eingegeben wird.

Die Daten werden in den Prog_Generator übertragen indem man auf den „Programm Generator“ Knopf klickt.

Auf diese Art und Weise können beliebige Anzeigen mit einer einzelnen LED oder einer oder mehrerer RGB LEDs erstellt werden.

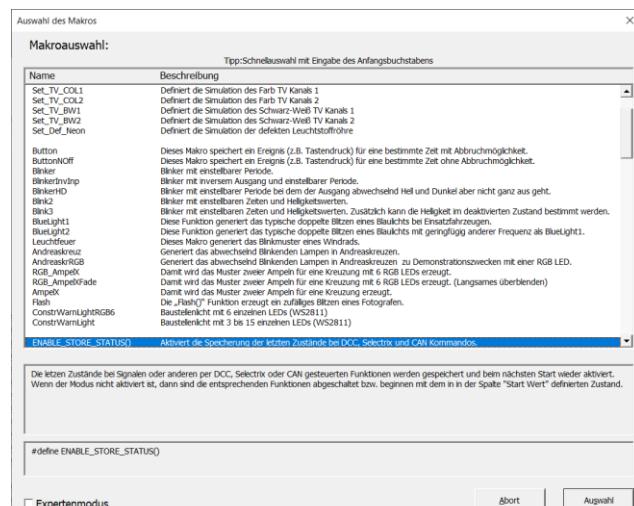
6 Zustände dauerhaft Speichern

Ein weiteres wichtiges Feature der neuen Version ist die dauerhafte Speicherung der Zustände. Diese Funktionalität hat Jürgen für die MobaLedLib implementiert.

Effekte welche per Taster An- und Abschaltbar sind, haben bisher ihren Zustand vergessen, wenn der Strom abgeschaltet wurde. Das ist nicht immer erwünscht. Mit der Erweiterung von Jürgen kann der letzte Zustand jeder schaltbaren Funktion gespeichert werden. Das ist ganz besonders wichtig bei Weichen oder Signalen.

Die Speicherung der Zustände wird über den Befehl „ENABLE_STORE_STATUS“ aktiviert.

Damit merkt sich der Arduino nicht nur die Zustände der per Taster aktivierte Funktionen, sondern genau so die per DCC (Selectrix oder CAN) aktivierten Funktionen. Bei Selectrix werden die als „AnAus“ benutzten Kanäle allerdings überschrieben sobald die Zentrale Daten sendet, weil hier alle Kommandos permanent gesendet werden.



6.1 Speicherung für bestimmte Funktionen Ein- / Ausschalten

Wenn der Befehl „ENABLE_STORE_STATUS“ aktiviert ist, dann müssen nicht alle die Zustände gespeichert werden. Bei bestimmten Funktionen ist es nicht erwünscht, wenn sie nach einem Stromausfall sofort wieder aktiviert werden. Ein Beispiel dafür sind die Sound Funktionen. Diese Funktionen speichern ihren Zustand nicht.

Bei Funktionen welche sich nach einer bestimmten Zeit selber deaktivieren (Timeout) ist die Speicherung ebenso abgeschaltet. Hier kann die Speicherung aber aktiviert werden. Dazu trägt man in die Spalte „Startwert“ ein „*“ ein.

Die meisten anderen Funktionen speichern ihren letzten Zustand automatisch. Ist das bei bestimmten Zeilen nicht gewünscht ist, dann trägt man bei „Startwert“ eine „0“ ein, wenn das Makro nach dem Einschalten deaktiviert sein soll. Wenn es aber immer nach dem Einschalten der Versorgungsspannung aktiviert sein soll, dann wird hier eine „1“ eingetragen.

7 Verbessertes Benutzerinterface der „Zum Arduino schicken“ Funktionalität

In der neuen Version der MobaLedLib wurde das Benutzerinterface welches beim Drücken des „Z. Arduino schicken“ Knopfes deutlich verbessert. Das ist durch ein geniales Programmpaket von Jürgen möglich geworden. Vielen Dank.



Das dazu gehörende Benutzer Interface ist recht komplex geworden, weil sehr viele Verzweigungen im Ablauf möglich sind. Dabei soll der Benutzer so geleitet werden, dass er nichts falsch machen kann. In dem Dokument „Zum Arduino schicken Varianten.pdf“ findet man einen Überblick der verschiedenen Pfade.

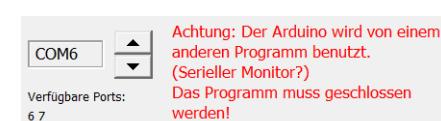
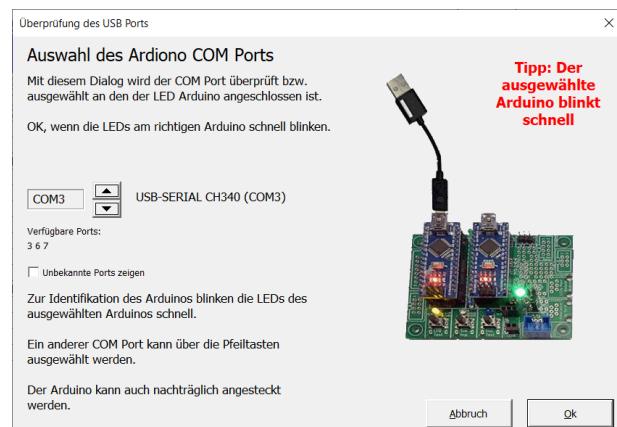
7.1 Automatische Erkennung des angeschlossenen Arduinos

In der normalen Arduino IDE kann zwei angeschlossene Arduinos nicht unterscheiden. Zur Identifikation eines Arduinos muss man diesen abstecken und kontrollieren welcher Port verschwindet. In der alten Version der MobaLedLib wurde dieser Prozess so weit automatisiert, dass man sich nicht merken musste welcher COM Port verändert wurde. Allerdings musste man den Arduino Ab- und wieder Anstecken.

Das ist jetzt nicht mehr nötig. Das Programm erkennt alle angeschlossenen Arduinos und lässt die LEDs des zuletzt benutzten Arduinos schnell blitzen. Wenn nicht der gewünschte Arduino blitzt, dann kann über einen Dialog komfortabel ein anderer Port ausgewählt werden.

Ein nachträglich angeschlossener Arduino wird automatisch erkannt und aktiviert.

Der Dialog zeigt außerdem die Kennung des Arduinos an. Leider wird bei den billigen Nachbauten aus China nur der Chip Typ „USB-SERIAL CH340“ angezeigt. Das Programm erkennt auch wenn der ausgewählte COM Port bereits von einem anderen Programm belegt ist. Das kann z.B. der serielle Monitor oder der serielle



Plotter der Arduino IDE sein. Es könnte aber auch das Farbtest Programm oder ein anderes Tool sein. Das entsprechende Programm muss geschlossen werden bevor das Programm zum Arduino geschickt wird. Das passiert häufig, das wenn man zur Diagnose die Ausgaben des Arduinos in der Konsole betrachten will. Dieser Fehler wurde früher erst nach dem komplizieren entdeckt was dazu geführt hat, dass die langwierige Programmgenerierung noch mal gestartet werden musste.

Der Port kann aber auch belegt sein, wenn das Hochladen zum Arduino fehlgeschlagen ist. Dann läuft versteckt im Hintergrund der Prozess noch einige Zeit weiter, ohne dass man erkennen konnte wann dieser sich beendet hat. Mit dem neuen Dialog wird das automatisch erkannt und angezeigt.

Mit der Checkbox „Unbekannte Ports zeigen“ können auch Komponenten angezeigt werden welche das Programm noch nicht als Arduino erkennt. Damit können auch unbekannte Arduino Nachbauten verwendet werden.

7.2 Automatische Erkennung des Bootloaders

Ein ganz großes Ärgernis sind die zwei verschiedenen Bootloader Typen welche es seit April 2019 gibt (<https://www.heise.de/make/artikel/Arduino-Nano-mit-neuem-Bootloader-4011641.html>).

Dummerweise sieht man es einem Arduino nicht an welcher Bootloader verwendet wird. Wenn man den falschen Bootloader verwendet, dann dauert es ewig bis eine kryptische Fehlermeldung erscheint. Der „Old Bootloader“ ist aber auch nach einem Jahr noch der Bootloader welcher üblicherweise in einem neuen Arduino benutzt wird.

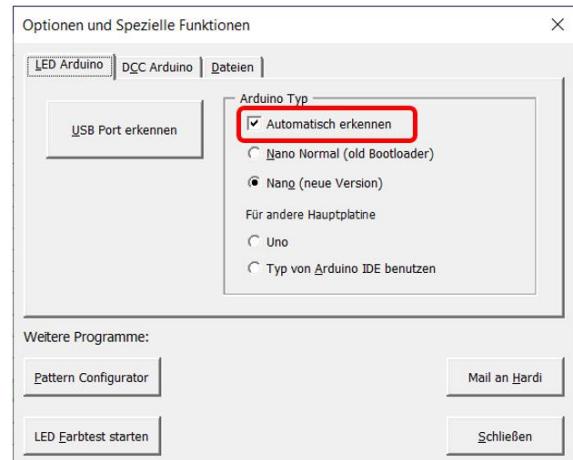
Mit der neuen MobaLedLib wird der Bootloader automatisch erkannt. Im Optionsdialog ist dieses Feature standardmäßig aktiviert.

Die im Bild rechts gezeigten Arduino Typen werden bei der Erkennung als Ausgangswert verwendet. Dadurch wird der Arduino schneller erkannt. Der detektierte Typ wird Automatik Modus anschließend gespeichert damit die nächste Erkennung noch schneller geht.

Die Automatische Erkennung ist ganz besonders wichtig, weil die angezeigten Fehlermeldungen keinen Rückschluss auf das Problem erlauben. Selbst erfahrene Benutzer wissen erst einmal nicht was die Meldungen bedeuten.

Die beiden unterschiedlichen Bootloader haben in der alten Version der Bibliothek zu vielen Anfragen der Benutzer im Forum und per Mail geführt.

Sicherlich nicht nur in der MobaLedLib, sondern auch allgemein bei den Arduino Nutzern. Komisch, dass es in der Arduino IDE keine solche Erkennung gibt.



```
C:\WINDOWS\system32\cmd.exe
Konfiguration wird geladen...
Pakete werden initialisiert...
Boards werden vorbereitet...
Überprüfungsvergang...
Der Sketch verwendet 19390 Bytes (63%) des Programmspeicherplatzes. Das Maximum sind 38722 Bytes.
Globale Variablen verwenden 522 Bytes (25%) des dynamischen Speichers, 1526 Bytes für lokale Variablen verbleiben. Das Maximum sind 2048 Bytes.
Hochladen...
Beim Hochladen des Sketches ist ein Fehler aufgetreten
avrduude: stk500_getsync() attempt 1 of 10: not in sync: resp=0x48
avrduude: stk500_getsync() attempt 2 of 10: not in sync: resp=0x3e
avrduude: stk500_getsync() attempt 3 of 10: not in sync: resp=0xca
avrduude: stk500_getsync() attempt 4 of 10: not in sync: resp=0x48
avrduude: stk500_getsync() attempt 5 of 10: not in sync: resp=0x4a
avrduude: stk500_getsync() attempt 6 of 10: not in sync: resp=0x4a
avrduude: stk500_getsync() attempt 7 of 10: not in sync: resp=0x0e
avrduude: stk500_getsync() attempt 8 of 10: not in sync: resp=0x4a
avrduude: stk500_getsync() attempt 9 of 10: not in sync: resp=0x4a
avrduude: stk500_getsync() attempt 10 of 10: not in sync: resp=0xc4
*****
Da ist was schief gegangen ;-( *****
Drücken Sie eine beliebige Taste . . .
```

7.3 Erkennung wenn der COM Port belegt ist

Ein weiteres Problem hat immer wieder zu viel Frust geführt. Eine serielle Schnittstelle kann immer nur exklusiv von einem einzigen Programm benutzt werden. Wenn man sich während der Konfiguration die Meldungen des Arduinos über den seriellen Monitor von Arduino angezeigt hat,

dann konnten die Daten neuen Programmdaten nicht zum Arduino übertragen werden. Das hat man aber erst erkannt nachdem langwierigen generieren des Programms erkannt.

Jetzt wird das noch vor der Kompilierung erkannt und gemeldet.

7.4 Verbesserung am Arduino DCC Programm

Während den Tests des Programmes ist ein weiterer Fehler im Programm des DCC Arduinos erkannt und behoben worden der dazu geführt hat, dass das Programm des LED Arduinos nicht hochgeladen werden konnte, ohne dass der DCC Arduino ausgesteckt wurde. Nach dem Start ohne DCC Daten und ohne richtiges Programm auf dem LED Arduino wurde die serielle Schnittstelle des DCC Arduinos nicht deaktiviert. Da die beiden Arduinos über diese Schnittstelle miteinander kommunizieren war das Hochladen des Programmes gestört.