Lightstribe WS2811/WS2812

Generated by Doxygen 1.8.10

Thu May 5 2016 13:11:01

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 control via ESP8266 section shows how this could be done by using a webserver on the ESP8266. 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 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 the figure one shows using an external 16 MHz crystal.

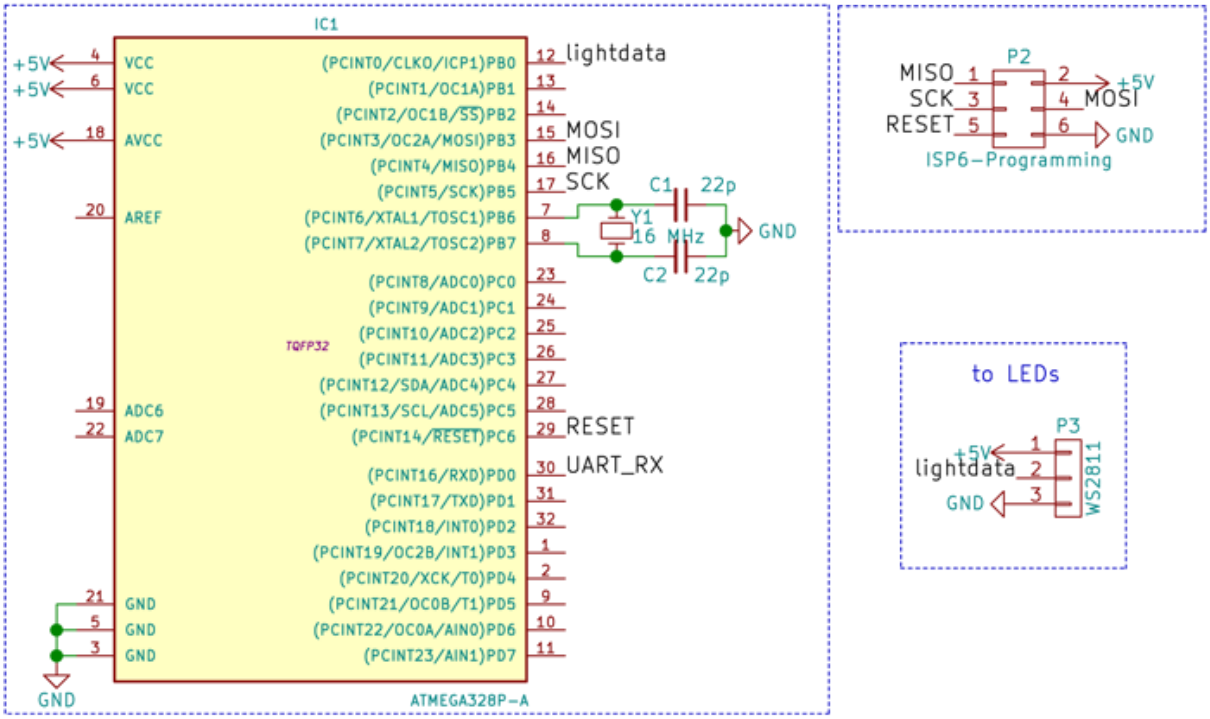

Figure 1: schematic of the AVR to controll WS2812/WS2811

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 the figure two.

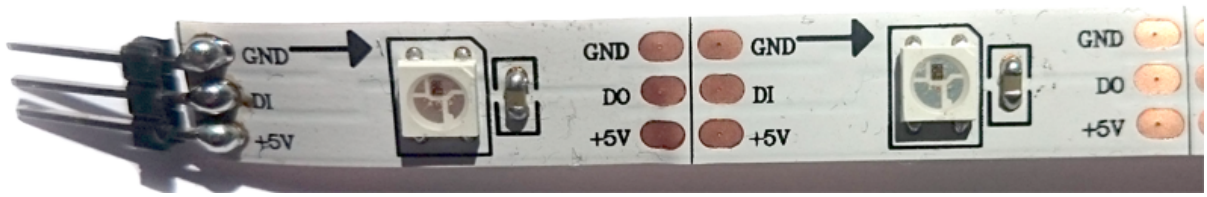

Figure 2: WS2812 stribe with pin header

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 software implementation section.

1.3 software implementation

1.4 Installation

1.4.1 Step 1: Opening the box

etc...

1.5 protocol overview

1.6 control via ESP8266

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.

4.1.2.2 uint8_t green

8 Bit green

Definition at line 18 of file Lightstribe.h.

4.1.2.3 uint8_t red

8 Bit red

Definition at line 17 of file Lightstribe.h.

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, no changes should be done here

```
#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 50, 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, no changes should be done here

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.

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.

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.

5.3.2.4 void easteregbase (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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 FADEN 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.

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.

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.

5.5.2.4 void easteregbase (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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 FADEN 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 eastereggbase(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.

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.

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.

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.

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.

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.

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 #define SETHIGH PORTB=0x01;\
00045                 asm ("nop");\
00046                 asm ("nop");\
00047                 asm ("nop");\
00048                 asm ("nop");\
00049                 asm ("nop");\
00050                 PORTB=0x00;\
00051                 asm ("nop");\
00052                 asm ("nop");
00053 #endif
00054
00055
00056
00057 #if F_CPU == 16000000
00058
00059 #define SETLOW PORTB=0x01;\
00060               asm ("nop");\
00061               asm ("nop");\
00062               asm ("nop");\
00063               asm ("nop");\
00064               asm ("nop");\
00065               PORTB=0x00;\
00066               asm ("nop");\
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 #elif F_CPU == 8000000
00076 #define SETLOW PORTB=0x01;\
00077                asm ("nop");\
00078                asm ("nop");\
00079                PORTB=0x00;\
00080                asm ("nop");\
00081                asm ("nop");\
00082                asm ("nop");
00083 #endif
00084
00085
00086 //function to change one LED at a specific position; all other LEDs are not changed; run transmit2leds
       afterwards
00087 void changeled(struct color24bit color, uint8_t *lightdata, uint8_t lednr);
00088 //function to set one LED at a specific position; all other LEDs are turned off; run transmit2leds
       afterwards
00089 void setled(struct color24bit color, uint8_t *lightdata, uint8_t lednr);
00090 //transmit the color array to the stribe --> one wire data transmission
00091 void transmit2leds(uint8_t lightdata[]);
```

```
00092
00093 #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 prefered, 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, should be max. 200 (dynamic memory allocation for the lightdata array)

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 174 of file ws2811lichterkette.c.

5.12 ws2811lichterkette.c

```
00001
00071 /*************************************************************************/
00116 //define global variables
00117 #define EXTERN
00118 #include "globals.h"
00119
00120 #include <avr/io.h>
00121 #include <util/delay.h>
00122 #include <avr/interrupt.h>
00123 #include <stdio.h>
00124 #include <stdlib.h>
00125 #include <string.h>
00126
00127 #include "Lightstribe.h"
00128 #include "LedEffects.h"
00129
00130 //UART basic definitions
00132 #define BAUD 38400
00133
00134 #define MYUBRR F_CPU/16/BAUD -1
00135
00136 #define BAUD_REAL (F_CPU/(16*(MYUBRR+1)))       /*real baudrate in this configuration*/
00137 #define BAUD_ERROR ((BAUD_REAL*1000)/BAUD)      /*calculate baudrate error*/
00138 #if ((BAUD_ERROR<990) || (BAUD_ERROR>1010))
00139     #error baudrate error greater 1% !          /*show an error message if the baudrate error is greater
      than 1%*/
00140 #endif
00141
00142 //Protocol definition for UART communication
00143 //The protocol is defined as:
00144 //1 Byte preamble (254)
00145 //1 Byte total packet length (including the preamble)
00146 //1 Byte effect
00147 //1 Byte effect delay (effect speed)
00148 //1 Byte number of LEDs to control
00149 //n Bytes containing 8-Bit color values (RGB 3-3-2), depended on the effect, max. 50 values
00150
00152 #define PREAMBLE 254
00153
00154 #define LENINDEX 1
00155
00156 #define EFFECTINDEX 2
00157
00158 #define DELAYINDEX 3
00159
00160 #define NUMOFLEDINDEX 4
00161
00162
00163 //compiling info output
00164 #pragma message("MYUBRR: " _STR(MYUBRR))
00165 #pragma message("CPU Frequency: " _STR(F_CPU) "Hz")
00166 #pragma message("Baudrate: " _STR(BAUD))
00167 #pragma message("Configuration: MAXNUMCOLORS=" _STR(MAXNUMCOLORS)  " | UART_BUFFER_SIZE="
      _STR(UART_BUFFER_SIZE) " | PREAMBLE=" _STR(PREAMBLE))
00168
00174 void init_uart(void)
00175 {
00176     DDRD |= _BV(PD1);
00177     DDRD &= ~_BV(PD0);
00178
00179      //Set BAUD
00180      UBRR0H = ((MYUBRR) >> 8);
00181      UBRR0L = MYUBRR;
00182
00183      UCSR0B |= (1 << RXEN0) ;//| (1 << TXEN0);     // Enable receiver and transmitter
00184      UCSR0B |= (1 << RXCIE0);                    // Enable the receiver interrupt
00185      UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);    // 8 data Bit, one stop Bit
00186 }
00187
00189 int main(void)
00190 {
00191     uint16_t i,j;                           //helper variables (counters)
00192     uint8_t TempBuffer[UART_BUFFER_SIZE];   //Temp. buffer for copy of the UART data to
      achieve data consistency
```

```
00193     uint8_t *lightdata;                      //lightdata pointer for lightdata array; the array size is
      dynamic to controll different numbers of LEDs
00194
00195     NumOfLeds=50;                           //default number of LEDs is 50 => one stribe
00196     //Flag initializations
00197     PacketComplete=0;
00198     IsReading=0;
00199     PaketStart=0;
00200     BufferCounter = 0;
00201
00202     memset(RecBuffer,0,sizeof(RecBuffer[0])*UART_BUFFER_SIZE);     //clear
      the buffer
00203     memset(TempBuffer,0,sizeof(RecBuffer[0])*UART_BUFFER_SIZE);     //clear the buffer
00204     ledtype = BASELEDTYPE;                     //set default ledtype, 11 =>WS2811, 12
      =>WS2812
00205
00206     //Set the LED output Port (Pin B0 is used for LED data output)
00207     DDRB = 0x01;
00208     PORTB = 0x00;
00209
00210     //Basic initializations
00211     ReceivedChar = 1;
00212     effectime = 10;
00213     effect=255;
00214     BufferCounter=0;
00215
00216     init_uart();                              //Init the hardware UART
00217     sei();                                     //enable global interrupts
00218
00219     //main system loop
00220     while(1){
00221         if (PacketComplete==1)     //new UART package containing color and effect data is
      available
00222         {
00223             //Prohibit the access to the UART RecBuffer while copying the data to a Temp Buffer
00224             IsReading=1;
00225             PaketStart=0;
00226             memcpy(TempBuffer,RecBuffer,DataLen);   //Copy the UART data to a temp array
00227             effect=TempBuffer[EFFECTINDEX] & 0x7F; //get the effect from the temp array
00228             effectime=TempBuffer[DELAYINDEX];        //get the delay time for the effect
      form the temp array
00229             ledtype=BASELEDTYPE+((TempBuffer[EFFECTINDEX] & 0x80)>>7);//
      configure the ledtype depending on the MSBit of the effect
00230             NumOfLeds=TempBuffer[NUMOFLEDINDEX];  //get the number of leds to control
00231             IsReading=0;                            //allow access to the UART RecBuffer
00232             memcpy(CompColorArray,&TempBuffer[5],DataLen-5);   //generate compressed
      color array
00233             if (lightdata!=NULL)
00234             {
00235                 free(lightdata);
00236             }
00237             lightdata = (uint8_t *) malloc (NumOfLeds*3);      //allocate the lightdata array for
      uncompressed colors
00238             PacketComplete=0;                    //reset PacketComplete flag
00239         }
00240         else
00241         {
00242             //main switch for effect handling
00243             switch(effect)
00244             {
00245                 case SETFULLCOLOR:
00246                     setfullcolor(colorconv8to24(
      CompColorArray[0]),lightdata);
00247                     transmit2leds(lightdata);
00248                     break;
00249                 case FILLUP:
00250                     fillup(colorconv8to24(CompColorArray[0]),
      colorconv8to24(CompColorArray[1]),lightdata);
00251                     transmit2leds(lightdata);
00252                     break;
00253                 case BLINK:
00254                     blinkled(colorconv8to24(CompColorArray[0]),
      lightdata);
00255                     break;
00256                 case RUNLED:
00257                     initrunled(colorconv8to24(
      CompColorArray[0]),lightdata,colorconv8to24(
      CompColorArray[1]));
00258                     effect++;
00259                 case 4:
00260                     runrunled(lightdata,1);
00261                     break;
00262                 case ALTERNATE:
00263                     init_alternating(colorconv8to24(
      CompColorArray[0]),colorconv8to24(CompColorArray[1]),lightdata);
00264                     effect++;
00265                 case 6:
```

```
00266                     run_alternating(lightdata);
00267                     break;
00268                 case RECOLOR:
00269                     recolor(colorconv8to24(CompColorArray[0]),lightdata)
    ;
00270                     effect=255;
00271                     break;
00272                 case FADEN:
00273                     faden(colorconv8to24(CompColorArray[0]),lightdata);
00274                     break;
00275                 case INITRAINBOW:
00276                     initrainbow(lightdata);
00277                     transmit2leds(lightdata);
00278                     break;
00279                 case ROTATE_R:
00280                     rotate(lightdata,0);
00281                     effectdelay(effectime);
00282                     transmit2leds(lightdata);
00283                     break;
00284                 case ROTATE_L:
00285                     rotate(lightdata,1);
00286                     effectdelay(effectime);
00287                     transmit2leds(lightdata);
00288                     break;
00289                 case CUSTOM:
00290                 //The custom effect assigns up to MAXNUMCOLORS=50 individual colors to the stribe
00291                 //if the number of colors is smaller than the number of LEDs the colors are repeated using
00292                 //modulo operation
00293                     for (i=0;i<NumOfLeds;i++)
00294                     {
00295                         j = i % (DataLen-5);
00296                         changeled(colorconv8to24(
    CompColorArray[j]),lightdata,i);
00297                     }
00298                     transmit2leds(lightdata);
00299                     effect=255;
00300                     break;
00301                 case EASTEREGG:
00302                     easteregg(lightdata);
00303                     break;
00304                 default:    //do nothing
00305                     break;
00306             }
00307         }
00308 
00309 
00310     }
00311 
00312 }
00313 
00314 
00316 ISR (USART_RX_vect)
00317 {
00318     ReceivedChar = UDR0;                            //Read data from the RX buffer
00319     if (ReceivedChar==PREAMBLE && IsReading==0)        //Store data in the
     RecBuffer array only if it is not accessed by the main function
00320     {
00321         PacketComplete=0;
00322         PaketStart=1;                              //Set packet start flag (-->254=PREAMBLE has
     been received)
00323         memset(RecBuffer,0,sizeof(RecBuffer[0])*
    UART_BUFFER_SIZE);//clear the buffer
00324         BufferCounter=0;
00325         RecBuffer[0]=ReceivedChar;                 //Store the preamble
00326     }
00327     else if (PaketStart==1)
00328     {
00329         //Store all Bytes after the preamble
00330         BufferCounter++;
00331         RecBuffer[BufferCounter]=ReceivedChar;
00332         DataLen=RecBuffer[LENINDEX];                //Store data len of the data
     packet (preamble included)
00333         if (DataLen==BufferCounter+1)
00334         {
00335             PacketComplete=1;                      //a whole packet has been received, update
     the effect in main
00336         }
00337     }
00338 }
```

Index