Lightstribe WS2811/WS2812

Generated by Doxygen 1.8.10

Fri May 6 2016 13:52:25

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 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)

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.

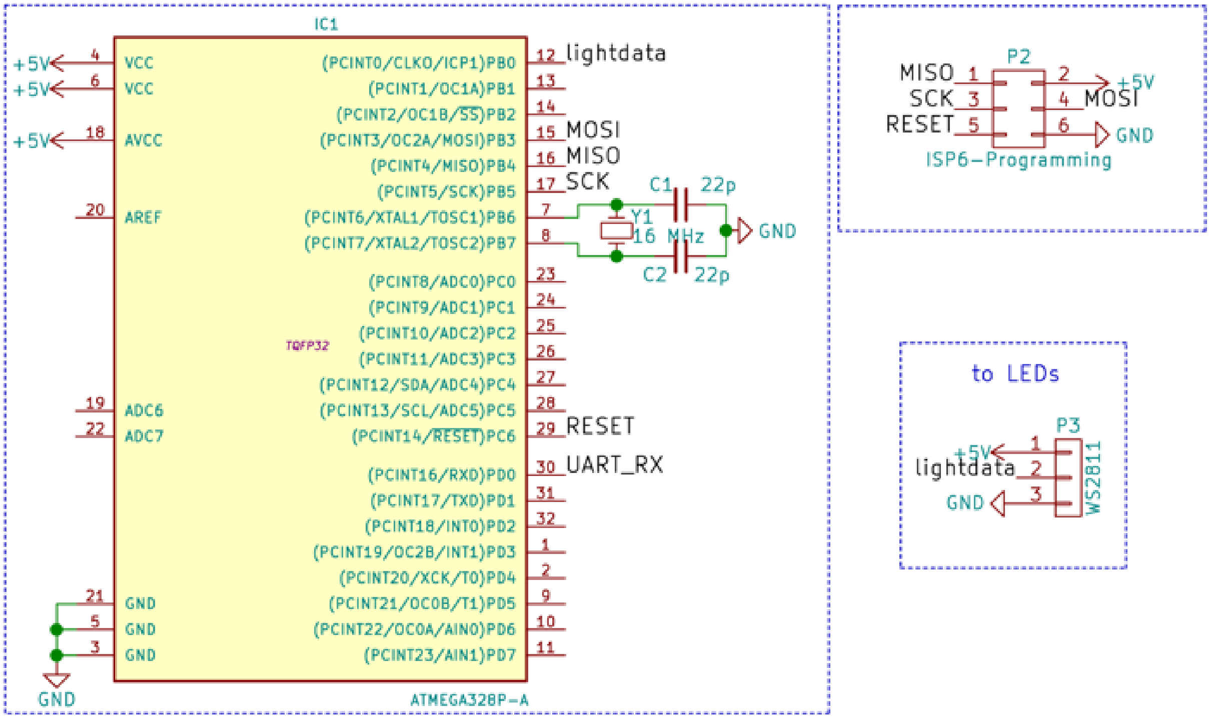

Figure 1: schematic of the AVR to controll WS2812/WS2811

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.

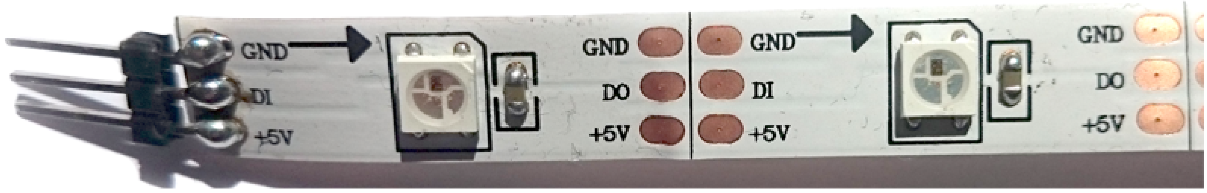

Figure 2: WS2812 stribe with pin header

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.

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.

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.

1.6 Implement further effects

1.7 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.

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

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

#ifndef GLOBALS_H_
#define GLOBALS_H_

//macros to display infos for CPU Frequency or other defines
#define _STR_EXPAND(tok) #tok
#define _STR(tok) _STR_EXPAND(tok)
#define _CPU_INFO(x) CPU_FREQUENCY##x

#ifndef EXTERN
#define EXTERN extern
#endif

EXTERN uint8_t NumOfLeds;
EXTERN uint16_t effectime;
EXTERN uint8_t ledtype;
#define BASELEDTYPE 11

#define MAXNUMCOLORS 50

#define UART_BUFFER_SIZE 80

EXTERN uint8_t CompColorArray[MAXNUMCOLORS];
EXTERN uint8_t RecBuffer[UART_BUFFER_SIZE];
EXTERN uint8_t BufferCounter;
EXTERN uint8_t DataLen;
EXTERN uint8_t effect;

//EXTERN uint8_t speed;

EXTERN uint8_t PacketComplete;
EXTERN uint8_t PaketStart;
EXTERN uint8_t IsReading;
EXTERN volatile char ReceivedChar;

#ifndef F_CPU
#define F_CPU 8000000
#endif
#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 YELLOW 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)
```

```
{
    struct color24bit color;
    color.blue = 0x00;
    color.green= 0x00;
    color.red = 0x00;
    setfullcolor(color, lightdata);
}

void rotate(uint8_t *lightdata, uint8_t direction)
{
    uint8_t temp1, temp2, temp3;
    uint8_t *tempp;
    uint16_t i;

    if (direction==0)
    {
        //Store overflowing LED
        temp1 = *lightdata;
        temp2= *(lightdata+1);
        temp3 =*(lightdata+2);
        //Rotate the array (minus 1 LED-->overflow; 1 LED correlate three 8 Bit color values)
        for (i=0;i<NumOfLeds*3-3;i++)
        {   //increase the array pointer step by step
            *lightdata = *(lightdata+3);
            lightdata++;
        }
        //assign overflowed LED
        *lightdata++ = temp1;
        *lightdata++ = temp2;
        *lightdata++ = temp3;

    }
    else
    {
        //Set a pointer to the end of the lightdata
        tempp = lightdata + NumOfLeds*3 -1;
        //Store overflowing LED
        temp1 = *tempp;
        temp2 = *(tempp-1);
        temp3 = *(tempp-2);

        //Rotate the array (minus 1 LED-->overflow; 1 LED correlate three 8 Bit color values)
        for (i=0;i<(NumOfLeds*3-3);i++)
        {   //decrease the array pointer step by step
            *tempp = *(tempp-3);
            tempp--;
        }
        //assign overflowed LED
        *tempp--=temp1;
        *tempp--=temp2;
        *tempp = temp3;
    }
}

void rotateN(uint8_t *lightdata, uint8_t direction, uint8_t width)
{
    uint8_t i;
    for (i=0;i<width;i++)
    {
        rotate(lightdata,direction);
    }
}

void initrunled(struct color24bit color, uint8_t *lightdata, struct color24bit background)
{
    setfullcolor(background,lightdata);
    changeled(color, lightdata,0);
}

void runrunled(uint8_t *lightdata, uint8_t direction)
{
    uint8_t i;

    //Run from left to right and back, one loop in this function, main while repeats the effect
    if (direction==2)
    {
        for (i=0;i<NumOfLeds;i++)
        {
            transmit2leds(lightdata);
            rotate(lightdata,1);
            effectdelay(effectime);
            if (PacketComplete==1)
                break;
        }
        for (i=0;i<NumOfLeds;i++)
        {

            rotate(lightdata,0);
            transmit2leds(lightdata);
            effectdelay(effectime);
            if (PacketComplete==1)
                break;
        }
    }
    else
    {   //Only one rotation is done, main while does the effect
        rotate(lightdata,direction);
        transmit2leds(lightdata);
        effectdelay(effectime);
    }
}

void blinkled(struct color24bit color, uint8_t *lightdata)
{
    //Set the chosen color
    setfullcolor(color, lightdata);
    transmit2leds(lightdata);
    effectdelay(effectime);
    //Turn the stribe off
    resetstribe(lightdata);
    transmit2leds(lightdata);
    effectdelay(effectime);
}

void init_alternating(struct color24bit color, struct color24bit backcolor, uint8_t *lightdata)
{
    uint16_t i;
    setfullcolor(backcolor, lightdata);     //Set background color
    for (i=0;i<NumOfLeds;i++)
    {
        if(i%2==0)
        {
            changeled(color,lightdata,i);  //set the even LEDs
        }
    }
}

void run_alternating(uint8_t *lightdata )
{
    transmit2leds(lightdata);
    effectdelay(effectime);
    rotate(lightdata,1);
}

void recolor(struct color24bit color, uint8_t *lightdata)
{
    uint8_t i;
    for (i=0;i<NumOfLeds;i++)
    {
        changeled(color,lightdata,i);
        transmit2leds(lightdata);
        effectdelay(effectime);
        if (PacketComplete==1)
            break;
    }
}

void faden(struct color24bit color, uint8_t *lightdata)
{
    uint8_t i;
    uint8_t maxgreen, maxred, maxblue;
    maxgreen =color.green;
    maxblue = color.blue;
    maxred = color.red;
    for (i=0;i<255;i++) //Fade down to LED off
    {
        setfullcolor(color,lightdata);
        transmit2leds(lightdata);
        effectdelay(effectime);
        //Decrease the color values that are greater than 0, stop if every value is 0
        if (color.green > 0)
        {
            --color.green;
        }
        if (color.blue > 0)
        {
            --color.blue;
        }
        if (color.red > 0)
        {
            --color.red;
        }
        if (color.red == 0 && color.blue == 0 && color.green == 0)
        {
            break;
        }
        if (PacketComplete==1)
        {
            break;
        }
    }

    for (i=0;i<255;i++) //Fade up to chosen color
    {
        setfullcolor(color,lightdata);
        transmit2leds(lightdata);
        effectdelay(effectime);
        //Increase the color values is they are lower than the chosen color value, stop if all maximums are reached
        if (color.green < maxgreen)
        {
            ++color.green;
        }
        if (color.blue < maxblue)
        {
            ++color.blue;
        }
        if (color.red < maxred)
        {
            ++color.red;
        }
        if (color.red == maxred && color.blue == maxblue && color.green == maxgreen)
        {
            break;
        }
        if (PacketComplete==1)
        {
            break;
        }
    }
}

void initrainbow(uint8_t *lightdata)
{
    uint8_t steps = NumOfLeds / 5;
    struct color24bit color;
    uint8_t i,j;
    //Start rainbow with red color
    color.red = 0xFF;
    color.blue= 0x00;
    color.green=0x00;
    j=0;
    for(i=0;i<NumOfLeds;i++)
    {
        if (j<steps)
        {
            color.blue = 0x00+0xFF/steps*j;     //increase blue to get violett
        }
        else if(j>steps && j<=2*steps)
        {
            color.red = 0xFF-0xFF/steps*(j/2);   //decrease red to get blue
        }
        else if(j>2*steps && j<=3*steps)
        {
            color.green = 0x00+0xFF/steps*(j/3);//increase green to get cyan
        }
        else if(j>3*steps && j<=4*steps)
        {
            color.blue = 0xFF-0xFF/steps*(j/4); //decrease blue to get green
        }
        else if(j>4*steps && j<=5*steps)
        {
            color.red = 0x00+0xFF/steps*(j/5);   //increase red to get yellow
        }
        else if(j>6*steps)
        {
            color.green = 0xFF-0xFF/steps*(j/6);//decrease green to get red
        }
        j++;
        changeled(color,lightdata,i);
    }
}

void eastereggbase(struct color24bit color, uint8_t *lightdata)
{
    uint8_t i,j;
    uint8_t n;
    j=NumOfLeds;
    for (i=0;i<NumOfLeds;i++)
    {
        n=(j-i);
        changeled(color,lightdata,0);
        while(n-->0)
        {
            rotate(lightdata,1);
            transmit2leds(lightdata);
            effectdelay(effectime);
        }
        if (PacketComplete==1)
        break;
    }
}

void easteregg(uint8_t *lightdata)
{
    struct color24bit color, color2;
    uint8_t i;
    color=colorconv8to24(252);
    color2=colorconv8to24(201);
    eastereggbase(color2,lightdata);
    for (i=0;i<100;i++)
    {
        if (PacketComplete==1)
        break;
        _delay_ms(50);
    }
    eastereggbase(color,lightdata);
    for (i=0;i<100;i++)
    {
        if (PacketComplete==1)
        break;
        _delay_ms(50);
    }
}

void fillup(struct color24bit color,struct color24bit backcolor, uint8_t * lightdata)
{
    uint8_t i,j;
    for (i=0;i<NumOfLeds;i++)
    {
        for (j=0;j<NumOfLeds-i;j++)
        {
            changeled(color,lightdata,j);          //running LED, foreground
            if (j>0)
            {
                changeled(backcolor,lightdata,j-1);    //background LEDs
            }
            transmit2leds(lightdata);
            effectdelay(effectime);
        }
        if (PacketComplete==1)
            break;
        effectdelay(effectime);
    }
}
```

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

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

#ifndef LEDEFFECTS_H_
#define LEDEFFECTS_H_

//EFFECTS
#define SETFULLCOLOR 0

#define FILLUP 1

```

```
#define BLINK 2

#define RUNLED 3

#define ALTERNATE 5

#define RECOLOR 7

#define FADE 8

#define INITRAINBOW 9

#define ROTATE_R 10

#define ROTATE_L 11

#define CUSTOM 12

#define EASTEREGG 13

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
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
void effectdelay(uint16_t delay);
//a simple variable delay function
void setfullcolor(struct color24bit color, uint8_t *lightdata);
//set the whole stribe in one color, call transmit2leds afterwards
void resetstribe(uint8_t *lightdata);
//set the whole stribe off, call transmit2leds afterwards
void rotate(uint8_t *lightdata, uint8_t direction);                                              //
rotate the color array by one position
void rotateN(uint8_t *lightdata, uint8_t direction, uint8_t width);                              //
rotate the color array by n positions
void initrunled(struct color24bit color, uint8_t *lightdata, struct
color24bit background);      //initialize the runled effect, call runrunled afterwards
void runrunled(uint8_t *lightdata, uint8_t direction);                                         //
runs the runled effect, call initrunled before
void blinkled(struct color24bit color, uint8_t *lightdata);
//generate a blinking effect
void init_alternating(struct color24bit color, struct
color24bit backcolor, uint8_t *lightdata);//initialize the alternating effect, 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 stribe step by step, stand alone function, ends after execution
void faden(struct color24bit color, uint8_t *lightdata);
//color fading effect, stand alone effect
void initrainbow(uint8_t *lightdata);
//init the stribe with rainbow colors, call transmit2leds afterwards
void eastereggbase(struct color24bit color, uint8_t *lightdata);
//part of the easteregg effect, do not call directly
void easteregg(uint8_t *lightdata);
//easteregg effect, try out and have fun :-)
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

#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

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

void changeled(struct color24bit color, uint8_t *lightdata, uint8_t lednr)
{
    if (lednr>NumOfLeds)
        return;
    lightdata=lightdata+lednr*3;
    *lightdata++=color.green;
    *lightdata++=color.red;
    *lightdata++=color.blue;
}
```

```

void setled(struct color24bit color, uint8_t *lightdata, uint8_t lednr)
{
    uint8_t ledcolor;
    uint16_t i;
    if (lednr>NumOfLeds)
        return;
    //Loop over the whole color array (-->NumOfLeds*3)
    for (i=0;i<NumOfLeds*3;i++)
    {
        if (i==(lednr*3) || i==(lednr*3+1) || i==(lednr*3+2))
        {   //position of the LED to set
            ledcolor = i%3;
            if (ledcolor==0)
                *lightdata++=color.green;
            else if(ledcolor==1)
                *lightdata++=color.red;
            else
                *lightdata++=color.blue;
        }
        else
        {   //all others off (0x00-->black)
            ledcolor = i%3;
            if (ledcolor==0)
                *lightdata++=0x00;
            else if(ledcolor==1)
                *lightdata++=0x00;
            else
                *lightdata++=0x00;
        }
    }
}

void transmit2leds(uint8_t lightdata[])
{
    uint16_t i ;
    uint8_t byte2send ;
    for(i=0;i<NumOfLeds*3;i++)
    {
        byte2send = lightdata[i];
        //Transmit each Bit of one Byte using the One Wire Protocoll
        if ((byte2send & 128)==0)
        {
            SETLOW
        }
        else
        {
            SETHIGH
        }
        if ((byte2send & 64)==0)
        {
            SETLOW
        }
        else
        {
            SETHIGH
        }
        if ((byte2send & 32)==0)
        {
            SETLOW
        }
        else
        {
            SETHIGH
        }
        if ((byte2send & 16)==0)
        {
            SETLOW
        }
        else
        {
            SETHIGH
        }
        if ((byte2send & 8)==0)
        {
            SETLOW
        }
        else
        {
            SETHIGH
        }
        if ((byte2send & 4)==0)
        {
            SETLOW
        }
        else
        {
            SETHIGH
        }
        if ((byte2send & 2)==0)
        {
            SETLOW
        }
        else
        {
            SETHIGH
        }
        if ((byte2send & 1)==0)
        {
            SETLOW
        }
        else
        {
            SETHIGH
        }
    }
    _delay_us(55);      //defined delay after the transmission is complete (Datasheet says >=50us)
}
```

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

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

#ifndef LIGHTSTRIBE_H_
#define LIGHTSTRIBE_H_

struct color24bit{
    uint8_t red;
    uint8_t green;
    uint8_t blue;
};

#if F_CPU == 16000000
#pragma message("Use 16 MHz Macros")

#define SETHIGH PORTB=0x01;\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               PORTB=0x00;\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");

#elif F_CPU == 8000000
#pragma message("Use 8 MHz Macros")

#define SETHIGH PORTB=0x01;\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");\
               PORTB=0x00;\
               asm ("nop");\
               asm ("nop");
#endif



#if F_CPU == 16000000

#define SETLOW PORTB=0x01;\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");\
              PORTB=0x00;\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");\
              asm ("nop");
#elif F_CPU == 8000000

```

```
#define SETLOW PORTB=0x01;\
               asm ("nop");\
               asm ("nop");\
               PORTB=0x00;\
               asm ("nop");\
               asm ("nop");\
               asm ("nop");
#endif


//function to change one LED at a specific position; all other LEDs are not changed; run transmit2leds afterwards
void changeled(struct color24bit color, uint8_t *lightdata, uint8_t lednr);
//function to set one LED at a specific position; all other LEDs are turned off; run transmit2leds afterwards
void setled(struct color24bit color, uint8_t *lightdata, uint8_t lednr);
//transmit the color array to the stribe --> one wire data transmission
void transmit2leds(uint8_t lightdata[]);

#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, 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 229 of file ws2811lichterkette.c.

References MYUBRR.

Referenced by main().

5.12 ws2811lichterkette.c

```

/*****************************************************************************/
//define global variables
#define EXTERN
#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"

//UART basic definitions
#define BAUD 38400

#define MYUBRR F_CPU/16/BAUD -1

#define BAUD_REAL (F_CPU/(16*(MYUBRR+1)))       /*real baudrate in this configuration*/
#define BAUD_ERROR ((BAUD_REAL*1000)/BAUD)      /*calculate baudrate error*/
#if ((BAUD_ERROR<990) || (BAUD_ERROR>1010))
    #error baudrate error greater 1% !          /*show an error message if the baudrate error is greater than 1%*/
#endif

//Protocol definition for UART communication
//The protocol is defined as:
//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

#define PREAMBLE 254

#define LENINDEX 1

#define EFFECTINDEX 2

#define DELAYINDEX 3

#define NUMOFLEDINDEX 4


//compiling info output
#pragma message("MYUBRR: " _STR(MYUBRR))
#pragma message("CPU Frequency: " _STR(F_CPU) "Hz")
#pragma message("Baudrate: " _STR(BAUD))
#pragma message("Configuration: MAXNUMCOLORS=" _STR(MAXNUMCOLORS)  " | UART_BUFFER_SIZE=" _STR(UART_BUFFER_SIZE) " | PREAMBLE=" _STR(PREAMBLE))
```

```

void init_uart(void)
{
    DDRD |= _BV(PD1);
    DDRD &= ~_BV(PD0);

     //Set BAUD
     UBRR0H = ((MYUBRR) >> 8);
     UBRR0L = MYUBRR;

     UCSR0B |= (1 << RXEN0) ;//| (1 << TXEN0);      // Enable receiver (and transmitter; commited out)
     UCSR0B |= (1 << RXCIE0);                       // Enable the receiver interrupt
     UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);       // 8 data Bit, one stop Bit
}

int main(void)
{
    uint16_t i,j;                            //helper variables (counters)
    uint8_t TempBuffer[UART_BUFFER_SIZE];    //Temp. buffer for copy of the UART data to achieve data consistency
    uint8_t *lightdata;                      //lightdata pointer for lightdata array; the array size is dynamic to controll different numbers of LEDs

    NumOfLeds=50;                           //default number of LEDs is 50 => one stribe
    //Flag initializations
    PacketComplete=0;
    IsReading=0;
    PaketStart=0;
    BufferCounter = 0;

    memset(RecBuffer,0,sizeof(RecBuffer[0])*UART_BUFFER_SIZE);    //clear the buffer
    memset(TempBuffer,0,sizeof(RecBuffer[0])*UART_BUFFER_SIZE);    //clear the buffer
    ledtype = BASELEDTYPE;                     //set default ledtype, 11 =>WS2811, 12 =>WS2812

    //Set the LED output Port (Pin B0 is used for LED data output)
    DDRB = 0x01;
    PORTB = 0x00;

    //Basic initializations
    ReceivedChar = 1;
    effectime = 10;
    effect=255;
    BufferCounter=0;

    init_uart();                            //Init the hardware UART
    sei();                                   //enable global interrupts

    //main system loop
    while(1){
        if (PacketComplete==1)    //new UART package containing color and effect data is available
        {
            //Prohibit the access to the UART RecBuffer while copying the data to a Temp Buffer
            IsReading=1;
            PaketStart=0;
            memcpy(TempBuffer,RecBuffer,DataLen);   //Copy the UART data to a temp array
            effect=TempBuffer[EFFECTINDEX] & 0x7F; //get the effect from the temp array
            effectime=TempBuffer[DELAYINDEX];        //get the delay time for the effect form the temp array
            ledtype=BASELEDTYPE+((TempBuffer[EFFECTINDEX] & 0x80)>>7);//configure the ledtype depending on the MSBit of the effect
            NumOfLeds=TempBuffer[NUMOFLEDINDEX];  //get the number of leds to control
            IsReading=0;                           //allow access to the UART RecBuffer
            memcpy(CompColorArray,&TempBuffer[5],DataLen-5);   //generate compressed color array
            if (lightdata!=NULL)
            {
                free(lightdata);
            }
            lightdata = (uint8_t *) malloc (NumOfLeds*3);      //allocate the lightdata array for uncompressed colors
            PacketComplete=0;                      //reset PacketComplete flag
        }
        else
        {
            //main switch for effect handling
            switch(effect)
            {
                case SETFULLCOLOR:
                    setfullcolor(colorconv8to24(CompColorArray[0]),lightdata);
                    transmit2leds(lightdata);
                    break;
                case FILLUP:
                    fillup(colorconv8to24(CompColorArray[0]),
colorconv8to24(CompColorArray[1]),lightdata);
                    transmit2leds(lightdata);
                    break;
                case BLINK:
                    blinkled(colorconv8to24(CompColorArray[0]),
lightdata);
                    break;
                case RUNLED:
                    initrunled(colorconv8to24(
CompColorArray[0]),lightdata,colorconv8to24(
CompColorArray[1]));
                    effect++;
                case 4:
                    runrunled(lightdata,1);
                    break;
                case ALTERNATE:
                    init_alternating(colorconv8to24(
CompColorArray[0]),colorconv8to24(CompColorArray[1]),lightdata);
                    effect++;
                case 6:
                    run_alternating(lightdata);
                    break;
                case RECOLOR:
                    recolor(colorconv8to24(CompColorArray[0]),lightdata)
;
                    effect=255;
                    break;
                case FADE:
                    faden(colorconv8to24(CompColorArray[0]),lightdata);
                    break;
                case INITRAINBOW:
                    initrainbow(lightdata);
                    transmit2leds(lightdata);
                    break;
                case ROTATE_R:
                    rotate(lightdata,0);
                    effectdelay(effectime);
                    transmit2leds(lightdata);
                    break;
                case ROTATE_L:
                    rotate(lightdata,1);
                    effectdelay(effectime);
                    transmit2leds(lightdata);
                    break;
                case CUSTOM:
                //The custom effect assigns up to MAXNUMCOLORS individual colors to the stribe
                //if the number of colors is smaller than the number of LEDs the colors are repeated using
                //modulo operation
                    for (i=0;i<NumOfLeds;i++)
                    {
                        j = i % (DataLen-5);
                        changeled(colorconv8to24(
CompColorArray[j]),lightdata,i);
                    }
                    transmit2leds(lightdata);
                    effect=255;
                    break;
                case EASTEREGG:
                    easteregg(lightdata);
                    break;
                default:    //do nothing
                    break;
            }
        }


    }

}


ISR (USART_RX_vect)
{
    ReceivedChar = UDR0;                            //Read data from the RX buffer
    if (ReceivedChar==PREAMBLE && IsReading==0)        //Store data in the
 RecBuffer array only if it is not accessed by the main function
    {
        PacketComplete=0;
        PaketStart=1;                             //Set packet start flag (-->254=PREAMBLE has
 been received)
        memset(RecBuffer,0,sizeof(RecBuffer[0])*
UART_BUFFER_SIZE);//clear the buffer
        BufferCounter=0;
        RecBuffer[0]=ReceivedChar;                 //Store the preamble
    }
    else if (PaketStart==1)
    {
        //Store all Bytes after the preamble
        BufferCounter++;
        RecBuffer[BufferCounter]=ReceivedChar;
        DataLen=RecBuffer[LENINDEX];                //Store data len of the data packet (preamble included)
        if (DataLen==BufferCounter+1)
        {
            PacketComplete=1;                    //a whole packet has been received, update the effect in main
        }
    }
}
```

Index