Lightstribe WS2811/WS2812

Generated by Doxygen 1.8.10

Sun May 22 2016 15:43:31

Contents

1 Use WS2811/WS2812 LEDs with an AVR

1.1 Introduction

This project is about using an WS2811 or WS2812 lightstribe with an AVR controller. It is possible to handle up to 250 LEDs at the same time, so I chose an Atmega328p with enough RAM amount. If you want to handle less LEDs you can use most parts of this project with every AVR. The AVR is programmed to receive the light data over UART so you can control the LEDs by using a serial interface. The interface uses a specified simple protocol which is described in Protocol overview section. Everything has been developed in a university course to control the lights of a Christmas tree. In the original implementation there were some further components included. This is a simplified version of the implementation so that everyone can use it. As an example for controlling the LEDs using a smart phone the Example usage with an ESP8266 section shows how this could be done by using a webserver on the E↩ SP8266. You can use everything else that provide a serial interface (maybe connect with a bluetooth serial module). The structure of this documentation is split in a hardware part for the AVR that describes the basic hardware that should be used. The next part is about how the software is working on the AVR that handles the LEDs and different effects. You may include some more stuff in your own. After that you can see a small protocol overview, where you find which command can be sent to the AVR to control the LEDs. Be aware that at the initialization state all LEDs are off. At the last point you can find an example how to use the implementation with an ESP8266 with a webserver. You will find the source code for the ESP8266 and the basic hardware setup.

1.2 Basic usage

For using this implementation follow this steps:

- set up the hardware as descriped in section Hardware
- set the F_CPU clock to the value for your hardware
- set the BAUD to the value you like, 76800 or 38400 are suggested
- compile your implementation (only O1 optimization is supported)
- program your AVR with your binaries
- set the clock divider fuse and the clock source fuse referring to your implementation
- send protocol data (see section Protocol overview) to the RX pin of the AVR over a serial device, e.g. an FTDI, ESP8266 or Arduino (UART is 8N1 on your chosen BAUD)(example data 254 6 0 1 20 22 = 0xFE 0x06 0x00 0x01 0x14 0x16)

1.3 Hardware

The basic hardware you need is a AVR controller an some WS2811 or WS2812 LEDs you want to control. The AVR controller should have an hardware UART, otherwise you need to write some code for a software serial. In the project we chose an Atmega328p that has enough RAM to control 250 LEDs. The internal software structure

buffers the color data for the LEDs to achieve an accurate timing, see section Software implementation. The AVR can be used with the internal clock at 8 MHz, remember to clear the clock divider fuse. Otherwise an external 8 MHz or 16 MHz clock source can be used, the definition F_CPU must be set to the frequency you chose (remember to set the fuses for an external clock source). As an example figure 1 shows using an external 16 MHz crystal.

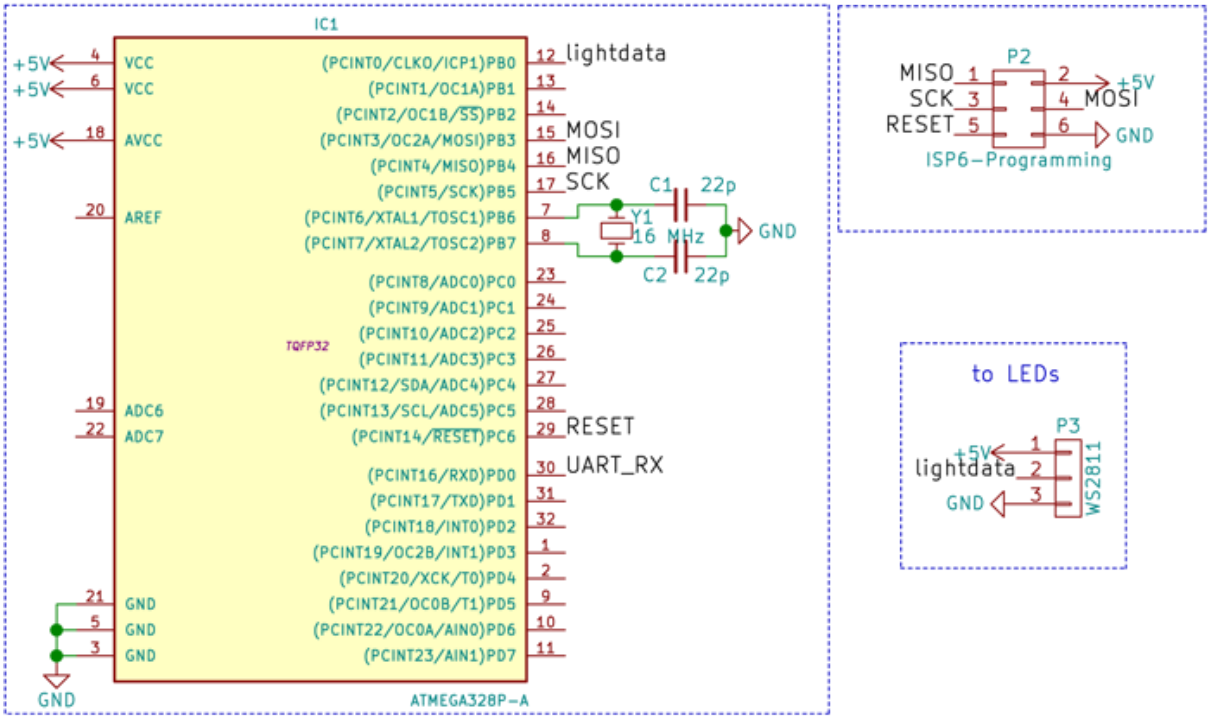

Figure 1: schematic of the AVR to controll WS2812/WS2811

As you can see in the picture the AVR is programmed by using the ISP interface. The WS2812/WS2811 get the same voltage as the AVR, the light data is available at PinB0, you may change this if you like. Referring to the LEDs be aware of the current amount they may draw if every LED has its full brightness. One WS2812 can draw up to 60 mA, so one meter with 30 LEDs already need 1,8 A. If you want to control more LEDs you may have a problem with the voltage drop along the stribe. For example if you control 180 LEDs at six meters you not only need 10,8 A, furthermore you will probably have a voltage drop up to 2 V. To reduce the voltage drop you must increase the wire size with parallel wires to you stribe. You can see the voltage drop if you set all LEDs to white. If you have only a small voltage drop every LED will have the same color. If the voltage drop is too much you can see that the last LEDs will have less blue color, so they will light in a warm white color even up to red. If you want to try out the LEDs with the AVR you can build up everything on a breadboard. Pinheaders can be soldered easy at the light stribes as you can see in figure 2.

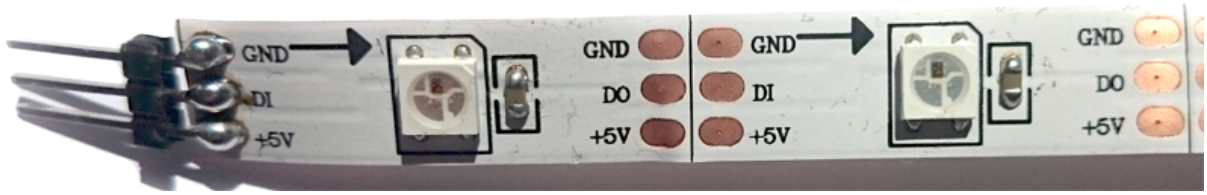

Figure 2: WS2812 stribe with pin header

The connect GND to the common ground with the AVR, 5 V should be connected to a power supply that can handle the current you need. DI is the data in line, this should be connected to PinB0 at the AVR. The stribe is like a big shifting register, all the data you sent is shifted bit by bit through the stribe. So DO is the data out pin, you see

some data at this pin if all LEDs before had already received their color data. The one wire protocol of the LEDs is described in the next section Software implementation.

Datasheets:
Datasheet WS2812
Datasheet WS2811
Datasheet Atmega328

1.4 Software implementation

If your hardware is ready you must flash your AVR device with the provided software. Therefore the ISP-6 connector should be used. To get the right timing remember to set the F_CPU definition to the frequency you are working at. Furthermore set the fuses of the AVR referring to your implementation. This means you have to clear the clock divider fuse and may have to change the clock source. I suggest to use the AtmelStudio to program your AVR and its fuses.
The WS2812/WS2811 are controlled by one data line that works with a one wire protocol. Because of the missing clock line the timing is really important, this can either be achieved by doing some trick with the hardware interfaces (e.g. using the spi interface) or by bit banging. In this implementation bit banging is used. To get a good timing all color data must be transmitted in one block that is not interrupted by some other code. The timing specifications of the WS2812/WS2811 LEDs can be found in table 1 which refers to the datasheet (**WS2812**).

Information	Timing	Tolerance +/-
Transfer 1 Bit	HighTime+LowTime=1,25 µs	600 ns
send 0, high time	0,35 µs	150 ns
send 0, low time	0,8 µs	150 ns
send 1, high time	0,7 µs	150 ns
send 1, low time	0,6 µs	150 ns
data transmission complete, low time	>50 µs	-

Table 1: Timing table for WS2812/WS2811 one wire protocol

The timing is done by setting the output and wait the required time by doing nothing (call assembly NOPs). So it is important to compile the provided software at O1, other optimization levels may influence the timing. To send one bit (either high or low) two different macros are defined in Lightstribe.h (SETHIGH and SETLOW), one LED needs 24 color bits. The macros depend on the value of F_CPU you entered in globals.h. Furthermore the header file Lightstrib.h declares a color struct to handle 24 bit colors (color24bit) and three basic functions to control the LEDs. The corresponding c file Lightstribe.c implements these functions. The most important function is the transmit2leds function. This function and only this function transmits data to the stribe. All other functions either call this function or manipulate the color array. To achieve the right timing all effects and operations are done on a color array that stores the color information for the LEDs. The information is sent to the LEDs by calling transmit2leds with the lightdata pointer that points to an dynamically allocated array that stores the color information depending on the number of LEDs you want to control. Therefore your color array must at least be able to contain 24 bits x your number of LEDs. It can be bigger, what will allow you to create even more effects (e.g. if you rotate a rainbow array). So the effects that are implemented in LedEffects.c change the color array and afterwards the transmit2leds is called. The c file LedEffects.c not only contains effects but also different necessary functions for the effects and the serial color handling. The colorconv8to24 function converts the received 8 bit colors from the serial port to 24 bit colors for the lightstribes. So you only sent 8 bit colors over the serial port to the AVR to reduce data size. Further information can be found in the Protocol overview section. The colors are decompressed with a simple map function you may know from Arduino. The main.c file initializes the hardware and handles the LEDs. A serial interrupt stores the data temporary. If the data transmission is complete the main function will extract the information and set the new configuration for the lightstribe.
The last points to be mentioned in this section are some things you need to be careful. The first thing is that the 8 bit colors are in an RGB 3-3-2 format. The 24 bit color format depend on the LEDs. WS2812 LEDs use a GRB color scheme while WS2811 use a RGB color scheme. This is important, to achieve the right color the protocol includes a bit that decides the color scheme. The right color is resolved by the decompressing function colorconv8to24.

Another thing is that the colors are not linearized, what means that you cannot say that a color you got from a color table will be look like this. As an example you picked an orange from a 3-3-2 rgb color table. This orange will not be the same orange on the LED stribe. This depends on many parameters so linearizing is too much effort and almost impossible (to achieve linearization you would have to measure each color, compare it and evaluate correction parameters).

1.5 Protocol overview

This section gives an overview of the implemented serial protocol. The goal of the protocol was to be as simple as possible, to be easily implemented on the AVR and to use as less resources as possible. The figure 3 shows the base structure of the protocol.

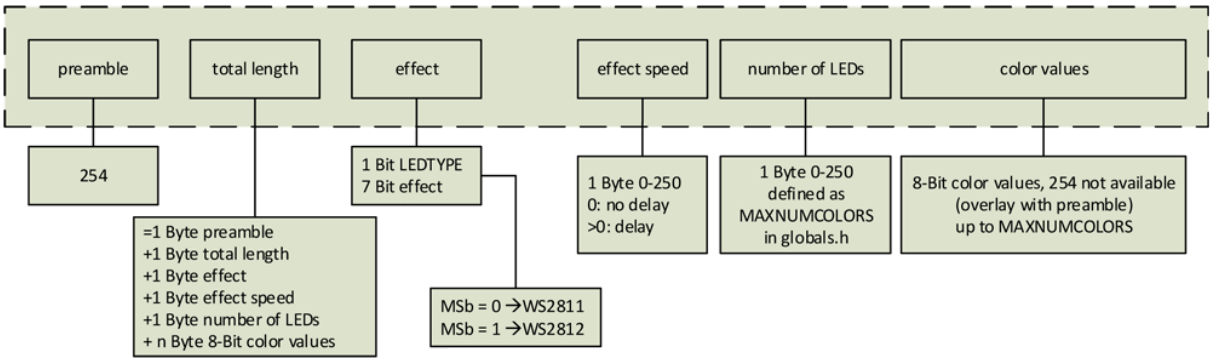

Figure 3: serial protocol structure

As you can see a data transmission always starts with the preamble 254(0xFE). For a fast and easy implementation this preamble value must only be used as preamble and must never be another field value (e.g. you must not send the color value 254). The next byte that is sent contains the total length of the packet including the preamble and the length byte. If you sent a wrong length you may get an unexpected behavior until a new correct data packet is sent. The third field contains the effect. The different effects are listed in table XXX. In Bit 7 (MSb) you can choose the LED type you want to control, set the bit to 0 for WS2811 and to 1 for WS2812 LEDs. The next byte is a value to control the effect speed. You can set a delay between 0 (no delay) and 250 (longest delay possible). The value unit and is not a repeatable setting for different effects. This means that the delay is no correct wait function (e.g. wait for n milliseconds). Furthermore the effects work on the color array what may be faster for some effects and slower for others. The best thing is to try the effects with different values. The next field contains the number of the LEDs that should be controlled. Be aware that the maximum supported number of LEDs is 250, but this depends on your hardware. The chosen Atmega328p can handle this amount, if you choose an AVR with less RAM this will not work. What you can do is to allocate more LEDs than you actually have. This gives you the possibility to create further effects. What happens is that only a part of the array is sent to the LEDs but the other color values are stored internally in the AVR (in fact all color data is transmitted to the LEDs but the superfluous information is overwritten by new data). The last data field are the color values. One color is 8 bit RGB 3-3-2 and you should sent the right amount of colors for your chosen effect. If you sent to less information the data block will not be evaluated because the total length does not match. For sending some data do not forget to configure your UART (8N1 BAUD) on both sides.

The missing numbers in table 2 are internally used by the AVR and must not be sent over the serial port.

1.6 Implement further effects

This section tells you how to implement further effects. You may use already existing functions to generate new effects or add something completely new. All effects should be written in the LedEffects.c file and declared in its header file LedEffects.h. You must know that everything works on a lightdata array that contains the colors stored in an array. The array is sent directly to the stribe if the transmit2leds function is called. So you first need to manipulate the array and than send it to the stribe. The array is ordered in GRB color because the implementation has

been done for WS2812 LEDs (WS2811 LEDs can be used the colors are converted in the colorconv8to24 function referring to the MSb of the effect you sent, for more information see section Protocol overview). So lightdata[0] contains one byte green data, lightdata[1] one byte red data, lightdata[2] blue data and so on. In general you can say lightdata[N%3==0] contains green, lightdata[N%3==1], lightdata[N%3==2] data. So the color array has a size of MAXNUMCOLORS ∗ 3. So your function must at least have a pointer to the lightdata array as a call value. For creating your effect some nice functions are already implemented you may use. You can find a list of them in table 3.

Your written effect should get an own definition in LedEffects.h . The last thing is to add your definition in the main switch case structure. Referring to the implemented protocol your effect is available with the number you defined in LedEffects.h. You must sent the neccessary information for your effect, for example the color values you need and so on. To get the color value you sent you need to call colorconv8to24 to convert the 8 bit RGB color into a 24 bit color. All colors you sent are available in the CompColorArray. The first color you sent is stored in index zero. Your implemented function must not care about the color order if you use the colorconv8to24 function. This does the conversion depending on the MSb of the effect you sent over the serial port. The delay is handled by the global var effectime and the number of LEDs to control is stored in NumOfLeds. The effect is stored in the effect variable. You should not do any changes on the serial part and the protocol reading, otherwise you will change to complete behavior of this implementation.

1.7 Requirements and Limitations

The implementation to control has the following requirements and limitations:

- colors are 8 bit compressed so you cannot get every color value of the LEDs
- the protocol implementation with the preamble 254 prohibits this value for other protocol fields (e.g. color)
- approximate amount of RAM (in bytes) you need: MAXNUMCOLORS(=number of LEDs to control)∗3 + U↩ ART_BUFFER_SIZE ∗2 + MAXNUMCOLORS + 160
- only O1 optimization is supported
- 8 MHz and 16 MHz clock support
- fuses must be programmed manually (clock source and clock divider)
- WS2801 stribes not supported (different hardware interface with two wires)
- AVR should run on 5 V

1.8 Example usage with an ESP8266

This section gives a short introduction about using the provided programm with an ESP8266. In this example the ESP8266 works as a wifi hotspot you can connect with and browse a website which allows different settings for the light stribe. The website is quite simple and only a few effects and colors are supported. If you enter the button "DO IT" your configuration is transmitted over the serial interface to the AVR. This is done through a software serial implementation, you find all necessary files below. You should step through all instructions to get the example work.

1.8.1 ESP8266 setup

First you need to setup the ESP8266. Because of different versions of ESP8266 modules you may miss something, this is just a quick guide. For more information you can browse the web. First you must connect your ESP8266 to a host computer over a serial interface for example using an FTDI. Remember to cross RX and TX of the serial port. Furthermore be aware of the ESP8266 voltage, it is 3,3 V. The current a serial chip may provide (some FTDIs provide some current) may not be enough for the ESP8266 and what can cause different problems.
So first you need to flash your ESP8266 with the nodemcu firmware that provides a software serial. The binaries that have been used in this example can be found here:
Binaries part 1
Binaries part 2

For uploading this binaries to the ESP8266 you should use the `nodeMCUFlasher` that can be found on github. You need to set the serial port to which your ESP8266 is connected with and configure the source files for flashing the firmware. You need to set the COM port to which you ESP8266 is connected to (see figure 4). Furthermore you must consider the following hardware configuration:

- 3,3 V logic level
- bootmode low (IO15)
- chip enable high (CH_PD)
- reset high (drive low to reset the module)
- IO0 low for firmware flashing (high for programming and normal operation)

The firmware programmer waits for the MAC of the ESP8266 module which will be successfully read if everything is done fine. As you can see in figure figure 4 the firmware programmer is still waiting for an ESP8266.

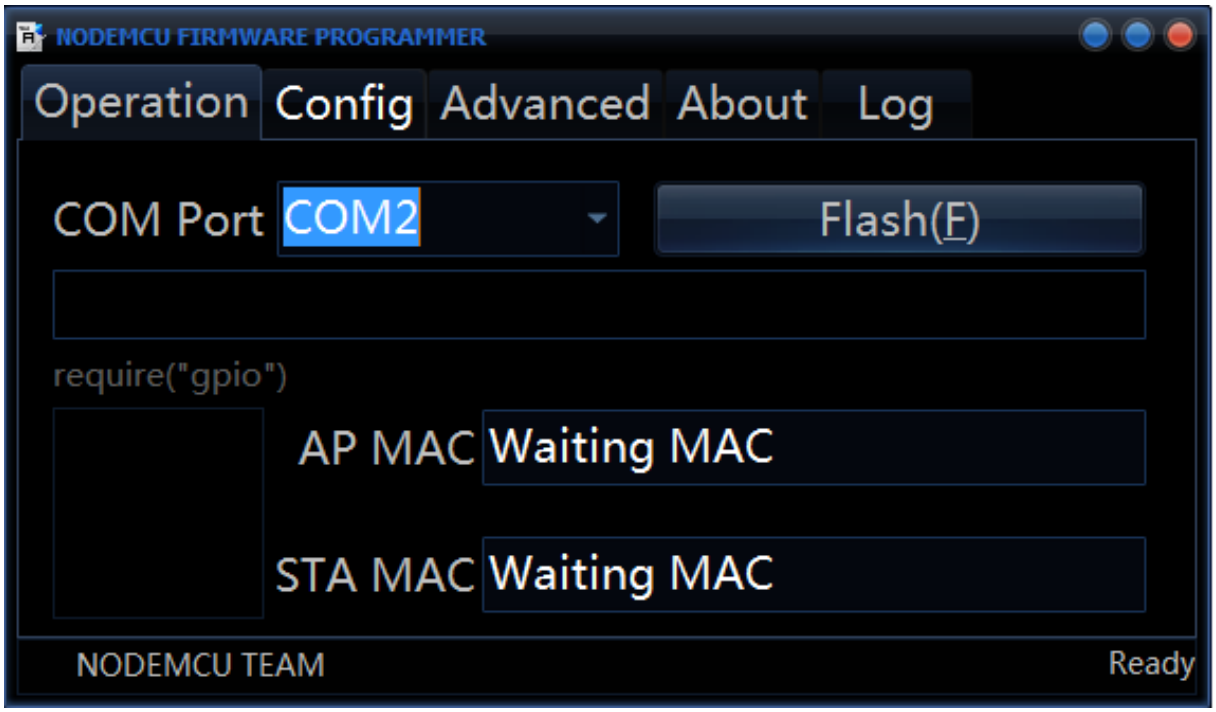

Figure 4: serial protocol structure

If the ESP8266 is connected right you now set the configuration to the provided binaries as you can see in figure 5. You must browse to the binary files and set the destination address. Now you can hit the "Flash"-Button (see figure 4).

Figure 5: serial protocol structure

After flashing the firmware you need to reboot the ESP8266 module. Before you do this you should change the IO0 to high level (3V3) because after the reboot we want to program the module with our own program. The reset can be done by setting reset low. The program we will upload to the module is written in Lua. Lua is a scripting language that is interpreted by the firmware running on the module. So the performance is not the best, but the programming is quite simple. The little program that you can find below set up the module as an access point, runs a simple webserver that interacts with a software serial to control the AVR. One thing you must know about the Lua programming is that the variable types are assigned implicit so you cannot control whether a number is stored as 16 bit or 32 bit signed or unsigned variable. Another thing you should know is that the program you write needs the full memory space of its file. That means shorter variable names save memory and furthermore documentation should be as short as possible or left.
For writing your program you can use any text editor, notepad++ is a good choice. If you are finished you must upload the file to the module. Therefore you use the same setup as for firmware updating but you must set IO0 to high level. For uploading your program you can use the ESP8266 Lua Loader. It is easy to handle and you can try out several things first, before you upload your code. You can find the main window of the ESP8266 Lua Loader in figure 6.

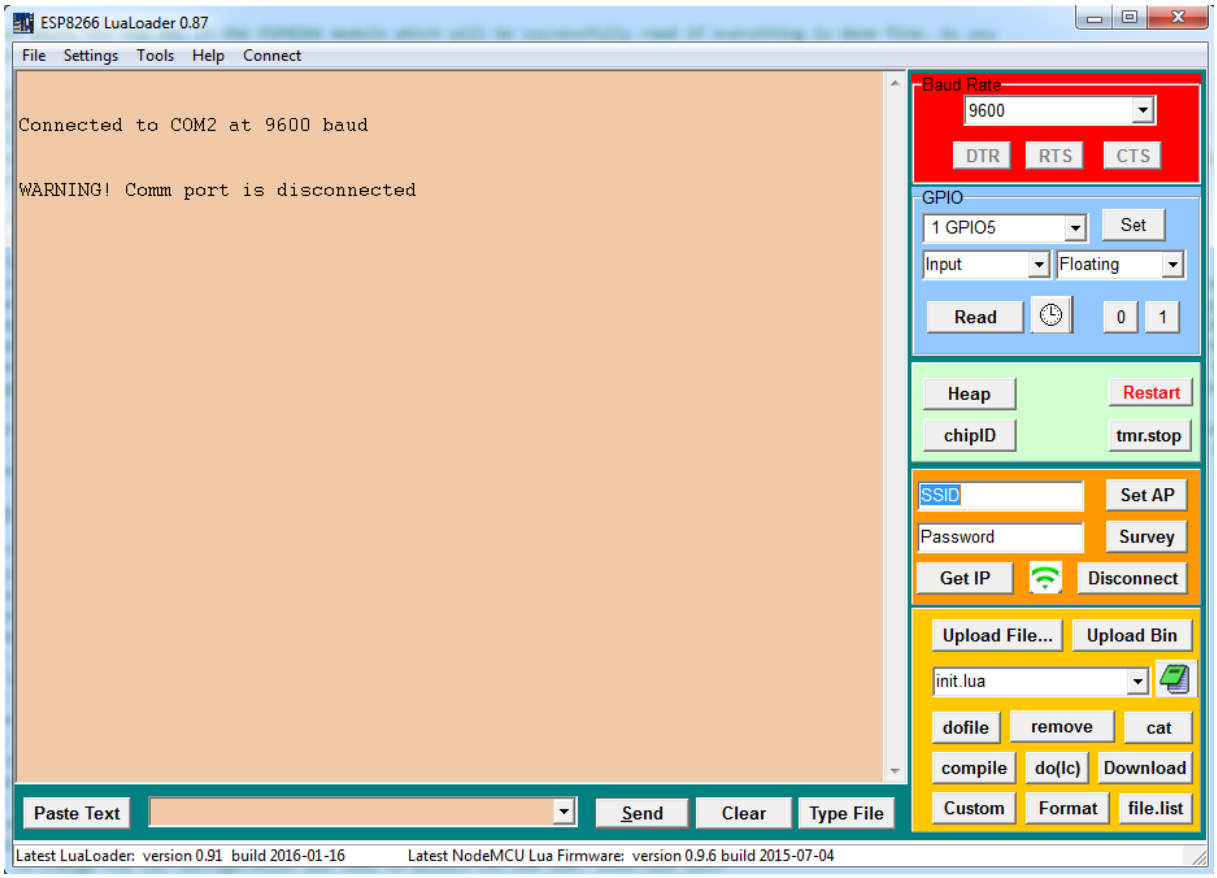

Figure 6: ESP8266 Lua Loader

On the right side you can set the baud rate for uploading you program to the module. In the GPIO section you can easily set and reset them to try your wiring. By using the restart button you will restart the module. This may be necessary if your heap (RAM) is to low. This is caused by inefficient programming or by a program that is to big for the module. Global variables need a lot of heap. For uploading your program hit the "Upload File..." button. In a file browser you choose your program that should be transferred to the module. After completion you hit the "dofile" button to run the program. This short description should be enough.
So now we upload the Lua program that starts the webserver and sends data over a software serial to the AVR. You can find this program here:
lua program for controlling the AVR
The program does the following:

- set up the ESP8266 module as an access point (SSID=Lichterkette, password=12345678, you may change this)
- start a webserver that listens on port 80
- load the index.html website and handle requests
- sent UART commands matching for the AVR implementation to generate different effects depending on the request

Some further things you should know:

- the maximum of parallel accessing devices is four
- parallel devices can never access another device
- the software uart only supports TX (8N1 up to 38400 baud)

- if you change the website you must change the hard coded content length of the website

After uploading the main program file you need to upload the **_index.html** file. Before uploading remove the underscore so that the files name is index.html (the underscore has been inserted because of conflicts with this html documentation). For uploading other file types (than lua programs) to your ESP8266 module you need to use the "Upload Bin" button of the Lua Loader. The file will be uploaded to the file system on the ESP8266.
Now your ESP8266 module is ready to try the first communication with the AVR.

author: Florian Wank, 2016

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

4 Data Structure Documentation

4.1 color24bit Struct Reference

24 Bit color structure RGB 8-8-8

```
#include <Lightstribe.h>
```

Data Fields

- uint8_t red

- uint8_t green
- uint8_t blue

4.1.1 Detailed Description

24 Bit color structure RGB 8-8-8

Definition at line 16 of file Lightstribe.h.

4.1.2 Field Documentation

4.1.2.1 uint8_t blue

8 Bit blue

Definition at line 19 of file Lightstribe.h.

Referenced by changeled(), colorconv8to24(), faden(), initrainbow(), resetstribe(), setfullcolor(), and setled().

4.1.2.2 uint8_t green

8 Bit green

Definition at line 18 of file Lightstribe.h.

Referenced by changeled(), colorconv8to24(), faden(), initrainbow(), resetstribe(), setfullcolor(), and setled().

4.1.2.3 uint8_t red

8 Bit red

Definition at line 17 of file Lightstribe.h.

Referenced by changeled(), colorconv8to24(), faden(), initrainbow(), resetstribe(), setfullcolor(), and setled().

The documentation for this struct was generated from the following file:

- Lightstribe.h

5 File Documentation

5.1 globals.h File Reference

file that contains basic and global definitions, changes should be done carefully

```
#include <stdint.h>
```

Macros

- #define **_STR_EXPAND**(tok) #tok
- #define **_STR**(tok) _STR_EXPAND(tok)
- #define **_CPU_INFO**(x) CPU_FREQUENCY##x
- #define EXTERN extern

  macro for global variable management
- #define BASELEDTYPE 11

  default LED type of the stribe (11 for WS2811, do not change here! change ledtype in main function!)
- #define MAXNUMCOLORS 50

definition for maximum number of different colors that can be handled at the same time (the maximum value should be 250, a higher value may result in an memory overflow refering to 2kByte (atmega328p))

- #define UART_BUFFER_SIZE 80

  definition for UART Buffer, must be at least MAXNUMCOLORS+5

- #define F_CPU 8000000

  CPU Frequency definition for avr delay function.

Variables

- EXTERN uint8_t NumOfLeds

  global variable for number of leds to control

- EXTERN uint16_t effectime

  global effectime for effect delays, a higher value means a higher delay

- EXTERN uint8_t ledtype

  global ledtype, 11 = WS2811 (RGB Color), 12 = WS2812 (GRB Color)

- EXTERN uint8_t CompColorArray [MAXNUMCOLORS]

  color array containing the received packed 8-Bit colors

- EXTERN uint8_t RecBuffer [UART_BUFFER_SIZE]

  receive buffer for UART communication

- EXTERN uint8_t BufferCounter

  counter for accessing the CompColorArray indices for data income

- EXTERN uint8_t DataLen

  variable to store the current packet length of the UART packet

- EXTERN uint8_t effect

  global effect variable to switch between the effects

- EXTERN uint8_t PacketComplete

  flag to store if a UART packet is complete; a packet is complete if the BufferCounter equals DataLen

- EXTERN uint8_t PaketStart

  flag to store if the PREAMBLE has been received

- EXTERN uint8_t IsReading

  flag to show if the RecBuffer is in copy process so that the array cannot be filled with new data from UART

- EXTERN volatile char ReceivedChar

  current data received from UART

5.1.1 Detailed Description

file that contains basic and global definitions, changes should be done carefully

Version

V1.00

Date

05.01.2016

Authors

Wank Florian

Definition in file globals.h.

5.2 globals.h

```
00001 /*************************************************************************/
00009 #include <stdint.h>
00010
00011 #ifndef GLOBALS_H_
00012 #define GLOBALS_H_
00013
00014 //macros to display infos for CPU Frequency or other defines
00015 #define _STR_EXPAND(tok) #tok
00016 #define _STR(tok) _STR_EXPAND(tok)
00017 #define _CPU_INFO(x) CPU_FREQUENCY##x
00018
00020 #ifndef EXTERN
00021 #define EXTERN extern
00022 #endif
00023
00025 EXTERN uint8_t NumOfLeds;
00027 EXTERN uint16_t effectime;
00029 EXTERN uint8_t ledtype;
00031 #define BASELEDTYPE 11
00032
00035 #define MAXNUMCOLORS 50
00036
00037 #define UART_BUFFER_SIZE 80
00038
00040 EXTERN uint8_t CompColorArray[MAXNUMCOLORS];
00042 EXTERN uint8_t RecBuffer[UART_BUFFER_SIZE];
00044 EXTERN uint8_t BufferCounter;
00046 EXTERN uint8_t DataLen;
00048 EXTERN uint8_t effect;
00049
00050 //EXTERN uint8_t speed;
00051
00053 EXTERN uint8_t PacketComplete;
00055 EXTERN uint8_t PaketStart;
00057 EXTERN uint8_t IsReading;
00059 EXTERN volatile char ReceivedChar;
00060
00062 #ifndef F_CPU
00063 #define F_CPU 8000000
00064 #endif
00065 #endif /* GLOBALS_H_ */
```

5.3 LedEffects.c File Reference

effect functions for controlling WS2811/WS2812 LEDs

```
#include "globals.h"
#include "Lightstribe.h"
#include "LedEffects.h"
#include <util/delay.h>
```

Functions

- uint8_t map (uint8_t x, uint8_t in_min, uint8_t in_max, uint8_t out_min, uint8_t out_max)

  Arduino map function; used for color conversion.
- struct color24bit colorconv8to24 (uint8_t startcolor)

  color conversion function; converts a 8 Bit color (RGB 3-3-2) to a 24 Bit color (RGB 8-8-8)
- void effectdelay (uint16_t delay)

  simple delay function; no concrete delay time
- void setfullcolor (struct color24bit color, uint8_t *lightdata)

  Set all LEDs to the chosen color; run transmit2leds afterwards to update the LEDs.
- void resetstribe (uint8_t *lightdata)

  Set all LEDs off; run transmit2leds afterwards to update the LEDs.
- void rotate (uint8_t *lightdata, uint8_t direction)

  Rotate the lightdata for 1 LED Position; run transmit2leds afterwards to update the LEDs.
- void rotateN (uint8_t *lightdata, uint8_t direction, uint8_t width)

Rotate the lightdata for n LED Positions; run transmit2leds afterwards to update the LEDs.

- void initrunled (struct color24bit color, uint8_t *lightdata, struct color24bit background)

  init the runled effect; run runrunled afterwards to start the effect
- void runrunled (uint8_t *lightdata, uint8_t direction)

  Do the runled effect; before this function is called the lightdata needs to be initiliazed using initrunled!
- void blinkled (struct color24bit color, uint8_t *lightdata)

  blink the whole stribe; this function does not need another function call
- void init_alternating (struct color24bit color, struct color24bit backcolor, uint8_t *lightdata)

  initialize the alternating function; call run_alternating afterwards
- void run_alternating (uint8_t *lightdata)

  Run the alternating effect; call init_alternating before.
- void recolor (struct color24bit color, uint8_t *lightdata)

  Recolor the LED stribe; no other function call is necessary.
- void faden (struct color24bit color, uint8_t *lightdata)

  Generate a fading color effect. No other function call is necessary.
- void initrainbow (uint8_t *lightdata)

  Initialize a rainbow on the color array; to show the rainbow run transmit2leds afterwards.
- void eastereggbase (struct color24bit color, uint8_t *lightdata)

  Initialize the easteregg; do not use directly; this function is used by the easteregg function.
- void easteregg (uint8_t *lightdata)

  Run the easteregg; No other function call is necessary.
- void fillup (struct color24bit color, struct color24bit backcolor, uint8_t *lightdata)

  This function fills up the stribe; No other function call is necessary.

5.3.1 Detailed Description

effect functions for controlling WS2811/WS2812 LEDs

This file contains different effect functions to control WS2811/WS2812 LEDs using an AVR. It also contains a conversion function to convert 8 Bit color values (RGB 3-3-2) to 24 Bit color values (RGB/GRB 8-8-8). The effects control first the lightdata array and then transmit the array data to the stribe. Using different operations result in different effects. You can add different functions if you like to. But remember that all operations need to be done on the lightdata array that needs to be transmitted at one block to the LEDs after your array has been changed.

Version

V1.00

Date

05.01.2016

Authors

Wank Florian

Definition in file LedEffects.c.

5.3.2 Function Documentation

5.3.2.1 void blinkled (struct color24bit *color*, uint8_t * *lightdata*)

blink the whole stribe; this function does not need another function call

This function creates a blinking effect. First all LEDs are set to the chosen color, after the defined delay the LEDs are turned off. This is repeated in the main while loop.

Parameters

in	*struct*	color24bit color : color for the blink effect
in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

No need to run transmit2leds afterwards! This is already done in the function.

Definition at line 278 of file LedEffects.c.

References effectdelay(), effectime, resetstribe(), setfullcolor(), and transmit2leds().

Referenced by main().

5.3.2.2 struct color24bit colorconv8to24 (uint8_t *startcolor*)

color conversion function; converts a 8 Bit color (RGB 3-3-2) to a 24 Bit color (RGB 8-8-8)

Parameters

in	*uint8_t*	startcolor: 8 Bit color to convert

Returns

struct color24bit : 24 Bit color result

Note

This function converts the 8 Bit color to a 24 Bit color depending on the ledtype. This is neccessary because of differnt color formats (WS2811->RGB ; WS2812->GRB). Original the whole environment was for WS2812 LEDs!

Definition at line 45 of file LedEffects.c.

References color24bit::blue, color24bit::green, ledtype, map(), and color24bit::red.

Referenced by easteregg(), and main().

5.3.2.3 void easteregg (uint8_t ∗ *lightdata*)

Run the easteregg; No other function call is necessary.

Parameters

in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

Just try it :-) funny looking effect

Definition at line 514 of file LedEffects.c.

References colorconv8to24(), eastereggbase(), and PacketComplete.

Referenced by main().

5.3.2.4 void eastereggbase (struct color24bit *color,* uint8_t * *lightdata*)

Initialize the easteregg; do not use directly; this function is used by the easteregg function.

Parameters

in	*struct*	color24bit color : color for the easteregg
in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

Do not use this function directly; this function is used by the easteregg function

Definition at line 489 of file LedEffects.c.

References changeled(), effectdelay(), effectime, NumOfLeds, PacketComplete, rotate(), and transmit2leds().

Referenced by easteregg().

5.3.2.5 void effectdelay (uint16_t *delay*)

simple delay function; no concrete delay time

Parameters

in	*uint16_t*	delay : delay value

Returns

void

Note

This function is just a variable delay, there is no coherence with a concrete time (i.e. s, ms)

Definition at line 72 of file LedEffects.c.

References PacketComplete.

Referenced by blinkled(), eastereggbase(), faden(), fillup(), main(), recolor(), run_alternating(), and runrunled().

5.3.2.6 void faden (struct color24bit *color,* uint8_t ∗ *lightdata*)

Generate a fading color effect. No other function call is necessary.

This function generates a fading color effect. At the beginning the whole stribe is filled with the chosen color. The color intensity of each color channel (blue, red, green) is decreased until the stribe is off. After that the color values are increased until the chosen color values are reached. The effect looks different depending on the chosen color because the color value proportion is not kept over the whole effect.

Parameters

in	*struct*	color24bit color : color that is used for the fading effect
in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

No need to run transmit2leds afterwards! The effect is standalone and ends is looped in the main while loop. The color value proportion is not kept over the whole effect.

Definition at line 366 of file LedEffects.c.

References color24bit::blue, effectdelay(), effectime, color24bit::green, PacketComplete, color24bit::red, setfullcolor(), and transmit2leds().

Referenced by main().

5.3.2.7 void fillup (struct color24bit *color*, struct color24bit *backcolor*, uint8_t * *lightdata*)

This function fills up the stribe; No other function call is necessary.

This function fills up the whole stribe and beginns again if it is finished. First one LED moves in the chosen color stepwise through the whole stribe and recolors all LEDs in the background color which have already been passed. At the end of the stribe the LED stays an the next single LED is going to move to the last-1 position. The next LED to the last-2 position. This is going on until the whole stribe is colored. Then the effect restarts (main while loop).

Parameters

in	*struct*	color24bit color : foreground color for the moving LED
in	*struct*	color24bit backcolor : background color
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

This is a standalone effect.

Definition at line 549 of file LedEffects.c.

References changeled(), effectdelay(), effectime, NumOfLeds, PacketComplete, and transmit2leds().

Referenced by main().

5.3.2.8 void init_alternating (struct color24bit *color*, struct color24bit *backcolor*, uint8_t * *lightdata*)

initialize the alternating function; call run_alternating afterwards

This function initializes the alternating effect. The effect assigns every even LED number in one color and the odd numbers in the background color. If the effect is running, the odd and even LED switch positions.

Parameters

in	*struct*	color24bit color : color for the alternate effect (Init even LEDs)
in	*struct*	color24bit backcolor : color for the alternate effect bakckground (Init odd LEDs)
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

Run run_alternating afterwards to start the effect!

Definition at line 300 of file LedEffects.c.

References changeled(), NumOfLeds, and setfullcolor().

Referenced by main().

5.3.2.9 void initrainbow (uint8_t * *lightdata*)

Initialize a rainbow on the color array; to show the rainbow run transmit2leds afterwards.

This function fills the color array with rainbow colors. For this effect the color array is filled with different colors that are calculated by increasing and decreasing the color channels to loop over a RGB palette.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

Run transmit2leds afterwards! A nice effect is to rotate the array stepwise after the rainbow initialization (run transmit2leds after every rotation). The effect directly sets color values, so there may be a problem with the color profiles (RGB vs. GRB). The function was primary written for WS2812 LEDs (GRB)! The effect needs a minimum number of 20 LEDs to look nice!

Definition at line 442 of file LedEffects.c.

References color24bit::blue, changeled(), color24bit::green, NumOfLeds, and color24bit::red.

Referenced by main().

5.3.2.10 void initrunled (struct color24bit *color,* uint8_t * *lightdata,* struct color24bit *background*)

init the runled effect; run runrunled afterwards to start the effect

This function initializes the running LED effect. The running LED effect has a background color that is used for all LEDs except one. One LED is in the foreground color an moves stepwise along the stribe. The initialization prepares the lightdata array by setting one LED at the start position and filling the others with the background color.

Parameters

in	*struct*	color24bit color : 24 Bit color for the effect
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe
in	*struct*	color24bit background : 24 Bit color for the effect background

Returns

void

Note

Run runrunled afterwards to start the effect!

Definition at line 217 of file LedEffects.c.

References changeled(), and setfullcolor().

Referenced by main().

5.3.2.11 uint8_t map (uint8_t *x,* uint8_t *in_min,* uint8_t *in_max,* uint8_t *out_min,* uint8_t *out_max*)

Arduino map function; used for color conversion.

Parameters

in	*uint8_t*	x: value to map
in	*uint8_t*	in_min : minimum value input reference
in	*uint8_t*	in_max : maximum value input reference
in	*uint8_t*	out_min : minimum value output reference
in	*uint8_t*	out_max : maximum value output reference

Returns

uint8_t : mapped value referring to the input

Note

This function is used for color conversion from 8 Bit to 24 Bit colors; How it works: in_min $<$ x $<$ in_max convert to out_min $<$ returnvalue $<$ out_max by positioning the x proportionally in the new number range

Definition at line 33 of file LedEffects.c.

Referenced by colorconv8to24().

5.3.2.12 void recolor (struct color24bit *color*, uint8_t * *lightdata*)

Recolor the LED stribe; no other function call is necessary.

This function generates a recolor effect. The old configuration of the LEDs is overwritten with the new color step by step. When the whole stribe is filled with the new color the effect ends.

Parameters

in	*struct*	color24bit color : color that is used for recoloring
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

No need to run transmit2leds afterwards! The effect is standalone and ends if the stribe is recolored.

Definition at line 340 of file LedEffects.c.

References changeled(), effectdelay(), effectime, NumOfLeds, PacketComplete, and transmit2leds().

Referenced by main().

5.3.2.13 void resetstribe (uint8_t * *lightdata*)

Set all LEDs off; run transmit2leds afterwards to update the LEDs.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

This function sets the lightdata array to 0x00. To update the stribe run transmit2leds afterwards!

Definition at line 118 of file LedEffects.c.

References color24bit::blue, color24bit::green, color24bit::red, and setfullcolor().

Referenced by blinkled().

5.3.2.14 void rotate (uint8_t * *lightdata*, uint8_t *direction*)

Rotate the lightdata for 1 LED Position; run transmit2leds afterwards to update the LEDs.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe
in	*uint8_t*	direction : direction to rotate

Returns

void

Note

This function rotates lightdata array. To update the stribe run transmit2leds afterwards! The rotation "moves every LED" by one step, the overflowing LED is appended at the other ending. Example: RED BLUE YELLOW GREEN ... rotate... BLUE YELLOW GREEN RED other direction: RED BLUE YELLOW GREEN ... rotate... GREEN RED BLUE YELLOW

Definition at line 138 of file LedEffects.c.

References NumOfLeds.

Referenced by eastereggbase(), main(), rotateN(), run_alternating(), and runrunled().

5.3.2.15 void rotateN (uint8_t * *lightdata*, uint8_t *direction*, uint8_t *width*)

Rotate the lightdata for n LED Positions; run transmit2leds afterwards to update the LEDs.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe
in	*uint8_t*	direction : direction to rotate
in	*uint8_t*	width : width to rotate

Returns

void

Note

This function rotates lightdata array. To update the stribe run transmit2leds afterwards! The rotation "moves every LED" by n steps, the overflowing LEDs are appended at the other ending. Example: RED BLUE YEL↩ LOW GREEN PINK ... rotate 2 ... YELLOW GREEN PINK RED BLUE other direction: RED BLUE YELLOW GREEN PINK ... rotate 2 ... GREEN PINK RED BLUE YELLOW

Definition at line 196 of file LedEffects.c.

References rotate().

5.3.2.16 void run_alternating (uint8_t * *lightdata*)

Run the alternating effect; call init_alternating before.

This function runs the alternating effect. The effect assigns every even LED number in one color and the odd numbers in the background color. If the effect is running, the odd and even LED switch positions. This function rotates the LEDs by one position to achieve the effect. The rotation direction is not of importance.

Parameters

in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

No need to run transmit2leds afterwards! The effect is generated by the main while loop.

Definition at line 323 of file LedEffects.c.

References effectdelay(), effectime, rotate(), and transmit2leds().

Referenced by main().

5.3.2.17 void runrunled (uint8_t ∗ *lightdata*, uint8_t *direction*)

Do the runled effect; before this function is called the lightdata needs to be initiliazed using initrunled!

This function runs the running LED effect. The running LED effect has a background color that is used for all LEDs except one. The one LED moves stepwise to the next position depending on the chosen direction. Direction 0/1 are right/left, direction 2 runs from left to right an back again. For direction 0/1 the running LED overflows and begins on the other ending.

Parameters

in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe
in	*uint8_t*	direction : movement direction, 0/1 = right/left, 2 = left->right and back

Returns

void

Note

No need to run transmit2leds afterwards! This is already done in the function. The function is interrupted if a new UART package is completely received so a new effect gets active.

Definition at line 236 of file LedEffects.c.

References effectdelay(), effectime, NumOfLeds, PacketComplete, rotate(), and transmit2leds().

Referenced by main().

5.3.2.18 void setfullcolor (struct color24bit *color*, uint8_t ∗ *lightdata*)

Set all LEDs to the chosen color; run transmit2leds afterwards to update the LEDs.

Parameters

in	*struct*	color24bit color : color to set
in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

This function sets the lightdata array. To update the stribe run transmit2leds afterwards!

Definition at line 96 of file LedEffects.c.

References color24bit::blue, color24bit::green, NumOfLeds, and color24bit::red.

Referenced by blinkled(), faden(), init_alternating(), initrunled(), main(), and resetstribe().

5.4 LedEffects.c

```
00001 /*************************************************************************/
00016 #include "globals.h"
00017 #include "Lightstribe.h"
00018 #include "LedEffects.h"
00019 #include <util/delay.h>
00020
00033 uint8_t map(uint8_t x, uint8_t in_min, uint8_t in_max, uint8_t out_min, uint8_t out_max)
00034 {
00035     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
00036 }
00037
00045 struct color24bit colorconv8to24(uint8_t startcolor)
00046 {
00047     struct color24bit color;
00048     if (ledtype==11)
00049     {   //color conversion for WS2811 LEDs (RGB color)
00050         //the converted values are assigned to the colors of the struct, red an green are switched
00051         //because of the different color profiles
00052         color.blue =map((0b00000011 & startcolor),0,3,0,255);    //2 Bit blue converted to 8 bit
00053         color.red=map((0b00011100 & startcolor)>>2,0,7,0,255);    //3 Bit green converted to 8 bit,
      assigned to red (color profiles!)
00054         color.green=map((0b11100000 & startcolor)>>5,0,7,0,255);//3 Bit red converted to 8 bit,
      assigned to green (color profiles!)
00055     }
00056     else
00057     {   //color conversion for WS2812 LEDs (GRB color)
00058         //the converted values are assigned to the colors of the struct
00059         //no color switching is done, the environment is for WS2812 LEDs (GRB)
00060         color.blue =map((0b00000011 & startcolor),0,3,0,255);    //2 Bit blue
00061         color.green=map((0b00011100 & startcolor)>>2,0,7,0,255);//3 Bit green
00062         color.red=map((0b11100000 & startcolor)>>5,0,7,0,255);    //3 Bit red
00063     }
00064     return color;
00065 }
00066
00072 void effectdelay(uint16_t delay)
00073 {
00074     uint16_t j;
00075     if (delay==0)
00076         return;
00077     do
00078     {
00079         j=2000;
00080         if (PacketComplete==1)    //interrupt the function if new settings have been received
00081             break;
00082         do
00083         {
00084             asm ("nop");
00085         } while (--j);
00086     } while (--delay);
00087
00088 }
00089
00096 void setfullcolor(struct color24bit color, uint8_t *lightdata)
00097 {
00098     uint8_t ledcolor;
00099     uint16_t i;
00100     for (i=0;i<NumOfLeds*3;i++)    //Loop over color array (lightdata)
00101     {
00102         ledcolor = i%3;
00103         //set the array elements
00104         if (ledcolor==0)
00105             *lightdata++=color.green;
00106         else if(ledcolor==1)
00107             *lightdata++=color.red;
00108         else
00109             *lightdata++=color.blue;
00110     }
00111 }
00112
00118 void resetstribe(uint8_t *lightdata)
```

```
00119 {
00120     struct color24bit color;
00121     color.blue = 0x00;
00122     color.green= 0x00;
00123     color.red = 0x00;
00124     setfullcolor(color, lightdata);
00125 }
00126
00138 void rotate(uint8_t *lightdata, uint8_t direction)
00139 {
00140     uint8_t temp1, temp2, temp3;
00141     uint8_t *tempp;
00142     uint16_t i;
00143
00144     if (direction==0)
00145     {
00146         //Store overflowing LED
00147         temp1 = *lightdata;
00148         temp2= *(lightdata+1);
00149         temp3 =*(lightdata+2);
00150         //Rotate the array (minus 1 LED-->overflow; 1 LED correlate three 8 Bit color values)
00151         for (i=0;i<NumOfLeds*3-3;i++)
00152         {   //increase the array pointer step by step
00153             *lightdata = *(lightdata+3);
00154             lightdata++;
00155         }
00156         //assign overflowed LED
00157         *lightdata++ = temp1;
00158         *lightdata++ = temp2;
00159         *lightdata++ = temp3;
00160
00161     }
00162     else
00163     {
00164         //Set a pointer to the end of the lightdata
00165         tempp = lightdata + NumOfLeds*3 -1;
00166         //Store overflowing LED
00167         temp1 = *tempp;
00168         temp2 = *(tempp-1);
00169         temp3 = *(tempp-2);
00170
00171         //Rotate the array (minus 1 LED-->overflow; 1 LED correlate three 8 Bit color values)
00172         for (i=0;i<(NumOfLeds*3-3);i++)
00173         {   //decrease the array pointer step by step
00174             *tempp = *(tempp-3);
00175             tempp--;
00176         }
00177         //assign overflowed LED
00178         *tempp--=temp1;
00179         *tempp--=temp2;
00180         *tempp = temp3;
00181     }
00182 }
00183
00196 void rotateN(uint8_t *lightdata, uint8_t direction, uint8_t width)
00197 {
00198     uint8_t i;
00199     for (i=0;i<width;i++)
00200     {
00201         rotate(lightdata,direction);
00202     }
00203 }
00204
00217 void initrunled(struct color24bit color, uint8_t *lightdata, struct
      color24bit background)
00218 {
00219     setfullcolor(background,lightdata);
00220     changeled(color, lightdata,0);
00221 }
00222
00236 void runrunled(uint8_t *lightdata, uint8_t direction)
00237 {
00238     uint8_t i;
00239
00240     //Run from left to right and back, one loop in this function, main while repeats the effect
00241     if (direction==2)
00242     {
00243         for (i=0;i<NumOfLeds;i++)
00244         {
00245             transmit2leds(lightdata);
00246             rotate(lightdata,1);
00247             effectdelay(effectime);
00248             if (PacketComplete==1)
00249                 break;
00250         }
00251         for (i=0;i<NumOfLeds;i++)
00252         {
```

```
00253
00254             rotate(lightdata,0);
00255             transmit2leds(lightdata);
00256             effectdelay(effectime);
00257             if (PacketComplete==1)
00258                 break;
00259         }
00260     }
00261     else
00262     {   //Only one rotation is done, main while does the effect
00263         rotate(lightdata,direction);
00264         transmit2leds(lightdata);
00265         effectdelay(effectime);
00266     }
00267 }
00268
00278 void blinkled(struct color24bit color, uint8_t *lightdata)
00279 {
00280     //Set the chosen color
00281     setfullcolor(color, lightdata);
00282     transmit2leds(lightdata);
00283     effectdelay(effectime);
00284     //Turn the stribe off
00285     resetstribe(lightdata);
00286     transmit2leds(lightdata);
00287     effectdelay(effectime);
00288 }
00289
00300 void init_alternating(struct color24bit color, struct
    color24bit backcolor, uint8_t *lightdata)
00301 {
00302     uint16_t i;
00303     setfullcolor(backcolor, lightdata);     //Set background color
00304     for (i=0;i<NumOfLeds;i++)
00305     {
00306         if(i%2==0)
00307         {
00308             changeled(color,lightdata,i);  //set the even LEDs
00309         }
00310     }
00311 }
00312
00323 void run_alternating(uint8_t *lightdata )
00324 {
00325     transmit2leds(lightdata);
00326     effectdelay(effectime);
00327     rotate(lightdata,1);
00328 }
00329
00340 void recolor(struct color24bit color, uint8_t *lightdata)
00341 {
00342     uint8_t i;
00343     for (i=0;i<NumOfLeds;i++)
00344     {
00345         changeled(color,lightdata,i);
00346         transmit2leds(lightdata);
00347         effectdelay(effectime);
00348         if (PacketComplete==1)
00349             break;
00350     }
00351 }
00352
00366 void faden(struct color24bit color, uint8_t *lightdata)
00367 {
00368     uint8_t i;
00369     uint8_t maxgreen, maxred, maxblue;
00370     maxgreen =color.green;
00371     maxblue = color.blue;
00372     maxred = color.red;
00373     for (i=0;i<255;i++) //Fade down to LED off
00374     {
00375         setfullcolor(color,lightdata);
00376         transmit2leds(lightdata);
00377         effectdelay(effectime);
00378         //Decrease the color values that are greater than 0, stop if every value is 0
00379         if (color.green > 0)
00380         {
00381             --color.green;
00382         }
00383         if (color.blue > 0)
00384         {
00385             --color.blue;
00386         }
00387         if (color.red > 0)
00388         {
00389             --color.red;
00390         }
```

```
00391         if (color.red == 0 && color.blue == 0 && color.green == 0)
00392         {
00393             break;
00394         }
00395         if (PacketComplete==1)
00396         {
00397             break;
00398         }
00399     }
00400
00401     for (i=0;i<255;i++) //Fade up to chosen color
00402     {
00403         setfullcolor(color,lightdata);
00404         transmit2leds(lightdata);
00405         effectdelay(effectime);
00406         //Increase the color values is they are lower than the chosen color value, stop if all maximums are
      reached
00407         if (color.green < maxgreen)
00408         {
00409             ++color.green;
00410         }
00411         if (color.blue < maxblue)
00412         {
00413             ++color.blue;
00414         }
00415         if (color.red < maxred)
00416         {
00417             ++color.red;
00418         }
00419         if (color.red == maxred && color.blue == maxblue && color.green == maxgreen)
00420         {
00421             break;
00422         }
00423         if (PacketComplete==1)
00424         {
00425             break;
00426         }
00427     }
00428 }
00429
00442 void initrainbow(uint8_t *lightdata)
00443 {
00444     uint8_t steps = NumOfLeds / 5;
00445     struct color24bit color;
00446     uint8_t i,j;
00447     //Start rainbow with red color
00448     color.red = 0xFF;
00449     color.blue= 0x00;
00450     color.green=0x00;
00451     j=0;
00452     for(i=0;i<NumOfLeds;i++)
00453     {
00454         if (j<steps)
00455         {
00456             color.blue = 0x00+0xFF/steps*j;     //increase blue to get violett
00457         }
00458         else if(j>steps && j<=2*steps)
00459         {
00460             color.red = 0xFF-0xFF/steps*(j/2);   //decrease red to get blue
00461         }
00462         else if(j>2*steps && j<=3*steps)
00463         {
00464             color.green = 0x00+0xFF/steps*(j/3);//increase green to get cyan
00465         }
00466         else if(j>3*steps && j<=4*steps)
00467         {
00468             color.blue = 0xFF-0xFF/steps*(j/4); //decrease blue to get green
00469         }
00470         else if(j>4*steps && j<=5*steps)
00471         {
00472             color.red = 0x00+0xFF/steps*(j/5);   //increase red to get yellow
00473         }
00474         else if(j>6*steps)
00475         {
00476             color.green = 0xFF-0xFF/steps*(j/6);//decrease green to get red
00477         }
00478         j++;
00479         changeled(color,lightdata,i);
00480     }
00481 }
00482
00489 void eastereggbase(struct color24bit color, uint8_t *lightdata)
00490 {
00491     uint8_t i,j;
00492     uint8_t n;
00493     j=NumOfLeds;
00494     for (i=0;i<NumOfLeds;i++)
```

```
00495     {
00496         n=(j-i);
00497         changeled(color,lightdata,0);
00498         while(n-->0)
00499         {
00500             rotate(lightdata,1);
00501             transmit2leds(lightdata);
00502             effectdelay(effectime);
00503         }
00504         if (PacketComplete==1)
00505         break;
00506     }
00507 }
00508
00514 void easteregg(uint8_t *lightdata)
00515 {
00516     struct color24bit color, color2;
00517     uint8_t i;
00518     color=colorconv8to24(252);
00519     color2=colorconv8to24(201);
00520     eastereggbase(color2,lightdata);
00521     for (i=0;i<100;i++)
00522     {
00523         if (PacketComplete==1)
00524         break;
00525         _delay_ms(50);
00526     }
00527     eastereggbase(color,lightdata);
00528     for (i=0;i<100;i++)
00529     {
00530         if (PacketComplete==1)
00531         break;
00532         _delay_ms(50);
00533     }
00534 }
00535
00549 void fillup(struct color24bit color,struct color24bit backcolor, uint8_t *
      lightdata)
00550 {
00551     uint8_t i,j;
00552     for (i=0;i<NumOfLeds;i++)
00553     {
00554         for (j=0;j<NumOfLeds-i;j++)
00555         {
00556             changeled(color,lightdata,j);          //running LED, foreground
00557             if (j>0)
00558             {
00559                 changeled(backcolor,lightdata,j-1);    //background LEDs
00560             }
00561             transmit2leds(lightdata);
00562             effectdelay(effectime);
00563         }
00564         if (PacketComplete==1)
00565             break;
00566         effectdelay(effectime);
00567     }
00568 }
```

5.5 LedEffects.h File Reference

file that contains different effect definitions for the lightstribe

```
#include <stdint.h>
```

Macros

- #define SETFULLCOLOR 0

  define for the setfullcolor effect, used for main switch
- #define FILLUP 1

  define for the the fillup effect, used for main switch
- #define BLINK 2

  define for the blink effect, used for main switch
- #define RUNLED 3

  define for the runled effect, used for main switch, refers to the runled init

- #define ALTERNATE 5

  define for the alternating effect, used for main switch, refers to the alternate init
- #define RECOLOR 7

  define for the recolor effect, used for main switch
- #define FADE 8

  define for the fade effect, used for main switch
- #define INITRAINBOW 9

  define for the initrainbow function, used for main switch
- #define ROTATE_R 10

  define for the the rotate function right, used for main switch
- #define ROTATE_L 11

  define for the the rotate function left, used for main switch
- #define CUSTOM 12

  define for the custom effect, used for main switch, every LED is filled in a userdefined color (up to MAXNUMCOLORS, then reloop the colors)
- #define EASTEREGG 13

  define for the easteregg effect, used for main switch

Functions

- uint8_t map (uint8_t x, uint8_t in_min, uint8_t in_max, uint8_t out_min, uint8_t out_max)

  Arduino map function; used for color conversion.
- struct color24bit colorconv8to24 (uint8_t startcolor)

  color conversion function; converts a 8 Bit color (RGB 3-3-2) to a 24 Bit color (RGB 8-8-8)
- void effectdelay (uint16_t delay)

  simple delay function; no concrete delay time
- void setfullcolor (struct color24bit color, uint8_t *lightdata)

  Set all LEDs to the chosen color; run transmit2leds afterwards to update the LEDs.
- void resetstribe (uint8_t *lightdata)

  Set all LEDs off; run transmit2leds afterwards to update the LEDs.
- void rotate (uint8_t *lightdata, uint8_t direction)

  Rotate the lightdata for 1 LED Position; run transmit2leds afterwards to update the LEDs.
- void rotateN (uint8_t *lightdata, uint8_t direction, uint8_t width)

  Rotate the lightdata for n LED Positions; run transmit2leds afterwards to update the LEDs.
- void initrunled (struct color24bit color, uint8_t *lightdata, struct color24bit background)

  init the runled effect; run runrunled afterwards to start the effect
- void runrunled (uint8_t *lightdata, uint8_t direction)

  Do the runled effect; before this function is called the lightdata needs to be initiliazed using initrunled!
- void blinkled (struct color24bit color, uint8_t *lightdata)

  blink the whole stribe; this function does not need another function call
- void init_alternating (struct color24bit color, struct color24bit backcolor, uint8_t *lightdata)

  initialize the alternating function; call run_alternating afterwards
- void run_alternating (uint8_t *lightdata)

  Run the alternating effect; call init_alternating before.
- void recolor (struct color24bit color, uint8_t *lightdata)

  Recolor the LED stribe; no other function call is necessary.
- void faden (struct color24bit color, uint8_t *lightdata)

  Generate a fading color effect. No other function call is necessary.
- void initrainbow (uint8_t *lightdata)

  Initialize a rainbow on the color array; to show the rainbow run transmit2leds afterwards.
- void eastereggbase (struct color24bit color, uint8_t *lightdata)

Initialize the easteregg; do not use directly; this function is used by the easteregg function.

- void easteregg (uint8_t *lightdata)

  Run the easteregg; No other function call is necessary.

- void fillup (struct color24bit color, struct color24bit backcolor, uint8_t *lightdata)

  This function fills up the stribe; No other function call is necessary.

5.5.1 Detailed Description

file that contains different effect definitions for the lightstribe

Version

V1.00

Date

05.01.2016

Authors

Wank Florian

Definition in file LedEffects.h.

5.5.2 Function Documentation

5.5.2.1 void blinkled (struct color24bit *color*, uint8_t * *lightdata*)

blink the whole stribe; this function does not need another function call

This function creates a blinking effect. First all LEDs are set to the chosen color, after the defined delay the LEDs are turned off. This is repeated in the main while loop.

Parameters

in	*struct*	color24bit color : color for the blink effect
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

No need to run transmit2leds afterwards! This is already done in the function.

Definition at line 278 of file LedEffects.c.

References effectdelay(), effectime, resetstribe(), setfullcolor(), and transmit2leds().

Referenced by main().

5.5.2.2 struct color24bit colorconv8to24 (uint8_t *startcolor*)

color conversion function; converts a 8 Bit color (RGB 3-3-2) to a 24 Bit color (RGB 8-8-8)

Parameters

in	*uint8_t*	startcolor: 8 Bit color to convert

Returns

struct color24bit : 24 Bit color result

Note

This function converts the 8 Bit color to a 24 Bit color depending on the ledtype. This is neccessary because of differnt color formats (WS2811->RGB ; WS2812->GRB). Original the whole environment was for WS2812 LEDs!

Definition at line 45 of file LedEffects.c.

References color24bit::blue, color24bit::green, ledtype, map(), and color24bit::red.

Referenced by easteregg(), and main().

5.5.2.3 void easteregg (uint8_t * *lightdata*)

Run the easteregg; No other function call is necessary.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

Just try it :-) funny looking effect

Definition at line 514 of file LedEffects.c.

References colorconv8to24(), eastereggbase(), and PacketComplete.

Referenced by main().

5.5.2.4 void eastereggbase (struct color24bit *color,* uint8_t * *lightdata*)

Initialize the easteregg; do not use directly; this function is used by the easteregg function.

Parameters

in	*struct*	color24bit color : color for the easteregg
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

Do not use this function directly; this function is used by the easteregg function

Definition at line 489 of file LedEffects.c.

References changeled(), effectdelay(), effectime, NumOfLeds, PacketComplete, rotate(), and transmit2leds().

Referenced by easteregg().

5.5.2.5 void effectdelay (uint16_t *delay*)

simple delay function; no concrete delay time

Parameters

in	*uint16_t*	delay : delay value

Returns

void

Note

This function is just a variable delay, there is no coherence with a concrete time (i.e. s, ms)

Definition at line 72 of file LedEffects.c.

References PacketComplete.

Referenced by blinkled(), eastereggbase(), faden(), fillup(), main(), recolor(), run_alternating(), and runrunled().

5.5.2.6 void faden (struct color24bit *color*, uint8_t * *lightdata*)

Generate a fading color effect. No other function call is necessary.

This function generates a fading color effect. At the beginning the whole stribe is filled with the chosen color. The color intensity of each color channel (blue, red, green) is decreased until the stribe is off. After that the color values are increased until the chosen color values are reached. The effect looks different depending on the chosen color because the color value proportion is not kept over the whole effect.

Parameters

in	*struct*	color24bit color : color that is used for the fading effect
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

No need to run transmit2leds afterwards! The effect is standalone and ends is looped in the main while loop. The color value proportion is not kept over the whole effect.

Definition at line 366 of file LedEffects.c.

References color24bit::blue, effectdelay(), effectime, color24bit::green, PacketComplete, color24bit::red, setfullcolor(), and transmit2leds().

Referenced by main().

5.5.2.7 void fillup (struct color24bit *color*, struct color24bit *backcolor*, uint8_t * *lightdata*)

This function fills up the stribe; No other function call is necessary.

This function fills up the whole stribe and beginns again if it is finished. First one LED moves in the chosen color stepwise through the whole stribe and recolors all LEDs in the background color which have already been passed. At the end of the stribe the LED stays an the next single LED is going to move to the last-1 position. The next LED to the last-2 position. This is going on until the whole stribe is colored. Then the effect restarts (main while loop).

Parameters

in	*struct*	color24bit color : foreground color for the moving LED
in	*struct*	color24bit backcolor : background color
in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

This is a standalone effect.

Definition at line 549 of file LedEffects.c.

References changeled(), effectdelay(), effectime, NumOfLeds, PacketComplete, and transmit2leds().

Referenced by main().

5.5.2.8 void init_alternating (struct color24bit *color,* struct color24bit *backcolor,* uint8_t ∗ *lightdata*)

initialize the alternating function; call run_alternating afterwards

This function initializes the alternating effect. The effect assigns every even LED number in one color and the odd numbers in the background color. If the effect is running, the odd and even LED switch positions.

Parameters

in	*struct*	color24bit color : color for the alternate effect (Init even LEDs)
in	*struct*	color24bit backcolor : color for the alternate effect bakckground (Init odd LEDs)
in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

Run run_alternating afterwards to start the effect!

Definition at line 300 of file LedEffects.c.

References changeled(), NumOfLeds, and setfullcolor().

Referenced by main().

5.5.2.9 void initrainbow (uint8_t ∗ *lightdata*)

Initialize a rainbow on the color array; to show the rainbow run transmit2leds afterwards.

This function fills the color array with rainbow colors. For this effect the color array is filled with different colors that are calculated by increasing and decreasing the color channels to loop over a RGB palette.

Parameters

in	*uint8_t*	∗lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

Run transmit2leds afterwards! A nice effect is to rotate the array stepwise after the rainbow initialization (run transmit2leds after every rotation). The effect directly sets color values, so there may be a problem with the color profiles (RGB vs. GRB). The function was primary written for WS2812 LEDs (GRB)! The effect needs a minimum number of 20 LEDs to look nice!

Definition at line 442 of file LedEffects.c.

References color24bit::blue, changeled(), color24bit::green, NumOfLeds, and color24bit::red.

Referenced by main().

5.5.2.10 void initrunled (struct color24bit *color*, uint8_t * *lightdata*, struct color24bit *background*)

init the runled effect; run runrunled afterwards to start the effect

This function initializes the running LED effect. The running LED effect has a background color that is used for all LEDs except one. One LED is in the foreground color an moves stepwise along the stribe. The initialization prepares the lightdata array by setting one LED at the start position and filling the others with the background color.

Parameters

in	*struct*	color24bit color : 24 Bit color for the effect
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe
in	*struct*	color24bit background : 24 Bit color for the effect background

Returns

void

Note

Run runrunled afterwards to start the effect!

Definition at line 217 of file LedEffects.c.

References changeled(), and setfullcolor().

Referenced by main().

5.5.2.11 uint8_t map (uint8_t *x*, uint8_t *in_min*, uint8_t *in_max*, uint8_t *out_min*, uint8_t *out_max*)

Arduino map function; used for color conversion.

Parameters

in	*uint8_t*	x: value to map
in	*uint8_t*	in_min : minimum value input reference
in	*uint8_t*	in_max : maximum value input reference
in	*uint8_t*	out_min : minimum value output reference
in	*uint8_t*	out_max : maximum value output reference

Returns

uint8_t : mapped value referring to the input

Note

This function is used for color conversion from 8 Bit to 24 Bit colors; How it works: in_min $<$ x $<$ in_max convert to out_min $<$ returnvalue $<$ out_max by positioning the x proportionally in the new number range

Definition at line 33 of file LedEffects.c.

Referenced by colorconv8to24().

5.5.2.12 void recolor (struct color24bit *color,* uint8_t * *lightdata*)

Recolor the LED stribe; no other function call is necessary.

This function generates a recolor effect. The old configuration of the LEDs is overwritten with the new color step by step. When the whole stribe is filled with the new color the effect ends.

Parameters

in	*struct*	color24bit color : color that is used for recoloring
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

No need to run transmit2leds afterwards! The effect is standalone and ends if the stribe is recolored.

Definition at line 340 of file LedEffects.c.

References changeled(), effectdelay(), effectime, NumOfLeds, PacketComplete, and transmit2leds().

Referenced by main().

5.5.2.13 void resetstribe (uint8_t * *lightdata*)

Set all LEDs off; run transmit2leds afterwards to update the LEDs.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

This function sets the lightdata array to 0x00. To update the stribe run transmit2leds afterwards!

Definition at line 118 of file LedEffects.c.

References color24bit::blue, color24bit::green, color24bit::red, and setfullcolor().

Referenced by blinkled().

5.5.2.14 void rotate (uint8_t * *lightdata,* uint8_t *direction*)

Rotate the lightdata for 1 LED Position; run transmit2leds afterwards to update the LEDs.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe
in	*uint8_t*	direction : direction to rotate

Returns

void

Note

This function rotates lightdata array. To update the stribe run transmit2leds afterwards! The rotation "moves every LED" by one step, the overflowing LED is appended at the other ending. Example: RED BLUE YELLOW GREEN ... rotate... BLUE YELLOW GREEN RED other direction: RED BLUE YELLOW GREEN ... rotate... GREEN RED BLUE YELLOW

Definition at line 138 of file LedEffects.c.

References NumOfLeds.

Referenced by eastereggbase(), main(), rotateN(), run_alternating(), and runrunled().

5.5.2.15 void rotateN (uint8_t * *lightdata*, uint8_t *direction*, uint8_t *width*)

Rotate the lightdata for n LED Positions; run transmit2leds afterwards to update the LEDs.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe
in	*uint8_t*	direction : direction to rotate
in	*uint8_t*	width : width to rotate

Returns

void

Note

This function rotates lightdata array. To update the stribe run transmit2leds afterwards! The rotation "moves every LED" by n steps, the overflowing LEDs are appended at the other ending. Example: RED BLUE YEL↩ LOW GREEN PINK ... rotate 2 ... YELLOW GREEN PINK RED BLUE other direction: RED BLUE YELLOW GREEN PINK ... rotate 2 ... GREEN PINK RED BLUE YELLOW

Definition at line 196 of file LedEffects.c.

References rotate().

5.5.2.16 void run_alternating (uint8_t * *lightdata*)

Run the alternating effect; call init_alternating before.

This function runs the alternating effect. The effect assigns every even LED number in one color and the odd numbers in the background color. If the effect is running, the odd and even LED switch positions. This function rotates the LEDs by one position to achieve the effect. The rotation direction is not of importance.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

No need to run transmit2leds afterwards! The effect is generated by the main while loop.

Definition at line 323 of file LedEffects.c.

References effectdelay(), effectime, rotate(), and transmit2leds().

Referenced by main().

5.5.2.17 void runrunled (uint8_t * *lightdata*, uint8_t *direction*)

Do the runled effect; before this function is called the lightdata needs to be initiliazed using initrunled!

This function runs the running LED effect. The running LED effect has a background color that is used for all LEDs except one. The one LED moves stepwise to the next position depending on the chosen direction. Direction 0/1 are right/left, direction 2 runs from left to right an back again. For direction 0/1 the running LED overflows and begins on the other ending.

Parameters

in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe
in	*uint8_t*	direction : movement direction, 0/1 = right/left, 2 = left->right and back

Returns

void

Note

No need to run transmit2leds afterwards! This is already done in the function. The function is interrupted if a new UART package is completely received so a new effect gets active.

Definition at line 236 of file LedEffects.c.

References effectdelay(), effectime, NumOfLeds, PacketComplete, rotate(), and transmit2leds().

Referenced by main().

5.5.2.18 void setfullcolor (struct color24bit *color*, uint8_t * *lightdata*)

Set all LEDs to the chosen color; run transmit2leds afterwards to update the LEDs.

Parameters

in	*struct*	color24bit color : color to set
in	*uint8_t*	*lightdata : lightdata array that holds the color values for the stribe

Returns

void

Note

This function sets the lightdata array. To update the stribe run transmit2leds afterwards!

Definition at line 96 of file LedEffects.c.

References color24bit::blue, color24bit::green, NumOfLeds, and color24bit::red.

Referenced by blinkled(), faden(), init_alternating(), initrunled(), main(), and resetstribe().

5.6 LedEffects.h

```
00001 /*************************************************************************/
00009 #include <stdint.h>
00010
00011 #ifndef LEDEFFECTS_H_
00012 #define LEDEFFECTS_H_
00013
00014 //EFFECTS
00016 #define SETFULLCOLOR 0
00017
00018 #define FILLUP 1
00019
```

```
00020 #define BLINK 2
00021
00022 #define RUNLED 3
00023
00024 #define ALTERNATE 5
00025
00026 #define RECOLOR 7
00027
00028 #define FADE 8
00029
00030 #define INITRAINBOW 9
00031
00032 #define ROTATE_R 10
00033
00034 #define ROTATE_L 11
00035
00036 #define CUSTOM 12
00037
00038 #define EASTEREGG 13
00039
00040 uint8_t map(uint8_t x, uint8_t in_min, uint8_t in_max, uint8_t out_min, uint8_t out_max);        //Map
       function for color conversion; calcualates a value in a new number range
00041 struct color24bit colorconv8to24(uint8_t startcolor);
      //Convert a 8 Bit color (RGB 3-3-2) to 24 Bit color (RGB 8-8-8); color assignment depends on the ledtype
00042 void effectdelay(uint16_t delay);
      //a simple variable delay function
00043 void setfullcolor(struct color24bit color, uint8_t *lightdata);
      //set the whole stribe in one color, call transmit2leds afterwards
00044 void resetstribe(uint8_t *lightdata);
      //set the whole stribe off, call transmit2leds afterwards
00045 void rotate(uint8_t *lightdata, uint8_t direction);                                              //
      rotate the color array by one position
00046 void rotateN(uint8_t *lightdata, uint8_t direction, uint8_t width);                              //
      rotate the color array by n positions
00047 void initrunled(struct color24bit color, uint8_t *lightdata, struct
      color24bit background);       //initialize the runled effect, call runrunled afterwards
00048 void runrunled(uint8_t *lightdata, uint8_t direction);                                          //
      runs the runled effect, call initrunled before
00049 void blinkled(struct color24bit color, uint8_t *lightdata);
      //generate a blinking effect
00050 void init_alternating(struct color24bit color, struct
      color24bit backcolor, uint8_t *lightdata);//initialize the alternating effect, call
       run_alternating afterwards
00051 void run_alternating(uint8_t *lightdata );
      //run the alternating effect, call init_alternating before
00052 void recolor(struct color24bit color, uint8_t *lightdata);
      //recolor the stribe step by step, stand alone function, ends after execution
00053 void faden(struct color24bit color, uint8_t *lightdata);
      //color fading effect, stand alone effect
00054 void initrainbow(uint8_t *lightdata);
      //init the stribe with rainbow colors, call transmit2leds afterwards
00055 void easthereggbase(struct color24bit color, uint8_t *lightdata);
      //part of the easteregg effect, do not call directly
00056 void easteregg(uint8_t *lightdata);
      //easteregg effect, try out and have fun :-)
00057 void fillup(struct color24bit color,struct color24bit backcolor, uint8_t *
      lightdata);         //fill the stribe step by step until the stribe has one color, the background color is filled
       behind
00058
00059 #endif /* LEDEFFECTS_H_ */
```

5.7 Lightstribe.c File Reference

basic functions for controlling WS2811/WS2812 LEDs

```
#include "globals.h"
#include "Lightstribe.h"
#include <util/delay.h>
```

Functions

- void changeled (struct color24bit color, uint8_t *lightdata, uint8_t lednr)

  change the color of one LED at a specific position; run transmit2leds afterwards to update the LEDs

- void setled (struct color24bit color, uint8_t *lightdata, uint8_t lednr)

  set the color of one LED at a specific position, all others are off; run transmit2leds afterwards to update the LEDs

- void transmit2leds (uint8_t lightdata[])
  transmit the color array to the stribe

5.7.1 Detailed Description

basic functions for controlling WS2811/WS2812 LEDs

This file contains the basic functions to control WS2811/WS2812 LEDs using an AVR. It declares the function to transmit lightdata to a stribe using the one wire protocol. For the right timing be aware of the crystal frequency! This code is written for using an extern clock of 16 MHz, if you change it you need to modify the number of NOPs in the macros defined in the header file. This file also contains the basic functions to set or to change one LED in the stribe. The whole system is working with a color array that stores the 24 Bit colors for all LEDs in an GRB format (WS2812). Every effect changes the array, after that the array is sent out by the transmit2leds function. This guarantees a correct timing. The most functions base on uint8_t variables so the maximum length of the stribe to control contains 255 LEDs. This should not be changed because you have hardware limitations as well that will limit a basic setup to 200-250 LEDs.

Version

V1.00

Date

05.01.2016

Authors

Wank Florian

Definition in file Lightstribe.c.

5.7.2 Function Documentation

5.7.2.1 void changeled (struct color24bit *color,* uint8_t * *lightdata,* uint8_t *lednr*)

change the color of one LED at a specific position; run transmit2leds afterwards to update the LEDs

Parameters

in	*struct*	color24bit color : 24 bit color in GRB format
in	*uint8_t*	*lightdata : pointer to the complete lightdata that contains all color values
in	*uint8_t*	lednr : position of the LED that should be changed

Returns

void

Note

the right color format is created using the colorconv8to24-function with the ledtype predefined

Definition at line 33 of file Lightstribe.c.

References color24bit::blue, color24bit::green, NumOfLeds, and color24bit::red.

Referenced by eastereggbase(), fillup(), init_alternating(), initrainbow(), initrunled(), main(), and recolor().

5.7.2.2 void setled (struct color24bit *color,* uint8_t * *lightdata,* uint8_t *lednr*)

set the color of one LED at a specific position, all others are off; run transmit2leds afterwards to update the LEDs

Parameters

in	*struct*	color24bit color : 24 bit color in GRB format
in	*uint8_t*	∗lightdata : pointer to the complete lightdata that contains all color values
in	*uint8_t*	lednr : position of the LED that should be set

Returns

void

Note

the right color format is created using the colorconv8to24-function with the ledtype predefined; all other LEDs are cleared so they are off

Definition at line 51 of file Lightstribe.c.

References color24bit::blue, color24bit::green, NumOfLeds, and color24bit::red.

5.7.2.3 void transmit2leds (uint8_t *lightdata[ ]*)

transmit the color array to the stribe

To control the LEDs of type WS2811/WS2812 a critical timing is necessary. To achieve the correct timing and to create effects the lightdata is stored in an array first. All operations effect the color array. If the color array is prepared it is transmitted to the stribes via a one-wire protocol using this function. This function generates the high and low times using assembler NOPs to achieve the timing. The number of NOPs are stored in macros for transmitting a Low Bit (SETLOW) or a High Bit (SETHIGH). This function should not be changed or optimized because of the timing!

Parameters

in	*uint8_t*	lightdata[] : data with the colors for each LED to control

Returns

void

Note

This function should not be changed or optimized because of the timing! Do not use higher optimization than O1!!! Do not remove the {} brackets because SETLOW/SETHIGH are definitions with several commands!

Definition at line 96 of file Lightstribe.c.

References NumOfLeds.

Referenced by blinkled(), eastereggbase(), faden(), fillup(), main(), recolor(), run_alternating(), and runrunled().

5.8 Lightstribe.c

```
00001 /*************************************************************************/
00022 #include "globals.h"
00023 #include "Lightstribe.h"
00024 #include <util/delay.h>
00025
00033 void changeled(struct color24bit color, uint8_t *lightdata, uint8_t lednr)
00034 {
00035     if (lednr>NumOfLeds)
00036         return;
00037     lightdata=lightdata+lednr*3;
00038     *lightdata++=color.green;
00039     *lightdata++=color.red;
00040     *lightdata++=color.blue;
00041 }
```

```
00042
00051 void setled(struct color24bit color, uint8_t *lightdata, uint8_t lednr)
00052 {
00053     uint8_t ledcolor;
00054     uint16_t i;
00055     if (lednr>NumOfLeds)
00056         return;
00057     //Loop over the whole color array (-->NumOfLeds*3)
00058     for (i=0;i<NumOfLeds*3;i++)
00059     {
00060         if (i==(lednr*3) || i==(lednr*3+1) || i==(lednr*3+2))
00061         {   //position of the LED to set
00062             ledcolor = i%3;
00063             if (ledcolor==0)
00064                 *lightdata++=color.green;
00065             else if(ledcolor==1)
00066                 *lightdata++=color.red;
00067             else
00068                 *lightdata++=color.blue;
00069         }
00070         else
00071         {   //all others off (0x00-->black)
00072             ledcolor = i%3;
00073             if (ledcolor==0)
00074                 *lightdata++=0x00;
00075             else if(ledcolor==1)
00076                 *lightdata++=0x00;
00077             else
00078                 *lightdata++=0x00;
00079         }
00080     }
00081 }
00082
00096 void transmit2leds(uint8_t lightdata[])
00097 {
00098     uint16_t i ;
00099     uint8_t byte2send ;
00100     for(i=0;i<NumOfLeds*3;i++)
00101     {
00102         byte2send = lightdata[i];
00103         //Transmit each Bit of one Byte using the One Wire Protocoll
00104         if ((byte2send & 128)==0)
00105         {
00106             SETLOW
00107         }
00108         else
00109         {
00110             SETHIGH
00111         }
00112         if ((byte2send & 64)==0)
00113         {
00114             SETLOW
00115         }
00116         else
00117         {
00118             SETHIGH
00119         }
00120         if ((byte2send & 32)==0)
00121         {
00122             SETLOW
00123         }
00124         else
00125         {
00126             SETHIGH
00127         }
00128         if ((byte2send & 16)==0)
00129         {
00130             SETLOW
00131         }
00132         else
00133         {
00134             SETHIGH
00135         }
00136         if ((byte2send & 8)==0)
00137         {
00138             SETLOW
00139         }
00140         else
00141         {
00142             SETHIGH
00143         }
00144         if ((byte2send & 4)==0)
00145         {
00146             SETLOW
00147         }
00148         else
00149         {
```

```
00150             SETHIGH
00151         }
00152         if ((byte2send & 2)==0)
00153         {
00154             SETLOW
00155         }
00156         else
00157         {
00158             SETHIGH
00159         }
00160         if ((byte2send & 1)==0)
00161         {
00162             SETLOW
00163         }
00164         else
00165         {
00166             SETHIGH
00167         }
00168     }
00169     _delay_us(55);      //defined delay after the transmission is complete (Datasheet says >=50us)
00170 }
```

5.9 Lightstribe.h File Reference

basic functions for controlling WS2811/WS2812 LEDs

```
#include <stdint.h>
#include <avr/io.h>
```

Data Structures

- struct color24bit

  24 Bit color structure RGB 8-8-8

Functions

- void changeled (struct color24bit color, uint8_t *lightdata, uint8_t lednr)

  change the color of one LED at a specific position; run transmit2leds afterwards to update the LEDs
- void setled (struct color24bit color, uint8_t *lightdata, uint8_t lednr)

  set the color of one LED at a specific position, all others are off; run transmit2leds afterwards to update the LEDs
- void transmit2leds (uint8_t lightdata[])

  transmit the color array to the stribe

5.9.1 Detailed Description

basic functions for controlling WS2811/WS2812 LEDs

Version

V1.00

Date

05.01.2016

Authors

Wank Florian

Definition in file Lightstribe.h.

5.9.2 Function Documentation

5.9.2.1 void changeled (struct color24bit *color,* uint8_t * *lightdata,* uint8_t *lednr*)

change the color of one LED at a specific position; run transmit2leds afterwards to update the LEDs

Parameters

in	*struct*	color24bit color : 24 bit color in GRB format
in	*uint8_t*	∗lightdata : pointer to the complete lightdata that contains all color values
in	*uint8_t*	lednr : position of the LED that should be changed

Returns

void

Note

the right color format is created using the colorconv8to24-function with the ledtype predefined

Definition at line 33 of file Lightstribe.c.

References color24bit::blue, color24bit::green, NumOfLeds, and color24bit::red.

Referenced by eastereggbase(), fillup(), init_alternating(), initrainbow(), initrunled(), main(), and recolor().

5.9.2.2 void setled (struct color24bit *color,* uint8_t ∗ *lightdata,* uint8_t *lednr*)

set the color of one LED at a specific position, all others are off; run transmit2leds afterwards to update the LEDs

Parameters

in	*struct*	color24bit color : 24 bit color in GRB format
in	*uint8_t*	∗lightdata : pointer to the complete lightdata that contains all color values
in	*uint8_t*	lednr : position of the LED that should be set

Returns

void

Note

the right color format is created using the colorconv8to24-function with the ledtype predefined; all other LEDs are cleared so they are off

Definition at line 51 of file Lightstribe.c.

References color24bit::blue, color24bit::green, NumOfLeds, and color24bit::red.

5.9.2.3 void transmit2leds (uint8_t *lightdata[ ]*)

transmit the color array to the stribe

To control the LEDs of type WS2811/WS2812 a critical timing is necessary. To achieve the correct timing and to create effects the lightdata is stored in an array first. All operations effect the color array. If the color array is prepared it is transmitted to the stribes via a one-wire protocol using this function. This function generates the high and low times using assembler NOPs to achieve the timing. The number of NOPs are stored in macros for transmitting a Low Bit (SETLOW) or a High Bit (SETHIGH). This function should not be changed or optimized because of the timing!

Parameters

in	*uint8_t*	lightdata[] : data with the colors for each LED to control

Returns

void

Note

This function should not be changed or optimized because of the timing! Do not use higher optimization than O1!!! Do not remove the {} brackets because SETLOW/SETHIGH are definitions with several commands!

Definition at line 96 of file Lightstribe.c.

References NumOfLeds.

Referenced by blinkled(), eastereggbase(), faden(), fillup(), main(), recolor(), run_alternating(), and runrunled().

5.10 Lightstribe.h

```
00001 /*************************************************************************/
00009 #include <stdint.h>
00010 #include <avr/io.h>
00011
00012 #ifndef LIGHTSTRIBE_H_
00013 #define LIGHTSTRIBE_H_
00014
00016 struct color24bit{
00017     uint8_t red;
00018     uint8_t green;
00019     uint8_t blue;
00020 };
00021
00022 #if F_CPU == 16000000
00023 #pragma message("Use 16 MHz Macros")
00024
00025 #define SETHIGH PORTB=0x01;\
00026                 asm ("nop");\
00027                 asm ("nop");\
00028                 asm ("nop");\
00029                 asm ("nop");\
00030                 asm ("nop");\
00031                 asm ("nop");\
00032                 asm ("nop");\
00033                 asm ("nop");\
00034                 asm ("nop");\
00035                 asm ("nop");\
00036                 asm ("nop");\
00037                 PORTB=0x00;\
00038                 asm ("nop");\
00039                 asm ("nop");\
00040                 asm ("nop");
00041
00042 #elif F_CPU == 8000000
00043 #pragma message("Use 8 MHz Macros")
00044
00045 #define SETHIGH PORTB=0x01;\
00046                 asm ("nop");\
00047                 asm ("nop");\
00048                 asm ("nop");\
00049                 asm ("nop");\
00050                 asm ("nop");\
00051                 PORTB=0x00;\
00052                 asm ("nop");\
00053                 asm ("nop");
00054 #endif
00055
00056
00057
00058 #if F_CPU == 16000000
00059
00060 #define SETLOW PORTB=0x01;\
00061               asm ("nop");\
00062               asm ("nop");\
00063               asm ("nop");\
00064               asm ("nop");\
00065               asm ("nop");\
00066               PORTB=0x00;\
00067               asm ("nop");\
00068               asm ("nop");\
00069               asm ("nop");\
00070               asm ("nop");\
00071               asm ("nop");\
00072               asm ("nop");\
00073               asm ("nop");\
00074               asm ("nop");\
00075               asm ("nop");
00076 #elif F_CPU == 8000000
00077
```

```
00078 #define SETLOW PORTB=0x01;\
00079                asm ("nop");\
00080                asm ("nop");\
00081                PORTB=0x00;\
00082                asm ("nop");\
00083                asm ("nop");\
00084                asm ("nop");
00085 #endif
00086
00087
00088 //function to change one LED at a specific position; all other LEDs are not changed; run transmit2leds
       afterwards
00089 void changeled(struct color24bit color, uint8_t *lightdata, uint8_t lednr);
00090 //function to set one LED at a specific position; all other LEDs are turned off; run transmit2leds
       afterwards
00091 void setled(struct color24bit color, uint8_t *lightdata, uint8_t lednr);
00092 //transmit the color array to the stribe --> one wire data transmission
00093 void transmit2leds(uint8_t lightdata[]);
00094
00095 #endif /* LIGHTSTRIBE_H_ */
```

5.11 ws2811lichterkette.c File Reference

main file for interfacing WS2811/WS2812 LEDs

```
#include "globals.h"
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Lightstribe.h"
#include "LedEffects.h"
```

Macros

- #define **EXTERN**
- #define BAUD 38400

  Baudrate definition, choose 76800 or 38400, faster value preferred, the maximum speed of ESP8266 software-UART is 38400.
- #define MYUBRR F_CPU/16/BAUD -1

  calculate baudrate register value
- #define **BAUD_REAL** (F_CPU/(16∗(MYUBRR+1))) /∗real baudrate in this configuration∗/
- #define **BAUD_ERROR** ((BAUD_REAL∗1000)/BAUD) /∗calculate baudrate error∗/
- #define PREAMBLE 254

  definition of the preamble is 254, no other data field must contain this value
- #define LENINDEX 1

  definition of the second field; contains the total packet length (including the preamble)
- #define EFFECTINDEX 2

  definition of 1 Byte effect at third position, the MSBit is used to choose WS2811/WS2812 (color profile RGB or GRB)
- #define DELAYINDEX 3

  definition of the delay field, contains the delay duplicator
- #define NUMOFLEDINDEX 4

  field position for the number of LEDs to control

Functions

- void init_uart (void)

  Init the hardware UART with Baud = 76800/38400, depending on BAUD definition, 8 Databits, 1 Stopbit, no Parity.

- int main (void)

  main function, should never end, effects are handled in main while
- ISR (USART_RX_vect)

  UART Interrupt handler, interrupts when new data is available in the RX buffer.

5.11.1 Detailed Description

main file for interfacing WS2811/WS2812 LEDs

This file contains the main environment for interfacing WS2811/WS2812 LEDs with an AVR. The implementation has been done for an atmega328p. You may use another controller but be aware of the memory you need for the color array (dynamically allocated). The AVR interfaces the one wire of the LEDs. All operations (effects, colorchange etc.) are done on an lightdata array, that needs to be transmitted to the LEDs after your operations. The reason for this is the critical timing for interfacing the LEDs. So also be aware if you change the clock speed. If you do so you have to change the number of NOPs in the macros of Lightstribe.h. Because of the critical timing compile all files at optimization O1! Furthermore be aware of the BAUDRATE changes, the BAUD error may be to worse if you change the CPU frequency.
The one wire output is on the PIN B0! You can change in the main and Lightstribe.h.
By default this file just initializes the AVR system, no updates to the LEDs are done by default. To change the LED configuration you need to access the AVR UART Interface with another controller (FTDI is also possible). Over the UART you send a message containing all relevant information for the system. Therefore a simple protocol is used: 1 Byte preamble (254) 1 Byte total packet length (including the preamble) 1 Byte effect 1 Byte effect delay (effect speed) 1 Byte number of LEDs to control n Bytes containing 8-Bit color values (RGB 3-3-2), depended on the effect, max. 50 values The preamble 254 must never be used at another position!!!
Protocol examples:
SETFULLCOLOR: 254 6 0 1 20 22
FILLUP: 254 7 1 22 20 22 201
BLINK: 254 6 2 55 20 56
RUNLED: 254 7 3 55 20 56 151
INITRAINBOW: 254 5 9 0 20
ROTATE_R: 254 5 11 23 20
CUSTOM: 254 8 12 1 20 22 201 60
EASTEREGG: 254 5 13 2 20

The UART communication is done by using an RX interrupt an storing the data into a temp array. In the main loop a flag shows if a data packet is complete. So you will get no update on the LEDs if the UART package was wrong (too short). In the project this programm has been written the UART was controlled by an ESP8266 or BLE113. Have Fun!

Version

V1.00

Date

05.01.2016

Authors

Wank Florian

Definition in file ws2811lichterkette.c.

5.11.2 Function Documentation

5.11.2.1 void init_uart (void)

Init the hardware UART with Baud = 76800/38400, depending on BAUD definition, 8 Databits, 1 Stopbit, no Parity.

Returns

void

Note

This function depends on the oscillator clock frequency and the BAUD defintion. If your UART is not working first check all frequency issues (Fuse settings, clock speed, clock divider, Baudrate)

Definition at line 399 of file ws2811lichterkette.c.

References MYUBRR.

Referenced by main().

5.12 ws2811lichterkette.c

```
00001
00296 /************************************************************************/
00341 //define global variables
00342 #define EXTERN
00343 #include "globals.h"
00344
00345 #include <avr/io.h>
00346 #include <util/delay.h>
00347 #include <avr/interrupt.h>
00348 #include <stdio.h>
00349 #include <stdlib.h>
00350 #include <string.h>
00351
00352 #include "Lightstribe.h"
00353 #include "LedEffects.h"
00354
00355 //UART basic definitions
00357 #define BAUD 38400
00358
00359 #define MYUBRR F_CPU/16/BAUD -1
00360
00361 #define BAUD_REAL (F_CPU/(16*(MYUBRR+1)))       /*real baudrate in this configuration*/
00362 #define BAUD_ERROR ((BAUD_REAL*1000)/BAUD)      /*calculate baudrate error*/
00363 #if ((BAUD_ERROR<990) || (BAUD_ERROR>1010))
00364     #error baudrate error greater 1% !          /*show an error message if the baudrate error is greater
       than 1%*/
00365 #endif
00366
00367 //Protocol definition for UART communication
00368 //The protocol is defined as:
00369 //1 Byte preamble (254)
00370 //1 Byte total packet length (including the preamble)
00371 //1 Byte effect
00372 //1 Byte effect delay (effect speed)
00373 //1 Byte number of LEDs to control
00374 //n Bytes containing 8-Bit color values (RGB 3-3-2), depended on the effect, max. 50 values
00375
00377 #define PREAMBLE 254
00378
00379 #define LENINDEX 1
00380
00381 #define EFFECTINDEX 2
00382
00383 #define DELAYINDEX 3
00384
00385 #define NUMOFLEDINDEX 4
00386
00387
00388 //compiling info output
00389 #pragma message("MYUBRR: " _STR(MYUBRR))
00390 #pragma message("CPU Frequency: " _STR(F_CPU) "Hz")
00391 #pragma message("Baudrate: " _STR(BAUD))
00392 #pragma message("Configuration: MAXNUMCOLORS=" _STR(MAXNUMCOLORS)  " | UART_BUFFER_SIZE="
       _STR(UART_BUFFER_SIZE) " | PREAMBLE=" _STR(PREAMBLE))
00393
00399 void init_uart(void)
00400 {
00401     DDRD |= _BV(PD1);
00402     DDRD &= ~_BV(PD0);
00403
00404      //Set BAUD
00405      UBRR0H = ((MYUBRR) >> 8);
00406      UBRR0L = MYUBRR;
```

```
00407
00408      UCSR0B |= (1 << RXEN0) ;//| (1 << TXEN0);      // Enable receiver (and transmitter; commited out)
00409      UCSR0B |= (1 << RXCIE0);                       // Enable the receiver interrupt
00410      UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);       // 8 data Bit, one stop Bit
00411 }
00412
00414 int main(void)
00415 {
00416     uint16_t i,j;                            //helper variables (counters)
00417     uint8_t TempBuffer[UART_BUFFER_SIZE];    //Temp. buffer for copy of the UART data to
      achieve data consistency
00418     uint8_t *lightdata;                      //lightdata pointer for lightdata array; the array size is
      dynamic to controll different numbers of LEDs
00419
00420     NumOfLeds=50;                           //default number of LEDs is 50 => one stribe
00421     //Flag initializations
00422     PacketComplete=0;
00423     IsReading=0;
00424     PaketStart=0;
00425     BufferCounter = 0;
00426
00427     memset(RecBuffer,0,sizeof(RecBuffer[0])*UART_BUFFER_SIZE);     //clear
      the buffer
00428     memset(TempBuffer,0,sizeof(RecBuffer[0])*UART_BUFFER_SIZE);     //clear the buffer
00429     ledtype = BASELEDTYPE;                    //set default ledtype, 11 =>WS2811, 12
      =>WS2812
00430
00431     //Set the LED output Port (Pin B0 is used for LED data output)
00432     DDRB = 0x01;
00433     PORTB = 0x00;
00434
00435     //Basic initializations
00436     ReceivedChar = 1;
00437     effectime = 10;
00438     effect=255;
00439     BufferCounter=0;
00440
00441     init_uart();                            //Init the hardware UART
00442     sei();                                   //enable global interrupts
00443
00444     //main system loop
00445     while(1){
00446         if (PacketComplete==1)    //new UART package containing color and effect data is
      available
00447         {
00448             //Prohibit the access to the UART RecBuffer while copying the data to a Temp Buffer
00449             IsReading=1;
00450             PaketStart=0;
00451             memcpy(TempBuffer,RecBuffer,DataLen);   //Copy the UART data to a temp array
00452             effect=TempBuffer[EFFECTINDEX] & 0x7F; //get the effect from the temp array
00453             effectime=TempBuffer[DELAYINDEX];        //get the delay time for the effect
      form the temp array
00454             ledtype=BASELEDTYPE+((TempBuffer[EFFECTINDEX] & 0x80)>>7);//
      configure the ledtype depending on the MSBit of the effect
00455             NumOfLeds=TempBuffer[NUMOFLEDINDEX];  //get the number of leds to control
00456             IsReading=0;                           //allow access to the UART RecBuffer
00457             memcpy(CompColorArray,&TempBuffer[5],DataLen-5);   //generate compressed
      color array
00458             if (lightdata!=NULL)
00459             {
00460                 free(lightdata);
00461             }
00462             lightdata = (uint8_t *) malloc (NumOfLeds*3);      //allocate the lightdata array for
      uncompressed colors
00463             PacketComplete=0;                    //reset PacketComplete flag
00464         }
00465         else
00466         {
00467             //main switch for effect handling
00468             switch(effect)
00469             {
00470                 case SETFULLCOLOR:
00471                     setfullcolor(colorconv8to24(
      CompColorArray[0]),lightdata);
00472                     transmit2leds(lightdata);
00473                     break;
00474                 case FILLUP:
00475                     fillup(colorconv8to24(CompColorArray[0]),
      colorconv8to24(CompColorArray[1]),lightdata);
00476                     transmit2leds(lightdata);
00477                     break;
00478                 case BLINK:
00479                     blinkled(colorconv8to24(CompColorArray[0]),
      lightdata);
00480                     break;
00481                 case RUNLED:
00482                     initrunled(colorconv8to24(
```

```
      CompColorArray[0]),lightdata,colorconv8to24(
      CompColorArray[1]));
00483                     effect++;
00484                 case 4:
00485                     runrunled(lightdata,1);
00486                     break;
00487                 case ALTERNATE:
00488                     init_alternating(colorconv8to24(
      CompColorArray[0]),colorconv8to24(CompColorArray[1]),lightdata);
00489                     effect++;
00490                 case 6:
00491                     run_alternating(lightdata);
00492                     break;
00493                 case RECOLOR:
00494                     recolor(colorconv8to24(CompColorArray[0]),lightdata)
      ;
00495                     effect=255;
00496                     break;
00497                 case FADE:
00498                     faden(colorconv8to24(CompColorArray[0]),lightdata);
00499                     break;
00500                 case INITRAINBOW:
00501                     initrainbow(lightdata);
00502                     transmit2leds(lightdata);
00503                     break;
00504                 case ROTATE_R:
00505                     rotate(lightdata,0);
00506                     effectdelay(effectime);
00507                     transmit2leds(lightdata);
00508                     break;
00509                 case ROTATE_L:
00510                     rotate(lightdata,1);
00511                     effectdelay(effectime);
00512                     transmit2leds(lightdata);
00513                     break;
00514                 case CUSTOM:
00515                 //The custom effect assigns up to MAXNUMCOLORS individual colors to the stribe
00516                 //if the number of colors is smaller than the number of LEDs the colors are repeated using
00517                 //modulo operation
00518                     for (i=0;i<NumOfLeds;i++)
00519                     {
00520                         j = i % (DataLen-5);
00521                         changeled(colorconv8to24(
      CompColorArray[j]),lightdata,i);
00522                     }
00523                     transmit2leds(lightdata);
00524                     effect=255;
00525                     break;
00526                 case EASTEREGG:
00527                     easteregg(lightdata);
00528                     break;
00529                 default:    //do nothing
00530                     break;
00531             }
00532         }
00533 
00534 
00535     }
00536 
00537 }
00538 
00539 
00541 ISR (USART_RX_vect)
00542 {
00543     ReceivedChar = UDR0;                            //Read data from the RX buffer
00544     if (ReceivedChar==PREAMBLE && IsReading==0)        //Store data in the
       RecBuffer array only if it is not accessed by the main function
00545     {
00546         PacketComplete=0;
00547         PaketStart=1;                              //Set packet start flag (-->254=PREAMBLE has
       been received)
00548         memset(RecBuffer,0,sizeof(RecBuffer[0])*
      UART_BUFFER_SIZE);//clear the buffer
00549         BufferCounter=0;
00550         RecBuffer[0]=ReceivedChar;                 //Store the preamble
00551     }
00552     else if (PaketStart==1)
00553     {
00554         //Store all Bytes after the preamble
00555         BufferCounter++;
00556         RecBuffer[BufferCounter]=ReceivedChar;
00557         DataLen=RecBuffer[LENINDEX];                //Store data len of the data
       packet (preamble included)
00558         if (DataLen==BufferCounter+1)
00559         {
00560             PacketComplete=1;                      //a whole packet has been received, update
       the effect in main
```

```
00561         }
00562     }
00563 }
```

Effect number	Number of colors (1 byte RGB 3-3-2)	Description	Example (decimal)
0 = SETFULLCOLOR	1	All LEDs glow at the same color without changes.	254 6 1 20 22
1 = FILLUP	2 (foreground, background)	One LED steps through the stribe in the foreground color and colors all LEDs after it in the background color. At the end of the stribe the LED stays at the foreground color and another LED starts to step through the stribe. This continues until the whole stribe is filled in the foreground color. Then the stribe is cleared to the background color and the effects begins again.	254 7 1 22 20 22 201
2 = BLINK	1	The stribe blinks in the chosen color and to off (=black) repeatedly.	254 6 2 55 20 56
3 = RUNLED	2 (foreground, background)	All LEDs but one are colored in the background color. The one in the foreground color walks through the stribe with overflowing to the beginning.	254 7 3 55 20 56 151
5 = ALTERNATE	2 (foreground, background)	The LEDs are alternating in the foreground and the background color. First the even LEDs are colored in foreground and the uneven in the background color, after that vice versa.	254 7 5 55 20 56 151
7 = RECOLOR	1	The stribe is filled in a new color step by step until the whole stribe stays in the new color.	254 6 7 55 20 38
8 = FADE	1	The destination color is set and the base colors red, green and blue are decreased step by step until the stribe is off. After that the color values are increased until the destination color is reached. This generates a color fading effect. The color fading is not linearized.	254 6 8 55 20 201
9 = INITRAINBOW	no color	Set the stribe in a static rainbow color.	254 5 9 0 20
10 = ROTATE_R	no color	Rotate all LEDs one step to the right side (depends on lightdata array).	254 5 10 232 20
11 = ROTATE_L	no color	Rotate all LEDs one step to the left side (depends	254 5 11 23 20

Function Name	call values	operation
map	x,in_min,in_max,out_min,out_max	calculate an x value to a new number range
effectdelay	delay	wait some time dependend on delay
resetstribe	∗lightdata	clear the stribe (all LEDs off)
rotate	∗lightdata, direction	rotate stribe by one position (means 3 bytes) in direction (right/left)
rotateN	∗lightdata, direction,width	rotate LEDs by "width" positions (means width ∗ 3 bytes) in direction (right/left)
setled	color, ∗lightdata, lednr	set one LED a position lednr in the chosen color, others off (black)
changeled	color, ∗lightdata, lednr	change the color of one LED at position lednr, others are unchanged

Table 3: Provided help functions for your own effect

Index