MobaLedLib: Short Overview

Contents

This document has been translated from German to English by
https://www.onlinedoctranslator.com/de/

This sometimes generates funny results. Unfortunately, also some of the keywords have been changed. If in doubt, the German version should be consulted

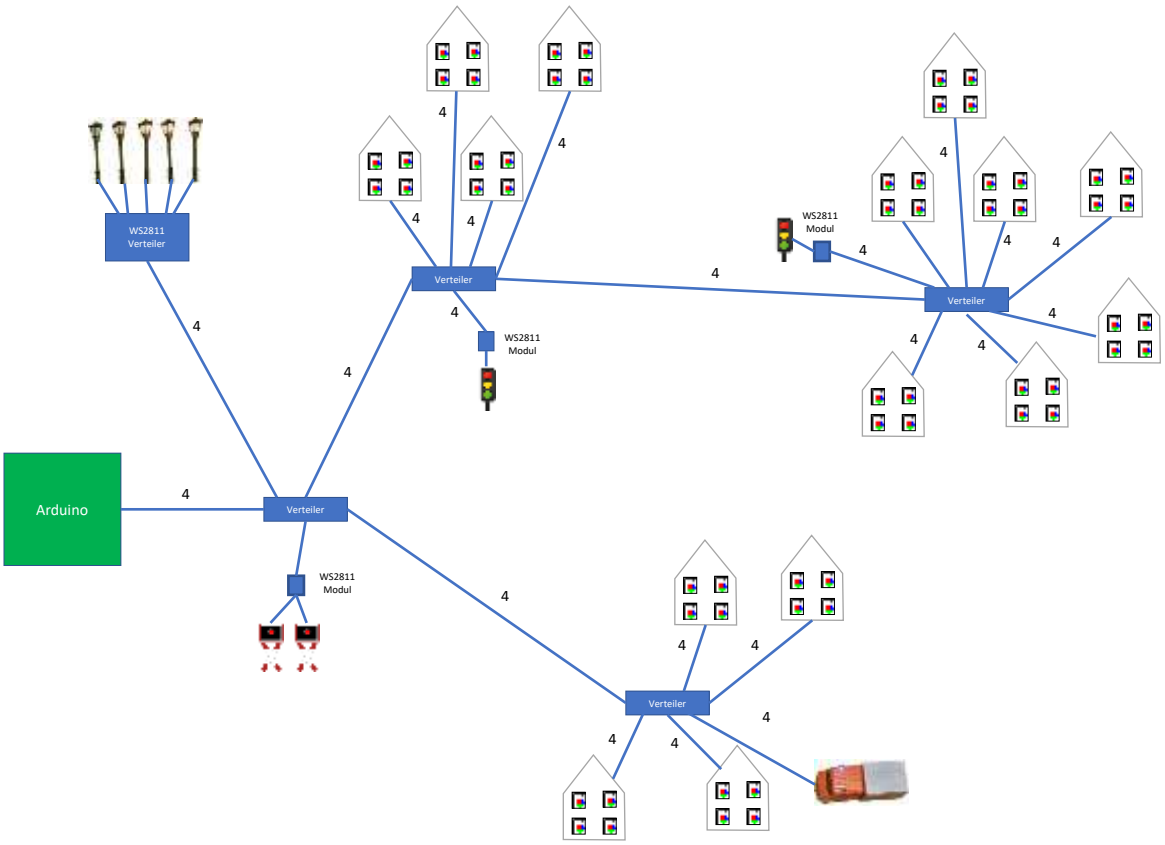
1 Introduction

This document describes the MobaLedLib for the Arduino. With the library up to **768 LEDs** and other loads can be controlled by a single signal line from an Arduino.

The library is intended for use on a **model railway**. Some functions of the library can surely be used in other applications.

For controlling the LEDs and other consumer devices chips based on the **WS2811 / WS2812** are used. Since these ICs cost only a few cents (7-12 cents) very cheap and also very flexible lighting can be realizing at a railroad.

By controlling everything with a **single signal line**, the wiring is extremely simple. Consumers are connected via **4-pin ribbon cable** and inserted in **distribution strips** which could arbitrarily cascaded.



All rooms in a model house could be equipped with its own RGB LED while the whole house is connected with a 4-pin connector on the distributor. In this case, each room could be individually turned on and off. In addition, the brightness and the color of each compartment can be adjusted. This also allows effects such as a TV or fireplace simulations.

In addition to the houses, there are a variety of other lights on a model system that can be controlled with this library. These are, for example, light signals, Andrew's crosses, traffic lights, flashing emergency vehicles, site protection, disco or fair effects, ...

The "Single wire Concept" can also be used to control several **sound modules** around the installation. The corresponding sound module with a matching SD card is available for two euros. So, the station

announcements, railway noise (at railroad crossing), animal sounds, church bells and more can then be played.

The method is also suitable for controlling of **moving components**. By means of a small additional circuit, the signals for driving **servo** or **stepper motors** can be generated.

With a transistor to amplify solenoids or DC motors can also be used.

The effects can be controlled **automatically** or **manually** with or without a computer.

The library contains a module with is able to read **80 and more switches** using just a few ports of the Arduino.

It also supports the import of commands via the **CAN bus**.

To get started easily, the library contains **many examples** which show clearly how the individual functions are used. So, the system can also be used **without programming skills** and adapted to your needs.

1.1 Using a configuration array

The library is adjusted via an array to the individual circumstances. For this purpose, various commands are available which define what the program should do. The keyword "House()" is used to define the number of rooms a building has and how many of them should be illuminated in average. In addition, the type of illumination (brightness, color of light, lamp type, ...) and other effects like a TV set could be configured.

Internally, this information is stored in a byte array. This method was chosen that it is possible later to create a configuration using a Graphical User Interface. However, the input via a text editor is so easy and in addition so flexible that a GUI is actually not necessary.

The use of a configuration array also requires minimal memory (FLASH) and can be processed quickly. The configuration commands also ensure that the RAM required is already provided when compiling the program. Thus, the memory usage is already fixed on program start and is monitored by the compiler. A sophisticated dynamic memory management is eliminated. On a microcontroller such as the Arduino very little memory is available. Therefore, the program must be very economical with it.

To these internal details, however, the users of the library does not have to worry about.

2 Configuration macros

This section describes the configuration macros only briefly. A detailed documentation is omitted because it no one reads it ...

If there is a need for further documentation, write to MobaLedLib@gmx.de.

Suggestions, bug fixes, ... are also welcome.

To configure C++ macros are used. This allows a certain input validation be made without using program storage.
These macros consist of a name on which follow several parameters. The parameters are set in parentheses.

Note: After the configuration macro, different than usually in C++, there is no tailing semicolon. This is because the macros are "only" an array of bytes which create a comma separated list. A semicolon is not allowed here.

The macros generate constant data bytes which are stored in the configuration array. This array is read by the program to generate the lighting effects. Since this array is stored in FLASH of Arduinos all data must be constant. Therefore, it is not possible to assign variable macros. Calculations using constants, however, are allowed.

Below the term "macro", "function" or "command" is used alternately. This serves to loosen up the dry document. Actually, they are C++ macros which are created by the "#define" statement.

2.1 Commonly used parameters

First, the parameters used in the following macros are described so that they need not be explained in any of the commands.

2.1.1 LED

Contains the number of LEDs in the string. All LEDs are so connected in series that the output of the first LED is connected to the input of the next LED. This method is used for addressing the individual LEDs. Internally, each LED uses the first three brightness values which it receives via the signal line for controlling the three colors red, green and blue. All following values are forwarded to the output. Therefore, the second LED in the series just gets the data from the second record. It uses the first three brightness values for himself and gives the following on to the next. So, the colors of each LED can be individually controlled.

The WS2811 chips are not integrated into the RGB LEDs as the next generation of modules (WS2812). Thus, the WS2811 ICs are suitable for controlling individual LEDs as they are for example used in a street lamp. One IC can control three outputs which could be three lanterns. From the perspective of the program these three street lights have the same LED number. The individual lamps are addressed by the channel number (Cx) which is described in the next section.

The WS2811 modules could also be used to control servo motors, sound modules or other actuators. Nevertheless, the following documentation always used the term "LED" for the number in the string.

2.1.2 Cx

The Cx parameter describes the channel number an RGB LED or a WS2811 module. Here one of the constants is entered:
C1, C2, C3, C12, C23, C_ALL, C_RED, C_GREEN, C_BLUE, C_WHITE, C_YELLOW, C_CYAN

2.1.3 InCh

Many of the effects can be switched on and off. The parameter "InCh" describes the number of the input. There are 256 different inputs possible. As an input channel can be a switch or a special function. It is also possible to receive the input via the CAN bus of a model railway.

2.1.4 VAL0

Contains the brightness or general the duty cycle of the output when the input is switched off. The parameter is a number between 0 and 255 where 0 is the minimum value (LED dark), and 255 to the maximum value.

2.1.5 val1

Is contains the value that is used when the input is turned on. See "VAL0".

2.1.6 TimeOut / Duration

Includes the time after the counter is reset to zero or the output is disabled. The time is expressed in milliseconds and can be supplemented with an attached "Sec" or "Min". The maximum time is 17 minutes.

2.2 Variables / Input Channels

Most macros have an input "InCh" with is used to activate or deactivate the function. The parameter "InCh" contains the number of the desired input channel. This number refers to one of 256 variables. These variables can be set in the main program with the command „Set_Input()“. In the example "Switched_Houses" it will be shown how this can be done:

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

It is recommended that a symbolic name is used instead of the number. This can be defined at the beginning of the program with the "#define" command.

```
#define INCH_HOUSE_A 0
```

Overlaps can be prevented if all definitions are in one place.

The input variables can also be set by other macros. In the section "2.7 Commands" on page 16 describes the commands which could be used for that purpose.

The variables can be either 0 or 1. The library in addition stores the last state of the variable. This is used to detect whether the variable has changed. Many of the actions in the library will only be performed if the relevant input changes. This saves a lot of computing time.

The variables from number 240 are reserved for special functions. At the moment, the following are specific input variables defined (SI = Special Input):

2.2.1 SI_Enable_Sound

With this input variable, the sound may be globally switched on and off. It is initialized at the program start to 1, but it can be changed by the program.

2.2.2 SI_LocalVar

This variable is used in the Pattern function if the start value is to be read from a local or global variable. The variable to be declared by one of the commands "New_Local_Var()", "Use_GlobalVar()" or "InCh_to_TmpVar()" are.

2.2.3 SI_0

This special input variable is always 0. This variable is needed for example when the "Reset" input of the counter function is not used.

2.2.4 SI_1

If a function is to be active, this variable can be used. It is always set to 1.

2.3 Single and multiple lights

2.3.1 RGB_Heartbeat (LED)

RGB LED with slowly changing and flashing colors for monitoring the program health.

2.3.2 Const (LED, Cx, InCh, VAL0, Val1)

LED which is controlled by "InCh". It’s permanently on or off.

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

This is probably the most frequently used function on a model railway. With it a "lively" House is simulated. In this house some of the rooms are randomly illuminated. The color and brightness of the lighting can be adjusted individually. It is possible to configure a flicker of fireplace for individual rooms also certain effects such as TV. In addition, the switch-on behavior can be adjusted (neon lights flickering or slow brightening gas lamps).

The parameter "On_Min" describes how many rooms should be at least illuminated. After turning on as the lights are turned on after a random time until the predetermined number is reached. The activated rooms are also determined randomly.

The parameter "On_Limit" determines how many chambers should be used simultaneously. If a corresponding number of LEDs are reached a lamp is turned off to the next randomly selected, time. If this parameter is greater than the number of rooms, then all the lights are on after some time (This corresponds to our home).

Now it's enough, the library has to go out before Christmas. I will finish the manual adjustments to the English translation in the next version. Have fun with the machine translation ...

If the houses will have a manually operated switch turned on and off, the user will see direct feedback when pressing the switch. Therefore, immediately when switched on the input (InCh) is activated lighting and according to a deactivated when the switch is OFF.

The three dots "..." in the macro definition represent the position at which the list of room lighting is entered. You can specify up to 2,000 rooms (castle).
The lighting of the room is set with the following constants:
Colors / Brightness:
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

Animated effects:
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

Special lamps:
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

Unused space:
SKIP_ROOM

Example: **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, ...)

Corresponds to the House() macro. Here are two additional parameters "MIN_T" and "Max_T" can be specified. These numbers describe in seconds how long it takes randomly until the next change occurs. In the "House) (" function these times of global definitions

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

specified.

2.3.5 Gas Lights (LED, InCh, ...)

Street lights are an important part of a virtual city. They illuminate the streets at night to create a warm atmosphere especially when it comes to gas lanterns. These lamps were initially ignited in the real world by man and later watches or light sensors. This is described here very nicely: http://www.gaswerk-augsburg.de/fernzuendung.html, the lanterns go to not simultaneously by different times or lighting conditions. I observe time and again on my way home. The lights go on at random and grow brighter gradually until they reach full brightness. This behavior also have the lamps are

controlled via which the macro "(Gas Lights)". Here is also still a random flickering implemented which can be caused by fluctuations in the gas pressure or by wind gusts. They are controlled by the lamps WS2811 chip. In light bulbs all three outputs are connected in parallel with LED lamps IC controls three lamps. In the configuration, the order ..1., ..2., ..3 needs. be used as shown in the example below. This ensures that the program sequentially uses the outputs of a WS2811 chips and does not change to the next channel. The attached "D" in the example below means "Dark". These lamps light darker than the others.

Example: gas Lights(Gas_Lights1, 67, GAS_LIGHT1D, GAS_LIGHT2D, GAS_LIGHT3D, GAS_LIGHT)

2.3.6 Set_ColTab (red0, green0, Blue0 ... Red14, Green14, Blue14)

The macro "(Set_ColTab)" you can adjust the color and brightness of the lamps individually. For this, a list of 15 RGB values is specified. The command can be used multiple times in the configuration and affects all following "House()" or "Gas Lights()" line.

Here is an example of the command:

```
//          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 Sequential events

This section describes functions which represent temporal sequences. Sequential events come and reality and of course on a model train frequently. One example is the traffic light. Here the corresponding traffic light phases are sequentially displayed. Several lamps must be switched coordinates for this representation.

In the library, these controls are "(Pattern)" from the function generated. Such sequences can be generated using the Excel program "Pattern_Configurator.xlsm". the chapter 5 on page 28 it's further described that.

2.4.1 Button (LED, Cx, InCh, duration, VAL0, Val1)

This macro saves a button for a certain time. This will enable our railway smoke generator in the "Burning" house. The output can be deactivated before the expiration of the time when the button is pressed a second time.

The duration determines the duration for which the output is activated when the button is pressed. The time is specified in milliseconds and may be between 16 ms and 17 minutes (1,048,560 ms). When the time is to be specified in seconds or minutes, then "Sec" or "Min" can be written behind the value. Here, the upper and lower case and at least one space to the previous number of important (for example, 3.5 min). A combination of minutes, seconds and milliseconds is also possible. Example: 3 min + 2 sec + 17 ms

Example: **button**(10, C_ALL, 0, 3.5 min, 0, 255)

2.4.2 Indicators (LED, Cx, InCh, Period)

Implements a turn signal at a predetermined period. The period is specified in milliseconds. Here, too, can "Sec" or "Min" is appended as the "Button()" function. The maximum period is two minutes (131070 ms).

2.4.3 BlinkerInvInp (LED, Cx, InCh, Period)

Corresponds to the "flasher) (" function. In this case, the turn signal is then activated when the input channel (InChes) off.

2.4.4 BlinkerHD (LED, Cx, InCh, Period)

Another variation of the turn indicator. Here changes the brightness of the LED between "bright" and "dark".

2.4.5 Blink2 (LED, Cx, InCh, pause, Act, VAL0, Val1)

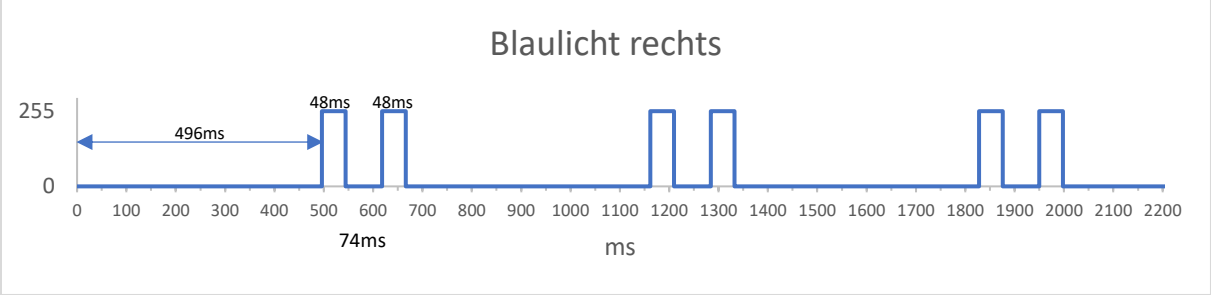
In this variant, the duration of the two phases and the brightness in the phases can be specified. "Pause" defines the time in the "VAL0" is output and "Act" is the time during which "Val1" is used.

2.4.6 Blink3 (LED, Cx, InChes, pause, Act, VAL0, Val1, Off)

A third variant of the "Flash()" function. Here, the brightness value can also be specified in the off state (Off).

2.4.7 BlueLight1 (LED, Cx, InCh)

This function generates the typical double flash of blue light on emergency vehicles.



2.4.8 BlueLight2 (LED, Cx, InCh)

Blue light having a slightly different period. By the use of two blue lights of slightly different period, a more realistic effect.

2.4.9 Leuchtfeuer(LED, Cx, InCh)

This macro generates the flashing pattern of a wind turbine. The light is one second, then half a second off and then back for a second. This is followed by a pause of 1.5 seconds. (Please referhttps://www.windparkwaldhausen.de/contentbeitrag-170-84-kennzeichnung_befeuerung_von_windkraftanlagen_.html)

2.4.10 LeuchtfeuerALL (LED, InCh)

This beacon, all three channels are used. This corresponds to the "Leuchtfeuer()" command with the parameter "C_ALL".

2.4.11 Andrew's Cross (LED, Cx, InCh)

For controlling the alternate the flashing lights in St. Andrews crosses this function can be used. "Cx" determines the first channel used. This Blinks alternately to the following channel. LED brightness changes slowly so that the typical "soft" flashing occurs.

2.4.12 AndreaskrRGB (LED, InCh)

This function uses two RGB LEDs for simulating the St. Andrew's Cross. , Only the red LED is activated in each case. This macro is only intended for testing purposes with a LED stripe.

2.4.13 RGB_AmpelX (LED, InCh)

Thus, the patterns of two lights is generated, each with 3 RGB LEDs for an intersection. This macro is only intended for testing purposes with a LED stripe. On the model railway traffic lights will be employed with individual LEDs which are then activated via a WS2811 module.

In the Excel program "Pattern_Configurator.xlsm" which is located in the library some sites are describing the configuration of the traffic lights ("AmpelX" .. "RGB_AmpelX_Fade_Off").

2.4.14 RGB_AmpelXFade (LED, InCh)

Another example for the control of traffic lights with RGB LEDs. Here the lights are slowly fades which generates a more realistic impression. On the "RGB_AmpelX_Fade" in "Pattern_Configurator.xlsm" as expanding is shown schematically works.

2.4.15 AmpelX (LED, InCh)

This light is designed for use on site. So that individual LEDs over two WS2811 modules are controlled. In the example, "09.TrafficLight_Pattern_Func." If one finds a "circuit diagram" to do so. To secure a crossing 4 traffic lights are needed. The opposing light signals always show the same pattern. For driving the oppositelying traffic light can be used another WS2811 chip which receives the same input signal as its counterpart. In the example shown schematically.

can arbitrarily complicated traffic light installations with multiple lanes and pedestrian traffic lights and be configured with the program "Pattern_Configurator.xlsm".

2.4.16 AmpelXFade (LED, InCh)

This is the same function as above except that here the lights slowly in and be Hidden.

2.5 Random effects

This section describes some random effects.

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

The "Flash()" function generates a random flashes of a photographer. Passed to the "MinTime" and "MaxTime" is determined how often the flash fires. The first parameter determines to be maintained at least until the next flash how long. Accordingly, "MaxTime" describes the maximum time. Between these two periods, the library determines a random time.

The macro consists of two different macros. "(Const)" and the "random()" macro (See "2.7.24 **Random**(DstVar, Inch, fashion, MinTime, MaxTime, Minon, Maxon)" on page20) .Dabei is the "Const()" function from the "Random()" control function. To an intermediate variable needed their number as a parameter "Var" is entered. This is one of the 256 input variables which in the chapter "2.1.3 InCh" on page 7have been described. Attention this intermediate variable may be nowhere where used differently.

Example: **Flash**(11, C_ALL, SI_1, 200, 5 sec, 20 sec)

2.5.2 Fire (LED, InCh, LedCnt, brightnes)

"(Fire)" function can be simulated larger fire. For this, several RGB LEDs are used which at different points of "fire" light. In our system, the function to simulate a "Burning" house is used.

2.5.3 Welding (LED, InCh)

"(Welding)" function can be simulated a welding light. This light flickers a while bright white and then goes out for a while. After welding, the "weld" afterglow red briefly. This function should be controlled by a higher-level function ("The worker also wants to take a break").

2.5.4 RandWelding (LED, InCh, Var, MinTime, MaxTime, MINON, Maxon)

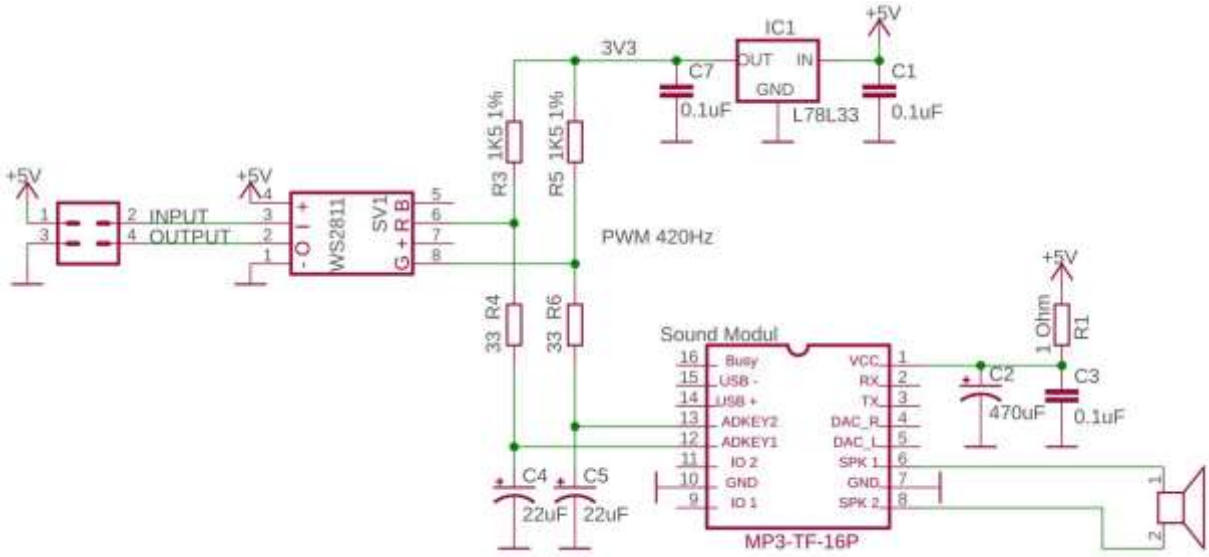
This feature allows the welding light is controlled by chance. The times "MinTime" and "MaxTime" determine the accidental start time. About "Minon" and "Maxon" indicates how long a work takes on a workpiece. The "Var" parameter contains the number of an intermediate variable that for controlling the "(Welding)" function is used. Attention this intermediate variable may be nowhere where used differently.

Example: **RandWelding**(12, SI_1, 201, 1 min, 3 min, 50 sec, 2 min)

2.6 sound

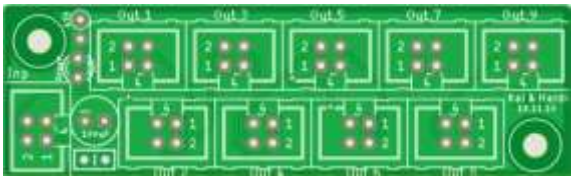
The library can MobaLedLib matching sounds are played back to the light effects. For example, the people of the barrier can be reproduced with the flashing of the St. Andrew's Cross. For this, a sound additional module is required using MP3 TF-16P:

This module is available for a Euro in China. Additionally you need an SD card and a speaker. With this module, MP3 and WAV files can be played via a built-in 3 watt amplifier. Normally, the MP3 files are about 20 switches which are connected via different resistances to the module played. The sound module can also be controlled via the same signal line as the light-emitting diodes by simulating keystrokes. For this, a WS2811 chip is used. This must be connected with a couple of resistors and capacitors for filtering the signals. The following diagram shows the structure:



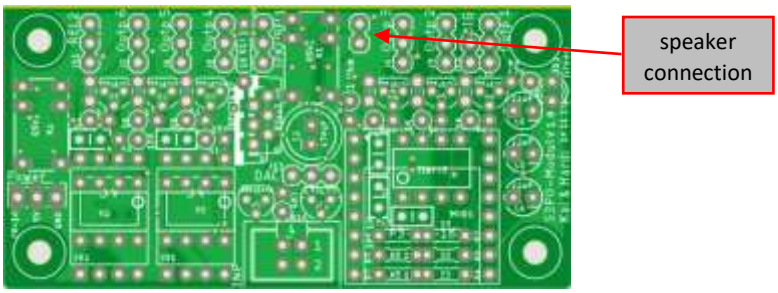
The zip file S3PO_Modul_WS2811.zip in the "extras" includes a schematic and a board with this construction. On the circuit some more components are provided with which higher power can be switched via a WS2811 module. In addition, you can head to the circuit servo or stepper motors. However, these are not required for sound reproduction.

The board is composed which can be separated by buckling of three parts. The two upper parts (bottom only one is shown) each containing a 9-way distributor for connecting the LEDs and other components based on the WS281x:



This distribution can be arbitrarily placed at central positions of the plant. In marked "Out ..." plug the houses, traffic lights, sound modules and other consumers are infected. Here even other distribution boards can be connected. The labeled "inp" connector is connected to the Arduino or a previous distribution board via a ribbon cable. Caution: When unused slots, the pins must be bridged with a jumper 2 and 4 or the corresponding solder jumper to be connected to the rear.

The next picture shows the board for the sound module. only the components must be fitted the wiring diagram shown above.



Important: There are some solder connectors on the rear panel. For the function of the sound module without the other components must be SJ1 and SJ21 connected.

The following are the commands for sound output are briefly introduced.

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

To play certain sounds, there are 14 commands in the library. The command "Sound_Seq1()", the first MP3 or WAV file on the SD card will resist given. Accordingly, the second file is "(Sound_Seq2)" to play ...

According to the documentation of the sound module to MP3 or WAV files must have specific names and are stored in specific subdirectories. This seems to apply only to the playback via the serial interface. If the files are played via buttons, then that's not necessary. This also applies to the presented circuit simulating the keystrokes on a WS2811 module. The file name does not matter. "(Sound_Seq1)" command will always play the file which was copied first to the SD card. Hence the name "..Seq ...". When a file is deleted and then a new file is copied to the card, then the new file is entered in place of the deleted file in the directory on the card. This can be very confusing. Unfortunately, the Windows Explorer displays the files on an SD card to always sorted. There is no option with which you can disable the sorting entirely. To check the order you have to open a

command window (cmd.exe) and "f you" type (The drive letter "f" may have to actually acknowledged thebe adapted to SD card reader):

In this SD card, the "001.mp3" file will be played, when the input of "Sound_Seq1()" command is activated. 6. The entry "Muh.wav" output "(Sound_Seq6)" with the macro.

The files will be played over simulated "keystrokes" which are encoded via different resistances. Unfortunately, the distances of the resistors in the upper area are relatively small so that it can be an overlap "(Sound_Seq14)" by component tolerances or temperature variations in the sound

```
C: \ Users \ Hardi> dir f:
 Volume in drive F: has no name.
 Volume Serial Number: A87B-A154

 Directory of F: \

18.03.2018 12:58 51,471 001.mp3
30/01/2018 21:04 5564 005.wav
10.03.2018 21:29 18,143 007.mp3
25/07/2015 16:04 2284950 018.mp3
10.03.2018 21:31 612667 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
10.03.2018 10:29 19,640 S1-bd.mp3
10.03.2018 10:30 39.168 S1-huehner.mp3
10.03.2018 10:30 19,562 S1-kapelle.mp3
10.03.2018 10:31 26,710 S1-kirche.mp3
10/03/2018 21:28 30867 S1-schafe.mp3
10.03.2018 21:29 35,091 S2Voegel.mp3
10/03/2018 21:44 3703998 voegel-in-forest-with-bachlauf.mp3
10.03.2018 21:42 48,109 motorcycle-start mit.mp3
10.03.2018 21:41 368,265 motorcycle start-leerlauf.mp3
10/03/2018 21:39 218 219 fireworks-short with-heuler.mp3
10.03.2018 21:38 51,034 sparrow sparrow-twitters-3.mp3
10.03.2018 21:38 23,867 bird-brown owl-eule.mp3
10.03.2018 21:36 62,738 bird voice-spatz.mp3
              21 file (s), 8,343,862 bytes
               0 folder (s) 116,072,448 bytes free

C: \ Users \ Hardi>
```

commands "Sound_Seq13()" and.

The output of sounds can be globally switched on and off via the "SI_Enable_Sound". In this way, you can disable a switch all the noise.

2.6.2 Sound_PlayRandom (LED, InCh, MaxSoundNr)

With this command, a Random file between 1 and "MaxSoundNr" will resist given when the input of the command is activated. This can be used for the announcement of station announcements, for example.

2.6.3 Sound_Next_of_N (LED, InCh, MaxSoundNr)

This macro file the next of between 1 and "MaxSoundNr" will resist given when the input of the command is activated. This can be used for playing the hourly ringing of church bells, for example.

2.6.4 Sound_Next_of_N_Reset (LED, InCh, INReset, MaxSoundNr)

This command corresponds to the previous one. Here there is also the possibility of a further input "INReset" to reset the counter.

2.6.5 Sound_Next (LED, InCh)

This command uses an internal function of the sound module with which the next sound file can be output. The command is not limited to the 14 selectable by "keys" files as the previous commands. In this way, all files can be played. However, a specific limitation of the scope is not possible. This can be pre-made on the selection of files on the SD card.

2.6.6 Sound_Prev (LED, InCh)
This command corresponds to the "Sound_Next()" command. With this, the previous file is played on the SD card with each pulse.

2.6.7 Sound_PausePlay (LED, InCh)
So the playback can be paused and resumed.

2.6.8 Sound_Loop (LED, InCh)
This command is evidence of the origin as a music player. Thus, all songs can be played on an SD card in order. For the railroad could use the function possibly with special sound files with long pauses.

2.6.9 Sound_USDSPI (LED, InCh)
I did not understand this function. It activates the button which, according to the "extensive" documentation of the sound module between "V / SD / SPI" switches.

2.6.10 Sound_PlayMode (LED, InCh)
If the "Loop" mode is active, you can switch to this "key" to the "Play Mode". I have not quite understood. There seems to be following modes:
"Sequence", "Sequence", "Repeat same", "Random", "Loop off"
How and whether the two "Sequence" modes differ in the beginning I do not know.

2.6.11 Sound_DecVol (LED, InCh, Steps)
The playback volume of the sound module is set to the maximum value after switching on. This is quite loud thanks to the built-in 3 watt amplifier. The "(Sound_DecVol)" macro, the volume can be reduced. The parameter "Steps" indicates the number of steps for reducing the volume. The number may be in the range 1 to 30 To veränderrung the volume the corresponding "key" must be held longer than one second. Then the volume every 150 ms is reduced by one step. The appropriate timing assumes the macro. But remember, no further instruction to the sound module may be sent is changed while the volume. An automatic verrigellung been omitted for space reasons. Unfortunately, the module does not remember the last setting. This means that the volume after each switch needs to be reset. Possibly. it is better if you "quieter" converts the sound files with a suitable program.

2.6.12 Sound_IncVol (LED, InCh, Steps)
Thus, the volume can be increased again.

2.7 Commands
With this section describes commands one or more variables are set. These variables can then be read by other macros and evaluated. As described in "2.2 Variables / Input Channels" on page 8described there are 256 input variables. From the perspective of the functions presented here are output variables. They are referred to in the parameter list with "DestVar". At this point in the macro, the number of variables come. It is recommended that a symbolic name is used instead of the number. This can be defined at the beginning of the program with the "#define" command.

2.7.1 ButtonFunc (DstVar, InCh, Duration)
This macro corresponds to a stairwell light switch. The output variable "DstVar" is one if the "Inch" is active. The output remains after the input has been disabled active for the "duration" milliseconds. The macro corresponding to a static, retriggerable monoflop. The time can be specified by appending of "Sec" or "Min". The maximum time is 17 minutes.

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

The "Schedule" Macro a timetable for turning on and off several lights can be created. This plan is, however, provide only rough conditions. When the outputs are actually switched the program determines at random so that a real impression.

are connected the output variables "DstVar1" to "DstVarN". They are turned on randomly between the time "start" and "end" when it is "Evening" and just as random off again on "Tomorrow". Whether it's "Evening" or "Morning" determines the global variable "DayState". It is "evening" to "SunSet", and "Morning" to "SunRise". The second variable "Darkness" determined by a number between 0 and 255 as "dark" it is. So that it represents the time.

The time can be generated in different ways. The easiest timer is the brightness. So that the lights can be turned on when it is dark. It can be measured with a light dependent resistor (LDR). This method allows a very simple and credible control. The advantage of this method is that the lights are automatically switched to match the lighting in the room. In the example, "Darkness_Detection" this method is used. In the example, it is also shown how to create a switch "day" and "night".

but it is also possible to receive the time from an external model railroad timer. You can, for example read via the CAN bus. This is useful whom the room is controlled illumination over the same time.

Of course, the model time can also be generated over a few lines in the program. This is demonstrated by "Schedule". Here, the value of the "darkness" of the time is derived.

In addition to the "darkness" ("Darkness") will be the day state ("DayState") for the "Schedule()" function required. Both are global variables that must be set accordingly. The examples show how this can be done.

2.7.3 Logic (DstVar, ...)

"(Logic)" function can be implemented logical links. With it more input variables "NOT", "AND" and "OR" are linked together and written to the output variable "DstVar". The logic must as a Disjunctive Normal Form (DNF) to be expressed. In this representation, groups of "AND" linkages with "OR" are combined:

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

The brackets are not displayed in the DNF (Implicit parenthesis). In macro that looks like this:

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

The input variables A to D must be defined previously:

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

The "Logic()" function can also have a second parent control input to which the link can be switched on and off:

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

or

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

The example "Logic" shows the use of "Logic()" function.

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

The "Counter()" function can be used for a variety of tasks. It is controlled by a "mode" parameters. The following flags are defined. Several flags can with the or operator, | ' be linked.

CM_NORMAL normal ZählerCF_INV_INPUT INCH is inverted
CF_INV_ENABLE Enable invert input
CF_BINARY Binary counter, otherwise the outputs are one by one
aktiviertCF_RESET_LONG reset active when input longer than 1.5 seconds istCF_UP_DOWN
Up / down counter (enable => reverse)
CF_ROTATE Counter starts again at the beginning when the end is reached
wurdeCF_PINGPONG Counter switches at the ends of the RichtungCF_SKIP0 Skips
NullCF_RANDOM Random count for each pulse at EingangCF_LOCAL_VAR
Count is written to the predefined local variable
(please refer "2.7.25 New_Local_Var()" on page 20 and
"2.7.26 Use_GlobalVar (GlobVarNr)" on page 21)
CF_ONLY_LOCALVAR The count is only written to the local variable. There are no
other outputs used. The number after "TimeOut" contains the number of
Counter levels (0 ... N-1).

The "..." in the functional description represent a variable number of output channels. Here, the numbers of the variables used are entered. These variables are other macros as input. In normal mode, the outputs are activated sequentially. When the CF_BINARY flag is specified, then the outputs, such as the individual bits of a binary number to be controlled.

The following are some macros are presented which are based "(counter)" on the macro.

2.7.5 Monoflop (DstVar, InCh, Duration)

A monoflop is a function which enables the output for a certain time when a change from zero to one (rising edge) is detected at the input. The time period is extended with each other edge, but if the input is not continuously active.

2.7.6 Long-shot reset (DstVar, InCh, Duration)

Is a mono recording flop of reset if the input is longer than 1.5 seconds active. The duration may, as in the previous function can be extended.

2.7.7 RS_FlipFlop (DstVar, S_InCh, R_InCh)

A flip-flop can accept two states (0 or 1). In an RS flip-flop, the states two inputs are determined. A positive edge at "S_InCH" sets the flip-flop (output = 1), an edge at "R_InCh" clears the flip-flop (output = 0).

2.7.8 RS_FlipFlopTimeout (DstVar, S_InCh, R_InCh, timeout)

This macro is equivalent to the previous ones. Here also exists that determines when the flip-flop is automatically deleted a "Timeout" parameter.

2.7.9 T_FlipFlopReset (DstVar, T_InCh, R_InCh)

The output of a "toggle flip-flop" is switched on every rising edge of input. This feature has a "reset" input to the flip-flop can be set to zero in addition. If this input is not required, it can be subject to "SI_0".

2.7.10 T_FlipFlopResetTimeout (DstVar, T_InCh, R_InCh, timeout)

Has an additional parameter "Timeout".

2.7.11 MonoFlopInv (DstVar, InCh, Duration)

This monostable multivibrator has an inverse output. That is, the output is at the beginning of active (1) and is deactivated with a positive edge of "InCh". The time can be extended as in normal MF with a rising edge.

2.7.12 MonoFlopInvLongReset (DstVar, InCh, Duration)

Here are the characteristics of inverse and reset if the input is longer than 1.5 seconds actively combined.

2.7.13 RS_FlipFlopInv (DstVar, S_InCh, R_InCh)

This flip-flop is active at the beginning.

2.7.14 RS_FlipFlopInvTimeout (DstVar, S_InCh, R_InCh, timeout)

Here, the "timeout" parameter is coming back to it.

2.7.15 T_FlipFlopInvReset (DstVar, T_InCh, R_InCh)

And another flip-flop inverse starting value.

2.7.16 T_FlipFlopInvResetTimeout (DstVar, T_InCh, R_InCh, timeout)

That's the last flip-flop with an output.

2.7.17 MonoFlop2 (DstVar0, DstVar1, InCh, Duration)

The following macros have two outputs the inverse are connected to each other. The functions are the same as in the foregoing, it is omitted here detailed description.

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, fashion, MinTime, MaxTime)

The "RandMux()" function is activated at random one of the outputs to which "DstVarN" are defined by the numbers "DstVar1". About "MinTime" is determined how long an output is minimally active. "MaxTime" describes the analog maximum time which the output should remain active. The program determines between the two corner points of a random point in time at which a random other channel is activated. This function can be switched, for example between different lighting effects for a disco.

With the flag "RF_SEQ" as "Mode" parameter of the next output is not selected chance, but the outputs are activated sequentially.

2.7.24 Random(DstVar, Inch, fashion, MinTime, MaxTime, Minon, Maxon)

The "Random()" function activates an output after a random period. The duty may also be random. So you can have an effect randomly taxes. One application of this is the flash of a Fotografens to which from time to flash.

The parameters "MinTime" and "MaxTime" is determined in which the temporal area of the function should be active. About "Minon" and "Maxon" the duration is specified. is used for the flash here "Minon" = "Maxon" = 30 ms used (See "2.5.1 Flash (LED, Cx, InCh, Var, MinTime, MaxTime) on page 12).

The flash is defined as a macro:

```
  #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)
```

Another example of using the "RandWelding()" function on page 13,

currently the random() function knows a special mode: RF_STAY_ON. With this switch the output remains as long until the predetermined about "Minon" and "Maxon" time has elapsed, if the input is no longer active. This is where "Button()" macro used which, like "Flash()" and "RandWelding()", the "random()" function uses:

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

2.7.25 New_Local_Var()

Most functions of MobaLedLib be controlled by one of the 256 logical variables. The number of the variable used as a parameter "InCh".

There are also cases where more than two states (on / off) are needed. For this purpose, no fixed number of variables are made available because the RAM of an Arduino is very limited in the library though.

The macro "(New_Local_Var)" creates a variable of type "ControlVar_t" if necessary. You can take values between 0 and 255 and has additional flags with which changes can be detected. This variable can then be set by a function and are evaluated by one or more other features. Through this

approach valuable RAM is only used when it is actually needed. Additionally, the user does not have to as the "Flash()" or "RandWelding()" function to an intermediate variable care.

To set the local variable of "Counter()" command (Page 18) Used (See page21).

the variable is evaluated (with the Pattern functions from page28).

The example "RailwaySignal_Pattern_Func" shows how this is done.

The memory for the variable is created automatically. For this is not as usual in C ++ using the "new" command, but when compiling the array "Config_RAM []" in the required size static created. This has the advantage that the RAM requirement is known at compile time already, and the compiler may optionally generate alerts if the memory is running out. In this case the following message in the Arduino IDE is shown:

```
Sketch uses 20,388 bytes (66%) of program storage space. Maximum is 30,720
bytes.

Global variables use 1,556 bytes (75%) of dynamic memory, leaving 492 bytes for
local variables. Maximum is 2048 bytes.

Low memory available, stability problems May Occur.
```

The warning appears because very little RAM memory for the program is left. This memory is needed in the C ++ program for dynamically created variables and for the stack. It is not possible to determine from the point of view of the compiler how much memory is required to do just that. Therefore, the memory in the library is statically reserved.

2.7.26 Use_GlobalVar (GlobVarNr)

The library can be extended with custom functions in the C ++ program. With the function "(Use_GlobalVar)" can exchange data their own parts of the program with the library's internal functions. Global variables can be used just like the local variables are created which the function "(New_Local_Var)". The global variables are, however, stored in a separate array. This array must be defined in the user's Sketch:

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

And the library in the "setup()" function will be announced:

```
MobaLedLib_Assigne_GlobalVar (GlobalVar);
```

This allows "(Use_GlobalVar)" then the command can be used. To set the variable, for example, this function can be added to the sketch of the user:

```
// ---------------------------------------- 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 (First InChes InCh_Cnt)

This command creates a temporary 8-bit variable is filled with the values of several logical variables. In contrast to the "New_Local_Var) (" function and "Use_GlobalVar()" function here requires no additional memory. This is possible because the same memory can be used multiple times. In the two other cases must be saved if the input changed because only then will an action triggered. With this function, the change of the input variables is used.

In the example, "CAN_Bus_MS2_RailwaySignal" the "InCh_to_TmpVar()" function is used.

2.8 other commands

2.8.1 CopyLED (LED, InCh, SrcLED)

"(CopyLED)" command is the brightness of the three colors of the "SrcLED" copied to the "LED". This is useful, for example, at a traffic light at an intersection. Here is the Opposite lights should show the same picture.

When two RGB LEDs should show the same, then you can reach the well through the electrical wiring. For this, the "DIN" are parallel lines of both LEDs. Usually all LEDs are in a chain so that lined up each LED can be individually addressed. If two LEDs are intended to show precisely the same, then the switch is an alternative parallel. In the example, "TrafficLight_Pattern_Func" is outlined.

3 Macros and functions of the main program

The following macros and functions are in the main program, the .ino file (or on Arduinisch "Sketch") is used.

The macros were introduced so that the sample programs are clearer and easier to maintain. In the macro, the actual name is separated by an underscore "_" from the leading "MobaLedLib".

The macros and functions must be in certain places within the program. This is described in the documentation of the individual elements.

3.1 MobaLedLib

The library contains several classes that can be used individually. In the next section, the class main class "MobaLedLib" is described.

3.1.1 MobaLedLib_Configuration()

This macro initiates the configuration of the LEDs. In the configuration is defined as the LEDs and other modules to behave. The configuration area in Braces, enclosed and finished with a semicolon "{...}". In the last line of the configuration, the "EndCfg" keyword should be.

The macro is in the main program after the #include "MobaLedLib.h". Here, the configuration of the example of "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 All which is
   // | | always on
   // | | On_Min: minimum number of active rooms. At least two rooms are
   // | | | illuminated.
   // | | | On_Max: Number of maximum 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
  };
// ********************************************** ******************
```

Internally, the macro is defined as follows:

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

It defines an array of 'unsigned char' which are in the "PROGMEM" which is located in the FLASH Arduinos. Without the "PROGMEM" the array would be copied into RAM what Währe possible due to the small memory of a Arduinos only for small configurations.

3.1.2 MobaLedLib_Create (leds)

This macro the class "MobaLedLib" is generated. So that the memory is allocated and initialized. The parameter "leds" tells the class where the brightness values of light emitting diodes are stored. This is to array of type "CRGB" which in the "FastLEDs" library is defined.

The macro is in the main program after the definition of "leds" Arrays:

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

MobaLedLib_Create (LEDs); // Define the MobaLedLib instance
```

In addition to the memory for the LEDs, the library needs memory for each configuration lines. For this purpose, a special method was used. The memory is allocated by each entry in the configuration when compiling. Usually this is done for the duration of the program with the command "new". but the "usual" approach has the disadvantage that the compiler does not know how much RAM is needed in the operation. RAM can at the scarce resource which quickly lead to a microcontroller to problems. That is why a different approach was used here. This does using the example of "House()" function can be explained:

```
#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__,
```

Decisive here is the term "RAMH" which describes the memory requirements of a house. He will be replaced by a further macro by RAM2. This macro has the following structure:

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

After that, "RAM 2" is composed of two "RAM1" together. The latter is the ultimate macro. It contains the C ++ preprocessor macro "__COUNTER__" which represents a counter which is incremented by 1 each time when it is used.

When first using the macro "RAM1" situation is as follows: 1 +*0* - *1*
This makes "RAM1" = 0. In the second use of the macro situation is similar: 1 + *2* - *3*
Which is also the 0th Thus, even "RAM2" and accordingly "RAMH" as 0. But is crucial that "__COUNTER__" has been changed. With each use of "RAM1", the counter is increased by two. And that's what is used to declare the configuration memory "Config_RAM []". This is an array of type "unit8_t":

```
uint8_t Config_RAM [__ __ COUNTER / 2];
```

With the "__COUNTER__" trick the array is the same size that it can provide the required memory for all configuration lines. Since the preprocessor macros are evaluated at compile the compiler knows exactly how much memory is in use and can check if the RAM of the processor it is sufficient. If a critical value is exceeded, then this warning:

```
Sketch uses 20,388 bytes (66%) of program storage space. Maximum is 30,720
bytes.

Global variables use 1,556 bytes (75%) of dynamic memory, leaving 492 bytes for
local variables. Maximum is 2048 bytes.

Low memory available, stability problems May Occur.
```

In this way, the main memory can be monitored without the need for a program line is needed. In addition, the warning is displayed during the creation of the program on the screen. An error message which is generated by the Arduino to maturity can not be reliably indicated the absence of standardized output device.

The macro "MobaLedLib_Create()" contains the line for the generation of the array and the actual initialization of the class:

```
#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)

If "(Use_GlobalVar)" in the configuration, the macro is used, then the library needs to know where the global variables are and how many are available.

The function is in the "setup()" function of the main program called. The corresponding array must be declared before:

```
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)

the MobaLedLib used to control the LEDs is an array with logic values which additionally stores the previous value. This library can detect whether an input has changed and only trigger the appropriate action.

If the input signals from the main program in the so-called "InpStruct" to be fed, then used this function is used. This will in the example "Switches_80_and_more" used to copy the keyboard array.

As an input, the function expects an array of individual bits representing the individual input channels. The parameter "Src" contains this array. The parameter "BYTECNT" indicates how many bytes are to be copied. "ChannelNr" indicates the target position in the input structure "InpStrucktArray". This corresponds to the "InCh" in the configuration macros. his respect for the "ChannelNr" divisible by 4.

In the program this macro is used function ") (loop" in the.

If only individual bits are to be copied to the input structure of the library can to the command "Set_Input()" are used (See section 3.1.6)

3.1.5 MobaLedLib.Update()

This is the crucial function of MobaLedLib. It calculates the states of the LEDs in each main loop pass new. She works one by one all entries in the configuration array, and thus determines the color and brightness of each LED. In developing the library great emphasis was placed thereon, that the function is processed very quickly, even with large configurations.

The function must be called in the "loop()" function of the Arduino program.

3.1.6 MobaLedLib.Set_Input (uint8_t channel, uint8_t On)

This function allows an input variable of MobaLedLib can be set. Thus the library switch settings or other input values can be supplied. The parameter "channel" is the number of input variables is described which is used in the configuration macro always "InCh".

The function "(loop)" in the function and, if appropriate, in the "setup()" function of the program used.

It is used in almost all the examples to read the switch. In the example, "CAN_Bus_MS2_RailwaySignal" the CAN data from the thus be read "Mobile Station".

If more bits to be read at once, then the macro "MobaLedLib_Copy_to_InpStruct" which on page 24 described are used.

3.1.7 MobaLedLib.Get_Input (uint8_t channel)

To read individual input channels, this function can be used. This can be especially useful for testing purposes.

3.1.8 MobaLedLib.Print_Config()

This feature allows the contents of the configuration array for debugging purposes can be output via the serial interface. But for the following line must be activated in the file "Lib_Config.h":

```
#define _PRINT_DEBUG_MESSAGES must be enabled in "Lib_Config.h"
```

and the serial interface; initialize "Serial.begin (9600)". The "Lib_Config.h" file can be found under Windows in the directory "C: \ Users \ <username> \ Documents \ Arduino \ libraries \ MobaLedLib \ src".

Note: This requires a lot of memory (4258 byte FLASH, 175 bytes of RAM). So you should enable testing only the Kompilerschalter.

The function and the initialization of the serial interface is made "(setup)" in the function.

3.2 Heartbeat of the program

The class "LED_Heartbeat_C" is a small additional class with an LED for monitoring the functioning of the microcontroller and the current on the program can be used. If the LED flashes regularly, then the program is running normally.

The class can be used independently of the MobaLedLib class.

3.2.1 LED_Heartbeat_C (uint8_t Pin No.)

The class "LED_Heartbeat_C" is initialized with the following call in the main program. The parameter "Pin No." contains the number of Arduino terminal to which the light-emitting diode is connected. The digital inputs / outputs of a Arduinos be addressed with the numbers 2 through thirteenth The analog inputs 0-5 can also be used to drive the LED. They are selected on the constants A0 to A5. The analog inputs A6 and A7 of the "nano" can not be used as a base because they have no corresponding output stage. In most instances, the built-in LED Arduinos which is used by the constant "LED_BUILDIN" is passed to the class. In the examples which utilize the CAN bus that can not be, because the pin of the internal LED is also used as clock generator for the SPI bus.

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

3.2.2 Update()

The functionality of the program it is checked in that the function can blink which the heartbeat LED "(loop)" in the function of the program is called. If the LED flashes, then you know that the program calls the appropriate place regularly. To this line must be installed in the "loop()" function:

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

4 Many switch with a few pins

The library provides to the module "Keys_4017.h" an incredibly flexible way to read very many switch via a few signal lines are available.

If the "Keys_4017.h" file is integrated into the user program, an interrupt routine is activated automatically in the background which is a matrix consisting of a large number of switches consist can queries. The remarkable thing is that this very few signal lines are required. This is important because the Arduino has only a limited number of inputs and outputs. A few signal lines are also desirable if the switches are not directly near the Arduino. This saves cables and connectors.

The switches can be read independently. Any number of switches are activated simultaneously.

All switches are read within 100 milliseconds. Thus an immediate response to the change of a switch is guaranteed.

The use in the user program is very simple, because the queries performed automatically in the background.

The module can be used independently of the MobaLedLib.

4.1 configurability

Which and how many ports can be used for reading the switch be freely configured. A minimum of three processor connections to read the switches are required. but it can also be used up to ten pins. To read the switch one or more ICs of the type CD4017 (0.31 €) will be needed. The number of these devices depends on the number of read in switch and the number of signal lines.

With a CD4017 and three signal lines 10 switches can already be processed. With each additional signal line 10 other switches can be read. At 10 lines can be read in this way, 80 switch. Theoretically, more than 10 lines are possible. Then, however, resistors must be used as shown below in the circuit, otherwise the output current of the ICs can be too big bigger pull down.

The number of switches can be increased but also through the use of multiple CD4017. With two blocks and three signal lines 18 switches can be read. Three ICs used, then 26 switches can be read. At 10 ICs, it would switch 82 which can be read using only three signal lines from the Arduino.

The number of switches is calculated from:

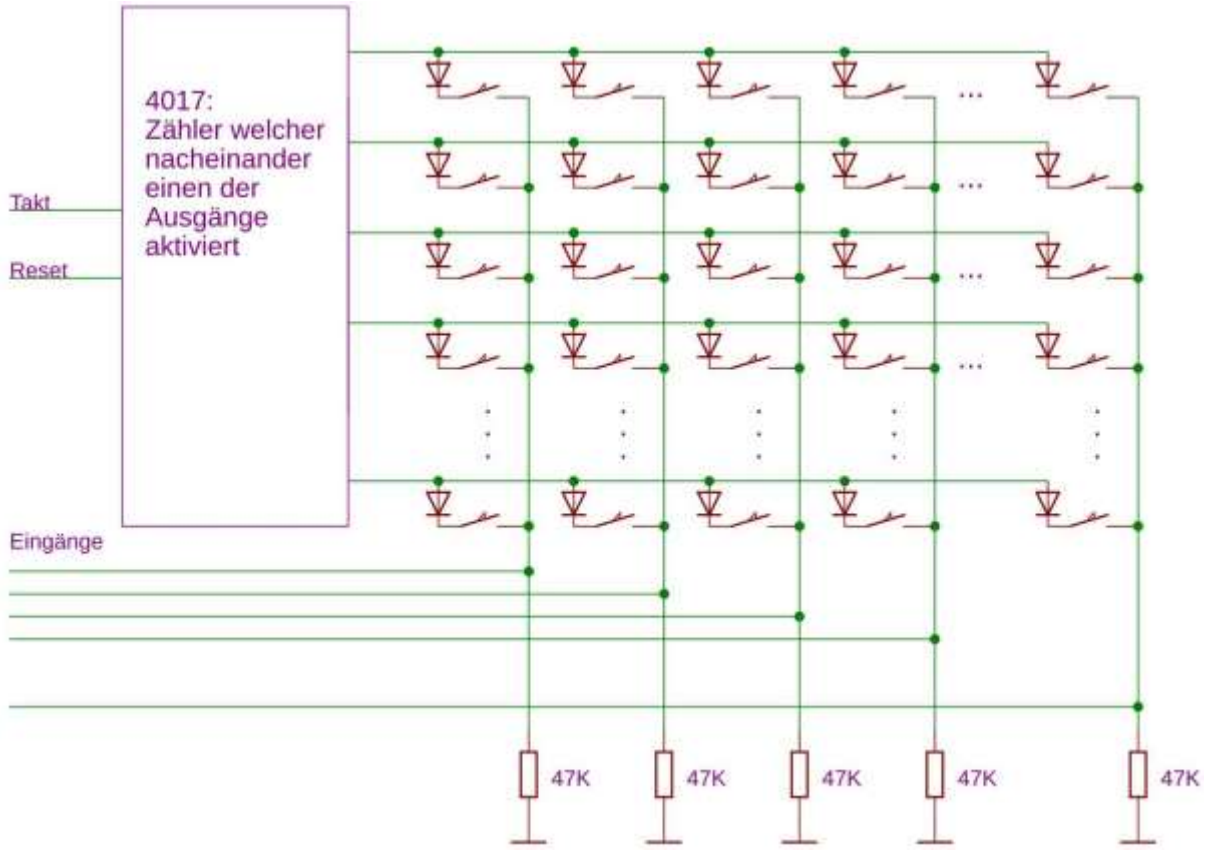
(IC Number * 8 + 2) * (signal lines - 2)

4.2 two groups of switches

The module can read two such groups of switches simultaneously. A group may be housed for example in a turnout control panel and another group can read distributed at the edge of the installation switch. The first group can consist of 80 switches which are read in over 10 signal lines. The second group may consist of several modules, each with a CD4017 made which are connected with each other via only 3 signal lines. These switches can be read in so-called "push-button actions." two of the signal lines are thereby used by both groups together so that only 11 ports of Arduinos in total are required!

4.3 principle

The IC CD4017 is a counter sequentially activated at each input pulse its outputs.



At the beginning of the uppermost output of the counter is enabled. So that the switch can be read in the top row. With each clock signal at the input of the counter the next output is activated. In the second step, the switches can be read from the second row like that. The block has ten outputs. This means that 80 switches can be read at eight input channels. prevent the diode in the circuit that the switch affect each other.

4.4 Integration into the program

To integrate the module in the user program only a few lines are required:

```
#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"
```

With the "#defines" the counter channels used and the pins of the Arduino be set. The above example defines two groups of switches.

The first group uses all ten channels of a CD4017 (CTR_CHANNELS_1 10). The constant "BUTTON_INP_LIST_1" contains a list of eight input pin numbers. Thus, there is this group of 80 switches.

The second group we set with the constant "CTR_CHANNELS_2" and "BUTTON_INP_LIST_2". Here are two counter ICs are used which are read via an input. So there are 18 switches in the group.

With the last two constants of the connection of the clock line and the reset line is specified. These signals are shared by both groups.

The module asks all switches off within 100 milliseconds. Thus an immediate response to the change of a switch is guaranteed. It writes the state of the switches in a bit array. For each group its own array exists:

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

which can be read by the user program. The integration of these arrays in the MobaLedLib class, the following lines are in the "loop()" function of the program requires:

```
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 Freely available board

In the "extras" directory of the library, the S3PO_Modul_WS2811.zip file in which the schematic and the appropriate board for import of switches via this module.

The board contains in addition to the CD4017 and a NAND gate with which the signals are passed on to the next counter WS2811 three modules with light-emitting diodes which can be controlled in the switches. Alternatively, switch with integrated RGB LEDs can be used.

The circuit can be used for distributed reading "push-button actions" and for reading many switches in a switch control board.

A detailed documentation of the circuit is refilled when required (mail MobaLedLib@gmx.de).

4.6 additional libraries

The module uses the libraries "TimerOne.h" and "DIO2.h". Both can be installed using the Arduino IDE. Do this by calling the (integrated Sketch / Library / manage library) the library administration and are "TimerOne" or "DIO" in the "Narrow your search" box. The found entry must be selected and can be installed on "Install".

4.7 Limitations

The keys are read by interrupt. For this, the timer interrupt 1 is used. Therefore, this interrupt can not be used for other tasks. By default, the timer 1 for the servo library is used. If the switches are read in this module, can not be driven simultaneously servos. but that is anyway in connection with the "FastLED" library that builds the whole project is not possible because the interrupts are updated while the LEDs need to be locked because the timing of WS281x chip is very critical.

5 CAN Message Filter

In this section, the module "Add_Message_to_Filter.h" describes ...

But enough has been written. It reads yes but no ...

If you want to read more, then encourage me with a mail: MobaLedLib@gmx.de

6 Connection concept with distribution modules

The biggest advantage of WS281x modules is the easy wiring. By using four-pole connectors which can be simply plugged into power strips, illuminating a complex system is very simple. This section is described in more detail ...

Images…

7 Details Pattern function

The Pattern feature is incredibly powerful. With her komplexesten Animations can be easily configured. This section will explain the ...

7.1 The various commands Pattern

```
PatternT1 (LED, NStru, InCh, LEDs, VAL0, Val1, Off, Fashion, T1, ...)
:
PatternT20 (LED, NStru, InCh, LEDs, VAL0, Val1, Off, fashion, T1, T2, T3, T4, T5, T6, T7, T8,
T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20, ...)

APatternT1 (LED, NStru, InCh, LEDs, VAL0, Val1, Off, Fashion, T1, ...)
:
APatternT20 (LED, NStru, InCh, LEDs, VAL0, Val1, Off, fashion, 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, Fashion, T1, ...)
:
XPatternT20 (LED, NStru, InCh, LEDs, VAL0, Val1, Off, fashion, 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, Fashion, T1, ...)
:
PatternTE20 (LED, NStru, InCh, Enable, LEDs, VAL0, Val1, Off, fashion, T1, T2, T3, T4, T5, T6,
T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20, ...)

APatternTE1 (LED, NStru, InCh, Enable, LEDs, VAL0, Val1, Off, Fashion, T1, ...)
:
APatternTE20 (LED, NStru, InCh, Enable, LEDs, VAL0, Val1, Off, fashion, 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, Fashion, T1, ...)
:
XPatternTE20 (LED, NStru, InCh, Enable, LEDs, VAL0, Val1, Off, fashion, 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

This section explains how the Excel "Pattern_Configurator.xlsm" program is used ...

8 Troubleshooting

Here I want to explain how to prevent errors in configuration, detects and corrects ...

9 Manuele tests

The library provides several tools with which you can test the Connected LEDs. This means that individual LEDs can be controlled, change the color and carry out performance measurements.

10 constants

The constant should also be described in more detail ...

10.1 Constant for the channel number Cx.

Pictured right WS2811 module for 12V is shown. Here connectors were soldered to connect individual LEDs. On the upper side of the board, an additional board is used with the positive terminal is distributed to all three connectors.

Note: For other WS2811 modules can vary the connection configuration.

Surname	description
C1 = C_RED	A first channel WS2811 module or the Red LED
C2 = C_GREEN	A second channel module or the Green LED WS2811
C3 = C_BLUE	A third channel WS2811 module or the Blue LED
C12 = C_YELLOW	First and second channel of a module WS2811
C23 = C_CYAN	Second and third channel of WS2811 module
C_ALL	All three channels of a module WS2811 (Weis)

10.2 Constants for hours ("Timeout", "Duration"):

Surname	description
min	In minutes
Sec = sec	In seconds
Ms = ms	In milliseconds

In minutes and seconds and decimals can be used: Example: 1.5 min

The times can also be Adds: Example: 1 min + 13 sec

What is important is the distance between number and unity.

10.3 Constants of the Pattern function

Surname	description
PM_NORMAL	Normal mode (as a wildcard in Excel)
PM_SEQUENZ_W_RESTART	Rising edge-triggered unique sequence. A new edge starts with state 0th
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	Do not 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 (transition) 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 and modes (for Random) and RandMux()

Surname	description
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 and modes for the Counter) Function

Surname	description
CM_NORMAL	Normal Counter mode
CF_INV_INPUT	Invert input
CF_INV_ENABLE	input Enable
CF_BINARY	Maximum 8 outputs
CF_RESET_LONG	Button long = Reset
CF_UP_DOWN	An RS flip-flop can be made with CM_UP_DOWN without CF_ROTATE
CF_ROTATE	Begins at the end all over again
CF_PINGPONG	Changes at the end of the direction
CF_SKIP0	Skips 0. The 0 comes only with a timeout or when the button is pushed long
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 maximum counts-1 (counter => 0 .. n-1))
CF_ONLY_LOCALVAR	Do not write to the LEDs with defined DestVar. The first contains the number DestVar maximum 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 Lighting types of rooms:

Surname	R	G	B	description
ROOM_DARK	50	50	50	Room with low light,
ROOM_BRIGHT	255	255	255	Bright room with lighting
ROOM_WARM_W	147	77	8th	Room with warm white light
ROOM_RED	255	0	0	Room with bright red light
ROOM_D_RED	50	0	0	Dark room with red light
ROOM_COL0				Room with open Kammin. This produces a flickering reddish light which (hopefully) is similar to a fireplace. The Kammin not always burn. From time to time (random controlled) and a normal light is on.
ROOM_COL1				With this constant, the flicker is simulated an ongoing TV. For this, the RGB LEDs are controlled by chance. If the Preiserlein times not check the Internet then burns a normal light.
ROOM_COL2				In this room, sometimes the TV is or it burns the fire and from time to time a book is read well in normal light.
ROOM_COL3				With this type of a second television program is simulated. In our model world there are only two different television programs. but these are, after all, even in color. Various programs are required, making it flicker different in adjacent windows to see. Other TV stations can be activated in the program with the compiler switch TV_CHANNELS. A downgrade to black / white TV could be added in the program if that fits better in the era of the plant.
ROOM_COL4				How ROOM_TV0_CHIMNEY only with ZDF.
ROOM_COL5				Room with user defined color 5
ROOM_COL345				Room with user defined color 3, 4 or 5 All 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

Surname	R	G	B	description
ROOM_TV1_CHIMNEYD				With TV channel 1 and fire or Light
ROOM_TV1_CHIMNEYB				With TV channel 1 and fire or Light
The program function which is controls the houses "(gas Light)" also by the macro used which is described in the next section. The following constants are intended. They simulate gas lamps which only slowly reach full brightness and occasionally flicker. These lamps can of course also be used in a house.				
GAS_LIGHT	255	255	255	Gas lantern with bulb which consumes between 20mA and 60mA at 12V. The lamp is driven at full brightness.
GAS_LIGHT1	255	-	-	Gas lantern with LED which the first channel (red) of a WS2811 chip is connected.
GAS_LIGHT2	-	255	-	Gas lantern with LED which the second channel (green) of a WS2811 chip is connected.
GAS_LIGHT3	-	-	255	Gas lantern with LED which on the third channel (blue) is connected a WS2811 chips.
GAS_LIGHTD	50	50	50	Gas lantern with bulb which consumes between 20mA and 60mA at 12V. The lamp is driven with reduced brightness.
GAS_LIGHT1D	50	-	-	Dark LED to channel 1
GAS_LIGHT2D	-	50	-	Dark LED on channel 2
GAS_LIGHT3D	-	-	50	Dark LED on channel 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 All which is not controlled with by the house() function (Useful for shops in a house Because this lights are always on at night)