MobaLedLib: Ein kurzer Überblick

Inhaltsverzeichnis

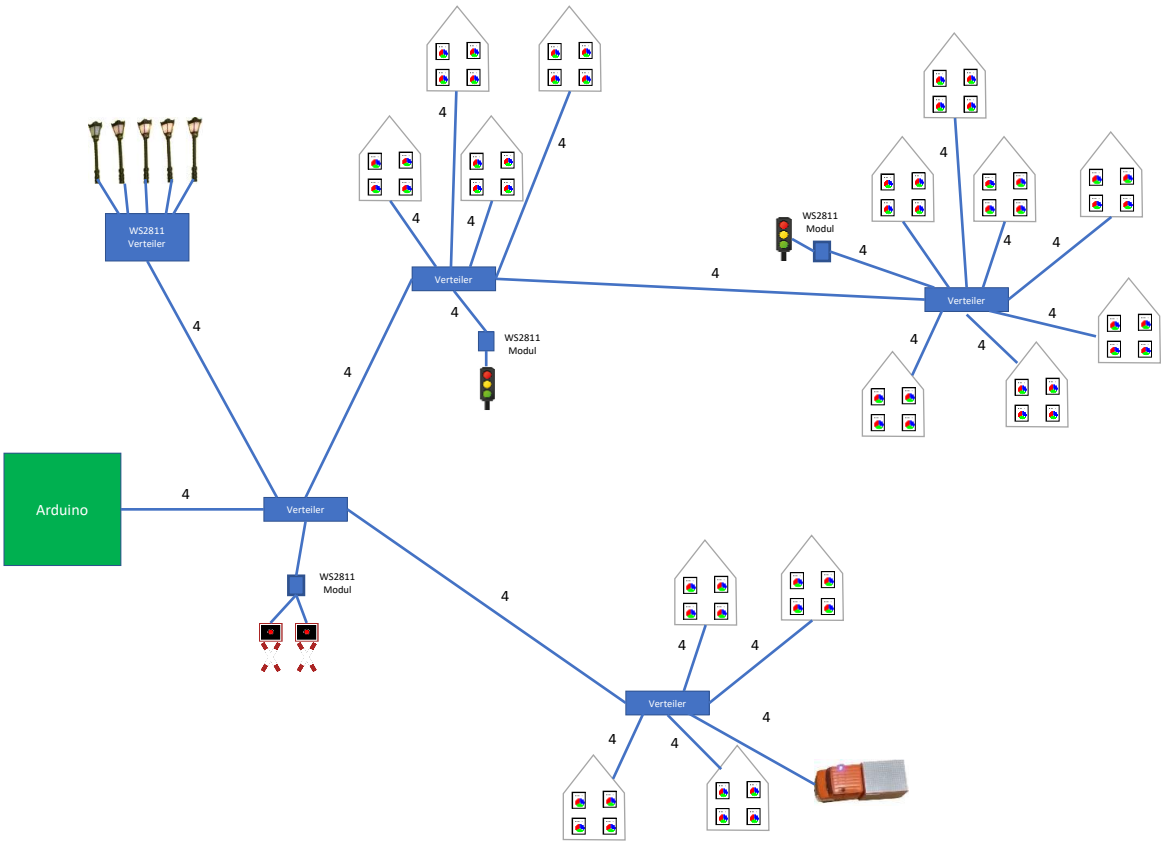
1 Einleitung

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 **Modeleisenbahn** 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 **einzigen Signalleitung** ist auch die Verkabelung extrem einfach. Die Verbraucher werden über **4-polige Flachkabel** in Verteilerleisten gesteckt welche beliebig kaskadier bar sind.



So können z.B. alle Zimmer in einem Modelhaus 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 jedes Raums 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 welche 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 Bahnhofsansagen, Eisenbahngeräusche (Glocke an Bahnschranke), Tierlaute, Kirchturmglocken 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 den **CAN Bus**.

Zum einfachen Einstieg enthält die Bibliothek sehr **viele Beispiele** welche 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 das 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 welche Kommas getrennt sind. Ein Strichpunkt darf hier nicht vorkommen.

Die Makros generieren **konstante** Datenbytes welche 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 sind 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 welche per „#define“ angelegt sind.

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 Servo Motoren, 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 ober 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 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 maximal Wert.

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 17 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

2.3.1 RGB_Heartbeat(LED)

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

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

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

2.3.3 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)

2.3.4 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.

2.3.5 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)

2.3.6 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 und 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 31 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 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).

2.4.3 BlinkerInvInp(LED,Cx,InCh,Period)

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

2.4.4 BlinkerHD(LED,Cx,InCh,Period)

Eine weitere Variante des Blinkers. Hier wechselt die Helligkeit der LED zwischen „Hell“ und „Dunkel“.

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

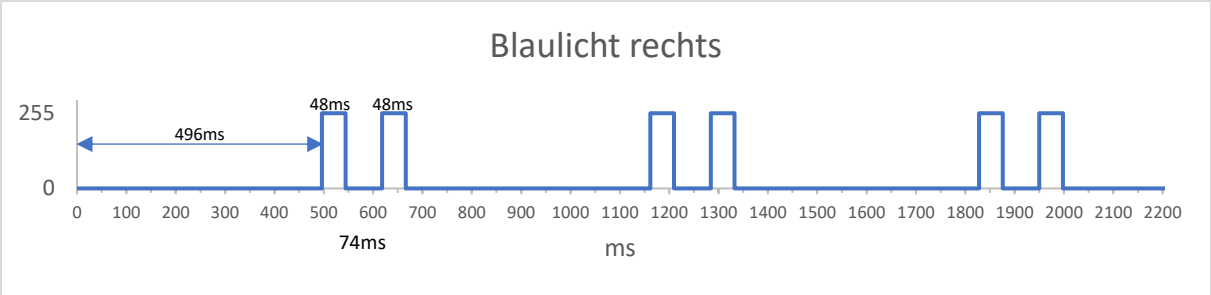
Bei dieser Variante kann die Dauer der beiden Phasen und die Helligkeit in den Phasen angegeben werden. „Pause“ bestimmt die Zeit in der „Val0“ ausgegeben wird und „Act“ die Zeit während dessen „Val1“ verwendet wird.

2.4.6 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).

2.4.7 BlueLight1(LED,Cx,InCh)

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



2.4.8 BlueLight2(LED,Cx,InCh)

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

2.4.9 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_befeuerung_von_windkraftanlagen_.html)

2.4.10 LeuchtfeuerALL(LED,InCh)

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

2.4.11 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.

2.4.12 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.

2.4.13 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“).

2.4.14 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.

2.4.15 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 gegenüber-

liegenden 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.

2.4.16 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 22). 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 7 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)

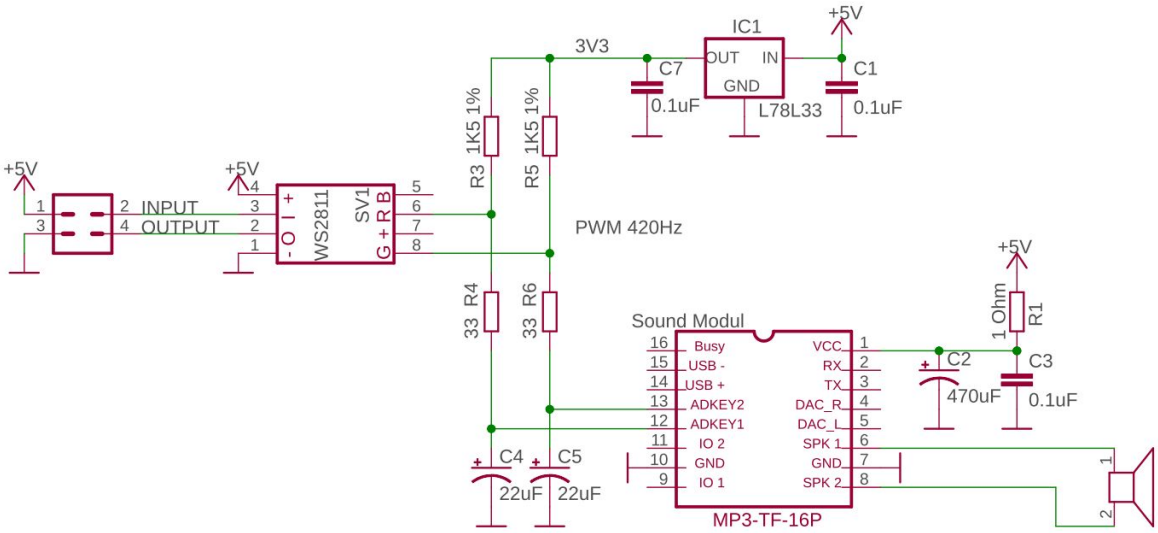
Mit dieser Funktion wird das Schweißlicht zufällig gesteuert. Die Zeiten „MinTime" und „MaxTime" bestimmen den Zufälligen Startzeitpunkt. Über „MinOn" und „MaxOn" wird angegeben wie lange eine die Arbeit an einem Werkstück dauert. Der Parameter „Var" enthält die Nummer einer Zwischenvariable welche zur 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

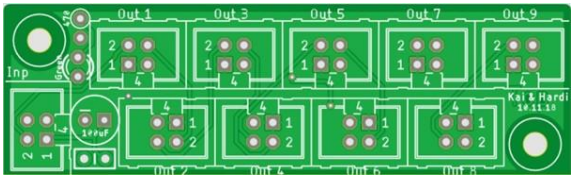
Die MobaLedLib Bibliothek können zu den Lichteffekten passende Geräusche wiedergegeben werden. So kann zum Beispiel das Leuten der Schranke mit dem Blinken des Andreaskreuzes wiedergegeben werden. Dazu wird ein zusätzliche Soundmodul vom Typ MP3-TF-16P benötigt:

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 Soundwidergabe nicht benötigt.

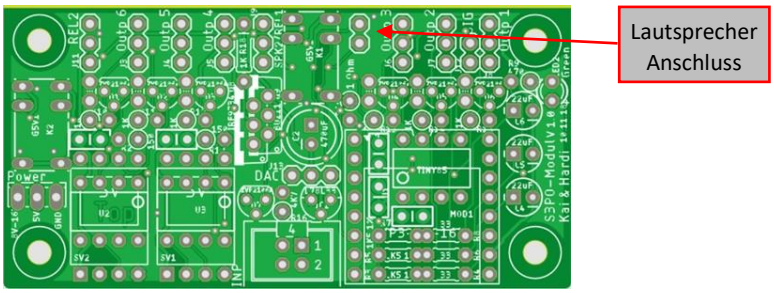
Die Platine besteht aus drei Teilen welche durch knicken getrennt werden können. Die beiden oberen Teile (Im Bild unten ist nur einer gezeigt) enthalten jeweils einen 9-fach Verteiler für den Anschluss der LEDs und anderer auf dem WS281x basierender Komponenten:



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 das oben gezeigten Schaltplans bestückt werden.



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

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 widergegeben. 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 Widergabe 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 Tastendrücke 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 dieser SD-Karte 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.

Die Dateien werden über simulierte „Tastendrücke“ 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

```
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>
```

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 Widergabe 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 Modis 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 veränderrung 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 verrigellung 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.

2.7 Steuer Befehle

Mit den in diesem Abschnitt beschrieben 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 8 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(DstVar, InCh, Duration)

Dieses Makro entspricht einem Treppenhaus Lichtschalter. Die Ausgangsvariable „DstVar" 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 wen die Raum Beleuchtung 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, DISBALE 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_SKIP0	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 22 und „2.7.26 Use_GlobalVar(GlobVarNr)" auf Seite 23)
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 Zustande 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 wenn der Eingang länger als 1.5 Sekunden aktiv ist kombiniert.

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 welche 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 verschieden 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 welches 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 13).

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 ist die „RandWelding()“ Funktion auf Seite 13.

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, wenn 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 Zustande (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 20) benutzt (Siehe Seite 23).

Ausgewertet wird die Variable mit den Pattern Funktionen (ab Seite 31).

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 Kompiler 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 Kompilers 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 Logische 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 das genau gleiche zeigen sollen, dann ist das parallel schalten eine Alternative. Im Beispiel „TrafficLight_Pattern_Func“ ist das skizziert.

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

Intern ist das Makro folgendermaßen definiert:

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

Es definiert ein Array aus „unsigned char“ welche im Bereich „PROGMEM“ stehen welcher 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 währe.

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 Kompiler 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“ welcher 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__“ welches 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 „unit8_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 Kompiler 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 zu 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.

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 welches 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 welche 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 „InpStrucktArray“ 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“ welches auf Seite 26 beschrieben ist eingesetzt werden.

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 must be enabled in "Lib_Config.h"
```

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.

Die 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 welche über die Konstante „LED_BUILDIN“ an die Klasse übergeben wird. Bei den Beispielen welche den CAN Bus nutzen geht das nicht, weil der Pin der internen LED gleichzeitig als Taktgenerator für den SPI Bus genutzt wird. Hier muss eine externe 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 welche im Hintergrund automatisch eine Matrix welche aus sehr vielen Schaltern bestehen kann abfragt. 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 Wiederstände 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:

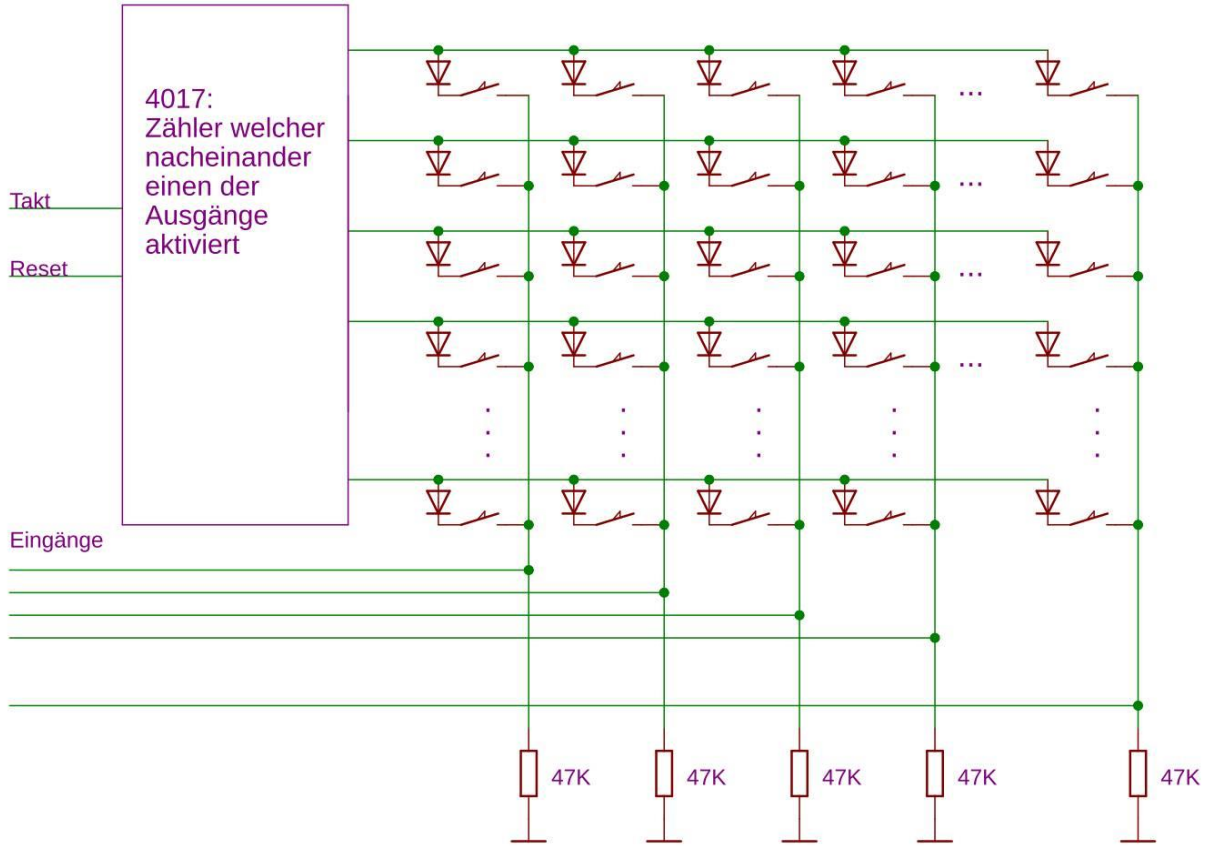
(IC Anzahl * 8 + 2) * (Signalleitungen – 2)

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 BUTTON_INP_LIST_1 2,7,8,9,10,11,12,A1
#define CTR_CHANNELS_2    18
#define BUTTON_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 „BUTTON_INP_LIST_1“ enthält eine Liste mit acht Eingangs Pin Nummern. Damit besteht diese Gruppe aus 80 Schaltern.

Die zweite Gruppe wir mit der Konstante „CTR_CHANNELS_2“ und „BUTTON_INP_LIST_2“ parametriert. Hier sind zwei Zähler ICs verwendet welche 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 in welcher der Schaltplan und die entsprechende 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.

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.

5 CAN Message Filter

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

Aber jetzt ist genug geschrieben. 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 vier poligen Steckern welche einfach in Verteilerleisten gesteckt werden können ist die Beleuchtung einer Komplexen Anlage sehr einfach. In diesem Abschnitt wird das näher beschrieben …

Bilder…

7 Details zur Pattern Funktion

Die Pattern Funktion ist unglaublich leistungsfähig. Mit ihr können die komplexesten Animationen sehr einfach konfiguriert werden. Dieser Abschnitt wird das erklären …

7.1 Die verschiedenen Pattern Befehle

```
PatternT1( LED,NStru,InCh,LEDs,Val0,Val1,Off,Mode,T1, ...)
:
PatternT20(LED,NStru,InCh,LEDs,Val0,Val1,Off,Mode,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T
14,T15,T16,T17,T18,T19,T20,...)

APatternT1( LED,NStru,InCh,LEDs,Val0,Val1,Off,Mode,T1, ...)
:
APatternT20(LED,NStru,InCh,LEDs,Val0,Val1,Off,Mode,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,
T14,T15,T16,T17,T18,T19,T20,...)

XPatternT1( LED,NStru,InCh,LEDs,Val0,Val1,Off,Mode,T1, ...)
:
XPatternT20(LED,NStru,InCh,LEDs,Val0,Val1,Off,Mode,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,
T14,T15,T16,T17,T18,T19,T20,...)

PatternTE1( LED,NStru,InCh,Enable,LEDs,Val0,Val1,Off,Mode,T1, ...)
:
PatternTE20(LED,NStru,InCh,Enable,LEDs,Val0,Val1,Off,Mode,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T
12,T13,T14,T15,T16,T17,T18,T19,T20,...)

APatternTE1( LED,NStru,InCh,Enable,LEDs,Val0,Val1,Off,Mode,T1, ...)
:
APatternTE20(LED,NStru,InCh,Enable,LEDs,Val0,Val1,Off,Mode,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,
T12,T13,T14,T15,T16,T17,T18,T19,T20,...)

XPatternTE1( LED,NStru,InCh,Enable,LEDs,Val0,Val1,Off,Mode,T1, ...)
:
XPatternTE20(LED,NStru,InCh,Enable,LEDs,Val0,Val1,Off,Mode,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,
T12,T13,T14,T15,T16,T17,T18,T19,T20,...)
```

7.2 New_HSV_Group()

7.3 Pattern_Configurator

Mit diesem Abschnitt wird erklärt wie das Excel Programm „Pattern_Configurator.xlsm“ verwendet wird…

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_SKIP0	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_SKIP0	Rotate and Skip 0
_CF_P_P_SKIP0	

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_COL0				Raum mit offenen Kammin. Dieser erzeugt ein flackerndes rötliches Licht welches (hoffentlich) einem offenen Kamin ähnlich ist. Der Kammin 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 Compiler-schalter TV_CHANNELS aktiviert werden. Ein Downgrade auf Schwarz/Weiß Fernsehen könnte

Name	R	G	B	Beschreibung
				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)
FIRED				Dark chimney "
FIREB				Bright chimney "
ROOM_CHIMNEY				With chimney fire or Light (RAM is used to store the Heat_p for the chimney)
ROOM_CHIMNEYD				With dark chimney fire or Light "
ROOM_CHIMNEYB				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_CHIMNEYD				With TV channel 0 and fire or Light
ROOM_TV0_CHIMNEYB				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
ROOM_TV1_CHIMNEYD				With TV channel 1 and fire or Light
ROOM_TV1_CHIMNEYB				With TV channel 1 and fire or Light
Das Programmfunktion welche 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)

Name	R	G	B	Beschreibung
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 was 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 und Schrittmotoren
- Programm Code aufräumen
- Bilder / Videos erstellen und veröffentlichen
- …