

MobaLedLib: Ein kurzer Überblick

Inhaltsverzeichnis

1	Einleitung	6
1.1	Verwendung eines Konfigurationsarrays	7
2	Konfigurationsmakros	8
2.1	Allgemein verwendete Parameter	8
2.1.1	LED.....	8
2.1.2	Cx.....	9
2.1.3	InCh	9
2.1.4	Val0.....	9
2.1.5	Val1.....	9
2.1.6	TimeOut / Duration.....	9
2.2	Variablen / Eingangskanäle	9
2.2.1	SI_Enable_Sound.....	10
2.2.2	SI_LocalVar	10
2.2.3	SI_0.....	10
2.2.4	SI_1.....	10
2.3	Einzelne und mehrere Lichter	10
2.3.1	Const(LED,Cx,InCh,Val0, Val1).....	10
2.3.2	House(LED,InCh, On_Min,On_Limit, ...)	10
2.3.3	HouseT(LED, InCh, On_Min, On_Limit, Min_T, Max_T, ...)	11
2.3.4	GasLights(LED,InCh, ...).....	11
2.3.5	Set_ColTab(Red0,Green0,Blue0, ... Red14,Green14,Blue14).....	11
2.4	Sequenzielle Abläufe	12
2.4.1	Button(LED,Cx,InCh,Duration,Val0, Val1).....	12
2.4.2	RGB_Heartbeat(LED).....	12
2.4.3	Blinker(LED,Cx,InCh,Period)	12
2.4.4	BlinkerInvInp(LED,Cx,InCh,Period)	12
2.4.5	BlinkerHD(LED,Cx,InCh,Period)	13
2.4.6	Blink2(LED,Cx,InCh,Pause,Act,Val0,Val1)	13
2.4.7	Blink3(LED,Cx,InCh,Pause,Act,Val0,Val1,Off)	13
2.4.8	BlueLight1(LED,Cx,InCh)	13
2.4.9	BlueLight2(LED,Cx,InCh)	13
2.4.10	Leuchtf Feuer(LED,Cx,InCh).....	13
2.4.11	Leuchtf FeuerALL(LED,InCh).....	13

2.4.12	Andreaskreuz(LED,Cx,InCh).....	13
2.4.13	AndreaskrRGB(LED,InCh).....	13
2.4.14	RGB_AmpelX(LED,InCh).....	13
2.4.15	RGB_AmpelXFade(LED,InCh).....	14
2.4.16	AmpelX(LED,InCh)	14
2.4.17	AmpelXFade(LED,InCh)	14
2.5	Zufällige Effekte	14
2.5.1	Flash(LED, Cx, InCh, Var, MinTime, MaxTime)	14
2.5.2	Fire(LED,InCh, LedCnt, Brightnes)	14
2.5.3	Welding(LED, InCh).....	14
2.5.4	RandWelding(LED, InCh, Var, MinTime, MaxTime, MinOn, MaxOn)	15
2.6	Sound	15
2.6.1	Sound_Seq1(LED, InCh) ... Sound_Seq14(LED, InCh)	16
2.6.2	Sound_PlayRandom(LED, InCh, MaxSoundNr)	17
2.6.3	Sound_Next_of_N(LED, InCh, MaxSoundNr)	17
2.6.4	Sound_Next_of_N_Reset(LED, InCh, InReset, MaxSoundNr)	17
2.6.5	Sound_Next(LED, InCh)	18
2.6.6	Sound_Prev(LED, InCh).....	18
2.6.7	Sound_PausePlay(LED, InCh).....	18
2.6.8	Sound_Loop(LED, InCh).....	18
2.6.9	Sound_USDSPI(LED, InCh)	18
2.6.10	Sound_PlayMode(LED, InCh).....	18
2.6.11	Sound_DecVol(LED, InCh, Steps).....	18
2.6.12	Sound_IncVol(LED, InCh, Steps)	18
2.6.13	JQ6500 Sound Modul	18
2.7	Steuer Befehle	24
2.7.1	ButtonFunc(DstVar, InCh, Duration)	24
2.7.2	Schedule(DstVar1, DstVarN, EnableCh, Start, End).....	25
2.7.3	Logic(DstVar, ...)	26
2.7.4	Counter(Mode, InCh, Enable, TimeOut, ...).....	26
2.7.5	MonoFlop(DstVar, InCh, Duration)	27
2.7.6	MonoFlopLongReset(DstVar, InCh, Duration).....	27
2.7.7	RS_FlipFlop(DstVar, S_InCh, R_InCh)	27
2.7.8	RS_FlipFlopTimeout(DstVar, S_InCh, R_InCh, Timeout)	27
2.7.9	T_FlipFlopReset(DstVar, T_InCh, R_InCh)	27
2.7.10	T_FlipFlopResetTimeout(DstVar, T_InCh, R_InCh, Timeout)	27

2.7.11 MonoFlopInv(DstVar, InCh, Duration)	27
2.7.12 MonoFlopInvLongReset(DstVar, InCh, Duration).....	27
2.7.13 RS_FlipFlopInv(DstVar, S_InCh, R_InCh)	27
2.7.14 RS_FlipFlopInvTimeout(DstVar, S_InCh, R_InCh, Timeout)	27
2.7.15 T_FlipFlopInvReset(DstVar, T_InCh, R_InCh)	27
2.7.16 T_FlipFlopInvResetTimeout(DstVar, T_InCh, R_InCh, Timeout)	28
2.7.17 MonoFlop2(DstVar0, DstVar1, InCh, Duration)	28
2.7.18 MonoFlop2LongReset(DstVar0, DstVar1, InCh, Duration).....	28
2.7.19 RS_FlipFlop2(DstVar0, DstVar1, S_InCh, R_InCh)	28
2.7.20 RS_FlipFlop2Timeout(DstVar0, DstVar1, S_InCh, R_InCh, Timeout)	28
2.7.21 T_FlipFlop2Reset(DstVar0, DstVar1, T_InCh, R_InCh)	28
2.7.22 T_FlipFlop2ResetTimeout(DstVar0, DstVar1, T_InCh, R_InCh, Timeout)	28
2.7.23 RandMux(DstVar1, DstVarN, InCh, Mode, MinTime, MaxTime)	28
2.7.24 Random(DstVar, InCh, Mode, MinTime, MaxTime, MinOn, MaxOn)	28
2.7.25 New_Local_Var().....	29
2.7.26 Use_GlobalVar(GlobVarNr)	29
2.7.27 InCh_to_TmpVar(FirstInCh, InCh_Cnt).....	30
2.8 Sonstige Befehle	30
2.8.1 CopyLED(LED, InCh, SrcLED).....	30
2.8.2 PushButton_w_LED_0_2(B_LED, B_LED_Cx, InNr, TmpNr, Rotate, Timeout)	30
3 Makros und Funktionen des Hauptprogramms	31
3.1 MobaLedLib	31
3.1.1 MobaLedLib_Configuration()	31
3.1.2 MobaLedLib_Create(leds).....	31
3.1.3 MobaLedLib_Assigne_GlobalVar(GlobalVar).....	33
3.1.4 MobaLedLib_Copy_to_InpStruct(Src, ByteCnt, ChannelNr)	33
3.1.5 MobaLedLib.Update()	33
3.1.6 MobaLedLib.Set_Input(uint8_t channel, uint8_t On).....	33
3.1.7 MobaLedLib.Get_Input(uint8_t channel)	34
3.1.8 MobaLedLib.Print_Config()	34
3.2 Herzschlag des Programms	34
3.2.1 LED_Heartbeat_C(uint8_t PinNr).....	34
3.2.2 Update()	34
4 Viele Schalter mit wenigen Pins	35
4.1 Konfigurierbarkeit	35
4.2 Zwei Schaltergruppen	35

4.3	Prinzip	36
4.4	Integration in das Programm	36
4.5	Frei verfügbare Platine	37
4.6	Zusätzliche Bibliotheken	38
4.7	Einschränkungen	38
4.8	Einfache Möglichkeit zum Einlesen von Druckknopf Aktionen	39
5	Steuerung über eine Zentrale	41
5.1	Einlesen von DCC Kommandos	41
5.1.1	Verschiedene DCC Kommandos	41
5.1.2	Vom InCh zu den LEDs	43
5.1.3	Verwirrende Nummern	45
5.1.4	Anpassungen am Programm	48
5.2	CAN Bus	53
5.2.1	CAN Message Filter	53
6	Anschlusskonzept mit Verteilermodulen	54
6.1	Zusätzliche Stromspeisungen	56
7	Details zur Pattern Funktion	58
7.1	Pattern_Configurator	58
7.1.1	Einfache Beispiele	58
7.1.2	Verwendung im Arduino Programm	60
7.1.3	Einzelne LED Kanäle:	61
7.1.4	Eigenes Makro	62
7.1.5	Beispiel: Ampel	63
7.1.6	Grafische Anzeige	65
7.1.7	Maximal und Minimal Helligkeiten	66
7.1.8	Langsames Auf- und Abblenden der LEDs	67
7.1.9	Ausgeschaltete Blinkende Ampel	69
7.1.10	Beispiel: Baustellen Absicherung	71
7.1.11	Der Goto Mode	72
7.1.12	Weitere Eigenschaften der GotoZeile	74
7.1.13	Status Button Beispiel	76
7.1.14	Zusammenfassung der Goto Eigenschaften	76
7.1.15	Ansteuern von Servos	76
7.1.16	Neues Blatt anlegen	77
7.1.17	Menü des Pattern_Configurators	77
7.2	Parameter der Pattern Funktion	78

7.2.1	Umwandlung der Daten in einzelne Bytes.....	78
7.2.2	Der „normale“ Pattern Befehl.....	78
7.2.3	Weitere Pattern Befehle	83
7.3	HSV Mode der Pattern Funktion	86
8	Fehlersuche	89
9	Manuele Tests	90
10	Konstanten	91
10.1	Konstanten für die Kanalnummer Cx.	91
10.2	Konstanten für Zeiten („Timeout“, „Duration“):	91
10.3	Konstanten der Pattern Funktion	91
10.4	Flags und Modes für Random() und RandMux()	92
10.5	Flags und Modes für die Counter() Funktion	92
10.6	Beleuchtungstypen der Zimmer:	93
11	Offene Punkte.....	95

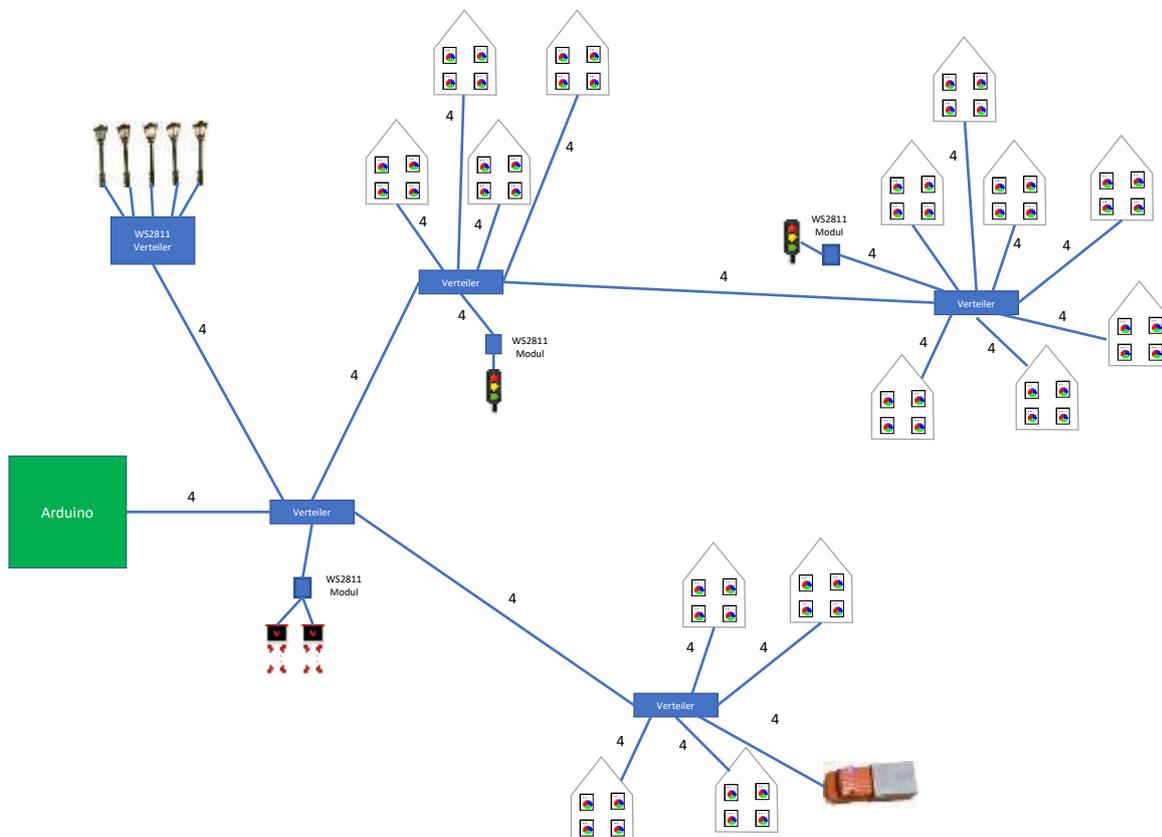
1 Einleitung

Dieses Dokument beschreibt die MobaLedLib für den Arduino. Mit der Bibliothek können bis zu **768 LEDs** und andere Verbraucher über eine einzige Signalleitung von einem Arduino aus gesteuert werden.

Die Bibliothek ist für die Verwendung auf einer **Modelleisenbahn** gedacht. Sicherlich lassen sich auch einige Funktionen der Bibliothek in anderen Anwendungen nutzen.

Zur Ansteuerung der LEDs und anderer Verbraucher werden Bausteine auf Basis des **WS2811 / WS2812** eingesetzt. Da diese ICs nur wenige Cents kosten (7 – 12 Cent) lassen sich damit sehr billige und gleichzeitig sehr flexible Beleuchtungen auf einer Eisenbahn realisieren.

Durch die Ansteuerung mit einer **einzigsten Signalleitung** ist auch die Verkabelung extrem einfach. Die Verbraucher werden über **4-polige Flachkabel** in Verteilerleisten gesteckt, die beliebig kaskadierbar sind.



So können z.B. alle Zimmer in einem Modellhaus mit einer eigenen RGB LED bestückt werden und trotzdem wird das gesamte Haus nur mit einem 4-poligen Stecker am Verteiler angesteckt. Dabei kann jedes Zimmer individuell ein- und ausgeschaltet werden. Zusätzlich können die Helligkeit und die Farbe von jedem Raum verstellt werden. Damit sind dann auch Effekte wie ein Fernseher oder ein Kaminfeuer möglich.

Neben den Häusern gibt es noch eine Vielfalt weiterer Beleuchtungen auf einer Modellanlage, die mit dieser Bibliothek gesteuert werden können. Das sind z.B. Lichtsignale, Andreaskreuze, Verkehrsampeln, Blinkende Einsatzfahrzeuge, Baustellenabsicherungen, Disko oder Kirmes Effekte, ...

Die „Ein Draht Verkabelung“ kann zusätzlich zur Ansteuerung von mehreren **Sound** Modulen auf der ganzen Anlage eingesetzt werden. Entsprechende Soundmodule mit einer passenden SD-Karte gibt

es für zwei Euro. Damit können dann Bahnofsansagen, Eisenbahngeräusche (Glocke an Bahnschranke), Tierlaute, Kirchturmglöcken und vieles mehr wiedergegeben werden.

Das Verfahren eignet sich auch zur Ansteuerung von **bewegten Komponenten**. Mit Hilfe einer kleinen zusätzlichen Schaltung können die Signale zum Ansteuern von **Servos** oder **Schrittmotoren** generiert werden.

Mit einem Transistor zur Verstärkung können Elektromagnete oder Gleichstrommotoren ebenso verwendet werden.

Die Effekte können **automatisch** oder **manuell** gesteuert werden. Die Steuerung kann ganz unabhängig von einem Computer betrieben werden oder seine Befehle von diesem erhalten.

Die Bibliothek enthält ein Modul mit dem **80 und mehr Schalter** über wenige Anschlüsse des Arduinos eingelesen werden können.

Sie unterstützt ebenso das Einlesen von Kommandos über das **DCC Protokoll** oder den **CAN Bus**.

Zum einfachen Einstieg enthält die Bibliothek sehr **viele Beispiele**, die anschaulich zeigen, wie die einzelnen Funktionen genutzt werden. Damit kann das System auch **ohne Programmierkenntnisse** eingesetzt und an die eigenen Bedürfnisse angepasst werden.

1.1 Verwendung eines Konfigurationsarrays

Die Bibliothek wird über ein Array an die individuellen Gegebenheiten angepasst. Dazu werden verschiedene Befehle verwendet, mit denen beschrieben wird, was das Programm machen soll. Mit dem „House()“ Schlüsselwort wird zum Beispiel definiert, wie viele Zimmer ein Gebäude hat und wie viele davon durchschnittlich beleuchtet sein sollen. Außerdem wird die Art der Beleuchtung (Helligkeit, Lichtfarbe, Lampentyp, ...) und weitere Effekte wie ein Fernsehgerät konfiguriert.

Intern werden diese Angaben in einem Byte Array gespeichert. Diese Methode wurde gewählt, damit es später möglich ist, so eine Konfiguration über ein Grafisches Benutzerinterface zu erstellen. Allerdings ist die Eingabe über einen Texteditor so einfach und vor allem sehr flexibel, dass ein GUI eigentlich nicht nötig ist.

Die Verwendung eines Konfigurationsarrays benötigt außerdem wenig Speicherplatz (FLASH) und kann schnell abgearbeitet werden. Die Konfigurationsbefehle sorgen außerdem dafür, dass der benötigte RAM schon beim Kompilieren des Programms bereitgestellt wird. Dadurch ist der Speicherverbrauch schon zu Programmbeginn festgelegt und wird vom Compiler überwacht. Ein aufwändiges dynamisches Speichermanagement entfällt. Auf einem Mikrokontroller, wie dem Arduino, ist nur sehr wenig Speicher vorhanden. Darum muss das Programm sehr sparsam damit umgehen.

Um diese internen Details muss sich der Benutzer der Bibliothek aber nicht kümmern.

2 Konfigurationsmakros

In diesem Abschnitt werden die Konfigurationsmakros nur kurz beschrieben. Auf eine ausführliche Dokumentation wird verzichtet, weil es ja doch keiner liest...

Falls Bedarf nach weiterführender Dokumentation besteht, dann schreibt an MobaLedLib@gmx.de.

Anregungen, Fehlerkorrekturen, ... sind natürlich auch gerne gesehen.

Zur Konfiguration werden C++ Makros eingesetzt. Damit kann eine gewisse Überprüfung der Eingaben gemacht werden ohne dass dazu Programmspeicher benötigt wird.

Diese Makros bestehen aus einem Namen auf welche mehrere Parameter folgen. Die Parameter werden in runde Klammern gesetzt.

Achtung: Nach dem Konfigurationsmakro darf, anders als sonst bei C++, kein Semikolon kommen. Das liegt daran, dass die Makros „nur“ ein Array aus Bytes erstellen, die durch Kommata getrennt sind. Ein Strichpunkt darf hier nicht vorkommen.

Die Makros generieren **konstante** Datenbytes, die in dem Konfigurationsarray gespeichert werden. Aus diesem Array liest das Programm die zu generierenden Lichteffekte aus. Da dieses Array im FLASH des Arduinos abgelegt ist, müssen alle Daten **konstant** sein. Darum ist es nicht möglich, den Makros Variablen zuzuweisen. Berechnungen mit Konstanten sind dagegen erlaubt.

Im Folgenden wird abwechselnd die Bezeichnung „Makro“, „Funktion“ oder „Befehl“ benutzt. Das dient zur Auflockerung des trockenen Dokuments. Tatsächlich handelt es sich um C++ Makros, die per „#define“ angelegt wurden.

2.1 Allgemein verwendete Parameter

Zunächst sollen die in den folgenden Makros benutzten Parameter beschrieben werden, damit diese nicht in jedem Befehl erklärt werden müssen.

2.1.1 LED

Enthält die Nummer der LED in dem Strang. Alle LEDs sind so hintereinandergeschaltet, dass der Ausgang der ersten LED mit dem Eingang der nächsten LED verbunden ist. Diese Methode wird zur Adressierung der einzelnen Leuchtdioden benutzt.

Intern verwendet jede LED die ersten drei Helligkeitswerte welche es über die Signalleitung empfängt zur Ansteuerung der drei Farben Rot, Grün und Blau. Alle Folgenden Werte werden wieder am Ausgang ausgegeben. Damit „sieht“ die zweite LED in der Reihe nur die Daten ab dem zweiten Datensatz. Davon nutzt diese wiederum die ersten drei Helligkeitswerte für sich und gibt die Folgenden an den Nächsten weiter. So können die Farben jeder LED individuell gesteuert werden.

Die WS2811 Chips sind nicht in die RGB LEDs integriert wie die nachfolgende Generation der WS2812 Bausteinen. Damit eignen sich die WS2811 ICs zur Ansteuerung einzelner LEDs wie sie z.B. in einer Straßenlaterne benutzt werden. Ein IC dann kann mit seinen drei Ausgängen z.B. 3 Laternen ansteuern. Aus der Sicht des Programms haben diese drei Straßenlaternen aber eine LED Nummer. Die einzelnen Laternen werden über die Kanalnummer (Cx) angesprochen welche im nächsten Abschnitt beschrieben ist.

Mit den WS2811 Modulen können auch Servomotoren, Sound Module oder ganz andere Aktuatoren angesteuert werden. Trotzdem wird in der folgenden Dokumentation immer der Begriff „LED“ für die Nummer im Strang verwendet.

2.1.2 Cx

Der Parameter Cx beschreibt die Kanalnummer einer RGB LED oder eines WS2811 Bausteins. Hier wird eine der folgenden Konstanten eingetragen:

C1, C2, C3, C12, C23, C_ALL, C_RED, C_GREEN, C_BLUE, C_WHITE, C_YELLOW, C_CYAN

2.1.3 InCh

Viele der Effekte können über einen Eingang Ein- und Ausgeschaltet werden. Der Parameter „InCh“ beschreibt die Nummer des Eingangs. Es sind 256 verschiedene Eingänge möglich. So ein Eingangskanal kann z.B. ein Schalter oder eine spezielle Funktion sein. Es ist auch möglich die Eingangskanäle über das DCC Protokoll oder den CAN Bus von einer Modelleisenbahn Steuerung zu empfangen.

2.1.4 Val0

Enthält den Helligkeitswert oder Allgemein das Tastverhältnis des Ausgangs, wenn der Eingang abgeschaltet ist. Der Parameter ist eine Zahl zwischen 0 und 255. Dabei entspricht 0 dem minimal Wert (LED Dunkel) und 255 dem Maximalwert.

2.1.5 Val1

Ist enthält den Wert der verwendet wird, wenn der Eingang eingeschaltet ist. Siehe „Val0“.

2.1.6 TimeOut / Duration

Enthält die Zeit nachdem der Zähler auf Null gesetzt wird oder der Ausgang deaktiviert wird. Die Zeit wird in Millisekunden angegeben und kann mit angehängtem „Sec“ oder „Min“ ergänzt werden. Die Maximalzeit beträgt 17 Minuten.

2.2 Variablen / Eingangskanäle

Die meisten Makros besitzen einen Eingang „InCh“ mit dem das Makro aktiviert bzw. deaktiviert werden kann. Der Parameter „InCh“ enthält die Nummer des gewünschten Eingangskanals. Diese Nummer verweist auf eine der 256 Variablen. Diese Variablen können im Hauptprogramm mit dem Befehl „Set_Input()“ gesetzt werden. In dem Beispiel „Switched_Houses“ wird gezeigt wie das gemacht werden kann:

```
MobaLedLib.Set_Input(INCH_HOUSE_A, digitalRead(SWITCH0_PIN));
```

Es wird empfohlen, dass anstelle der Nummer ein Symbolischer Name verwendet wird. Dieser kann zu Beginn des Programms mit dem „#define“ Befehl definiert werden.

```
#define INCH_HOUSE_A 0
```

Wenn alle Definitionen an einer Stelle stehen können Überlappungen verhindert werden.

Die Eingangsvariablen können aber auch von anderen Makros gesetzt werden. Im Abschnitt „2.7 Steuer Befehle“ auf Seite 24 werden die Befehle beschrieben mit denen das gemacht werden kann.

Die Variablen können entweder 0 oder 1 sein. Die Bibliothek speichert zusätzlich den letzten Zustand. Daraus kann dann abgeleitet werden ob sich die Variable verändert hat. Viele der Aktionen in der Bibliothek werden nur dann durchgeführt, wenn sich der entsprechende Eingang ändert. Damit kann sehr viel Rechenzeit gespart werden.

Die Variablen ab der Nummer 240 sind reserviert für besondere Funktionen. Bis jetzt sind die Folgenden besonderen Eingangsvariablen definiert (SI = Special Input):

2.2.1 SI_Enable_Sound

Mit dieser Eingangsvariable können die Sound Ausgaben global Ein- und Ausgeschaltet werden. Sie wird zu Programmbeginn auf 1 initialisiert, kann aber vom Programm verändert werden.

2.2.2 SI_LocalVar

Diese Variable wird in der Pattern Funktion verwendet, wenn die der Startwert aus einer Lokalen oder Globalen Variable gelesen werden soll. Die zu verwendende Variable muss zuvor mit einem der Befehle „New_Local_Var()“, „Use_GlobalVar()“ oder „InCh_to_TmpVar()“ deklariert werden.

2.2.3 SI_0

Diese Spezielle Eingangsvariable ist immer 0. Diese Variable braucht man z.B. dann, wenn der „Reset“ Eingang der Zähler Funktion nicht benutzt wird.

2.2.4 SI_1

Wenn eine Funktion immer aktiv sein soll, dann kann diese Variable benutzt werden. Sie ist immer 1.

2.3 Einzelne und mehrere Lichter

In diesem Abschnitt werden die Makros erläutert mit denen einzelne oder mehrere LEDs gesteuert werden können. Die LEDs werden über das Programm oder von außen Ein- und Ausgeschaltet.

2.3.1 Const(LED,Cx,InCh,Val0, Val1)

LED welche, gesteuert von „InCh“, dauerhaft An oder Aus ist.

2.3.2 House(LED,InCh, On_Min,On_Limit, ...)

Das ist vermutlich die am häufigsten genutzte Funktion auf einer Modelleisenbahn. Mit Ihr wird ein „belebtes“ Haus nachgebildet. In diesem Haus sind zufällig nur einige der Räume beleuchtet. Die Farbe und die Helligkeit der Beleuchtungen können individuell vorgegeben werden. Es lassen sich auch bestimmte Effekte wie Fernseher flackern oder ein offener Kamin für einzelne Räume konfigurieren. Außerdem kann das Einschaltverhalten angepasst werden (Neonröhrenflackern oder langsam heller werdende Gaslampen).

Der Parameter „On_Min“ beschreibt wie viele Räume mindestens beleuchtet sein sollen. Nach dem Einschalten werden nach einer zufälligen Zeit so lange Lichter eingeschaltet bis die vorgegebene Anzahl erreicht ist. Dabei werden auch die Zimmer zufällig bestimmt.

Der Parameter „On_Limit“ bestimmt wie viele Räume gleichzeitig benutzt sein sollen. Wenn entsprechend viele LEDs an sind wird zum nächsten, zufällig gewählten, Zeitpunkt eine Lampe ausgeschaltet. Wenn dieser Parameter größer als die Anzahl der Räume ist, dann sind nach einiger Zeit alle Lichter an (Das entspricht unserem Zuhause).

Wenn die Häuser über einen manuell betätigten Schalter Ein- und Ausgeschaltet werde, dann soll der Benutzer ein direktes Feedback beim betätigen des Schalters erkennen. Darum wird sofort beim Einschalten des Eingangs (InCh) eine Beleuchtung aktiviert und entsprechend Eine deaktiviert, wenn der Schalter Ausgeschaltet wird.

Die drei Punkte „...“ in der Makrodefinition repräsentieren die Position an der die Liste der Raumbeleuchtungen eingetragen wird. Es können bis zu 2000 Räume angegeben werden (Schloss). Die Beleuchtung der Zimmer wird mit den folgenden Konstanten festgelegt:

Farben/Helligkeit:

ROOM_DARK, ROOM_BRIGHT, ROOM_WARM_W, ROOM_RED, ROOM_D_RED, ROOM_COL0,
ROOM_COL1, ROOM_COL2, ROOM_COL3, ROOM_COL4, ROOM_COL5, ROOM_COL345

Animierte Effekte:

FIRE, FIRED, FIREB, ROOM_CHIMNEY, ROOM_CHIMNEYD, ROOM_CHIMNEYB, ROOM_TV0,
ROOM_TV0_CHIMNEY, ROOM_TV0_CHIMNEYD, ROOM_TV0_CHIMNEYB, ROOM_TV1,
ROOM_TV1_CHIMNEY, ROOM_TV1_CHIMNEYD, ROOM_TV1_CHIMNEYB

Besondere Lampen:

GAS_LIGHT, GAS_LIGHT1, GAS_LIGHT2, GAS_LIGHT3, GAS_LIGHTD, GAS_LIGHT1D, GAS_LIGHT2D,
GAS_LIGHT3D, NEON_LIGHT, NEON_LIGHT1, NEON_LIGHT2, NEON_LIGHT3, NEON_LIGHTD,
NEON_LIGHT1D, NEON_LIGHT2D, NEON_LIGHT3D, NEON_LIGHTM, NEON_LIGHT1M,
NEON_LIGHT2M, NEON_LIGHT3M, NEON_LIGHTL, NEON_LIGHT1L, NEON_LIGHT2L, NEON_LIGHT3L

Nicht verwendeter Raum:

SKIP_ROOM

Beispiel: **House(0, SI_1, 2, 3, ROOM_DARK, ROOM_BRIGHT, ROOM_WARM_W)**

2.3.3 HouseT(LED, InCh, On_Min, On_Limit, Min_T, Max_T, ...)

Entspricht dem House() Makro. Hier können zwei zusätzliche Parameter „Min_T“ und „Max_T“ angegeben werden. Diese Zahlen beschreiben in Sekunden wie lange es zufällig dauern soll bis die nächste Änderung eintritt. Bei der „House()“ Funktion werden diese Zeiten über die globalen Definitionen

```
#define HOUSE_MIN_T 50 // Minimal time [s] to the next event (1..255)
#define HOUSE_MAX_T 150 // Maximal random time [s] "
```

vorgegeben.

Die beiden Konstanten können auch im Anwendungsprogramm definiert werden. Dazu müssen die Beiden Zeilen nur vor der „#include „MobaLedBib.h“ Zeile stehen wie das in den beiden „House“ Beispielen gezeigt ist.

2.3.4 GasLights(LED,InCh, ...)

Straßenlaternen sind ein wichtiger Bestandteil einer virtuellen Stadt. Sie beleuchten die nächtlichen Straßen und erzeugen eine warme Atmosphäre insbesondere, wenn es sich um Gaslaternen handelt. Diese Lampen wurden in der realen Welt anfangs von Menschenhand und später von Uhren oder Lichtsensoren gezündet. Das wird hier sehr schön beschrieben:

<http://www.gaswerk-augsburg.de/fernzuendung.html>.

Durch unterschiedliche Uhrzeiten oder Lichtverhältnisse gehen die Laternen nicht gleichzeitig an. Das beobachte ich immer wieder auf meinem Heimweg. Die Lampen gehen zufällig an und werden dann langsam heller bis sie die volle Helligkeit erreichen. Dieses Verhalten haben auch die Lampen welche über das Makro „GasLights()“ gesteuert werden. Hier ist außerdem noch ein zufälliges Flackern implementiert welches durch Schwankungen im Gasdruck oder durch Windböen entstehen kann. Angesteuert werden die Lampen von WS2811 Chips. Bei Glühbirnen sind alle drei Ausgänge parallelgeschaltet, bei LED Lampen steuert ein IC drei Lampen. In der Konfiguration muss die Reihenfolge ..1., ..2., ..3. verwendet werden wie im Beispiel unten gezeigt. Das sorgt dafür, dass das Programm nacheinander die Ausgänge eines WS2811 Chips benutzt und nicht zum nächsten Kanal wechselt. Das angehängte „D“ im Beispiel unten bedeutet „Dark“. Diese Lampen leuchten dunkler als die Anderen.

Beispiel: **GasLights(Gas_Lights1, 67, GAS_LIGHT1D, GAS_LIGHT2D, GAS_LIGHT3D, GAS_LIGHT)**

2.3.5 Set_ColTab(Red0,Green0,Blue0, ... Red14,Green14,Blue14)

Mit dem Makro „Set_ColTab()“ kann man die Farben und Helligkeiten der Lampen individuell anpassen. Dazu wird eine Liste von 15 RGB Werten angegeben. Der Befehl kann mehrfach in der Konfiguration verwendet werden und betrifft alle folgenden „House()“ oder „GasLights()“ Zeilen.

Hier ein Beispiel des Befehls:

```
//      Red Green Blue
Set_ColTab( 1, 0, 0, // 0 ROOM_COL0      Dark red for demonstration
            0, 1, 0, // 1 ROOM_COL1      " green "
            0, 0, 1, // 2 ROOM_COL2      " blue "
            100, 0, 0, // 3 ROOM_COL345    red for demonstration   randomly color
            0, 100, 0, // 4 ROOM_COL345    green "                   3, 4 or 5 is
            0, 0, 100, // 5 ROOM_COL345    blue "                   used
            50, 50, 50, // 6 Gas light
            255, 255, 255, // 7 Gas light
            20, 20, 27, // 8 Neon light
            70, 70, 80, // 9 Neon light
            245, 245, 255, // 10 Neon light
            50, 50, 20, // 11 TV0 and chimney color A randomly color A or B is used
            70, 70, 30, // 12 TV0 and chimney color B
            50, 50, 8, // 13 TV1 and chimney color A
            50, 50, 8) // 14 TV2 and chimney color B
```

2.4 Sequenzielle Abläufe

Dieser Abschnitt beschreibt Funktionen welche Zeitlichen Abläufe abbilden. Sequenzielle Abläufe kommen in der Realität und natürlich auch auf einer Modelleisenbahn häufig vor. Ein Beispiel dafür ist die Verkehrsampel. Hier werden nacheinander die entsprechenden Ampelphasen angezeigt. Mehrere Lampen müssen für diese Darstellung koordiniert geschaltet werden.

In der Bibliothek werden diese Steuerungen von der „Pattern()“ Funktion generiert. Mit dem Excel Programm „Pattern_Configurator.xlsm“ können solche Sequenzen generiert werden. Im Kapitel 5 ab Seite 41 wird das näher beschrieben.

2.4.1 Button(LED,Cx,InCh,Duration,Val0, Val1)

Dieses Makro speichert einen Tastendruck für eine bestimmte Zeit. Damit wird auf unserer Eisenbahn der Rauchgenerator im „Brennenden“ Haus aktiviert. Der Ausgang kann vor dem Ablauf der Zeit deaktiviert werden, wenn die Taste ein zweites Mal gedrückt wird.

Die Duration bestimmt die Dauer für die der Ausgang aktiviert wird, wenn der Taster gedrückt wird. Die Zeit wird in Millisekunden angegeben und kann zwischen 16ms und 17 Minuten (1048560 ms) liegen. Wenn die Zeit in Sekunden oder Minuten angegeben werden soll, dann kann „Sec“ oder „Min“ hinter den Wert geschrieben werden. Dabei sind die Groß- und Kleinschreibung und mindestens ein Leerzeichen zur vorangegangenen Zahl wichtig (Beispiel: 3.5 Min).

Eine Kombination von Minuten, Sekunden und Millisekunden ist auch möglich.

Beispiel: 3 Min + 2 Sec + 17 ms

Beispiel: **Button(10, C_ALL,0, 3.5 Min, 0, 255)**

2.4.2 RGB_Heartbeat(LED)

Mit wechselnden Farben und sanftem Übergang blinkende RGB LED zur Überwachung der Programmfunktion.

2.4.3 Blinker(LED,Cx,InCh,Period)

Implementiert einen Blinker mit einer vorgegebenen Periode. Die Periode wird in Millisekunden angegeben. Auch hier können „Sec“ oder „Min“ angehängt werden wie bei der „Button()“ Funktion. Die Maximalperiode beträgt zwei Minuten (131070 ms).

2.4.4 BlinkerInvInp(LED,Cx,InCh,Period)

Entspricht der „Blinker()“ Funktion. In diesem Fall ist der Blinker dann Aktiv, wenn der Eingangskanal (InCh) ausgeschaltet ist.

2.4.5 BlinkerHD(LED,Cx,InCh,Period)

Eine weitere Variante des Blinkers. Hier wechselt die Helligkeit der LED zwischen „Hell“ und „Dunkel“. Die Funktion wird in dem Beispiel „Macro_Fire_truck“ benutzt.

2.4.6 Blink2(LED,Cx,InCh,Pause,Act,Val0,Val1)

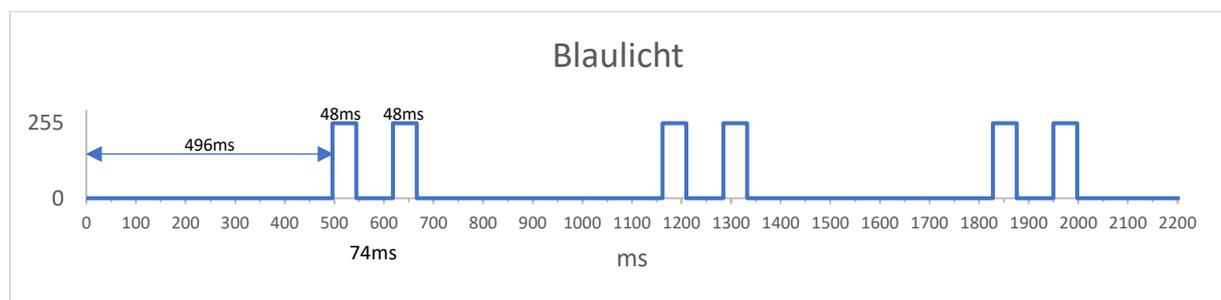
Bei dieser Variante kann die Dauer der beiden Phasen und die Helligkeit in den Phasen angegeben werden. „Pause“ bestimmt die Zeit in der „Val0“ ausgegeben wird und „Act“ die Zeit während dessen „Val1“ verwendet wird.

2.4.7 Blink3(LED,Cx,InCh,Pause,Act,Val0,Val1,Off)

Eine dritte Variante der „Blink()“ Funktion. Hier kann zusätzlich der Helligkeitswert im ausgeschalteten Zustand angegeben werden (Off).

2.4.8 BlueLight1(LED,Cx,InCh)

Diese Funktion generiert das typische doppelte Blitzen eines Blaulichts bei Einsatzfahrzeugen.



2.4.9 BlueLight2(LED,Cx,InCh)

Zweites Blaulicht mit geringfügig anderer Periode. Durch die Verwendung von zwei Blaulichtern mit geringfügig unterschiedlicher Periode entsteht ein realistischerer Effekt.

2.4.10 Leuchtfeuer(LED,Cx,InCh)

Dieses Makro generiert das Blinkmuster eines Windrads. Das Licht ist eine Sekunde lang an, dann eine halbe Sekunde aus und anschließend wieder eine Sekunde lang an. Darauf folgt eine Pause von 1.5 Sekunden. (Siehe https://www.windparkwaldhausen.de/contentbeitrag-170-84-kennzeichnung_befuerung_von_windkraftanlagen_.html)

2.4.11 LeuchtfeuerALL(LED,InCh)

Bei diesem Leuchtfeuer werden alle drei Kanäle benutzt. Das entspricht dem „Leuchtfeuer()“ Befehl mit dem Parameter „C_ALL“.

2.4.12 Andreaskreuz(LED,Cx,InCh)

Zur Ansteuerung der abwechselnd Blinkenden Lampen in Andreaskreuzen kann diese Funktion benutzt werden. „Cx“ bestimmt den ersten verwendeten Kanal. Dieser blinkt im Wechsel zu dem folgenden Kanal. Die Helligkeit der LEDs ändert sich langsam so dass das Typische „Weiche“ Blinken entsteht.

2.4.13 AndreaskrRGB(LED,InCh)

Diese Funktion benutzt zwei RGB LEDs zur Simulation des Andreaskreuzes. Dazu wird jeweils nur die Rote LED angesteuert. Dieses Makro ist nur zu Testzwecken mit einem LED Stripe gedacht.

2.4.14 RGB_AmpelX(LED,InCh)

Damit wird das Muster zweier Ampeln mit jeweils 3 RGB LEDs für eine Kreuzung erzeugt. Dieses Makro ist nur zu Testzwecken mit einem LED Stripe gedacht. Auf der Modelleisenbahn wird man Ampeln mit einzelnen LEDs einsetzen welche dann über ein WS2811 Modul angesteuert werden.

In dem Excel Programm „Pattern_Configurator.xlsm“ welches sich im Verzeichnis der Bibliothek befindet sind einige Seiten welche die Konfiguration der Ampeln beschreiben („AmpelX“ .. „RGB_AmpelX_Fade_Off“).

2.4.15 RGB_AmpelXFade(LED,InCh)

Ein weiteres Beispiel zur Ansteuerung von Ampeln mit RGB LEDs. Hier werden die Lampen langsam übergeblendet was einen realistischeren Eindruck generiert. Auf der Seite „RGB_AmpelX_Fade“ in „Pattern_Configurator.xlsm“ wird schematisch gezeigt wie das Einblenden funktioniert.

2.4.16 AmpelX(LED,InCh)

Diese Ampel ist für den Einsatz auf der Anlage gedacht. Damit werden einzelne LEDs über zwei WS2811 Module angesteuert. Im Beispiel „09.TrafficLight_Pattern_Func.“ Findet man einen „Schaltplan“ dazu. Zur Absicherung einer Kreuzung werden 4 Ampeln benötigt. Dabei zeigen die sich gegenüberstehenden Lichtzeichen immer das gleiche Muster. Zur Ansteuerung der gegenüberliegenden Ampel kann ein weiterer WS2811 Chip verwendet werden welcher das gleiche Eingangssignal wie sein Counterpart bekommt. Im Beispiel ist das schematisch gezeigt.

Mit dem Programm „Pattern_Configurator.xlsm“ können beliebig komplizierte Ampelanlagen mit mehreren Fahrspuren und Fußgängerampeln und konfiguriert werden.

2.4.17 AmpelXFade(LED,InCh)

Das ist die gleiche Funktion wie oben nur dass hier die Lampen langsam Ein- und Ausgeblendet werden.

2.5 Zufällige Effekte

Dieser Abschnitt beschreibt einige zufällige Effekte.

2.5.1 Flash(LED, Cx, InCh, Var, MinTime, MaxTime)

Die „Flash()“ Funktion erzeugt ein zufälliges Blitzen eines Fotografen. Über die Parameter „MinTime“ und „MaxTime“ wird bestimmt wie häufig der Blitz ausgelöst wird. Der Erste Parameter bestimmt wie lange mindestens bis zum nächsten Blitz gewartet werden. Entsprechend beschreibt „MaxTime“ die Maximale Zeit. Zwischen diesen beiden Zeiten bestimmt die Bibliothek einen zufälligen Zeitpunkt.

Das Makro besteht aus zwei anderen Makros. Dem „Const()“ und dem „Random()“ Makro (Siehe Kapitel „2.7.24 **Random**(DstVar, InCh, Mode, MinTime, MaxTime, MinOn, MaxOn)“ auf Seite 28). Dabei wird die „Const()“ Funktion von der „Random()“ Funktion gesteuert. Dazu wird eine Zwischenvariable benötigt deren Nummer als Parameter „Var“ eingetragen wird. Dabei handelt es sich um eine der 256 Eingangsvariablen welche im Kapitel „2.1.3 InCh“ auf Seite 9 beschrieben wurden. Achtung diese Zwischenvariable darf nirgend wo anders benutzt werden.

Beispiel: **Flash(11, C_ALL, SI_1, 200, 5 Sec, 20 Sec)**

2.5.2 Fire(LED,InCh, LedCnt, Brightnes)

Mit der „Fire()“ Funktion können größere Feuer simuliert werden. Dazu werden mehrere RGB LEDs verwendet welche an unterschiedlichen Stellen des „Feuers“ leuchten. Auf unserer Anlage wird die Funktion zur Simulation eines „Brennenden“ Hauses eingesetzt.

2.5.3 Welding(LED, InCh)

Mit der „Welding()“ Funktion kann ein Schweißlicht simuliert werden. Dieses Licht flackert eine gewisse Zeit hell Weiß und verlischt dann für eine Weile. Nach dem Schweißvorgang glüht die „Schweißstelle“ kurz rot nach. Diese Funktion sollte von einer übergeordneten Funktion gesteuert werden („Der Arbeiter will ja auch mal eine Pause“).

2.5.4 RandWelding(LED, InCh, Var, MinTime, MaxTime, MinOn, MaxOn)

Mit dieser Funktion wird das Schweißlicht zufällig gesteuert. Die Zeiten „MinTime“ und „MaxTime“ bestimmen den Zufälligen Startzeitpunkt. Über „MinOn“ und „MaxOn“ wird angegeben wie lange eine die Arbeit an einem Werkstück dauert. Der Parameter „Var“ enthält die Nummer einer Zwischenvariablen welche zur Steuerung der „Welding()“ Funktion benutzt wird. Achtung diese Zwischenvariable darf nirgend wo anders benutzt werden.

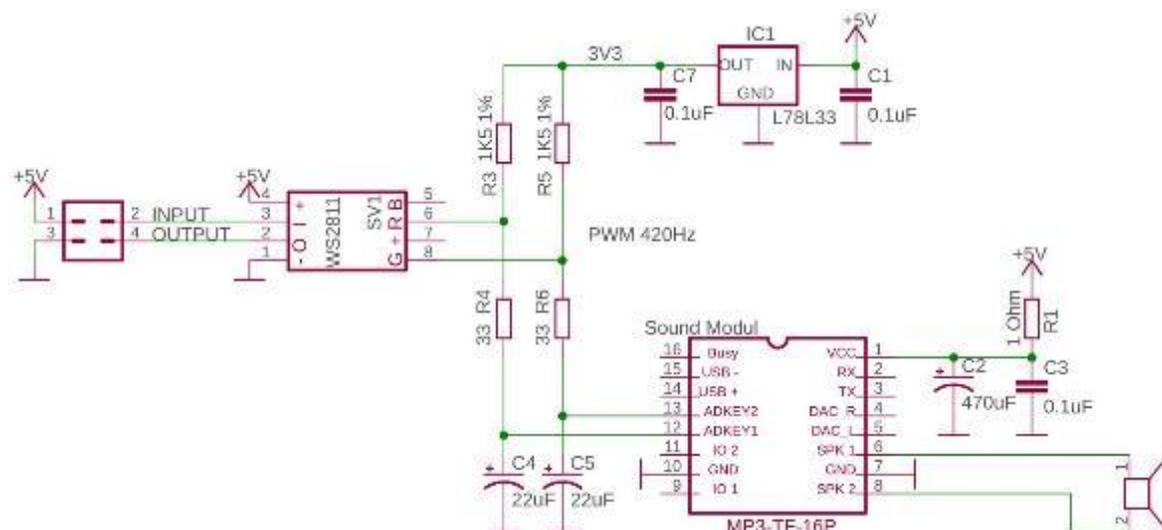
Beispiel: `RandWelding(12, SI_1, 201, 1 Min, 3 Min, 50 Sec, 2 Min)`

2.6 Sound

Die MobaLedLib Bibliothek können zu den Lichteffekten passende Geräusche wiedergegeben werden. So kann zum Beispiel das Leuten der Schranke mit dem Blinken des Andreaskreuzes wiedergegeben werden. Dazu wird ein zusätzliches Soundmodul vom Typ MP3-TF-16P benötigt:

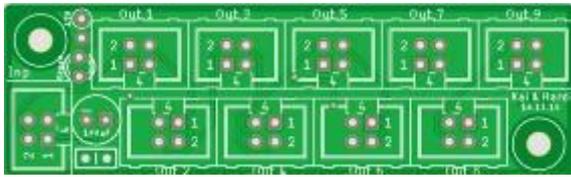


Dieses Modul ist in China für einen Euro erhältlich. Zusätzlich benötigt man eine SD Karte und einen Lautsprecher. Mit diesem Modul können MP3 und WAV Dateien über einen eingebauten 3 Watt Verstärker abgespielt werden. Normalerweise werden die MP3 Dateien über 20 Taster welche über verschiedene Widerstände mit dem Modul verbunden sind abgespielt. Das Sound Modul kann auch über die gleiche Signalleitung wie die Leuchtdioden gesteuert werden indem man die Tastendrücke simuliert. Dazu wird ein WS2811 Chip benutzt. Dieser muss mit ein paar Widerständen und Kondensatoren zur Filterung der Signale beschaltet werden. Das folgende Schaltbild zeigt den Aufbau:



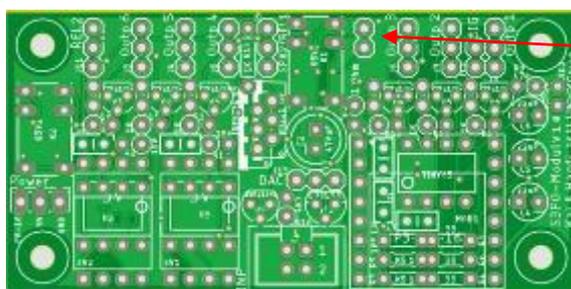
Die Zip Datei S3PO_Modul_WS2811.zip im Verzeichnis „extras“ enthält einen Schaltplan und eine Platine mit für diesen Aufbau. Auf der Schaltung sind einige weitere Bauteile vorgesehen mit denen größere Leistungen über ein WS2811 Modul geschaltet werden können. Zusätzlich kann man mit der Schaltung Servos oder Schrittmotoren ansteuern. Diese werden aber für die Soundwiedergabe nicht benötigt.

Die Platine besteht aus drei Teilen welche durch knicken getrennt werden können. Die beiden oberen Teile (Im Bild unten ist nur einer gezeigt) enthalten jeweils einen 9-fach Verteiler für den Anschluss der LEDs und anderer auf dem WS281x basierender Komponenten:



Diese Verteiler können beliebig an zentralen Positionen unter der Anlage platziert werden. In die mit „Out ...“ bezeichneten Stecker werden die Häuser, Ampeln, Sound Module und andere Verbraucher angesteckt. Hier können auch weitere Verteiler Platinen angeschlossen werden. Der mit „Inp“ bezeichnete Stecker wird mit dem Arduino oder einer vorangegangenen Verteilerplatine über ein Flachkabel verbunden. Achtung: Bei unbenutzten Steckplätzen müssen die Pins 2 und 4 mit einem Jumper überbrückt werden oder der entsprechende Lötjumper auf der Rückseite verbunden werden.

Das nächste Bild zeigt die Platine für das Sound Modul. Es müssen nur die Bauteile des oben gezeigten Schaltplans bestückt werden.



Lautsprecher
Anschluss

Achtung: Auf der Rückseite befinden sich einige Lötverbinder. Für die Funktion des Sound Moduls ohne die anderen Komponenten müssen SJ1 und SJ21 verbunden werden.

Die Bibliothek unterstützt auch das [Sound Modul JQ6500](#). Dieses Modul benötigt keine SD Karte zum Abspeichern der Sound Dateien. Es kann allerdings nur 5 verschiedene Geräusche per direktem Zugriff abspielen. Im Abschnitt „2.6.13 JQ6500 Sound Modul“ wird dieses Sound Modul vorgestellt.

Im Folgenden werden die Befehle zur Soundausgabe kurz vorgestellt.

2.6.1 Sound_Seq1(LED, InCh) ... Sound_Seq14(LED, InCh)

Zur Wiedergabe bestimmter Geräusche gibt es 14 Befehle in der Bibliothek. Mit dem Befehl „Sound_Seq1()“ wird die erste MP3 oder WAV Datei auf der SD Karte wiedergegeben. Entsprechend wird mit „Sound_Seq2()“ die zweite Datei abgespielt...

Laut der Dokumentation des Sound Moduls müssen die MP3 oder WAV Dateien in spezielle Namen haben und in bestimmten Unterverzeichnissen abgelegt werden. Das scheint nur für die Wiedergabe per serieller Schnittstelle zu gelten. Wenn die Dateien über Taster wiedergegeben werden, dann ist das nicht nötig. Das gilt auch für die hier vorgestellte Schaltung welche die Tastendrucke über ein WS2811 Modul simuliert. Der Dateiname spielt keine Rolle. Mit dem „Sound_Seq1()“ Befehl wird immer die Datei abgespielt welche als erstes auf die SD Karte kopiert wurde. Daher auch der Name „...Seq...“. Wenn eine Datei gelöscht wird und später eine neue Datei auf die Karte kopiert wird, dann wird die Neue Datei im Inhaltsverzeichnis auf der Karte an die Stelle der gelöschten Datei

eingetragen. Das kann sehr verwirrend sein. Dummerweise zeigt der Windows Explorer die Dateien auf einer SD-Karte immer sortiert an. Es gibt keine Option mit der man das Sortieren ganz deaktivieren kann. Zur Überprüfung der Reihenfolge muss man eine Kommandozeilenfenster öffnen (cmd.exe) und „dir f:“ eintippen (Der Laufwerksbuchstabe „f:“ muss evtl. an den tatsächlich erkannte SD-Kartenleser angepasst werden):

Bei der SD-Karte im folgenden Bild wird die Datei „001.mp3“ abgespielt, wenn der Eingang des „Sound_Seq1()“ Befehls aktiviert wird. Der 6. Eintrag „Muh.wav“ wird mit dem Makro „Sound_Seq6()“ ausgegeben.

```
C:\Users\Hardi>dir f:
Volume in Laufwerk F: hat keine Bezeichnung.
Volumeseriennummer: A87B-A154

Verzeichnis von F:\

18.03.2018  00:58          51.471 001.mp3
30.01.2018  21:04           5.564 005.wav
03.10.2018  21:29          18.143 007.mp3
25.07.2015  16:04       2.284.950 018.mp3
03.10.2018  21:31       612.667 Big Ben MP3 Klingelton.mp3
18.03.2018  01:01       239.848 Muh.wav
18.03.2018  01:00       465.808 Muhh.wav
03.10.2018  21:29          18.143 S1-b-ch.mp3
03.10.2018  10:29          19.640 S1-b-d.mp3
03.10.2018  10:30          39.168 S1-huehner.mp3
03.10.2018  10:30          19.562 S1-kapelle.mp3
03.10.2018  10:31          26.710 S1-kirche.mp3
03.10.2018  21:28          30.867 S1-schafe.mp3
03.10.2018  21:29          35.091 S2Voegel.mp3
03.10.2018  21:44       3.703.998 voegel-im-wald-mit-bachlauf.mp3
03.10.2018  21:42          48.109 motorrad-starten-mit.mp3
03.10.2018  21:41          368.265 motorrad-starten-leerlauf.mp3
03.10.2018  21:39          218.219 feuerwerk-kurz-mit-heuler.mp3
03.10.2018  21:38          51.034 spatz-sperling-zwitschert-3.mp3
03.10.2018  21:38          23.867 vogel-waldkauz-eule.mp3
03.10.2018  21:36          62.738 vogelstimme-spatz.mp3
                21 Datei(en),      8.343.862 Bytes
                0 Verzeichnis(se), 116.072.448 Bytes frei

C:\Users\Hardi>
```

Die Dateien werden über simulierte „Tastendrucke“ abgespielt welche über verschiedene Widerstände kodiert sind. Unglücklicherweise sind die Abstände der Widerstände im oberen Bereich relativ klein so dass es durch Bauteiltoleranzen oder Temperaturschwankungen bei den Sound Befehlen „Sound_Seq13()“ und „Sound_Seq14()“ zu Überschneidungen kommen kann.

Die Ausgabe von Sounds kann über die Variable „SI_Enable_Sound“ global ein und ausgeschaltet werden. Auf diese Weise kann man mit einem Schalter alle Geräusche deaktivieren.

2.6.2 Sound_PlayRandom(LED, InCh, MaxSoundNr)

Mit diesem Befehl wird eine Zufällige Datei zwischen 1 und „MaxSoundNr“ widergegeben, wenn der Eingang des Befehls aktiviert wird. Das kann z.B. für die Ansage von Bahnhofsdurchsagen verwendet werden.

2.6.3 Sound_Next_of_N(LED, InCh, MaxSoundNr)

Mit diesem Makro wird die nächste Datei zwischen 1 und „MaxSoundNr“ widergegeben, wenn der Eingang des Befehls aktiviert wird. Das kann z.B. für das Abspielen des Stundenläutens von Kirchenglocken benutzt werden.

2.6.4 Sound_Next_of_N_Reset(LED, InCh, InReset, MaxSoundNr)

Dieser Befehl entspricht dem vorangegangenen. Hier besteht zusätzlich die Möglichkeit über einen weiteren Eingang „InReset“ den Zähler zurück zu setzen.

2.6.5 Sound_Next(LED, InCh)

Dieser Befehl nutzt eine interne Funktion des Sound Moduls mit welcher die nächste Sound Datei ausgegeben werden kann. Der Befehl ist nicht beschränkt auf die 14 per „Tasten“ auswählbaren Dateien wie die vorangegangenen Befehle. Auf diese Weise können alle Dateien nacheinander abgespielt werden. Eine gezielte Einschränkung des Bereichs ist aber nicht möglich. Das kann vorab über die Auswahl der Dateien auf der SD-Karte gemacht werden.

2.6.6 Sound_Prev(LED, InCh)

Dieser Befehl entspricht dem „Sound_Next()“ Befehl. Hiermit wird mit jedem Impuls die vorangegangene Datei auf der SD-Karte gespielt.

2.6.7 Sound_PausePlay(LED, InCh)

Damit kann die Wiedergabe angehalten und fortgesetzt werden.

2.6.8 Sound_Loop(LED, InCh)

Dieser Befehl zeugt von der Herkunft als Musikplayer. Damit können nacheinander alle Lieder auf einer SD-Karte wiedergegeben werden. Für die Eisenbahn könnte man die Funktion evtl. mit speziellen Sound Dateien mit langen Pausen nutzen.

2.6.9 Sound_USDSPI(LED, InCh)

Diese Funktion habe ich nicht verstanden. Sie aktiviert die Taste welche laut der „ausführlichen“ Dokumentation des Sound Moduls zwischen „U / SD / SPI“ umschaltet.

2.6.10 Sound_PlayMode(LED, InCh)

Wenn der „Loop“ Modus aktiv ist, dann kann man mit dieser „Taste“ den „Play Mode“ umschalten. Das habe ich auch nicht ganz verstanden. Es scheint folgende Modi zu geben: „Sequence“, „Sequence“, „Repeat same“, „Random“, „Loop off“

Wie und ob sich die beiden „Sequence“ Modes am Anfang unterscheiden weiß ich nicht.

2.6.11 Sound_DecVol(LED, InCh, Steps)

Die Wiedergabelautstärke des Sound Moduls ist nach dem Einschalten auf den Maximalwert gestellt. Das ist dank des eingebauten 3 Watt Verstärkers relativ laut. Mit dem „Sound_DecVol()“ Makro kann die Lautstärke reduziert werden. Der Parameter „Steps“ gibt die Anzahl der Schritte zur Reduzierung der Lautstärke an. Die Zahl kann im Bereich von 1 bis 30 liegen. Zur Veränderung der Lautstärke muss die entsprechende „Taste“ länger als eine Sekunde gehalten werden. Dann wird die Lautstärke alle 150 ms um einen Schritt reduziert. Das entsprechende Timing übernimmt das Makro. Man muss aber bedenken, kein weiterer Befehl an das Sound Modul geschickt werden darf während die Lautstärke verändert wird. Auf eine automatische Verriegelung wurde aus Speicherplatzgründen verzichtet.

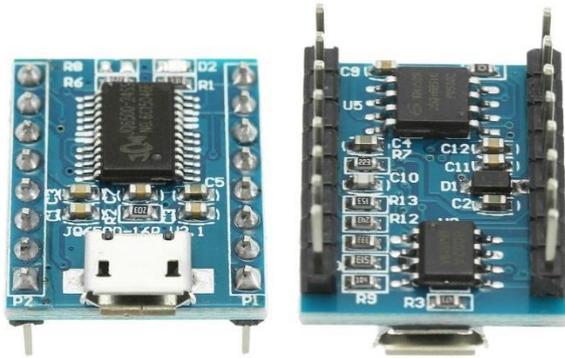
Dummerweise merkt sich das Modul die letzte Einstellung nicht. Das bedeutet, dass die Lautstärke nach jedem Einschalten neu gesetzt werden muss. Evtl. ist es besser wenn man die Sound Dateien mit einem geeigneten Programm „leiser“ rechnet.

2.6.12 Sound_IncVol(LED, InCh, Steps)

Damit kann die Lautstärke wieder erhöht werden.

2.6.13 JQ6500 Sound Modul

Die Bibliothek unterstützt alternativ auch das Sound Modul JQ6500. Dieses Modul benötigt keine SD Karte zum Abspeichern der Sound Dateien. Es kann allerdings nur 5 verschiedene Geräusche per direktem Zugriff abspielen. Weitere Dateien können über die „Next“ und „Prev“ Funktion abgerufen werden. Das Modul kann für sehr kleines Geld in China bestellt werden.



Das JQ6500 Modul ist unter dem Strich etwas günstiger, weil man hier keine zusätzliche SD Karte benötigt. Allerdings bedeutet das auch einen gewissen Zusatzaufwand. Zum Aufspielen der Sounds benötigt man eine Software. Hier kann man das Tool herunterladen (Siehe „English Language MusicDownload.exe“): <https://sparks.gogo.co.nz/jq6500/index.html>

Auf der Web Seite wird auf Risiken hingewiesen. Es hat bisher aber problemlos funktioniert.

Ein Vorteil des Moduls gegenüber dem MP3-TF-16P ist, dass es sich die zuletzt verwendete Lautstärke merkt. Das kann aber auch Probleme bereiten, wenn die Lautstärke versehentlich auf 0 reduziert wurde.

Die Makro Aufrufe entsprechen denen des MP3-TF-16P Moduls. Zur Unterscheidung enthalten sie ein „_JQ6500_“ im Namen. Folgende Befehle sind verfügbar:

```
Sound_JQ6500_Prev(LED, InCh)
Sound_JQ6500_Next(LED, InCh)

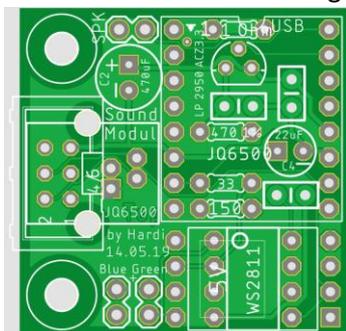
Sound_JQ6500_DecVol(LED, InCh, Steps)
Sound_JQ6500_IncVol(LED, InCh, Steps)

Sound_JQ6500_Seq1(LED, InCh)
Sound_JQ6500_Seq2(LED, InCh)
Sound_JQ6500_Seq3(LED, InCh)
Sound_JQ6500_Seq4(LED, InCh)
Sound_JQ6500_Seq5(LED, InCh)

Sound_JQ6500_PlayRandom(LED, InCh, MaxSoundNr)
Sound_JQ6500_Next_of_N_Reset(LED, InCh, InReset, MaxSoundNr)
Sound_JQ6500_Next_of_N(LED, InCh, MaxSoundNr)
```

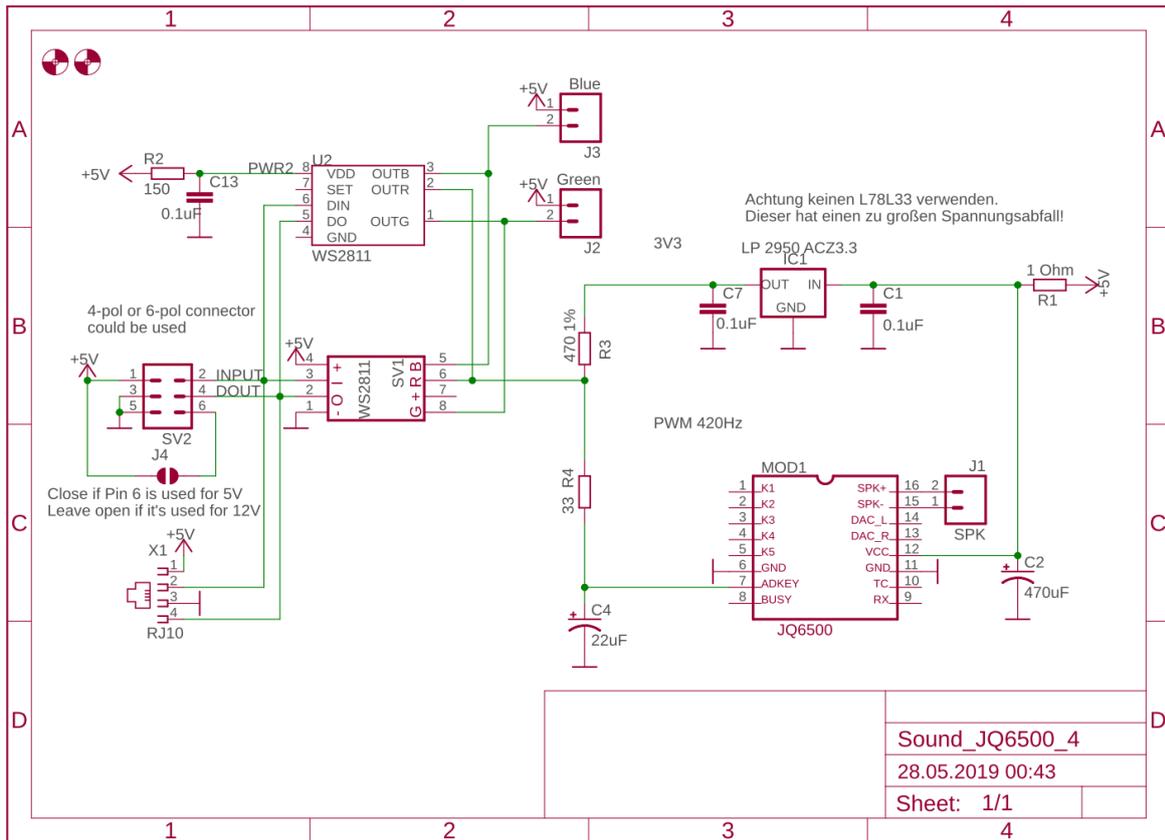
Zur Integration des Moduls in die MobaLedLib wird ein WS2811 Modul benötigt.

Dazu ist diese Platine verfügbar, die im Extras Verzeichnis der Bibliothek zu finden ist:



Sie kann mit verschiedenen Steckern (RJ10, 4-pol oder 6-pol Wannenstecker) bestückt werden. Außerdem kann ein WS2811 IC im DIL8 Gehäuse oder WS2811 Modul verwendet werden.

Hier der Schaltplan dazu:

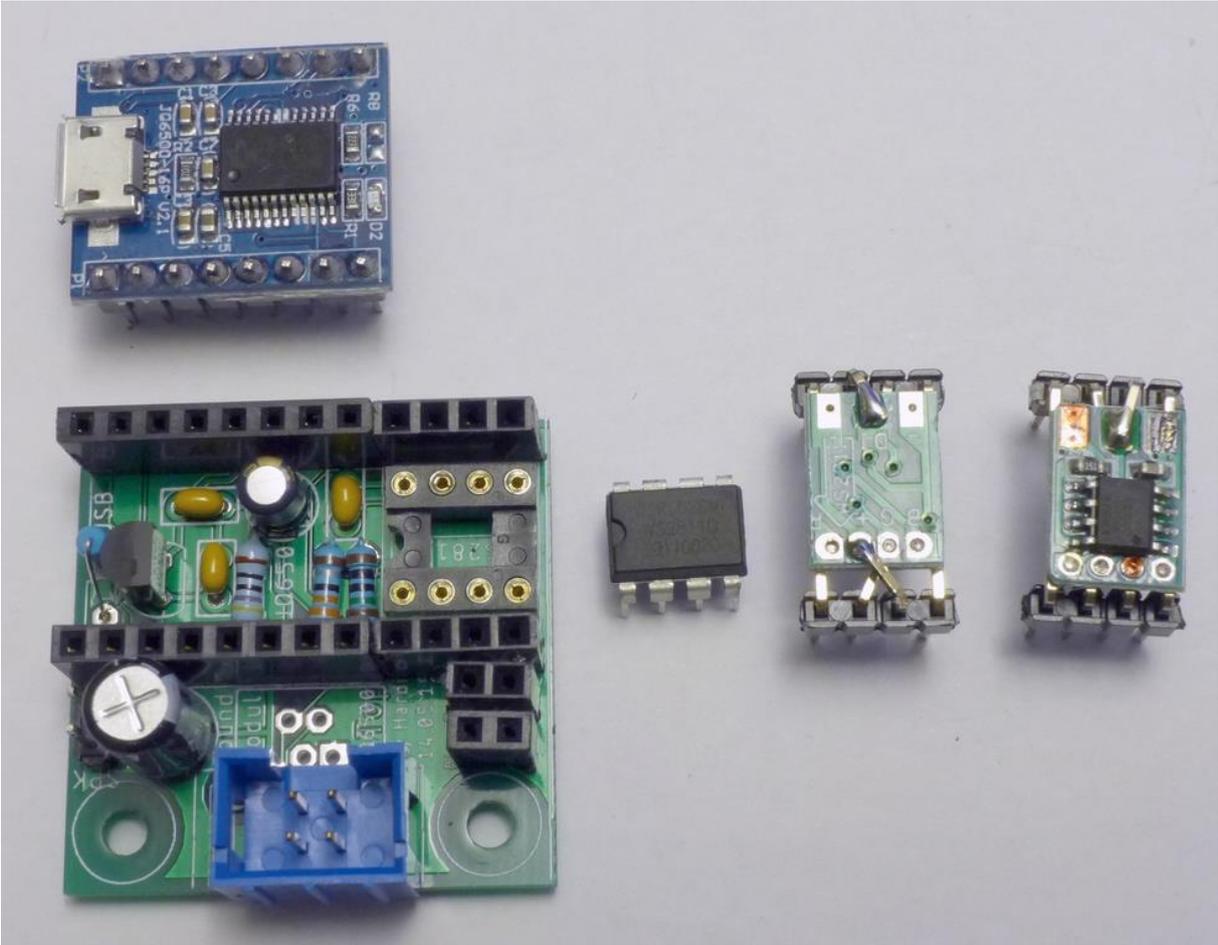


Leider gibt es verschiedene WS2811 Module, die sich durch die Pin Belegung und die elektrischen Eigenschaften unterscheiden. Die Sound Platine wurde für das im folgenden Bild rechte Modul entworfen. Das zweite Modul ist erst später dazu gekommen.

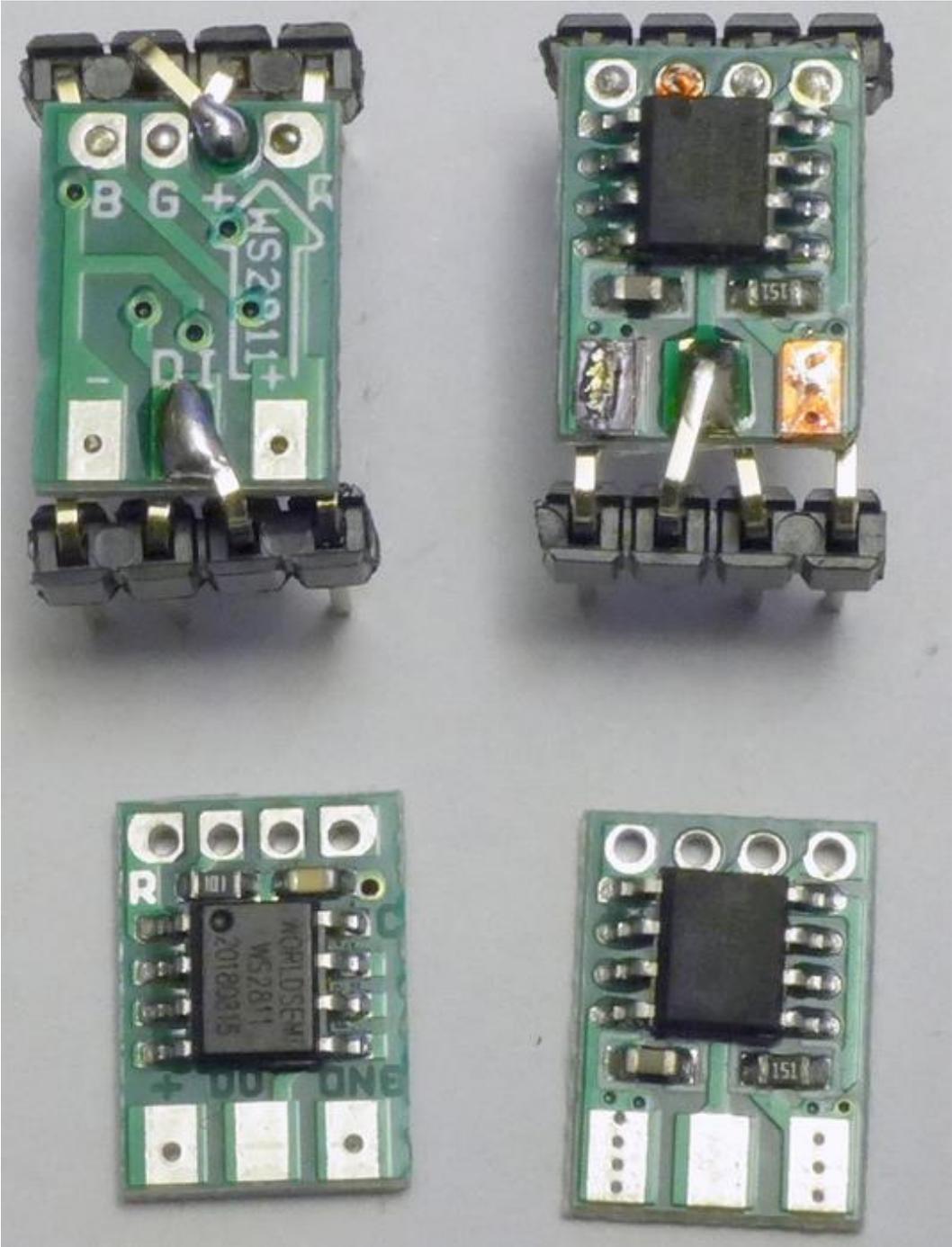
Die WS2811 Module werden über gewinkelte Stecker auf die Sound Platine gesteckt. Dabei müssen die Anschlüsse der Stecker etwas gebogen werden. Bei dem zweiten Modul überkreuzen sich die Anschlüsse auch noch, weil es ein anderes Anschlussbild besitzt. Trotz der Kreuzung sind der blaue und grüne Ausgang gegenüber dem ersten Modul vertauscht. Außerdem generiert das zweite Modul andere PWM Signale, die über eine modifizierte Tabelle in der Bibliothek ausgeglichen wurden.

Am Sichersten ist die Verwendung eines WS2811 ICs im DIL8 Gehäuse wie es im Bild in der Mitte gezeigt ist.

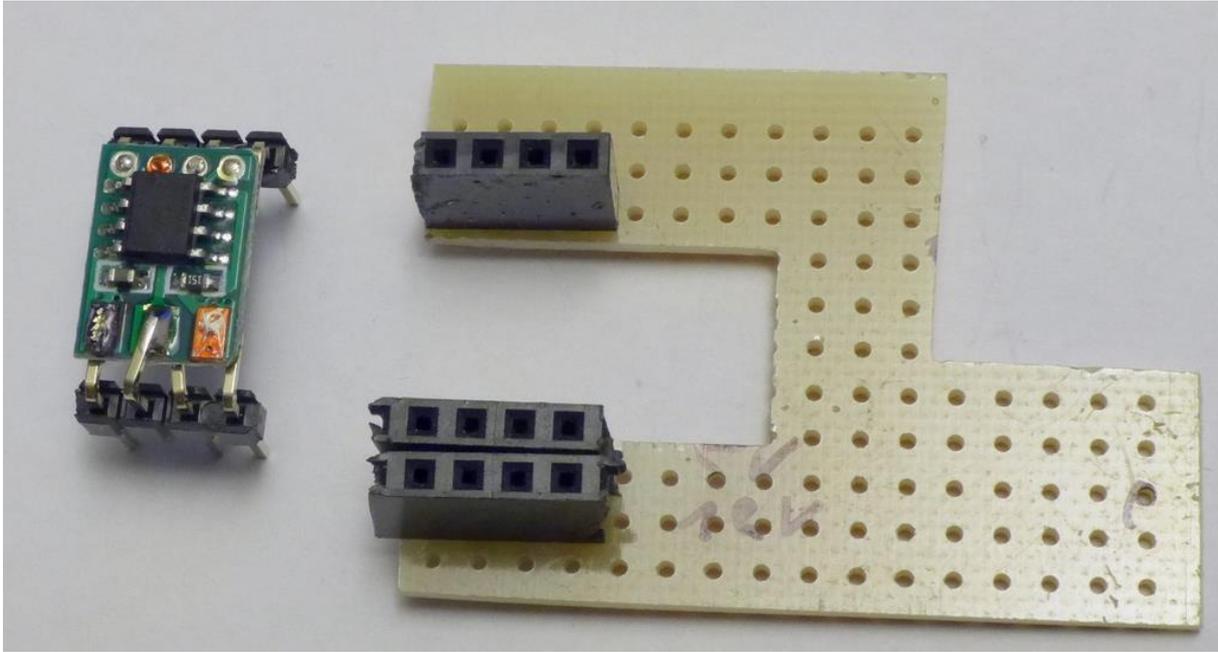
Wenn das Modul mit den vertauschten Ausgängen benutzt werden soll, dann müssen die Namen der verwendeten Sound Makros um ein „_BG“ ergänzt werden: Sound_JQ6500_BG_Seq1



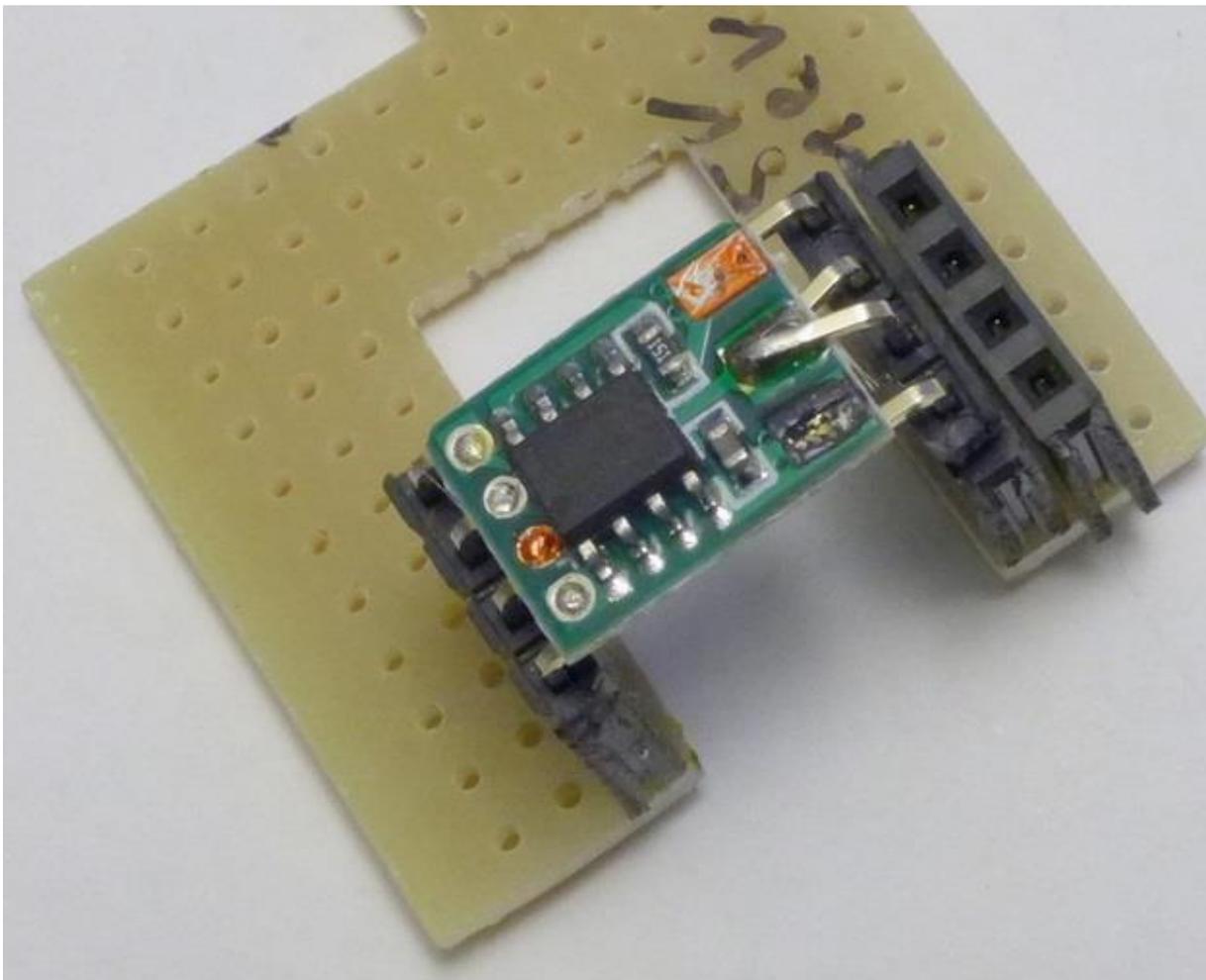
Hier noch mal eine Detailansicht der beiden WS2811 Varianten von oben und unten. Links das Modul mit den vertauschten Anschlüssen und den veränderten PWM Signalen und rechts das zum DIL8 IC kompatible Modul:



Zum Anlöten der Anschlussstecker kann ein solches Werkzeug benutzt werden:

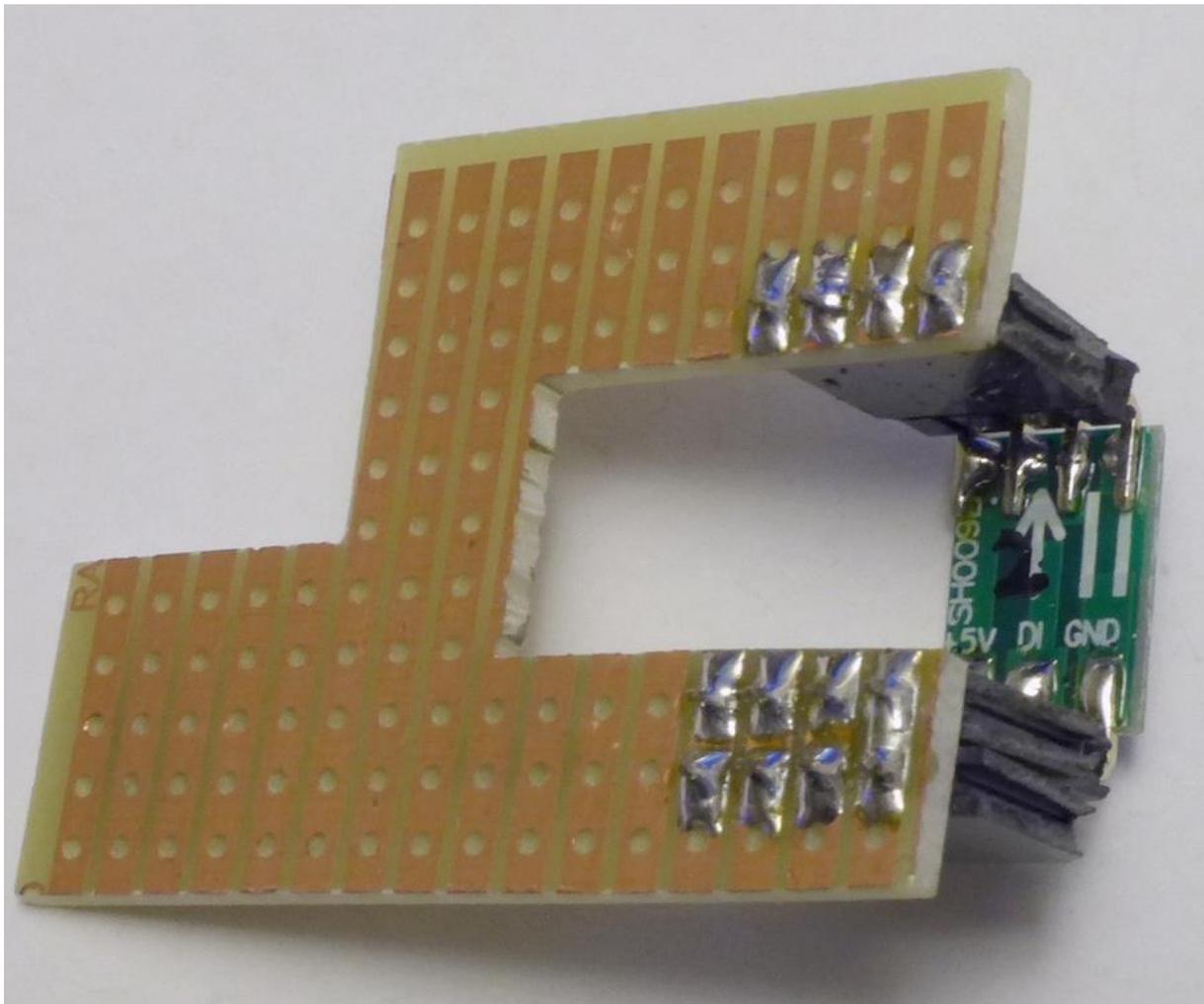


Die Pins, die an der Oberseite der Platine gelötet werden, müssen etwas herausgezogen werden, während die anderen etwas tiefer in den Kunststoffhalter geschoben werden. Man sieht auch auf den Bildern, dass einige der Anschlüsse gebogen werden.



Die zweite (unbenutzte) Steckerreihe ist für ein 12V Modul vorgesehen, das hier nicht verwendet werden kann.

Durch den Ausschnitt in der Platine kann man auch die unteren Lötstellen erreichen.



2.7 Steuer Befehle

Mit den in diesem Abschnitt beschriebenen Befehlen werden eine oder mehrere Variablen gesetzt. Diese Variablen können dann von anderen Makros eingelesen und ausgewertet werden. Wie im Abschnitt „2.2 Variablen / Eingangskanäle“ auf Seite 9 beschrieben, gibt es 256 Eingangsvariablen. Aus der Sicht der hier vorgestellten Funktionen sind es Ausgangsvariablen. Sie werden in der Parameterliste mit „DestVar“ bezeichnet. An diese Stelle im Makro kommt die Nummer der Variable. Es wird empfohlen, dass anstelle der Nummer ein symbolischer Name verwendet wird. Dieser kann zu Beginn des Programms mit dem „#define“ Befehl definiert werden.

2.7.1 ButtonFunc(DestVar, InCh, Duration)

Dieses Makro entspricht einem Treppenhaus Lichtschalter. Die Ausgangsvariable „DestVar“ wird 1, wenn der Eingang „InCh“ aktiv ist. Der Ausgang bleibt nachdem der Eingang deaktiviert wurde für „Duration“ Millisekunden aktiv. Das Makro entspricht einem statischen, retriggerbaren Mono Flop. Die Zeit kann auch durch anhängen von „Sec“ oder „Min“ angegeben werden. Die maximale Zeit ist 17 Minuten.

2.7.2 Schedule(DstVar1, DstVarN, EnableCh, Start, End)

Mit dem „Schedule“ Makro kann ein Zeitplan für das Ein- und Ausschalten mehrerer Lichter erstellt werden. Dieser Plan gibt aber nur die groben Rahmenbedingungen vor. Wann die Ausgänge tatsächlich geschaltet werden, bestimmt das Programm zufällig damit ein realer Eindruck entsteht.

Geschaltet werden die Ausgangsvariablen „DstVar1“ bis „DstVarN“. Sie werden zufällig zwischen dem Zeitpunkt „Start“ und „End“ eingeschaltet, wenn es „Abend“ ist und genauso Zufällig wieder am „Morgen“ ausgeschaltet. Ob es „Abend“ oder „Morgen“ ist, bestimmt die globale Variable „DayState“. Sie ist „Abends“ auf „SunSet“ gesetzt und „Morgens“ auf „SunRise“. Die zweite Variable „Darkness“ bestimmt über eine Zahl zwischen 0 und 255 wie „Dunkel“ es ist. Damit repräsentiert sie die Zeit.

Die Zeit kann auf verschiedene Arten generiert werden. Der einfachste Zeitgeber ist die Helligkeit. Damit können die Lichter angeschaltet werden, wenn es dunkler wird. Sie kann mit einem lichtabhängigen Widerstand (LDR) gemessen werden. Diese Methode erlaubt eine sehr einfache und glaubhafte Steuerung. Der Vorteil dieses Verfahrens ist, dass die Lichter automatisch passend zur Beleuchtung im Raum geschaltet werden. Im Beispiel „Darkness_Detection“ wird diese Methode verwendet. In dem Beispiel wird auch gezeigt, wie man mit einem Schalter „Tag“ und „Nacht“ erzeugen kann.

Es ist aber auch möglich, die Zeit von einem externen Modellbahnzeitgeber zu empfangen. Sie kann z.B. über den CAN Bus eingelesen werden. Das ist dann sinnvoll, wenn die Raumbeleuchtung über die gleiche Zeit gesteuert wird.

Natürlich kann die Modellzeit auch über ein paar Zeilen im Programm erzeugt werden. Das zeigt das Beispiel „Schedule“. Hier wird der Wert der „Dunkelheit“ aus der Zeit abgeleitet.



Neben der „Dunkelheit“ („Darkness“) wird noch der Tageszustand („DayState“) für die „Schedule()“ Funktion benötigt. Beides sind globale Variablen welche entsprechend gesetzt werden müssen. Die Beispiele zeigen wie das gemacht werden kann.

2.7.3 Logic(DstVar, ...)

Mit der „Logic()“ Funktion können logische Verknüpfungen umgesetzt werden. Mit ihr werden mehrere Eingangsvariablen über „NOT“, „AND“ und „OR“ miteinander verknüpft und in die Ausgangsvariable „DstVar“ geschrieben. Die logischen Verknüpfungen müssen als Disjunktive Normalform (DNF) ausgedrückt werden. Bei dieser Darstellung werden Gruppen von „AND“ Verknüpfungen mit „OR“ kombiniert:

(A AND B) OR (A AND NOT C) OR D

Die Klammern werden bei der DNF nicht dargestellt (Implizite Klammerung). Im Makro sieht das dann so aus:

```
Logic(ErgVar, A AND B OR A AND NOT C OR D)
```

Die Eingangsvariablen A bis D müssen zuvor definiert werden:

```
#define A 1
#define B 2
#define C 3
#define D 4
```

Die „Logic()“ Funktion kann auch noch einen zweiten übergeordneten Steuereingang haben, mit dem die Verknüpfung Ein- und Ausgeschaltet werden kann:

```
Logic(ErgVar, ENABLE EnabInp, A AND B OR A AND NOT C OR D)
```

oder

```
Logic(ErgVar, DISBLE DisabInp, A AND B OR A AND NOT C OR D)
```

Das Beispiel „Logic“ zeigt die Verwendung der „Logic()“ Funktion.

2.7.4 Counter(Mode, InCh, Enable, TimeOut, ...)

Die „Counter()“ Funktion kann für die verschiedensten Aufgaben verwendet werden. Sie wird über einen „Mode“ Parameter gesteuert. Folgende Flags sind definiert. Mehrere Flags können mit dem oder Operator ‚|‘ verknüpft werden.

CM_NORMAL	Normaler Zähler
CF_INV_INPUT	InCh wird invertiert
CF_INV_ENABLE	Enable Eingang invertieren
CF_BINARY	Binär Zähler, sonst werden die Ausgänge einzeln nacheinander aktiviert
CF_RESET_LONG	Reset wenn Eingang länger als 1.5 Sekunden aktiv ist
CF_UP_DOWN	Vorwärts/Rückwärts Zähler („Enable“ => Rückwärts)
CF_ROTATE	Zähler beginnt wieder am Anfang wenn das Ende erreicht wurde
CF_PINGPONG	Zähler wechselt an den Enden die Richtung
CF_SKIPO	Überspringt die Null
CF_RANDOM	Zufälliger Zählerstand bei jedem Impuls am Eingang
CF_LOCAL_VAR	Zählerstand wird in die zuvor definierte Lokale Variable geschrieben (siehe „2.7.25 New_Local_Var()“ auf Seite 29 und „2.7.26 Use_GlobalVar(GlobVarNr)“ auf Seite 29)
CF_ONLY_LOCALVAR	Der Zählerstand wird nur in die Lokale Variable geschrieben. Es werden keine

weiteren Ausgänge benutzt. The Zahl nach „TimeOut“ enthält die Anzahl der Zählerstufen (0...N-1).

Die „...“ in der Funktionsbeschreibung repräsentieren eine Variable Anzahl von Ausgabekanälen. Hier werden die Nummern der verwendeten Variablen eingetragen. Diese Variablen dienen anderen Makros als Eingang. Im normalen Modus werden die Ausgänge nacheinander aktiviert. Wenn das Flag CF_BINARY angegeben ist, dann werden die Ausgänge wie die einzelnen Bits einer Binärzahl angesteuert.

Im Folgenden werden einige Makros vorgestellt welche auf dem „Counter()“ Makro basieren.

2.7.5 MonoFlop(DstVar, InCh, Duration)

Ein Mono Flop ist eine Funktion welche den Ausgang für eine bestimmte Zeit aktiviert, wenn am Eingang ein Wechsel von Null nach Eins (Positive Flanke) erkannt wurde. Die Zeitdauer wird mit jeder weiteren Flanke verlängert, aber nicht, wenn der Eingang dauerhaft aktiv ist.

2.7.6 MonoFlopLongReset(DstVar, InCh, Duration)

Ist ein Mono Flop, der zurückgesetzt werden kann, wenn der Eingang länger als 1.5 Sekunden aktiv ist. Die Dauer kann ebenso wie bei der vorangegangenen Funktion verlängert werden.

2.7.7 RS_FlipFlop(DstVar, S_InCh, R_InCh)

Ein Flip-Flop kann zwei Zustände annehmen (0 oder 1). Bei einem RS Flip-Flop werden die Zustände über zwei Eingänge bestimmt. Eine Positive Flanke an „S_InCh“ setzt das Flip-Flop (Ausgang = 1), eine Flanke an „R_InCh“ löscht das Flip-Flop (Ausgang = 0).

2.7.8 RS_FlipFlopTimeout(DstVar, S_InCh, R_InCh, Timeout)

Dieses Makro entspricht dem vorangegangenen. Hier existiert zusätzlich ein Parameter „Timeout“ der bestimmt, wann das Flip-Flop automatisch gelöscht wird.

2.7.9 T_FlipFlopReset(DstVar, T_InCh, R_InCh)

Der Ausgang eines „Toggle Flip-Flops“ wird bei jeder positiven Flanke an Eingang umgeschaltet. Diese Funktion hat zusätzlich einen „Reset“ Eingang mit dem das Flip-Flop auf Null gesetzt werden kann. Wenn dieser Eingang nicht benötigt wird, dann kann er mit „SI_0“ belegt werden.

2.7.10 T_FlipFlopResetTimeout(DstVar, T_InCh, R_InCh, Timeout)

Hat zusätzlich einen Parameter „Timeout“.

2.7.11 MonoFlopInv(DstVar, InCh, Duration)

Dieser Mono Flop besitzt einen inversen Ausgang. Das bedeutet, der Ausgang ist zu Beginn aktiv (1) und wird deaktiviert mit einer positiven Flanke an „InCh“. Die Zeit kann wie beim normale MF mit einer steigenden Flanke verlängert werden.

2.7.12 MonoFlopInvLongReset(DstVar, InCh, Duration)

Hier werden die Eigenschaften "Invers" und "Reset" kombiniert, wenn der Eingang länger als 1.5 Sekunden aktiv ist.

2.7.13 RS_FlipFlopInv(DstVar, S_InCh, R_InCh)

Dieses Flip-Flop ist zu Beginn aktiv.

2.7.14 RS_FlipFlopInvTimeout(DstVar, S_InCh, R_InCh, Timeout)

Hier kommt wieder der „Timeout“ Parameter dazu.

2.7.15 T_FlipFlopInvReset(DstVar, T_InCh, R_InCh)

Und noch ein Flip-Flop mit inversem Startwert.

2.7.16 T_FlipFlopInvResetTimeout(DstVar, T_InCh, R_InCh, Timeout)

Das ist das Letzte Flip-Flop mit einem Ausgang.

2.7.17 MonoFlop2(DstVar0, DstVar1, InCh, Duration)

Die Folgenden Makros besitzen zwei Ausgänge, die Invers zueinander geschaltet sind. Die Funktionen sind gleich wie bei den Vorangegangenen, darum wird hier auf eine detaillierte Beschreibung verzichtet.

2.7.18 MonoFlop2LongReset(DstVar0, DstVar1, InCh, Duration)

2.7.19 RS_FlipFlop2(DstVar0, DstVar1, S_InCh, R_InCh)

2.7.20 RS_FlipFlop2Timeout(DstVar0, DstVar1, S_InCh, R_InCh, Timeout)

2.7.21 T_FlipFlop2Reset(DstVar0, DstVar1, T_InCh, R_InCh)

2.7.22 T_FlipFlop2ResetTimeout(DstVar0, DstVar1, T_InCh, R_InCh, Timeout)

2.7.23 RandMux(DstVar1, DstVarN, InCh, Mode, MinTime, MaxTime)

Die „RandMux()“ Funktion aktiviert zufällig einen der Ausgänge, die über die Zahlen „DstVar1“ bis „DstVarN“ definiert werden. Über „MinTime“ wird bestimmt, wie lange ein Ausgang minimal aktiv ist. „MaxTime“ beschreibt analog die maximale Zeit, die der Ausgang aktiv bleiben soll. Das Programm bestimmt zwischen diesen beiden Eckpunkten einen zufälligen Zeitpunkt zu dem ein zufälliger anderer Kanal aktiviert wird. Mit dieser Funktion kann zum Beispiel zwischen verschiedenen Lichteffekten für eine Disco umgeschaltet werden.

Mit dem Flag „RF_SEQ“ als Parameter „Mode“ wird der nächste Ausgang nicht zufällig gewählt, sondern die Ausgänge werden nacheinander aktiviert.

2.7.24 Random(DstVar, InCh, Mode, MinTime, MaxTime, MinOn, MaxOn)

Die Funktion „Random()“ aktiviert einen Ausgang nach einer zufälligen Zeit. Die Einschaltdauer kann ebenfalls zufällig sein. Damit kann man einen Effekt zufällig steuern. Eine Anwendung dafür ist das Blitzlicht eines Fotografens, das ab und zu blitzen soll.

Mit den Parametern „MinTime“ und „MaxTime“ wird bestimmt in welchem zeitlichen Bereich die Funktion aktiv werden soll. Über „MinOn“ und „MaxOn“ wird die Dauer vorgegeben. Für das Blitzlicht wird hier „MinOn“ = „MaxOn“ = 30 ms verwendet (Siehe „2.5.1 Flash(LED, Cx, InCh, Var, MinTime, MaxTime)“ auf Seite 14).

Das Blitzlicht ist als Makro definiert:

```
#define Flash(LED, Cx, InCh, Var, MinTime, MaxTime) \
    Random(Var, InCh, RM_NORMAL, MinTime, MaxTime, 30 ms, 30 ms) \
    Const(LED, Cx, Var, 0, 255)
```

Ein anderes Beispiel für die Verwendung der „Random()“ Funktion ist die „RandWelding()“ Funktion auf Seite 15.

Die „Random()“ Funktion kennt momentan einen besonderen Mode: RF_STAY_ON. Mit diesem Schalter bleibt der Ausgang noch so lange an, bis die über „MinOn“ und „MaxOn“ vorgegebene Zeit abgelaufen ist und der Eingang nicht mehr aktiv ist. Das wird in dem „Button()“ Makro genutzt welches ebenso wie „Flash()“ und „RandWelding()“ die „Random()“ Funktion nutzt:

```
#define ButtonFunc(DstVar, InCh, Duration) \
    Random(DstVar, InCh, RF_STAY_ON, 0, 0, (Duration), (Duration))
```

2.7.25 New_Local_Var()

Die meisten Funktionen der MobaLedLib werden über eine der 256 logischen Variablen gesteuert. Die Nummer der verwendeten Variable wird als Parameter „InCh“ angegeben.

Es gibt aber auch Fälle in denen mehr als zwei Zustände (Ein/Aus) benötigt werden. Für diesen Zweck werden in der Bibliothek aber keine feste Anzahl von Variablen zur Verfügung gestellt, weil der RAM eines Arduinos sehr beschränkt ist.

Mit dem Makro „New_Local_Var()“ wird bei Bedarf eine Variable vom Typ „ControlVar_t“ angelegt. Sie kann Werte zwischen 0 und 255 annehmen und besitzt zusätzliche Flags mit denen Änderungen erkannt werden können. Diese Variable kann dann von einer Funktion gesetzt werden und von einer oder mehreren anderen Funktionen ausgewertet werden. Durch diesen Ansatz wird nur dann wertvoller RAM benutzt, wenn es tatsächlich nötig ist. Außerdem muss der Anwender sich nicht wie bei der „Flash()“ oder „RandWelding()“ Funktion um eine Zwischenvariable kümmern.

Zum Setzen der lokalen Variable wird der Befehl „Counter()“ (Seite 26) benutzt.

Ausgewertet wird die Variable mit den Pattern Funktionen (ab Seite 41).

Das Beispiel „RailwaySignal_Pattern_Func“ zeigt wie das gemacht wird.

Der Speicher für die Variable wird automatisch angelegt. Dazu wird nicht wie bei C++ üblich, der „new“ Befehl benutzt, sondern beim Kompilieren das Array „Config_RAM[]“ in der benötigten Größe statisch angelegt. Das hat den Vorteil, dass der RAM Bedarf schon beim Kompilieren bekannt ist und der Compiler gegebenenfalls Warnungen generieren kann falls der Speicher knapp wird. In diesem Fall wird folgende Message in der Arduino IDE gezeigt:

```
Sketch uses 20388 bytes (66%) of program storage space. Maximum is 30720 bytes.
Global variables use 1556 bytes (75%) of dynamic memory, leaving 492 bytes for
local variables. Maximum is 2048 bytes.
Low memory available, stability problems may occur.
```

Die Warnung erscheint, weil nur noch wenig RAM Speicher für das Programm übrig ist. Dieser Speicher wird im C++ Programm für dynamisch angelegte Variablen und für den Stack benötigt. Es ist aus der Sicht des Compilers nicht möglich zu bestimmen wie viel Speicher dazu genau benötigt wird. Darum wird der Speicher in der Bibliothek statisch reserviert.

2.7.26 Use_GlobalVar(GlobVarNr)

Die Bibliothek kann mit eigenen Funktionen im C++ Programm erweitert werden. Mit der Funktion „Use_GlobalVar()“ können die eigenen Programmteile mit den bibliotheksinternen Funktionen Daten austauschen. Die globalen Variablen können genauso benutzt werden wie die lokalen Variablen welche mit der Funktion „New_Local_Var()“ angelegt werden. Die Globalen Variablen werden allerdings in einem eigenen Array abgelegt. Dieses Array muss im Sketch des Benutzers definiert werden:

```
ControlVar_t GlobalVar[5];
```

und der Bibliothek in der „setup()“ Funktion bekanntgegeben werden:

```
MobaLedLib_Assigne_GlobalVar(GlobalVar);
```

Damit kann dann der Befehl „Use_GlobalVar()“ benutzt werden. Zum Setzen der Variable kann z.B. diese Funktion in den Sketch des Benutzers eingefügt werden:

```
//-----
void Set_GlobalVar(uint8_t Id, uint8_t Val)
//-----
{
    uint8_t GlobalVar_Cnt = sizeof(GlobalVar)/sizeof(ControlVar_t);
    if (Id < GlobalVar_Cnt)
    {
        GlobalVar[Id].Val = Val;
        GlobalVar[Id].ExtUpdated = 1;
    }
}
```

2.7.27 InCh_to_TmpVar(FirstInCh, InCh_Cnt)

Mit diesem Befehl wird eine temporäre 8 Bit Variable mit den Werten aus mehreren Logischen Variablen gefüllt. Im Gegensatz zur „New_Local_Var()“ Funktion und zur „Use_GlobalVar()“ Funktion wird hier kein zusätzlicher Speicher benötigt. Das ist möglich, weil derselbe Speicher mehrfach benutzt werden kann. In den beiden anderen Fällen muss gespeichert werden ob sich der Eingang verändert denn nur dann soll eine Aktion ausgelöst. Bei dieser Funktion wird die Änderung der Eingangsvariablen benutzt.

Im Beispiel „CAN_Bus_MS2_RailwaySignal“ wird die „InCh_to_TmpVar()“ Funktion benutzt.

2.8 Sonstige Befehle

2.8.1 CopyLED(LED, InCh, SrcLED)

Mit dem „CopyLED()“ Befehl wird die Helligkeit der drei Farben der „SrcLED“ in die „LED“ kopiert. Das ist zum Beispiel bei einer Ampel an einer Kreuzung sinnvoll. Hier sollen die gegenüberliegenden Ampeln das gleiche Bild zeigen.

Wenn zwei RGB LEDs das gleiche zeigen sollen, dann kann man das auch durch die elektrische Verkabelung erreichen. Dazu werden die „DIN“ Leitungen beider LEDs parallelgeschaltet.

Normalerweise sind alle LEDs in einer Kette aneinandergereiht, damit jede LED einzeln angesprochen werden kann. Wenn zwei LEDs genau das gleiche zeigen sollen, dann ist das Parallelschalten eine Alternative. Im Beispiel „TrafficLight_Pattern_Func“ ist das skizziert.

2.8.2 PushButton_w_LED_0_2(B_LED, B_LED_Cx, InNr, TmpNr, Rotate, Timeout)

Mit dieser Funktion wird ein Taster eingelesen mit dem die „Druckknopf Aktionen“ gesteuert werden können. Die Funktion zählt die Tastendrucke und aktiviert eine von 3 temporäre Variable. Zusätzlich wird die LED im Schalter angesteuert. Sie blinkt entsprechend der Anzahl der Tastendrucke. Beim ersten Tastendruck einmal, nach dem zweiten Tastendruck zweimal und entsprechend dreimal beim dritten Druck auf die Taste. Mit dem 4. Tastendruck beginnt der Zähler wieder mit 1.

Parameter:

B_LED: Nummer der RGB LED im Taster.
B_LED_Cx: Kanal der LED (C1 ... C3).
InNr: Nummer der Eingangsvariable welche mit dem Taster gesteuert wird.
TmpNr: Nummer der ersten Temporären Variable. Das Makro benutzt 3 Variablen.
Rotate: Wenn dieser Parameter auf 1 gesetzt wird, dann startet der Zähler am Ende erneut.
Timeout: Zeit nach dem die Funktion deaktiviert wird.

3 Makros und Funktionen des Hauptprogramms

Die folgenden Makros und Funktionen werden im Hauptprogramm, der .ino-Datei (oder auf Arduinisch „Sketch“) verwendet.

Die Makros wurden eingeführt damit die Beispielprogramme übersichtlicher und einfacher zu warten sind. Bei den Makros ist der eigentliche Name mit einem Unterstrich „_“ von dem führenden „MobaLedLib“ getrennt.

Die Makros und Funktionen müssen an bestimmten Stellen innerhalb des Programms stehen. Das wird in der Dokumentation der einzelnen Elemente beschrieben.

3.1 MobaLedLib

Die Bibliothek enthält mehrere Klassen die einzeln genutzt werden können. Im nächsten Abschnitt wird die Klasse Hauptklasse „MobaLedLib“ beschrieben.

3.1.1 MobaLedLib_Configuration()

Dieses Makro leitet die Konfiguration der LEDs ein. In der Konfiguration wird definiert, wie sich die LEDs und anderen Module verhalten sollen. Der Konfigurationsbereich wird in geschweifte Klammern „{...};“ eingeschlossen und mit einem Strichpunkt abgeschlossen. In der letzten Zeile der Konfiguration muss das Schlüsselwort „EndCfg“ stehen.

Das Makro steht im Hauptprogramm nach dem #include „MobaLedLib.h“.

Hier die Konfiguration aus dem Beispiel „House“:

```

//*****
// *** Configuration array which defines the behavior of the LEDs ***
MobaLedLib_Configuration()
{
  // LED:           First LED number in the stripe
  // |             Input channel. Here the special input 1 is used which is
  // |             always on
  // |             On_Min: Minimal number of active rooms. At least two rooms are
  // |             |       illuminated.
  // |             On_Max: Number of maximal active lights.
  // |             |       Rooms: List of room types (see documentation for possible
  // |             |       types).
  // |             |       |
  House(0, SI_1, 2, 5, ROOM_DARK, ROOM_BRIGHT, ROOM_WARM_W, ROOM_TV0, NEON_LIGHT,
        ROOM_D_RED, ROOM_COL2) // House with 7 rooms
  EndCfg // End of the configuration
};
//*****

```

Intern ist das Makro folgendermaßen definiert:

```
#define MobaLedLib_Configuration() const PROGMEM unsigned char Config[] =
```

Es definiert ein Array aus „unsigned char“ die im Bereich „PROGMEM“ stehen, der im FLASH des Arduinos liegt. Ohne das „PROGMEM“ würde das Array in den RAM kopiert, was aufgrund des geringen Speichers eines Arduinos nur bei kleinen Konfigurationen möglich wäre.

3.1.2 MobaLedLib_Create(leds)

Mit diesem Makro wird die Klasse „MobaLedLib“ erzeugt. Damit wird der Speicher reserviert und initialisiert. Der Parameter „leds“ teilt der Klasse mit, wo die Helligkeitswerte der Leuchtdioden gespeichert werden. Dabei handelt es sich um Array vom Typ „CRGB“, welcher in der „FastLEDs“ Bibliothek definiert ist.

Das Makro steht im Hauptprogramm nach der Definition des „leds“ Arrays:

```

CRGB leds[NUM_LEDS]; // Define the array of leds

MobaLedLib_Create(leds); // Define the MobaLedLib instance

```

Neben dem Speicher für die LEDs benötigt die Bibliothek Speicher für die einzelnen Konfigurationszeilen. Hierfür wurde eine besondere Methode verwendet. Der Speicher wird von jedem Eintrag in der Konfiguration beim Kompilieren reserviert. Üblicherweise wird das zur Laufzeit des Programms mit dem Befehl „new“ gemacht. Der „Übliche“ Ansatz hat aber den Nachteil, dass der Compiler nicht weiß, wie viel RAM im Betrieb benötigt wird. Bei der knappen Ressource RAM kann das bei einem Mikrokontroller schnell zu Problemen führen. Darum wurde hier ein anderer Weg benutzt. Das soll exemplarisch am Beispiel der „House()“ Funktion erklärt werden:

```
#define House(LED,InCh, On_Min, On_Limit, ...) \
    HOUSE_T, _CHKL(LED)+RAMH, _ChkIn(InCh), On_Min, On_Limit, \
    HOUSE_MIN_T, HOUSE_MAX_T, COUNT_VARARGS(__VA_ARGS__), __VA_ARGS__,
```

Entscheidend ist hier der Ausdruck „RAMH“, der den Speicherbedarf eines Hauses beschreibt. Er wird über ein weiteres Makro durch RAM2 ersetzt. Dieses Makro hat folgenden Aufbau:

```
#define RAM1 1 + __COUNTER__ - __COUNTER__
#define RAM2 RAM1+RAM1
```

Danach setzt sich „RAM2“ aus zweimal „RAM1“ zusammen. Das letztgenannte Makro ist das entscheidende. Es enthält das C++ Präprozessor Makro „__COUNTER__“, das einen Zähler repräsentiert, der jedes Mal um 1 erhöht wird, wenn er benutzt wird.

Bei der ersten Verwendung des Makros „RAM1“ ergibt sich folgende Situation:

$1 + 0 - 1$

Damit ist „RAM1“ = 0. Bei der zweiten Verwendung des Makros sieht es ähnlich aus:

$1 + 2 - 3$

Was ebenfalls 0 ist. Somit ist auch „RAM2“ und entsprechend „RAMH“ ebenso 0. Aber das entscheidende ist, dass „__COUNTER__“ verändert wurde. Bei jeder Verwendung von „RAM1“ wird der Zähler um zwei erhöht. Und genau das wird zur Deklaration des Konfigurationsspeichers „Config_RAM[]“ benutzt. Dieser ist ein Array vom Typ „uint8_t“:

```
uint8_t Config_RAM[__COUNTER__/2];
```

Mit dem „__COUNTER__“ Trick ist das Array genau so groß, dass es den benötigten Speicher für alle Konfigurationszeilen bereitstellen kann. Da die Präprozessor Makros beim Kompilieren ausgewertet werden, weiß der Compiler genau wie viel Speicher belegt ist und kann prüfen, ob der RAM des Prozessors dafür ausreichend ist. Wenn ein kritischer Wert überschritten wird, dann wird diese Warnung angezeigt:

```
Sketch uses 20388 bytes (66%) of program storage space. Maximum is 30720 bytes.
Global variables use 1556 bytes (75%) of dynamic memory, leaving 492 bytes for
local variables. Maximum is 2048 bytes.
Low memory available, stability problems may occur.
```

Auf diese Weise kann der Hauptspeicher überwacht werden, ohne dass dazu eine Programmzeile nötig ist. Außerdem erscheint die Warnung während des Erzeugens des Programms auf dem Bildschirm. Eine Fehlermeldung, welche vom Arduino zur Laufzeit generiert wird, kann mangels standardisiertem Ausgabegerät nicht zuverlässig angezeigt werden.

Das Makro „MobaLedLib_Create()“ enthält die Zeile zur Erzeugung des Arrays und die eigentliche Initialisierung der Klasse:

```
#define MobaLedLib_Create(leds) \
    uint8_t Config_RAM[__COUNTER__/2]; \
    MobaLedLib_C MobaLedLib(leds, sizeof(leds)/sizeof(CRGB), \
```

```
Config, Config_RAM, sizeof(Config_RAM));
```

3.1.3 MobaLedLib_Assigne_GlobalVar(GlobalVar)

Wenn in der Konfiguration das Makro „Use_GlobalVar()“ benutzt wird, dann muss die Bibliothek wissen wo sich die globalen Variablen befinden und wie viele davon verfügbar sind. Das wird ihr mit diesem Befehl mitgeteilt.

Die Funktion wird in der „setup()“ Funktion des Hauptprogramms aufgerufen. Das entsprechende Array muss vorher deklariert werden:

```
ControlVar_t GlobalVar[5];

void setup(){
    MobaLedLib_Assigne_GlobalVar(GlobalVar); // Assigne the GlobalVar array to the MobaLedLib
}
```

3.1.4 MobaLedLib_Copy_to_InpStruct(Src, ByteCnt, ChannelNr)

Zur Steuerung der LEDs verwendet die MobaLedLib ein Array mit logischen Werten, das zusätzlich den vorangegangenen Wert speichert. Daraus kann die Bibliothek erkennen ob sich ein Eingang geändert hat und nur dann die entsprechende Aktion auslösen.

Wenn Eingangssignale vom Hauptprogramm aus in die sogenannte „InpStruct“ eingespeist werden sollen, dann wird dazu diese Funktion benutzt. Das wird im Beispiel „Switches_80_and_more“ zum Kopieren der Tastatur Arrays verwendet.

Als Eingang erwartet die Funktion ein Array mit einzelnen Bits, die die einzelnen Eingangskanäle repräsentieren. Der Parameter „Src“ enthält dieses Array. Mit dem Parameter „ByteCnt“ wird angegeben, wie viele Bytes kopiert werden sollen. „ChannelNr“ gibt die Zielposition in der Eingangsstruktur „InpStructArray“ an. Das entspricht dem „InCh“ in den Konfigurationsmakros. Achtung die „ChannelNr“ muss durch 4 teilbar sein.

Im Programm wird dieses Makro in der „loop()“ Funktion benutzt.

Wenn nur einzelne Bits in die Eingangsstruktur der Bibliothek kopiert werden sollen kann dazu der Befehl „Set_Input()“ benutzt werden (Siehe Abschnitt 3.1.6)

3.1.5 MobaLedLib.Update()

Dies ist die entscheidende Funktion der MobaLedLib. Sie berechnet die Zustände der LEDs in jedem Hauptschleifendurchgang neu. Sie arbeitet nacheinander alle Einträge im Konfigurationsarray ab und bestimmt damit die Farben und Helligkeiten der einzelnen LEDs. Bei der Entwicklung der Bibliothek wurde sehr großen Wert daraufgelegt, dass die Funktion auch bei großen Konfigurationen sehr schnell abgearbeitet wird.

Die Funktion muss in der „loop()“ Funktion des Arduino Programms aufgerufen werden.

3.1.6 MobaLedLib.Set_Input(uint8_t channel, uint8_t On)

Mit dieser Funktion kann eine Eingabevariable der MobaLedLib gesetzt werden. Damit können der Bibliothek Schalterstellungen oder andere Eingangswerte zugeführt werden. Mit dem Parameter „channel“ wird die Nummer der Eingangsvariable beschrieben welche in den Konfigurationsmakros immer „InCh“ verwendet wird.

Die Funktion wird in der „loop()“ Funktion und gegebenenfalls in der „setup()“ Funktion des Programms eingesetzt.

Sie wird in fast allen Beispielen zum Einlesen der Schalter benutzt. Im Beispiel „CAN_Bus_MS2_RailwaySignal“ werden damit die CAN Daten von der „Mobile Station“ eingelesen.

Wenn mehrere Bits auf einmal eingelesen werden sollen, dann kann das Makro „MobaLedLib_Copy_to_InpStruct“ eingesetzt werden, das auf Seite 33 beschrieben ist.

3.1.7 MobaLedLib.Get_Input(uint8_t channel)

Zum Lesen einzelner Eingangskanäle kann diese Funktion verwendet werden. Das kann vor allem zu Testzwecken sinnvoll sein.

3.1.8 MobaLedLib.Print_Config()

Mit dieser Funktion kann der Inhalt des Konfigurationsarrays zu Debug zwecken über die serielle Schnittstelle ausgegeben werden. Dazu muss aber die folgende Zeile in der Datei „Lib_Config.h“ aktiviert werden:

```
#define _PRINT_DEBUG_MESSAGES
```

und die serielle Schnittstelle mit „Serial.begin(9600);“ initialisiert werden. Die Datei „Lib_Config.h“ findet man unter Windows im Verzeichnis „C:\Users\<<Benutzername>\Documents\Arduino\libraries\MobaLedLib\src“.

Achtung: Dadurch wird sehr viel Speicher (4258 Byte FLASH, 175 Byte RAM) benötigt. Man sollte den Kompilerschalter also nur zu Testzwecken aktivieren.

Der Aufruf der Funktion und die Initialisierung der seriellen Schnittstelle wird in der „setup()“ Funktion gemacht.

3.2 Herzschlag des Programms

Die Klasse „LED_Heartbeat_C“ ist eine kleine Zusatzklasse mit der eine LED zur Überwachung der Funktionsweise des Mikrokontrollers und des darauf laufenden Programms eingesetzt werden kann. Wenn die LED regelmäßig blinkt, dann läuft das Programm normal.

Die Klasse kann unabhängig von der MobaLedLib Klasse eingesetzt werden.

3.2.1 LED_Heartbeat_C(uint8_t PinNr)

Die Klasse „LED_Heartbeat_C“ wird mit dem folgenden Aufruf im Hauptprogramm initialisiert. Der Parameter „PinNr“ enthält die Nummer des Arduino Anschlusses an dem die Leuchtdiode angeschlossen ist. Die digitalen Ein/Ausgänge eines Arduinos werden mit den Zahlen 2 bis 13 angesprochen. Die Analogen Eingänge 0-5 können ebenfalls zum Ansteuern der LED benutzt werden. Sie werden über die Konstanten A0 bis A5 ausgewählt. Die analogen Eingänge A6 und A7 des „Nano“ können nicht als Ausgang benutzt werden, weil sie keine entsprechende Ausgangsstufe haben. In den meisten Beispielen wird die Eingebaute LED des Arduinos benutzt, die über die Konstante „LED_BUILTIN“ an die Klasse übergeben wird. Bei den Beispielen die den CAN Bus nutzen geht das nicht, weil der Pin der internen LED gleichzeitig als Taktgenerator für den SPI Bus genutzt wird. Achtung: Hier muss eine einfache LED und keine RGB LED verwendet werden.

```
LED_Heartbeat_C LED_Heartbeat(LED_BUILTIN); // Use the build in LED as heartbeat
```

3.2.2 Update()

Die Funktionalität des Programms wird dadurch überprüft, dass die Funktion welche die Heartbeat LED blinken lässt in der „loop()“ Funktion des Programms aufgerufen wird. Wenn die LED blinkt, dann weiß man, dass das Programm regelmäßig die entsprechende Stelle aufruft. Dazu muss diese Zeile in die „loop()“ Funktion eingebaut werden:

```
LED_Heartbeat.Update(); // Update the heartbeat LED.
```

4 Viele Schalter mit wenigen Pins

Die Bibliothek stellt mit dem Modul „Keys_4017.h“ eine unglaublich flexible Methode zum Einlesen sehr **vieler Schalter** über **wenige Signalleitungen** zur Verfügung.

Wenn die Datei „Keys_4017.h“ in das Benutzerprogramm eingebunden ist, dann wird eine Interrupt Routine aktiviert, die im Hintergrund automatisch eine Matrix abfragt, die aus sehr vielen Schaltern bestehen kann. Das Besondere daran ist, dass dazu nur sehr wenige Signalleitungen benötigt werden. Das ist wichtig, weil der Arduino nur über eine beschränkte Anzahl an Ein- und Ausgängen verfügt. Wenige Signalleitungen sind auch dann wünschenswert, wenn die Schalter nicht direkt in der Nähe des Arduinos sind. Dadurch spart man Kabel und Steckverbinder.

Die Schalter können unabhängig voneinander eingelesen werden. Es können beliebig viele Schalter gleichzeitig aktiviert werden.

Alle Schalter werden innerhalb von 100 Millisekunden eingelesen. Damit ist eine sofortige Reaktion auf die Änderung eines Schalters gewährleistet.

Die Verwendung im Anwenderprogramm ist sehr einfach, weil die Abfragen automatisch im Hintergrund erfolgen.

Das Modul kann unabhängig von der MobaLedLib eingesetzt werden.

4.1 Konfigurierbarkeit

Welche und wie viele Anschlüsse zum Einlesen der Schalter benützt werden kann frei konfiguriert werden. Minimal sind drei Prozessoranschlüsse zum Einlesen der Schalter nötig. Es können aber auch bis zu zehn Pins benutzt werden. Zum Einlesen der Schalter werden ein oder mehrere ICs vom Typ CD4017 (0.31 €) benötigt. Die Anzahl dieser Bausteine hängt von der Anzahl der einzulesenden Schalter und der Anzahl der Signalleitungen ab.

Mit einem CD4017 und drei Signalleitungen können bereits 10 Schalter verarbeitet werden. Mit jeder zusätzlichen Signalleitung können 10 weitere Schalter eingelesen werden. Bei 10 Leitungen kann man auf diese Weise bereits 80 Schalter lesen. Theoretisch sind auch mehr als 10 Leitungen möglich. Dann müssen allerdings größer Pull Down Widerstände verwendet werden als in der unten gezeigten Schaltung, weil sonst der Ausgangsstrom der ICs zu groß werden kann.

Die Anzahl der Schalter lässt sich aber auch durch den Einsatz mehrerer CD4017 erhöhen. Mit zwei Bausteinen und drei Signalleitungen können 18 Schalter gelesen werden. Werden drei ICs eingesetzt, dann können 26 Schalter gelesen werden. Bei 10 ICs wären es 82 Schalter welche über nur drei Signalleitungen vom Arduino ausgelesen werden können.

Die Zahl der Schalter berechnet sich aus:

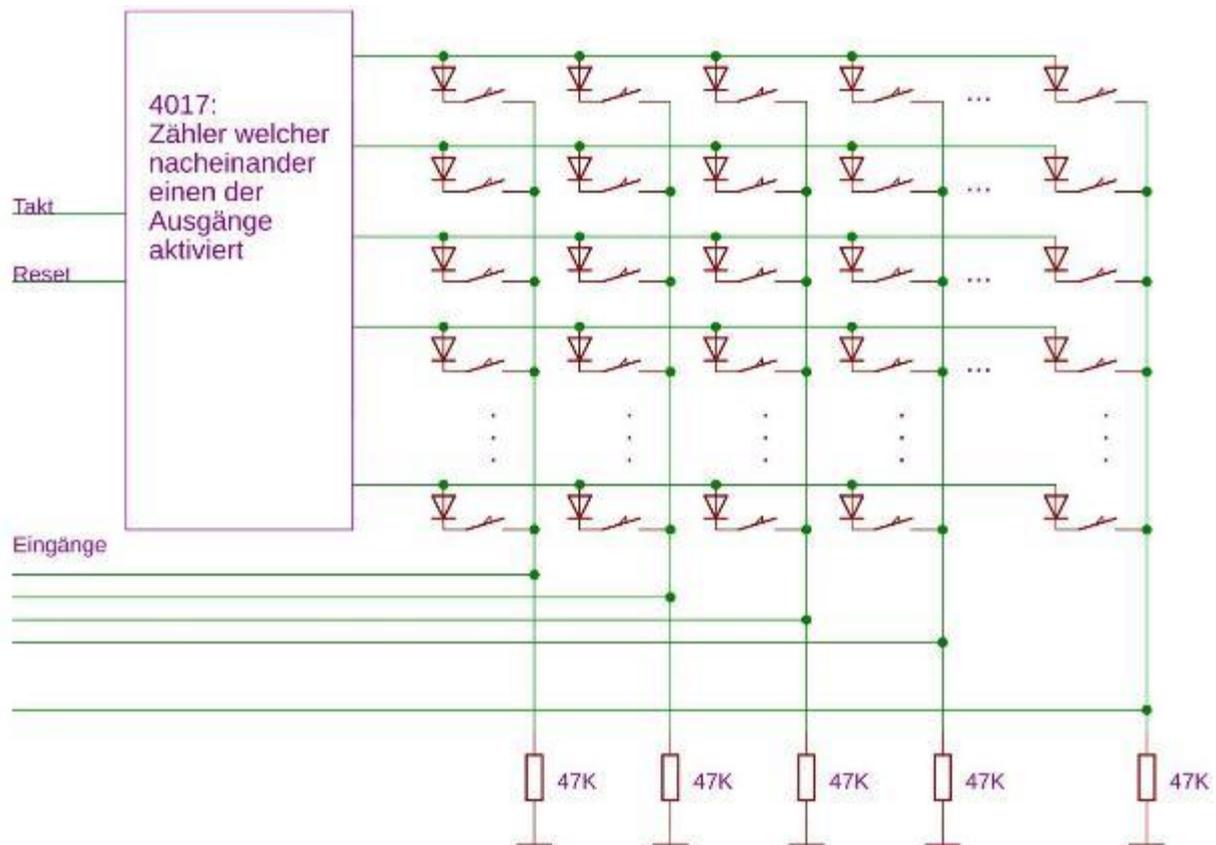
$$(\text{IC Anzahl} * 8 + 2) * (\text{Signalleitungen} - 2)$$

4.2 Zwei Schaltergruppen

Das Modul kann gleichzeitig zwei solche Schaltergruppen einlesen. Eine Gruppe kann beispielsweise in einem Weichenstellpult untergebracht sein und eine andere Gruppe kann am Rand der Anlage verteilte Schalter einlesen. Die erste Gruppe kann z.B. aus 80 Schaltern bestehen welche über 10 Signalleitungen eingelesen werden. Die zweite Gruppe kann aus mehreren Modulen mit jeweils einem CD4017 bestehen welche über nur 3 Signalleitungen miteinander verbunden sind. Diese Schalter können dann so genannte „Knopfdruck Aktionen“ einlesen. zwei der Signalleitungen werden dabei von beiden Gruppen gemeinsam genutzt so dass in Summe nur 11 Anschlüsse des Arduinos benötigt werden!

4.3 Prinzip

Das IC CD4017 ist ein Zähler der mit jedem Eingangsimpuls nacheinander seine Ausgänge aktiviert.



Zu Beginn ist der oberste Ausgang des Zählers aktiviert. Damit können die Schalter in der oberen Reihe eingelesen werden. Mit jedem Taktsignal am Eingang des Zählers wird der nächste Ausgang aktiviert. Im zweiten Schritt können so die Schalter aus der zweiten Reihe eingelesen werden. Der Baustein hat zehn Ausgänge. Damit können bei acht Eingangskanälen 80 Schalter gelesen werden. Die Dioden in der Schaltung verhindern, dass sich die Schalter gegenseitig beeinflussen.

4.4 Integration in das Programm

Zur Integration des Moduls in das Benutzerprogramm werden nur wenige Zeilen benötigt:

```
#define CTR_CHANNELS_1    10
#define BUTON_INP_LIST_1 2,7,8,9,10,11,12,A1
#define CTR_CHANNELS_2    18
#define BUTON_INP_LIST_2  A0
#define CLK_PIN            A4
#define RESET_PIN         A5

#include "Keys_4017.h"
```

Mit den „#defines“ werden die verwendeten Zählerkanäle und die Pins des Arduinos festgelegt. Das Beispiel oben definiert zwei Schaltergruppen.

Die erste Gruppe benutzt alle zehn Kanäle eines CD4017 (CTR_CHANNELS_1 10). Die Konstante „BUTON_INP_LIST_1“ enthält eine Liste mit acht Eingangs Pin Nummern. Damit besteht diese Gruppe aus 80 Schaltern.

Die zweite Gruppe wird mit der Konstante „CTR_CHANNELS_2“ und „BUTON_INP_LIST_2“ parametrisiert. Hier sind zwei Zähler ICs verwendet die über einen Eingang eingelesen werden. Es sind also 18 Schalter in der Gruppe.

Mit den beiden letzten Konstanten wird der Anschluss der Taktleitung und der Resetleitung spezifiziert. Diese Signale werden von beiden Gruppen gemeinsam benutzt.

Das Modul fragt alle Schalter innerhalb von 100 Millisekunden ab. Damit ist eine sofortige Reaktion auf die Änderung eines Schalters gewährleistet. Es schreibt den Zustand der Schalter in ein Bit Array. Für jede Gruppe existiert ein eigenes Array:

```
uint8_t Keys_Array_1[KEYS_ARRAY_BYTE_SIZE_1];  
uint8_t Keys_Array_2[KEYS_ARRAY_BYTE_SIZE_2];
```

welches vom Anwenderprogramm gelesen werden kann. Zur Integration dieser Arrays in die MobaLedLib Klasse werden die folgenden Zeilen in der „loop()“ Funktion des Programms benötigt:

```
MobaLedLib_Copy_to_InpStruct(Keys_Array_1, KEYS_ARRAY_BYTE_SIZE_1, 0);  
MobaLedLib_Copy_to_InpStruct(Keys_Array_2, KEYS_ARRAY_BYTE_SIZE_2, START_SWITCHES_2);
```

4.5 Frei verfügbare Platine

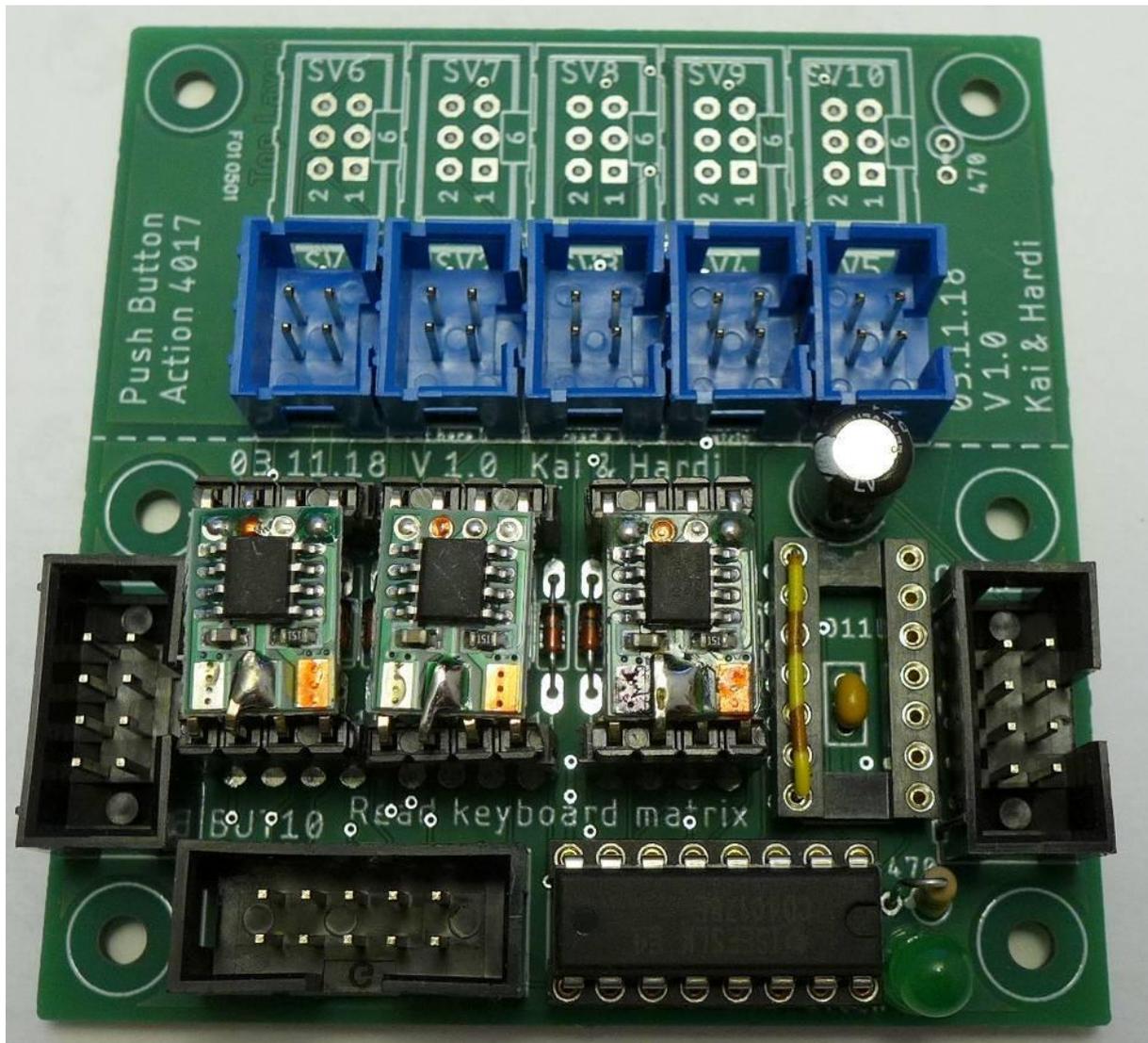
Im „extras“ Verzeichnis der Bibliothek befindet sich die Datei S3PO_Modul_WS2811.zip mit dem Schaltplan und der entsprechenden Platine zum Einlesen von Schaltern über dieses Modul.

Die Platine enthält neben den CD4017 und einem NAND Gatter mit dem die Signale zum nächsten Zähler weitergereicht werden noch drei WS2811 Module mit denen Leuchtdioden in den Schaltern angesteuert werden können. Alternativ können auch Schalter mit integrierten RGB LEDs benutzt werden.

Die Schaltung kann für das verteilte Einlesen von „Druckknopf Aktionen“ und zum Einlesen von vielen Schaltern in einem Weichenstellpult eingesetzt werden.

Im folgenden Bild wird die Platine gezeigt. Sie ist so bestückt, dass damit 5 Taster für Druckknopfaktionen eingelesen werden können. Von den drei WS2811 Modulen im Bild werden eigentlich nur die ersten beiden benötigt, wenn nur Taster verwendet werden. Die Kabelbrücke in der IC Fassung wird eingesetzt, wenn die Platine zum Einlesen der letzten Druckknopf Aktionen verwendet wird. Normalerweise steckt an dieser Stelle ein CD4011, der die Signale zur nächsten Schaltung generiert.

Die Platinen kann an der gestrichelten Linie abgesägt werden und in einem Weichenstellpult zum Einlesen von bis zu 89 Schaltern verwendet werden, wie das im Schaltplan oben gezeigt ist.



Eine Ausführliche Dokumentation der Schaltung wird bei Bedarf nachgereicht (Mail an MobaLedLib@gmx.de).

4.6 Zusätzliche Bibliotheken

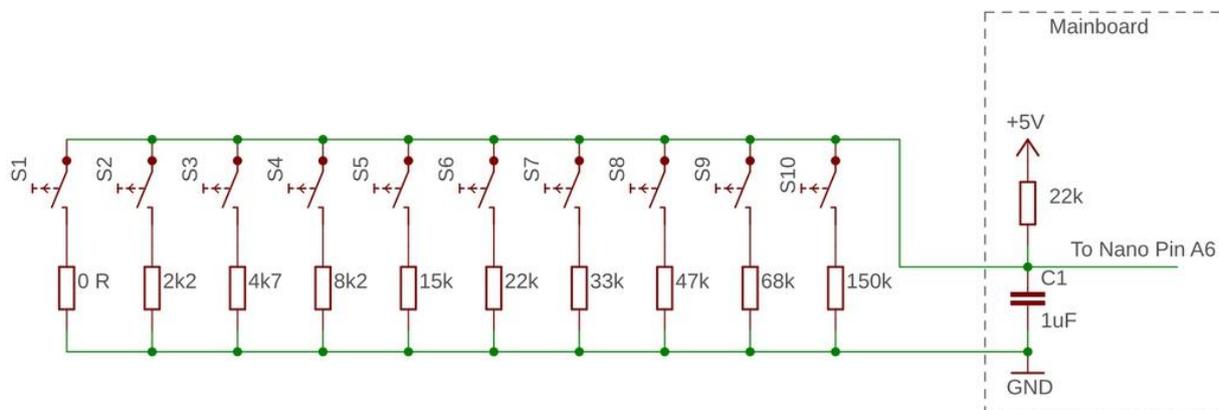
Das Modul benutzt die Bibliotheken „TimerOne.h“ und „DIO2.h“. Beide können über die Arduino IDE installiert werden. Dazu ruft man den die Bibliotheksverwaltung auf (Sketch / Bibliothek einbinden / Bibliothek verwalten) und gibt „TimerOne“ bzw. „DIO“ in das „Grenzen Sie ihre Suche ein“ Feld ein. Der gefundene Eintrag muss ausgewählt werden und kann dann über „Install“ installiert werden.

4.7 Einschränkungen

Die Tasten werden per Interrupt eingelesen. Dazu wird der Timer Interrupt 1 benutzt. Darum kann dieser Interrupt nicht mehr für andere Aufgaben benutzt werden. Standardmäßig wird der Timer 1 für die Servo Bibliothek benutzt. Wenn die Schalter über dieses Modul eingelesen werden, dann können nicht gleichzeitig Servos angesteuert werden. Das ist aber sowieso in Verbindung mit der „FastLED“ Bibliothek auf die das ganze Projekt aufbaut nicht möglich, weil die Interrupts während die LEDs aktualisiert werden gesperrt werden müssen, weil das Timing der WS281x Chips sehr kritisch ist.

4.8 Einfache Möglichkeit zum Einlesen von Druckknopf Aktionen

Es geht noch viel einfacher. Dazu benötigt man nur ein paar Widerstände und die Schalter:



Mit dieser Widerstandskaskade können bis zu 10 Taster über einen analogen Eingang des Arduinos eingelesen werden. Da ein Arduino Nano 8 analoge Eingänge besitzt, kann man auf diese Weise auch 80 Taster einlesen!

Allerdings dürfen die Taster nicht gleichzeitig betätigt werden, weil dann falsche Taster detektiert werden. Bei einer kleineren Hobbyanlage sollte das aber unkritisch sein. Diese Methode eignet sich nicht zum Einlesen von Schaltern mit denen man z.B. die Beleuchtung eines Hauses steuern kann, weil in diesem Fall mehrere Schalter gleichzeitig geschlossen sind. Dazu benötigt man die in den vorangegangenen Abschnitten gezeigte Lösung mit einem zusätzlichen Zähler Baustein.

Im Beispiel „25.Analog_Push_Button“ wird gezeigt wie man 10 Taster über einen Eingang des Arduinos einliest und damit verschiedene Aktionen steuert.

Für das Beispiel werden nur die ersten 5 Taster benutzt damit noch Platz für eigene Aktionen ist.

Die Taster werden gemäß dem oberen Bild an den Nano angeschlossen. Achtung: Den 22k Widerstand gegen +5V und den Kondensator nicht vergessen.

Der erste Taster steuert 6 Gas Straßenlaternen die zufällig nacheinander angehen und dabei ihre volle Helligkeit erst langsam erreichen. Durch Druckschwankungen oder Windböen passiert es manchmal, dass eine Lampe flackert. Bei einer der Lampen sind einige Glühstrümpfe defekt. Darum leuchtet sie schwächer als die anderen.

Mit dem zweiten Taster werden die Lichter in einem Bürogebäude ein und ausgeschaltet, welches mit Neonröhren beleuchtet ist. Hier werden die Lichter in den verschiedenen Räumen zufällig nacheinander eingeschaltet. Abhängig davon welches Helligkeitsbedürfnis die Preiser in den Büros haben.

Der Taster 3 aktiviert die Beleuchtung eines Feuerwehrfahrzeugs. Hier blitzen die Blaulichter mit geringfügig unterschiedlicher Frequenz und Frontscheinwerfer und Warnblinker blinken.

Wenn ein Sound Modul an die LED 15 angeschlossen wird, wie das hier (#13:

[https://www.stummiforum.de/viewtopic.php ... 3#p1912437](https://www.stummiforum.de/viewtopic.php...3#p1912437)) gezeigt wird, dann wird mit jedem Tastendruck ein zufälliger Sound wiedergegeben. Das macht den Besuchern besonders viel Spaß. Wenn Ihr anstelle eines Sound Moduls eine RGB LED anschließt, dann wird diese kurz aufblitzen.

Mit dem letzten belegten Taster wird auf unserer Anlage die Beleuchtung der verfallenen Burg aktiviert. Die „Knopf Druck Aktion“ hat zwei verschiedene Funktionen. Beim ersten Druck auf den Taster werden nacheinander weiße Strahler aktiviert. Mit dem zweiten Druck wird eine bunte Illumination eingeschaltet. Das ist etwas kitschig, gefällt mir aber trotzdem sehr gut.

Jeder Taster hat eine eigene Status LED welche anzeigt, ob die Aktion gerade aktiv ist. Die LEDs leuchten normalerweise blau und wechseln ihre Farbe bzw. blinken, wenn die Taster gedrückt wurden.

Über ein Timeout im Programm kann angegeben werden, wie lange so eine Aktion dauern soll.

5 Steuerung über eine Zentrale

Die MobaLedLib kann vollständig autark betrieben werden, ist aber auch in der Lage, auf äußere Einflüsse zu reagieren. Die einfachste Interaktion mit der Umwelt ist das Einlesen eines Helligkeitssensors, wie es im Abschnitt „2.7.2 Schedule(DstVar1, DstVarN, EnableCh, Start, End)“ beschrieben wurde.

Noch komfortabler aber auch komplexer wird es, wenn man die LEDs über eine Zentrale steuert. Das kann momentan über DCC oder den CAN Bus erfolgen. Die Hauptplatine ist bereits für das Einlesen von Kommandos über LocoNet vorbereitet. Das entsprechende Programm existiert aber leider noch nicht. Hier seid Ihr gefragt.

5.1 Einlesen von DCC Kommandos

DCC (Digital Command Control) ist ein weit verbreitetes Protokoll zur Steuerung von Modelleisenbahnen. Es kann zur Steuerung der Züge und Zubehörartikel verwendet werden. Das DCC Protokoll wird über die Schienenspannung übertragen und kann daher überall auf der Anlage unkompliziert abgegriffen werden.

Die MobaLedLib kann die Zubehörbefehle zum Schalten von Verbrauchern nutzen. Dazu wurden bis jetzt zwei Beispiele entwickelt, mit denen DCC Zubehörbefehle ausgewertet werden können. Das Einlesen der DCC Kommandos ist nicht in die Bibliothek integriert und wird im Beispiel selber gemacht. Zum Einlesen der Befehle wird die NmraDcc Bibliothek benutzt. Weil die Interrupts während der Ansteuerung der LEDs für längere Zeit gesperrt werden müssen und dabei DCC Kommandos verloren gehen können, wird ein zwei Prozessor System verwendet. Ein Arduino Nano liest die DCC Kommandos ein und speichert sie zwischen. Ein zweiter Arduino steuert die LEDs an. Er liest per serieller Schnittstelle die Kommandos vom „DCC Arduino“.

Das Beispiel „23_A.DCC_Rail_Decoder_Transmitter“ wird dazu auf den ersten Arduino geladen und das eigentliche Beispiel „23_B.DCC_Rail_Decoder_Receiver“ auf den LED Prozessor. Am ersten Arduino muss normalerweise nichts verändert werden. Er empfängt alle Zubehörbefehle und leitet sie an seinen Kollegen weiter.

Das Programm auf dem LED Arduino dagegen wird man an die eigenen Bedürfnisse anpassen. Das soll im Folgenden beschrieben werden.

5.1.1 Verschiedene DCC Kommandos

Es gibt verschiedene Arten von Zubehörkommandos. Hier soll nur auf die wichtigen Teile des „Einfachen“ Formats eingegangen werden.

Die DCC Zubehörkommandos wurden zunächst zum Schalten von Weichenantrieben oder Signalen entwickelt. Darum gibt es zu jeder Adresse eine Funktion, mit der die Weiche auf Geradeaus und eine mit der sie auf Abzweigung gestellt werden kann. Bei Signalen hat man die Zustände Halt und Fahren unterschieden. Oft verwendet man die Allgemeinere Beschreibung „Rot“ und „Grün“ für die beiden Zustände:



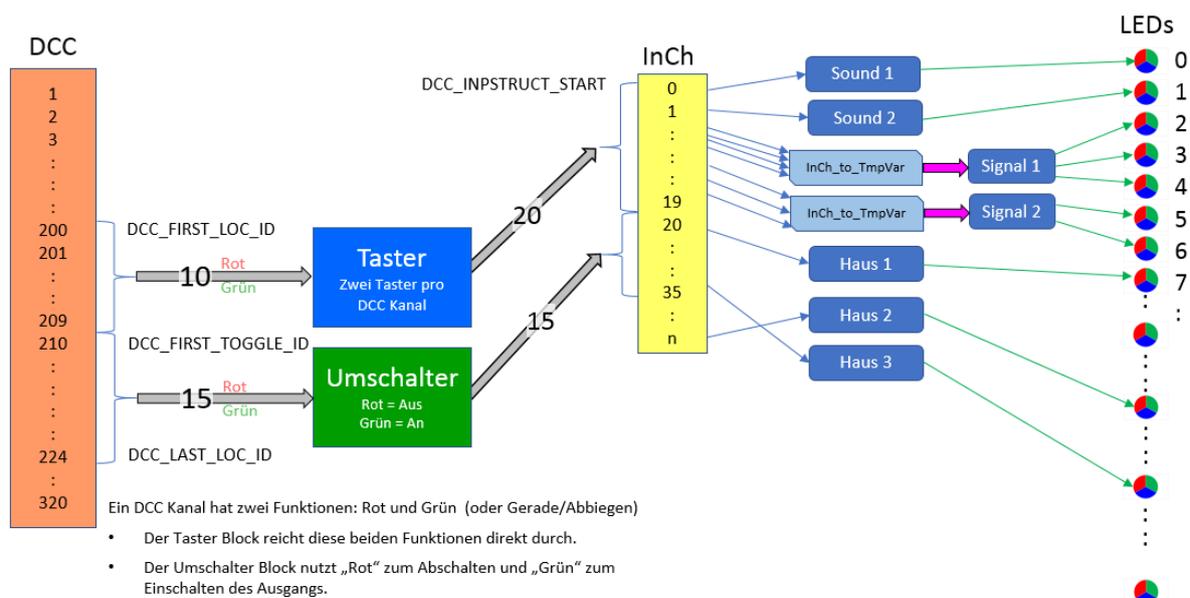
Diese beiden Zustände wertet auch das Beispielprogramm aus. Das wird auf zwei verschiedene Arten gemacht:

- Taster
- Umschalter

Die einfachste Variante zur Auswertung der DCC Kommandos ist, die Taster der Steuerung direkt zur Bibliothek weiterzuleiten. Immer dann, wenn eine Taste gedrückt ist, bekommt die Bibliothek eine entsprechende Meldung. Jeder Taster generiert eine eigene Meldung. Das bedeutet, dass jeder DCC Adresse zwei Tastereignissen zugeordnet sind. Die Ereignisse werden in zwei Variablen gespeichert welche aktiv sind während der entsprechende Taster gedrückt wird und deaktiviert wird, wenn der Taster losgelassen wird. Diese Art der Auswertung eignet sich z.B. zum Ansteuern von Sound Modulen. Hier wird pro Taste eine bestimmte Datei abgespielt.

Die zweite Methode zum Einlesen der Kommandos nutzt einen Speicher in dem der Zustand abgelegt ist. Dieser wird mit der einen Taste aktiviert und mit der anderen Taste deaktiviert. Damit kann man z.B. die Beleuchtung eines Hauses Ein- und Ausschalten.

In dem Beispielprogramm „23_B.DCC_Rail_Decoder_Receiver“ sind die beiden Varianten so implementiert, das im ersten Adressbereich die Kommandos als Taster ausgewertet werden und im zweiten Bereich als Umschalter genutzt werden. Das folgende Bild zeigt das im linken Teil:



Die DCC Kommandos 200 bis 209 werden als Taster eingelesen. Die folgenden Adressen ab 210 werden als Umschalter verwendet.

Den verwendeten DCC Adressbereich beschreiben drei Konstanten welche auch im Bild oben zu finden sind:

```
// Define which accessories CAN messages should be used.
#define DCC_FIRST_LOC_ID 200 // First local ID which should be copied to the
// InpStructArray[] of the MobaLedLib
#define DCC_FIRST_TOGGLE_ID 210 // DCC addresses greater equal this number are used
// to toggle an entry in the InpStructArray[]
// The DCC addresses smaller than this number are treated as
// momentary events.
#define DCC_LAST_LOC_ID 224 // Last local ID which should be copied to the
// InpStructArray[] of the MobaLedLib
```

Mit der Konstante „DCC_FIRST_LOC_ID“ bestimmt man die erste vom Programm verwendete lokale DCC Adresse und mit „DCC_LAST_LOC_ID“ entsprechend die letzte Adresse. Alle anderen DCC

Kommandos werden nicht beachtet. Mit „DCC_FIRST_TOGGLE_ID“ bestimmt man ab welcher DCC Adresse die Kommandos als Umschaltbefehle genutzt werden. Wenn man keine Taster verwenden will, dann kann man diese Konstante gleich der ersten Adresse setzen. Anders herum ist es genau so möglich nur Taster zu nutzen, indem „DCC_FIRST_TOGGLE_ID“ größer als die letzte benutzte Adresse ist. Diese Aufteilung in zwei Bereiche ist nicht zwingend. Es könnten auch beliebige andere Zuordnungen getroffen werden, indem das Beispiel entsprechend angepasst wird. Fortgeschrittene Programmierer können sich hier beliebig austoben.

Im Beispielprogramm „23_B.DCC_Rail_Decoder_Receiver“ werden noch diese Konstanten verwendet:

```
#define DCC_FIRST_LOC_ID      1
#define DCC_FIRST_TOGGLE_ID 11
#define DCC_LAST_LOC_ID     30
```

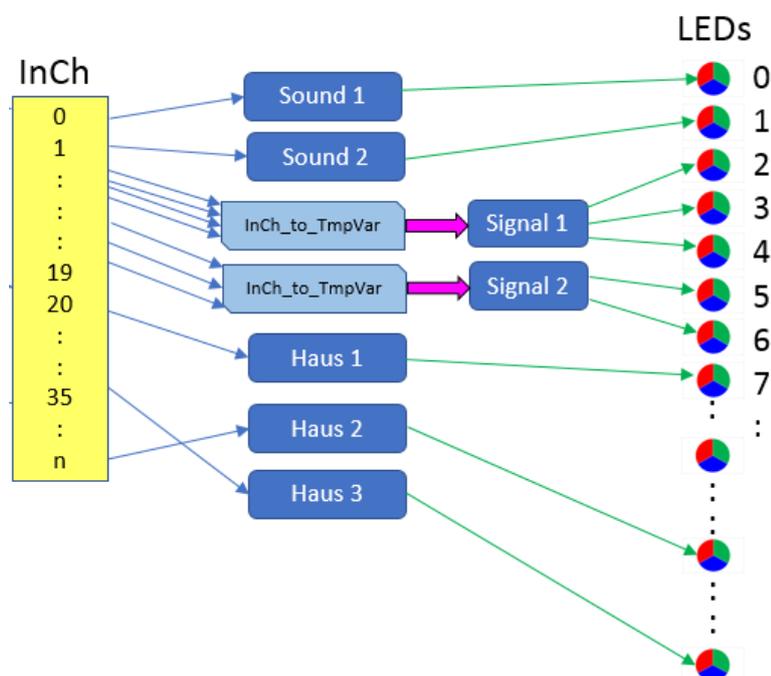
Für die Dokumentation wurde der benutzte Bereich verschoben. Das zeigt, dass man den verwendeten DCC Adressbereich sehr einfach anpassen kann. Aber dabei ist zu beachten, dass sich damit auch die Zuordnung zu den Eingangsvariablen (InCh) der LEDs verändert. Das tritt dann auf, wenn „DCC_FIRST_TOGGLE_ID“ relativ zu „DCC_FIRST_LOC_ID“ verschoben wird. Im Abschnitt „5.1.4 Anpassungen am Programm“ wird darauf nochmal eingegangen.

Unabhängig davon, ob die DCC Kommandos als Taster oder als Umschalter benutzt werden, wird ein Eintrag in dem Array „InCh[]“ gemacht. Dieses Array kann bis zu 256 Einträge fassen. Damit werden dann die LEDs oder andere Ausgänge angesteuert. Das beschreibt der nächste Abschnitt.

5.1.2 Vom InCh zu den LEDs

In dem Array „InCh[]“ werden die Zustände der DCC Kommandos abgelegt. Es kann aber auch von anderen Quellen beschrieben werden. In dem Beispiel „03.Switched_Houses“ der MobaLedLib wird gezeigt, wie man direkt am Arduino angeschlossene Schalter zum Setzen des „InCh[]“ Arrays benutzt. Dazu wird die gleiche Funktion „Set_Input()“ benutzt, die auch in dem Beispiel zum Einlesen der DCC Kommandos benutzt wird.

Die Ansteuerung der LEDs aus dem Array ist unabhängig davon wie „InCh[]“ befüllt wurde. Es gibt verschiedene Möglichkeiten wie man die LEDs ansteuert. Das folgende Bild zeigt drei Varianten.



1. Sound Module werden über Tastendrucke gesteuert. Wenn man eine bestimmte Taste betätigt wird die entsprechende Geräuschdatei abgespielt. Im Konfigurationsarray fügt man dazu die folgende Zeile ein:

```
Sound_Seq7(16, 1)
```

Der Sound Befehl erwartet zwei Parameter. Der erste Parameter bestimmt die LED Nummer. Mit dem zweiten Argument bestimmt man den Eingangskanal, also die Nummer in dem Array „InCh[]“. Die Nummer 7 im Namen des Befehls zeigt an, dass dieser Befehl die siebte Datei abspielt.

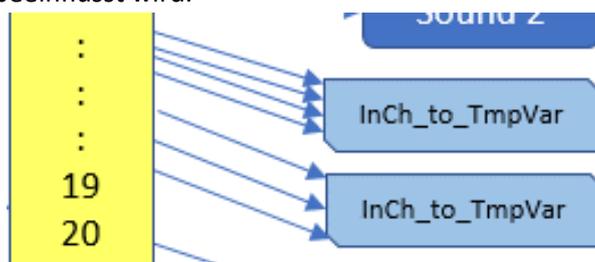
Für den „Sound 2“ im Bild oben würde man also

```
Sound_Seq2(1, 1)
```

verwenden. Für das JQ6500 Sound Modul entsprechend

```
Sound_JQ6500_Seq2(1, 1)
```

2. Das „InCh[]“ Array kennt zwei Zustände: 0 / 1 oder „Nicht Aktiv“ / „Aktiv“. Zur Ansteuerung von Lichtsignalen mit mehr als 2 Begriffen benötigt man mehr Zustände (Vier für ein Ausfahrtsignal mit HP0, HP1, HP2, HP0+SH1). Dafür stellt die MobaLedLib das Makro „InCh_to_TmpVar“ zur Verfügung. Es wird von mehreren Eingangskanälen beeinflusst. Das Bild zeigt, dass der erste Block von 4 Eingängen, der zweite Block von drei Eingängen beeinflusst wird.



Im Konfigurationsarray wird das für den ersten Block folgendermaßen definiert:

```
InCh_to_TmpVar(2, 4)
```

Mit dem ersten Argument wird bestimmt, dass die Eingänge ab dem „InCh“ 2 benutzt werden sollen. Der zweite Parameter definiert, dass 4 Eingangskanäle verwendet werden. Damit kann man ein 4 begriffiges Ausfahrtsignal steuern. Im Konfigurationsarray muss die Zeile zum Ansteuern des Signals direkt auf die „InCh_to_TmpVar“ Zeile folgen:

```
InCh_to_TmpVar(2, 4)
```

```
Dep_Signal4_RGB(2)
```

Mit diesen zwei Zeilen kann ein Signal per DCC gesteuert werden. Die 2 beim Dep_Signal4_RGB() Aufruf bestimmt dabei die Nummer der ersten LED des Signals.

3. Die dritte Möglichkeit zum Ansteuern der LEDs per DCC, die hier gezeigt werden soll, ist wieder einfach. Zum Schalten eines „Belebten“ Hauses wird nur eine Zeile im Konfigurationsarray benötigt:

```
House(7, 20, 1, 2, ROOM_DARK, ROOM_WARM_W, ROOM_TV0)
```

Mit der 7 wird, wie bei den anderen Funktionen, bestimmt welches die erste LED ist. Mit der 20 wird die Nummer innerhalb des „InCh[]“ Arrays angegeben. Die anderen Parameter bestimmen das Verhalten des „belebten“ Hauses. Die 1 sagt aus, dass mindestens ein Raum beleuchtet sein soll und die 2 bestimmt die maximal „benutzten“ Zimmer.

Eine Besonderheit der „House()“ Funktion ist ja, dass die Lichter erst zufällig nacheinander angehen, wenn man den Einschaltbefehl gibt. Damit man aber ein direktes Feedback bekommt, wenn man ein Haus per DCC ein- oder ausschaltet, wird sofort eine LED eingeschaltet bzw. ausgeschaltet, wenn das entsprechende Kommando kommt. Wenn man mit dem DCC Schalter „spielt“, wird mit der „Grünen“ Taste eine LED angehen und mit der

„Roten“ Taste eine LED ausgehen. Wenn man das schnell hintereinander macht, wird mit dem Ausschaltbefehl die eben angeschaltete Lampe wieder ausgeschaltet und mit dem nächsten Einschaltbefehl wieder zufällig eine andere Lampe angehen. Das verwirrt im ersten Moment. Ist das Einschaltkommando längere Zeit aktiv werden weitere Räume „bewohnt“ und evtl. geht das Licht auch in anderen Räumen wieder aus. Das hängt ganz von den Aktivitäten der „Preiser“ in dem Hause ab.

5.1.3 Verwirrende Nummern

Das Einlesen von DCC Kommandos wird von vielen Zahlen beeinflusst. Angefangen von der DCC Adresse über die Nummer im „InCh[]“ Array zu der Sequenz Nummer der LED. Erschwerend sind außerdem die unterschiedlichen Bereiche.

Bei den Tastern können mit einer DCC Adresse zwei Ausgänge angesteuert werden (Rot / Grün). Hier gibt es noch eine Besonderheit bei den Signalen, wo mehrere DCC Adressen ein Signal beeinflussen. Das Signal besteht aber wiederum aus verschiedenen LED Kanälen.

Im Bereich der Schalter ist die Zuordnung von DCC zu „InCh[]“ direkter, aber auch hier werden mehrere LEDs pro Haus angesprochen.

Bisher wurden nur einige der Möglichkeiten der MobaLedLib angesprochen. Es gibt noch viele weitere Funktionen, die es nicht einfacher machen.

Ein weiteres Problem tritt auf, wenn man die Geschichte erweitern will. Dann verschieben sich alle Nummern. Das fängt schon auf der DCC Seite an. Wenn man zunächst von 10 Tastern für die Wiedergabe von Geräuschen ausgegangen ist, aber dann ein weiterer Sound wiedergegeben werden muss, dann verschieben sich alle folgenden DCC Adressen. Man kann natürlich entsprechende Reserven einplanen, aber auch diese werden schneller als man glaubt belegt sein.

Auf der Seite der LEDs hat man das gleiche Problem. Wenn hier noch unbedingt eine weitere Attraktion eingebaut wird, führt das dazu, dass sich alle folgenden LED Nummern verschieben.

Wie kann man das Übersichtlicher gestalten?

Ganz einfach mit einer entsprechenden Dokumentation. Diese kann man zum Beispiel in Excel machen. Mit ein paar Tricks kann man aus der Dokumentation auch gleich den Quellcode für das Programm erzeugen. Ebenso lässt sich daraus ein „Handzettel“ erzeugen, mit dem man die Funktionen am Steuergerät wiederfindet. Evtl. kann man das gleiche Excel Sheet auch für die Konfiguration der Zentrale verwenden.

Im Programm kann man mit dem „#define“ Befehl symbolische Namen anlegen. Diese Namen werden dann im Konfigurationsarray anstelle der Nummern verwendet. Dadurch hat man alle Nummern an einer Stelle und kann diese einfach anpassen. Im Beispiel „03.Switched_Houses“ der Bibliothek wird das gezeigt:

```
#define INCH_HOUSE_A 0 // Define names for the input channels to be able to change them easily.
#define INCH_HOUSE_B 1 // In this small example this is not necessary, but it's useful in a
#define INCH_HOUSE_C 2 // large configuration.
#define INCH_SHOP_C 3

#define HOUSE_A 0 // Define names for the LED numbers of the houses.
// HOUSE_A 1 // In a real setup the names could be: "RailwayStation", "Town_Hall", "Pub", ...
// HOUSE_A 2 // Each room gets an own name.
// HOUSE_A 3 // Only the first LED numbers are used in the configuration,
// HOUSE_A 4 // But it's a good practice to list the other rooms to because
// HOUSE_A 5 // then the corresponding numbers are increasing without gaps.
// HOUSE_A 6 // This is useful if an additional house is inserted.
#define HOUSE_B 7 // In this case the sequence could easily be checked / updated and
// HOUSE_B 8 // the additional lines could be used for documentation
// HOUSE_B 9
// HOUSE_B 10
#define HOUSE_C 11
#define HOUSE_C_SHOP 12 // There is a shop in the house which is controlled separately
// HOUSE_C 13
```

```
// HOUSE_C 14
// HOUSE_C 15
```

Hier wird eine Reihe von Konstanten definiert, mit denen die Nummern im „InCh[]“ Array und die Adresse der LEDs beschrieben wird. Natürlich wird man auf der eigenen Anlage aussagekräftiger Bezeichnungen für die Konstanten verwenden. Anstelle von „HOUSE_A“ kann man z.B. „BAHNHOF_MAINZ“ verwenden. Auf diese Weise erkennt man sofort was gemeint ist.

Die ersten vier Konstanten beschreiben die Nummer im „InCh[]“ Array, die restlichen die LED Nummern. Hier fällt auf, dass zu einem Haus mehrere Zeilen benutzt werden von denen nur die erste ein vorangestelltes „#define“ hat. Alle anderen Zeilen sind Kommentare. Das ist bewusst so gemacht, damit man auf einen Blick erkennt, wie viele „Zimmer“ ein Haus hat. Diese Zeilen sind wichtig, damit man die Nummern der LEDs einfach hochzählen kann. Man kann sie aber auch noch zur Dokumentation der einzelnen Zimmer benutzen. Das ist für die Konfiguration hilfreich, wenn man weiß, welche LED das Treppenhaus beleuchtet, welche in einem Schaufenster verwendet wird oder welche in einem Wohnzimmer verbaut ist. Abhängig davon wird man den entsprechenden Effekt auswählen. Des Weiteren ist die Dokumentation wichtig falls Fehler gesucht werden.

In dem Konfigurationsarray werden die Konstanten so verwendet:

```
// LED:           First LED number in the stripe
// |             Input channel. The inputs are read in below using the
// |             digitalRead() function.
// |             On_Min: Minimal number of active rooms. At least two rooms are
// |             |       illuminated.
// |             |       On_Max: Number of maximal active lights.
// |             |       |       rooms: List of room types (see documentation for possible
// |             |       |       types).
// |             |       |       |
House(HOUSE_A,   INCH_HOUSE_A, 2, 5, ROOM_DARK, ROOM_BRIGHT, ROOM_WARM_W, ROOM_TV0, \
ROOM_COL345, ROOM_D_RED, ROOM_COL2)
House(HOUSE_B,   INCH_HOUSE_B, 3, 3, NEON_LIGHT, NEON_LIGHTL, NEON_LIGHTL, NEON_LIGHT)
House(HOUSE_C,   INCH_HOUSE_C, 2, 5, ROOM_TV0, SKIP_ROOM, ROOM_WARM_W, ROOM_COL1, \
ROOM_CHIMNEY)
```

Wenn man jetzt zusätzlich eine Straßenbeleuchtung einbauen will, dann geht das ganz einfach.

Man fügt im „#define“ Bereich die entsprechenden Zeilen ein und ändert die Nummern:

```
#define INCH_S_LIGHT 0
#define INCH_HOUSE_A 1 // Define names for the input channels to be able to change them easily.
#define INCH_HOUSE_B 2 // In this small example this is not necessary, but it's useful in a
#define INCH_HOUSE_C 3 // large configuration.
#define INCH_SHOP_C 4

#define HOUSE_A 0 // Define names for the LED numbers of the houses.
// HOUSE_A 1 // In a real setup the names could be: "RailwayStation", "Town_Hall", "Pub", ...
// HOUSE_A 2 // Each room gets an own name.
// HOUSE_A 3 // Only the first LED numbers are used in the configuration,
// HOUSE_A 4 // But it's a good practice to list the other rooms to because
// HOUSE_A 5 // then the corresponding numbers are increasing without gaps.
// HOUSE_A 6 // This is useful if an additional house is inserted.
#define HOUSE_B 7 // In this case the sequence could easily be checked / updated and
// HOUSE_B 8 // the additional lines could be used for documentation
// HOUSE_B 9
// HOUSE_B 10
#define S_LIGHT 11
// S_LIGHT 12
// S_LIGHT 13
// S_LIGHT 14
// S_LIGHT 15
// S_LIGHT 16
#define HOUSE_C 17
#define HOUSE_C_SHOP 18 // There is a shop in the house which is controlled separately
// HOUSE_C 19
// HOUSE_C 20
// HOUSE_C 21
```

Hier wurde der Eingangskanal für die Straßenbeleuchtung ganz an den Anfang gestellt, die LEDs aber zwischen Haus B und C gesetzt. In diesem Beispiel ist das willkürlich gewählt, in der Realität wird man aber immer wieder Gründe finden warum es genauso gemacht werden muss.

Die zusätzliche Zeile für die Straßenlaternen kann im Konfigurationsarray an einer beliebigen Stelle ergänzt werden:

```
// LED:                               First LED number in the stripe
// |                                   Input channel. The inputs are read in below using the
// |                                   digitalRead() function.
// |                                   On_Min:  Minimal number of active rooms. At least two rooms are
// |                                   |       illuminated.
// |                                   |       On_Max:  Number of maximal active lights.
// |                                   |       |       rooms: List of room types (see documentation for possible
// |                                   |       |       types).
// |                                   |       |       |
House(HOUSE_A,    INCH_HOUSE_A, 2, 5,    ROOM_DARK, ROOM_BRIGHT, ROOM_WARM_W, ROOM_TV0, \
House(HOUSE_B,    INCH_HOUSE_B, 3, 3,    ROOM_COL345, ROOM_D_RED, ROOM_COL2)
House(HOUSE_C,    INCH_HOUSE_C, 2, 5,    NEON_LIGHT, NEON_LIGHTL, NEON_LIGHTL, NEON_LIGHT)
                                                ROOM_TV0, SKIP_ROOM, ROOM_WARM_W, ROOM_COL1, \
                                                ROOM_CHIMNEY)
GasLights(S_LIGHT, INCH_S_LIGHT, ...)    GAS_LIGHT1, GAS_LIGHT2, GAS_LIGHT3D, GAS_LIGHT1, \
                                                GAS_LIGHT2, GAS_LIGHT3)
```

Auf diese Weise ist das Einfügen eines neuen Elements gar nicht so aufwändig.

Noch eleganter geht es, wenn man die Daten in eine Excel Tabelle einträgt wie im nächsten Bild gezeigt.

Achtung Excel verwendet den Schrägstrich am Anfang einer Zelle als Kommando zur Auswahl der Ribbon Menüs. Damit kann man das Programm ohne Maus bedienen. Das ist in diesem Fall hinderlich. Als Abhilfe muss ein einfacher Apostroph am Anfang einer Kommentarzeile eingegeben werden: „// “. Ein Leerzeichen am Anfang erfüllt den gleichen Zweck. Der Apostroph ist in der Tabelle nicht mehr vorhanden. Er wird nur als Steuerzeichen zur Eingabe benutzt.

	A	B	C	D	E	F	G	H	I	J
1	// InCh Namen	InCh Nr	DCC Nr	Beschreibung						
2	#define INCH_HOUSE_A	0 //	200	Define names for the input channels to be able to change them easily.						
3	#define INCH_HOUSE_B	1 //	201	In this small example this is not necessary, but it's useful in a						
4	#define INCH_HOUSE_C	2 //	202	large configuration.						
5	#define INCH_SHOP_C	3 //	203							
6	/									
7	// LED Namen	LED Nr	Beschreibung							
8	#define HOUSE_A	0 //	Define names for the LED numbers of the houses.							
9	// HOUSE_A	1 //	In a real setup the names could be: "RailwayStation", "Town_Hall", "Pub", ...							
10	// HOUSE_A	2 //	Each room gets an own name.							
11	// HOUSE_A	3 //	Only the first LED numbers are used in the configuration,							
12	// HOUSE_A	4 //	But it's a good practice to list the other rooms to because							
13	// HOUSE_A	5 //	then the corresponding numbers are increasing without gaps.							
14	// HOUSE_A	6 //	This is useful if an additional house is inserted.							
15	#define HOUSE_B	7 //	In this case the sequence could easily be checked / updated and							
16	// HOUSE_B	8 //	the additional lines could be used for documentation							
17	// HOUSE_B	9								
18	// HOUSE_B	10								
19	#define S_LIGHT									
20	// S_LIGHT									
21	// S_LIGHT									
22	// S_LIGHT									
23	// S_LIGHT									
24	// S_LIGHT									
25	#define HOUSE_C	11								
26	#define HOUSE_C_SHOP	12 //	The	which is controlled separately						
27	// HOUSE_C	13								
28	// HOUSE_C	14								
29	// HOUSE_C	15								

Nachdem man die zusätzlichen Zeilen eingefügt hat kann man mit der Funktion zum automatischen ausfüllen von Excel die Nummern der LEDs ganz schnell aktualisieren. Dazu markiert man die letzten beiden LED Nummern und klickt doppelt auf das kleine Kästchen des Rahmens. Damit werden dann

automatisch die fehlenden Nummern eingetragen. Diesen Vorgang wiederholt man dann noch mal am Ende des neuen Bereichs, damit auch die folgenden Nummern wieder stimmen.

Für das Einfügen der „#define INCH_S_LIGHT 0“ Zeile geht man genauso vor. Hier kann man zusätzlich noch eine Spalte für die DCC Nummer einfügen.

Die geänderte Tabelle kopiert man dann einfach über die Zwischenablage in das Programm.

	A	B	C	D	E	F	G	H	I	J
1	// InCh Namen	InCh Nr	DCC Nr	Beschreibung						
2	#define INCH_S_LIGHT	0	//	200						
3	#define INCH_HOUSE_A	1	//	201	Define names for the input channels to be able to change them easily.					
4	#define INCH_HOUSE_B	2	//	202	In this small example this is not necessary, but it's useful in a					
5	#define INCH_HOUSE_C	3	//	203	large configuration.					
6	#define INCH_SHOP_C	4	//	204						
7	/									
8	// LED Namen	LED Nr		Beschreibung						
9	#define HOUSE_A	0	//	Define names for the LED numbers of the houses.						
10	// HOUSE_A	1	//	In a real setup the names could be: "RailwayStation", "Town_Hall", "Pub", ...						
11	// HOUSE_A	2	//	Each room gets an own name.						
12	// HOUSE_A	3	//	Only the first LED numbers are used in the configuration,						
13	// HOUSE_A	4	//	But it's a good practice to list the other rooms to because						
14	// HOUSE_A	5	//	then the corresponding numbers are increasing without gaps.						
15	// HOUSE_A	6	//	This is useful if an additional house is inserted.						
16	#define HOUSE_B	7	//	In this case the sequence could easily be checked / updated and						
17	// HOUSE_B	8	//	the additional lines could be used for documentation						
18	// HOUSE_B	9								
19	// HOUSE_B	10								
20	#define S_LIGHT	11								
21	// S_LIGHT	12								
22	// S_LIGHT	13								
23	// S_LIGHT	14								
24	// S_LIGHT	15								
25	// S_LIGHT	16								
26	#define HOUSE_C	17								
27	#define HOUSE_C_SHOP	18	//	There is a shop in the house which is controlled separately						
28	// HOUSE_C	19								
29	// HOUSE_C	20								
30	// HOUSE_C	21								

Auf diese Weise hat man eine übersichtliche Dokumentation und kann Änderungen einfach vornehmen.

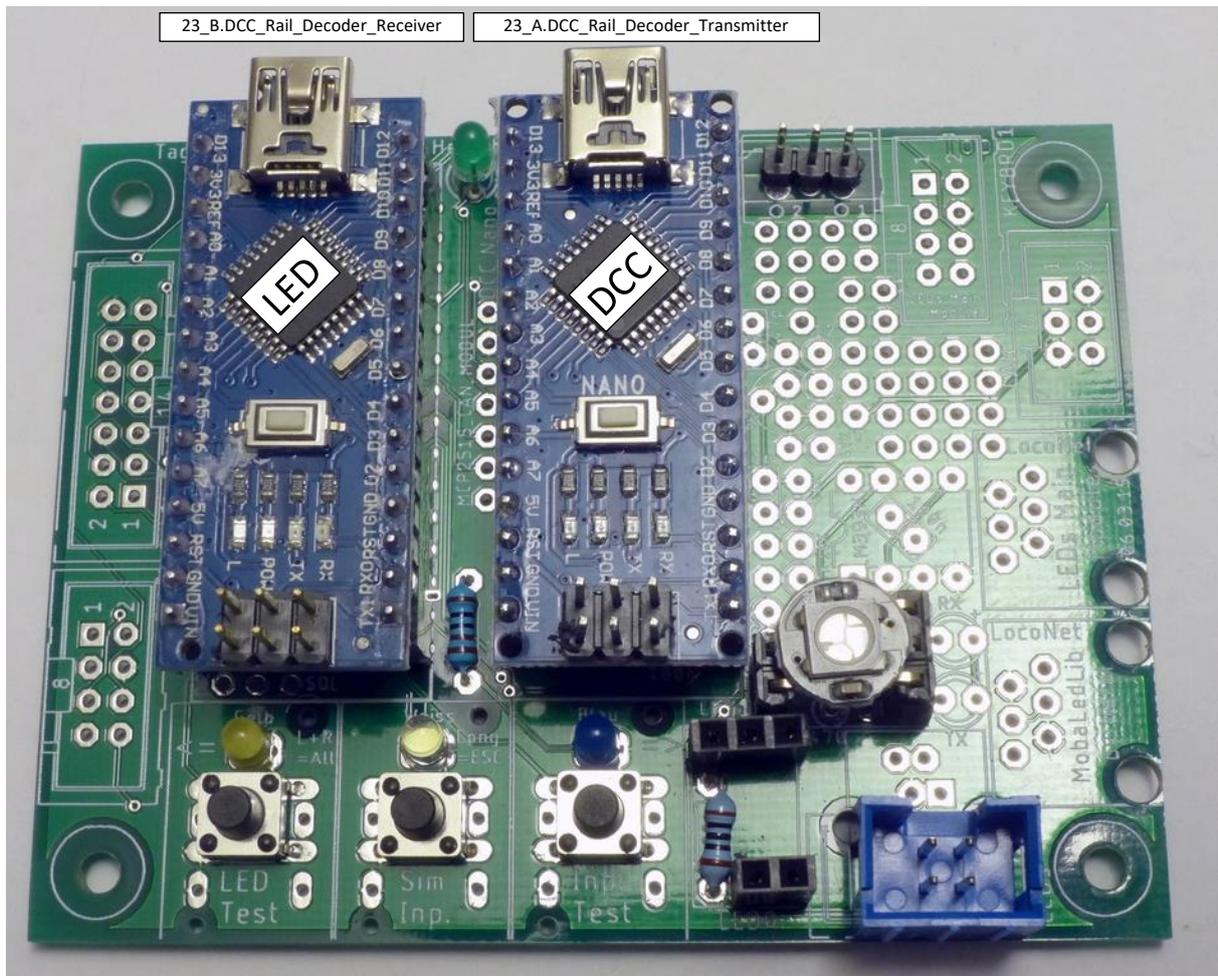
5.1.4 Anpassungen am Programm

In diesem Abschnitt wird beschrieben wie man das bestehende Beispielprogramm „23_B.DCC_Rail_Decoder_Receiver“ an die eigenen Bedürfnisse anpasst.

Im ersten Schritt sollte man das Beispielprogramm unverändert testen. Damit wird sichergestellt, dass die Hardware richtig aufgebaut ist. Das Beispiel verwendet die DCC Adressen 1 bis 30. Das wird sehr wahrscheinlich zu Überlappungen mit bestehenden Geräten führen. Aber das sollte zunächst kein Problem darstellen. Im Gegenteil. Auf diese Weise hat man eine einfache Kontrolle ob das DCC Signal tatsächlich gesendet wird. Wenn Beispielsweise die erste DCC Adresse eine Weiche steuert, dann erkennt man am Umschalten dieser, dass der entsprechende Befehl gesendet wurde.

Von da an muss geprüft werden, ob das Signal auch entsprechend von der MobaLedLib Hardware gelesen wird. Hier können verschiedene Fehler auftreten. Das können fehlende Kabel, Bestückungsfehler der Platine oder falsch aufgespielte Software und leider auch defekte Arduino Hardware sein. Darum werden hier einige Möglichkeiten zur Diagnose aufgezeigt.

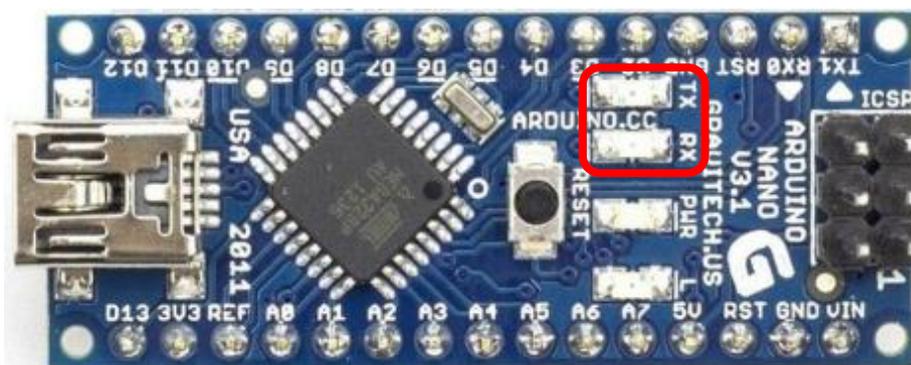
Das Programm „23_A.DCC_Rail_Decoder_Transmitter“ für den DCC Arduino wird auf den rechten Nano hochgeladen. Der LED-Arduino bekommt das Programm „23_B.DCC_Rail_Decoder_Receiver“.



Abhängig von den vorher auf den Mikrocontrollern installierten Programmen kann es Schwierigkeiten beim Hochladen des Programms geben. Das liegt daran, dass diese über die serielle Schnittstelle miteinander kommunizieren. Diese wird aber auch zur Installation des Programmes per USB benutzt. Diese Probleme treten nicht auf, wenn die Arduinos beim ersten Programmieren nicht auf der Platine stecken. Spätere Änderungen am Programm sind unkritisch. Dann wird die Kommunikation unterbrochen, wenn das Programm übertragen wird.

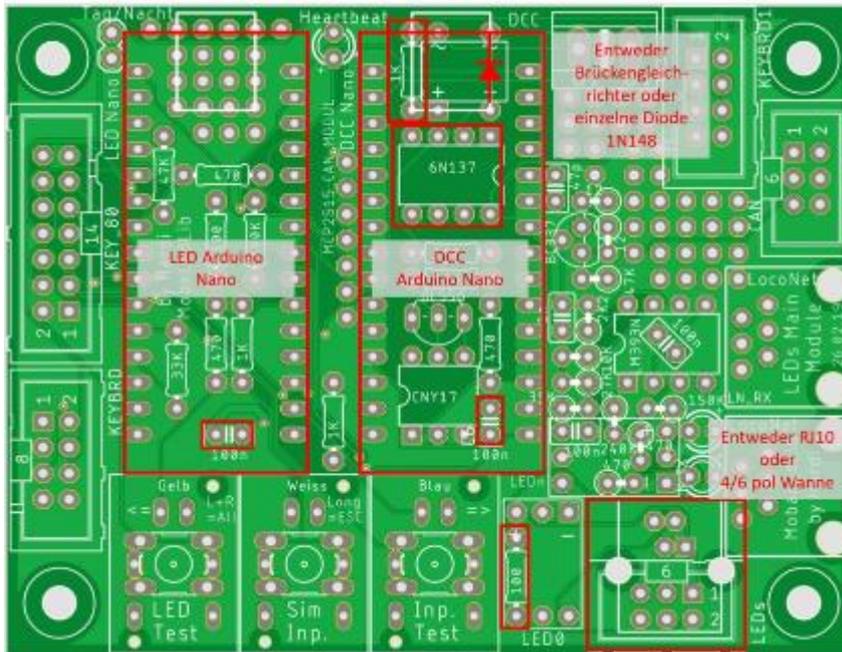
Überprüfung der Kommunikation

Wenn DCC Kommandos empfangen werden, flackert die RS232 Status LED auf beiden Arduinos.



Beim DCC Arduino sollte die TX LED flackern, beim LED Arduino die RX LED.

Wenn das nicht der Fall ist, muss geprüft werden ob das DCC Signal richtig ankommt, die Diode bzw. der Gleichrichter richtig herum eingebaut ist und (das ist mir passiert) der Optokoppler 6N137 eingesteckt ist und natürlich die anderen rot markierten Bauteile.



Zusätzlich können die empfangenen DCC Befehle über den seriellen Monitor der Arduino IDE betrachtet werden. Achtung: Dazu muss die Baudrate auf 115200 Baud eingestellt sein und das USB Kabel an den DCC Nano angeschlossen werden. Der LED Arduino wird von seinem Kollegen mit 5V versorgt.

```
DCC_Rail_Decoder Serial test

DCC_Rail_Decoder_Transmitter Example A
Init Done
@ 1 00 01
@ 1 00 01
@ 1 00 01
@ 1 00 01
@ 1 01 01
@ 1 01 01
@ 1 01 01
@ 1 01 01
```

Wenn man zum Senden der DCC Befehle eine MS2 von Märklin verwendet, dann muss das DCC Protokoll für das Zubehör aktiviert werden. Standardmäßig ist es bei dieser Steuerung nicht eingeschaltet. Bei anderen Steuerungen kann das auch der Fall sein.

Wenn die Kommandos soweit erkannt werden, kann man den LED Arduino betrachten. Wenn dessen RX LED flackert bekommt er Daten. Das könnten aber auch Debug- oder Statusmeldungen vom DCC Arduino sein. Darum ist es hilfreich, wenn man sich auch die seriellen Ausgaben das LED Arduinos anschaut. Das USB Kabel wird entsprechend umgesteckt. Es können aber auch zwei IDEs gleichzeitig mit verschiedenen Schnittstellen (z.B. COM3, COM4) geöffnet werden. Wenn man diese nacheinander anschließt, kann man Verwechslungen vermeiden.

```
DCC_Rail_Decoder_Transmitter Example B
DCCtest
DCC_Rail_Decoder_Transmitter Example A
Init Done
```

Reset am DCC
Arduino gedrückt

Die Ausgabe des LED Arduinos zeigt zusätzlich die Meldungen des DCC Arduinos. Darum sieht man hier auch die Einschaltmeldung des anderen Nanos. Die DCC Kommandos selber werden aber nicht mehr gezeigt. Im Beispiel Programm gibt es verschiedene auskommentierte Debug Zeilen mit denen man sich anschauen kann was empfangen wird.

Zum Testen der Ausgaben empfiehlt es sich, wenn man zunächst nur LEDs und noch keine speziellen Module anschließt. Dazu kann man einen LED Stripe oder ein solches LED Panel benutzen:



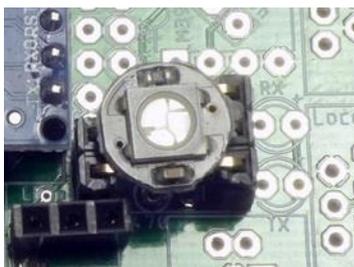
Die Das Beispiel Programm nutzt die ersten LEDs zur Demonstration zweier Signale. Die LEDs 4 bis 6 simulieren ein Einfahrtssignal welches HP0, HP1 und HP2 zeigen kann. Beim Start wird HP0 gezeigt. Es leuchte die rote LED. Das zweite Signal, ein Ausfahrtssignal, belegt die LEDs 11 bis 16. Auch hier sind zu Beginn die roten LEDs aktiv. Das verwirrt etwas, weil ja noch kein DCC Kommando empfangen wurde. Im Programm sind dafür diese beiden Zeilen in der „setup()“ Funktion verantwortlich:

```
MobaLedLib.Set_Input(0, 1); // Set the default value for the entry light signal at power on
MobaLedLib.Set_Input(4, 1); // Set the default value for the departure light signal at power on
```

Achtung: Sie müssen entfernt werden, wenn man das Programm an die eigenen Bedürfnisse anpasst.

Die LEDs 17 bis 23 zeigen wie man zwei Häuser und ein Schweißlicht per DCC schalten kann.

LEDs ab der Nummer 24 dienen der Visualisierung des „InCh[]“ Arrays. Für die Taster existiert jeweils eine rote und eine grüne LED, die dann leuchten solange der DCC-Taster betätigt wird. Die Umschaltfunktionen werden mit einer blauen LED für jeden Kanal überwacht. Im Konfigurationsarray kann dieser Block deaktiviert werden, indem man das „#if 1“ auf 0 setzt.



Achtung: Falls man die Test RGB LED auf der Hauptplatine verwendet, dann verschieben sich die Nummern aller LEDs um eins.

Eigene Anpassungen

Wenn die Kommunikation funktioniert und das Beispiel macht was es soll, dann kann man mit der Anpassung an die eigenen Bedürfnisse beginnen. Dazu ist es sicherlich hilfreich, wenn man sich aufschreibt, was genau gemacht werden soll. Das kann man in der oben vorgeschlagenen Excel Tabelle machen.

Vermutlich wird man den benutzten DCC Bereich anpassen müssen. Dazu passt man die drei Konstanten im Programm an die eigenen Wünsche an:

```
#define DCC_FIRST_LOC_ID      x
#define DCC_FIRST_TOGGLE_ID  (DCC_FIRST_LOC_ID + 10)
#define DCC_LAST_LOC_ID      (DCC_FIRST_LOC_ID + 29)
```

Man kann in „#define“ Zeilen auch Berechnungen benutzen. Der Vorteil dabei ist, dass die Berechnungen während des Kompilierens gemacht werden. Sie kosten also keine Rechenzeit und keinen Speicher auf dem Arduino. Wenn diese Zeilen wie oben ergänzt wurden, dann kann man die eigne erste DCC Adresse einfach bei „x“ eintragen. Und das Beispiel nochmal testen.

Es empfiehlt sich grundsätzlich, dass man immer nur kleine Änderungen beim Arduino Programmieren macht. Dadurch lassen sich Fehler leichter eingrenzen und beheben.

Den Bereich der Taster und Umschalter sollte man erst im nächsten Schritt verändern, ansonsten wird das Beispiel nicht mehr richtig funktionieren. Je nach Komplexität der Änderungen wird man ganz schnell den Überblick verlieren. Am besten speichert man das Programm unter einem neuen Namen ab und kommentiert alle Zeilen im Konfigurationsarray aus. Dann passt man den Start der Umschaltfunktionen und gegebenenfalls die letzte lokale DCC Adresse an. Anschließend fügt man Zeile für Zeile die eigenen Funktionen ein und testet diese jedes Mal. Auf diese Weise wird man schneller zum Ziel kommen als wenn man alles auf einmal ändert. Die „#define“ Zeilen mit den symbolischen Konstanten aus der Excel Tabelle (wie oben gezeigt) können auf einmal eingefügt werden.

5.2 CAN Bus

5.2.1 CAN Message Filter

In diesem Abschnitt wird das Modul „Add_Message_to_Filter.h“ beschrieben...

Aber nicht mehr in dieser Version... Es liest ja doch keiner...

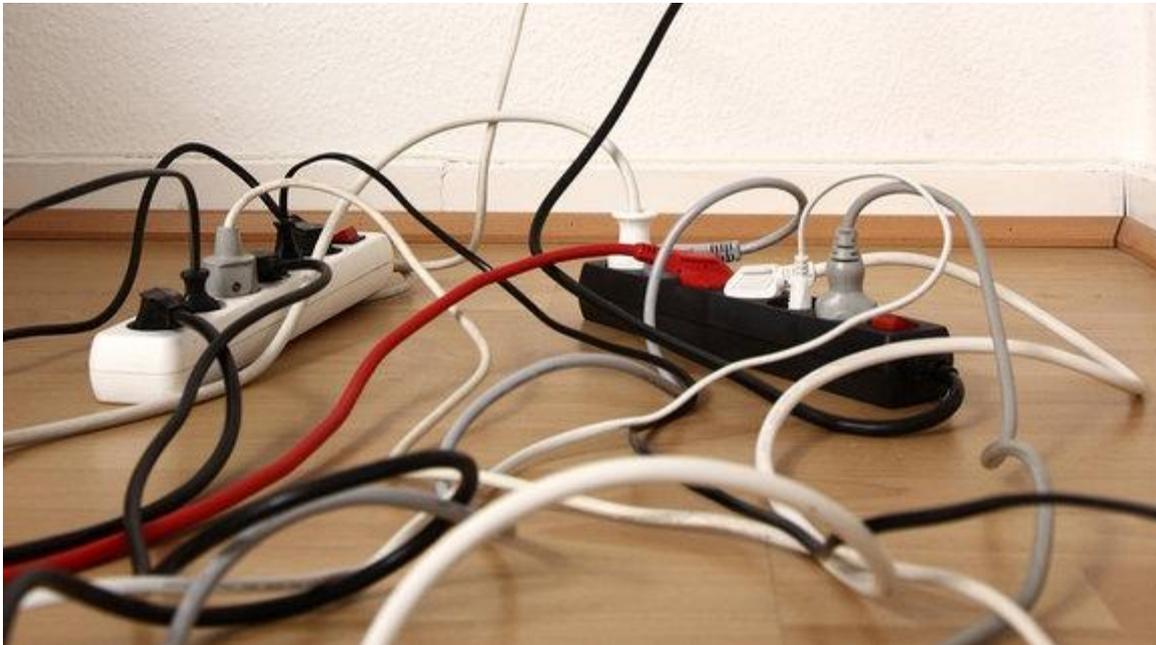
Wenn Du mehr lesen willst, dann ermutige mich mit einer Mail an: MobaLedLib@gmx.de

6 Anschlusskonzept mit Verteilermodulen

Der größte Vorteil der WS281x Module ist die einfache Verkabelung. Durch die Benutzung von 4- oder 6-poligen Steckern, die einfach in Verteilerleisten gesteckt werden können, ist die Beleuchtung einer komplexen Anlage sehr einfach. In diesem Abschnitt wird das näher beschrieben.

Normalerweise werden WS281x LEDs in einer Kette angeordnet (LED Stripe). Für den Einsatz auf der Modelleisenbahn ist eine sternförmige Verkabelung aber einfacher. Auf der Anlage gibt es verschiedene Zentren (z.B. Städte) an denen mehrere LEDs platziert sind und dann wieder Bereiche in denen keine Verbraucher benötigt werden. Es ist außerdem wichtig, dass man einfach zusätzliche Teilnehmer anschließen kann. Durch einen einfachen Trick kann man aus der Kette eine einfach zu handhabende flexible Struktur erzeugen. Alle Verbraucher werden einfach in eine Verteilerleiste eingesteckt. Wenn zusätzlicher Geräte angeschlossen werden sollen, dann fügt man einen weiteren Verteiler ein.

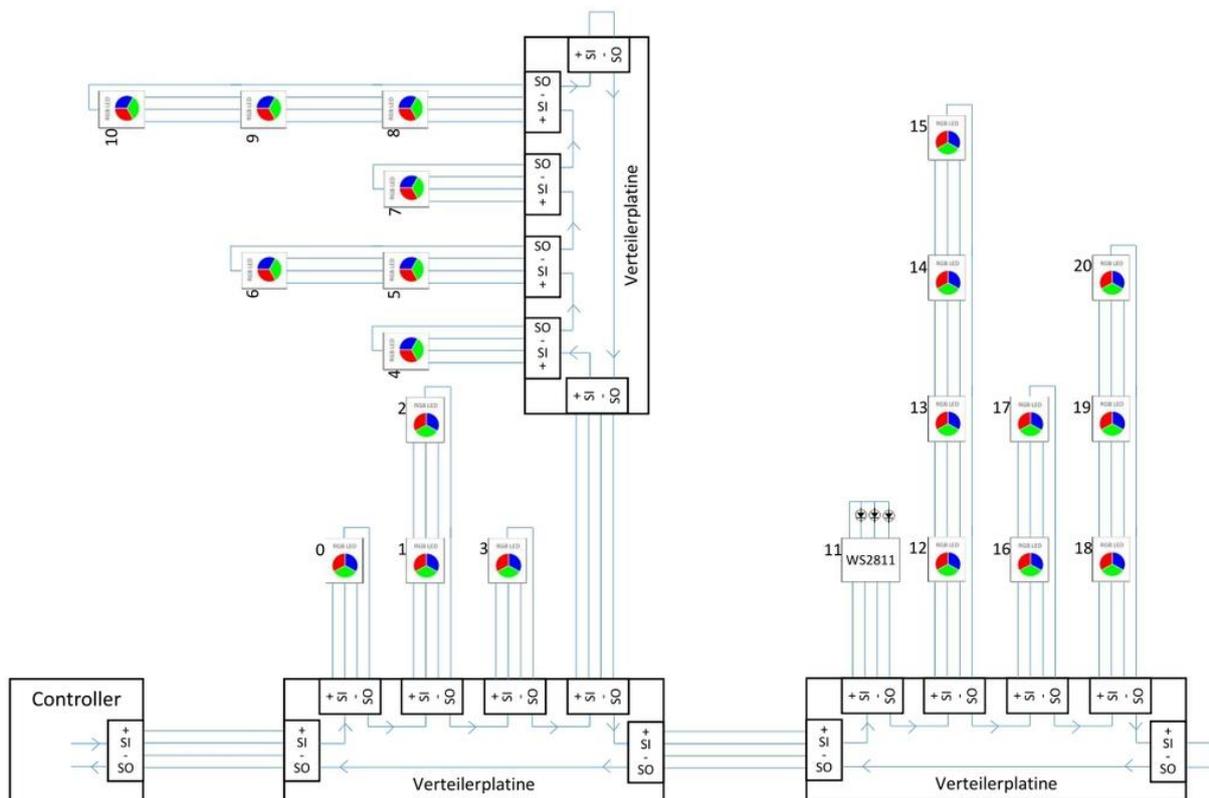
Das ist so einfach wie man das mit Mehrfachsteckdosen im Haushalt macht. Wenn nicht genügend Steckdosen zur Verfügung sind, kann man einfach mit einer weiteren Mehrfachsteckdose zusätzliche Anschlussmöglichkeiten schaffen.



Achtung: Solche Erweiterungen können brandgefährlich sein! Die Maximale Strombelastbarkeit muss in jedem Fall beachtet werden!

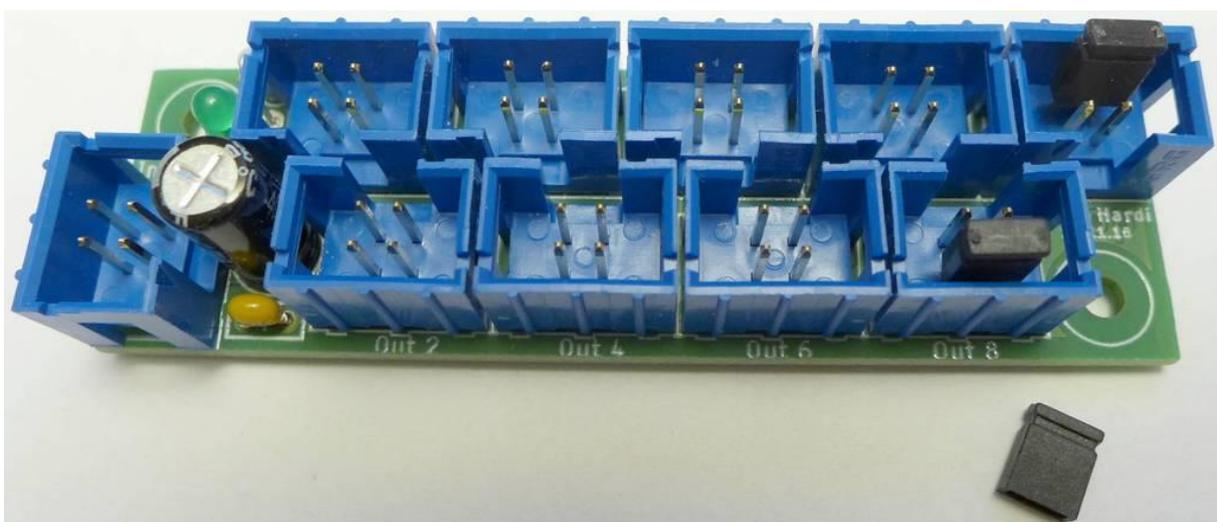
Mit den hier vorgestellten Verteilerplatinen kann man die Beleuchtung auf der Modelleisenbahn ähnlich einfach gestalten. Aber auch hier muss die Stromstärke berücksichtigt werden und es müssen gegebenenfalls zusätzlicher Leitungen mit größerem Querschnitt zwischen den Verteilern eingesetzt werden.

Doch zunächst mal zum Prinzip. Anstelle der sonst bei WS281x Systemen üblichen 3 Leitungen werden bei hier 4 Leitungen benutzt: +5V, Signal In, Masse, Signal Out. Damit können die LEDs einfach in einen Verteiler gesteckt werden die Ausgangsleitung der vorangegangenen LED wird im Verteiler mit dem Eingang der nächsten LED verbunden. So kann man einen beliebigen mechanischen Aufbau umsetzen bei dem die Elemente elektrisch in einer Kette miteinander verbunden sind.

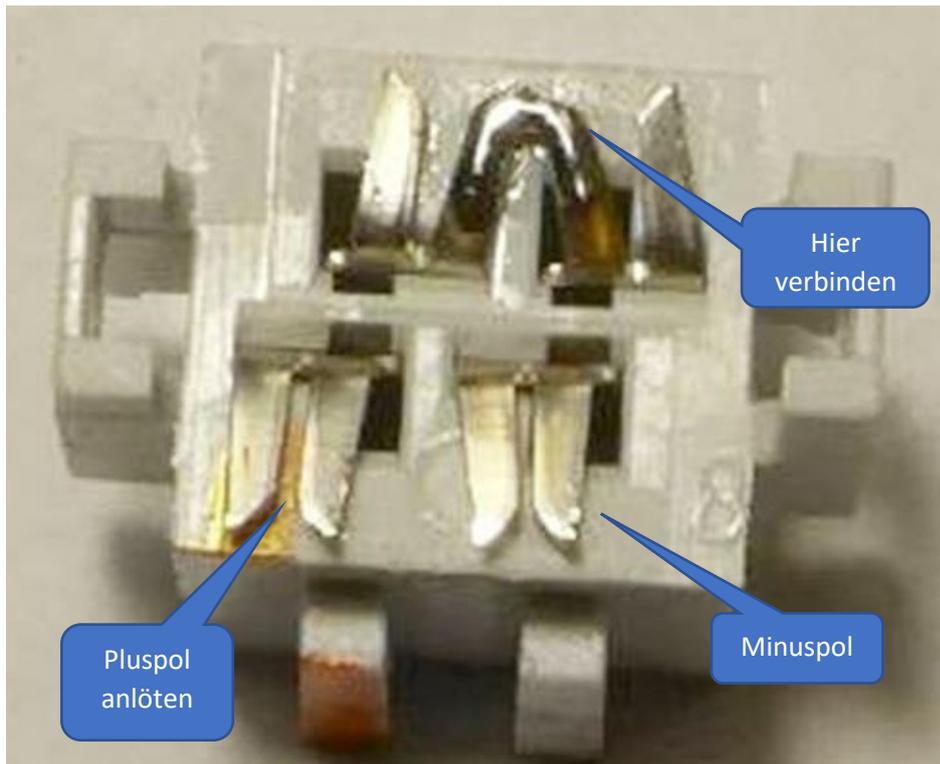


Auf dem Bild sind drei Verteiler abgebildet. In den einzelnen Anschlüssen können Verbraucher mit einer oder mehreren LEDs angesteckt werden. Das kann ein Haus mit mehreren Zimmern, eine Ampel, ein Sound Modul oder eine weitere Verteilerplatine sein.

Man erkennt auf dem Bild sehr schön, dass die Daten immer von einer LED zur nächsten weitergereicht werden. Nicht benutzte Anschlüsse müssen überbrückt werden. Dazu können Jumper eingesetzt werden, die früher bei Computer Platinen üblich waren. Im nächsten Bild sieht man eine solche Verteilerplatine mit zwei überbrückten Anschlüssen und einem unbenutzten Jumper darunter. Überbrückt werden die Pins 2 und 4.



Da die 4-poligen Stecker schwerer zu beschaffen und teurer sind als die 6-polige Variante, wurden später Verteiler entwickelt, die sowohl mit 4-poligen als auch mit 6-poligen Wannensteckern bestückt werden können. Bei der Verwendung von 6-poligen Verbindungen können die 4 Leitungen



Aus dem Befestigungsbügel wurde etwas herausgefeilt so das man diesen wieder schließen kann:



Achtung: Darauf achten, dass die Versorgungsspannungskabel nicht die Datenanschlüsse berühren.

7 Details zur Pattern Funktion

Die Pattern Funktion ist unglaublich leistungsfähig. Mit ihr können die komplexesten Animationen sehr einfach konfiguriert werden. Damit können nicht nur Lichter sondern auch Bewegungen und Sounds gesteuert werden.

Viele der vorgestellten Makros beruhen auf der Pattern Funktion. Das populärste Beispiel ist die Ampelsteuerung wie sie in Abschnitt „2.4.14 RGB_AmpelX(LED,InCh)“ kurz vorgestellt wurde. Bei einer Ampelanlage leuchtet einem sofort ein, dass hier die Lichter nach einem bestimmten Muster angesteuert werden müssen (Rot, Rot + Gelb, Grün, Gelb, Rot).

Das gilt auch für Steuerung eines Blinkers. Auch hier wird in regelmäßigen Abständen eine LED An- und Ausgeschaltet.

Doch selbst die „Const()“ Funktion aus Abschnitt „2.3.1 Const(LED,Cx,InCh,Val0, Val1)“ wird mit der Pattern Funktion umgesetzt. Hier wird die LED nur einmal eingeschaltet und nicht wieder ausgeschaltet.

Das zeigt, dass solche Animationen zyklisch ablaufen können wie bei der „Ampel“ oder nur einmalig eine Funktion ausführen und sich dann beenden wie bei der „Const()“ Funktion. Ein weiteres Beispiel für so eine einmalige Aktion ist die „Button()“ Funktion („2.4.1 Button(LED,Cx,InCh,Duration,Val0, Val1)“). Hier wird der Ausgang für eine bestimmte Zeit aktiviert und danach wieder abgeschaltet. Die Zeit kann bis zu 17 Minuten lang sein.

Die Pattern Funktion kennt noch einige weitere Betriebsarten auf die in den folgenden Abschnitten eingegangen wird.

Mit dem Konzept der Pattern Funktion kann eine sehr große Anzahl von Funktionen implementiert werden ohne dass dazu zusätzlicher Programmcode erzeugt werden muss! Das ist bei den beschränkten Ressourcen eines Arduinos enorm wichtig. Für eine neue Funktionalität muss nur eine Konfiguration erstellt werden, welche in der Pattern Funktion beschreibt, was sie machen soll. Diese Konfiguration bestimmt dann ob ein Blinker oder die Ampelanlage einer mehrspurigen Kreuzung erzeugt wird.

Zur Erstellung solcher Konfigurationen kann das Excel Programm Pattern_Configurator eingesetzt werden. Hier muss man sich nicht um die Details der Funktion kümmern. Es müssen nur einige Tabellen ausgefüllt werden mit denen das Verhalten der Pattern Funktion konfiguriert wird. Im Folgenden wird beschrieben was man damit machen kann.

7.1 Pattern_Configurator

Mit diesem Abschnitt wird erklärt, wie das Excel Programm „Pattern_Configurator.xlsm“ verwendet wird. Es befindet sich im Verzeichnis „Extras“ der Bibliothek. Bei einem Windows Rechner wird die Bibliothek unter „Eigene Dokumente\Arduino\libraries\MobaLedLib“ installiert. Wenn man die Bibliothek direkt von GitHub heruntergeladen hat, dann wird noch „-master“ an den Namen angehängt. Es existiert auch eine Version des Programms für das kostenlose „Libre Office“ welches auch unter Linux verwendet werden kann. Diese Version unterstützt aber nicht alle Features des Excel Programms. Hier wird nur auf die Excel Version eingegangen.

In den folgenden Seiten werden die einzelnen Möglichkeiten des Programms Stück für Stück beschrieben. Dabei wird mit den gebräuchlichsten Einstellungen begonnen.

7.1.1 Einfache Beispiele

Nach dem Start des Programms wird dieser Bildschirm angezeigt (Die tatsächliche Anzeige kann sich bei neueren Versionen ändern):

Das erste Blatt zeigt die Konfiguration für einen Wechselblinker. In der Tabelle unten sind zwei LEDs gelistet welche abwechselnd an sind:

LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	8	9	10	11	12
1	LED 1	x											
2	LED 2		x										

Immer dann, wenn in der Tabelle ein x eingetragen ist leuchtet die entsprechende LED.

In der Tabelle darüber wird bestimmt wie lange die entsprechende Spalte angezeigt wird. In dem einfachen Beispiel ist „1 Sec“ eingetragen:

Dauer	1 Sec												

Wenn die Zeiten mehrerer Spalte gleich lang sind, dann wird die Dauer nur einmal eingetragen. Das Programm wiederholt die vorangegangenen Zeiten für alle folgenden Spalten. Das reduziert den Speicherbedarf im Arduino. Zwischen den beiden Tabellen findet man die Angabe des aktuell benötigten Flash Speichers: „Flash usage: 14 Bytes“. Wenn man die Dauer für die zweite Spalte ebenfalls einträgt erhöht sich der Flash Bedarf unnötigerweise um zwei Bytes auf 16.

Dauer	1 Sec	1 Sec											

FLASH usage: 16 Bytes

Nur dann, wenn die Spalten unterschiedlich lange aktiv sein sollen, werden die Zeiten entsprechend eingetragen. Ein Beispiel dafür ist die Befuerung eines Windrades. Das Warnlicht ist eine Sekunde lang an, dann eine halbe Sekunde aus und anschließend wieder eine Sekunde lang an. Darauf folgt eine Pause von 1.5 Sekunden. (Siehe https://www.windparkwaldhausen.de/contentbeitrag-170-84-kennzeichnung_befuerung_von_windkraftanlagen_.html)

Mit dem Pattern_Configurator kann man das so umsetzen:

Dauer	1 Sec	0.5 Sec	1 Sec	1.5 Sec									
FLASH usage:	20 Bytes												
LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	8	9	10	11	12
1	LED 1	x		x	-								

Für dieses Beispiel benötigen wir nur eine LED. Darum wurde im oberen Teil des Programms die „Anzahl der Kanäle“ auf 1 gesetzt:

Anzahl der Ausgabe Kanäle: 1

Damit wird die Größe der Tabelle unten automatisch angepasst.

Die LED wird mit dieser einfachen Tabelle genau das machen was gefordert ist. Sie ist eine Sekunde lang an, dann kurz aus und danach noch mal eine Sekunde an. Anschließend kommt eine Pause von 1.5 Sekunden bevor der Zyklus von vorne beginnt. Das „X“ in der Tabelle markiert die Spalten in denen die LED an sein soll. Wenn nichts eingetragen ist, dann leuchtet die LED nicht. Das gilt auch für ein Minus Zeichen. Dieses muss am Ende eingefügt werden, wenn keine weitere Spalte kommt in der die LED an sein soll damit das Programm weiß wie lange der Zyklus sein soll.

7.1.2 Verwendung im Arduino Programm

Mit dem Pattern_Configurator erstellt man einen Befehl für das Arduino Programm den man in das Konfigurationsarray des Arduino Programms einträgt. Für das „Windrad“ Beispiel generiert das Programm diese Ergebniszeile:

Ergebnis: PatternT4(0,128,SI_1,1,0,128,0,0,1 Sec,0.5 Sec,1 Sec,1.5 Sec,5)

Diese Zeile kopiert man Beispielsweise in das „House()“ Beispiel:

```
//*****
// *** Configuration array which defines the behavior of the LEDs ***
MobaLedLib_Configuration()
{
  // LED: First LED number in the stripe
  // | InCh: Input channel. Here the special input 1 is used which is
  // | | always on
  // | | On_Min: Minimal number of active rooms. At least two rooms are
  // | | | illuminated.
  // | | | On_Max: Number of maximal active lights.
  // | | | Rooms: List of room types (see documentation for possible
  // | | | | types).
  // | | | |
  House(0, SI_1, 2, 5, ROOM_DARK, ROOM_BRIGHT, ROOM_WARM_W, ROOM_TV0, NEON_LIGHT,
  ROOM_D_RED, ROOM_COL2) // House with 7 rooms

  PatternT4(0,128,SI_1,1,0,128,0,0,1 Sec,0.5 Sec,1 Sec,1.5 Sec,5) // Windrad

  EndCfg // End of the configuration
};
//*****
```

Das wird aber zunächst nicht so richtig funktionieren, weil jetzt die LED 0 von zwei verschiedenen Zeilen angesteuert wird. Aus der Sicht des „House()“ Makros beleuchten die LEDs 0 bis 6 die verschiedenen „Zimmer“. Diese werden zufällig An- und Ausgeschaltet zur Simulation eines „Belebten Hauses“. Gleichzeitig wird aber die „Windrad“ Zeile die erste (Rote) LED im ersten Zimmer als Warnlicht ansteuern. Es wird zu Überlagerungen der beiden Effekte kommen. Solange das erste Zimmer „unbenutzt“ ist, wird man diese Überlagerung nicht bemerken. Die erste LED wird wie gewünscht Rot Blinken. Sobald aber das Licht im ersten Raum angeschaltet wird ergeben sich falsche Farben (Hellblau, wenn die Windrad LED aus ist und Weis, wenn sie an ist).

Eine LED soll normalerweise nicht von verschiedenen Zeilen im Programm angesteuert werden. Darum kann man im Pattern_Configurator angeben, welche LED Nummer von einer Konfiguration

angesteuert werden soll. Hier ist die „Erste RGB LED“ bereits angepasst:

Erste RGB LED: 7
Startkanal der RGB LED: 0

Damit kommt es jetzt nicht mehr zu einer Überlagerung der Ausgänge. Das Ergebnis Makro passt sich automatisch an:

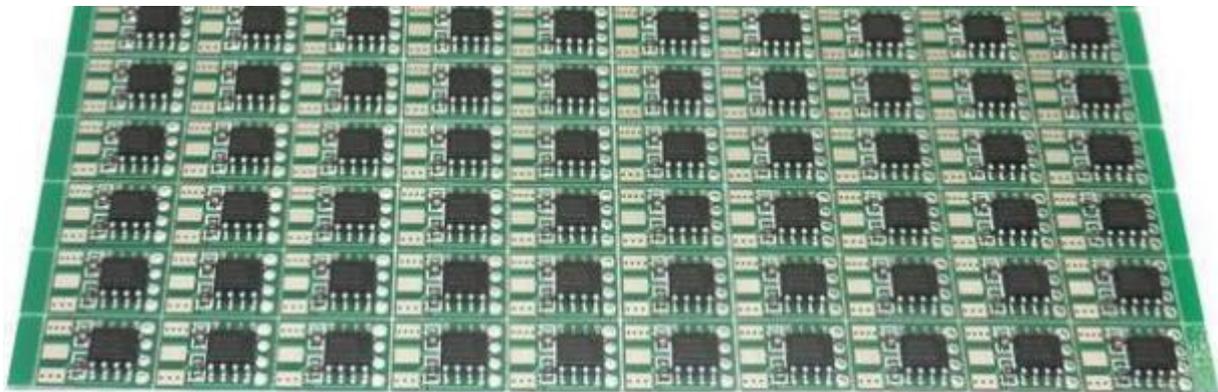
Ergebnis: `PatternT4(7,128,SI_1,1,0,128,0,0,1 Sec,0.5 Sec,1 Sec,1.5 Sec,5)`

Die erste Zahl der „PatternT4() Zeile hat sich von 0 auf 7 geändert. Wenn man diese Änderung in das Programm übernimmt, dann sollte die LED wie in diesem Video gezeigt blinken.

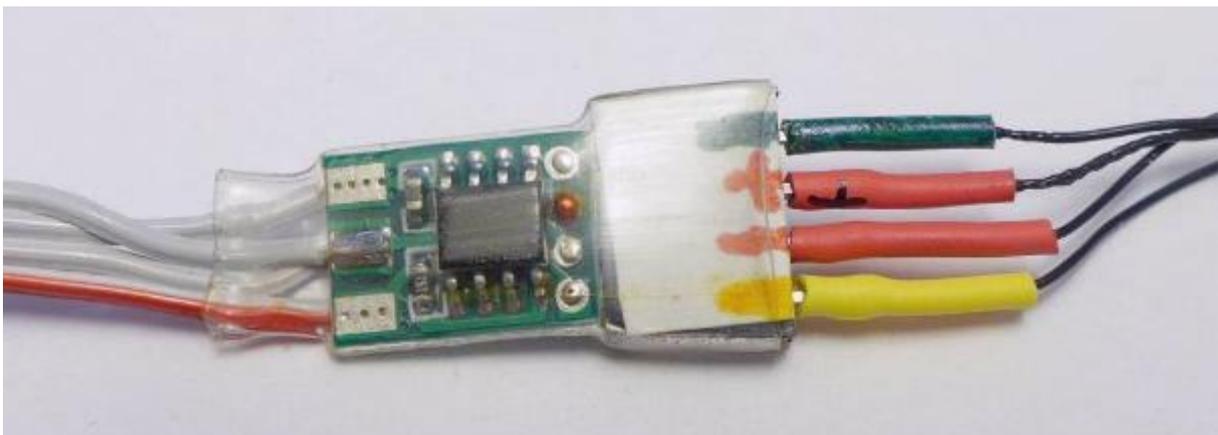
<https://vimeo.com/310209169>

7.1.3 Einzelne LED Kanäle:

Das Warnlicht in einem Windrad wird man vermutlich nicht mit einer RGB LED machen. Hier reicht eine einfache rote LED. Zur Ansteuerung dieser wird ein WS2811 Modul eingesetzt. Diese bekommt man für wenige Euros in China. Dabei sind 100 Stück auf einer Platine die sich wie eine Schokoladentafel auseinanderbrechen lassen.



Mit so einem Modul können drei einzelne LEDs angesteuert werden. Hier ist ein Bild eines solchen Modules mit einem auf der rechten Seite angelöteten Stecker in einem durchsichtigen Schrumpfschlauch:



Im Pattern_Configurator kann man bei der Verwendung eines solchen Moduls angeben, an welchem Ausgang die LED angeschlossen ist. Das trägt man in dem Feld „Startkanal der RGB LED:“ ein:

Erste RGB LED: 7
Startkanal der RGB LED: 0

Die Nummer 0 entspricht dem Roten Kanal. Im Bild oben, dem dritten Anschluss. Der Startkanal 1 entspricht der Grünen LED (erster Anschluss im Bild oben) und der Kanal 2 ist der letzte Anschluss im

Bild oben. Normalerweise entspricht er der Farbe Blau. Hier wurde der Anschluss Gelb markiert, weil damit eine Ampel oder ein Lichtsignal angesteuert wird. Der gemeinsame Pluspol ist der zweite Pin.

Achtung es gibt verschiedene Arten der WS2811 Module, die unterschiedliche Anschlussbelegungen und Spannungen haben. Für die meisten Anwendungen wird man 5V Module einsetzen. Von diesen gibt es zwei verschiedene Anschlussvarianten. Bei der einen Variante ist der Plus Pol wie im Bild oben auf der unteren Seite. Bei dem anderen Typ ist er oben! Auch die Anschlüsse der LEDs unterscheiden sich. Und zur großen Verwirrung stimmt die Beschriftung der LED Farben nicht. Das liegt aber am verwendeten Treiber. Wenn der „NEOPIXEL“ Treiber verwendet wird, dann ist die Beschriftung vom Blau und Grün vertauscht. Dieser Treiber passt aber zu den WS2812 und WS2812B Modulen, die den Chip in der LED integriert haben. Darum wird in allen Beispielen dieser Treiber benutzt.

7.1.4 Eigenes Makro

Wenn man mehrere Windräder auf der Anlage einsetzen will, dann wird man nicht jedes Mal den Pattern_Configurator verwenden wollen damit man die Nummer der LED und des benutzten Kanals anpassen kann. Dafür stellt das Programm zwei weitere Makro Zeilen bereit:

```
Makro Name: Windrad
Makro: #define Windrad(LED,InNr)           PatternT4(LED,128,InNr,1,0,128,0,0,1 Sec,0.5 Sec,1 Sec,1.5 Sec,5)
      #define Windrad_StCh(LED,StCh,InNr) PatternT4(LED,StCh+128,InNr,1,0,128,0,0,1 Sec,0.5 Sec,1 Sec,1.5 Sec,5)
```

Zunächst sollte der im Feld „Makro Name:“ eingetragene Name eine aussagekräftige Beschreibung der Funktion des Makros enthalten. Dann kann man die graue Zeile in das eigene Arduino Programm kopieren. Dabei handelt es sich um ein Makro welches mehrfach mit unterschiedlichen Parametern benutzt werden kann. Die Makros werden **vor** dem Konfigurationsarray eingetragen und können dann in der Konfiguration mehrfach benutzt werden. Das kann dann so aussehen (Der „\“ kann in C zur Fortsetzung einer Zeile in der nächsten Zeile verwendet werden. Er wurde hier eingefügt damit das Makro besser auf die Seite passt.):

```

//*****
// Makro Definitionen:
#define Windrad_StCh(LED,StCh,InNr) \
    PatternT4(LED,StCh+128,InNr,1,0,128,0,0,1 Sec,0.5 Sec,1 Sec,1.5 Sec,5)

//*****
// *** Configuration array which defines the behavior of the LEDs ***
MobaLedLib_Configuration()
{
  // LED: WS2811 number in the stripe
  // | StCh: Channel of the WS2811
  // | | InCh: Input channel (SI_1 = allways on)
  // | | |
  Windrad(6,0,SI_1)
  Windrad(6,1,SI_1)
  Windrad(6,2,SI_1)
  Windrad(7,0,SI_1)

  EndCfg // End of the configuration
};
//*****

```

Die Verwendung von Makros macht die Konfiguration auch gleich viel übersichtlicher, insbesondere wenn man noch zusätzliche Kommentare einfügt.

Für das Beispiel wurde nicht die „Grüne“ Zeile benutzt, weil hier der Kanal des WS2811 Moduls (StCh) nicht als Parameter vorhanden ist. Der dritte Parameter „InCh“ wird hier nicht benutzt. Er beschreibt die Nummer der Eingangsvariable mit der das Leuchtfeuer eingeschaltet werden könnte. Hier soll es aber immer an sein. Darum wird in der Konfiguration die Konstante „SI_1“ verwendet wodurch das Warnlicht immer aktiviert ist. Wenn alle Windräder permanent an sein sollen, dann könnte man die Makro Zeile auch so anpassen:

```
#define Windrad_StCh(LED,StCh) \  
    PatternT4(LED,StCh+128,SI_1,1,0,128,0,0,1 Sec,0.5 Sec,1 Sec,1.5 Sec,5)
```

Hier ist der Parameter „InNr“ weggelassen worden und in der „PatternT4“ Zeile durch „SI_1“ ersetzt worden. Im Konfigurationsarray wird dann der Parameter „SI_1“ ebenfalls weggelassen.

Das zeigt wie flexibel man mit Makros umgehen kann.

7.1.5 Beispiel: Ampel

Als nächstes Beispiel soll eine Ampel vorgestellt werden. Im Pattern_Configurator existieren mehrere Beispielseiten mit denen eine Ampel konfiguriert wird. Hier soll die Seite „RGB_AmpelX“ betrachtet werden. Sie wird angezeigt, indem man auf den entsprechenden Reiter klickt:



Es ist ein Beispiel für eine einfache Ampelanlage an einer Kreuzung. Es gibt 4 Ampeln von denen jeweils die zwei gegenüberliegenden das gleiche Bild zeigen. Darum wird hier nur auf die zwei unterschiedliche Ampeln eingegangen. Im folgenden Bild wird die gesamte Konfigurationsseite gezeigt.

Das Beispiel soll mit mehreren RGB LEDs getestet werden, weil davon ausgegangen wird, dass nicht jeder eine passende Ampel zur Verfügung hat. Für den Test werden 6 RGB LEDs eines LED Streifens benötigt. Eine „richtige“ Ampel mit einzelnen LEDs würde, wie oben im Abschnitt „7.1.3 Einzelne LED Kanäle:“ gezeigt, über WS2811 Module angeschlossen werden. Das Beispiel „AmpelX“ enthält die Einstellungen hierfür.

Ver.: 0.7 17.05.19

Erste RGB LED: 0
 Startkanal der RGB LED: 0
 Schalter Nummer: SI_1
 Anzahl der Ausgabe Kanäle: 18
 Bits pro Wert: 1 => 2 Helligkeitsstufen (0..1)
 Wert Min: 0
 Wert Max: 255
 Wert ausgeschaltet: 0
 Mode: PF_NO_SWITCH_
 Analoges Überblenden: 0
 Goto Mode: 0
 Grafische Anzeige:

Neues Blatt

Aktualisieren

by Hardi Dieses Beispiel zeigt die Konfiguration einer Ampelanlage bestehend aus zwei unterschiedlichen Ampeln für jede Fahrtrichtung (Horizontal/Vertikal) eines Ampelstrahls. Für jede Fahrtrichtung (Horizontal/Vertikal) einer Ampel ist gleich geschaltet. Für dieses Beispiel sind die Zeiten für die Ampelstrahler weggelassen. Sie können durch kopieren des Ergebnisses in den Makro definiert werden.
 Das Beispiel benutzt die LEDs eines RGB LED Strahlers. Für eine spezielle Hardware benötigt.
 Das folgende Beispiel zeigt wie das aussieht wenn es durch einen Mikrocontroller angesteuert werden kann.

Ergebnis: `PatternT4(0,0,SI_1,18,0,255,0,PF_NO_SWITCH_OFF,2 Sec,1 Sec,10 Sec,3 Sec,1,2,100,8,0,40,0,134,`

Makro Name: `RGB_AmpelX`

Makro: `#define RGB_AmpelX(LED,InNr) PatternT4(LED,0,InNr,18,0,255,0,PF_NO_SWITCH_OFF,2 Sec,1 Sec,10 Sec,3 Sec,1,2,100,8,0,40,0,134,`
`#define RGB_AmpelX_StCh(LED,StCh,InNr) PatternT4(LED,StCh+0,InNr,18,0,255,0,PF_NO_SWITCH_OFF,2 Sec,1 Sec,10 Sec,3 Sec,1,2,100,8,0,40,0,134,`

Wenn gleiche Zeiten verwendet werden, dann sollten nur die ersten Zeiten eingetragen werden. Bei leerer Spalte werden die Zeiten der vorherigen Spalte verwendet.

Dauer	2 Sec	1 Sec	10 Sec	3 Sec	2 Sec	1 Sec	10 Sec	3 Sec						
FLASH usage:	37 Bytes													
LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Rot 1	x	x			x	x	x	x					
2														
3														
4	Gelb 1		x		x									
5	Gelb 1		x		x									
6														
7														
8	Grün 1			x										
9														
10	Rot 2	x	x	x	x	x	x							
11														
12														
13	Gelb 2						x		x					
14	Gelb 2						x		x					
15														
16														
17	Grün 2							x						
18														

BEREIT

Die Tabelle zeigt wann und wie lange die einzelnen Ampelfarben leuchten sollen. Die Zeiten sind dabei willkürlich gewählt. (Ich weiß immer noch nicht wie man bei einer Modelleisenbahn die Zeiten maßstäblich anpasst. Ein Modellbahn Tag dauert nur wenige Minuten. Bei Geschwindigkeiten skaliert man meistens nur die Strecke, Züge brauchen von einem zum anderen Bahnhof nur wenige Sekunden...).

Die LED Nummern 1 bis 9 entsprechen der ersten Ampel, die anderen Zeilen stehen für die zweite Ampel.

1. Zu Beginn zeigen beide Ampeln für die Dauer von zwei Sekunden Rot.
2. Danach springt die erste Ampel auf Rot+Gelb. Diese Phase dauert eine Sekunde.
3. In dem dritten Abschnitt zeigt die Ampel 1 Grün. Ampel 2 ist immer noch Rot. Die Fahrzeuge der einen Richtung dürfen 10 Sekunden lang fahren.
4. In der nächsten Spalte wird die erste Ampel Gelb. Dazu wird die Rote und Grüne LED der entsprechenden RGB LED aktiviert womit die Farbe Gelb entsteht. Hier haben die Fahrer drei Sekunden Zeit noch schnell über die Kreuzung zu fahren.

- 5. Jetzt sind wieder beide Ampeln Rot.
Damit wiederholt sich das Spiel mit der zweiten Ampel während die erste Ampel Rot ist (Spalten 5-8).

In der „Dauer“ Tabelle werden die Zeiten für die zweite Ampel Grau dargestellt. Hier wurde nur ein Verweis auf die Zeiten der ersten Ampel eingetragen. Für diesen Verweis wird eine Excel Formel verwendet. Dazu gibt man ein Gleichheitszeichen ein und klickt dann in die Zelle auf die verwiesen werden soll. Während der Eingabe wird das so dargestellt:

Dauer	2 Sec	1 Sec	10 Sec	3 Sec	=E23	1 Sec	10 Sec	3 Sec			
-------	-------	-------	--------	-------	------	-------	--------	-------	--	--	--

„E23“ ist die Adresse der ersten „Dauer“. Excel fügt diese automatisch ein. Wenn die Eingabe mit „Enter“ abgeschlossen wird zeigt die Zelle den gleichen Wert wie die Zelle auf die verwiesen wird. Optisch zeigt das Programm diesen Verweis durch eine graue Schrift an. Diese Zeiten belegen keinen Speicher im Arduino. Diese Verknüpfungen dienen nur der Dokumentation. Sie müssen nicht eingegeben werden. Wenn man die Einträge löscht wird genau so viel Speicher belegt:

Dauer	2 Sec	1 Sec	10 Sec	3 Sec							
FLASH usage:		37 Bytes									

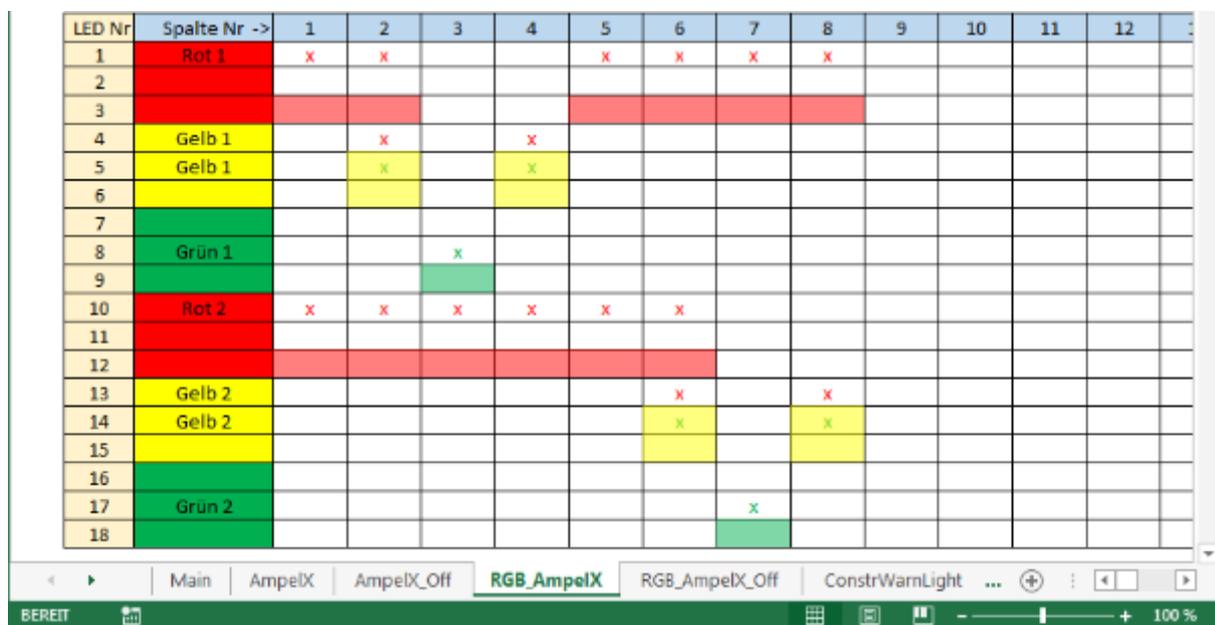
Achtung: Wenn man aber die gleichen Zeiten noch mal eintippt, dann wird mehr Speicher belegt:

Dauer	2 Sec	1 Sec	10 Sec	3 Sec	2 Sec	1 Sec	10 Sec	3 Sec			
FLASH usage:		45 Bytes									

Das sind zwar nur 8 Bytes, aber bei einer großen Konfiguration mit mehreren Ampeln kann das eine Rolle spielen. Ein Arduino hat ja nur 32 KByte Flash.

7.1.6 Grafische Anzeige

Wenn man im Feld „Grafische Anzeige“ eine 1 einträgt: Grafische Anzeige: 1 dann werden die Helligkeiten der LEDs mit gefüllten Flächen visualisiert:



Damit kann man die Situation schneller erfassen. Zur Füllung wird die Farbe in der Spalte ganz links verwendet. Hier allerdings etwas blasser, weil die Flächen transparent dargestellt sind damit man die darunter liegenden Einträge noch sehen kann.

Wenn die drei Kanäle einer RGB LED mit der gleichen Hintergrundfarbe gefüllt werden, dann werden sie als Einheit betrachtet und erhalten eine gemeinsame Füllung. Das zeigt, dass die Helligkeit des gelben Ampellichts hier heller ist als die rote und grüne Farbe. Das liegt daran, dass Gelb durch Mischung von Rot und Grün mit der RGB LED generiert wird.

Das kann man ändern in dem man die Tabelle anpasst:

Ver.: 0.7 17.05.19

Erste RGB LED: 0

Startkanal der RGB LED: 0

Schalter Nummer: SI_1

Anzahl der Ausgabe Kanäle: 18

Bits pro Wert: 3 => 8 Helligkeitsstufen (0..7)

Wert Min: 0

Wert Max: 255

Wert ausgeschaltet: 0

Mode: PF_NO_SWITCH_

Analoges Überblenden: 0

Goto Mode: 0

Grafische Anzeige: 1

Ergebnis: `PatternT4(0,8,SI_1,18,0,255,0,PF_NO_SWITCH_OFF,2 Sec,1 Sec,10 Sec,3 Sec,7,0,0,56,0,0,192,1,18`

Makro Name: RGB_AmpelX

Makro: `#define RGB_AmpelX(LED,InNr) PatternT4(LED,8,InNr,18,0,255,0,PF_NO_SWITCH_`
`#define RGB_AmpelX_StCh(LED,StCh,InNr) PatternT4(LED,StCh+8,InNr,18,0,255,0,PF_NO_SWITCH_`

Wenn gleiche Zeiten verwendet werden, dann sollten nur die ersten Zeiten eingetragen werden. Bei lee

Dauer	2 Sec	1 Sec	10 Sec	3 Sec	2 Sec	1 Sec	10 Sec	3 Sec					
FLASH usage:	73 Bytes												
LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	8	9	10	11	12
1	Rot 1	x	x			x	x	x	x				
2													
3													
4	Gelb 1		4		4								
5	Gelb 1		4		4								
6													
7													
8	Grün 1			x									
9													
10	Rot 2	x	x	x	x	x	x						
11													
12													
13	Gelb 2						4		4				
14	Gelb 2						4		4				
15													
16													
17	Grün 2							x					
18													

The file will be closed in 02:50 minutes to give other users a chance

Dazu ändert man in das „Bits pro Wert“ von 1 auf 3. Damit können jetzt 8 verschiedene Helligkeitsstufen dargestellt werden. Das nutzt man in den Feldern der Tabelle in denen „Gelb“ gezeigt werden soll. Hier wird jetzt eine 4 anstelle von einem „x“ eingetragen. Damit ist die resultierende Helligkeit fast gleich groß wie bei Rot und Grün. Mit einer feineren Unterteilung der Helligkeiten könnte man die Gelbe Lampe noch besser anpassen. Dazu würde man den „Bits pro Wert“ noch weiter vergrößern. Man muss allerdings beachten, dass damit der Speicherverbrauch wächst. Bei 3 Bits benötigt man bereits 73 Byte anstelle von 37 Bytes.

Das Beispiel zeigt wie man die Helligkeit einzelner LEDs anpassen kann. Auf der Modelleisenbahn wird man vermutlich keine Ampeln mit RGB LEDs einsetzen, aber das Verfahren lässt sich für viele andere Bereiche anwenden.

7.1.7 Maximal und Minimal Helligkeiten

Eine weitere Möglichkeit zur Anpassung der Helligkeiten der LEDs stellen die Felder „Wert Min:“ und „Wert Max:“ zur Verfügung.

Wert Min: 0
Wert Max: 255

Mit „Wert Min:“ kann bestimmt werden wie Hell die LEDs leuchten, wenn nichts in der LED Tabelle eingetragen ist. Standardmäßig ist bei „Wert Min:“ eine 0 eingetragen. Damit ist die LED aus.

Wichtiger ist in der Regel der Eintrag bei „Wert Max:“. Damit wird die maximale Helligkeit der LEDs bestimmt. Damit lässt sich ganz einfach die Gesamthelligkeit der Ampel verändern. Wenn hier eine 128 eingetragen ist, dann leuchten alle Farben nur noch halb so hell. Die Balken zeigen das anschaulich:

Wert Min: 0

Wert Max: 128

Wert ausgeschaltet: 0

Mode: PF_NO_SWITCH_OFF

Analoges Überblenden: 0

Goto Mode: 0

Grafische Anzeige: 1

Aktualisieren

Das Beispiel benutzt c
spezielle Hardware bei
Das folgend Beispiel z
angesteuert werden.

Ergebnis: **PatternT4(0,0,SI_1,18,0,128,0,PF_NO_SWITCH_OFF,2 Sec,1 Sec,10 Sec,3 Sec,1,2,100,8,0,40,**

Makro Name: **RGB_AmpelX**

```

Makro: #define RGB_AmpelX(LED,InNr)          PatternT4(LED,0,InNr,18,0,128,0,PF_NO_SWI
#define RGB_AmpelX_StCh(LED,StCh,InNr)    PatternT4(LED,StCh+0,InNr,18,0,128,0,PF_NO_SI
    
```

Wenn gleiche Zeiten verwendet werden, dann sollten nur die ersten Zeiten eingetragen werden. I

Dauer	2 Sec	1 Sec	10 Sec	3 Sec	2 Sec	1 Sec	10 Sec	3 Sec
Flash Bedarf: 37 Bytes								

LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	8	9
1	Rot 1	x	x			x	x	x	x	
2										
3										
4	Gelb 1		x		x					
5	Gelb 1		x		x					
6										
7										
8	Grün 1			x						
9										
10	Rot 2	x	x	x	x	x	x			
11										
12										
13	Gelb 2						x		x	
14	Gelb 2						x		x	
15										
16										
17	Grün 2							x		
18										

7.1.8 Langsames Auf- und Ablenden der LEDs

In der Realität werden noch meistens Glühbirnen in den Verkehrsampeln eingesetzt. Hier verwendet man sehr langlebige Birnen, die etwas länger brauchen bis sie die volle Helligkeit erreichen. Die Zeit dafür ist gering, sie fällt aber im Vergleich zu Leuchtdioden auf. Und da die Modellbauer der Realität immer 150 prozentig abbilden müssen, soll die MobaLedLib dieses Verhalten auch mit LEDs simulieren. Dazu gibt es den Schalter „Analoges Überblenden“. Wenn hier eine 1 eingetragen wird, dann werden die Helligkeiten der Ausgänge langsam von einem Zustand in den anderen

übergeblendet. In der Visualisierung sieht das so aus:

Anzahl der Ausgabe Kanäle: 18
 Bits pro Wert: 1 => 2 Helligkeitsstufen (0..1)
 Wert Min: 0
 Wert Max: 255
 Wert ausgeschaltet: 0
 Mode: PF_NO_SWITCH
 Analoges Überblenden: 1
 Goto Mode: 0
 Grafische Anzeige: 1 Aktualisieren

weggelassen. Sie können durch kopieren des Blocks in der Tabelle angelegt werden.
 Das Beispiel benutzt die LEDs eines RGB LED Stripes weil man dazu keine spezielle Hardware benötigt.

Ergebnis: `APatternT8(0,0,SI_1,18,0,255,0,PF_NO_SWITCH_OFF,200 ms,2 Sec,200 ms,1 Sec,200 ms,10 Sec,200 ms,3 Sec,1,2,4,8,144,33,64,1,1)`

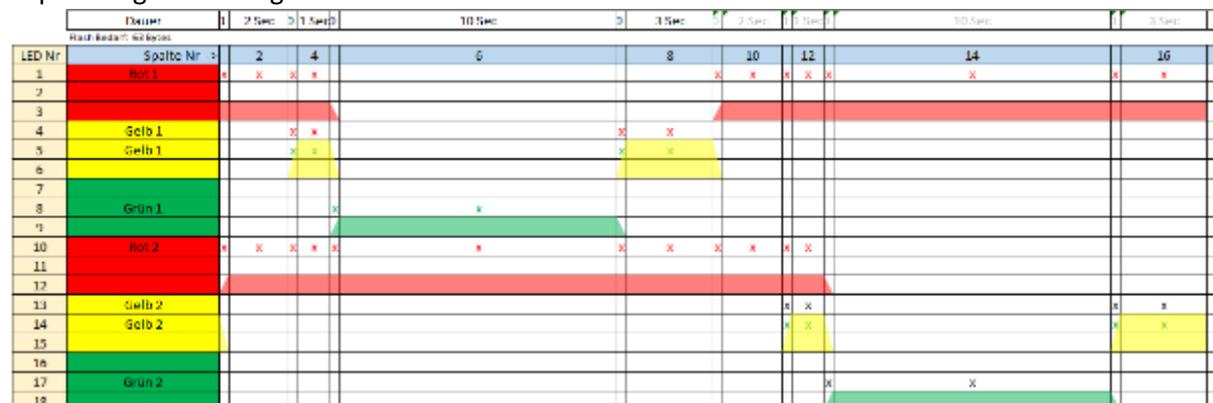
Makro Name: `RGB_AmpelX_Fa`
 Makro: `#define RGB_AmpelX_Fade(LED,InNr) APatternT8(LED,0,InNr,18,0,255,0,PF_NO_SWITCH_OFF,200 ms,2 Sec,200 ms,1 Sec,200 ms,10 Sec,200 ms,3 Sec,1,2,4,8,144,33,64,1,1)`
`#define RGB_AmpelX_Fade_StCh(LED,StCh,InNr) APatternT8(LED,StCh+0,InNr,18,0,255,0,PF_NO_SWITCH_OFF,200 ms,2 Sec,200 ms,1 Sec,200 ms,10 Sec,200 ms,3 Sec,1,2,4,8,144,33,64,1,1)`

Wenn gleiche Zeiten verwendet werden, dann sollten nur die ersten Zeiten eingetragen werden. Bei leeren Spalten werden die vorangegangenen verwendet.

Dauer	200 ms	2 Sec	200 ms	1 Sec	200 ms	10 Sec	200 ms	3 Sec	200 ms	2 Sec	200 ms	1 Sec	200 ms	10 Sec	200 ms	3 Sec	
LED Nr	FLASH usage: 63 Bytes																
Spalte Nr ->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
1	Rot 1	x	x	x	x					x	x	x	x	x	x	x	
2																	
3																	
4	Gelb 1			x	x			x	x								
5	Gelb 1			x	x			x	x								
6																	
7																	
8	Grün 1					x	x										
9																	
10	Rot 2	x	x	x	x	x	x	x	x	x	x	x	x				
11																	
12																	
13	Gelb 2										x	x			x	x	
14	Gelb 2										x	x			x	x	
15																	
16																	
17	Grün 2											x	x				
18																	

Die Helligkeitsbalken laufen jetzt an den Enden spitz zu. Das zeigt, dass sich die Helligkeit in diesen Bereichen kontinuierlich verändert. Dazu wurden vor und nach den eigentlichen Ampelphasen zusätzliche kurze Zeitabschnitte (200 ms) eingefügt, in denen die Lampen heller bzw. dunkler werden. Auf der Modellanlage sieht das sehr schön aus.

Die Spalten in der Tabelle haben die gleiche Breite. Das ist zur Eingabe der Werte praktisch. Allerdings erscheinen dadurch die Helligkeitsverläufe etwas lange. Wenn man die Breiten der Spalten anpasst ergibt sich folgendes Bild:

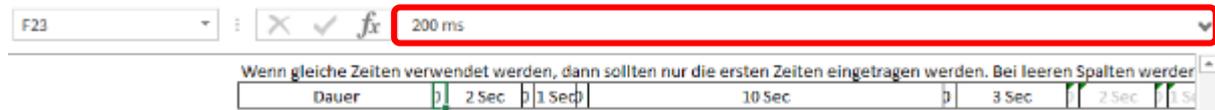


Hier erkennt man auch, dass die Grünphase deutlich länger als die Zwischenphasen ist. Allerdings sind die Werte in der „Dauer“ Reihe nicht mehr lesbar. Diese Ansicht wird aktiviert indem man ein

„D“ in das „Grafische Anzeige“ Feld einträgt:



Wenn man die Werte in der „Dauer“ Zeile lesen will, kann man entweder wieder das „D“ aus dem „Grafische Anzeige“ Feld entfernen oder auf den „Aktualisieren“ Knopf mit gedrückter „Alt“ Taste klicken. Alternativ kann man den Wert auch in der Eingabezeile von Excel sehen oder verändern, wenn man in das entsprechende Feld geht:



Bei sehr großen Unterschieden zwischen minimaler und maximaler Dauer der Spalten ist eine maßstäbliche Darstellung nicht möglich.

7.1.9 Ausgeschaltete Blinkende Ampel

In der realen Welt werden die Ampelanlagen in der Nacht, wenn niemand mehr unterwegs ist, abgeschaltet. Dann Blinkt nur noch die gelbe Lampe. Das muss natürlich im Model genauso umgesetzt werden.

Dazu benötigt man eine Konfiguration, mit der die gelben Lichter der Ampeln blinken. Im Pattern_Configurator findet man das auf der Seite „RGB_AmpelX_Fade_Off“.

Ver.: 0.75 22.05.19 Neues Blatt by Hardi

Erste RGB LED: 0
 Startkanal der RGB LED: 0
 Schalter Nummer: SI_1
 Anzahl der Ausgabe Kanäle: 18
 Bits pro Wert: 1 => 2 Helligkeitsstufen (0..1)
 Wert Min: 0
 Wert Max: 255
 Wert ausgeschaltet: 0
 Mode: PF_INVERT_INP | PF_NO_SWITCH_OFF
 Analoges Überblenden: 1
 Goto Mode: 0
 Grafische Anzeige: 1 Aktualisieren

Ergebnis: APatternT4(0,0,SI_1,18,0,255,0,PF_INVERT_INP | PF_NO_SWITCH_OFF,100 ms,300 ms,100 ms,500 ms,24,48,96,192

Makro Name: RGB_AmpelX_Fade_Off
 Makro: #define RGB_AmpelX_Fade_Off(LED,InNr) APatternT4(LED,InNr,18,0,255,0,PF_INVERT_INP | PF_NO_SWITCH_OFF,100 ms,300 ms,100 ms,500 ms,24,48,96,192)

Wenn gleiche Zeiten verwendet werden, dann sollten nur die ersten Zeiten eingetragen werden. Bei leeren Spalten werden

Dauer	100 ms	300 ms	100 ms	500 ms									
-------	--------	--------	--------	--------	--	--	--	--	--	--	--	--	--

Flash Bedarf: 28 Bytes

LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	8	9	10	11	12
1	Rot 1												
2													
3													
4	Gelb 1	x	x										
5	Gelb 1	x	x	.	.								
6													
7													
8	Grün 1												
9													
10	rot 2												
11													
12													
13	Gelb 2	x	x										
14	Gelb 2	x	x										
15													
16													
17	Grün 2												
18													

Diese Tabelle entspricht den vorangegangenen und muss darum nicht näher besprochen werden.

Neu ist aber der Eintrag im „Mode:“ Feld:

```
Mode: PF_INVERT_INP | PF_NO_SWITCH_OFF
```

Hier findet man zwei Schalter:

- PF_INVERT_INP
- PF_NO_SWITCH_OFF

Mit dem ersten Schalter wird der Eingang invertiert. Damit ist dieses Muster immer dann an, wenn der Eingang ausgeschaltet wird. Zur Steuerung der Ampel wird eine Eingangsvariable eingesetzt. Die Nummer dieser Variable gibt man im Feld „Schalter Nummer:“ an. Die Variable ist aktiv, wenn die Ampel normal arbeitet. Die Eingangsvariable ist 0, wenn die Ampel abgeschaltet ist und nur noch blinken soll. Mit einer Invertierung des Eingangs erreichen wir hier genau den gewünschten Effekt.

Der Schalter „PF_NO_SWITCH_OFF“ ist angegeben, damit sich die „normale“ Ampel und die „blinkende“ Ampel nicht in die Quere kommen. Die beiden Makros sollen ja die gleichen LEDs beeinflussen. Normalerweise schaltet die Pattern Funktion die LEDs aus (Genauer gesagt auf den Wert von „Wert ausgeschaltet:“), wenn der Eingang nicht aktiv ist. Das darf nicht erfolgen, wenn zwei Makros dieselben LEDs ansteuern. Dann würde das zweite nicht aktive Makro die LEDs, die vom ersten Makro aktiviert wurden, wieder ausschalten. Im Abschnitt „Mode:“ auf Seite 80 werden die weiteren möglichen Modes beschrieben.

Für die nachts nicht aktive Ampel benötigt man zwei Makros. Eines das „Tagsüber“ aktiv ist und ein weiteres welches die Ampel „nachts“ blinken lässt. Im Beispiel „09.TrafficLight_Pattern_Func“ der MobaLedLib wird das gezeigt.

Das Excel Programm liefert die beiden Zeilen:

```
APatternT8(LED,StCh+0,InNr,18,0,255,0,PF_NO_SWITCH_OFF,200 ms,2 Sec,200 ms,1 Sec,200 ms, \
    10 Sec,200 ms,3 Sec,1,2,4,8,144,33,64,134,0,128,2,0,10,128,33,0,134, \
    0,1,2,4,8,16,32,67,128,12,1,0,5,0,20,0,67,0,12)
APatternT4(LED,StCh+0,InNr,18,0,255,0,PF_INVERT_INP | PF_NO_SWITCH_OFF,100 ms,300 ms, \
    100 ms,500 ms,24,48,96,192,0,0,0,0,0)
```

Sie werden zu einem neuen Makro zusammengefasst:

```
#define RGB_AmpelX_Fade_IO(LED, InNr) /* Fading Traffic light for tests with RGB LEDs */ \
    APatternT8(LED,0,InNr,18,0,255,0,PF_NO_SWITCH_OFF,200 ms,2 Sec,200 ms,1 Sec,200 ms, \
    10 Sec,200 ms,3 Sec,1,2,4,8,144,33,64,134,0,128,2,0,10,128,33,0,134, \
    0,1,2,4,8,16,32,67,128,12,1,0,5,0,20,0,67,0,12) \
    APatternT4(LED, 0,InNr,18,0,255,0,PF_INVERT_INP | PF_NO_SWITCH_OFF,100 ms,300 ms, \
    100 ms,500 ms,24,48,96,192,0,0,0,0,0)
```

Der Startkanal der RGB LED „StCh“ wurde weggelassen, da er immer 0 ist.

Das Makro „RGB_AmpelX_Fade_IO(LED, InNr)“ kann man einfach in dem Konfigurationsarray verwenden.

```
//*****
// *** Configuration array which defines the behavior of the LEDs ***
MobaLedLib_Configuration()
{
    LED:          First LED number in the stripe
    //           | InCh: Input channel. The input is read in below using the
    //           | | digitalRead() function.
    RGB_AmpelX_Fade_IO(0, 0) // Fading demonstration with a RGB LED stripe
    EndCfg
    // End of the configuration
};
//*****
```

Auf diese Weise kann man beliebige Ampelanlagen zusammenstellen. Es müssen nur die entsprechenden Ampelphasen und die dazugehörigen Zeiten in die Tabellen eingetragen werden. Das Excel Programm erstellt daraus die Konfiguration für das Konfigurationsarray.

Das ist sicherlich ein Feature, das sich mit anderen Mitteln nicht so einfach umsetzen lässt.

Was bei den Beispielen fehlt ist natürlich die Berücksichtigung der Fußgänger, welche in Mainz auf besondere Weise umgesetzt wird:
Das ist eine schöne Übungsaufgabe...



7.1.10 Beispiel: Baustellen Absicherung

Zur Absicherung von Baustellen verwendet man häufig solche Lauflichter:

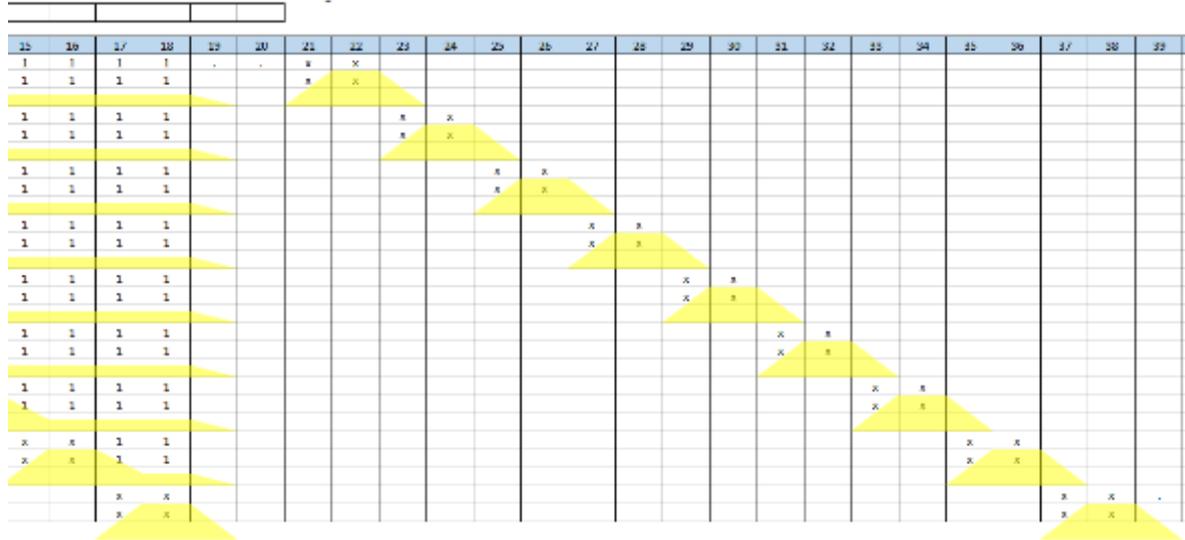


Dafür gibt es die verschiedensten Lösungen für eine Modelleisenbahn. Allerdings lässt sich das noch flexibler durch die Verwendung der MobaLedLib gestalten. Hier kann man die Anzahl der Lampen und den Effekt ganz einfach variieren.

		100 ms		200 ms																
LED Nr	Spalte Nr	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	LED 0 R	x	x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	LED 0 G	x	x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	LED 0 B																			
4	LED 1 R			x	x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	LED 1 G			x	x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	LED 1 B																			
7	LED 2 R					x	x	1	1	1	1	1	1	1	1	1	1	1	1	1
8	LED 2 G					x	x	1	1	1	1	1	1	1	1	1	1	1	1	1
9	LED 2 B																			
10	LED 3 R							x	x	1	1	1	1	1	1	1	1	1	1	1
11	LED 3 G							x	x	1	1	1	1	1	1	1	1	1	1	1
12	LED 3 B																			
13	LED 4 R									x	x	1	1	1	1	1	1	1	1	1
14	LED 4 G									x	x	1	1	1	1	1	1	1	1	1
15	LED 4 B																			
16	LED 5 R											x	x	1	1	1	1	1	1	1
17	LED 5 G											x	x	1	1	1	1	1	1	1
18	LED 5 B																			
19	LED 6 R														x	x	1	1	1	1
20	LED 6 G														x	x	1	1	1	1
21	LED 6 B																			
22	LED 7 R																x	x	1	1
23	LED 7 G																x	x	1	1
24	LED 7 B																			
25	LED 8 R																		x	x
26	LED 8 G																		x	x
27	LED 8 B																			

Diese Tabelle zeigt wie man die Warnlichter so konfiguriert, dass sie kurz aufblitzen und dann mit reduzierter Helligkeit weiter leuchten. Diesen Effekt findet man bei vielen Baustellen. In dem Beispiel

„RGB_ConstrWarnLight“ wurde das noch mit einem einfachen Lauflicht kombiniert:



Die Tabelle in dem Beispiel hat scheinbar eine Reihe zu wenig. Das wurde absichtlich so eingestellt, weil die letzte blaue LED in dem Beispiel nicht benutzt wird. Damit werden 10 Bytes eingespart. Eigentlich könnte man alle blauen LEDs weglassen, aber das wird von der Bibliothek nicht unterstützt. Auf der Anlage wird man ja auch keine RGB LEDs einsetzen, sondern einzelne LEDs welche per WS2811 angesteuert werden.

Das Beispiel zeigt, dass man mit der MobaLedLib Lauflichter und andere animierte Lichteffekte im Handumdrehen erstellen und anpassen kann.

7.1.11 Der Goto Mode

„Goto“ ist unter Programmieren ein Unwort. Das hat man früher in Basic Programmen verwendet. Trotzdem wird es in der MobaLedLib verwendet, weil man damit bestimmte Aufgaben einfach und Ressourcenschonend umsetzen kann. Hier wird es verwendet, wenn die Bibliothek auf verschiedene Ereignisse von außen reagieren soll und dabei die gleichen LEDs betroffen sind. Das soll am Beispiel eines Lichtsignals gezeigt werden. Mit dem Pattern_Configurator kann man das ganz einfach umsetzen. Dazu wird ein Ausfahrtsignal gewählt welches 4 Aspekte darstellen soll: HP0, HP1, HP2 und HP0 + SH1. Im Excel Programm ist das auf der Seite „Dep Signal4“ vorbereitet. Dieses Mal wird die Version betrachtet, die für die Verwendung mit einzelnen LEDs in einem Modellbausignal gedacht ist und nicht wie bei den vorangegangenen Beispielen die RGB Version. Das folgende Bild zeigt die Excel Seite:

Ver.: 0.75 22.05.19

Erste RGB LED: 1
 Startkanal der RGB LED: 0
 Schalter Nummer: SI_1
 Anzahl der Ausgabe Kanäle: 5
 Bits pro Wert: 4 => 16 Helligkeitsstufen (0..15)
 Wert Min: 0
 Wert Max: 128
 Wert ausgeschaltet: 0
 Mode: 0
 Analoges Überblenden: X
 Goto Mode: 1
 Grafische Anzeige: 1

Ergebnis: XPatternT1(1,12,SI_LocalVar,5,0,128,0,0,500 ms,15,240,0,15,0,240,15,240,0,16 ,63,191,191,191) _Dep_Signal4

Makro Name: _Dep_Signal4
 Makro: #define _Dep_Signal4(LED) XPatternT1(LED,12,SI_LocalVar,5,0,128,0,0,500 ms,15,240,0,15,0,240,15,240,0,16 ,

#define _Dep_Signal4_StCh(LED,StCh) XPatternT1(LED,StCh+12,SI_LocalVar,5,0,128,0,0,500 ms,15,240,0,15,0,240,15,240,0,1

Wenn gleiche Zeiten verwendet werden, dann sollten nur die ersten Zeiten eingetragen werden. Bei leeren Spalten werden die vor

Dauer	500 ms													
Flash Bedarf: 23 Bytes														

LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Rot 1	x			x									
2	Grün		x											
3	Gelb			x										
4	Rot 2	x												
5	Weiss				1									

HP0 HP1 HP2 HP0+SH1

Hier ist die neue „Goto“ Tabelle zu sehen. Sie wird eingeblendet, wenn in dem „Goto Mode:“ Feld eine 1 eingetragen ist. In der Tabelle findet man ein paar Buchstaben. Damit wird bestimmt, welche Eigenschaften die entsprechende Spalte hat. Das „E“ bedeutet, dass die Abarbeitung in der Spalte beendet wird. Mit dem „S“ kennzeichnet man Spalten welche als Start Spalten verwendet werden können. Die Pfeile darüber visualisieren das. Sie werden angezeigt, wenn in dem Feld „Grafische Anzeige“ eine Eins steht.

Die erste Spalte enthält kein „S“ und trotzdem zeigt der Pfeil, dass sie angesprungen werden kann. Das liegt daran, dass die erste Spalte immer mit der 0 angesprungen werden kann. In den vorangegangenen Beispielen wurde das stillschweigend vorausgesetzt. Die Abarbeitung der LED Tabelle beginnt normalerweise immer in der ersten Spalte.

Wenn mit einer anderen Spalte begonnen werden soll, dann muss man das dem Programm über einen Parameter mitteilen. Dazu kann man beispielsweise DCC Kommandos verwenden (Siehe Kapitel „5.1.2 Vom InCh zu den LEDs“). Über eine Taste an der DCC Zentrale wird ein Befehl zur MobaLedLib geschickt. Der Tastendruck wird dort in dem Array „InCh[]“ gespeichert. Mit der Funktion „InCh_to_TmpVar()“ werden mehrere „InCh[]“ Werte zu einer Zahl umgewandelt. Diese Zahl wird dann an die Pattern Funktion übergeben, die über die Goto Tabelle entscheidet, welche Spalte angesprungen werden soll wenn sich der Eingang geändert hat. Für das Signal werden 4 Taster benötigt. Mit dem ersten Taster wird HP0 angesprungen, mit dem Zweiten HP1...

Warum zwei Dreiecke?

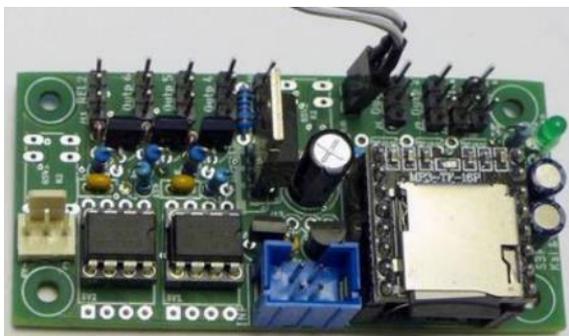
Die Grafik in der LED Tabelle zeigt in vielen Feldern zwei übereinanderliegende Dreiecke. Damit soll ausgedrückt werden, dass unklar ist ob die entsprechende LED zu Beginn an oder aus ist. Das ist abhängig von dem vorangegangenen Zustand.



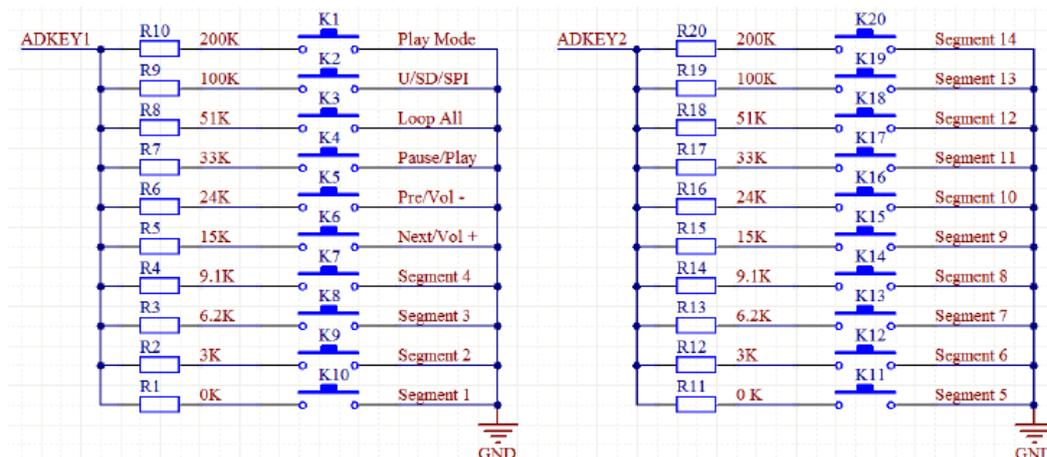
Durch die Verwendung der Goto Funktionalität ist das aber unklar. Wenn die LED aus war, dann wird sie stufenlos heller bis sie am Ende den entsprechenden Endwert erreicht hat. Das dunkle Dreieck zeigt diesen Verlauf. Wenn die LED bereits von einem vorangegangenen Befehl eingeschaltet wurde, dann bleibt sie weiterhin an. Natürlich ist es auch möglich, dass die LED zu Beginn einen Zwischenwert hatte. In diesem Fall verändert sich die Helligkeit in der angegebenen Zeit bis hin zum Endwert. Dieser Verlauf zeigt das helle Dreieck. Der Endwert ist in diesem Beispiel über den Parameter „Wert Max:“ auf 128 beschränkt. Darum füllt die Grafik nicht die ganze Zelle. Für dieses Überblenden von einem vorangegangenen Wert auf einen neuen Wert benötigt die Bibliothek ein zusätzliches Byte RAM pro LED. Diese spezielle Funktionalität wird aktiviert indem ein „X“ in das Feld „Analoges Überblenden:“ eingetragen wird.

7.1.12 Weitere Eigenschaften der GotoZeile

In dem letzten Beispiel wurden die „S“ = Start und „E“ = Ende Eigenschaften der Goto Zeile besprochen. Es können aber noch weitere Eigenschaften verwendet werden. Das wird anhand von dem Beispiel „LocalVar Sound“ erklärt. Geräusche können mit der MobaLedLib über sehr günstige kleine Sound Module wiedergegeben werden (Siehe „2.6 Sound“). Eins davon ist das „MP3-TF-16P“ Sound Modul (Im Bild rechts). Es kann verschiedene MP3 oder WAV Dateien auf Knopfdruck von einer SD Karte wiedergeben.



Zur Auswahl der zu spielenden Datei werden normalerweise Widerstandskodierte Taster verwendet:



Sie werden über die zwei Eingänge ADKEY1 und ADKEY2 mit dem Sound Modul verbunden. Diese Eingänge werden von der Bibliothek über ein WS2811 IC angesprochen. Dazu wird eine bestimmte „Helligkeit“ generiert, die vom Sound Modul als gedrückte Taste interpretiert wird. Die Pattern Funktion wird dazu verwendet, dass die gewünschte „Taste“ für eine bestimmte Zeit gedrückt wird und anschließend wieder losgelassen wird. Auch diese Aufgabe wurde über dem Pattern_Configurator erstellt:

Ver.: 0.75 22.05.19 Neues Blatt by Hardi

Erste RGB LED: 0
 Startkanal der RGB LED: 0
 Schalter Nummer: SI_1
 Anzahl der Ausgabe Kanäle: 2
 Bits pro Wert: 8 => 256 Helligkeitsstufen (0..255)
 Wert Min: 0
 Wert Max: 255
 Wert ausgeschaltet: 0
 Mode: 0
 Analoges Überblenden: 0
 Goto Mode: 1
 Grafische Anzeige: 1

Ergebnis: PatternT1(0,28,SI_LocalVar,2,0,255,0,0,200 ms,,,255,,134,,70,,49,,,255,,134,,70,,49,,37,,29,,25,,22,,17,,11,0,0 ,1,129,129,129,129,129,129,129,129,125

Makro Name: LocalVar_Sound
 Makro: #define LocalVar_Sound(LED) PatternT1(LED,28,SI_LocalVar,2,0,255,0,0,200 ms,,,255,,134,,70,,49,,,255,,134,,70,,49,,37,,29,,25,,22,,17,,11,0

#define LocalVar_Sound_StCh(LED,StCh) PatternT1(LED,StCh+28,SI_LocalVar,2,0,255,0,0,200 ms,,,255,,134,,70,,49,,,255,,134,,70,,49,,37,,29,,25,,22,,17,,1

Wenn gleiche Zeiten verwendet werden, dann sollten nur die ersten Zeiten eingetragen werden. Bei leeren Spalten werden die vorangegangenen Zeiten wiederholt.

Dauer	200 ms																	
Flash Bedarf: 45 Bytes																		

Goto Tabelle

LED Nr	Spalte Nr ->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	ADKey 1		255	134	70	49											0
2	ADKey 2						255	134	70	49	37	29	25	22	17	11	0

In dem Bildschirmfoto sieht man wieder die Goto Tabelle. Hier werden zusätzlich Linien unterhalb der Zeile gezeigt. Sie visualisieren Sprunganweisungen innerhalb der Tabelle.

Für jede Sound Datei gibt es eine Spalte (2-15). Sie enthält den „Helligkeitswert“, die vom Sound Modul als Spannung interpretiert wird und dem entsprechenden Widerstand entspricht. Der „Taster“ soll für eine bestimmte Zeit betätigt werden und anschließend wieder losgelassen werden. Das wird über die Zeile „Dauer“ festgelegt. Nach dieser Zeit wird die Spalte 16 angesprungen. Hier wird die „Taste“ wieder „losgelassen“ indem eine 0 ausgegeben wird.

Der Sprung zur Spalte 16 wird über den Eintrag „G 1“ in der „Goto“ Tabelle ausgelöst. Mit „G 1“ wird der Pattern Funktion mitgeteilt, dass sie nach dem Ablauf der entsprechenden Dauer zu der Position 1 springen soll. Welche Position das ist bestimmt das „P“ in der letzten Spalte. Ein „G 2“ würde zum zweiten „P“ in der Tabelle springen. In diesem Beispiel wird aber nur eine Position angesprungen. Es können maximal 62 Positionen (1-62) benutzt werden.

Zur Ansteuerung der verschiedenen „Tasten“ wird eine feine Abstufung der „Helligkeitswerte“ benötigt. Darum ist im Feld „Bits pro Wert“ 8 eingetragen. Damit können 256 verschiedene PWM Werte generiert werden.

7.1.16 Neues Blatt anlegen

Neues Blatt

Mit dem Knopf „Neues Blatt“ wird, wie der Name sagt, eine neue Excel Seite angelegt. Diese Seite kann eine Kopie der aktuellen Seite sein oder keine Daten enthalten. Das kann der Benutzer auf Anfrage festlegen.

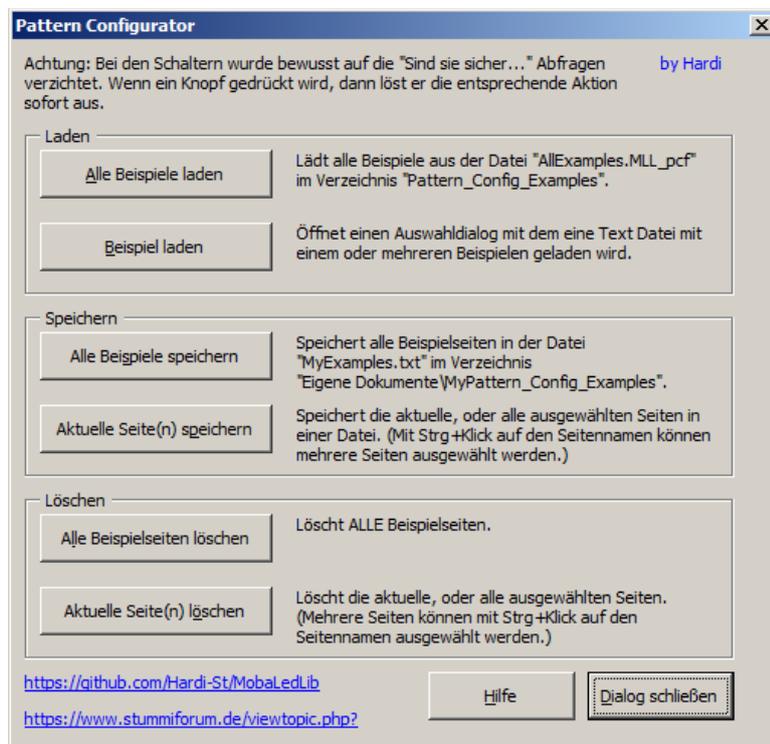
Mit einem zweiten Dialog wird der Name des neuen Blattes angefragt. Hier sollte man natürlich einen möglichst aussagekräftigen Namen verwenden. Der Name wird auch im Feld „Makro Name“ als Default wert eingetragen.

7.1.17 Menü des Pattern_Configurators

Weitere Funktionen des Programms verstecken sich unter dem LED Kranz oben links:



Wenn dieser angeklickt wird erscheint die folgende Auswahl:



Der Dialog ist so gestaltet, dass man die Excel Seiten noch auswählen kann während der Dialog angezeigt wird. Damit kann man dann mehrere Seiten zum Speichern oder Löschen auswählen.

Zur Auswahl mehrerer Seiten gibt es zwei Möglichkeiten.

1. Man klickt nacheinander auf die auszuwählenden Seitennamen (Unten am Rand) mit festgehaltener „Strg“ Taste.



Zum Deselektieren einer Seite wird diese ein zweites Mal angeklickt.

2. Mehrere nebeneinander liegende Seiten können markiert werden, indem die erste Seite normal angeklickt wird und die letzte mit gleichzeitig gedrückter „Großschreib“ Taste.

Normalerweise sollte man beim Beenden die Datei über die normale Excel Funktion speichern.  Dann werden alle Änderungen an den Tabellen gespeichert. Mit den Funktionen des oben gezeigten Menüs kann man aber auch einzelne oder mehrere Seiten separat abspeichern. Das sollte man unbedingt vor dem Update auf eine neue Version des „Pattern_Configurators“ mit den eigenen Beispielen machen, denn sonst gehen diese Änderungen verloren.

7.2 Parameter der Pattern Funktion

Für den tieferen Einstieg in die Pattern Funktion ist die Kenntnis der Parameter wichtig. Eigentlich handelt es sich bei den Einträgen in das Konfigurationsarray nicht um Funktionsaufrufe, sondern um Makros welche konstante Datenbytes für das Array generieren. Mit dem Aufruf der Update Funktion (Siehe 3.1.5 MobaLedLib.Update()) werden diese Datenbytes ausgewertet. Die Makros sorgen dafür, dass die Bytes in der richtigen Reihenfolge abgelegt werden und übernehmen Umwandlungen und Überprüfungen.

7.2.1 Umwandlung der Daten in einzelne Bytes

Im Array werden nur Bytes gespeichert. Einige der Parameter belegen aber mehr als ein Byte. Das Makro splittet diese Werte in mehrere Bytes.

Ein Beispiel dafür sind die Zeitangaben welche im Pattern_Configurator in der „Dauer“ Tabelle stehen. Hier können Zeiten von einer Millisekunde bis zu 65.535 Sekunden verwendet werden. Dazu wird eine Vorzeichenlose 16 Bit Zahl verwendet. Das Untermakro „_T2B(t)“, welches wiederum „_W2B(w)“ aufruft, wandelt diese Zeiten in zwei Bytes:

```
#define _W2B(w)    (((uint32_t)(w)&0xFF), ((uint32_t)(w)>>8))
#define _T2B(t)    _W2B(t)
```

Man sieht hier sehr schön, dass das Makro zwei per Komma getrennte Zahlen generiert. Die erste Zahl enthält das untere Byte der Zeit, das zweite das obere.

Zur Umwandlung der Zeiten werden außerdem die folgenden Makros benutzt:

```
#define Min        *60000L
#define Sec        *1000L
#define Sek        Sec
#define sek        Sec
#define sec        Sec
#define Ms
#define ms
```

Damit werden die Eingaben in der Bibliothek leichter lesbar. Der Präprozessor von c++ wandelt sogar die Eingabe „1.5 Min“ korrekt in 90000 um.

Es ist ebenso möglich, Summen in der „Dauer“ Tabelle zu verwenden: „1 Min + 15 Sek“.

7.2.2 Der „normale“ Pattern Befehl

Eigentlich gibt es nicht eine Pattern Funktion, sondern gleich mehrere. Diese unterscheiden sich durch die Anzahl der Parameter und unterstützen spezielle Funktionen. Außerdem benötigen sie unterschiedlich viel RAM.

Zur Reduzierung des Speicherbedarfs werden nur so viele Daten wie nötig abgelegt. Ein Datensatz der Pattern Funktion besteht aus einem Anfang mit fester Länge und einem Anteil mit variabler Länge welcher abhängig von der Anzahl der verwendeten Daten ist.

Neben der LED Tabelle gibt es die Tabelle mit der Dauer der einzelnen Spalten welche ebenfalls eine variable Länge haben kann.

Unterschiedliche „Dauer“ Tabellen

Zur übersichtlicheren Darstellung gibt es für die verschiedenen Größen der „Dauer“ Tabelle ein geeignetes Makro. Diese unterscheiden sich über ein angehängtes „T1“ bis „T20“. Die Verwendung des richtigen Makros erledigt das Excel Programm „Pattern_Configurator“. Eine Pattern Funktion mit drei „Dauer“ Spalten benutzt das Makro „PatternT3()“.

Aus der Sicht der Implementierung ist der Name „PatternT3“ auch eine Konstante. Alle in dem Konfigurationsarray möglichen Makros werden über eine eigene Nummer identifiziert. Durch die Vergabe eines eigenen Makronamens für jede Länge der „Dauer“ Tabelle wird die Längeninformatio in diese Nummer kodiert. Es wird also kein zusätzliches Byte belegt welches die Länge der Tabelle enthält.

Parameter Beispiel

Das Makro „PatternT3()“ soll beispielhaft zur Beschreibung der Parameter benutzt werden:

```
#define PatternT3( LED,NStru,InCh,LEDs,Val0,Val1,Off,Mode,T1,T2,T3,...) \
    PATTERN3_T, _CHKL(LED)+RAM5,(NStru)&0xFF,_ChkIn(InCh),SI_1,LEDs, \
    Val0,Val1,Off,Mode,_T2B(T1),_T2B(T2),_T2B(T3), \
    _W2B(COUNT_VARARGS(__VA_ARGS__)), __VA_ARGS__,
```

LED

Der erste Parameter ist die Nummer der ersten benutzten RGB LED für die Pattern Funktion. Die LEDs werden beginnen von 0 bis 255 gezählt.

NStru

Der Parameter „NStru“ beinhaltet mehrere Zahlen welche als eine Struktur abgelegt sind. Dadurch wird der benötigte Platz optimiert. Die Bits innerhalb von „NStru“ haben folgende Positionen und Bedeutungen:

FFFNLL

LL: Die untersten beiden Bits beschreiben den Startkanal innerhalb der RGB LED. Null bedeutet das Muster beginnt mit dem ersten Kanal, welcher der Farbe Rot einer WS2812 LED entspricht. Entsprechend beginnt das Muster mit dem „Grünen“ Kanal, wenn eine Eins, und mit „Blau“ wenn eine Zwei verwendet wird.

NNN: Die Anzahl der Bits welche zur Speicherung der Helligkeit eines Ausgangs benutzt wird mit den drei mit NNN gekennzeichneten Bits abgelegt. Zu dem Zahlenwert wird Eins addiert so dass eine 0 abgelegt wird, wenn 1 Bit zur Speicherung der Helligkeit benutzt wird. Es können bis zu 8 Bits benutzt werden (NNN=7). Ein größerer Wert in NNN erhöht den Speicherbedarf der LED Tabelle ermöglicht aber auch feinere Schritte der Helligkeiten.

FFF: Enthält die Anzahl unbenutzten Bits in der LED Tabelle. Dieser Wert wird intern für die Berechnung der Zustände benutzt. Das Excel Programm ermittelt diesen Wert automatisch.

Der Parameter „NStru“ hat einen Komplexen Aufbau und kann nicht so einfach manuell berechnet werden.

InCh

Der dritte Parameter der Pattern Funktion enthält den Eingangskanal mit dem die Funktion ein- oder ausgeschaltet wird. Damit kann die Pattern Funktion nur dann aktiviert werden, wenn bestimmte äußere Bedingungen zutreffen. So kann zum Beispiel eine Ampel zu fortgeschrittener Stunde abgeschaltet werden. Der Eingangskanal ist Nummer einer Variablen zwischen 0 und 255. Diese Variablen können im Programm von DCC Kommandos, Schaltern oder anderen Eingangsgrößen gesetzt werden. Es gibt auch einige fest definierte besondere Eingänge welche über die folgenden Konstanten verwendet werden können. Diese speziellen Eingänge belegen die Nummern 253-255. Darum sollten diese Nummern nicht im Programm für andere Zwecke benutzt werden:

- SI_1: Dieser besondere Eingang ist immer an. Er wird dann verwendet, wenn eine Funktion nicht abschaltbar sein soll.
- SI_0: Mit diesem Eingang wird eine Funktion immer deaktiviert. Das kann für Eingänge mit inverser Logik oder unbenutzte Trigger Eingänge benutzt werden. Bei der Pattern Funktion darf diese Konstante nicht benutzt werden denn sie hat den gleichen Zahlenwert wie die nächste Konstante.
- SI_LocalVar: Zur Kennzeichnung einer alternativen Eingangsquelle wird diese Konstante eingesetzt. Mit ihr wird dem Programm mitgeteilt, dass nicht nur auf eine binäres Eingangssignal reagieren soll, sondern verschiedene Zustände aus einer lokalen 8 Bit Variable auswerten soll. Damit können zum Beispiel verschiedene Signalbilder generiert werden. Diese Variable wird von dem vorangegangenen Makro befüllt. Im Beispiel „10.RailwaySignal_Pattern_Func“ wird das gezeigt. Wenn der „Goto“ Mode verwendet wird trägt das Excel Programm diese Konstante automatisch ein.

LEDs:

Mit dem Parameter „LEDs“ wird die Anzahl der verwendeten LED Kanäle spezifiziert. Dabei werden die drei LEDs einer RGB LED einzeln gezählt. Im „Pattern_Configurator“ entspricht das der Anzahl der Zeilen der LEDs Tabelle. Es können maximal 100 LED Kanäle benutzt werden.

Val0:

Legt den minimalen Helligkeitswert aller LEDs fest. Dieser Wert wird verwendet, wenn in der LED Tabelle nichts, „0“, „.“ oder „-“ eingetragen ist. Er wird auch zur Berechnung der Zwischenwerte benutzt, wenn „Bits pro Wert“ größer als 1 und kleiner als 8 ist. Die Zwischenwerte werden über diese Formel ermittelt:

$$\text{Helligkeit} = \text{Eingabe} * \frac{\text{Val1} - \text{Val0}}{\text{Helligkeitsstufen}} + \text{Val0}$$

Val1:

Der Zahlenwert im „Val1“ bestimmt die maximale Helligkeit der Ausgänge.

Off:

Die Helligkeit der LED bei deaktivierter Funktion wird hiermit angegeben. Die Pattern Funktion kann über den Parameter „InCh“ deaktiviert werden.

Mode:

Der Mode Parameter bestimmt die Arbeitsweise der Pattern Funktion. Er kann aus verschiedenen Schaltern und Modes zusammengesetzt werden. Dazu sind verschiedene Konstanten in der Bibliothek definiert. Schalter beginnen mit „PF_“ was für „Pattern Flag“ steht. Diese Schalter können beliebig mit anderen Schaltern kombiniert werden. Neben den Flags gibt es eine Reihe verschiedene Modes. Hier darf immer nur ein Mode verwendet werden. Der Mode kann mit beliebigen Schaltern kombiniert werden. Achtung, nicht jede Kombination generiert ein sinnvolles Ergebnis.

Es gibt folgende Modes:

- PM_NORMAL: Normale Betriebsart der Pattern Funktion. Die LED Tabelle wird Spalte für Spalte abgearbeitet. Am Ende wird wieder von vorne begonnen.

- PM_SEQUENZ_W_RESTART:** In diesem Modus werden die LEDs entsprechend der „LED“ und „Dauer“ Tabelle nacheinander angesteuert. Die Sequenz bleibt beim letzten Zustand stehen. Es wird nicht wie beim „PM_NORMAL“ automatisch wieder mit dem Anfang begonnen. Der Ablauf kann mit einer neuen Flanke am Eingang (InCh) unterbrochen und neu gestartet werden. Der Ablauf beginnt dann wieder mit der ersten Spalte.
- PM_SEQUENZ_W_ABORT:** Wenn dieser Mode benutzt wird, dann kann der Ablauf mit einer neuen Flanke am Eingang (InCh) vorzeitig beendet werden.
- PM_SEQUENZ_NO_RESTART:** Bei dieser Betriebsart kann der Ablauf nicht unterbrochen und neu gestartet werden.
- PM_SEQUENZ_STOP:** Diese Sequenz wird sofort angehalten, wenn der Eingang 0 wird. Zum Starten wird, wie bei den vorangegangenen Modes, eine positive Flanke am Eingang benötigt.
- PM_PINGPONG:** In dieser Betriebsart wird die Sequenz der LEDs abwechselnd vom Anfang zum Ende und dann vom Ende zum Anfang, am Ende durchlaufen, wie ein Ball der hin und her geworfen wird.
- PM_HSV:** HSV steht für „Hue“, „Saturation“ und „Value“. Mit diesen drei Parametern kann ein alternatives Farbmodell zur Ansteuerung der LEDs benutzt werden. Normalerweise werden die drei Farben Rot, Grün und Blau der LEDs einzeln angesprochen. Bei einer Änderung der Farbe müssen alle drei Parameter verändert werden. Mit dem HSV Farbraum kann man über einen einzigen Wert (Hue) die Farbe einer RGB LED von Rot über Gelb, Grün und Blau bis Violett verändern. Über den Parameter „Saturation“ kann die Sättigung einer Farbe von der reinen Farbe bis zu Weiß modifiziert werden. Die Helligkeit wird mit „Value“ angegeben. Wenn dieser Schalter benutzt wird, spricht der erste Kanal einer LED nicht mehr die rote LED an, sondern den „Hue“ Wert. Entsprechend adressiert der nächste Startkanal die „Saturation“ und der dritte Kanal den „Value“. Die Startkanäle werden im „Pattern_Configurator“ von 0 bis 2 durchnummeriert. Mit diesen drei Parametern kann man sehr schöne Überblend-Effekte realisieren. Das Beispiel „16.Illumination_Pattern_Func“ zeigt wie das gemacht werden kann.
- Neben den verschiedenen Modes kann der Mode Parameter einige Flags enthalten. Mehrere dieser Flags können mit einer der „PM_...“ Modes kombiniert werden. Als Trennzeichen wird der „oder“ Operator „|“ von C verwendet.
- PF_NO_SWITCH_OFF** Wenn dieser Schalter benutzt wird, dann werden die LEDs nicht abgeschaltet, wenn der Eingang (InCh) abgeschaltet wird. Normalerweise werden alle Ausgänge auf den mit „Off“ spezifizierten Helligkeitswert gesetzt, wenn der Eingang abgeschaltet wird. Wenn der Schalter „PF_NO_SWITCH_OFF“ verwendet wird, dann werden die Ausgänge nicht verändert. Das ist wichtig, wenn bestimmte LEDs von verschiedenen Konfigurationszeilen angesteuert werden. Im Beispiel

	<p>„09.TrafficLight_Pattern_Func“ wird dieses Flag genutzt. Hier soll die normale Funktion der Ampel bei Nacht deaktiviert werden und durch ein blinkendes gelbes Licht ersetzt werden. Die Lampen der Ampel werden entweder von der „normalen“ Pattern Funktion gesteuert oder von einer zweiten Funktion welche die gelbe Lampe blinken lässt. Ohne den Schalter „PF_NO_SWITCH_OFF“ würden beim Umschalten von Nacht auf Tag alle LEDs bis zur nächsten normalen Ampelphase ausgeschaltet bleiben. Das liegt daran, dass das „Nacht“ Makro die LEDs nach dem „Tag“ Makro setzt. Im Umschaltmoment schreibt das „Tag“ Makro die Helligkeitswerte für die erste Ampelphase in das „leds[]“ Array. Die anschließend aufgerufene „Nacht“ Funktion schaltet ohne dem „PF_NO_SWITCH_OFF“ Flag alle Ausgänge ab, weil der „InCh“ deaktiviert wurde. Das bleibt dann so lange bis die „Tag“ Funktion die nächste Ampelphase anzeigt.</p> <p>Dazu muss man wissen, dass die Helligkeit der LEDs immer nur bei einer Änderung aktualisiert wird zur Minimierung der Rechenzeit.</p>
PF_EASEINOUT	<p>Das menschliche Auge nimmt Änderungen in den Randbereichen der Helligkeit (0 oder 255) anders wahr als in dem mittleren Helligkeitsbereich. Mit dem Schalter „PF_EASEINOUT“ wird das korrigiert.</p>
PF_SLOW	<p>Die Zeiten in der „Dauer“ Spalte werden normalerweise in Millisekunden gemessen. Mit den Makros „Sec“ oder „Min“ werden die angegebenen Werte entsprechend skaliert. Intern werden aber Millisekunden verwendet. Das bedeutet, dass normalerweise maximal 65535 ms = 65 Sekunden möglich sind. Mit dem Schalter PF_SLOW werden alle Zeiten einer Pattern Funktion durch 16 geteilt. Dadurch können dann Einzelne Abschnitte bis zu 17 Minuten lang sein.</p> <p>Achtung: Die Angabe in der „Dauer“ Tabelle muss „von Hand“ angepasst werden. Hier kann man z.B. diese Formulierung verwenden:</p> <p>„9 Min / 16“</p>
PF_INVERT_INP	<p>Im Ampel Beispiel („09.TrafficLight_Pattern_Func“) wird gezeigt, dass es unter Umständen sinnvoll ist, wenn man einen invertierten Eingang zur Steuerung der Pattern Funktion zur Verfügung hat. In dem Beispiel wird der „Nacht“ Modus aktiviert, wenn der Eingang ausgeschaltet ist. Mit dem Schalter „PF_INVERT_INP“ kann das einfach umgesetzt werden.</p>
_PF_XFADE	<p>Dieser Schalter wird automatisch benutzt, wenn eins der „XPattern?“ Makros verwendet wird. Normalerweise muss man ihn nicht manuell angeben, deshalb ist er mit einem vorangestellten Unterstrich markiert. Mit dem Schalter wird das Überblenden der Helligkeiten verändert. Wenn er aktiviert ist wird die Helligkeit einer LED von dem letzten tatsächlichen Helligkeitswert zum neuen Wert übergeblendet. Damit werden auch Überblendvorgänge bei der Verwendung der „Goto“ Funktion richtig unterstützt. Ohne dieses Flag wird nicht berücksichtigt, welche Spalte tatsächlich vorher aktiv war.</p> <p>Wenn der Schalter benutzt wird, dann muss für jede LED der Pattern Funktion ein Byte zum Speichern der letzten Helligkeit reserviert werden was u.U. bei dem kleinen RAM eines Arduinos zu Engpässen</p>

führen kann. Darum sollte das Flag nur dann benutzt werden, wenn es tatsächlich nötig ist.

Damit sind die Parameter der Pattern Funktion mit fester Anzahl beschrieben. Darauf folgen zwei Parametergruppen mit variabler Anzahl.

Zeiten T1..T20:

In der „Dauer“ Tabelle des Pattern_Configurators wird angegeben wie lange eine Spalte aktiv ist. Da es viele Anwendungen gibt, bei denen die Zeiten gleich sind oder sich mehrere Zeiten wiederholen, muss nicht jede Spalte eine eigene Zeitangabe haben. Spalten ohne Zeitangaben wiederholen die Angaben der vorangegangenen Spalten. Im einfachsten Fall wird eine Dauer angegeben, die für alle Spalten gültig ist. Es ist aber auch möglich, eine Sequenz mit verschiedenen Zeiten zu verwenden, die dann automatisch wiederholt wird. Im Beispiel „09.TrafficLight_Pattern_Func“ wird das verwendet. Hier sind die Ampelphasen der einen Ampel gleich lang wie die Zeiten der zweiten Ampel. Darum werden die Zeiten nur für die erste Ampel angegeben. Das Programm verwendet diese Einstellungen auch für die zweite Ampel.

Die Anzahl der verwendeten Zeiten wird dem Programm über den Namenszusatz „_T?“ des Makros mitgeteilt. Das Excel Programm macht das automatisch.

LED Datenbytes:

Als letzte Parameter der Pattern Funktion kommen die Datenbytes der „LED“ Tabelle. Die Anzahl dieser Bytes ist abhängig von der Anzahl der Spalten. Im Makro wird das über die drei Punkte angedeutet „...“. Das Excel Makro ermittelt aus den Eingaben des Benutzers eine Liste von Bytes welche an die Pattern Funktion übergeben werden. Dabei werden die Eingaben komprimiert abgelegt. Es werden immer nur so viele Bits gespeichert wie über das Feld „Bits pro Wert“ ausgewählt wurden. Wenn ein Bit pro Wert verwendet werden soll, dann werden 8 Einträge der „LED“ Tabelle in einem einzigen Byte gespeichert. Dadurch kann sehr viel FLASH Speicher gespart werden. Das ist eins der großen Vorzüge der Pattern Funktion.

7.2.3 Weitere Pattern Befehle

Neben dem „normalen“ Pattern Befehl gibt es noch weitere Pattern Funktionen, die hier beschrieben werden.

APatternT..

Der APattern Befehl (A = Analog) verwendet die gleichen Parameter wie der „normale“ Pattern Befehl. Er unterscheidet sich von der einfachen Variante dadurch, dass die Helligkeiten der Ausgänge nicht schlagartig, sondern langsam gewechselt werden.

Normale Glühlampen reagieren träge auf Ein- und Ausschaltbefehle. Das liegt daran, dass der Glühwendel eine gewisse Zeit benötigt bis er warm ist oder wieder abgekühlt ist. Bei Signalanlagen im Schienen- und Straßenverkehr setzt man besonders langlebige Glühlampen ein. Diese zeigen dieses verzögerte Ein- und Ausschalten noch deutlicher als Glühlampen welche im Haushalt eingesetzt werden. Wenn man diesen Effekt auf der Modelleisenbahn mit LEDs nachbilden will, muss die Helligkeit der LED langsam verändert werden. Das ist mit der analogen Pattern Funktion (APattern) möglich.

Aktiviert wird das Überblenden indem in das Feld „Analoges Überblenden:“ eine 1 eingetragen wird. In der „LED“ Tabelle definiert man dazu einen kurzen Bereich in dem die Helligkeit sich ändert und anschließend einen Bereich in der die LED konstant leuchtet. Am Ampel Beispiel sieht man das sehr

gut:

Spalte Nr ->	2	4	6	8	10	12	14	16	18
Rot 1	x	x	x	x					
Gelb 1									
Gelb 1									
Grün 1									
Rot 2	x	x	x	x	x	x	x	x	x
Gelb 2									
Gelb 2									
Grün 2									

Hier werden 200 ms zum Überblenden genutzt. Damit erreicht man einen sehr schönen Umschaltvorgang. Hier nochmal die „Dauer“ Zeile mit normaler Spaltenbreite:

Dauer	200 ms	2 Sec	200 ms	1 Sec	200 ms	10 Sec	200 ms	3 Sec
Flash Bedarf: 63 Bytes								
Spalte Nr ->	1	2	3	4	5	6	7	8

Zur Generierung der LED Tabelle oben wurde die Breite der Spalten an die Dauer angepasst. Dazu wird ein „D“ in das Feld „Grafische Anzeige:“ des Excel Programms eingefügt.

Das analoge Überblenden kann man aber auch für andere Anwendungen nutzen. Im Beispiel „16.Illumination_Pattern_Func“ wird damit ganz langsam von einer Farbe zur nächsten gewechselt.

Durch die freie Konfigurierbarkeit der Pattern Funktion gibt es noch viele andere Anwendungsbereiche.

XPatternT..

Der „X“ Pattern Befehl ist eine Erweiterung des „APattern“ Kommandos, der benötigt wird, wenn die Sequenz der dargestellten Zeilen nicht fest ist. Bei der Ampel ist fest vorgegeben, in welcher Reihenfolge die einzelnen Ampelphasen angezeigt werden. Erst ist die Ampel Rot, dann kommt Gelb dazu. Anschließend wird zu Grün gewechselt. Nach der Grünphase kommt eine kurze Gelbe Phase. Danach beginnt wieder der Rote Bereich. Es ist also in zu jedem Zeitpunkt bekannt, was die vorangegangene „Farbe“ der Ampel war. Daraus kann das Programm die Überblendfunktion berechnen.

Bei Anwendungen bei denen die Reihenfolge und der zeitliche Ablauf nicht konstant sind wird es komplizierter. Das ist zum Beispiel bei einem Lichtsignal der Fall. Hier soll das angezeigte Bild über einen von außen kommendem Befehl (z.B. über DCC) gesteuert werden. Das nächste Signalbild und der Zeitpunkt des Wechsels sind nicht bekannt. Darum kann die Übergangsfunktion nicht so einfach berechnet werden. Das Programm benötigt dazu einen zusätzlichen Speicher für jede LED in dem der alte Helligkeitswert abgelegt wird.

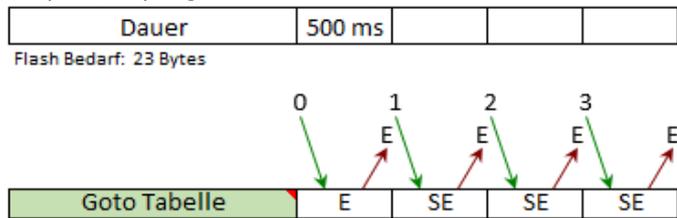
Bei einem „Großen“ Computer ist das kein Problem. Dort ist genügend RAM zur Speicherung der Daten vorhanden. Auf einem Arduino Uno/Nano stehen aber nur 2 KByte RAM zur Verfügung. Darum wird der Speicher nur dann reserviert, wenn er tatsächlich benötigt wird.

In der MobaLedLib wird der Speicher außerdem auf eine besondere Weise reserviert. Bei der klassischen C++ Programmierung verwendet man den „new“ Befehl während der Initialisierung zur Allokation von Speicher. Für einen Mikrocontroller ist das aber nicht immer optimal, weil man dann erst zur Laufzeit des Programms bemerkt, dass nicht genügend RAM zur Verfügung steht. Erschwerend kommt hinzu, dass ein Mikrocontroller normalerweise über keine Anzeige verfügt auf der Fehlermeldungen ausgegeben werden können. Darum wird in der Bibliothek ein vom Compiler

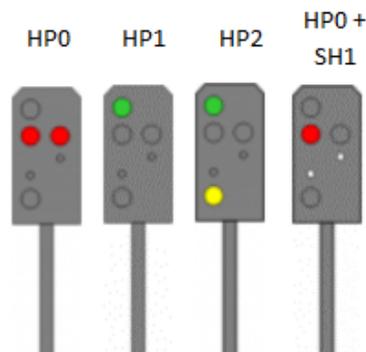
generiertes festes Array benutzt. Damit wird beim kompilieren des Programmes bereits eine Fehlermeldung am PC erzeugt falls der Speicher knapp wird.

Aktiviert wird die „XPattern“ Funktion im „Pattern_Configurator“ indem ein „X“ in das Feld „Grafische Anzeige:“ eingetragen wird.

Im Excel Beispiel „Dep Signal4“ wird die „XPattern“ Funktion benutzt:



LED Nr	Spalte Nr ->	1	2	3	4
1	Rot 1	x			x
2	Grün		x	x	
3	Gelb			x	
4	Rot 2	x			
5	Weiss				1



Hier erkennt man sehr schön, dass die Helligkeit der einzelnen Ausgänge zu Beginn einer Phase nicht bestimmt ist. Das wird durch die dreieckigen Helligkeitsverläufe visualisiert. Die erste rote LED in Spalte 1 kann zu Beginn Aus- oder Eingeschaltet sein. Sie kann sogar einen beliebigen Zwischenwert haben, abhängig davon, wann die vorangegangene Phase unterbrochen wurde. Am Ende von Spalte 1 leuchtet die LED aber. Doch warum leuchtet sie nicht mit voller Helligkeit? Das liegt daran, dass in diesem Beispiel die maximale Helligkeit über das Feld „Wert Max:“ auf 128 begrenzt ist.

Die Pfeile in der „Goto“ Tabelle deuten an, dass die einzelnen Phasen über Befehle von außen angesprochen werden können.

[?PatternTE..](#)

Die Pattern Funktion kennt noch eine weitere Betriebsart. Mit einem „E“ welches an den Namen der drei vorgestellten Pattern Funktionen angehängt wird (PatternTE., APatternTE.. und XPatternTE..) kommt ein zusätzlicher „Enable“ Eingang zu den Parametern hinzu. Mit diesem Eingang kann man die Pattern Funktion über eine zusätzlichen Eingangskanal ein- und ausschalten. Das ist dafür vorgesehen, dass man alle Geräuschausgaben mit einem gemeinsamen Schalter deaktivieren kann.

```
PatternTE1 ( LED,NStru,InCh,Enable,LEDs,Val0,Val1,Off,Mode,T1, ... )
:
```

```

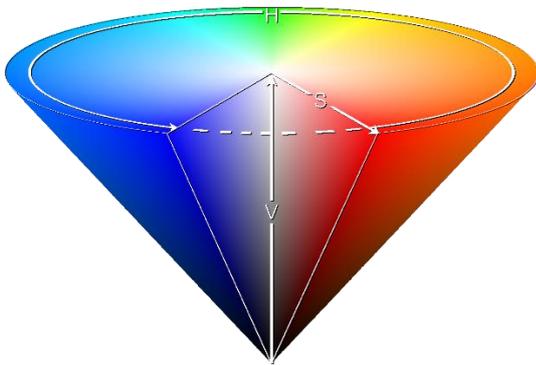
APatternTE1 ( LED, NStru, InCh, Enable, LEDs, Val0, Val1, Off, Mode, T1, ... )
:
XPatternTE1 ( LED, NStru, InCh, Enable, LEDs, Val0, Val1, Off, Mode, T1, ... )
:

```

7.3 HSV Mode der Pattern Funktion

In der Beschreibung zum Mode Parameter der Pattern Funktion wurde der Mode „PM_HSV“ kurz angesprochen (Siehe Seite 81). Hier soll dieser Modus etwas näher beschrieben werden.

Mit dem HSV Modus können die Farben der LEDs auf eine alternative Weise dargestellt werden. Dabei verwendet man die drei Komponenten „H“ = Farbwert (Englisch: Hue), „S“ = Sättigung (Englisch: Saturation) und „V“ = Helligkeit (Englisch: Value). Diese drei Komponenten werden gerne so dargestellt:



Der Vorteil dieser Transformation ist, dass man durch Verändern von nur einer Komponente die gewünschten Änderungen an der resultierenden Farbe erzielen kann. Mit der Änderung von „Hue“ können alle möglichen Farben erzeugt werden. Durch Variation der Sättigung kann eine Farbe bis zum Weiß übergeblendet werden. Mit dem dritten Parameter „Value“ kann eine Beleuchtung langsam ein- oder ausgeblendet werden. Damit lassen sich schöne „Illuminationen“ kreieren. Im Beispiel „16.Illumination_Pattern_Func“ wird dieser Farbraum benutzt.

Die FastLED Bibliothek, auf der die MobaLedLib aufbaut, benutzt für alle drei Parameter eine Zahl zwischen 0 und 255. Mit der analogen Pattern Funktion können diese drei Werte einzeln und langsam verändert werden.

Zur Umwandlung der HSV Farben in die von den LEDs benutzten RGB Werte benötigt das Programm einen Zwischenspeicher im RAM. Dieser Speicher wird mit dem Befehl „New_HSV_Group()“ angelegt. Er muss vor der Verwendung des HSV Modes in dem Konfigurationsarray verwendet werden. In dem Illumination Beispiel wird das folgendermaßen gemacht:

```

#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)

```

Hier werden die Parameter „Hue“ und „Value“ einer LED über das Makro „Illumination“ verändert. In dem Makro sorgt der „New_HSV_Group()“ Aufruf dafür, dass ein gemeinsamer Speicher für die beiden folgenden Zeilen zur Verfügung steht.

Mit dem Makro „Changing_Hue“ wird eine Funktion erstellt, mit der die Farbe einer LED langsam verändert werden kann. Diese Funktion entspricht eigentlich nur einer Pattern Funktion, die unter einem neuen Namen aufgerufen werden kann. Das erleichtert das Verständnis und stellt der aufrufenden Funktion nur die Parameter zur Verfügung, die tatsächlich benötigt werden.

Es wurde mit der dem „Pattern_Configurator“ folgendermaßen erstellt:

```

Makro: #define Changing_Hue(LED,InNr)          APatternT2(LED,192,InNr,1,0,255,0,PM_HSV,60 Sek,0 ms,1)
#define Changing_Hue_StCh(LED,StCh,InNr)     APatternT2(LED,StCh+192,InNr,1,0,255,0,PM_HSV,60 Sek,0 ms,1)
Wenn gleiche Zeiten verwendet werden, dann sollten nur die ersten Zeiten eingetragen werden. Bei leeren Spalten wer
Flash Bedarf: 16 Bytes


| Dauer  | 60 Sek       | 0 ms |  |  |  |  |  |  |  |  |  |  |
|--------|--------------|------|--|--|--|--|--|--|--|--|--|--|
| LED Nr | Spalte Nr -> |      |  |  |  |  |  |  |  |  |  |  |
| 1      | Hue          | X    |  |  |  |  |  |  |  |  |  |  |


```

und anschließend manuell erweitert, in dem die Periode als Parameter hinzugefügt wurde und das Flag „PF_SLOW“ und die Umrechnung (/16) ergänzt wurden.

Das Makro verändert den „Hue“ Wert langsam in der über „Periode“ angegebenen Zeit von 0 auf 255. Dann springt der Wert sofort (0 ms) zurück auf 0 und das Spiel beginnt von vorne.

Das zweite für die Illumination benötigte Untermakro ist „Pulsating_Val“. Es verändert die Helligkeit der LEDs langsam von 0 auf 255. Dort bleibt die Helligkeit dann für eine gewisse Zeit bis sie wieder langsam auf 0 reduziert wird.

```

Makro: #define Pulsating_Val(LED,InNr)        APatternT5(LED,98,InNr,1,0,255,0,PM_HSV|PF_EASEINOUT,50 m
#define Pulsating_Val_StCh(LED,StCh,InNr)    APatternT5(LED,StCh+96,InNr,1,0,255,0,PM_HSV|PF_EASEINOUT,50
Wenn gleiche Zeiten verwendet werden, dann sollten nur die ersten Zeiten eingetragen werden. Bei leeren Spalten wer
Flash Bedarf: 22 Bytes


| Dauer  | 50 ms        | 5 Sek | 2 Sek | 5 Sek | 10 Sek |  |  |  |  |  |  |  |
|--------|--------------|-------|-------|-------|--------|--|--|--|--|--|--|--|
| LED Nr | Spalte Nr -> |       |       |       |        |  |  |  |  |  |  |  |
| 1      | Val          |       | X     | X     |        |  |  |  |  |  |  |  |


```

In dem Excel Ausschnitt sieht man, dass zu Beginn und am Ende eine Pause vorgesehen ist. Auch hier wurden die Zeiten der einzelnen Abschnitte als Parameter des neuen Makros verwendet. Auf diese Weise kann es mehrfach mit unterschiedlichen Zeiten eingesetzt werden. Dadurch entsteht beim Betrachter eine sich scheinbar ständig verändernde Beleuchtungssituation.

Das Makro „Illumination()“ wird dann für jede LED mit geringfügig unterschiedlichen Parametern aufgerufen. Das wurde wieder in dem Makro „Illumination1()“ zusammengefasst:

```

#define Illumination1(LED0, InNr) /* 480 Byte FLASH, 209 Byte RAM */ \
/* LED InNr HuePeriod Delay RampTime HoldTime Pause*/ \
Illumination(LED0+0, InNr, 70 Sec, 0 Sek, 5 Sek, 2 Sek, 10 Sek) \
Illumination(LED0+1, InNr, 30 Sec, 6 Sek, 5 Sek, 2 Sek, 19 Sek) \
Illumination(LED0+2, InNr, 40 Sec, 11 Sek, 5 Sek, 4 Sek, 23 Sek) \
Illumination(LED0+3, InNr, 65 Sec, 24 Sek, 8 Sek, 2 Sek, 1 Sek) \
Illumination(LED0+4, InNr, 60 Sec, 30 Sek, 5 Sek, 2 Sek, 0 Sek) \
Illumination(LED0+5, InNr, 50 Sec, 33 Sek, 5 Sek, 3 Sek, 2 Sek) \
Illumination(LED0+6, InNr, 75 Sec, 28 Sek, 10 Sek, 2 Sek, 1 Sek) \
Illumination(LED0+7, InNr, 25 Sec, 4 Sek, 5 Sek, 5 Sek, 4 Sek) \
Illumination(LED0+8, InNr, 45 Sec, 13 Sek, 15 Sek, 9 Sek, 7 Sek) \
Illumination(LED0+9, InNr, 55 Sec, 21 Sek, 5 Sek, 3 Sek, 21 Sek) \
Illumination(LED0+10, InNr, 65 Sec, 38 Sek, 5 Sek, 2 Sek, 2 Sek) \
Illumination(LED0+11, InNr, 50 Sec, 1 Sek, 9 Sek, 2 Sek, 32 Sek)

```

Den Effekt den man damit erreichen kann, ist in dem Video schön zu sehen:

<https://vimeo.com/326218227>



Der erste Teil im Film zeigt das zeitlich verzögerte Einschalten der LEDs welches ebenfalls per Pattern Funktion gemacht ist, aber hier nicht näher vorgestellt werden soll.

8 Fehlersuche

Hier möchte ich erklären wie man Fehler in der Konfiguration verhindert, erkennt und behebt ...

9 Manuele Tests

Die Bibliothek enthält einige Funktionen mit denen man die angeschlossenen LEDs testen kann. Damit lassen sich einzelne LEDs ansteuern, die Farbe verändern und Performance Messungen durchführen.

10 Konstanten

Die Konstanten sollten auch noch ausführlicher beschrieben werden...

10.1 Konstanten für die Kanalnummer Cx.

Im Bild rechts ist ein WS2811 Modul für 12V gezeigt. Hier wurden Stecker für den Anschluss einzelner LEDs aufgelötet. Auf der oberen Seite der Platine wird eine zusätzliche Platine verwendet mit der der Pluspol auf alle drei Stecker verteilt wird.

Achtung: Bei anderen WS2811 Modulen kann die Anschlussbelegung abweichen.



Name	Beschreibung
C1 = C_RED	Erster Kanal eines WS2811 Moduls bzw. die Rote LED
C2 = C_GREEN	Zweiter Kanal eines WS2811 Moduls bzw. die Grüne LED
C3 = C_BLUE	Dritter Kanal eines WS2811 Moduls bzw. die Blaue LED
C12 = C_YELLOW	Erster und zweiter Kanal eines WS2811 Moduls
C23 = C_CYAN	Zweiter und dritter Kanal eines WS2811 Moduls
C_ALL	Alle drei Kanäle eines WS2811 Moduls (Weis)

10.2 Konstanten für Zeiten („Timeout“, „Duration“):

Name	Beschreibung
Min	Angabe in Minuten
Sec = Sek	Angabe in Sekunden
Ms = ms	Angabe in Millisekunden

Bei Minuten und Sekunden können auch Dezimalzahlen verwendet werden:

Beispiel: 1.5 Min

Die Zeiten können auch addiert werden:

Beispiel: 1 Min + 13 Sec

Wichtig ist der Abstand zwischen Zahl und Einheit.

10.3 Konstanten der Pattern Funktion

Name	Beschreibung
PM_NORMAL	Normal Mode (Als Platzhalter in Excel)
PM_SEQUENZ_W_RESTART	Rising edge-triggered unique sequence. A new edge starts with state 0.
PM_SEQUENZ_W_ABORT	Rising edge-triggered unique sequence. Abort the sequence if new edge is detected during run time.
PM_SEQUENZ_NO_RESTART	Rising edge-triggered unique sequence. No restart if new edge is detected
PM_SEQUENZ_STOP	Rising edge-triggered unique sequence. A new edge starts with state 0. If input is turned off the sequence is stopped immediately.
PM_PINGPONG	Change the direction at the end: 118 bytes
PM_HSV	Use HSV values instead of RGB for the channels
PM_RES	Reserved mode

PM_MODE_MASK	Defines the number of bits used for the modes (currently 3 => Modes 0...7)
_PF_XFADE	Special fade mode which starts from the actual brightness value instead of the value of the previous state
PF_NO_SWITCH_OFF	Don't switch of the LEDs if the input is turned off. Useful if several effects use the same LEDs alternated by the input switch.
PF_EASEINOUT	Easing function (Übergangsfunktion) is used because changes near 0 and 255 are noticed different than in the middle
PF_SLOW	Slow timer (divided by 16) to be able to use longer durations
PF_INVERT_INP	Invert the input switch => Effect is active if the input is 0

10.4 Flags und Modes für Random() und RandMux()

Name	Beschreibung
RM_NORMAL	Normal
RF_SLOW	Time base is divided by 16 This Flag is set automatically if the time is > 65535 ms
RF_SEQ	Switch the outputs of the RandMux() function sequential and not random
RF_STAY_ON	Flag for the Ranom() function. The Output stays on until the input is turned off. MinOn, MaxOn define how long it stays on.

10.5 Flags und Modes für die Counter() Funktion

Name	Beschreibung
CM_NORMAL	Normal Counter mode
CF_INV_INPUT	Invert Input
CF_INV_ENABLE	Input Enable
CF_BINARY	Maximal 8 outputs
CF_RESET_LONG	Taste Lang = Reset
CF_UP_DOWN	Ein RS-FlipFlop kann mit CM_UP_DOWN ohne CF_ROTATE gemacht werden
CF_ROTATE	Fängt am Ende wieder von vorne an
CF_PINGPONG	Wechselt am Ende die Richtung
CF_SKIPO	Überspringt die 0. Die 0 kommt nur bei einem Timeout oder wenn die Taste lange gerückt wird
CF_RANDOM	Generate random numbers
CF_LOCAL_VAR	Write the result to a local variable which must be created with New_Local_Var() prior
_CF_NO_LEDOUTP	Disable the LED output (the first DestVar contains the maximal counts-1 (counter => 0 .. n-1))
CF_ONLY_LOCALVAR	Don't write to the LEDs defined with DestVar. The first DestVar contains the number maximal number of the counter-1 (counter => 0 .. n-1).
_CM_RS_FlipFlop1	RS Flip Flop with one output (Edge triggered)
_CM_T_FlipFlop1	T Flip Flop with one output
_CM_RS_FlipFlop2	RS Flip Flop with two outputs (Edge triggered)
_CM_T_FlipFlopEnable2	T Flip Flop with two outputs and enable
_CM_T_FlipFlopReset2	T Flip Flop with two outputs and reset
_CF_ROT_SKIPO	Rotate and skip 0
_CF_P_P_SKIPO	Ping Pong and skip 0

10.6 Beleuchtungstypen der Zimmer:

Name	R	G	B	Beschreibung
ROOM_DARK	50	50	50	Raum mit dunkler Beleuchtung
ROOM_BRIGHT	255	255	255	Raum mit Heller Beleuchtung
ROOM_WARM_W	147	77	8	Raum mit Warm Weißer Beleuchtung
ROOM_RED	255	0	0	Raum mit hellem rotem Licht
ROOM_D_RED	50	0	0	Raum mit Dunklem roten Licht
ROOM_COLO				Raum mit offenem Kamin. Dieser erzeugt ein flackerndes rötliches Licht, das (hoffentlich) einem offenen Kamin ähnlich ist. Der Kamin brennt nicht immer. Ab und zu (Zufallsgesteuert) brennt auch eine normale Beleuchtung.
ROOM_COL1				Mit dieser Konstante wird das Flackern eines laufenden Fernsehgeräts simuliert. Dazu werden die RGB LEDs zufällig angesteuert. Wenn die Preiserlein mal nicht in die Röhre gucken, dann brennt ein normales Licht.
ROOM_COL2				In diesem Raum läuft manchmal der Fernseher oder es brennt der Kamin und ab und zu wird auch bei normalem Licht ein Buch gelesen.
ROOM_COL3				Mit diesem Typ wird ein zweites Fernsehprogramm simuliert. In unserer Modellwelt gibt es nur zwei verschiedene Fernsehprogramme. Diese sind aber immerhin schon in Farbe. Verschiedene Programme werden benötigt, damit bei benachbarten Fenstern unterschiedliches flackern zu sehen ist. Weitere TV Sender können im Programm mit dem Compilerschalter TV_CHANNELS aktiviert werden. Ein Down grade auf Schwarz/Weiß Fernsehen könnte im Programm ergänzt werden, falls das besser in die Epoche der Anlage passt.
ROOM_COL4				Wie ROOM_TV0_CHIMNEY nur mit dem ZDF.
ROOM_COL5				Room with user defined color 5
ROOM_COL345				Room with user defined color 3, 4 or 5 which is randomly activated
FIRE				Chimney fire (RAM is used to store the Heat_p)
FIRE_D				Dark chimney "
FIRE_B				Bright chimney "
ROOM_CHIMNEY				With chimney fire or Light (RAM is used to store the Heat_p for the chimney)
ROOM_CHIMNEY_D				With dark chimney fire or Light "
ROOM_CHIMNEY_B				With bright chimney fire or Light "
ROOM_TV0				With TV channel 0 or Light
ROOM_TV0_CHIMNEY				With TV channel 0 and fire or Light
ROOM_TV0_CHIMNEY_D				With TV channel 0 and fire or Light
ROOM_TV0_CHIMNEY_B				With TV channel 0 and fire or Light
ROOM_TV1				With TV channel 1 or Light
ROOM_TV1_CHIMNEY				With TV channel 1 and fire or Light

Name	R	G	B	Beschreibung
ROOM_TV1_CHIMNEYD				With TV channel 1 and fire or Light
ROOM_TV1_CHIMNEYB				With TV channel 1 and fire or Light
Die Programmfunktion, die die Häuser steuert, wird auch von dem Makro „GasLight()“ benutzt, welches im nächsten Abschnitt beschrieben wird. Die folgenden Konstanten sind dafür vorgesehen. Sie simulieren Gaslampen welche erst langsam die volle Helligkeit erreichen und ab und zu flackern. Diese Lampen können natürlich auch in einem Haus benutzt werden.				
GAS_LIGHT	255	255	255	Gaslaterne mit Glühbirne welche zwischen 20mA und 60mA bei 12V verbraucht. Die Lampe wird mit voller Helligkeit angesteuert.
GAS_LIGHT1	255	-	-	Gaslaterne mit LED welche am ersten Kanal (Rot) eines WS2811 Chips angeschlossen ist.
GAS_LIGHT2	-	255	-	Gaslaterne mit LED welche am zweiten Kanal (Grün) eines WS2811 Chips angeschlossen ist.
GAS_LIGHT3	-	-	255	Gaslaterne mit LED welche am dritten Kanal (Blau) eines WS2811 Chips angeschlossen ist.
GAS_LIGHTD	50	50	50	Gaslaterne mit Glühbirne welche zwischen 20mA und 60mA bei 12V verbraucht. Die Lampe wird mit reduzierter Helligkeit angesteuert.
GAS_LIGHT1D	50	-	-	Dunkle LED an Kanal 1
GAS_LIGHT2D	-	50	-	Dunkle LED an Kanal 2
GAS_LIGHT3D	-	-	50	Dunkle LED an Kanal 3
NEON_LIGHT				Neon light using all channels
NEON_LIGHT1				Neon light using one channel (R)
NEON_LIGHT2				Neon light using one channel (G)
NEON_LIGHT3				Neon light using one channel (B)
NEON_LIGHTD				Dark Neon light using all channels
NEON_LIGHT1D				Dark Neon light using one channel (R)
NEON_LIGHT2D				Dark Neon light using one channel (G)
NEON_LIGHT3D				Dark Neon light using one channel (B)
NEON_LIGHTM				Medium Neon light using all channels
NEON_LIGHT1M				Medium Neon light using one channel (R)
NEON_LIGHT2M				Medium Neon light using one channel (G)
NEON_LIGHT3M				Medium Neon light using one channel (B)
NEON_LIGHTL				Large room Neon light using all channels. A large room is equipped with several neon lights which start delayed
NEON_LIGHT1L				Large room Neon light using one channel (R)
NEON_LIGHT2L				Large room Neon light using one channel (G)
NEON_LIGHT3L				Large room Neon light using one channel (B)
SKIP_ROOM				Room which is not controlled with by the house() function (Useful for Shops in a house because this lights are always on at night)

11 Offene Punkte

Da spuken immer noch 1000 Ideen in meinem Kopf herum, die man noch besser machen könnte, aber dann wird die Bibliothek niemals fertig und Ihr habt nichts davon ;-)

Darum werde ich sie so wie sie ist veröffentlichen!

Hier ein paar Punkte welche unbedingt noch bemacht werden müssen:

- Dokumentation erweitern / fertig machen
- „Pattern_Configurator“ verbessern
- Programme zum Ansteuern von Servos, Schrittmotoren und Multiplex LEDs
- Programm Code aufräumen
- Bilder / Videos erstellen und veröffentlichen
- ...