Attention the changes made in the German version since January 2019 have not been updated in this document yet ;-(

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. If you find an error please contact the author at MobaLedLib@gmx.de

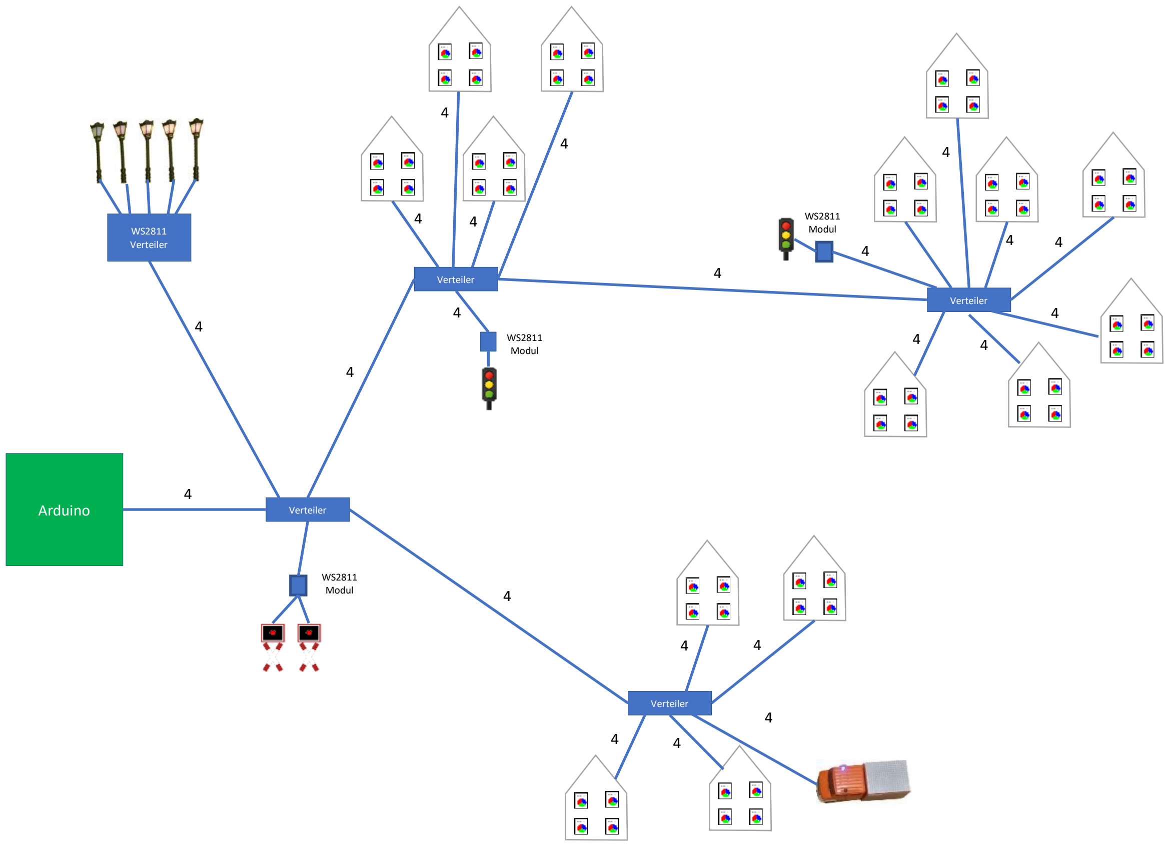
1 Introduction

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 22 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 RGB_Heartbeat2(LED, MinBrightness, MaxBrightness)

Like "RGB_Heartbeat ()" but this function has two more parameters that can be used to specify the minimum and maximum brightness.

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

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

2.3.4 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).

If the houses will have a manually operated switch, there should be a direct feedback when the switch is changed. Therefore, immediately when the input (InCh) is activated a light is turned on. On the other hand, one random light is turned off if the switch is turned 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 room:
SKIP_ROOM

Example: `House(0, SI_1, 2, 3, ROOM_DARK, ROOM_BRIGHT, ROOM_WARM_W)`

2.3.5 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 how long it takes randomly until the next change occurs in seconds. In the "House()" function these times are globally defined:

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

Both constants could be adapted in the user program by adding the lines above the "#include "MobaLedLib.h" line. In the two "House" examples this is shown.

2.3.6 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 (in German): http://www.gaswerk-augsburg.de/fernzuendung.html.
The lanterns are not turned on simultaneously caused by different times or lighting conditions. I observe this several times 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 which are controlled via the macro "(Gas Lights)". Here is also a random flickering implemented which can be caused by fluctuations in the gas pressure or by wind gusts. They are controlled by WS2811 chip. In light bulbs all three outputs are connected in parallel. With LED lamps one 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: GasLights(Gas_Lights1, 67, GAS_LIGHT1D, GAS_LIGHT2D, GAS_LIGHT3D, GAS_LIGHT)

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

The macro "Set_ColTab()" can be used to 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 "GasLights()" 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 processes are common in reality and, of course, on a model railway. One example is the traffic light. Here the corresponding traffic light phases are sequentially displayed. Several lamps must be switched coordinated for this display.

In the library, these controls are generated by the "Pattern()" function. Such sequences can be generated using the Excel program "Pattern_Configurator.xlsm". In chapter 5 on page 43 that’s described in detail.

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

This macro stores a button state for a certain time. This is used in our railway to enable the smoke generator in the "Burning" house. The output can be deactivated before the expiration of the time when the button is hold a for one second.

The duration determines how long output is activated when the button is pressed. The time is specified in milliseconds and may be between 16 ms and 17 minutes (1048560 ms). "Sec" or "Min" can be added to the value for longer times. 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 Blinker(LED, Cx, InCh, Period)

Implements a turn signal (Flasher) at a predetermined period. The period is specified in milliseconds. Here, "Sec" or "Min" could be added also as described in the "Button()" function. The maximum period is two minutes (131070 ms).

2.4.3 BlinkerInvInp(LED, Cx, InCh, Period)

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

2.4.4 BlinkerHD(LED, Cx, InCh, Period)

Another variation of the turn indicator. Here the brightness of the LED changes between "bright" and "dark". The function is used in the example "Macro_Fire_truck".

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

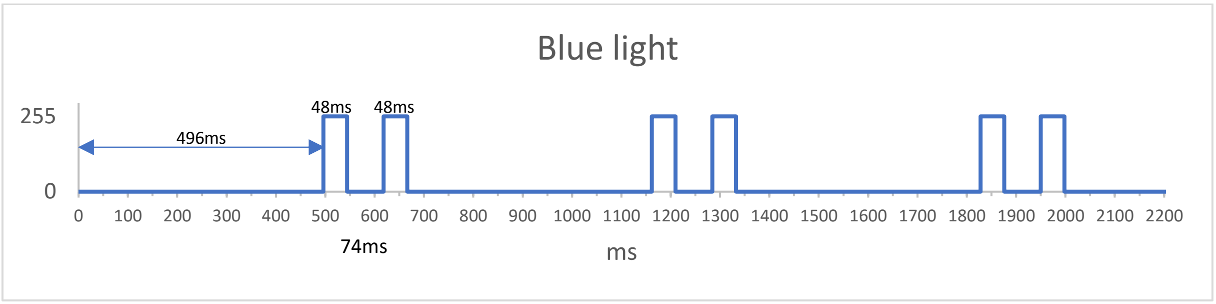
In this variant, the duration of the two phases and the brightness in the phases can be specified. "Pause" defines the time while "VAL0" is used and "Act" is the time while "Val1" is send to the output.

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

A third variant of the "Blinker()" function. Here, the brightness value (Off) can also be specified when InCh is disabled.

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)

Second Blue lights with a slightly different period. By the use of two blue lights with slightly different period, a more realistic effect is generated.

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 refer https://www.windparkwaldhausen.de/contentbeitrag-170-84-kennzeichnung_befeuerung_von_windkraftanlagen_.html)

2.4.10 LeuchtfeuerALL(LED, InCh)

This beacon uses all three channels of a WS2811 chip. This corresponds to the "Leuchtfeuer()" command with the parameter Cx = "C_ALL".

2.4.11 Andreaskreuz(LED, Cx, InCh) = "Andrew's Cross"

This function can be used for controlling the alternating flashing lights in St. Andrews crosses. "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)

This generates the pattern of two traffic lights with 3 RGB LEDs each for one 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 sheets 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 faded which generates a more realistic impression. The "RGB_AmpelX_Fade" page in "Pattern_Configurator.xlsm" shows how the fade in works.

2.4.15 AmpelX(LED, InCh)

This traffic light is designed for use on the model railway. It controls individual LEDs over two WS2811 modules. In the example "09.TrafficLight_Pattern_Func." one finds a "circuit diagram" which shows the wiring. To secure a crossing 4 traffic lights are needed. The opposing light signals always shows the same pattern. For driving the opposite lying traffic light another WS2811 chip which receives the same input signal as its counterpart could be used. In the example this is shown schematically.

With the program "Pattern_Configurator.xlsm" arbitrarily complicated traffic light systems with several lanes and pedestrian traffic lights and can be configured.

2.4.16 AmpelXFade(LED, InCh)

This is the same function as above except that here the lamps are slowly fading in and out.

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 flash of a photographer. The parameters "MinTime" and "MaxTime" determine how often the flash fires. The first parameter determines how long to wait at least until the next flash. Accordingly, "MaxTime" describes the maximum time. Between these two times, the library determines a random time.

The macro consists of two different macros. The "Const()" and the "Random()" macro (See "2.7.24 **Random**(DstVar, Inch, fashion, MinTime, MaxTime, Minon, Maxon)" on page26) . This requires an intermediate variable whose number is entered as parameter "Var". This is one of the 256 input variables which are described in chapter "2.1.3 InCh" on page 10. Attention this intermediate variable must not be used anywhere else.

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

2.5.2 Fire(LED, InCh, LedCnt, brightnes)

With the "Fire()" function, larger fires can be simulated. For this purpose, several RGB LEDs are used which light up in different places of the "fire". On our system, the function is used to simulate a "burning" house.

2.5.3 Welding(LED, InCh)

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

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

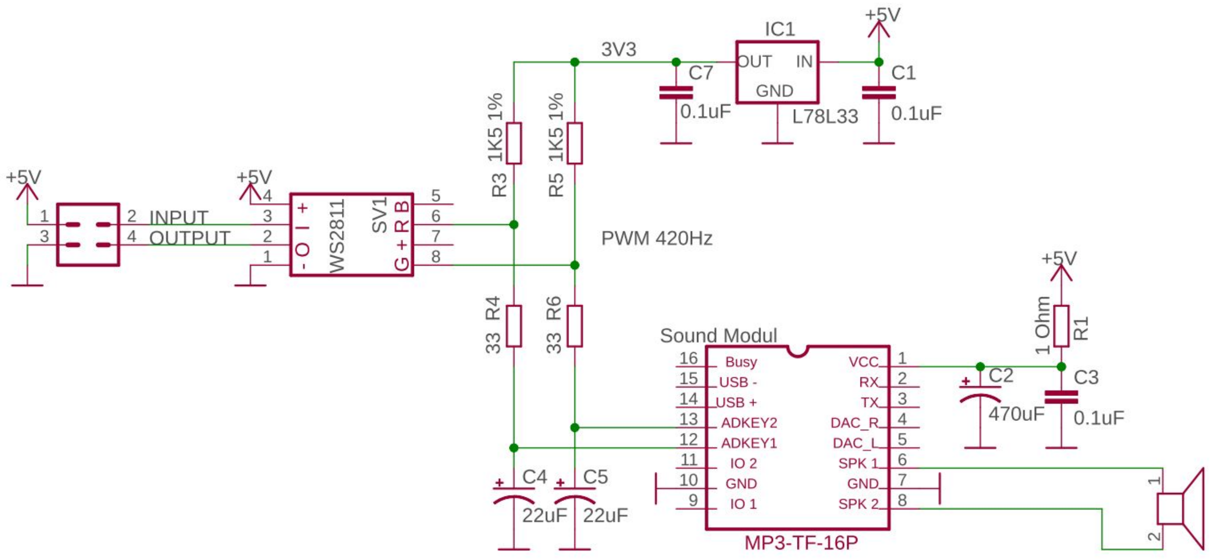
With this function, the welding light is controlled randomly. The times "MinTime" and "MaxTime" determine the random start time. "MinOn" and "MaxOn" indicate how long it takes to work on a workpiece. The parameter "Var" contains the number of an intermediate variable which is used to control the "Welding()" function. Attention this intermediate variable must not be used anywhere else.

Example: **RandWelding**(12, SI_1, 201, 1 Min, 3 Min, 50 Sec, 2 Min)

2.6 Sound

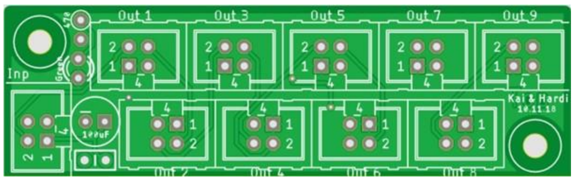
The MobaLedLib library can be used to reproduce sounds appropriate to the lighting effects. For example, the bell jar can be played with the flashing of the St. Andrew's cross. This requires an additional sound module of the type MP3-TF-16P:

This module is available in China for one euro. In addition, 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 played over 20 buttons which are connected to the module via different resistors. The sound module can be controlled via the same signal line as the light emitting diodes by simulating the keystrokes. For this a WS2811 chip is used. This must be connected with a few resistors and capacitors for filtering the signals. The following diagram shows the structure:



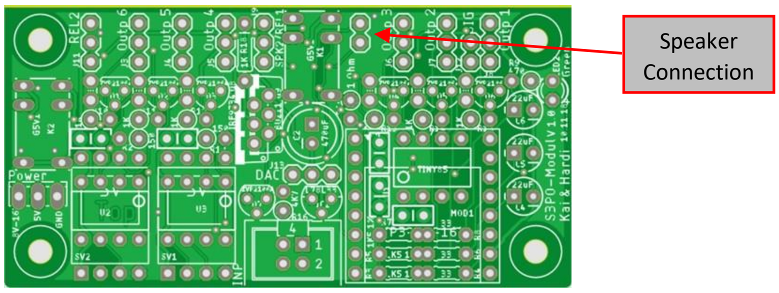
The zip file S3PO_Modul_WS2811.zip in the "extras" includes a schematic and a board with this construction. On the circuit some more, components are provided with which higher power can be switched via a WS2811 module. In addition, you can use the circuit to control servos or stepper motors. These are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

The board consists of three parts which can be separated by kinking. The two upper parts (only one is shown in the picture below) contain a 9-way distributor for connecting the LEDs and other components based on the WS281x:



These distributors can be placed anywhere in central positions under the system. The plugs marked "Out ..." plug the houses, traffic lights, sound modules and other consumers. Here also further distribution boards can be connected. The connector labeled "Inp" connects to the Arduino or a previous distribution board via a ribbon cable. Attention: In case of unused slots, pins 2 and 4 have to be jumpered or the corresponding solder jumpers on the back have to be connected.

The next picture shows the board for the sound module. Only the components have to be equipped which are shown in the circuit diagram above.



Attention: On the back there are some solder joints. For the function of the sound module without the other components, SJ1 and SJ21 must be connected.

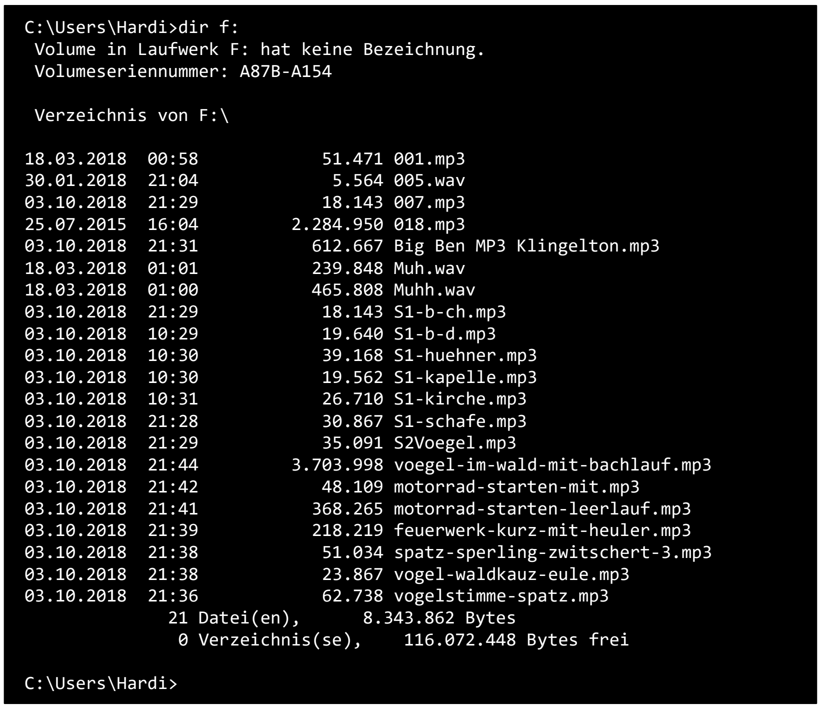
In the following, the commands for sound output are briefly introduced.

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

There are 14 commands in the library to play specific sounds. The command "Sound_Seq1()" plays the first MP3 or WAV file on the SD card. Accordingly, the second file is played with "Sound_Seq2()" ...

According to the documentation of the sound module, the MP3 or WAV files must have special names and be stored in certain subdirectories. This seems to apply only to the reproduction via serial interface. If the files are played via buttons, then this is not necessary. This also applies to the circuit presented here which simulates the keystrokes via a WS2811 module. The file name does not matter. The "Sound_Seq1()" command always plays the file which was first copied to the SD card. Hence the name "..Seq ...". If a file is deleted and a new file is later copied to the card, then the new file in the table of contents is entered on the card in the place of the deleted file. That can be very confusing. Unfortunately, Windows Explorer always displays the files sorted on an SD card. There is no option to completely disable sorting. To check the order, you have to open a command line window (cmd.exe) and type in "dir f:" (The drive letter "f:" may need to be adapted to the actual recognized SD card reader):

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



```
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>
```

The files are played over simulated "key presses" which are encoded by different resistors. Unfortunately, the upper resistance distances between the resistors are relatively small, which can lead to overlapping in the sound commands "Sound_Seq13()" and "Sound_Seq14()" due to component tolerances or temperature variations.

The output of sounds can be globally turned on and off via the variable "SI_Enable_Sound". In this way you can disable all sounds with a switch.

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)

This command plays a random file between 1 and "MaxSoundNr" when the input of the command is activated. This can e.g. be used for the announcement of station announcements.

2.6.3 Sound_Next_of_N(LED, InCh, MaxSoundNr)

With this macro, the next file between 1 and "MaxSoundNr" is displayed when the input of the command is activated. This can e.g. used for playing the hourly ringing of church bells.

2.6.4 Sound_Next_of_N_Reset(LED, InCh, InReset, MaxSoundNr)

This command is the same as the previous one. Here it is additionally possible to reset the counter via a further input "InReset".

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 "key" selectable files like the previous commands. In this way all files can be played one after the other. A targeted restriction of the area is not possible. This can be done in advance by selecting the 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)

This allows the rendering to be paused and resumed.

2.6.8 Sound_Loop(LED, InCh)

This command testifies to the origin as a music player. This allows all songs to be played on an SD card one after the other. For the railway you could possibly use the function with special sound files with long breaks.

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, then you can use this "button" to switch the "Play Mode". I did not quite understand that either. There seem to be the following modes:

"Sequence", "Sequence", "Repeat same", "Random", "Loop off"

How and if the two "Sequence" Modes are different at 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 being switched on. This is relatively loud thanks to the built-in 3-Watt amplifier. The volume can be reduced with the "Sound_DecVol()" macro. The parameter "Steps" indicates the number of steps to reduce the volume. The number can range from 1 to 30. To change the volume, the corresponding "button" must be held for longer than one second. Then the volume is reduced by one step every 150 ms. The corresponding timing is taken over by the macro. But you have to keep in mind, no further command may be sent to the sound module while the volume is changed. An automatic locking was omitted for space reasons.

Unfortunately, the module does not remember the last setting. This means that the volume must be reset each time you turn it on. Possibly. It is better if you expect the sound files with a suitable program "quieter".

2.6.12 Sound_IncVol(LED, InCh, Steps)

This can be used to increase the volume again.

2.7 Commands

The commands described in this section set one or more variables. These variables can then be read in and evaluated by other macros. As described in section "2.2 Variables / Input Channels" on page 11, there are 256 input variables. From the point of view of the functions presented here, these are output variables. They are called "DestVar" in the parameter list. At this point in the macro comes the number of the variable. It is recommended that a symbolic name be 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 staircase light switch. The output variable "DstVar" becomes 1 if the input "InCh" is active. The output remains active after the input has been disabled for "Duration" milliseconds. The macro corresponds to a static, retriggerable mono-flop. The time can also be specified by appending "Sec" or "Min". The maximum time is 17 minutes.

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

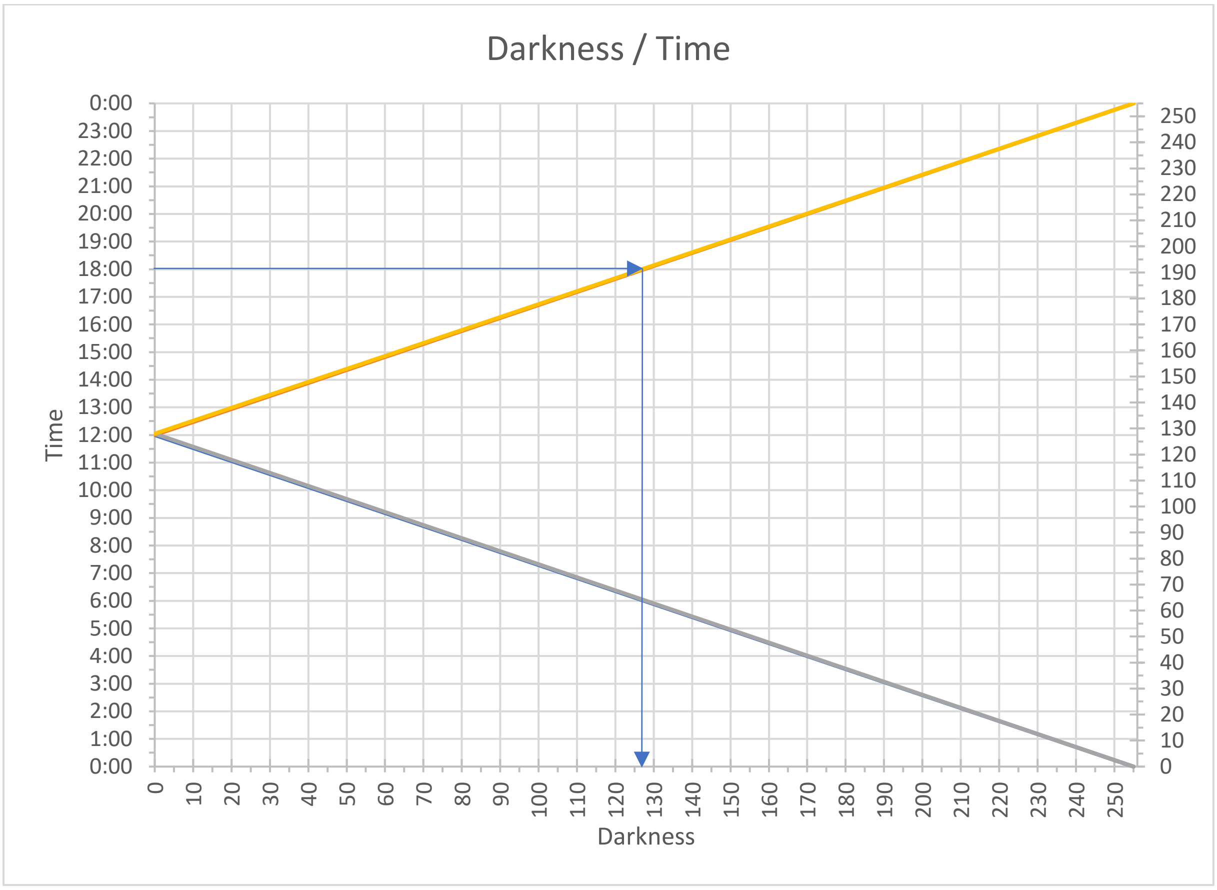
The "Schedule" macro can be used to create a schedule for switching several lights on and off. However, this plan only specifies the rough framework conditions. When the outputs are actually switched, the program randomly determines it creates a real impression.

The output variables "DstVar1" to "DstVarN" are switched. They are switched on randomly between the time "Start" and "End", if it is "evening" and just as coincidentally switched off again on the "morning". Whether it is "evening" or "morning" is determined by the global variable "DayState". The variable is set to "SunSet" "in the evening" and "SunRise" in the morning. The second variable "Darkness" determines how a number between 0 and 255 is "dark". That represents time.

Time can be generated in different ways. The simplest timer is the brightness. This allows the lights to turn on when it gets darker. 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 suit the lighting in the room. The example "Darkness_Detection" uses this method. The example also shows how to use a "Day" and "Night" switch.

But it is also possible to receive the time from an external model train timer. It can e.g. read in via the CAN bus. This makes sense if the room lighting is controlled at the same time.

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



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

2.7.3 Logic(DstVar, ...)

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

(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 higher-level control input with 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, DISABLE 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" parameter. The following flags are defined. Several flags can with the or operator, | ' be combined.

CM_NORMAL	normal counter
CF_INV_INPUT	InCh is inverted
CF_INV_ENABLE	Enable invert input
CF_BINARY	Binary counter, otherwise the outputs are enabled one by one
CF_RESET_LONG	reset active when input longer than 1.5 seconds active
CF_UP_DOWN	Up / down counter ("Enable" => reverse)
CF_ROTATE	Counter starts again at the beginning when the end is reached
CF_PINGPONG	Counter which switches the direction at the ends
CF_SKIP0	Skips the 0
CF_RANDOM	Generate a random count with each pulse at InCh
CF_LOCAL_VAR	Count is written to the predefined local variable (please refer "2.7.25 New_Local_Var()" on page 26 and "2.7.26 Use_GlobalVar(GlobVarNr)" on page 27)
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 function description represent a variable number of output channels. The numbers of the variables used are entered here. These variables serve other macros as input. In normal mode, the outputs are activated one after the other. If the flag CF_BINARY is specified, then the outputs are controlled like the individual bits of a binary number.

The following are some macros based on the "Counter ()" macro based.

2.7.5 Monoflop(DstVar, InCh, Duration)

A mono flop 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 MonoFlopLongReset(DstVar, InCh, Duration)

Is a mono flop that can be reset when the input is active for more than 1.5 seconds. The duration can be extended as with the previous function.

2.7.7 RS_FlipFlop(DstVar, S_InCh, R_InCh)

A flip-flop can accept two states (0 or 1). With an RS flip-flop, the states are determined via two inputs. 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 corresponds to the previous one. Here there is an additional parameter "Timeout" which determines when the flip-flop is automatically deleted.

2.7.9 T_FlipFlopReset(DstVar, T_InCh, R_InCh)
The output of a "toggle flip-flop" is switched to input on every positive edge. This feature also has a "reset" input that allows the flip-flop to be set to zero. If this input is not needed, it can be assigned "SI_0".

2.7.10 T_FlipFlopResetTimeout(DstVar, T_InCh, R_InCh, timeout)
Also has a parameter "Timeout".

2.7.11 MonoFlopInv(DstVar, InCh, Duration)
This mono flop has an inverse output. This means that the output is active at the beginning (1) and is deactivated with a positive edge at "InCh". The time can be extended as with normal MF with a rising edge.

2.7.12 MonoFlopInvLongReset(DstVar, InCh, Duration)
Here the properties "Inverse" and "Reset" when the input is active for more than 1.5 seconds are 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 comes again the "Timeout" parameter.

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 one output.

2.7.17 MonoFlop2(DstVar0, DstVar1, InCh, Duration)
The following macros have two outputs which are inversely connected to each other. The functions are the same as in the previous ones, so here is a detailed description omitted.

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 randomly activates one of the outputs defined by the numbers "DstVar1" to "DstVarN". "MinTime" determines how long an output is minimally active. "MaxTime" analogously describes the maximum time that the output should remain active. The program determines between these two vertices a random time at which a random other channel is activated. For example, this function can be used to switch between different lighting effects for a disco.

With the flag "RF_SEQ" as parameter "Mode", the next output is not chosen randomly, but the outputs are activated one after the other.

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

The function "Random ()" activates an output after a random time. The duty cycle can also be random. This can be used to randomize an effect. One application for this is the flash of a photographer who is supposed to flash from time to time.

With the parameters "MinTime" and "MaxTime" it is determined in which time range the function should become active. The duration is specified via "MinOn" and "MaxOn". For the flash, "MinOn" = "MaxOn" = 30 ms is used (See "2.5.1 Flash(LED, Cx, InCh, Var, MinTime, MaxTime) on page 17).

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 "Random()" function is the "RandWelding()" function on page 17.

The "Random()" function currently knows a special mode: RF_STAY_ON. With this switch, the output stays on until the time specified by "MinOn" and "MaxOn" has elapsed, when the input is no longer active. This is used in the "Button ()" macro which, like "Flash ()" and "RandWelding ()", uses the "Random ()" function:

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

2.7.25 New_Local_Var()

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

But there are also cases in which more than two states (on / off) are needed. For this purpose no fixed number of variables are provided in the library because the RAM of an Arduino is very limited.

If necessary, a variable of type "ControlVar_t" is created with the macro "New_Local_Var ()". It can accept values between 0 and 255 and has additional flags to detect changes. This variable can then be set by a function and evaluated by one or more other functions. This approach only uses only then more valuable RAM when it is actually needed. In addition, the user does not have to worry about an intermediate variable as with the "Flash()" or "RandWelding()" function.

To set the local variable, the "Counter ()" command could be used (page 20).

The variable is evaluated with the pattern functions (from page 31).

The example "RailwaySignal_Pattern_Func" shows how this is done.

The memory for the variable is created automatically. For this purpose, the "new" command is not used, as is usual with C ++, but when compiled, the array "Config_RAM []" is created statically in the required size. This has the advantage that the RAM requirement is already known when compiling and the compiler can generate warnings if the memory becomes scarce. In this case, the following message is shown in the Arduino IDE:

```
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 there is little RAM left for the program. This memory is needed in the C++ program for dynamically created variables and for the stack. From the perspective of the compiler, it is not possible to determine how much memory is needed exactly. Therefore, the memory in the library is statically reserved.

2.7.26 Use_GlobalVar(GlobVarNr)

The library can be extended with own functions in the C ++ program. With the function "Use_GlobalVar()" the own parts of the program can exchange data with the library-internal functions. The global variables can be used in the same way as the local variables which are created with the function "New_Local_Var()". However, the global variables are stored in their own array. This array must be defined in the user's sketch:

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

and announced to the library in the "setup ()" function:

```
MobaLedLib_Assigne_GlobalVar(GlobalVar);
```

With this the command "Use_GlobalVar ()" can be used. To set the variable, e.g. this function could be inserted into the user's sketch:

```
//-----------------------------------------
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 fills a temporary 8-bit variable with the values from several logical variables. Unlike the "New_Local_Var()" function and the "Use_GlobalVar()" function, no additional memory is needed here. This is possible because the same memory can be used multiple times. In the other two cases, it must be saved whether the input changes because only then should an action be triggered. This function uses the change of the input variables.

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

2.7.28 Bin_InCh_to_TmpVar(FirstInCh, InCh_Cnt)

This command is similar to the previous one. However, the inputs here are added in binary form:

Erg = InCh0 + InCh1 * 2 + InCh2 * 4 + InCh3 * 8 ...

This uses fewer input channels than the InCh_to_TmpVar() function. The function is used e.g. to interpret numbers received over the CAN bus.

2.8 Other commands

2.8.1 CopyLED(LED, InCh, SrcLED)

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

If two RGB LEDs are to show the same thing, then you can do that with the electrical wiring. For this purpose, the "DIN" lines of both LEDs are connected in parallel. Normally, all LEDs are strung together in a chain so that each LED can be individually addressed. If two LEDs are to show exactly the same, then the parallel switching is an alternative. This is outlined in the example "TrafficLight_Pattern_Func".

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

This function is used to read in a button with which could be used for the "push-button actions". The function counts the keystrokes and activates one of 3 temporary variables. In addition, the LED is activated in the switch. It flashes according to the number of keystrokes. At the first key press once, after the second key press twice and three times at the third press on the key. With the 4th key press the counter starts again with 1.

If the button is held for a longer time (1.5 seconds), then the "push button action" is ended.

Parameter:

B_LED:	Number of the RGB LED in the button.
B_LED_Cx:	Channel of the LED (C1 ... C3)
InNr:	Number of the input variable which is controlled by the button
TmpNr:	Number of the first temporary variable. The macro uses 3 variables
Rotate:	If this parameter is set to 1, the counter will restart at the end
Timeout:	Time after which the function is deactivated.

2.8.3 PushButton_w_LED_0_3(B_LED, B_LED_Cx, InNr, TmpNr, Rotate, Timeout)

Same as PushButton_w_LED_0_2 () only here are 4 temporary variables available.

2.8.4 PushButton_w_LED_0_4(B_LED, B_LED_Cx, InNr, TmpNr, Rotate, Timeout)

Same as PushButton_w_LED_0_2 () only here are 5 temporary variables available.

2.8.5 PushButton_w_LED_0_5(B_LED, B_LED_Cx, InNr, TmpNr, Rotate, Timeout)

Same as PushButton_w_LED_0_2 () only here are 6 temporary variables available.

3 Macros and functions of the main program

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

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

The macros and functions must be located 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 is used to create the MobaLedLib class. This will allocate and initialize the memory. The parameter "leds" tells the class where the brightness values of the LEDs are stored. This is an array of type "CRGB" which is defined in the "FastLEDs" library.

The macro is in the main program according to the definition of the "leds" array:

```
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 requires memory for each configuration line. For this a special method was used. The memory is reserved by each entry in the configuration during compilation. This is usually done at the runtime of the program with the command "new". However, the "usual" approach has the disadvantage that the compiler does not know how much RAM is needed during operation. With the scarce resource RAM, this can quickly lead to problems with a microcontroller. That's why another way was used here. This will be explained using the example of the "House ()" function as an example:

```
#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__,
```

Crucial here is the expression "RAMH" which describes the memory requirements of a house. It will be replaced by another macro by RAM2. This macro has the following structure:

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

After that, "RAM2" is composed of two times "RAM1". The latter macro is the crucial one. It contains the C ++ preprocessor macro "__COUNTER__" which represents a counter incremented by 1 each time it is used.

The first time you use the "RAM1" macro, the 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 0 Thus, "RAM2" and accordingly "RAMH" is also 0. But the key is that "__COUNTER__" has been changed. Each time you use "RAM1", the counter increments by two. And that is exactly 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 just as big that it can provide the needed memory for all configuration lines. Since the preprocessor macros are evaluated during compilation, the compiler knows exactly how much memory is occupied and can check if the RAM of the processor is sufficient for it. If a critical value is exceeded, this warning is displayed:

```
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. In addition, the warning appears on the screen while the program is being generated. An error message generated by the Arduino at runtime cannot be reliably displayed due to the lack of a standardized output device.

The macro "MobaLedLib_Create()" contains the line for creating 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 the macro "Use_GlobalVar()" is used in the configuration, then the library needs to know where the global variables are and how many of them are available. This will be assigned to the library with this command.

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

```
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)
To control the LEDs, MobaLedLib uses an array of logical values which also stores the previous value. From this, the library can recognize whether an input has changed and only then trigger the corresponding action.

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

As input, the function expects an array of individual bits representing the individual input channels. The parameter "Src" contains this array. The "ByteCnt" parameter specifies how many bytes should be copied. "ChannelNr" indicates the target position in the input structure "InpStrucktArray". This corresponds to the "InCh" in the configuration macros.

Attention the "ChannelNr" must be divisible by 4.

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

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

3.1.5 MobaLedLib.Update()
This is the crucial function of MobaLedLib. It recalculates the states of the LEDs in each main loop pass. It sequentially processes all entries in the configuration array and thus determines the colors and brightness's of the individual LEDs. During the development of the library, great emphasis was placed on the fact that the function is processed very quickly even in 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)
With this function an input variable of the MobaLedLib can be set. This can be used to supply switch positions or other input values to the library. The parameter "channel" describes the number of the input variable which is always used in the configuration macros "InCh".

The function is used in the "loop ()" function and possibly in the "setup ()" function of the program.

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

If more bits to be read at once, then the macro "MobaLedLib_Copy_to_InpStruct" which is described on page 31 described should be 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. For this, however, the following line in the file "Lib_Config.h" must be activated:

```
#define _PRINT_DEBUG_MESSAGES
```
and the serial interface must be initialized with "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). You should activate the compiler switch only for test purposes.

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

3.2 Heartbeat of the program

The class "LED_Heartbeat_C" is a small additional class with which an LED can be used to monitor the functioning of the microcontroller and the program running on it. 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 PinNr)

The class "LED_Heartbeat_C" is initialized with the following call in the main program. The parameter "PinNr" contains the number of the Arduino connector to which the LED is connected. The digital inputs / outputs of an Arduino are addressed with the numbers 2 to 13. Analog inputs 0-5 can also be used to drive the LED. They are selected via the constants A0 to A5. The analog inputs A6 and A7 of the "Nano" cannot be used as output because they do not have a corresponding output stage. Most of the examples use the built-in LED of the Arduino, which is passed to the class via the constant "LED_BUILDIN". This is not possible with the examples which use the CAN bus, because the pin of the internal LED is also used as a clock generator for the SPI bus. Here an external LED must be used.

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

3.2.2 Update()

The functionality of the program is checked by calling the function which causes the Heartbeat LED to flash in the "loop()" function of the program. If the LED flashes, then you know that the program regularly calls the appropriate place. For this purpose, this line must be built into the "loop()" function:

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

4 Many switches with a few pins

The library provides with the module "Keys_4017.h" an incredibly flexible method for reading in a **large number of switches** over a **few signal lines**.

If the file "Keys_4017.h" is integrated in the user program, then an interrupt routine is activated which in the background automatically queries a matrix which can consist of very many switches. The special thing about it is that it only requires very few signal lines. This is important because the Arduino has only a limited number of inputs and outputs. Few signal lines are desirable even when the switches are not directly near the Arduino. This saves cables and connectors.

The switches can be read in independently of each other. Any number of switches can be activated at the same time.

All switches are read in within 100 milliseconds. This ensures an immediate response to the change of a switch.

Use in the user program is very simple because the queries are automatically done in the background.

The module can be used independently of the MobaLedLib.

4.1 Configurability

Which and how many connections are used to read in the switches can be freely configured. At least three processor connections are required to read in the switches. But you can also use up to ten pins. To read in the switches one or more ICs of type CD4017 (0.31 €) are required. The number of these ICs depends on the number of switches to be read and the number of signal lines.

With a CD4017 and three signal lines already 10 switches can be processed. With each additional signal line, 10 more switches can be read. With 10 lines you can already read 80 switches in this way. Theoretically, more than 10 lines are possible. Then, however, larger pull-down resistors must be used than in the circuit shown below, because otherwise the output current of the ICs may become too large.

The number of switches can also be increased by using several CD4017s. With two ICs and three signal lines, 18 switches can be read. If three ICs are used, then 26 switches can be read. With 10 ICs, there would be 82 switches which can be read out of the Arduino via only three signal lines.

The number of switches is calculated from:

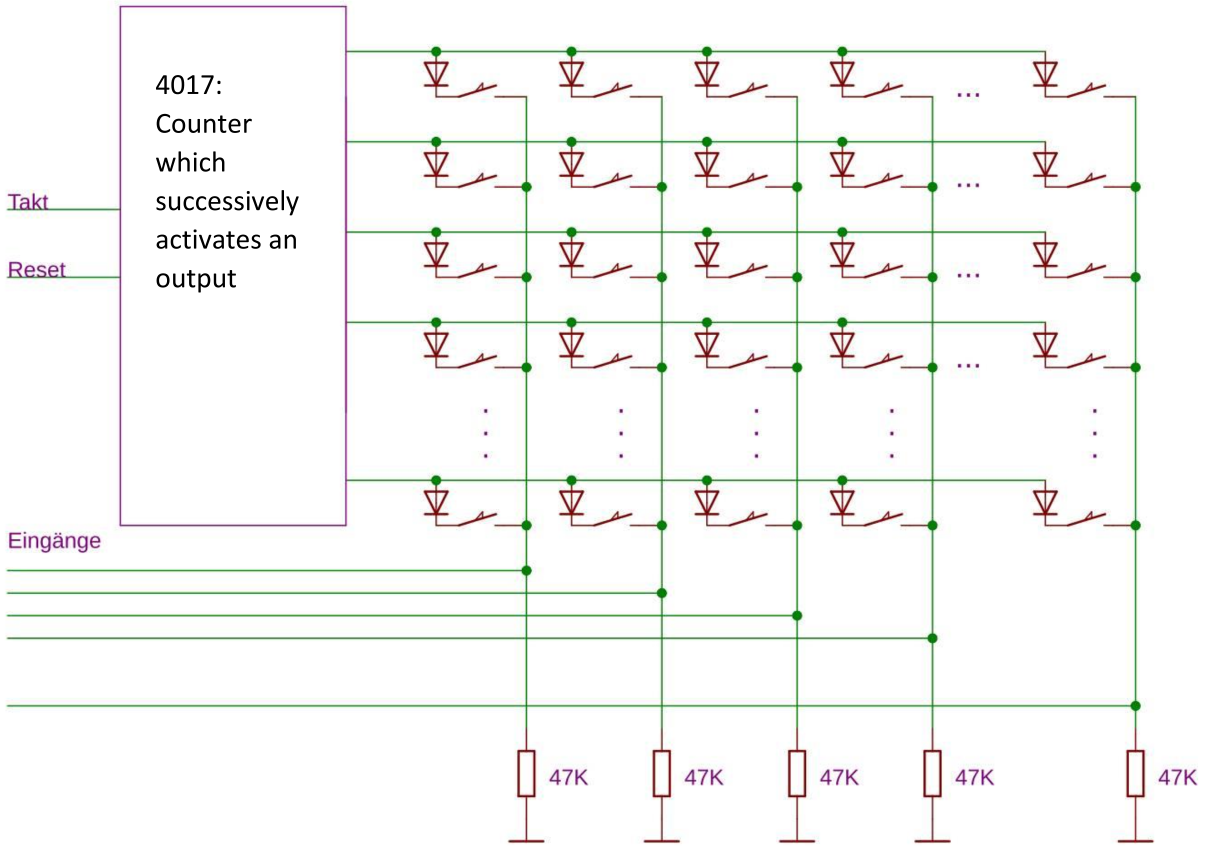
(IC Number * 8 + 2) * (signal lines - 2)

4.2 Two groups of switches

The module can read two such switch groups simultaneously. For example, one group may be located in a railway switch console, and another group may include switches distributed at the edge of the system. The first group may e.g. consist of 80 switches which are read in via 10 signal lines. The second group can consist of several modules each with a CD4017 which are connected to each other via only 3 signal lines. These switches can then read in so-called "push button actions". Two of the signal lines are shared by both groups so that in total only 11 ports of the Arduino are needed!

4.3 Principle

The IC CD4017 is a counter that activates its outputs one after the other with each input pulse.



At the beginning the uppermost output of the counter is activated. This allows the switches in the top row to be read. With each clock signal at the input of the counter, the next output is activated. In the second step, the switches from the second row can be read in this way. The block has ten outputs. Thus 80 switches can be read with eight input channels. The diodes in the circuit prevent the switches from interfering with 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 with eight input pin numbers. This group consists of 80 switches.

The second group is parameterized with the constants "CTR_CHANNELS_2" and "BUTTON_INP_LIST_2". Here are two counter ICs used which are read in via an input. So, there are 18 switches in the group.

The last two constants specify the connection of the clock line and the reset line. These signals are shared by both groups.

The module polls all switches within 100 milliseconds. This ensures an immediate response to the change of a switch. It writes the state of the switches into a bit array. Each group has its own array:

```
  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. To integrate these arrays into the MobaLedLib class, the following lines are needed in the program's "loop()" function:

```
  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 is the file S3PO_Modul_WS2811.zip in which the circuit diagram and the corresponding board for reading in switches via this module.

In addition to the CD4017 and a NAND gate with which the signals are forwarded to the next counter, the board also contains three WS2811 modules with which LEDs in the switches can be controlled. Alternatively, switches with integrated RGB LEDs can be used.

The circuit can be used for the distributed reading of "push button actions" and for reading in many switches in a points console.

Detailed documentation of the circuit will be provided later if required
(mail to 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 library administration (Sketch / Include library / Manage libraries) and enter "TimerOne" or "DIO" in the " Filter your search... " box. The found entry must be selected and can be installed with the "Install" button.

4.7 Limitations

The keys are read in by interrupt. The timer Interrupt 1 is used for this purpose. Therefore, this interrupt can no longer be used for other tasks. By default, timer 1 is used for the servo library. If the switches are read in via this module, then servos cannot be controlled at the same time. However, this is not possible in any case in connection with the "FastLED" library on which the whole project builds, because the interrupts have to be disabled while the LEDs are being updated because the timing of the WS281x chips is very critical.

5 CAN Message Filter

This section describes the module "Add_Message_to_Filter.h" ...

But now enough is written. Nobody reads ...

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

6 Connection concept with distribution modules

The biggest advantage of the WS281x modules is the simple wiring. By using four-pin plugs that can be easily plugged into distribution strips, the lighting of a complex system is very simple. This section describes this in more detail ...

Images…

7 Details Pattern function

The pattern function is incredibly powerful. With it, the most complex animations can be configured very easily. This section will explain that ...

7.1 The various Pattern commands

```
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 Manual 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.

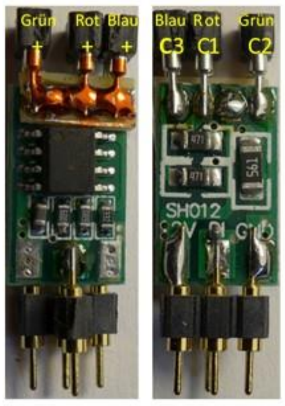
10 Constants

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.



Name	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 times ("Timeout", "Duration"):

Name	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 for the Pattern function

Name	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 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()

Name	**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

Name	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	Ping Pong and skip 0

10.6 Lighting types of rooms:

Name	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 an open fireplace. This produces a flickering reddish light which (hopefully) is similar to a fireplace. The fireplace not always burning. From time to time (random controlled) a normal light is on.
ROOM_COL1				This constant simulates the flickering of a running TV. For this, the RGB LEDs are randomly controlled. If the Preiserlein not look in the tube 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
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.

Name	R	G	B	Description
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 these lights are always on at night)