Sniffer RS-232

1.0

Generated by Doxygen 1.9.5

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Module Documentation

4.1 BSP button

Module of BSP button.

Data Structures

- struct button_init_ctx

  Initializing context of BSP button.

Macros

- #define **BUTTON_TIM_FREQ** (10000)

  Frequency of htim.
- #define TIM_TICK_TO_MS(X) ((1000 ∗ (X)) / BUTTON_TIM_FREQ)
- #define TIM_PERIOD_CALC(X) ((BUTTON_TIM_FREQ ∗ (X)) / 1000)

Enumerations

- enum button_action { BUTTON_NONE = 0 , BUTTON_PRESSED , BUTTON_LONG_PRESSED , BUTTON_ACTION_MAX }

  BSP button actions.

Functions

- uint8_t bsp_button_init (struct button_init_ctx ∗init_ctx)
- uint8_t bsp_button_deinit (void)
- static void __button_tim_msp_init (TIM_HandleTypeDef ∗htim)
- static void __button_tim_msp_deinit (TIM_HandleTypeDef ∗htim)
- static void __button_tim_period_elapsed_callback (TIM_HandleTypeDef ∗htim)
- static bool __button_tim_is_started (void)
- static uint8_t __button_tim_stop (void)
- static uint8_t __button_tim_start (uint32_t period_ms)
- void EXTI4_IRQHandler (void)
- void TIM7_IRQHandler (void)

Variables

- static EXTI_HandleTypeDef **hexti** = {.Line = EXTI_LINE_4}

  STM32 HAL EXTI instance, used to detect pushing and releasing actions on the button.
- static TIM_HandleTypeDef **htim** = {.Instance = TIM7}

  STM32 HAL TIM instance, used to detect long pressing and filter contact bounce.
- static struct button_init_ctx **ctx** = {0}

  BSP button context.
- static bool **button_pressed** = false

  Current state of the button: true - pressed, false - not.
- static bool **is_long_action** = false

  Flag whether button timer is checking of long press action on the button.

4.1.1 Detailed Description

Module of BSP button.

4.1.2 Macro Definition Documentation

4.1.2.1 TIM_PERIOD_CALC

```
#define TIM_PERIOD_CALC(
              X ) ((BUTTON_TIM_FREQ * (X)) / 1000)
```

MACRO TIM period register calculation

Parameters

in	*X*	desired TIM period in milliseconds

Returns

value of TIM period register

4.1.2.2 TIM_TICK_TO_MS

```
#define TIM_TICK_TO_MS(
              X ) ((1000 * (X)) / BUTTON_TIM_FREQ)
```

MACRO Conversion of TIM tick counter to milliseconds

Parameters

in	*X*	TIM tick counter

Returns

milliseconds elapsed from the start of the timer

4.1.3 Enumeration Type Documentation

4.1.3.1 button_action

```
enum button_action
```

BSP button actions.

Enumerator

BUTTON_NONE	None of button actions
BUTTON_PRESSED	Button is pressed for a short time not less than button_init_ctx::press_min_dur_ms but less than button_init_ctx::long_press_dur_ms
BUTTON_LONG_PRESSED	Button is pressed for a long time not less than button_init_ctx::long_press_dur_ms
BUTTON_ACTION_MAX	Count of types of button actions

4.1.4 Function Documentation

4.1.4.1 __button_tim_is_started()

```
static bool __button_tim_is_started (
             void  )  [static]
```

Flag whether button timer is started

Returns

true if the timer is started, false otherwise

4.1.4.2 __button_tim_msp_deinit()

```
static void __button_tim_msp_deinit (
             TIM_HandleTypeDef * htim )  [static]
```

STM32 HAL TIM MSP deinitialization

The function executes clock, NVIC deinitialization

Parameters

in	*htim*	STM32 HAL TIM instance, should equal to htim

4.1.4.3 __button_tim_msp_init()

```
static void __button_tim_msp_init (
             TIM_HandleTypeDef * htim )  [static]
```

STM32 HAL TIM MSP initialization

The function executes clock, NVIC initialization

Parameters

in	*htim*	STM32 HAL TIM instance, should equal to htim

4.1.4.4 __button_tim_period_elapsed_callback()

```
static void __button_tim_period_elapsed_callback (
             TIM_HandleTypeDef * htim )  [static]
```

Callback for button timer period elapsion

The function is designed to have two modes:

1. Timeout as protection against contact bounce is expired (is_long_action = false).
   After that next button action can be caught
2. Minimum duration to consider that the button is pressed for a long time (is_long_action = true)
   User callback button_init_ctx::button_isr_cb with parameter BUTTON_LONG_PRESSED is called

Parameters

in	*htim*	STM32 HAL TIM instance, should equal to htim

4.1.4.5 __button_tim_start()

```
static uint8_t __button_tim_start (
             uint32_t period_ms )  [static]
```

Start button timer

Parameters

in	*period_ms*	of the timer in ms

Returns

RES_OK on success error otherwise

4.1.4.6 __button_tim_stop()

```
static uint8_t __button_tim_stop (
             void ) [static]
```

Stop button timer

Returns

RES_OK on success error otherwise

4.1.4.7 bsp_button_deinit()

```
uint8_t bsp_button_deinit (
             void )
```

BSP button deinitialization

Returns

RES_OK on success error otherwise

4.1.4.8 bsp_button_init()

```
uint8_t bsp_button_init (
             struct button_init_ctx * init_ctx )
```

BSP button initialization

Parameters

in	*init_ctx*	initializing context of BSP button

Returns

RES_OK on success error otherwise

4.1.4.9 EXTI4_IRQHandler()

```
void EXTI4_IRQHandler (
             void  )
```

NVIC EXTI4 IRQ handler

The handler processes pushing/releasing actions on the button,
start button timer with appropriate period and etc.

4.1.4.10 TIM7_IRQHandler()

```
void TIM7_IRQHandler (
             void  )
```

NVIC IRQ TIM7 handler

4.2 BSP CRC

Module of BSP CRC.

Functions

- uint8_t bsp_crc_init (void)
- uint8_t bsp_crc_deinit (void)
- uint8_t bsp_crc_calc (uint8_t ∗data, uint32_t len, uint32_t ∗result)
- void HAL_CRC_MspInit (CRC_HandleTypeDef ∗hcrc)
- void HAL_CRC_MspDeInit (CRC_HandleTypeDef ∗hcrc)

Variables

- static CRC_HandleTypeDef **crc_module** = {.Instance = CRC}

  STM32 HAL CRC instance.

4.2.1 Detailed Description

Module of BSP CRC.

4.2.2 Function Documentation

4.2.2.1 bsp_crc_calc()

```
uint8_t bsp_crc_calc (
             uint8_t ∗ data,
             uint32_t len,
             uint32_t ∗ result )
```

BSP CRC calculation

Parameters

in	*data*	data over which CRC is calculated
in	*len*	size of data within which CRC is calculated
out	*result*	calculated CRC value

Returns

RES_OK on success error otherwise

4.2.2.2 bsp_crc_deinit()

```
uint8_t bsp_crc_deinit (
             void  )
```

BSP CRC deinitialization

Returns

RES_OK on success error otherwise

4.2.2.3 bsp_crc_init()

```
uint8_t bsp_crc_init (
             void  )
```

BSP CRC initialization

Returns

RES_OK on success error otherwise

4.2.2.4 HAL_CRC_MspDeInit()

```
void HAL_CRC_MspDeInit (
             CRC_HandleTypeDef * hcrc )
```

STM32 HAL MSP CRC deinitialization

Parameters

in	*hcrc*	STM32 HAL CRC instance, should equal to crc_module

4.2.2.5 HAL_CRC_MspInit()

```
void HAL_CRC_MspInit (
             CRC_HandleTypeDef * hcrc )
```

STM32 HAL MSP CRC initialization

Parameters

in	*hcrc*	STM32 HAL CRC instance, should equal to crc_module

4.3 BSP GPIO

Module of BSP GPIO.

Macros

- #define BSP_GPIO_PORT_READ(GPIOX, GPIO_PIN) (!!(GPIOX->IDR & GPIO_PIN))
- #define BSP_GPIO_PORT_WRITE(GPIOX, GPIO_PIN, LEVEL) (GPIOX->BSRR = LEVEL ? GPIO_PIN : ((uint32_t)GPIO_PIN << 16U))
- #define BSP_GPIO_FORCE_OUTPUT_MODE(GPIOX, GPIO_NUM)

Functions

- uint8_t bsp_gpio_bulk_read (GPIO_TypeDef *gpiox, const uint16_t *gpio_pins, uint16_t *gpio_states)
- uint8_t bsp_gpio_bulk_write (GPIO_TypeDef *gpiox, const uint16_t *gpio_pins, const uint16_t gpio_states)

4.3.1 Detailed Description

Module of BSP GPIO.

4.3.2 Macro Definition Documentation

4.3.2.1 BSP_GPIO_FORCE_OUTPUT_MODE

```
#define BSP_GPIO_FORCE_OUTPUT_MODE(
              GPIOX,
              GPIO_NUM )
```

Value:

```
do {\
    GPIOX->MODER = (GPIOX->MODER | (1 « (2 * GPIO_NUM))) & (~(1 « (2 * GPIO_NUM + 1)));\
} while(0)
```

MACRO Set output mode to a pin

The macro is used for fast setting of output mode of a pin

Parameters

in	*GPIOX*	port of a pin
in	*GPIO_NUM*	number of GPIO pin (0..15)

4.3.2.2 BSP_GPIO_PORT_READ

```
#define BSP_GPIO_PORT_READ(
              GPIOX,
              GPIO_PIN ) (!!(GPIOX->IDR & GPIO_PIN))
```

MACRO GPIO level get

The macro is used for fast reading of a level on a pin

Parameters

in	*GPIOX*	port of read pin
in	*GPIO_PIN*	read pin

Returns

read level on the pin

4.3.2.3 BSP_GPIO_PORT_WRITE

```
#define BSP_GPIO_PORT_WRITE(
              GPIOX,
              GPIO_PIN,
              LEVEL ) (GPIOX->BSRR = LEVEL ? GPIO_PIN : ((uint32_t)GPIO_PIN << 16U))
```

MACRO GPIO level set

The macro is used for fast setting of a level to a pin

Parameters

in	*GPIOX*	port of set pin
in	*GPIO_PIN*	set pin
in	*LEVEL*	set level, true if active one false otherwise

4.3.3 Function Documentation

4.3.3.1 bsp_gpio_bulk_read()

```
uint8_t bsp_gpio_bulk_read (
            GPIO_TypeDef * gpiox,
            const uint16_t * gpio_pins,
            uint16_t * gpio_states )
```

GPIO bulk reading

Parameters

in	*gpiox*	port of read pins
in	*gpio_pins*	GPIO pins levels on which should be read
out	*gpio_states*	array of levels read on `gpio_pins`

Returns

RES_OK on success error otherwise

4.3.3.2 bsp_gpio_bulk_write()

```
uint8_t bsp_gpio_bulk_write (
            GPIO_TypeDef * gpiox,
            const uint16_t * gpio_pins,
            const uint16_t gpio_states )
```

GPIO bulk writing

Parameters

in	*gpiox*	port of written pins
in	*gpio_pins*	GPIO pins levels on which should be set
in	*gpio_states*	array of levels set on `gpio_pins`

Returns

RES_OK on success error otherwise

4.4 BSP LCD1602

Module of BSP `LCD1602`

Data Structures

- struct lcd1602_settings

  Settings of BSP LCD1602.

Macros

- #define **MAX_CGRAM_ADDRESS** 0x3F

  Maximum address of CGRAM memory.
- #define **MAX_DDRAM_ADDRESS** 0x7F

  Maximum address of DDRAM memory.
- #define **LCD1602_LENGTH_LINE** 16

  Length of the line of LCD1602 in symbols.
- #define **LCD1602_MAX_STR_LEN** (4 $*$ LCD1602_LENGTH_LINE)

  Maximum length of buffered string used within the module.
- #define **LCD1602_DDRAM_START_LINE1** 0x00

  DDRAM address of start of first line.
- #define **LCD1602_DDRAM_END_LINE1** 0x27

  DDRAM address of end of first line (display is used in 2-line mode)
- #define **LCD1602_DDRAM_START_LINE2** 0x40

  DDRAM address of start of second line.
- #define **LCD1602_DDRAM_END_LINE2** 0x67

  DDRAM address of end of second line.
- #define **LCD1602_INSTR_REG** 0x0

  Level on signal RS to choose instruction register.
- #define **LCD1602_DATA_REG** 0x1

  Level on signal RS to choose data register.
- #define **LCD1602_READ_MODE** 0x1

  Level on signal R/W to set read mode.
- #define **LCD1602_WRITE_MODE** 0x0

  Level on signal R/W to set write mode.
- #define **TIME_FOR_DELAY** 1

  Time delay in us while waiting for BUSY flag, used in __lcd1602_wait.
- #define **WAIT_TMT** 500

  Timeout in ms for waiting for BUSY flag.
- #define TYPE_SHIFT_IS_VALID(X) (((uint8_t)(X)) $<$ LCD1602_SHIFT_MAX)
- #define NUM_LINE_IS_VALID(X) (((uint8_t)(X)) $<$ LCD1602_NUM_LINE_MAX)
- #define FONT_SIZE_IS_VALID(X) (((uint8_t)(X)) $<$ LCD1602_FONT_SIZE_MAX)
- #define TYPE_MOVE_CURSOR_IS_VALID(X) (((uint8_t)(X)) $<$ LCD1602_CURSOR_MOVE_MAX)
- #define SHIFT_ENTIRE_IS_VALID(X) (((uint8_t)(X)) $<$ LCD1602_SHIFT_ENTIRE_MAX)
- #define TYPE_INTERFACE_IS_VALID(X) (((uint8_t)(X)) $<$ LCD1602_INTERFACE_MAX)
- #define DISP_STATE_IS_VALID(X) (((uint8_t)(X)) $<$ LCD1602_DISPLAY_MAX)
- #define CURSOR_STATE_IS_VALID(X) (((uint8_t)(X)) $<$ LCD1602_CURSOR_MAX)
- #define CURSOR_BLINK_STATE_IS_VALID(X) (((uint8_t)(X)) $<$ LCD1602_CURSOR_BLINK_MAX)
- #define LCD1602_DATA_PINS

  All mixed GPIO pins from lcd1602_data_pins, used for (de-)initalizating purposes.

Enumerations

- enum lcd1602_type_shift {
  LCD1602_SHIFT_CURSOR_UNDEF = -1 , LCD1602_SHIFT_CURSOR_LEFT , LCD1602_SHIFT_CURSOR_RIGHT
  , LCD1602_SHIFT_DISPLAY_LEFT ,
  LCD1602_SHIFT_DISPLAY_RIGHT , LCD1602_SHIFT_MAX }

  Type of cursor/display shift.
- enum lcd1602_num_line { LCD1602_NUM_LINE_UNDEF = -1 , LCD1602_NUM_LINE_1 , LCD1602_NUM_LINE_2
  , LCD1602_NUM_LINE_MAX }

Numbrt line of LCD1602.

- enum lcd1602_font_size { LCD1602_FONT_SIZE_UNDEF = -1 , LCD1602_FONT_SIZE_5X8 , LCD1602_FONT_SIZE_5X11 , LCD1602_FONT_SIZE_MAX }

  Types of font size.
- enum lcd1602_type_move_cursor { LCD1602_CURSOR_MOVE_UNDEF = -1 , LCD1602_CURSOR_MOVE_LEFT , LCD1602_CURSOR_MOVE_RIGHT , LCD1602_CURSOR_MOVE_MAX }

  Move types of cursor.
- enum lcd1602_shift_entire_disp { LCD1602_SHIFT_ENTIRE_UNDEF = -1 , LCD1602_SHIFT_ENTIRE_PERFORMED , LCD1602_SHIFT_ENTIRE_NOT_PERFORMED , LCD1602_SHIFT_ENTIRE_MAX }

  Shift types of entire display.
- enum lcd1602_type_interface { LCD1602_INTERFACE_UNDEF = -1 , LCD1602_INTERFACE_4BITS , LCD1602_INTERFACE_8BITS , LCD1602_INTERFACE_MAX }

  Type of LCD1602 interfaces.
- enum lcd1602_disp_state { LCD1602_DISPLAY_UNDEF = -1 , LCD1602_DISPLAY_OFF , LCD1602_DISPLAY_ON , LCD1602_DISPLAY_MAX }

  Display states.
- enum lcd1602_cursor_state { LCD1602_CURSOR_UNDEF = -1 , LCD1602_CURSOR_OFF , LCD1602_CURSOR_ON , LCD1602_CURSOR_MAX }

  Cursor states.
- enum lcd1602_cursor_blink_state { LCD1602_CURSOR_BLINK_UNDEF = -1 , LCD1602_CURSOR_BLINK_OFF , LCD1602_CURSOR_BLINK_ON , LCD1602_CURSOR_BLINK_MAX }

  Cursor blink states.

Functions

- uint8_t bsp_lcd1602_init (struct lcd1602_settings ∗init_settings)
- uint8_t bsp_lcd1602_deinit (void)
- uint8_t bsp_lcd1602_printf (const char ∗line1, const char ∗line2,...)
- uint8_t bsp_lcd1602_cprintf (const char ∗line1, const char ∗line2,...)
- uint8_t bsp_lcd1602_ddram_address_set (const uint8_t address)
- uint8_t bsp_lcd1602_cgram_address_set (const uint8_t address)
- uint8_t bsp_lcd1602_function_set (const enum lcd1602_type_interface interface, const enum lcd1602_num_line num_line, const enum lcd1602_font_size font_size)
- uint8_t bsp_lcd1602_cursor_disp_shift (const enum lcd1602_type_shift shift)
- uint8_t bsp_lcd1602_display_on_off (const enum lcd1602_disp_state disp_state, const enum lcd1602_cursor_state cursor_state, const enum lcd1602_cursor_blink_state cursor_blink_state)
- uint8_t bsp_lcd1602_entry_mode_set (const enum lcd1602_type_move_cursor cursor, const enum lcd1602_shift_entire_disp shift_entire)
- uint8_t bsp_lcd1602_return_home (void)
- uint8_t bsp_lcd1602_display_clear (void)
- static uint8_t __lcd1602_read_write (uint8_t ∗data, uint8_t type_reg, uint8_t type_mode)
- static uint8_t __lcd1602_instruction_write (uint8_t instruction)
- static uint8_t __lcd1602_read_busy_flag (uint8_t ∗busy_flag, uint8_t ∗address_counter)
- static uint8_t __lcd1602_data_write (uint8_t data)
- static uint8_t __lcd1602_wait (const uint32_t timeout)
- static uint8_t __lcd1602_printf (const char ∗line1, const char ∗line2, bool is_centered, va_list argp)

Variables

- static const uint16_t lcd1602_data_pins [ ]

  Array of GPIO pins used for 8-bit parallel interface.
- static struct lcd1602_settings **settings**

  Local copy of display settings.

4.4.1 Detailed Description

Module of BSP `LCD1602`

```
The module does communication with display LCD1602 via 8-bit parallel interface using GPIO
```

4.4.2 Macro Definition Documentation

4.4.2.1 CURSOR_BLINK_STATE_IS_VALID

```
#define CURSOR_BLINK_STATE_IS_VALID(
              X ) (((uint8_t)(X)) < LCD1602_CURSOR_BLINK_MAX)
```

MACRO Cursor blink state is valid

The macro decides whether `X` is valid cursor blink state

Parameters

in	*X*	cursor blink state

Returns

true if `X` is valid cursor blink state false otherwise

4.4.2.2 CURSOR_STATE_IS_VALID

```
#define CURSOR_STATE_IS_VALID(
              X ) (((uint8_t)(X)) < LCD1602_CURSOR_MAX)
```

MACRO Cursor state is valid

The macro decides whether `X` is valid cursor state

Parameters

in	*X*	cursor state

Returns

true if `X` is valid cursor state false otherwise

4.4.2.3 DISP_STATE_IS_VALID

```
#define DISP_STATE_IS_VALID(
              X ) (((uint8_t)(X)) < LCD1602_DISPLAY_MAX)
```

MACRO Display state is valid

The macro decides whether `X` is valid display state

Parameters

in	*X*	display state

Returns

true if `X` is valid display state false otherwise

4.4.2.4 FONT_SIZE_IS_VALID

```
#define FONT_SIZE_IS_VALID(
              X ) (((uint8_t)(X)) < LCD1602_FONT_SIZE_MAX)
```

MACRO Font size is valid

The macro decides whether `X` is valid font size

Parameters

in	*X*	font size

Returns

true if `X` is valid font size false otherwise

4.4.2.5 LCD1602_DATA_PINS

```
#define LCD1602_DATA_PINS
```

Value:

```
(lcd1602_data_pins[0] | lcd1602_data_pins[1] | lcd1602_data_pins[2]
     | lcd1602_data_pins[3] | \
lcd1602_data_pins[4] | lcd1602_data_pins[5] | lcd1602_data_pins[6] |
     lcd1602_data_pins[7])
```

All mixed GPIO pins from lcd1602_data_pins, used for (de-)initalizating purposes.

4.4.2.6 NUM_LINE_IS_VALID

```
#define NUM_LINE_IS_VALID(
              X ) (((uint8_t)(X)) < LCD1602_NUM_LINE_MAX)
```

MACRO Number of line is valid

The macro decides whether `X` is valid number of line

Parameters

in	*X*	number of line

Returns

true if `X` is valid number of line false otherwise

4.4.2.7 SHIFT_ENTIRE_IS_VALID

```
#define SHIFT_ENTIRE_IS_VALID(
              X ) (((uint8_t)(X)) < LCD1602_SHIFT_ENTIRE_MAX)
```

MACRO Type shift of entire display is valid

The macro decides whether `X` is valid type shift of entire display

Parameters

in	*X*	type shift of entire display

Returns

true if `X` is valid type shift of entire display false otherwise

4.4.2.8 TYPE_INTERFACE_IS_VALID

```
#define TYPE_INTERFACE_IS_VALID(
              X ) (((uint8_t)(X)) < LCD1602_INTERFACE_MAX)
```

MACRO Type interface is valid

The macro decides whether `X` is valid type interface

Parameters

in	*X*	type interface

Returns

true if X is valid type interface false otherwise

4.4.2.9 TYPE_MOVE_CURSOR_IS_VALID

```
#define TYPE_MOVE_CURSOR_IS_VALID(
              X ) (((uint8_t)(X)) < LCD1602_CURSOR_MOVE_MAX)
```

MACRO Move type of cursor is valid

The macro decides whether X is valid move type of cursor

Parameters

in	*X*	move type of cursor

Returns

true if X is valid move type of cursor false otherwise

4.4.2.10 TYPE_SHIFT_IS_VALID

```
#define TYPE_SHIFT_IS_VALID(
              X ) (((uint8_t)(X)) < LCD1602_SHIFT_MAX)
```

MACRO Type shift is valid

The macro decides whether X is valid type shift

Parameters

in	*X*	type shift

Returns

true if X is valid type shift false otherwise

4.4.3 Enumeration Type Documentation

4.4.3.1 lcd1602_cursor_blink_state

```
enum lcd1602_cursor_blink_state
```

Cursor blink states.

Enumerator

LCD1602_CURSOR_BLINK_UNDEF	Cursor blink state is undefined.
LCD1602_CURSOR_BLINK_OFF	Cursor does NOT blink.
LCD1602_CURSOR_BLINK_ON	Cursor blinks.
LCD1602_CURSOR_BLINK_MAX	Count of cursor blink states.

4.4.3.2 lcd1602_cursor_state

```
enum lcd1602_cursor_state
```

Cursor states.

Enumerator

LCD1602_CURSOR_UNDEF	Cursor state is undefined.
LCD1602_CURSOR_OFF	Cursor is turned OFF.
LCD1602_CURSOR_ON	Cursor is turned ON.
LCD1602_CURSOR_MAX	Count of cursor states.

4.4.3.3 lcd1602_disp_state

```
enum lcd1602_disp_state
```

Display states.

Enumerator

LCD1602_DISPLAY_UNDEF	Display state is undefined.
LCD1602_DISPLAY_OFF	Display is turned OFF.
LCD1602_DISPLAY_ON	Display is turned ON.
LCD1602_DISPLAY_MAX	Count of display states.

4.4.3.4 lcd1602_font_size

```
enum lcd1602_font_size
```

Types of font size.

Enumerator

LCD1602_FONT_SIZE_UNDEF	Font size is undefined.
LCD1602_FONT_SIZE_5X8	Font size 5x8.
LCD1602_FONT_SIZE_5X11	Font size 5x11.
LCD1602_FONT_SIZE_MAX	Count of types of font size.

4.4.3.5 lcd1602_num_line

`enum lcd1602_num_line`

Numbrt line of LCD1602.

Enumerator

LCD1602_NUM_LINE_UNDEF	Number line is undefined.
LCD1602_NUM_LINE_1	First line.
LCD1602_NUM_LINE_2	Second line.
LCD1602_NUM_LINE_MAX	Count of lines.

4.4.3.6 lcd1602_shift_entire_disp

`enum lcd1602_shift_entire_disp`

Shift types of entire display.

Enumerator

LCD1602_SHIFT_ENTIRE_UNDEF	Shift type is undefined.
LCD1602_SHIFT_ENTIRE_PERFORMED	Shift of entire display is performed.
LCD1602_SHIFT_ENTIRE_NOT_PERFORMED	Shift of entire display is not performed.
LCD1602_SHIFT_ENTIRE_MAX	Count of shift types of entire display.

4.4.3.7 lcd1602_type_interface

`enum lcd1602_type_interface`

Type of LCD1602 interfaces.

Enumerator

LCD1602_INTERFACE_UNDEF	Inteface is undefined.
LCD1602_INTERFACE_4BITS	4-bit parallel interface
LCD1602_INTERFACE_8BITS	8-bit parallel interface
LCD1602_INTERFACE_MAX	Count of LCD1602 interfaces.

4.4.3.8 lcd1602_type_move_cursor

```
enum lcd1602_type_move_cursor
```

Move types of cursor.

Enumerator

LCD1602_CURSOR_MOVE_UNDEF	Move type is undefined.
LCD1602_CURSOR_MOVE_LEFT	Cursor moves left.
LCD1602_CURSOR_MOVE_RIGHT	Cursor moves right.
LCD1602_CURSOR_MOVE_MAX	Count of move types of cursor.

4.4.3.9 lcd1602_type_shift

```
enum lcd1602_type_shift
```

Type of cursor/display shift.

Enumerator

LCD1602_SHIFT_CURSOR_UNDEF	Type is undefined.
LCD1602_SHIFT_CURSOR_LEFT	Cursor shifts one position left.
LCD1602_SHIFT_CURSOR_RIGHT	Cursor shifts one position right.
LCD1602_SHIFT_DISPLAY_LEFT	Content of display shifts one position left.
LCD1602_SHIFT_DISPLAY_RIGHT	Content of display shifts one position right.
LCD1602_SHIFT_MAX	Count of shift types.

4.4.4 Function Documentation

4.4.4.1 __lcd1602_data_write()

```
static uint8_t __lcd1602_data_write (
             uint8_t data )  [static]
```

Write data to LCD1602

Parameters

in	*data*	value of data register

Returns

RES_OK on success error otherwise

4.4.4.2 __lcd1602_instruction_write()

```
static uint8_t __lcd1602_instruction_write (
             uint8_t instruction )  [static]
```

Write instruction to LCD1602

Parameters

in	*instruction*	value of insturction register

Returns

RES_OK on success error otherwise

4.4.4.3 __lcd1602_printf()

```
static uint8_t __lcd1602_printf (
             const char * line1,
             const char * line2,
             bool is_centered,
             va_list argp )  [static]
```

Print on display

The function prints formatted strings on LCD1602

Parameters

in	*line1*	formatted first line, if NULL - previous content remains
in	*line2*	formatted second line, if NULL - previous content remains
in	*is_centered*	flag whether content within each line should be centered
in	*argp*	formatting arguments over two lines

Returns

RES_OK on success error otherwise

4.4.4.4 __lcd1602_read_busy_flag()

```
static uint8_t __lcd1602_read_busy_flag (
            uint8_t * busy_flag,
            uint8_t * address_counter )  [static]
```

Read BUSY flag from LCD1602

Parameters

out	*busy_flag*	read BUSY flag, if NULL - not returned
out	*address_counter*	read address counter, if NULL - not returned

Returns

RES_OK on success error otherwise

4.4.4.5 __lcd1602_read_write()

```
static uint8_t __lcd1602_read_write (
            uint8_t * data,
            uint8_t type_reg,
            uint8_t type_mode )  [static]
```

Read/write operation with LCD1602

Parameters

in,out	*data*	read/written value from/to insturction/data register
in	*type_reg*	type of register, can be LCD1602_INSTR_REG or LCD1602_DATA_REG
in	*type_mode*	read or write mode, can be LCD1602_READ_MODE or LCD1602_WRITE_MODE

Returns

RES_OK on success error otherwise

4.4.4.6 __lcd1602_wait()

```
static uint8_t __lcd1602_wait (
            const uint32_t timeout )  [static]
```

Wait for finish LCD1602 operation

The function waits when BUSY flag is reset,
used to ensure that display is ready for next operation

Parameters

in	*timeout*	timeout for waiting in ms

Returns

RES_OK on success error otherwise

4.4.4.7 bsp_lcd1602_cgram_address_set()

```
uint8_t bsp_lcd1602_cgram_address_set (
            const uint8_t address )
```

CGRAM address set

Parameters

in	*address*	CGRAM address

Returns

RES_OK on success error otherwise

4.4.4.8 bsp_lcd1602_cprintf()

```
uint8_t bsp_lcd1602_cprintf (
            const char * line1,
            const char * line2,
             ... )
```

Centered print on display

API implemented as wrapper over centered __lcd1602_printf

Parameters

in	*line1*	formatted first line, if NULL - previous content remains
in	*line2*	formatted second line, if NULL - previous content remains
in	...	variable argument list for formatting of both lines

Returns

RES_OK on success error otherwise

4.4.4.9 bsp_lcd1602_cursor_disp_shift()

```
uint8_t bsp_lcd1602_cursor_disp_shift (
            const enum lcd1602_type_shift shift )
```

Cursor or display shift

The function makes shift according to `shift`

Parameters

in	*shift*	type of cursor/display shift

Returns

RES_OK on success error otherwise

4.4.4.10 bsp_lcd1602_ddram_address_set()

```
uint8_t bsp_lcd1602_ddram_address_set (
            const uint8_t address )
```

DDRAM address set

Parameters

in	*address*	DDRAM address

Returns

RES_OK on success error otherwise

4.4.4.11 bsp_lcd1602_deinit()

```
uint8_t bsp_lcd1602_deinit (
            void  )
```

LCD1602 deinitialization

The function clears display and does MSP deinitialization

Returns

RES_OK on success error otherwise

4.4.4.12 bsp_lcd1602_display_clear()

```
uint8_t bsp_lcd1602_display_clear (
             void  )
```

Display clear

Returns

RES_OK on success error otherwise

4.4.4.13 bsp_lcd1602_display_on_off()

```
uint8_t bsp_lcd1602_display_on_off (
             const enum lcd1602_disp_state disp_state,
             const enum lcd1602_cursor_state cursor_state,
             const enum lcd1602_cursor_blink_state cursor_blink_state )
```

Display ON/OFF

Parameters

in	*disp_state*	display state
in	*cursor_state*	cursor state
in	*cursor_blink_state*	cursor blink state

Returns

RES_OK on success error otherwise

4.4.4.14 bsp_lcd1602_entry_mode_set()

```
uint8_t bsp_lcd1602_entry_mode_set (
             const enum lcd1602_type_move_cursor cursor,
             const enum lcd1602_shift_entire_disp shift_entire )
```

Entry mode set

Parameters

in	*cursor*	move type of cursor
in	*shift_entire*	shift type of entire display

Returns

RES_OK on success error otherwise

4.4.4.15 bsp_lcd1602_function_set()

```
uint8_t bsp_lcd1602_function_set (
            const enum lcd1602_type_interface interface,
            const enum lcd1602_num_line num_line,
            const enum lcd1602_font_size font_size )
```

Set function to LCD1602

Parameters

in	*interface*	type of interface
in	*num_line*	1-line or 2-line mode of display
in	*font_size*	font size

Returns

RES_OK on success error otherwise

4.4.4.16 bsp_lcd1602_init()

```
uint8_t bsp_lcd1602_init (
            struct lcd1602_settings * init_settings )
```

LCD1602 initialization

The function does MSP initialization, sets settings to LCD1602 accoring to `init_settings`

Note

BSP LCD1602 is designed so that 4-bit interface is not supported

Parameters

in	*init_settings*	settings of LCD1602

Returns

RES_OK on success error otherwise

4.4.4.17 bsp_lcd1602_printf()

```
uint8_t bsp_lcd1602_printf (
            const char * line1,
            const char * line2,
             ... )
```

Not centered print on display

API implemented as wrapper over not centered __lcd1602_printf

Parameters

in	*line1*	formatted first line, if NULL - previous content remains
in	*line2*	formatted second line, if NULL - previous content remains
in	...	variable argument list for formatting of both lines

Returns

RES_OK on success error otherwise

4.4.4.18 bsp_lcd1602_return_home()

```
uint8_t bsp_lcd1602_return_home (
            void )
```

Return home

Value of address counter is set to 0 and
current position on display is set to start of first line

Returns

RES_OK on success error otherwise

4.4.5 Variable Documentation

4.4.5.1 lcd1602_data_pins

```
const uint16_t lcd1602_data_pins[ ]  [static]
```

Initial value:

```
= {GPIO_PIN_15, GPIO_PIN_14, GPIO_PIN_13, GPIO_PIN_7,
                                                  GPIO_PIN_8, GPIO_PIN_9, GPIO_PIN_12, GPIO_PIN_11, 0}
```

Array of GPIO pins used for 8-bit parallel interface.

4.5 BSP LED RGB

Module of BSP LED RGB

Data Structures

- struct bsp_led_rgb

    RGB LED structure.

- struct bsp_led_pwm

    Parameters of RGB LED blinking.

Macros

- #define BSP_LED_RGB_HARDFAULT()
- #define **RGB_TIM_FREQ** 1000

    Frequency of RGB timer in Hz.

- #define **RGB_TIM_PERIOD** UINT16_MAX

    Value of period register of RGB timer.

- #define **BLINK_TIM_FREQ** 2000

    Frequency of blink timer in Hz.

Functions

- uint8_t bsp_led_rgb_calibrate (const struct bsp_led_rgb ∗coef_rgb)
- uint8_t bsp_led_rgb_set (const struct bsp_led_rgb ∗rgb)
- uint8_t bsp_led_rgb_init (void)
- uint8_t bsp_led_rgb_deinit (void)
- uint8_t bsp_led_rgb_blink_enable (const struct bsp_led_pwm ∗pwm)
- uint8_t bsp_led_rgb_blink_disable (void)
- static void __led_rgb_tim_pwm_msp_init (TIM_HandleTypeDef ∗htim)
- static void __led_rgb_tim_pwm_msp_deinit (TIM_HandleTypeDef ∗htim)
- static void __led_rgb_tim_msp_post_init (void)
- static void __led_rgb_tim_msp_prev_deinit (void)
- static void __led_rgb_blink_tim_period_elapsed_callback (TIM_HandleTypeDef ∗htim)
- static void __led_rgb_blink_tim_pwm_pulse_finished_callback (TIM_HandleTypeDef ∗htim)
- static uint8_t __led_rgb_blink_start (void)
- static uint8_t __led_rgb_blink_stop (void)
- static bool __led_rgb_blink_is_started (void)
- void TIM2_IRQHandler (void)

Variables

- static TIM_HandleTypeDef **htim_rgb** = {.Instance = TIM1}

  STM32 HAL TIM instance of RGB timer.
- static TIM_HandleTypeDef **htim_blink** = {.Instance = TIM2}

  STM32 HAL TIM instance of blink timer.
- static uint32_t **led_rgb_tim_channels** [ ] = {TIM_CHANNEL_1, TIM_CHANNEL_2, TIM_CHANNEL_3}

  Array of STM32 HAL TIM channels.
- static float **coef_r** = 1.0f

  Corrective coefficient for red channel, set by bsp_led_rgb_calibrate.
- static float **coef_g** = 1.0f

  Corrective coefficient for green channel, set by bsp_led_rgb_calibrate.
- static float **coef_b** = 1.0f

  Corrective coefficient for blue channel, set by bsp_led_rgb_calibrate.

4.5.1 Detailed Description

Module of BSP LED RGB

```
The module includes two STM32 timers:
1. RGB timer which shapes RGB color of the LED by PWM
2. Blink timer which makes blinking of the LED
```

4.5.2 Macro Definition Documentation

4.5.2.1 BSP_LED_RGB_HARDFAULT

```
#define BSP_LED_RGB_HARDFAULT( )
```

Value:

```
do {\
    BSP_GPIO_FORCE_OUTPUT_MODE(GPIOA, 8);\
    BSP_GPIO_FORCE_OUTPUT_MODE(GPIOA, 9);\
    BSP_GPIO_FORCE_OUTPUT_MODE(GPIOA, 10);\
    \
    BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_8, false);\
    BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_10, false);\
    \
    while (true) {\
        BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_9, false);\
        INSTR_DELAY_US(100000);\
        BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_9, true);\
        INSTR_DELAY_US(100000);\
    }\
} while (0)
```

MACRO Activate specific RGB LED behaivour

The macro is used to activate specific RGB LED behaivour
in case of firmware hardfaults. Call of this macro
unconditionally disables other settings of RGB LED

Warning

The macro is firmware dead end and reset is needed to start firmware

4.5.3 Function Documentation

4.5.3.1 __led_rgb_blink_is_started()

```
static bool __led_rgb_blink_is_started (
            void  )  [static]
```

Flag whether blink timer is started

Returns

true if started false otherwise

4.5.3.2 __led_rgb_blink_start()

```
static uint8_t __led_rgb_blink_start (
            void  )  [static]
```

Start blink timer

The function starts blink timer, used to enable blink feature

Returns

RES_OK on success error otherwise

4.5.3.3 __led_rgb_blink_stop()

```
static uint8_t __led_rgb_blink_stop (
            void  )  [static]
```

Stop blink timer

The function stops blink timer, used to disable blink feature

Returns

RES_OK on success error otherwise

4.5.3.4 __led_rgb_blink_tim_period_elapsed_callback()

```
static void __led_rgb_blink_tim_period_elapsed_callback (
            TIM_HandleTypeDef * htim )  [static]
```

Callback by period elapsion of blink timer

The function enables outputs of RGB timer turning ON RGB LED (enabled phase of blink)

Parameters

in	*htim*	STM32 HAL TIM instance, should equal to htim_blink

4.5.3.5 __led_rgb_blink_tim_pwm_pulse_finished_callback()

```
static void __led_rgb_blink_tim_pwm_pulse_finished_callback (
             TIM_HandleTypeDef * htim )  [static]
```

Callback by pulse elapsion of blink timer

The function disables outputs of RGB timer turning OFF RGB LED (disabled phase of blink)

Parameters

in	*htim*	STM32 HAL TIM instance, should equal to htim_blink

4.5.3.6 __led_rgb_tim_msp_post_init()

```
static void __led_rgb_tim_msp_post_init (
             void  )  [static]
```

Timer posterior MSP initialization

The function executes GPIO initialization for PWM purposes of RGB timer

4.5.3.7 __led_rgb_tim_msp_prev_deinit()

```
static void __led_rgb_tim_msp_prev_deinit (
             void  )  [static]
```

Timer preliminary MSP deinitialization

The function executes deinitialization of GPIO used of RGB timer

4.5.3.8 __led_rgb_tim_pwm_msp_deinit()

```
static void __led_rgb_tim_pwm_msp_deinit (
             TIM_HandleTypeDef * htim )  [static]
```

Timer main MSP deinitialization

The function executes clock and NVIC deinitialization

Parameters

in	*htim*	STM32 HAL TIM instance, should equal to htim_rgb or htim_blink

4.5.3.9 __led_rgb_tim_pwm_msp_init()

```
static void __led_rgb_tim_pwm_msp_init (
            TIM_HandleTypeDef * htim )  [static]
```

Timer main MSP initialization

The function executes clock and NVIC initialization

Parameters

in	*htim*	STM32 HAL TIM instance, should equal to htim_rgb or htim_blink

4.5.3.10 bsp_led_rgb_blink_disable()

```
uint8_t bsp_led_rgb_blink_disable (
            void  )
```

LED RGB blinking disable

Returns

RES_OK on success error otherwise

4.5.3.11 bsp_led_rgb_blink_enable()

```
uint8_t bsp_led_rgb_blink_enable (
            const struct bsp_led_pwm * pwm )
```

LED RGB blinking enable

Parameters

in	*pwm*	parameters of LED blinking

Returns

RES_OK on success error otherwise

4.5.3.12 bsp_led_rgb_calibrate()

```
uint8_t bsp_led_rgb_calibrate (
            const struct bsp_led_rgb * coef_rgb )
```

Calibration of RGB LED

The function sets corrective coefficients, used to adjust
color of LED light according to perception.
Each corrective coefficient means maximum level which can be set so
the less the coefficient the less maximum brightness of appropriate color

As consequence if some coefficitient is:
255 - appropriate color channel is with no corrections,
0 - color channel is not used in LED light

Parameters

in	*coef_rgb*	RGB corrective coefficients

Returns

RES_OK on success error otherwise

4.5.3.13 bsp_led_rgb_deinit()

```
uint8_t bsp_led_rgb_deinit (
            void )
```

BSP RGB LED deinitialization

The function executes deinitialization of RGB & blink timers

Returns

RES_OK on success error otherwise

4.5.3.14 bsp_led_rgb_init()

```
uint8_t bsp_led_rgb_init (
            void  )
```

BSP RGB LED initialization

The function executes initialization of RGB & blink timers,
both of them are disabled afterwards

Returns

RES_OK on success error otherwise

4.5.3.15 bsp_led_rgb_set()

```
uint8_t bsp_led_rgb_set (
            const struct bsp_led_rgb * rgb )
```

Set RGB value for LED

Parameters

in	*rgb*	color of LED light in RGB format

Returns

RES_OK on success error otherwise

4.5.3.16 TIM2_IRQHandler()

```
void TIM2_IRQHandler (
            void  )
```

NVIC TIM2 (blink timer) IRQ handler

4.6 BSP RCC

Module of BSP RCC.

Macros

- #define TIM_APB_NUM_CLOCK_GET(INSTANCE)

Functions

- uint8_t bsp_rcc_main_config_init (void)
- uint32_t bsp_rcc_apb_timer_freq_get (TIM_TypeDef ∗instance)

4.6.1 Detailed Description

Module of BSP RCC.

4.6.2 Macro Definition Documentation

4.6.2.1 TIM_APB_NUM_CLOCK_GET

```
#define TIM_APB_NUM_CLOCK_GET(
              INSTANCE )
```

Value:

```
((IS_TIM_INSTANCE(INSTANCE)) ? (\
(((INSTANCE) == TIM1) || \
((INSTANCE) == TIM8) || \
((INSTANCE) == TIM9) || \
((INSTANCE) == TIM10) || \
((INSTANCE) == TIM11)) ?  2 :  1) :  0)
```

MACRO number of APB source clock for timers

Note

The macro is designed to used for chips STM32F446xx

Parameters

in	*INSTANCE*	TIM instance

Returns

1 for APB1, 2 for APB2, 0 if error

4.6.3 Function Documentation

4.6.3.1 bsp_rcc_apb_timer_freq_get()

```
uint32_t bsp_rcc_apb_timer_freq_get (
             TIM_TypeDef * instance )
```

Get frequency of TIM internal clock

Parameters

in	*instance*	STM32 HAL TIM instance

Returns

frequency in Hz

4.6.3.2 bsp_rcc_main_config_init()

```
uint8_t bsp_rcc_main_config_init (
            void  )
```

Configuration of main clocks

The function executes configuration of main MPU clocks

Returns

RES_OK on success error otherwise

4.7 BSP

Board suppport package.

Modules

- BSP button

  Module of BSP button.
- BSP CRC

  Module of BSP CRC.
- BSP GPIO

  Module of BSP GPIO.
- BSP LCD1602

  Module of BSP `LCD1602`
- BSP LED RGB

  Module of BSP LED RGB
- BSP RCC

  Module of BSP RCC.
- BSP UART

  Module of BSP UART.

4.7.1 Detailed Description

Board suppport package.

4.8 BSP UART

Module of BSP UART.

Data Structures

- struct uart_init_ctx

  BSP UART initializing context.
- struct uart_ctx

  Context of the BSP UART instance.

Macros

- #define UART_TYPE_VALID(X) (((uint32_t)(X) < BSP_UART_TYPE_MAX))
- #define UART_WORDLEN_VALID(X) (((X) == BSP_UART_WORDLEN_8) || ((X) == BSP_UART_WORDLEN_9))
- #define UART_PARITY_VALID(X) (((X) == BSP_UART_PARITY_NONE) || ((X) == BSP_UART_PARITY_EVEN) || ((X) == BSP_UART_PARITY_ODD))
- #define UART_STOPBITS_VALID(X) (((X) == BSP_UART_STOPBITS_1) || ((X) == BSP_UART_STOPBITS_2))
- #define **BSP_UART_ERROR_PE** HAL_UART_ERROR_PE

  BSP UART parity error.
- #define **BSP_UART_ERROR_NE** HAL_UART_ERROR_NE

  BSP UART noise error.
- #define **BSP_UART_ERROR_FE** HAL_UART_ERROR_FE

  BSP UART frame error.
- #define **BSP_UART_ERROR_ORE** HAL_UART_ERROR_ORE

  BSP UART overrun error.
- #define **BSP_UART_ERROR_DMA** HAL_UART_ERROR_DMA

  BSP UART DMA error.
- #define **BSP_UART_ERRORS_ALL** (BSP_UART_ERROR_PE | BSP_UART_ERROR_NE | BSP_UART_ERROR_FE | BSP_UART_ERROR_ORE | BSP_UART_ERROR_DMA)

  Mask including all possible BSP UART errors.
- #define HAL_UART_WORDLEN_TO(X) (((X) == BSP_UART_WORDLEN_8) ? UART_WORDLENGTH_8B : UART_WORDLENGTH_9B)
- #define HAL_UART_STOPBITS_TO(X) (((X) == BSP_UART_STOPBITS_1) ? UART_STOPBITS_1 ↩ : UART_STOPBITS_2)
- #define HAL_UART_PARITY_TO(X)

Enumerations

- enum uart_type { BSP_UART_TYPE_CLI = 0 , BSP_UART_TYPE_RS232_TX , BSP_UART_TYPE_RS232_RX , BSP_UART_TYPE_MAX }

  Types of BSP UART instances.
- enum uart_wordlen { BSP_UART_WORDLEN_8 = 8 , BSP_UART_WORDLEN_9 = 9 }

  BSP UART word length.
- enum uart_parity { BSP_UART_PARITY_NONE = 0 , BSP_UART_PARITY_EVEN = 1 , BSP_UART_PARITY_ODD = 2 }

  BSP UART parity types.
- enum uart_stopbits { BSP_UART_STOPBITS_1 = 1 , BSP_UART_STOPBITS_2 = 2 }

  BSP UART stop bits count.

Functions

- uint8_t bsp_uart_init (enum uart_type type, struct uart_init_ctx ∗init)
- uint8_t bsp_uart_deinit (enum uart_type type)
- uint8_t bsp_uart_read (enum uart_type type, uint8_t ∗data, uint16_t ∗len, uint32_t tmt_ms)
- uint8_t bsp_uart_write (enum uart_type type, uint8_t ∗data, uint16_t len, uint32_t tmt_ms)
- uint8_t bsp_uart_start (enum uart_type type)
- uint8_t bsp_uart_stop (enum uart_type type)
- bool bsp_uart_is_started (enum uart_type type)
- bool bsp_uart_rx_buffer_is_empty (enum uart_type type)
- static enum uart_type __uart_type_get (USART_TypeDef ∗instance)
- static uint8_t __uart_dma_deinit (enum uart_type type)
- static uint8_t __uart_msp_deinit (enum uart_type type)
- static uint8_t __uart_dma_init (enum uart_type type)
- static uint8_t __uart_msp_init (enum uart_type type)
- static void __uart_rx_callback (UART_HandleTypeDef ∗huart, uint16_t pos)
- static void __uart_error_callback (enum uart_type type, uint32_t error)
- static void __uart_irq_handler (enum uart_type type)
- void UART4_IRQHandler (void)
- void USART2_IRQHandler (void)
- void USART3_IRQHandler (void)
- void DMA1_Stream1_IRQHandler (void)
- void DMA1_Stream2_IRQHandler (void)
- void DMA1_Stream4_IRQHandler (void)
- void DMA1_Stream5_IRQHandler (void)

Variables

- struct {
  UART_HandleTypeDef **uart**
  STM32 HAL UART instance.
  struct uart_ctx ∗ **ctx**
  Context of the instance.
  } uart_obj [BSP_UART_TYPE_MAX]

4.8.1 Detailed Description

Module of BSP UART.

4.8.2 Macro Definition Documentation

4.8.2.1 HAL_UART_PARITY_TO

```
#define HAL_UART_PARITY_TO(
              X )
```

Value:

```
(((X) == BSP_UART_PARITY_NONE) ?  UART_PARITY_NONE : \
(((X) == BSP_UART_PARITY_EVEN) ?  UART_PARITY_EVEN : UART_PARITY_ODD))
```

MACRO BSP UART parity type typecasting

The macro makes typecasting of uart_parity to STM32 HAL UART parity type

Parameters

in	*X*	BSP UART parity type

Returns

STM32 HAL UART parity type

4.8.2.2 HAL_UART_STOPBITS_TO

```
#define HAL_UART_STOPBITS_TO(
              X ) (((X) == BSP_UART_STOPBITS_1) ? UART_STOPBITS_1 : UART_STOPBITS_2)
```

MACRO BSP UART stop bits count typecasting

The macro makes typecasting of uart_stopbits to STM32 HAL UART stop bits count

Parameters

in	*X*	BSP UART stop bits count

Returns

STM32 HAL UART stop bits count

4.8.2.3 HAL_UART_WORDLEN_TO

```
#define HAL_UART_WORDLEN_TO(
              X ) (((X) == BSP_UART_WORDLEN_8) ? UART_WORDLENGTH_8B : UART_WORDLENGTH_9B)
```

MACRO BSP UART word length typecasting

The macro makes typecasting of uart_wordlen to STM32 HAL UART word length

Parameters

in	*X*	BSP UART word length

Returns

STM32 HAL UART word length

4.8.2.4 UART_PARITY_VALID

```
#define UART_PARITY_VALID(
              X ) (((X) == BSP_UART_PARITY_NONE) || ((X) == BSP_UART_PARITY_EVEN) || ((X) ==
BSP_UART_PARITY_ODD))
```

MACRO Check BSP UART parity type

The macro checks whether `X` is valid BSP UART parity type

Parameters

in	*X*	BSP UART parity type

Returns

true if parity type is valid false otherwise

4.8.2.5 UART_STOPBITS_VALID

```
#define UART_STOPBITS_VALID(
              X ) (((X) == BSP_UART_STOPBITS_1) || ((X) == BSP_UART_STOPBITS_2))
```

MACRO Check BSP UART stop bits count

The macro checks whether `X` is valid BSP UART stop bits count

Parameters

in	*X*	BSP UART stop bits count

Returns

true if stop bits count is valid false otherwise

4.8.2.6 UART_TYPE_VALID

```
#define UART_TYPE_VALID(
              X ) (((uint32_t)(X) < BSP_UART_TYPE_MAX))
```

MACRO Check BSP UART type

The macro checks whether `X` is valid BSP UART type

Parameters

in	*X*	BSP UART type

Returns

true if type is valid false otherwise

4.8.2.7 UART_WORDLEN_VALID

```
#define UART_WORDLEN_VALID(
              X ) (((X) == BSP_UART_WORDLEN_8) || ((X) == BSP_UART_WORDLEN_9))
```

MACRO Check BSP UART word length

The macro checks whether `X` is valid BSP UART word length

Parameters

in	*X*	BSP UART word length

Returns

true if word length is valid false otherwise

4.8.3 Enumeration Type Documentation

4.8.3.1 uart_parity

```
enum uart_parity
```

BSP UART parity types.

Enumerator

BSP_UART_PARITY_NONE	None parity.
BSP_UART_PARITY_EVEN	Even parity.
BSP_UART_PARITY_ODD	Odd parity.

4.8.3.2 uart_stopbits

```
enum uart_stopbits
```

BSP UART stop bits count.

Enumerator

BSP_UART_STOPBITS↩_1	1 stop bit
BSP_UART_STOPBITS↩_2	2 stop bits

4.8.3.3 uart_type

```
enum uart_type
```

Types of BSP UART instances.

Enumerator

BSP_UART_TYPE_CLI	CLI.
BSP_UART_TYPE_RS232_TX	RS-232 TX channel (RX only)
BSP_UART_TYPE_RS232_RX	RS-232 RX channel (RX only)
BSP_UART_TYPE_MAX	Count of BSP UART types.

4.8.3.4 uart_wordlen

```
enum uart_wordlen
```

BSP UART word length.

Enumerator

BSP_UART_WORDLEN↩_8	Word length is 8 bits.
BSP_UART_WORDLEN↩_9	Word length is 9 bits.

4.8.4 Function Documentation

4.8.4.1 __uart_dma_deinit()

```
static uint8_t __uart_dma_deinit (
             enum uart_type type )  [static]
```

STM32 DMA UART deinitialization

The function executes DMA TX/RX (according to BSP UART type) deinitialization

Parameters

in	*type*	BSP UART type

Returns

RES_OK on success error otherwise

4.8.4.2 __uart_dma_init()

```
static uint8_t __uart_dma_init (
             enum uart_type type )  [static]
```

STM32 DMA UART initialization

The function executes DMA TX/RX (according to BSP UART type) initialization

Parameters

in	*type*	BSP UART type

Returns

RES_OK on success error otherwise

4.8.4.3 __uart_error_callback()

```
static void __uart_error_callback (
             enum uart_type type,
             uint32_t error )  [static]
```

Callback by BSP UART error

The function is called from __uart_irq_handler when BSP UART error occured
The function is the wrapper over user callback uart_init_ctx::error_isr_cb

Parameters

in	*type*	BSP UART type
in	*error*	mask of occured BSP UART errors

4.8.4.4 __uart_irq_handler()

```
static void __uart_irq_handler (
             enum uart_type type )  [static]
```

UART IRQ handler

The function is called from NVIC UART interrupts, processes receiption, errors and LIN break detection

Parameters

in	*type*	BSP UART type

4.8.4.5 __uart_msp_deinit()

```
static uint8_t __uart_msp_deinit (
             enum uart_type type )  [static]
```

STM32 UART MSP deinitialization

The function executes GPIO, DMA deinitialization, disables corresponding clocks

Parameters

in	*type*	BSP UART type

Returns

RES_OK on success error otherwise

4.8.4.6 __uart_msp_init()

```
static uint8_t __uart_msp_init (
             enum uart_type type )  [static]
```

STM32 UART MSP initialization

The function executes GPIO, DMA initialization, enables corresponding clocks

Parameters

in	*type*	BSP UART type

Returns

RES_OK on success error otherwise

4.8.4.7 __uart_rx_callback()

```
static void __uart_rx_callback (
            UART_HandleTypeDef * huart,
            uint16_t pos )  [static]
```

Callback by data reception

The function is called by STM32 HAL UART by idle detection if data was received
The function operates with write position of uart_ctx::rx_buff, set overflow flag
if appropriate event is occured

Parameters

in	*huart*	STM32 HAL UART instance
in	*pos*	current write position of uart_ctx::rx_buff

4.8.4.8 __uart_type_get()

```
static enum uart_type __uart_type_get (
            USART_TypeDef * instance )  [static]
```

Get BSP UART type by STM32 HAL UART instance

Parameters

in	*instance*	STM32 HAL UART instance

Returns

BSP UART type on success BSP_UART_TYPE_MAX otherwise

4.8.4.9 bsp_uart_deinit()

```
uint8_t bsp_uart_deinit (
             enum uart_type type )
```

Deinitialization of BSP UART instance

The function executes deinitizalition of BSP UART instance

Parameters

in	*type*	BSP UART type

Returns

RES_OK on success error otherwise

4.8.4.10 bsp_uart_init()

```
uint8_t bsp_uart_init (
             enum uart_type type,
             struct uart_init_ctx * init )
```

Initialization of BSP UART instance

The function executes initizalition of BSP UART instance according
to settings stored in `init`
if appropriate BSP UART instance is initialized it will be reinitialized

Parameters

in	*type*	BSP UART type
in	*init*	initializating context of BSP UART instance

Returns

RES_OK on success error otherwise

4.8.4.11 bsp_uart_is_started()

```
bool bsp_uart_is_started (
             enum uart_type type )
```

Flag whether BSP UART instance is started

Parameters

in	*type*	BSP UART type

Returns

true if the instance is started false otherwise

4.8.4.12 bsp_uart_read()

```
uint8_t bsp_uart_read (
            enum uart_type type,
            uint8_t * data,
            uint16_t * len,
            uint32_t tmt_ms )
```

Receive BSP UART data

The function executes reading of data received via DMA UART

Note

The function is blocking if `tmt_ms` is not zero

Parameters

in	*type*	BSP UART type
out	*data*	received data
out	*len*	size of received data
in	*tmt_ms*	timeout for receiving in ms

Returns

RES_OK on success error otherwise

4.8.4.13 bsp_uart_rx_buffer_is_empty()

```
bool bsp_uart_rx_buffer_is_empty (
            enum uart_type type )
```

Flag whether received buffer is empty

Parameters

in	*type*	BSP UART type

Returns

true if uart_ctx::rx_buff is empty false otherwise

4.8.4.14 bsp_uart_start()

```
uint8_t bsp_uart_start (
            enum uart_type type )
```

BSP UART instance start

The function (re-)start UART DMA reception, enable appropriate interrupts and etc.

Parameters

in	*type*	BSP UART type

Returns

RES_OK on success error otherwise

4.8.4.15 bsp_uart_stop()

```
uint8_t bsp_uart_stop (
            enum uart_type type )
```

BSP UART instance stop

The function stop UART DMA reception/sending

Parameters

in	*type*	BSP UART type

Returns

RES_OK on success error otherwise

4.8.4.16 bsp_uart_write()

```
uint8_t bsp_uart_write (
            enum uart_type type,
            uint8_t * data,
            uint16_t len,
            uint32_t tmt_ms )
```

Send BSP UART data

The function executes sending of data via DMA UART

Note

The function is blocking if previous DMA UART sending is not completed and `tmt_ms` is not zero

Parameters

in	*type*	BSP UART type
in	*data*	sent data
in	*len*	size of sent data
in	*tmt_ms*	timeout for sending in ms

Returns

RES_OK on success error otherwise

4.8.4.17 DMA1_Stream1_IRQHandler()

```
void DMA1_Stream1_IRQHandler (
             void )
```

NVIC DMA1 (Stream 1) IRQ handler

4.8.4.18 DMA1_Stream2_IRQHandler()

```
void DMA1_Stream2_IRQHandler (
             void )
```

NVIC DMA1 (Stream 2) IRQ handler

4.8.4.19 DMA1_Stream4_IRQHandler()

```
void DMA1_Stream4_IRQHandler (
             void )
```

NVIC DMA1 (Stream 4) IRQ handler

4.8.4.20 DMA1_Stream5_IRQHandler()

```
void DMA1_Stream5_IRQHandler (
             void  )
```

NVIC DMA1 (Stream 5) IRQ handler

4.8.4.21 UART4_IRQHandler()

```
void UART4_IRQHandler (
             void  )
```

NVIC UART4 IRQ handler

4.8.4.22 USART2_IRQHandler()

```
void USART2_IRQHandler (
             void  )
```

NVIC USART2 IRQ handler

4.8.4.23 USART3_IRQHandler()

```
void USART3_IRQHandler (
             void  )
```

NVIC USART3 IRQ handler

4.8.5 Variable Documentation

4.8.5.1

```
struct { ...  } uart_obj[BSP_UART_TYPE_MAX]  [static]
```

Initial value:

```
= {
    {.uart = {.Instance = UART4}, .ctx = NULL},
    {.uart = {.Instance = USART2}, .ctx = NULL},
    {.uart = {.Instance = USART3}, .ctx = NULL}
}
```

Array of BSP UART instances

Includes three instances:
BSP_UART_TYPE_CLI - CLI using STM32 UART4 TX/RX
BSP_UART_TYPE_RS232_TX - RS-232 TX channel using STM32 USART2 RX
BSP_UART_TYPE_RS232_RX - RS-232 RX channel using STM32 USART3 RX

4.9 Common

Common libraries for generic purposes.

Macros

- #define **RES_OK** 0

  Return code: Success.
- #define **RES_NOK** 1

  Return code: Not specified error.
- #define **RES_INVALID_PAR** 2

  Return code: Invalid input parameter(-s)
- #define **RES_MEMORY_ERR** 3

  Return code: Memory allocation error.
- #define **RES_TIMEOUT** 4

  Return code: Timeout occured.
- #define **RES_NOT_SUPPORTED** 5

  Return code: Some feature is not supported.
- #define **RES_OVERFLOW** 6

  Return code: Overflow of an object.
- #define **RES_NOT_INITIALIZED** 7

  Return code: An object is not initialized.
- #define **RES_NOT_ALLOWED** 8

  Return code: An object/feature is not allowed.
- #define ARRAY_SIZE(X) (sizeof(X) / sizeof(X[0]))
- #define MIN(X, Y) (((X) < (Y)) ? (X) : (Y))
- #define MAX(X, Y) (((X) > (Y)) ? (X) : (Y))
- #define IS_PRINTABLE(X) (X >= ' ' && X <= '~')
- #define INSTR_DELAY_US(DELAY)

4.9.1 Detailed Description

Common libraries for generic purposes.

4.9.2 Macro Definition Documentation

4.9.2.1 ARRAY_SIZE

```
#define ARRAY_SIZE(
              X ) (sizeof(X) / sizeof(X[0]))
```

MACRO Array size

The macro calculates size of an array, statically allocated

Parameters

in	*X*	an array

Returns

size of an array

4.9.2.2 INSTR_DELAY_US

```
#define INSTR_DELAY_US(
                DELAY )
```

Value:

```
do {\
    __IO uint32_t clock_delay = DELAY * (HAL_RCC_GetSysClockFreq() / 8 / 1000000);\
    do {\
        __NOP();\
    } while (--clock_delay);\
} while (0)
```

MACRO Delay by MPU instructions in us

The macro uses MPU instructions to make delay.
The macro normally used when it needs to make delay with duration less 1 ms
or other delay functions are unavailable

Warning

The delay is inaccurate as the function does not take into account
the time spent for interrups routine

Parameters

in	*DELAY*	value of delay in us

4.9.2.3 IS_PRINTABLE

```
#define IS_PRINTABLE(
                X ) (X >= ' ' && X <= '~')
```

MACRO Flag of an printable symbol

The macro decides whether symbol `X` is printable ASCII symbol

Parameters

in	*X*	symbol in char format

Returns

true if symbol is printable false otherwise

4.9.2.4 MAX

```
#define MAX(
              X,
              Y ) (((X) > (Y)) ? (X) : (Y))
```

MACRO Maximum from two values

Parameters

in	*X*	first value
in	*Y*	second value

Returns

maximal value

4.9.2.5 MIN

```
#define MIN(
              X,
              Y ) (((X) < (Y)) ? (X) : (Y))
```

MACRO Minimum from two values

Parameters

in	*X*	first value
in	*Y*	second value

Returns

minimal value

4.10 Application layer of RGB LED

Module of application layer of RGB LED.

Macros

- #define LED_EVENT_IS_VALID(X) (((uint32_t)(X)) < LED_EVENT_MAX)

Enumerations

- enum led_event {
  LED_EVENT_NONE = 0 , LED_EVENT_COMMON_ERROR , LED_EVENT_CRC_ERROR , LED_EVENT_FLASH_ERROR
  ,
  LED_EVENT_LCD1602_ERROR , LED_EVENT_IN_PROCESS , LED_EVENT_SUCCESS , LED_EVENT_FAILED
  ,
  LED_EVENT_UART_ERROR , LED_EVENT_UART_OVERFLOW , LED_EVENT_MAX }

    RGB LED event (type of LED behaivour)

Functions

- uint8_t app_led_init (void)
- uint8_t app_led_deinit (void)
- uint8_t app_led_set (enum led_event led_event)

Variables

- static const struct bsp_led_rgb **led_disabled** = {.r = 0, .g = 0, .b = 0}

    Settings for disabled LED.
- static const struct bsp_led_rgb **led_red** = {.r = 255, .g = 0, .b = 0}

    Settings for LED with RED color.
- static const struct bsp_led_rgb **led_green** = {.r = 0, .g = 255, .b = 0}

    Settings for LED with GREEN color.
- static const struct bsp_led_rgb **led_yellow** = {.r = 255, .g = 255, .b = 0}

    Settings for LED with YELLOW color.
- static const struct bsp_led_rgb **led_magenta** = {.r = 100, .g = 0, .b = 50}

    Settings for LED with MAGENTA color.
- static const struct bsp_led_pwm **blink_rare_on** = {.width_on_ms = 150, .width_off_ms = 1000}

    Settings to LED blinking with short enabled phase.
- static const struct bsp_led_pwm **blink_fast** = {.width_on_ms = 250, .width_off_ms = 250}

    Settings to LED fastly blinking with equaled enabled & disabled phases.
- static const struct bsp_led_pwm **blink_rare_off** = {.width_on_ms = 1000, .width_off_ms = 150}

    Settings to LED blinking with short disabled phase.

4.10.1 Detailed Description

Module of application layer of RGB LED.

4.10.2 Macro Definition Documentation

4.10.2.1 LED_EVENT_IS_VALID

```
#define LED_EVENT_IS_VALID(
            X ) (((uint32_t)(X)) < LED_EVENT_MAX)
```

MACRO RGB LED event is valid

The macro decides whether X is valid RGB LED event

Parameters

in	*X*	RGB LED event

Returns

true if `X` is valid RGB LED event false otherwise

4.10.3 Enumeration Type Documentation

4.10.3.1 led_event

```
enum led_event
```

RGB LED event (type of LED behaivour)

Enumerator

LED_EVENT_NONE	No events.
LED_EVENT_COMMON_ERROR	Unspecified error.
LED_EVENT_CRC_ERROR	Failed to initialize BSP CRC module.
LED_EVENT_FLASH_ERROR	Error in Configuration module.
LED_EVENT_LCD1602_ERROR	Failed communication with BSP LCD1602.
LED_EVENT_IN_PROCESS	Algorithm of Sniffer RS-232 is in process
LED_EVENT_SUCCESS	The firmware is in monitoring state with no UART errors.
LED_EVENT_FAILED	Algorithm of Sniffer RS-232 is failed
LED_EVENT_UART_ERROR	Error in module BSP UART during monitoring.
LED_EVENT_UART_OVERFLOW	Overflow in UART reception.
LED_EVENT_MAX	Count of types of RGB LED behaivours.

4.10.4 Function Documentation

4.10.4.1 app_led_deinit()

```
uint8_t app_led_deinit (
            void  )
```

Deinitialization of application layer of RGB LED

Returns

RES_OK on success error otherwise

4.10.4.2 app_led_init()

```
uint8_t app_led_init (
            void  )
```

Initialization of application layer of RGB LED

Returns

RES_OK on success error otherwise

4.10.4.3 app_led_set()

```
uint8_t app_led_set (
            enum led_event led_event )
```

Set RGB LED behaivour

Parameters

in	*led_event*	type of LED behaivour

Returns

RES_OK on success error otherwise

4.11 Basic interrupts

Handlers for basic MPU interrupts.

Functions

- void NMI_Handler (void)
- void HardFault_Handler (void)
- void MemManage_Handler (void)
- void BusFault_Handler (void)
- void UsageFault_Handler (void)
- void SVC_Handler (void)
- void DebugMon_Handler (void)
- void PendSV_Handler (void)
- void SysTick_Handler (void)

4.11.1 Detailed Description

Handlers for basic MPU interrupts.

4.11.2 Function Documentation

4.11.2.1 BusFault_Handler()

```
void BusFault_Handler (
             void  )
```

BusFault IRQ handler

The handler uses BSP_LED_RGB_HARDFAULT as firmware dead end

4.11.2.2 DebugMon_Handler()

```
void DebugMon_Handler (
             void  )
```

DebugMon IRQ handler

NOT used

4.11.2.3 HardFault_Handler()

```
void HardFault_Handler (
             void  )
```

Hardfault handler

The handler uses BSP_LED_RGB_HARDFAULT as firmware dead end

4.11.2.4 MemManage_Handler()

```
void MemManage_Handler (
             void  )
```

MemManage IRQ handler

The handler uses BSP_LED_RGB_HARDFAULT as firmware dead end

4.11.2.5 NMI_Handler()

```
void NMI_Handler (
             void  )
```

NMI IRQ handler

The handler uses BSP_LED_RGB_HARDFAULT as firmware dead end

4.11.2.6 PendSV_Handler()

```
void PendSV_Handler (
             void  )
```

PendSV IRQ handler

NOT used

4.11.2.7 SVC_Handler()

```
void SVC_Handler (
             void  )
```

SVC IRQ handler

NOT used

4.11.2.8 SysTick_Handler()

```
void SysTick_Handler (
             void  )
```

Systick IRQ handler

The handler makes count of HAL tick counter

4.11.2.9 UsageFault_Handler()

```
void UsageFault_Handler (
             void  )
```

UsageFault IRQ handler

The handler uses BSP_LED_RGB_HARDFAULT as firmware dead end

4.12 CLI

Command line interface.

Macros

- #define **UART_TRACE_BUFF_SIZE** (256)

  Size of string buffer used in cli_trace.
- #define **UART_RX_BUFF_SIZE** (256)

  Size of UART receive buffer for CLI BSP UART.
- #define **UART_TX_BUFF_SIZE** (6 ∗ UART_RX_BUFF_SIZE)

  Size of UART send buffer for CLI BSP UART.
- #define **TX_COLOR** MENU_COLOR_GREEN

  Color of traced RS-232 TX data.
- #define **RX_COLOR** MENU_COLOR_MAGENTA

  Color of traced RS-232 RX data.

Functions

- uint8_t cli_init (void)
- uint8_t cli_menu_start (struct flash_config ∗config)
- uint8_t cli_menu_exit (void)
- bool cli_menu_is_started (void)
- void cli_trace (const char ∗format,...)
- uint8_t cli_rs232_trace (enum uart_type uart_type, enum rs232_trace_type trace_type, uint8_t ∗data, uint32_t len, bool break_line)
- uint8_t cli_welcome (const char ∗welcome, uint8_t wait_time_s, bool ∗forced_exit, bool ∗is_pressed)
- void cli_terminal_reset (void)
- static uint8_t __cli_menu_entry (char ∗input, void ∗param)
- static uint8_t __cli_menu_set_defaults (char ∗input, void ∗param)
- static uint8_t __cli_menu_exit (char ∗input, void ∗param)
- static uint8_t __cli_menu_cfg_set (char ∗input, void ∗param)
- static char ∗ __cli_prompt_generator (const char ∗menu_item_label)
- static uint8_t __cli_menu_cfg_values_set (struct flash_config ∗config)
- static void __cli_uart_overflow_cb (enum uart_type type, void ∗params)
- static void __cli_uart_error_cb (enum uart_type type, uint32_t error, void ∗params)
- static uint8_t __cli_menu_write_cb (char ∗data)
- static uint8_t __cli_menu_read_cb (char ∗∗data)

Variables

- struct {
  bool **uart_error**
  Flag whether UART errors on CLI occured.
  bool **uart_overflow**
  Flag whether UART receive buffer is overflown.
  } **cli_state** = {0}

  State of UART CLI.
- static struct flash_config **old_config**

  Copy of input configuration.
- static struct flash_config ∗ **flash_config** = NULL

  Current configuration.
- static bool **is_config_changed** = false

  Flag whether configuration is changed.
- static struct menu_color_config **color_config_select** = MENU_COLOR_CONFIG_DEFAULT()

  Menu color settings for menus wihtout emphasised choice "yes-no".
- static struct menu_color_config color_config_choose

  Menu color settings for menus with emphasised choice "yes-no".
- static const char ∗ rs232_trace_type_str [ ]

  Array of string aliases for rs232_trace_type for output purposes.
- static const char ∗ rs232_interspace_type_str [ ]

  Array of string aliases for rs232_interspace_type for output purposes.
- static const char ∗ uart_parity_str [ ]

  Array of string aliases for uart_parity for output purposes.
- static const char ∗ rs232_channel_type_str [ ]

  Array of string aliases for rs232_channel_type for output purposes.

- struct {
  char ∗ **label**
  Label of menu.
  struct menu_color_config ∗ **color_config**
  Color settings of menu.
  } init_menus [ ]

  List of menus included in configuration menu.
- struct {
  char ∗ **menu_label**
  Label of menu which menu item belongs to.
  char ∗ **menu_item_label**
  Label of menu item.
  char ∗ **value_border**
  Border for value of menu item.
  uint8_t(∗ **callback**)(char ∗input, void ∗param)
  User callback by actions on menu item.
  char ∗ **menu_entry_label**
  Label of menu to which user can enter from menu item.
  } **init_menu_items** [ ]

  Structure of all menu items included in configuration menu.
- static uint8_t ∗ **__menu_rx_buff** = NULL

  Receive buffer for CLI BSP UART.

4.12.1 Detailed Description

Command line interface.

4.12.2 Function Documentation

4.12.2.1 __cli_menu_cfg_set()

```
static uint8_t __cli_menu_cfg_set (
            char ∗ input,
            void ∗ param )  [static]
```

Configuration set by menu action

The callback executes set of configuration according to chosen menu item and changed value of menu item and update appropriate menu items

Parameters

in	*input*	NOT used
in	*param*	NOT used

Returns

RES_OK on success error otherwise

4.12.2.2 __cli_menu_cfg_values_set()

```
static uint8_t __cli_menu_cfg_values_set (
             struct flash_config * config )  [static]
```

Set values of menu items

The function sets values for all menu items within configuration menu according to data from `config`

Parameters

in	*config*	configuration

Returns

RES_OK on success error otherwise

4.12.2.3 __cli_menu_entry()

```
static uint8_t __cli_menu_entry (
             char * input,
             void * param )  [static]
```

Menu entry

Parameters

in	*input*	NOT used
in	*param*	NOT used

Returns

RES_OK on success error otherwise

4.12.2.4 __cli_menu_exit()

```
static uint8_t __cli_menu_exit (
             char * input,
             void * param )  [static]
```

Menu exit

The callback for menu item "Configuration->Start" executes exit from menu

Parameters

in	*input*	NOT used
in	*param*	NOT used

Returns

RES_OK on success error otherwise

4.12.2.5 __cli_menu_read_cb()

```
static uint8_t __cli_menu_read_cb (
             char ** data )  [static]
```

Callback to read from console

The callback is used by Menu library to read data from console

Parameters

out	*data*	read data

Returns

RES_OK on success error otherwise

4.12.2.6 __cli_menu_set_defaults()

```
static uint8_t __cli_menu_set_defaults (
             char * input,
             void * param )  [static]
```

Set configuration to defaults

The callback for menu item "Algorithm->Defaults" resets configuration flash_config

Parameters

in	*input*	NOT used
in	*param*	NOT used

Returns

RES_OK on success error otherwise

4.12.2.7 __cli_menu_write_cb()

```
static uint8_t __cli_menu_write_cb (
            char * data )  [static]
```

Callback to write into console

The callback is used by Menu library to write data into console

Parameters

in	*data*	written data

Returns

RES_OK on success error otherwise

4.12.2.8 __cli_prompt_generator()

```
static char * __cli_prompt_generator (
            const char * menu_item_label )  [static]
```

Prompt generator

The function generates prompt by label of menu item

Parameters

in	*menu_item_label*	label of menu item

Returns

prompt of menu item, NULL if menu item does NOT have a prompt

4.12.2.9 __cli_uart_error_cb()

```
static void __cli_uart_error_cb (
            enum uart_type type,
            uint32_t error,
            void * params )  [static]
```

Callback for CLI UART errors

Callback is called from BSP UART when UART errors are occured on CLI
BSP UART of CLI will be restarted

Parameters

in	*type*	UART type, should be BSP_UART_TYPE_CLI
in	*error*	mask of occured UART errors
in	*params*	optional parameters

4.12.2.10 __cli_uart_overflow_cb()

```
static void __cli_uart_overflow_cb (
            enum uart_type type,
            void * params )  [static]
```

Callback for CLI UART overflow

Callback is called from BSP UART when overflow of RX buffer is occured

Parameters

in	*type*	UART type, should be BSP_UART_TYPE_CLI
in	*params*	optional parameters

4.12.2.11 cli_init()

```
uint8_t cli_init (
            void  )
```

CLI initialization

Returns

RES_OK on success error otherwise

4.12.2.12 cli_menu_exit()

```
uint8_t cli_menu_exit (
            void )
```

Configuration menu exit

Returns

RES_OK on success error otherwise

4.12.2.13 cli_menu_is_started()

```
bool cli_menu_is_started (
            void )
```

Check whether configuration menu is started

Returns

true if menu started false otherwise

4.12.2.14 cli_menu_start()

```
uint8_t cli_menu_start (
            struct flash_config * config )
```

Menu configuration start

Parameters

in,out	*config*	device configuration

Returns

RES_OK on success error otherwise

4.12.2.15 cli_rs232_trace()

```
uint8_t cli_rs232_trace (
            enum uart_type uart_type,
            enum rs232_trace_type trace_type,
            uint8_t * data,
            uint32_t len,
            bool break_line )
```

Trace of monitored RS-232 data

The function makes output of monitored RS-232 data into CLI

Parameters

in	*uart_type*	channel type of traced `data`, should be BSP_UART_TYPE_RS232_TX or BSP_UART_TYPE_RS232_RX
in	*trace_type*	trace type
in	*data*	traced data
in	*len*	length of traced data
in	*break_line*	flag whether symbol of LIN break should be traced first before `data`

Returns

RES_OK on success error otherwise

4.12.2.16 cli_terminal_reset()

```
void cli_terminal_reset (
            void )
```

Reset settings of console

The function resets settings of console to defaults via escape sequences

4.12.2.17 cli_trace()

```
void cli_trace (
            const char * format,
             ... )
```

Trace into CLI

Parameters

in	*format*	formatted string
in	...	variable argument list for formatting `format`

4.12.2.18 cli_welcome()

```
uint8_t cli_welcome (
            const char * welcome,
            uint8_t wait_time_s,
            bool * forced_exit,
            bool * is_pressed )
```

Welcome routine

The function performs welcome routine by the following scheme:

1. Trace welcome message set by `welcome`
2. Wait for any key pressing or external action (by `forced_exit`) within `wait_time_s` seconds
3. If key pressing took place the function is terminated with `is_pressed` = true
4. If timeout or external action occured the function is terminated with `is_pressed` = false

Parameters

in	*welcome*	welcome message
in	*wait_time←_s*	timeout in seconds
in	*forced_exit*	flag of occured external action
out	*is_pressed*	flag whether key pressing occured

Returns

RES_OK on success error otherwise

4.12.3 Variable Documentation

4.12.3.1 color_config_choose

```
struct menu_color_config color_config_choose [static]
```

Initial value:

```
= {.active = {.foreground = MENU_COLOR_YELLOW, .background = MENU_COLOR_RED},
                                                  .inactive = {.foreground = MENU_COLOR_WHITE,
     .background = MENU_COLOR_BLUE}}
```

Menu color settings for menus with emphasised choice "yes-no".

4.12.3.2

```
const struct { ... } init_menus[] [static]
```

Initial value:

```
= {
    {"MAIN MENU",            &color_config_select},
    {"CONFIGURATION",        &color_config_select},
    {"SAVE TO PRESETTINGS", &color_config_choose},
    {"PRESETTINGS",          &color_config_select},
    {"SAVE CONFIGURATION",  &color_config_choose},
    {"ALGORITHM",            &color_config_select},
    {"CHANNEL TYPE",         &color_config_select},
    {"LIN DETECTION",        &color_config_choose},
    {"RESET TO DEFAULTS",   &color_config_choose},
    {"TRACE TYPE",           &color_config_select},
    {"IDLE PRESENCE",        &color_config_select},
    {"TX/RX DELIMITER",      &color_config_select},
    {"LIN PROTOCOL",         &color_config_choose},
    {"WORD LENGTH",          &color_config_select},
    {"PARITY",               &color_config_select},
    {"STOP BITS",            &color_config_select},
    {"PRESETTINGS ENABLE",  &color_config_choose},
}
```

List of menus included in configuration menu.

4.12.3.3 rs232_channel_type_str

```
const char* rs232_channel_type_str[] [static]
```

Initial value:

```
= {
    "TX",
    "RX",
    "ANY",
    "ALL"
}
```

Array of string aliases for rs232_channel_type for output purposes.

4.12.3.4 rs232_interspace_type_str

```
const char* rs232_interspace_type_str[] [static]
```

Initial value:

```
= {
    "NONE",
    "SPACE",
    "NEW LINE",
    "INVALID"
}
```

Array of string aliases for rs232_interspace_type for output purposes.

4.12.3.5 rs232_trace_type_str

```
const char* rs232_trace_type_str[ ]  [static]
```

Initial value:

```
= {
    "HEX",
    "HEX/ASCII",
    "INVALID"
}
```

Array of string aliases for rs232_trace_type for output purposes.

4.12.3.6 uart_parity_str

```
const char* uart_parity_str[ ]  [static]
```

Initial value:

```
= {
    "NONE",
    "EVEN",
    "ODD",
    "INVALID"
}
```

Array of string aliases for uart_parity for output purposes.

4.13 Configuration

Module of firmware configuration stored in internal flash.

Data Structures

- struct uart_presettings

  UART presettings.
- struct flash_config

  Firmware configuration.

Macros

- #define RS232_TRACE_TYPE_VALID(X) ((X) < RS232_TRACE_MAX)
- #define RS232_INTERSPACE_TYPE_VALID(X) ((X) < RS232_INTERSPCACE_MAX)
- #define UART_PRESETTINGS_DEFAULT()
- #define FLASH_CONFIG_DEFAULT()
- #define **FLASH_SECTOR_CFG_ADDR** (0x08060000)

  Address of internal flash where configuration is stored.

Enumerations

- enum rs232_trace_type { RS232_TRACE_HEX = 0 , RS232_TRACE_HYBRID , RS232_TRACE_MAX }

  Trace type of RS-232 data.
- enum rs232_interspace_type { RS232_INTERSPCACE_NONE = 0 , RS232_INTERSPCACE_SPACE , RS232_INTERSPCACE_NEW_LINE , RS232_INTERSPCACE_MAX }

  Type of interspaces between RS-232 data.

Functions

- uint8_t config_save (struct flash_config ∗config)
- uint8_t config_read (struct flash_config ∗config)

4.13.1 Detailed Description

Module of firmware configuration stored in internal flash.

4.13.2 Macro Definition Documentation

4.13.2.1 FLASH_CONFIG_DEFAULT

```
#define FLASH_CONFIG_DEFAULT( )
```

Value:
```
    {\
    .alg_config = SNIFFER_RS232_CONFIG_DEFAULT(),\
    .presettings = UART_PRESETTINGS_DEFAULT(),\
    .trace_type = RS232_TRACE_HEX,\
    .idle_presence = RS232_INTERSPCACE_NONE,\
    .txrx_delimiter = RS232_INTERSPCACE_NONE,\
    .save_to_presettings = true\
}
```

MACRO Default configuration

Returns

initializer for flash_config

4.13.2.2 RS232_INTERSPACE_TYPE_VALID

```
#define RS232_INTERSPACE_TYPE_VALID(
                X ) ((X) < RS232_INTERSPCACE_MAX)
```

MACRO RS-S232 interspace type is valid

The macro decides whether `X` is valid RS-232 interspace type

Parameters

in	*X*	RS-232 interspace type

Returns

true if `X` is valid RS-232 interspace type false otherwise

4.13.2.3 RS232_TRACE_TYPE_VALID

```
#define RS232_TRACE_TYPE_VALID(
                X ) ((X) < RS232_TRACE_MAX)
```

MACRO RS-S232 trace type is valid

The macro decides whether `X` is valid RS-232 trace type

Parameters

in	*X*	RS-232 trace type

Returns

true if `X` is valid RS-232 trace type false otherwise

4.13.2.4 UART_PRESETTINGS_DEFAULT

```
#define UART_PRESETTINGS_DEFAULT( )
```

Value:

```
    {\
    .enable = false,\
    .parity = BSP_UART_PARITY_NONE,\
    .baudrate = 0,\
    .stopbits = BSP_UART_STOPBITS_1,\
    .wordlen = BSP_UART_WORDLEN_8,\
    .lin_enabled = false\
}
```

MACRO Default UART presettings

Returns

initializer for uart_presettings

4.13.3 Enumeration Type Documentation

4.13.3.1 rs232_interspace_type

```
enum rs232_interspace_type
```

Type of interspaces between RS-232 data.

Enumerator

RS232_INTERSPCACE_NONE	No interspaces.
RS232_INTERSPCACE_SPACE	Space between RS-232 data.
RS232_INTERSPCACE_NEW_LINE	New line symbol between RS-232 data.
RS232_INTERSPCACE_MAX	Count of interspace types.

4.13.3.2 rs232_trace_type

```
enum rs232_trace_type
```

Trace type of RS-232 data.

Enumerator

RS232_TRACE_HEX	Data is traced in HEX format.
RS232_TRACE_HYBRID	Data is traced as char if printable and as HEX if not.
RS232_TRACE_MAX	Count of trace types.

4.13.4 Function Documentation

4.13.4.1 config_read()

```
uint8_t config_read (
            struct flash_config * config )
```

Read configuration

The function reads configuration from flash

Parameters

out	*config*	read configuration

Returns

RES_OK on success error otherwise

4.13.4.2 config_save()

```
uint8_t config_save (
            struct flash_config * config )
```

Save configuration

The function saves configuration into flash

Parameters

in	*config*	saved configuration

Returns

RES_OK on success error otherwise

4.14 Application

Application layer of the firmware.

Modules

- Application layer of RGB LED

  Module of application layer of RGB LED.
- Basic interrupts

  Handlers for basic MPU interrupts.
- CLI

  Command line interface.
- Configuration

  Module of firmware configuration stored in internal flash.
- Main

  Firmware main routine.
- Menu library

  Library for console menu.
- Algorithm of Sniffer RS-232

  Module of recognizing algorithm of Sniffer RS-232.

4.14.1 Detailed Description

Application layer of the firmware.

4.15 Main

Firmware main routine.

Macros

- #define **APP_VERSION** "1.0-RC3"

    Firmware version.
- #define **UART_RX_BUFF** (256)

    Size of RX buffer to store data received from BSP UART.
- #define IS_UART_ERROR(X) (uart_flags[X].error || uart_flags[X].overflow)

Functions

- static void uart_lin_break_cb (enum uart_type type, void ∗params)
- static void uart_overflow_cb (enum uart_type type, void ∗params)
- static void uart_error_cb (enum uart_type type, uint32_t error, void ∗params)
- static void button_cb (enum button_action action)
- static bool button_wait_event (uint32_t tmt)
- static void internal_error (enum led_event led_event)
- int main ()

Variables

- static const char **uart_parity_sym** [ ] = {'N', 'E', 'O'}

    Array of char aliases for uart_parity for output purposes.
- static const char ∗ **display_uart_type_str** [ ] = {"CLI", "TX", "RX"}

    Array of string aliases for uart_type for output purposes.
- static bool **press_event** = false

    Flag whether press event on the button is occured.
-

  struct {
    uint32_t **error**
      Mask of UART errors.
    bool **overflow**
      Flag whether UART RX buffer is overflown before call bsp_uart_read.
    bool **lin_break**
      Flag whether LIN break detection is occured.
  } **uart_flags** [BSP_UART_TYPE_MAX] = {0}

    UART flags.

4.15.1 Detailed Description

Firmware main routine.

4.15.2 Macro Definition Documentation

4.15.2.1 IS_UART_ERROR

```
#define IS_UART_ERROR(
            X ) (uart_flags[X].error || uart_flags[X].overflow)
```

MACRO Flag whether UART errors occured

Parameters

`in`	*X*	type of UART, see uart_type

Returns

true if some errors took place, false otherwise

4.15.3 Function Documentation

4.15.3.1 button_cb()

```
static void button_cb (
             enum button_action action )  [static]
```

Callback for button actions

Callback is called from BSP button when actions on
the button are occured. Algorithm of the callback is the following:

1. Action BUTTON_PRESSED occured:
   If menu is started or waiting to be started - skip menu start
   otherwise - start/stop toggle of UART reception from RS-232 channels
2. Action BUTTON_LONG_PRESSED
   Software reset of the chip in any cases

Parameters

`in`	*action*	BUTTON action, see button_action

4.15.3.2 button_wait_event()

```
static bool button_wait_event (
             uint32_t tmt )  [static]
```

Wait for press event

The function waits when BUTTON_PRESSED occured, using press_event
press_event is cleared if was set

Parameters

`in`	*tmt*	timeout in ms for event waiting

Returns

true if event occured, false otherwise

4.15.3.3 internal_error()

```
static void internal_error (
            enum led_event led_event )  [static]
```

Routine for internal error

The function calls when occured errors on the firmware
do not let it working further. The function uses politics of LED signaling

Warning

The function is firmware dead end and reset is needed to start firmware

Parameters

in	*led_event*	politics of LED signaling

4.15.3.4 main()

```
int main ( )
```

Main routine of the firmware

Returns

NOT used

4.15.3.5 uart_error_cb()

```
static void uart_error_cb (
            enum uart_type type,
            uint32_t error,
            void * params )  [static]
```

Callback for UART errors

Callback is called from BSP UART when UART errors are occured

Parameters

in	*type*	UART type
in	*error*	mask of occured UART errors
in	*params*	optional parameters

4.15.3.6 uart_lin_break_cb()

```
static void uart_lin_break_cb (
             enum uart_type type,
             void * params )  [static]
```

Callback for UART LIN break detection

Callback is called from BSP UART when LIN break is detected

Parameters

in	*type*	UART type
in	*params*	optional parameters

4.15.3.7 uart_overflow_cb()

```
static void uart_overflow_cb (
             enum uart_type type,
             void * params )  [static]
```

Callback for UART overflow

Callback is called from BSP UART when overflow of RX buffer is occured

Parameters

in	*type*	UART type
in	*params*	optional parameters

4.16 Menu library

Library for console menu.

Data Structures

- struct menu_color

Menu color data.

- struct menu_color_config

  Menu color settings.
- struct menu_item

  Menu item context.
- struct menu_config

  Menu library settings.

Macros

- #define **MENU_MAX_STR_LEN** 256

  Maximum valid length of strings used within menu library.
- #define **MENU_COLOR_RESET** "\33[0;37;40m"

  Escape sequence to reset console colors.
- #define **MENU_RETURN_HOME** "\33[H"

  Escape sequence to return cursor to left top corner of console.
- #define **MENU_LINE_UP** "\33[A"

  Escape sequence to move cursor one line up.
- #define **MENU_LINE_DOWN** "\33[B"

  Escape sequence to move cursor one line down.
- #define **MENU_LINE_ERASE** "\33[2K"

  Escape sequence to erase current line.
- #define **MENU_SCREEN_ERASE** "\33[2J"

  Escape sequence to erase screen of console.
- #define MENU_COLOR_CONFIG_DEFAULT()
- #define **MENU_COLOR_SIZE** 10

  Length of escape sequence for colors.
- #define MENU_PASS_TYPE_IS_VALID(X) (((uint32_t)(X)) < MENU_PASS_MAX)
- #define MENU_NUM_TYPE_IS_VALID(X) (((uint32_t)(X)) < MENU_NUM_MAX)

Enumerations

- enum menu_color_type {
  MENU_COLOR_BLACK = 0 , MENU_COLOR_RED , MENU_COLOR_GREEN , MENU_COLOR_YELLOW
  ,
  MENU_COLOR_BLUE , MENU_COLOR_MAGENTA , MENU_COLOR_CYAN , MENU_COLOR_WHITE ,
  MENU_COLOR_MAX }

  Menu colors.
- enum menu_pass_type { MENU_PASS_NONE = 0 , MENU_PASS_WITH_PROMPT , MENU_PASS_ALWAYS
  , MENU_PASS_MAX }

  Type of passing input to menu_item::callback.
- enum menu_num_type {
  MENU_NUM_NONE = 0 , MENU_NUM_DIGITAL , MENU_NUM_UPPER_LETTER , MENU_NUM_LOWER_LETTER
  ,
  MENU_NUM_MAX }

  Numbering types.

Functions

- void menu_all_destroy (void)
- struct menu ∗ menu_create (char ∗label, char filler, struct menu_color_config ∗color_config)
- uint8_t menu_entry (struct menu ∗menu)
- uint8_t menu_item_value_set (struct menu_item ∗menu_item, const char ∗value)
- struct menu_item ∗ menu_current_item_get (void)
- char ∗ menu_item_label_get (struct menu_item ∗menu_item)
- struct menu ∗ menu_by_label_get (const char ∗label)
- struct menu_item ∗ menu_item_by_label_get (struct menu ∗menu, const char ∗label)
- struct menu_item ∗ menu_item_by_label_only_get (const char ∗label)
- bool menu_is_started (void)
- uint8_t menu_start (struct menu_config ∗config, struct menu ∗menu)
- uint8_t menu_exit (void)
- struct menu_item ∗ menu_item_add (struct menu ∗menu, const char ∗label, const char ∗prompt, const char ∗value_border, uint8_t(∗callback)(char ∗input, void ∗param), void ∗param, struct menu ∗menu_entry)
- static uint32_t __menu_strlen (const char ∗str)
- static struct menu_item ∗ __menu_get_last_item (void)
- static bool __menu_item_is_in_menu (struct menu ∗menu, struct menu_item ∗menu_item)
- static uint8_t __menu_enumerator_inc (enum menu_num_type num_type, char ∗enumerator, uint8_t enum←_len)
- static uint8_t __menu_redraw (struct menu_item ∗prev_item_active, struct menu_item ∗new_item_active)

Variables

- static struct menu_config **menu_config** = {0}

  Local copy of menu settings.
- static struct menu_item ∗ **cur_item** = NULL

  Current menu item from cur_menu.
- static struct menu_item ∗ **prev_item** = NULL

  Previous menu item.
- static struct menu ∗ **cur_menu** = NULL

  Current menu from menu_list.
- struct menu ∗ **menu_list** = NULL

  Menu list.
- static bool **__exit** = true

  Flag whether console menu is finished.

4.16.1 Detailed Description

Library for console menu.

4.16.2 Macro Definition Documentation

4.16.2.1 MENU_COLOR_CONFIG_DEFAULT

```
#define MENU_COLOR_CONFIG_DEFAULT( )
```

Value:

```
    {\
    .active = {.foreground = MENU_COLOR_BLUE, .background = MENU_COLOR_WHITE},\
    .inactive = {.foreground = MENU_COLOR_WHITE, .background = MENU_COLOR_BLUE}\
}
```

MACRO Default Menu color settings

Returns

initializer for menu_color_config

4.16.2.2 MENU_NUM_TYPE_IS_VALID

```
#define MENU_NUM_TYPE_IS_VALID(
                X ) (((uint32_t)(X)) < MENU_NUM_MAX)
```

MACRO Numbering type is valid

The macro decides whether `X` is valid numbering type,

See also

menu_num_type

Parameters

in	*X*	numbering type

Returns

true if `X` is valid numbering type false otherwise

4.16.2.3 MENU_PASS_TYPE_IS_VALID

```
#define MENU_PASS_TYPE_IS_VALID(
                X ) (((uint32_t)(X)) < MENU_PASS_MAX)
```

MACRO Passing type is valid

The macro decides whether `X` is valid passing type,

See also

menu_pass_type

Parameters

in	*X*	passing type

Returns

true if `X` is valid passing type false otherwise

4.16.3 Enumeration Type Documentation

4.16.3.1 menu_color_type

`enum menu_color_type`

Menu colors.

Enumerator

MENU_COLOR_BLACK	Black.
MENU_COLOR_RED	Red.
MENU_COLOR_GREEN	Green.
MENU_COLOR_YELLOW	Yellow.
MENU_COLOR_BLUE	Blue.
MENU_COLOR_MAGENTA	Magenta.
MENU_COLOR_CYAN	Cyan.
MENU_COLOR_WHITE	White.
MENU_COLOR_MAX	Count of menu colors.

4.16.3.2 menu_num_type

`enum menu_num_type`

Numbering types.

Enumerator

MENU_NUM_NONE	List of menu items is not numbered.
MENU_NUM_DIGITAL	List of menu items is numbered by numbers.
MENU_NUM_UPPER_LETTER	List of menu items is numbered by letters A..Z.
MENU_NUM_LOWER_LETTER	List of menu items is numbered by letters a..z.
MENU_NUM_MAX	Count of numbering types.

4.16.3.3 menu_pass_type

```
enum menu_pass_type
```

Type of passing input to menu_item::callback.

Enumerator

MENU_PASS_NONE	No passed input.
MENU_PASS_WITH_PROMPT	Input passed only if current menu item has prompt.
MENU_PASS_ALWAYS	Input always passed.
MENU_PASS_MAX	Count of passing types.

4.16.4 Function Documentation

4.16.4.1 __menu_enumerator_inc()

```
static uint8_t __menu_enumerator_inc (
             enum menu_num_type num_type,
             char * enumerator,
             uint8_t enum_len )  [static]
```

Enumerator increment

The function increments enumerator for numbered list of menu items

Parameters

in	*num_type*	numbering type
in,out	*enumerator*	enumerator as string
in	*enum_len*	maximum length of `enumerator`

Returns

RES_OK on success error otherwise

4.16.4.2 __menu_get_last_item()

```
static struct menu_item * __menu_get_last_item (
             void )  [static]
```

Get last menu item of current menu

Returns

last menu item from cur_menu on succes NULL otherwise

4.16.4.3 __menu_item_is_in_menu()

```
static bool __menu_item_is_in_menu (
            struct menu * menu,
            struct menu_item * menu_item )  [static]
```

Check if menu item belongs menu

Parameters

in	*menu*	menu
in	*menu_item*	menu item

Returns

true if `menu_item` belongs to `menu` false otherwise

4.16.4.4 __menu_redraw()

```
static uint8_t __menu_redraw (
            struct menu_item * prev_item_active,
            struct menu_item * new_item_active )  [static]
```

Redraw menu

The function draws menu in console.
The function uses position of previous and current menu items
to optimize redrawing being within 1 or 2 lines so here are few modes:

1. Full redraw - both `prev_item_active` & `new_item_active` are NULL
2. Redraw for new selected menu item - both `prev_item_active` & `new_item_active` are not NULL
3. Redraw content of single line - only one from `prev_item_active` & `new_item_active` is NULL

Parameters

in	*prev_item_active*	previous menu item
in	*new_item_active*	current menu item

Returns

RES_OK on success error otherwise

4.16.4.5 __menu_strlen()

```
static uint32_t __menu_strlen (
             const char * str )  [static]
```

String length

It is the wrapper over strnlen() guaranteeing return of string length
less than MENU_MAX_STR_LEN

Parameters

in	*str*	string

Returns

length of string

4.16.4.6 menu_all_destroy()

```
void menu_all_destroy (
             void  )
```

Free all allocated memory

The function frees all allocated memory within menu library

4.16.4.7 menu_by_label_get()

```
struct menu * menu_by_label_get (
             const char * label )
```

Get menu by label

Parameters

in	*label*	label of menu

Returns

menu on success NULL otherwise

4.16.4.8 menu_create()

```
struct menu * menu_create (
             char * label,
             char filler,
             struct menu_color_config * color_config )
```

Create menu

The function creates new menu, adds it to menu_list

Parameters

in	*label*	label of menu
in	*filler*	filler for label of menu to fill rest part of menu_config::width
in	*color_config*	color settings of new menu

Returns

new menu on success NULL otherwise

4.16.4.9 menu_current_item_get()

```
struct menu_item * menu_current_item_get (
            void )
```

Get current menu item of current menu

Returns

current menu item equaled to cur_item

4.16.4.10 menu_entry()

```
uint8_t menu_entry (
            struct menu * menu )
```

Menu entry

The function executes entry to new menu (set as current one).

Note

If `menu` is NULL menu from menu_item::menu_entry is used

Parameters

in	*menu*	new menu

Returns

RES_OK on success error otherwise

4.16.4.11 menu_exit()

```
uint8_t menu_exit (
            void )
```

Exit menu library

The function closes console menu by using __exit

Returns

RES_OK on success error otherwise

4.16.4.12 menu_is_started()

```
bool menu_is_started (
            void )
```

Start status of menu library

Returns

true if menu library is started false otherwise

4.16.4.13 menu_item_add()

```
struct menu_item * menu_item_add (
            struct menu * menu,
            const char * label,
            const char * prompt,
            const char * value_border,
            uint8_t(*)(char *input, void *param) callback,
            void * param,
            struct menu * menu_entry )
```

Add new menu item

The function adds menu item to selected menu

Note

`value_border` is a string border of value of menu item
`value_border` is represented in the format "<string>" = "<left border><right border>"
if length of string is even so first half is menu_item::value_left_border and
second part is menu_item::value_right_border
if length equals to 1 then "<string>" = "<left border>" (no right border)
In all other cases `value_border` is incorrect

Parameters

in	*menu*	menu into which menu item is being added
in	*label*	label of menu item
in	*prompt*	prompt of menu item, NULL if no prompt
in	*value_border*	string border
in	*callback*	user callback by actions on menu item
in	*param*	optional parameters passed to `callback`
in	*menu_entry*	menu to which user can enter from menu item

Returns

menu item on success NULL otherwise

4.16.4.14 menu_item_by_label_get()

```
struct menu_item * menu_item_by_label_get (
            struct menu * menu,
            const char * label )
```

Get menu item from menu by label

Parameters

in	*menu*	menu from which menu item is got
in	*label*	label of menu item

Returns

menu item on success NULL otherwise

4.16.4.15 menu_item_by_label_only_get()

```
struct menu_item * menu_item_by_label_only_get (
            const char * label )
```

Get menu item within all menus by label

The function seeks menu item among all existed menus
by combined label "<Menu label>\<Menu item label>"

Parameters

in	*label*	combined label

Returns

menu item on success NULL otherwise

4.16.4.16 menu_item_label_get()

```
char * menu_item_label_get (
            struct menu_item * menu_item )
```

Get label of menu item

Parameters

in	*menu_item*	menu item

Returns

label of menu item on success NULL otherwise

4.16.4.17 menu_item_value_set()

```
uint8_t menu_item_value_set (
            struct menu_item * menu_item,
            const char * value )
```

Set value of menu item

Parameters

in	*menu_item*	menu item
in	*value*	set value

Returns

RES_OK on success error otherwise

4.16.4.18 menu_start()

```
uint8_t menu_start (
            struct menu_config * config,
            struct menu * menu )
```

Start menu library

Note

The function makes menu routine until __exit is set by one of the user callbacks menu_item::callback

Parameters

in	*config*	menu library settings
in	*menu*	start menu

Returns

RES_OK on success error otherwise

4.17 Algorithm of Sniffer RS-232

Module of recognizing algorithm of Sniffer RS-232.

Data Structures

- struct sniffer_rs232_config

    Algorithm settings.
- struct hyp_check_ctx
- struct baud_calc_ctx
- struct hyp_ctx

Macros

- #define RS232_CHANNEL_TYPE_VALID(TYPE) (((uint32_t)(TYPE)) < RS232_CHANNEL_MAX)
- #define SNIFFER_RS232_CFG_PARAM_MIN(X) sniffer_rs232_config_item_range((uint32_t)&((struct sniffer_rs232_config∗)0)->X, true)
- #define SNIFFER_RS232_CFG_PARAM_MAX(X) sniffer_rs232_config_item_range((uint32_t)&((struct sniffer_rs232_config∗)0)->X, false)
- #define SNIFFER_RS232_CFG_PARAM_IS_VALID(X, V) (((V) >= SNIFFER_RS232_CFG_PARAM_MIN(X)) && ((V) <= SNIFFER_RS232_CFG_PARAM_MAX(X)))
- #define SNIFFER_RS232_CONFIG_DEFAULT()
- #define **BUFFER_SIZE** (512)

    Size of buffers tx_buffer & rx_buffer.
- #define **UART_BUFF_SIZE** (128)

    Size of receive buffer used in BSP UART.
- #define LIN_BREAK_MIN_LEN (10)

Enumerations

- enum rs232_channel_type {
RS232_CHANNEL_TX = 0 , RS232_CHANNEL_RX , RS232_CHANNEL_ANY , RS232_CHANNEL_ALL ,
RS232_CHANNEL_MAX }

Functions

- uint8_t sniffer_rs232_init (struct sniffer_rs232_config ∗__config)
- uint8_t sniffer_rs232_deinit (void)
- uint8_t sniffer_rs232_calc (struct uart_init_ctx ∗uart_params)
- uint32_t sniffer_rs232_config_item_range (uint32_t shift, bool is_min)
- bool sniffer_rs232_config_check (struct sniffer_rs232_config ∗__config)
- static void __sniffer_rs232_tim_msp_init (TIM_HandleTypeDef ∗htim)
- static void __sniffer_rs232_tim_msp_deinit (TIM_HandleTypeDef ∗htim)
- static uint32_t __sniffer_rs232_baudrate_get (uint32_t len_bit)
- static uint8_t __sniffer_rs232_line_baudrate_calc_init (GPIO_TypeDef ∗gpiox, uint16_t pin, IRQn_Type irq←_type)
- static void __sniffer_rs232_line_baudrate_calc (struct baud_calc_ctx ∗ctx)
- static uint8_t __sniffer_rs232_baudrate_calc (enum rs232_channel_type channel_type, uint32_t ∗baudrate, bool ∗lin_detected)
- static void __sniffer_rs232_uart_overflow_cb (enum uart_type type, void ∗params)
- static void __sniffer_rs232_uart_error_cb (enum uart_type type, uint32_t error, void ∗params)
- static uint8_t __sniffer_rs232_params_calc (enum rs232_channel_type channel_type, uint32_t baudrate, int8_t ∗hyp_num)
- void EXTI3_IRQHandler (void)
- void EXTI9_5_IRQHandler (void)

Variables

- static TIM_HandleTypeDef alg_tim = {.Instance = TIM6}
- static EXTI_HandleTypeDef hexti1 = {.Line = EXTI_LINE_3}
- static EXTI_HandleTypeDef hexti2 = {.Line = EXTI_LINE_5}
- static uint32_t **tx_cnt** = 0

  Current filling level of tx_buffer.
- static uint32_t **rx_cnt** = 0

  Current filling level of rx_buffer.
- static uint32_t tx_buffer [BUFFER_SIZE] = {0}
- static uint32_t rx_buffer [BUFFER_SIZE] = {0}
- static const uint32_t baudrates_list [ ] = {921600, 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800, 2400}
- static const struct hyp_ctx hyp_seq [ ]

  Sequence of hypotheses regarding UART parameters of RS-232 channels.
- static struct sniffer_rs232_config **config**

  Local copy of algorithm settings.

4.17.1 Detailed Description

Module of recognizing algorithm of Sniffer RS-232.

Algorithm consists of two parts:

1. Baudrate part - when baudrate calculated in EXTI mode
2. Parameter part - when other UART parameters (word length, parity type) calculated in UART mode

4.17.2 Macro Definition Documentation

4.17.2.1 LIN_BREAK_MIN_LEN

```
#define LIN_BREAK_MIN_LEN (10)
```

Minimal ratio between maximum and minimum widths of lower level
on the RS-232 lines to make decision about LIN break existence

4.17.2.2 RS232_CHANNEL_TYPE_VALID

```
#define RS232_CHANNEL_TYPE_VALID(
              TYPE ) (((uint32_t)(TYPE)) < RS232_CHANNEL_MAX)
```

MACRO Check if RS-232 channel detection type is valid

The macro checks whether *TYPE* is valid RS-232 channel detection type

Parameters

in	*TYPE*	RS-232 channel detection type

Returns

true if valid false otherwise

4.17.2.3 SNIFFER_RS232_CFG_PARAM_IS_VALID

```
#define SNIFFER_RS232_CFG_PARAM_IS_VALID(
              X,
              V ) (((V) >= SNIFFER_RS232_CFG_PARAM_MIN(X)) && ((V) <= SNIFFER_RS232_CFG_PARAM_MAX(X)))
```

MACRO Check whether parameter is valid

The macro checks whether parameter from algorithm settings is valid

Parameters

in	*X*	parameter name
in	*V*	value of a parameter

Returns

true if a parameter is valid false otherwise

4.17.2.4 SNIFFER_RS232_CFG_PARAM_MAX

```
#define SNIFFER_RS232_CFG_PARAM_MAX(
             X ) sniffer_rs232_config_item_range((uint32_t)&((struct sniffer_rs232_config*)0)->X,
false)
```

MACRO Get maximum valid value of a parameter

The macro returns maximum valid value of a parameter
from algoritm settings sniffer_rs232_config
The macro is wrapper over sniffer_rs232_config_item_range

Parameters

in	*X*	parameter name

Returns

maximum valid value

4.17.2.5 SNIFFER_RS232_CFG_PARAM_MIN

```
#define SNIFFER_RS232_CFG_PARAM_MIN(
             X ) sniffer_rs232_config_item_range((uint32_t)&((struct sniffer_rs232_config*)0)->X,
true)
```

MACRO Get minimum valid value of a parameter

The macro returns minimum valid value of a parameter
from algoritm settings sniffer_rs232_config
The macro is wrapper over sniffer_rs232_config_item_range

Parameters

in	*X*	parameter name

Returns

minimum valid value

4.17.2.6 SNIFFER_RS232_CONFIG_DEFAULT

```
#define SNIFFER_RS232_CONFIG_DEFAULT( )
```

Value:

```
            {\
            .channel_type = RS232_CHANNEL_ANY,\
            .valid_packets_count = 20,\
            .uart_error_count = 2,\
            .baudrate_tolerance = 10,\
            .min_detect_bits = 48,\
            .exec_timeout = 600,\
            .calc_attempts = 3,\
            .lin_detection = false\
        }
```

MACRO Default algorithm settings

Returns

initializer for sniffer_rs232_config

4.17.3 Enumeration Type Documentation

4.17.3.1 rs232_channel_type

```
enum rs232_channel_type
```

RS-232 channel detection type

Enumerator

RS232_CHANNEL_TX	Algorithm works only on RS-232 TX.
RS232_CHANNEL_RX	Algorithm works only on RS-232 RX.
RS232_CHANNEL_ANY	Algorithm works until one of the RS-232 channels calculated successfully.
RS232_CHANNEL_ALL	Algorithm works on both RS-232 lines.
RS232_CHANNEL_MAX	Count of RS-232 channel detection types.

4.17.4 Function Documentation

4.17.4.1 __sniffer_rs232_baudrate_calc()

```
static uint8_t __sniffer_rs232_baudrate_calc (
            enum rs232_channel_type channel_type,
            uint32_t * baudrate,
            bool * lin_detected )  [static]
```

Baudrate part of the algorithm

The function calculates baudrate on RS-232 TX/RX lines according to *channel_type*

Parameters

in	*channel_type*	RS-232 channel detection type
out	*baudrate*	calculated baudrate
out	*lin_detected*	flag whether LIN protocol is detected

Returns

RES_OK on success error otherwise

4.17.4.2 __sniffer_rs232_baudrate_get()

```
static uint32_t __sniffer_rs232_baudrate_get (
            uint32_t len_bit )  [static]
```

Baudrate calculation by width of a bit

The function calculates whether width of a bit corresponds one of the baudrate from baudrates_list

Parameters

in	*len_bit*	width of a bit

Returns

baudrate value in bods on success, 0 otherwise

4.17.4.3 __sniffer_rs232_line_baudrate_calc()

```
static void __sniffer_rs232_line_baudrate_calc (
            struct baud_calc_ctx * ctx )  [static]
```

Baudrate calculation on the RS-232 line

The function calculates baudrate on one RS-232 line

Parameters

in,out	*ctx*	context of baudrate calculation

4.17.4.4 __sniffer_rs232_line_baudrate_calc_init()

```
static uint8_t __sniffer_rs232_line_baudrate_calc_init (
            GPIO_TypeDef * gpiox,
            uint16_t pin,
            IRQn_Type irq_type )  [static]
```

Initialization of baudrate part of the algorithm

The function makes MSP EXTI initialization and waits for IDLE
state on the appropriate RS-232 line

Parameters

in	*gpiox*	GPIO port of *pin* used as EXTI
in	*pin*	GPIO pin used as EXTI
in	*irq_type*	NVIC IRQ type

Returns

RES_OK on success error otherwise

4.17.4.5 __sniffer_rs232_params_calc()

```
static uint8_t __sniffer_rs232_params_calc (
            enum rs232_channel_type channel_type,
            uint32_t baudrate,
            int8_t * hyp_num )  [static]
```

Parameter part of the algorithm

The function calculates other parameters of UART on RS-232 lines

Parameters

in	*channel_type*	RS-232 channel detection type
in	*baudrate*	baudrate in bods on RS-232 lines
out	*hyp_num*	number of approved hypothesis from hyp_seq on success, -1 otherwise

Returns

RES_OK on success error otherwise

4.17.4.6 __sniffer_rs232_tim_msp_deinit()

```
static void __sniffer_rs232_tim_msp_deinit (
            TIM_HandleTypeDef * htim )  [static]
```

STM32 HAL TIM MSP deinitialization

Parameters

in	*htim*	STM32 HAL TIM instance, should equal to alg_tim

4.17.4.7 __sniffer_rs232_tim_msp_init()

```
static void __sniffer_rs232_tim_msp_init (
             TIM_HandleTypeDef * htim )  [static]
```

STM32 HAL TIM MSP initialization

Parameters

in	*htim*	STM32 HAL TIM instance, should equal to alg_tim

4.17.4.8 __sniffer_rs232_uart_error_cb()

```
static void __sniffer_rs232_uart_error_cb (
             enum uart_type type,
             uint32_t error,
             void * params )  [static]
```

Callback for UART errors

Callback is called from BSP UART when UART errors are occured
Callback counts UART errors into check context of the current hypothesis hyp_check_ctx

Parameters

in	*type*	UART type
in	*error*	mask of occured UART errors
in	*params*	optional parameters, containing check context of the current hypothesis hyp_check_ctx

4.17.4.9 __sniffer_rs232_uart_overflow_cb()

```
static void __sniffer_rs232_uart_overflow_cb (
             enum uart_type type,
             void * params )  [static]
```

Callback for UART overflow

Callback is called from BSP UART when overflow of RX buffer is occured
If call occured the algorithm is terminated with fail

Parameters

in	*type*	UART type
in	*params*	optional parameters, containing check context of the current hypothesis hyp_check_ctx

4.17.4.10 EXTI3_IRQHandler()

```
void EXTI3_IRQHandler (
             void   )
```

NVIC IRQ EXTI3 handler

Handler is used to fill in tx_buffer

4.17.4.11 EXTI9_5_IRQHandler()

```
void EXTI9_5_IRQHandler (
             void   )
```

NVIC IRQ EXTI5 handler

Handler is used to fill in rx_buffer

4.17.4.12 sniffer_rs232_calc()

```
uint8_t sniffer_rs232_calc (
             struct uart_init_ctx * uart_params )
```

Algorithm calculation

The function executes the algorithm

Note

uart_init_ctx::baudrate is 0 if calculation failed
Despite of it the function returns RES_OK if all hypotheses have been tried

Parameters

out	*uart_params*	UART parameters of RS-232 lines

Returns

RES_OK on success error otherwise

4.17.4.13 sniffer_rs232_config_check()

```
bool sniffer_rs232_config_check (
            struct sniffer_rs232_config * __config )
```

Check algorithm settings

Parameters

in	*__config*	algorithm settings

Returns

true if settings are valid false otherwise

4.17.4.14 sniffer_rs232_config_item_range()

```
uint32_t sniffer_rs232_config_item_range (
            uint32_t shift,
            bool is_min )
```

Valid value range of items from algorithm settings

The function is used to validate settings for the algorithm

Parameters

in	*shift*	memory shift of an item over sniffer_rs232_config
in	*is_min*	flag indicating lower border of an range if true, upper border if false

Returns

value of a border of a range

4.17.4.15 sniffer_rs232_deinit()

```
uint8_t sniffer_rs232_deinit (
            void )
```

Algorithm deinitialization

Returns

RES_OK on success error otherwise

4.17.4.16 sniffer_rs232_init()

```
uint8_t sniffer_rs232_init (
            struct sniffer_rs232_config * __config )
```

Algorithm initialization

Parameters

in	*__config*	algorithm settings

Returns

RES_OK on success error otherwise

4.17.5 Variable Documentation

4.17.5.1 alg_tim

```
TIM_HandleTypeDef alg_tim = {.Instance = TIM6}  [static]
```

STM32 HAL TIM instance for timer used to count widths of lower level on the RS-232 lines

4.17.5.2 baudrates_list

```
const uint32_t baudrates_list[] = {921600, 460800, 230400, 115200, 57600, 38400, 19200, 9600,
4800, 2400}  [static]
```

List of baudrates which can be detected by the algorithm

4.17.5.3 hexti1

```
EXTI_HandleTypeDef hexti1 = {.Line = EXTI_LINE_3}  [static]
```

STM32 HAL EXTI instance used to detect falling & rising edges of signals on the RS-232 TX line

4.17.5.4 hexti2

```
EXTI_HandleTypeDef hexti2 = {.Line = EXTI_LINE_5}  [static]
```

STM32 HAL EXTI instance used to detect falling & rising edges of signals on the RS-232 RX line

4.17.5.5 hyp_seq

```
const struct hyp_ctx hyp_seq[ ]  [static]
```

Initial value:

```
= {
    {BSP_UART_WORDLEN_8, BSP_UART_PARITY_EVEN, 3},
    {BSP_UART_WORDLEN_8, BSP_UART_PARITY_ODD, 3},
    {BSP_UART_WORDLEN_8, BSP_UART_PARITY_NONE, 3},
    {BSP_UART_WORDLEN_9, BSP_UART_PARITY_EVEN, 0},
    {BSP_UART_WORDLEN_9, BSP_UART_PARITY_ODD, 0},
    {BSP_UART_WORDLEN_9, BSP_UART_PARITY_NONE, 0}
}
```

Sequence of hypotheses regarding UART parameters of RS-232 channels.

4.17.5.6 rx_buffer

```
uint32_t rx_buffer[BUFFER_SIZE] = {0}  [static]
```

Buffer storing timestamps of falling/rising edges of signal on the RS-232 RX line

4.17.5.7 tx_buffer

```
uint32_t tx_buffer[BUFFER_SIZE] = {0}  [static]
```

Buffer storing timestamps of falling/rising edges of signal on the RS-232 TX line

Chapter 5

Data Structure Documentation

5.1 baud_calc_ctx Struct Reference

Data Fields

- uint32_t ∗ **cnt**

  Pointer to tx_cnt or rx_cnt.
- uint32_t ∗ **buffer**

  Pointer to tx_buffer or rx_buffer.
- uint32_t **idx**

  Current position of buffer for analysis.
- uint32_t **min_len_bit**

  Minimum detected width of lower level on RS-232 line, valid over baudrates_list.
- uint32_t **max_len_bit**

  Maximum detected width of lower level on RS-232 line.
- uint32_t **baudrate**

  Calculated baudrate in bods.
- bool **toggle_bit**

  Flag showing current level on RS-232 line: true - upper one, false - lower one.
- bool **lin_detected**

  Flag whether LIN break is detected.
- bool **done**

  Flag whether baudrate calculation is finished.

5.1.1 Detailed Description

Context of baudrate calculation

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

- sniffer_rs232.c

5.2 bsp_led_pwm Struct Reference

Parameters of RGB LED blinking.

```
#include <bsp_led_rgb.h>
```

Data Fields

- uint32_t **width_on_ms**
  Width of enabled phase of blink in ms.
- uint32_t **width_off_ms**
  Width of disabled phase of blink in ms.

5.2.1 Detailed Description

Parameters of RGB LED blinking.

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

- bsp_led_rgb.h

5.3 bsp_led_rgb Struct Reference

RGB LED structure.

```
#include <bsp_led_rgb.h>
```

Data Fields

- uint8_t **r**
  RED value.
- uint8_t **g**
  GREEN value.
- uint8_t **b**
  BLUE value.

5.3.1 Detailed Description

RGB LED structure.

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

- bsp_led_rgb.h

5.4 button_init_ctx Struct Reference

Initializing context of BSP button.

```
#include <bsp_button.h>
```

Data Fields

- uint32_t press_delay_ms
- uint32_t press_min_dur_ms
- uint32_t long_press_dur_ms
- void(∗ button_isr_cb)(enum button_action action)

5.4.1 Detailed Description

Initializing context of BSP button.

5.4.2 Field Documentation

5.4.2.1 button_isr_cb

```
void(* button_isr_cb) (enum button_action action)
```

User callback called by button action

5.4.2.2 long_press_dur_ms

```
uint32_t long_press_dur_ms
```

Minimal duration in ms to detect long press action on the button

Note

should be more than press_min_dur_ms

5.4.2.3 press_delay_ms

```
uint32_t press_delay_ms
```

Delay in ms as time protection against contact bounce

5.4.2.4 press_min_dur_ms

```
uint32_t press_min_dur_ms
```

Minimal duration in ms of button pressing, also contact bounce protection together with press_delay_ms

Note

should be less than long_press_dur_ms

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

- bsp_button.h

5.5 flash_config Struct Reference

Firmware configuration.

```
#include <config.h>
```

Data Fields

- struct sniffer_rs232_config alg_config
- struct uart_presettings presettings
- enum rs232_trace_type trace_type
- enum rs232_interspace_type idle_presence
- enum rs232_interspace_type txrx_delimiter
- bool save_to_presettings
- uint32_t crc

5.5.1 Detailed Description

Firmware configuration.

5.5.2 Field Documentation

5.5.2.1 alg_config

```
struct sniffer_rs232_config alg_config
```

Algorithm settings sniffer_rs232_config

5.5.2.2 crc

```
uint32_t crc
```

CRC of configuration

5.5.2.3 idle_presence

```
enum rs232_interspace_type idle_presence
```

IDLE symbol for RS-232 data

5.5.2.4 presettings

```
struct uart_presettings presettings
```

UART presettings uart_presettings

5.5.2.5 save_to_presettings

```
bool save_to_presettings
```

Flag whether result of the algorithm Algorithm of Sniffer RS-232 is stored into uart_presettings

5.5.2.6 trace_type

```
enum rs232_trace_type trace_type
```

Trace type of RS-232 data rs232_trace_type

5.5.2.7 txrx_delimiter

```
enum rs232_interspace_type txrx_delimiter
```

Delimiter symbol between RS-232 TX & RX data

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

- config.h

5.6 hyp_check_ctx Struct Reference

Data Fields

- uint32_t error_parity_cnt

    Count of UART parity errors,.
- uint32_t error_frame_cnt

    Count of UART frame errors,.
- uint32_t **valid_cnt**

    Count of successfully received bytes over UART.
- bool **overflow**

    Flag whether overflow of receive buffer occured.

5.6.1 Detailed Description

Context of check of hypothesis

5.6.2 Field Documentation

5.6.2.1 error_frame_cnt

```
uint32_t error_frame_cnt
```

Count of UART frame errors,.

See also

BSP_UART_ERROR_FE

5.6.2.2 error_parity_cnt

```
uint32_t error_parity_cnt
```

Count of UART parity errors,.

See also

BSP_UART_ERROR_PE

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

- sniffer_rs232.c

5.7 hyp_ctx Struct Reference

Data Fields

- enum uart_wordlen wordlen
- enum uart_parity parity
- uint8_t jump

5.7.1 Detailed Description

Context of hypothesis

5.7.2 Field Documentation

5.7.2.1 jump

```
uint8_t jump
```

Next number of hypothesis from hyp_seq if count of
UART frame errors reach sniffer_rs232_config::uart_error_count

5.7.2.2 parity

```
enum uart_parity parity
```

Parity type

5.7.2.3 wordlen

```
enum uart_wordlen wordlen
```

Size of UART frame in bits

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

- sniffer_rs232.c

5.8 lcd1602_settings Struct Reference

Settings of BSP LCD1602.

```
#include <bsp_lcd1602.h>
```

Data Fields

- enum lcd1602_num_line **num_line**

  1-line or 2-line mode of display
- enum lcd1602_font_size **font_size**

  Font size.
- enum lcd1602_type_move_cursor **type_move_cursor**

  Move type of cursor.
- enum lcd1602_shift_entire_disp **shift_entire_disp**

  Shift type of entire display.
- enum lcd1602_type_interface **type_interface**

  Type of LCD1602 interface.
- enum lcd1602_disp_state **disp_state**

  Initial display state.
- enum lcd1602_cursor_state **cursor_state**

  Initial cursor state.
- enum lcd1602_cursor_blink_state **cursor_blink_state**

  Initial cursor blink state.

5.8.1 Detailed Description

Settings of BSP LCD1602.

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

- bsp_lcd1602.h

5.9 menu_item::menu Struct Reference

Menu context.

```
#include <menu.h>
```

Data Fields

- char ∗ **label**

  Label of menu.
- char **filler**

  Filler for label of menu.
- struct menu_color_config **color_config**

  Color settings of menu.
- struct menu_item ∗ **items**

  Menu items which menu includes.
- struct menu ∗ **next**

  Next menu in menu_list.

5.9.1 Detailed Description

Menu context.

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

- menu.h

5.10 menu_color Struct Reference

Menu color data.

```
#include <menu.h>
```

Data Fields

- enum menu_color_type **foreground**

  Color of foreground.
- enum menu_color_type **background**

  Color of background.

5.10.1 Detailed Description

Menu color data.

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

- menu.h

5.11 menu_color_config Struct Reference

Menu color settings.

```
#include <menu.h>
```

Data Fields

- struct menu_color **active**

  Colors of selected menu item.
- struct menu_color **inactive**

  Colors of not selected menu item.

5.11.1 Detailed Description

Menu color settings.

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

- menu.h

5.12 menu_config Struct Reference

Menu library settings.

```
#include <menu.h>
```

Data Fields

- bool is_looped
- uint32_t width
- uint32_t indent
- enum menu_pass_type pass_type
- enum menu_num_type num_type
- char num_delim
- uint8_t(∗ read_callback)(char ∗∗read_str)
- uint8_t(∗ write_callback)(char ∗write_str)

5.12.1 Detailed Description

Menu library settings.

5.12.2 Field Documentation

5.12.2.1 indent

```
uint32_t indent
```

With of vertical indent in symbols

5.12.2.2 is_looped

```
bool is_looped
```

Flag whether list of menu items is looped: position from first item moves to last one by moving up & position from last item moves to first one by moving down

5.12.2.3 num_delim

```
char num_delim
```

Delimiter between enumerator and label of menu item

5.12.2.4 num_type

```
enum menu_num_type num_type
```

Numbering type

5.12.2.5 pass_type

```
enum menu_pass_type pass_type
```

Type of passing input

5.12.2.6 read_callback

```
uint8_t(* read_callback) (char **read_str)
```

Callback to provide reading from console

5.12.2.7 width

```
uint32_t width
```

Width of menu in symbols

5.12.2.8 write_callback

```
uint8_t(* write_callback) (char *write_str)
```

Callback to provide writing to console

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

- menu.h

5.13 menu_item Struct Reference

Menu item context.

```
#include <menu.h>
```

Data Structures

- struct menu

  Menu context.

Data Fields

- struct menu_item ∗ **next**

  Next menu item in order.
- struct menu_item ∗ **prev**

  Previous menu item in order.
- struct menu_item::menu ∗ **menu_entry**

  Menu to which user can enter from menu item.
- uint8_t(∗ **callback**)(char ∗input, void ∗param)

  User callback called by actions on menu item.
- void ∗ **param**

  Optional parameters passed to menu_item::callback.
- char ∗ **prompt**

  Prompt of menu item.
- char ∗ **label**

  Label of menu item.
- char ∗ **value_left_border**

  Left border for value of menu item.
- char ∗ **value_right_border**

  Right border for value of menu item.
- char ∗ **value**

  Value of menu item.
- uint32_t **value_len**

  Length of value of menu item.

5.13.1 Detailed Description

Menu item context.

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

- menu.h

5.14 sniffer_rs232_config Struct Reference

Algorithm settings.

```
#include <sniffer_rs232.h>
```

Data Fields

- enum rs232_channel_type **channel_type**

  RS-232 channel detection type.
- uint32_t **valid_packets_count**

  Count of received bytes to approve a hypothesis.
- uint32_t **uart_error_count**

  Count of UART frame errors when hypothesis is failed.
- uint8_t **baudrate_tolerance**

  Tolerance of UART baudrate in percents.
- uint32_t **min_detect_bits**

  Minimum count of lower levels on RS-232 line to analyse baudrate.
- uint32_t **exec_timeout**

  Maximum time of algorithm execution.
- uint32_t **calc_attempts**

  Count of tries of algorithm calculation.
- bool **lin_detection**

  Flag whether LIN protocol should be detected.

5.14.1 Detailed Description

Algorithm settings.

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

- sniffer_rs232.h

5.15 uart_ctx Struct Reference

Context of the BSP UART instance.

Data Fields

- struct uart_init_ctx **init**

  Initializing context of the instance.
- uint8_t ∗ **tx_buff**

  Sent buffer used by DMA TX.
- uint8_t ∗ **rx_buff**

  Received buffer used by DMA RX.
- uint16_t **rx_idx_get**

  Read poisition in rx_buff used as ring buffer.
- uint16_t **rx_idx_set**

  Write poisition in rx_buff used as ring buffer.
- bool **frame_error**

  Flag whetner UART frame error is occured, used to separate LIN break from other frame errors.

5.15.1 Detailed Description

Context of the BSP UART instance.

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

- bsp_uart.c

5.16 uart_init_ctx Struct Reference

BSP UART initializing context.

```
#include <bsp_uart.h>
```

Data Fields

- uint32_t **baudrate**

  UART baudrate.
- uint32_t **tx_size**

  Size of sent buffer.
- uint32_t **rx_size**

  Size of received buffer.
- bool **lin_enabled**

  Flag whether LIN protocol is supported.
- enum uart_wordlen **wordlen**

  Word length.
- enum uart_parity **parity**

  Parity type.
- enum uart_stopbits **stopbits**

  Count of stop bits.
- void(∗ **error_isr_cb**)(enum uart_type type, uint32_t error, void ∗params)

  Callback for occurrence of BSP UART error.
- void(∗ **overflow_isr_cb**)(enum uart_type type, void ∗params)

  Callback for occurrence of overflow of receive buffer.
- void(∗ **lin_break_isr_cb**)(enum uart_type type, void ∗params)

  Callback for occurrence of LIN break detection.
- void ∗ **params**

  Optional parameters, passed to the callbacks.

5.16.1 Detailed Description

BSP UART initializing context.

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

- bsp_uart.h

5.17 uart_presettings Struct Reference

UART presettings.

```
#include <config.h>
```

Data Fields

- bool **enable**

  Flag whether presettings are enabled.
- uint32_t **baudrate**

  UART baudrate in bods.
- enum uart_wordlen **wordlen**

  Size of UART frame in bits.
- enum uart_parity **parity**

  Parity type.
- enum uart_stopbits **stopbits**

  Count of stop bits.
- bool **lin_enabled**

  Flag whether LIN protocol is supported.

5.17.1 Detailed Description

UART presettings.

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

- config.h

Chapter 6

File Documentation

6.1 app_led.h File Reference

Header of application layer of RGB LED.

```
#include "common.h"
#include <stdint.h>
#include <stddef.h>
```

Macros

- #define LED_EVENT_IS_VALID(X) (((uint32_t)(X)) < LED_EVENT_MAX)

Enumerations

- enum led_event {
  LED_EVENT_NONE = 0 , LED_EVENT_COMMON_ERROR , LED_EVENT_CRC_ERROR , LED_EVENT_FLASH_ERROR
  ,
  LED_EVENT_LCD1602_ERROR , LED_EVENT_IN_PROCESS , LED_EVENT_SUCCESS , LED_EVENT_FAILED
  ,
  LED_EVENT_UART_ERROR , LED_EVENT_UART_OVERFLOW , LED_EVENT_MAX }

  RGB LED event (type of LED behaivour)

Functions

- uint8_t app_led_init (void)
- uint8_t app_led_deinit (void)
- uint8_t app_led_set (enum led_event led_event)

6.1.1 Detailed Description

Header of application layer of RGB LED.

Author

JavaLandau

Copyright

MIT License

6.2 app_led.h

Go to the documentation of this file.

```
#ifndef __APP_LED_H
#define __APP_LED_H

#include "common.h"
#include <stdint.h>
#include <stddef.h>

enum led_event {
    LED_EVENT_NONE = 0,
    LED_EVENT_COMMON_ERROR,
    LED_EVENT_CRC_ERROR,
    LED_EVENT_FLASH_ERROR,
    LED_EVENT_LCD1602_ERROR,
    LED_EVENT_IN_PROCESS,
    LED_EVENT_SUCCESS,
    LED_EVENT_FAILED,
    LED_EVENT_UART_ERROR,
    LED_EVENT_UART_OVERFLOW,
    LED_EVENT_MAX
};

#define LED_EVENT_IS_VALID(X)          (((uint32_t)(X)) < LED_EVENT_MAX)

uint8_t app_led_init(void);

uint8_t app_led_deinit(void);

uint8_t app_led_set(enum led_event led_event);

#endif /* __APP_LED_H */
```

6.3 cli.h File Reference

Header of command line interface.

```
#include "common.h"
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include "config.h"
```

Functions

- uint8_t cli_init (void)
- uint8_t cli_menu_start (struct flash_config ∗config)
- uint8_t cli_menu_exit (void)
- bool cli_menu_is_started (void)
- void cli_trace (const char ∗format,...)
- uint8_t cli_rs232_trace (enum uart_type uart_type, enum rs232_trace_type trace_type, uint8_t ∗data, uint32_t len, bool break_line)
- uint8_t cli_welcome (const char ∗welcome, uint8_t wait_time_s, bool ∗forced_exit, bool ∗is_pressed)
- void cli_terminal_reset (void)

6.3.1 Detailed Description

Header of command line interface.

Author

JavaLandau

Copyright

MIT License

6.4 cli.h

Go to the documentation of this file.

```
#ifndef __CLI_H__
#define __CLI_H__

#include "common.h"
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include "config.h"

uint8_t cli_init(void);

uint8_t cli_menu_start(struct flash_config *config);

uint8_t cli_menu_exit(void);

bool cli_menu_is_started(void);

void cli_trace(const char *format, ...);

uint8_t cli_rs232_trace(enum uart_type uart_type,
                        enum rs232_trace_type trace_type,
                        uint8_t *data,
                        uint32_t len,
                        bool break_line);

uint8_t cli_welcome(const char *welcome, uint8_t wait_time_s, bool *forced_exit, bool *is_pressed);

void cli_terminal_reset(void);

#endif //__CLI_H__
```

6.5 config.h File Reference

Header of flash configuration.

```
#include "common.h"
#include "stm32f4xx_hal.h"
#include "sniffer_rs232.h"
```

Data Structures

- struct uart_presettings

  UART presettings.
- struct flash_config

  Firmware configuration.

Macros

- #define RS232_TRACE_TYPE_VALID(X) ((X) < RS232_TRACE_MAX)
- #define RS232_INTERSPACE_TYPE_VALID(X) ((X) < RS232_INTERSPCACE_MAX)
- #define UART_PRESETTINGS_DEFAULT()
- #define FLASH_CONFIG_DEFAULT()

Enumerations

- enum rs232_trace_type { RS232_TRACE_HEX = 0 , RS232_TRACE_HYBRID , RS232_TRACE_MAX }

  Trace type of RS-232 data.
- enum rs232_interspace_type { RS232_INTERSPCACE_NONE = 0 , RS232_INTERSPCACE_SPACE , RS232_INTERSPCACE_NEW_LINE , RS232_INTERSPCACE_MAX }

  Type of interspaces between RS-232 data.

Functions

- uint8_t config_save (struct flash_config ∗config)
- uint8_t config_read (struct flash_config ∗config)

6.5.1 Detailed Description

Header of flash configuration.

Author

JavaLandau

Copyright

MIT License

6.6 config.h

Go to the documentation of this file.

```
#ifndef __CONFIG_H__
#define __CONFIG_H__

#include "common.h"
#include "stm32f4xx_hal.h"
#include "sniffer_rs232.h"

#define RS232_TRACE_TYPE_VALID(X)       ((X) < RS232_TRACE_MAX)

#define RS232_INTERSPACE_TYPE_VALID(X)  ((X) < RS232_INTERSPCACE_MAX)

enum rs232_trace_type {
    RS232_TRACE_HEX = 0,
    RS232_TRACE_HYBRID,
    RS232_TRACE_MAX
};

enum rs232_interspace_type {
    RS232_INTERSPCACE_NONE = 0,
    RS232_INTERSPCACE_SPACE,
    RS232_INTERSPCACE_NEW_LINE,
    RS232_INTERSPCACE_MAX
};

struct uart_presettings {
    bool enable;
    uint32_t baudrate;
    enum uart_wordlen wordlen;
    enum uart_parity parity;
    enum uart_stopbits stopbits;
    bool lin_enabled;
};

#pragma pack(1)
struct flash_config {
    struct sniffer_rs232_config alg_config;
    struct uart_presettings presettings;
    enum rs232_trace_type trace_type;
    enum rs232_interspace_type idle_presence;
    enum rs232_interspace_type txrx_delimiter;
    bool save_to_presettings;
    uint32_t crc;
};
#pragma pack()

#define UART_PRESETTINGS_DEFAULT()  {\
.enable = false,\
.parity = BSP_UART_PARITY_NONE,\
.baudrate = 0,\
.stopbits = BSP_UART_STOPBITS_1,\
.wordlen = BSP_UART_WORDLEN_8,\
.lin_enabled = false\
}

#define FLASH_CONFIG_DEFAULT()  {\
.alg_config = SNIFFER_RS232_CONFIG_DEFAULT(),\
.presettings = UART_PRESETTINGS_DEFAULT(),\
.trace_type = RS232_TRACE_HEX,\
.idle_presence = RS232_INTERSPCACE_NONE,\
.txrx_delimiter = RS232_INTERSPCACE_NONE,\
.save_to_presettings = true\
}

uint8_t config_save(struct flash_config *config);

uint8_t config_read(struct flash_config *config);

#endif //__CONFIG_H__
```

6.7 menu.h File Reference

Header of menu library.

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

```
#include <stdbool.h>
#include <stddef.h>
```

Data Structures

- struct menu_color

    Menu color data.
- struct menu_color_config

    Menu color settings.
- struct menu_item

    Menu item context.
- struct menu_item::menu

    Menu context.
- struct menu_config

    Menu library settings.

Macros

- #define **MENU_MAX_STR_LEN** 256

    Maximum valid length of strings used within menu library.
- #define **MENU_COLOR_RESET** "\33[0;37;40m"

    Escape sequence to reset console colors.
- #define **MENU_RETURN_HOME** "\33[H"

    Escape sequence to return cursor to left top corner of console.
- #define **MENU_LINE_UP** "\33[A"

    Escape sequence to move cursor one line up.
- #define **MENU_LINE_DOWN** "\33[B"

    Escape sequence to move cursor one line down.
- #define **MENU_LINE_ERASE** "\33[2K"

    Escape sequence to erase current line.
- #define **MENU_SCREEN_ERASE** "\33[2J"

    Escape sequence to erase screen of console.
- #define MENU_COLOR_CONFIG_DEFAULT()

Enumerations

- enum menu_color_type {
  MENU_COLOR_BLACK = 0 , MENU_COLOR_RED , MENU_COLOR_GREEN , MENU_COLOR_YELLOW
  ,
  MENU_COLOR_BLUE , MENU_COLOR_MAGENTA , MENU_COLOR_CYAN , MENU_COLOR_WHITE ,
  MENU_COLOR_MAX }

    Menu colors.
- enum menu_pass_type { MENU_PASS_NONE = 0 , MENU_PASS_WITH_PROMPT , MENU_PASS_ALWAYS
  , MENU_PASS_MAX }

    Type of passing input to menu_item::callback.
- enum menu_num_type {
  MENU_NUM_NONE = 0 , MENU_NUM_DIGITAL , MENU_NUM_UPPER_LETTER , MENU_NUM_LOWER_LETTER
  ,
  MENU_NUM_MAX }

    Numbering types.

Functions

- void menu_all_destroy (void)
- struct menu ∗ menu_create (char ∗label, char filler, struct menu_color_config ∗color_config)
- uint8_t menu_entry (struct menu ∗menu)
- uint8_t menu_item_value_set (struct menu_item ∗menu_item, const char ∗value)
- struct menu_item ∗ menu_current_item_get (void)
- char ∗ menu_item_label_get (struct menu_item ∗menu_item)
- struct menu ∗ menu_by_label_get (const char ∗label)
- struct menu_item ∗ menu_item_by_label_get (struct menu ∗menu, const char ∗label)
- struct menu_item ∗ menu_item_by_label_only_get (const char ∗label)
- bool menu_is_started (void)
- uint8_t menu_start (struct menu_config ∗config, struct menu ∗menu)
- uint8_t menu_exit (void)
- struct menu_item ∗ menu_item_add (struct menu ∗menu, const char ∗label, const char ∗prompt, const char ∗value_border, uint8_t(∗callback)(char ∗input, void ∗param), void ∗param, struct menu ∗menu_entry)

6.7.1 Detailed Description

Header of menu library.

Author

JavaLandau

Copyright

MIT License

6.8 menu.h

Go to the documentation of this file.

```
#ifndef __MENU_H__
#define __MENU_H__

#include "common.h"
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>

#define MENU_MAX_STR_LEN                256

#define MENU_COLOR_RESET                "\33[0;37;40m"

#define MENU_RETURN_HOME                "\33[H"

#define MENU_LINE_UP                    "\33[A"

#define MENU_LINE_DOWN                  "\33[B"

#define MENU_LINE_ERASE                 "\33[2K"

#define MENU_SCREEN_ERASE               "\33[2J"

enum menu_color_type {
    MENU_COLOR_BLACK = 0,
    MENU_COLOR_RED,
    MENU_COLOR_GREEN,
    MENU_COLOR_YELLOW,
    MENU_COLOR_BLUE,
    MENU_COLOR_MAGENTA,
    MENU_COLOR_CYAN,
    MENU_COLOR_WHITE,
    MENU_COLOR_MAX
};

enum menu_pass_type {
    MENU_PASS_NONE = 0,
    MENU_PASS_WITH_PROMPT,
    MENU_PASS_ALWAYS,
    MENU_PASS_MAX
};

enum menu_num_type {
    MENU_NUM_NONE = 0,
    MENU_NUM_DIGITAL,
    MENU_NUM_UPPER_LETTER,
    MENU_NUM_LOWER_LETTER,
    MENU_NUM_MAX
};

struct menu_color {
    enum menu_color_type foreground;
    enum menu_color_type background;
};

struct menu_color_config {
    struct menu_color active;
    struct menu_color inactive;
};

struct menu_item {
    struct menu_item *next;
    struct menu_item *prev;

    struct menu {
        char *label;
        char filler;
        struct menu_color_config color_config;
        struct menu_item *items;
        struct menu *next;
    } *menu_entry;

    uint8_t (*callback) (char *input, void *param);
    void *param;
    char *prompt;
    char *label;
    char *value_left_border;
    char *value_right_border;
    char *value;
    uint32_t value_len;
};

struct menu_config {
    bool is_looped;
    uint32_t width;
    uint32_t indent;
    enum menu_pass_type pass_type;
    enum menu_num_type num_type;
    char num_delim;
    uint8_t (*read_callback) (char **read_str);
    uint8_t (*write_callback) (char *write_str);
};

#define MENU_COLOR_CONFIG_DEFAULT()  {\
.active = {.foreground = MENU_COLOR_BLUE, .background = MENU_COLOR_WHITE},\
.inactive = {.foreground = MENU_COLOR_WHITE, .background = MENU_COLOR_BLUE}\
}

void menu_all_destroy(void);

struct menu *menu_create(char *label, char filler, struct menu_color_config *color_config);

uint8_t menu_entry(struct menu *menu);

uint8_t menu_item_value_set(struct menu_item *menu_item, const char *value);

struct menu_item *menu_current_item_get(void);

char *menu_item_label_get(struct menu_item *menu_item);

struct menu *menu_by_label_get(const char *label);

struct menu_item *menu_item_by_label_get(struct menu *menu, const char *label);

struct menu_item *menu_item_by_label_only_get(const char *label);

bool menu_is_started(void);
```

```
uint8_t menu_start(struct menu_config *config, struct menu *menu);

uint8_t menu_exit(void);

struct menu_item * menu_item_add(struct menu *menu,
                                 const char *label,
                                 const char *prompt,
                                 const char *value_border,
                                 uint8_t (*callback) (char *input, void *param),
                                 void *param,
                                 struct menu *menu_entry);

#endif //__MENU_H__
```

6.9 sniffer_rs232.h File Reference

Header of algorithm of Sniffer RS-232.

```
#include "common.h"
#include "stm32f4xx_hal.h"
#include "bsp_uart.h"
#include <stdbool.h>
```

Data Structures

- struct sniffer_rs232_config
  Algorithm settings.

Macros

- #define RS232_CHANNEL_TYPE_VALID(TYPE) (((uint32_t)(TYPE)) < RS232_CHANNEL_MAX)
- #define SNIFFER_RS232_CFG_PARAM_MIN(X) sniffer_rs232_config_item_range((uint32_t)&((struct sniffer_rs232_config∗)0)->X, true)
- #define SNIFFER_RS232_CFG_PARAM_MAX(X) sniffer_rs232_config_item_range((uint32_t)&((struct sniffer_rs232_config∗)0)->X, false)
- #define SNIFFER_RS232_CFG_PARAM_IS_VALID(X, V) (((V) >= SNIFFER_RS232_CFG_PARAM_MIN(X)) && ((V) <= SNIFFER_RS232_CFG_PARAM_MAX(X)))
- #define SNIFFER_RS232_CONFIG_DEFAULT()

Enumerations

- enum rs232_channel_type {
  RS232_CHANNEL_TX = 0 , RS232_CHANNEL_RX , RS232_CHANNEL_ANY , RS232_CHANNEL_ALL ,
  RS232_CHANNEL_MAX }

Functions

- uint8_t sniffer_rs232_init (struct sniffer_rs232_config ∗__config)
- uint8_t sniffer_rs232_deinit (void)
- uint8_t sniffer_rs232_calc (struct uart_init_ctx ∗uart_params)
- uint32_t sniffer_rs232_config_item_range (uint32_t shift, bool is_min)
- bool sniffer_rs232_config_check (struct sniffer_rs232_config ∗__config)

6.9.1 Detailed Description

Header of algorithm of Sniffer RS-232.

Author

JavaLandau

Copyright

MIT License

6.10 sniffer_rs232.h

Go to the documentation of this file.

```
#ifndef __SNIFFER_RS232_H__
#define __SNIFFER_RS232_H__

#include "common.h"
#include "stm32f4xx_hal.h"
#include "bsp_uart.h"
#include <stdbool.h>

enum rs232_channel_type {
    RS232_CHANNEL_TX = 0,
    RS232_CHANNEL_RX,
    RS232_CHANNEL_ANY,
    RS232_CHANNEL_ALL,
    RS232_CHANNEL_MAX
};

#define RS232_CHANNEL_TYPE_VALID(TYPE)          (((uint32_t)(TYPE)) < RS232_CHANNEL_MAX)

struct sniffer_rs232_config {
    enum rs232_channel_type channel_type;
    uint32_t valid_packets_count;
    uint32_t uart_error_count;
    uint8_t baudrate_tolerance;
    uint32_t min_detect_bits;
    uint32_t exec_timeout;
    uint32_t calc_attempts;
    bool lin_detection;
};

#define SNIFFER_RS232_CFG_PARAM_MIN(X)          sniffer_rs232_config_item_range((uint32_t)&((struct
   sniffer_rs232_config*)0)->X, true)

#define SNIFFER_RS232_CFG_PARAM_MAX(X)          sniffer_rs232_config_item_range((uint32_t)&((struct
   sniffer_rs232_config*)0)->X, false)

#define SNIFFER_RS232_CFG_PARAM_IS_VALID(X, V)  (((V) >= SNIFFER_RS232_CFG_PARAM_MIN(X)) && ((V) <=
   SNIFFER_RS232_CFG_PARAM_MAX(X)))

#define SNIFFER_RS232_CONFIG_DEFAULT() {\
.channel_type = RS232_CHANNEL_ANY,\
.valid_packets_count = 20,\
.uart_error_count = 2,\
.baudrate_tolerance = 10,\
.min_detect_bits = 48,\
.exec_timeout = 600,\
.calc_attempts = 3,\
.lin_detection = false\
}

uint8_t sniffer_rs232_init(struct sniffer_rs232_config *__config);

uint8_t sniffer_rs232_deinit(void);

uint8_t sniffer_rs232_calc(struct uart_init_ctx *uart_params);

uint32_t sniffer_rs232_config_item_range(uint32_t shift, bool is_min);

bool sniffer_rs232_config_check(struct sniffer_rs232_config *__config);

#endif //__SNIFFER_RS232_H__
```

6.11 app_led.c File Reference

Application layer of RGB LED.

```
#include "common.h"
#include "app_led.h"
#include "bsp_led_rgb.h"
```

Functions

- uint8_t app_led_init (void)
- uint8_t app_led_deinit (void)
- uint8_t app_led_set (enum led_event led_event)

Variables

- static const struct bsp_led_rgb **led_disabled** = {.r = 0, .g = 0, .b = 0}

  Settings for disabled LED.
- static const struct bsp_led_rgb **led_red** = {.r = 255, .g = 0, .b = 0}

  Settings for LED with RED color.
- static const struct bsp_led_rgb **led_green** = {.r = 0, .g = 255, .b = 0}

  Settings for LED with GREEN color.
- static const struct bsp_led_rgb **led_yellow** = {.r = 255, .g = 255, .b = 0}

  Settings for LED with YELLOW color.
- static const struct bsp_led_rgb **led_magenta** = {.r = 100, .g = 0, .b = 50}

  Settings for LED with MAGENTA color.
- static const struct bsp_led_pwm **blink_rare_on** = {.width_on_ms = 150, .width_off_ms = 1000}

  Settings to LED blinking with short enabled phase.
- static const struct bsp_led_pwm **blink_fast** = {.width_on_ms = 250, .width_off_ms = 250}

  Settings to LED fastly blinking with equaled enabled & disabled phases.
- static const struct bsp_led_pwm **blink_rare_off** = {.width_on_ms = 1000, .width_off_ms = 150}

  Settings to LED blinking with short disabled phase.

6.11.1 Detailed Description

Application layer of RGB LED.

Author

JavaLandau

Copyright

MIT License

The file includes implementation of application layer of RGB LED

6.12 basic_interrupts.c File Reference

File of handlers for basic interrups.

```
#include "stm32f4xx_hal.h"
#include "bsp_led_rgb.h"
```

Functions

- void NMI_Handler (void)
- void HardFault_Handler (void)
- void MemManage_Handler (void)
- void BusFault_Handler (void)
- void UsageFault_Handler (void)
- void SVC_Handler (void)
- void DebugMon_Handler (void)
- void PendSV_Handler (void)
- void SysTick_Handler (void)

6.12.1 Detailed Description

File of handlers for basic interrups.

Author

JavaLandau

Copyright

MIT License

The file includes handlers for main MPU interrupts

6.13 cli.c File Reference

Command line interface.

```
#include "cli.h"
#include "menu.h"
#include "config.h"
#include "bsp_uart.h"
#include "sniffer_rs232.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <ctype.h>
```

Macros

- #define **UART_TRACE_BUFF_SIZE** (256)

  Size of string buffer used in cli_trace.
- #define **UART_RX_BUFF_SIZE** (256)

  Size of UART receive buffer for CLI BSP UART.
- #define **UART_TX_BUFF_SIZE** (6 ∗ UART_RX_BUFF_SIZE)

  Size of UART send buffer for CLI BSP UART.
- #define **TX_COLOR** MENU_COLOR_GREEN

  Color of traced RS-232 TX data.
- #define **RX_COLOR** MENU_COLOR_MAGENTA

  Color of traced RS-232 RX data.

Functions

- static uint8_t __cli_menu_entry (char ∗input, void ∗param)
- static uint8_t __cli_menu_set_defaults (char ∗input, void ∗param)
- static uint8_t __cli_menu_exit (char ∗input, void ∗param)
- static uint8_t __cli_menu_cfg_set (char ∗input, void ∗param)
- static char ∗ __cli_prompt_generator (const char ∗menu_item_label)
- static uint8_t __cli_menu_cfg_values_set (struct flash_config ∗config)
- static void __cli_uart_overflow_cb (enum uart_type type, void ∗params)
- static void __cli_uart_error_cb (enum uart_type type, uint32_t error, void ∗params)
- static uint8_t __cli_menu_write_cb (char ∗data)
- static uint8_t __cli_menu_read_cb (char ∗∗data)
- uint8_t cli_menu_exit (void)
- bool cli_menu_is_started (void)
- uint8_t cli_init (void)
- void cli_trace (const char ∗format,...)
- uint8_t cli_welcome (const char ∗welcome, uint8_t wait_time_s, bool ∗forced_exit, bool ∗is_pressed)
- void cli_terminal_reset (void)
- uint8_t cli_menu_start (struct flash_config ∗config)
- uint8_t cli_rs232_trace (enum uart_type uart_type, enum rs232_trace_type trace_type, uint8_t ∗data, uint32_t len, bool break_line)

Variables

- struct {
  bool **uart_error**
  Flag whether UART errors on CLI occured.
  bool **uart_overflow**
  Flag whether UART receive buffer is overflown.
  } **cli_state** = {0}

  State of UART CLI.
- static struct flash_config **old_config**

  Copy of input configuration.
- static struct flash_config ∗ **flash_config** = NULL

  Current configuration.
- static bool **is_config_changed** = false

  Flag whether configuration is changed.
- static struct menu_color_config **color_config_select** = MENU_COLOR_CONFIG_DEFAULT()

Menu color settings for menus wihtout emphasised choice "yes-no".

- static struct menu_color_config color_config_choose

  Menu color settings for menus with emphasised choice "yes-no".
- static const char ∗ rs232_trace_type_str [ ]

  Array of string aliases for rs232_trace_type for output purposes.
- static const char ∗ rs232_interspace_type_str [ ]

  Array of string aliases for rs232_interspace_type for output purposes.
- static const char ∗ uart_parity_str [ ]

  Array of string aliases for uart_parity for output purposes.
- static const char ∗ rs232_channel_type_str [ ]

  Array of string aliases for rs232_channel_type for output purposes.
- struct {
  char ∗ **label**
  Label of menu.
  struct menu_color_config ∗ **color_config**
  Color settings of menu.
  } init_menus [ ]

  List of menus included in configuration menu.
- struct {
  char ∗ **menu_label**
  Label of menu which menu item belongs to.
  char ∗ **menu_item_label**
  Label of menu item.
  char ∗ **value_border**
  Border for value of menu item.
  uint8_t(∗ **callback**)(char ∗input, void ∗param)
  User callback by actions on menu item.
  char ∗ **menu_entry_label**
  Label of menu to which user can enter from menu item.
  } **init_menu_items** [ ]

  Structure of all menu items included in configuration menu.
- static uint8_t ∗ **__menu_rx_buff** = NULL

  Receive buffer for CLI BSP UART.

6.13.1 Detailed Description

Command line interface.

Author

JavaLandau

Copyright

MIT License

The file includes API to communicate with the device via CLI

6.14 config.c File Reference

Flash configuration.

```
#include "config.h"
#include "bsp_crc.h"
#include <string.h>
```

Macros

- #define **FLASH_SECTOR_CFG_ADDR** (0x08060000)

    Address of internal flash where configuration is stored.

Functions

- uint8_t config_save (struct flash_config ∗config)
- uint8_t config_read (struct flash_config ∗config)

6.14.1 Detailed Description

Flash configuration.

Author

JavaLandau

Copyright

MIT License

The file includes API to save/load configuration into/from internal MPU flash

6.15 main.c File Reference

Main project file.

```
#include "common.h"
#include "stm32f4xx_hal.h"
#include "app_led.h"
#include "bsp_rcc.h"
#include "bsp_lcd1602.h"
#include "bsp_uart.h"
#include "bsp_crc.h"
#include "bsp_button.h"
#include "sniffer_rs232.h"
#include "config.h"
#include "cli.h"
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
```

Macros

- #define **APP_VERSION** "1.0-RC3"

  Firmware version.
- #define **UART_RX_BUFF** (256)

  Size of RX buffer to store data received from BSP UART.
- #define IS_UART_ERROR(X) (uart_flags[X].error || uart_flags[X].overflow)

Functions

- static void uart_lin_break_cb (enum uart_type type, void ∗params)
- static void uart_overflow_cb (enum uart_type type, void ∗params)
- static void uart_error_cb (enum uart_type type, uint32_t error, void ∗params)
- static void button_cb (enum button_action action)
- static bool button_wait_event (uint32_t tmt)
- static void internal_error (enum led_event led_event)
- int main ()

Variables

- static const char **uart_parity_sym** [ ] = {'N', 'E', 'O'}

  Array of char aliases for uart_parity for output purposes.
- static const char ∗ **display_uart_type_str** [ ] = {"CLI", "TX", "RX"}

  Array of string aliases for uart_type for output purposes.
- static bool **press_event** = false

  Flag whether press event on the button is occured.
- struct {
  uint32_t **error**
  Mask of UART errors.
  bool **overflow**
  Flag whether UART RX buffer is overflown before call bsp_uart_read.
  bool **lin_break**
  Flag whether LIN break detection is occured.
  } **uart_flags** [BSP_UART_TYPE_MAX] = {0}

  UART flags.

6.15.1 Detailed Description

Main project file.

Author

JavaLandau

Copyright

MIT License

The file includes main routine of the firmware: start menu of configuration, algorithm usage, error handlers and etc.

6.16 menu.c File Reference

Menu library.

```
#include "menu.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
```

Macros

- #define **MENU_COLOR_SIZE** 10

  Length of escape sequence for colors.
- #define MENU_PASS_TYPE_IS_VALID(X) (((uint32_t)(X)) < MENU_PASS_MAX)
- #define MENU_NUM_TYPE_IS_VALID(X) (((uint32_t)(X)) < MENU_NUM_MAX)

Functions

- static uint32_t __menu_strlen (const char ∗str)
- static struct menu_item ∗ __menu_get_last_item (void)
- static bool __menu_item_is_in_menu (struct menu ∗menu, struct menu_item ∗menu_item)
- static uint8_t __menu_enumerator_inc (enum menu_num_type num_type, char ∗enumerator, uint8_t enum←_len)
- static uint8_t __menu_redraw (struct menu_item ∗prev_item_active, struct menu_item ∗new_item_active)
- uint8_t menu_exit (void)
- struct menu_item ∗ menu_current_item_get (void)
- char ∗ menu_item_label_get (struct menu_item ∗menu_item)
- struct menu_item ∗ menu_item_by_label_get (struct menu ∗menu, const char ∗label)
- struct menu_item ∗ menu_item_by_label_only_get (const char ∗label)
- struct menu ∗ menu_by_label_get (const char ∗label)
- uint8_t menu_item_value_set (struct menu_item ∗menu_item, const char ∗value)
- uint8_t menu_entry (struct menu ∗menu)
- bool menu_is_started (void)
- uint8_t menu_start (struct menu_config ∗config, struct menu ∗menu)
- struct menu ∗ menu_create (char ∗label, char filler, struct menu_color_config ∗color_config)
- void menu_all_destroy (void)
- struct menu_item ∗ menu_item_add (struct menu ∗menu, const char ∗label, const char ∗prompt, const char ∗value_border, uint8_t(∗callback)(char ∗input, void ∗param), void ∗param, struct menu ∗menu_entry)

Variables

- static struct menu_config **menu_config** = {0}

  Local copy of menu settings.
- static struct menu_item ∗ **cur_item** = NULL

  Current menu item from cur_menu.
- static struct menu_item ∗ **prev_item** = NULL

  Previous menu item.
- static struct menu ∗ **cur_menu** = NULL

  Current menu from menu_list.
- struct menu ∗ **menu_list** = NULL

  Menu list.
- static bool **__exit** = true

  Flag whether console menu is finished.

6.16.1 Detailed Description

Menu library.

Author

JavaLandau

Copyright

MIT License

The file contains implementation and API for console menu library

6.17 sniffer_rs232.c File Reference

Algorithm of Sniffer RS-232.

```
#include "sniffer_rs232.h"
#include "bsp_gpio.h"
#include "bsp_rcc.h"
#include <stdbool.h>
#include <string.h>
```

Data Structures

- struct hyp_check_ctx
- struct baud_calc_ctx
- struct hyp_ctx

Macros

- #define **BUFFER_SIZE** (512)

  Size of buffers tx_buffer & rx_buffer.
- #define **UART_BUFF_SIZE** (128)

  Size of receive buffer used in BSP UART.
- #define LIN_BREAK_MIN_LEN (10)

Functions

- static void __sniffer_rs232_tim_msp_init (TIM_HandleTypeDef ∗htim)
- static void __sniffer_rs232_tim_msp_deinit (TIM_HandleTypeDef ∗htim)
- static uint32_t __sniffer_rs232_baudrate_get (uint32_t len_bit)
- static uint8_t __sniffer_rs232_line_baudrate_calc_init (GPIO_TypeDef ∗gpiox, uint16_t pin, IRQn_Type irq←_type)
- static void __sniffer_rs232_line_baudrate_calc (struct baud_calc_ctx ∗ctx)
- static uint8_t __sniffer_rs232_baudrate_calc (enum rs232_channel_type channel_type, uint32_t ∗baudrate, bool ∗lin_detected)
- static void __sniffer_rs232_uart_overflow_cb (enum uart_type type, void ∗params)
- static void __sniffer_rs232_uart_error_cb (enum uart_type type, uint32_t error, void ∗params)
- static uint8_t __sniffer_rs232_params_calc (enum rs232_channel_type channel_type, uint32_t baudrate, int8_t ∗hyp_num)
- uint32_t sniffer_rs232_config_item_range (uint32_t shift, bool is_min)
- bool sniffer_rs232_config_check (struct sniffer_rs232_config ∗__config)
- uint8_t sniffer_rs232_init (struct sniffer_rs232_config ∗__config)
- uint8_t sniffer_rs232_deinit (void)
- uint8_t sniffer_rs232_calc (struct uart_init_ctx ∗uart_params)
- void EXTI3_IRQHandler (void)
- void EXTI9_5_IRQHandler (void)

Variables

- static TIM_HandleTypeDef alg_tim = {.Instance = TIM6}
- static EXTI_HandleTypeDef hexti1 = {.Line = EXTI_LINE_3}
- static EXTI_HandleTypeDef hexti2 = {.Line = EXTI_LINE_5}
- static uint32_t **tx_cnt** = 0

  Current filling level of tx_buffer.
- static uint32_t **rx_cnt** = 0

  Current filling level of rx_buffer.
- static uint32_t tx_buffer [BUFFER_SIZE] = {0}
- static uint32_t rx_buffer [BUFFER_SIZE] = {0}
- static const uint32_t baudrates_list [ ] = {921600, 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800, 2400}
- static const struct hyp_ctx hyp_seq [ ]

  Sequence of hypotheses regarding UART parameters of RS-232 channels.
- static struct sniffer_rs232_config **config**

  Local copy of algorithm settings.

6.17.1 Detailed Description

Algorithm of Sniffer RS-232.

Author

JavaLandau

Copyright

MIT License

The file includes recognizing algorithm of RS-232 parameters

6.18 bsp_button.h File Reference

Header of BSP button module.

```
#include <stdint.h>
#include "stm32f4xx_hal.h"
```

Data Structures

- struct button_init_ctx

    Initializing context of BSP button.

Enumerations

- enum button_action { BUTTON_NONE = 0 , BUTTON_PRESSED , BUTTON_LONG_PRESSED , BUTTON_ACTION_MAX }

    BSP button actions.

Functions

- uint8_t bsp_button_init (struct button_init_ctx ∗init_ctx)
- uint8_t bsp_button_deinit (void)

6.18.1 Detailed Description

Header of BSP button module.

Author

JavaLandau

Copyright

MIT License

6.19 bsp_button.h

Go to the documentation of this file.

```

#ifndef __BSP_BUTTON_H__
#define __BSP_BUTTON_H__

#include <stdint.h>
#include "stm32f4xx_hal.h"

enum button_action {
    BUTTON_NONE = 0,
    BUTTON_PRESSED,
    BUTTON_LONG_PRESSED,
    BUTTON_ACTION_MAX
};

struct button_init_ctx {
    uint32_t press_delay_ms;
    uint32_t press_min_dur_ms;
    uint32_t long_press_dur_ms;
    void (*button_isr_cb)(enum button_action action);
};

uint8_t bsp_button_init(struct button_init_ctx *init_ctx);

uint8_t bsp_button_deinit(void);

#endif //__BSP_BUTTON_H__
```

6.20 bsp_crc.h File Reference

Header of BSP CRC module.

```
#include <stdint.h>
#include "stm32f4xx_hal.h"
```

Functions

- uint8_t bsp_crc_init (void)
- uint8_t bsp_crc_deinit (void)
- uint8_t bsp_crc_calc (uint8_t ∗data, uint32_t len, uint32_t ∗result)

6.20.1 Detailed Description

Header of BSP CRC module.

Author

JavaLandau

Copyright

MIT License

6.21 bsp_crc.h

Go to the documentation of this file.

```

#ifndef __BSP_CRC_H__
#define __BSP_CRC_H__

#include <stdint.h>
#include "stm32f4xx_hal.h"

uint8_t bsp_crc_init(void);

uint8_t bsp_crc_deinit(void);

uint8_t bsp_crc_calc(uint8_t *data, uint32_t len, uint32_t *result);

#endif //__BSP_CRC_H__
```

6.22 bsp_gpio.h File Reference

Header of BSP GPIO module.

```
#include <stdint.h>
#include "stm32f4xx_hal.h"
```

Macros

- #define BSP_GPIO_PORT_READ(GPIOX, GPIO_PIN) (!!(GPIOX->IDR & GPIO_PIN))
- #define BSP_GPIO_PORT_WRITE(GPIOX, GPIO_PIN, LEVEL) (GPIOX->BSRR = LEVEL ? GPIO_PIN : ((uint32_t)GPIO_PIN << 16U))
- #define BSP_GPIO_FORCE_OUTPUT_MODE(GPIOX, GPIO_NUM)

Functions

- uint8_t bsp_gpio_bulk_read (GPIO_TypeDef ∗gpiox, const uint16_t ∗gpio_pins, uint16_t ∗gpio_states)
- uint8_t bsp_gpio_bulk_write (GPIO_TypeDef ∗gpiox, const uint16_t ∗gpio_pins, const uint16_t gpio_states)

6.22.1 Detailed Description

Header of BSP GPIO module.

Author

JavaLandau

Copyright

MIT License

6.23 bsp_gpio.h

Go to the documentation of this file.

```

#ifndef __BSP_GPIO_H__
#define __BSP_GPIO_H__

#include <stdint.h>
#include "stm32f4xx_hal.h"

uint8_t bsp_gpio_bulk_read(GPIO_TypeDef* gpiox, const uint16_t *gpio_pins, uint16_t *gpio_states);

uint8_t bsp_gpio_bulk_write(GPIO_TypeDef* gpiox, const uint16_t *gpio_pins, const uint16_t gpio_states);


#define BSP_GPIO_PORT_READ(GPIOX, GPIO_PIN)             (!!(GPIOX->IDR & GPIO_PIN))

#define BSP_GPIO_PORT_WRITE(GPIOX, GPIO_PIN, LEVEL)     (GPIOX->BSRR = LEVEL ? GPIO_PIN : ((uint32_t)GPIO_PIN << 16U))

#define BSP_GPIO_FORCE_OUTPUT_MODE(GPIOX, GPIO_NUM)  \
do {\
GPIOX->MODER = (GPIOX->MODER | (1 << (2 * GPIO_NUM))) & (~(1 << (2 * GPIO_NUM + 1)));\
} while(0)

#endif //__BSP_GPIO_H__
```

6.24 bsp_lcd1602.h File Reference

Header of BSP LCD1602 module.

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

Data Structures

- struct lcd1602_settings

    Settings of BSP LCD1602.

Enumerations

- enum lcd1602_type_shift {
LCD1602_SHIFT_CURSOR_UNDEF = -1 , LCD1602_SHIFT_CURSOR_LEFT , LCD1602_SHIFT_CURSOR_RIGHT , LCD1602_SHIFT_DISPLAY_LEFT ,
LCD1602_SHIFT_DISPLAY_RIGHT , LCD1602_SHIFT_MAX }

    Type of cursor/display shift.
- enum lcd1602_num_line { LCD1602_NUM_LINE_UNDEF = -1 , LCD1602_NUM_LINE_1 , LCD1602_NUM_LINE_2 , LCD1602_NUM_LINE_MAX }

    Numbrt line of LCD1602.
- enum lcd1602_font_size { LCD1602_FONT_SIZE_UNDEF = -1 , LCD1602_FONT_SIZE_5X8 , LCD1602_FONT_SIZE_5X11 , LCD1602_FONT_SIZE_MAX }

    Types of font size.
- enum lcd1602_type_move_cursor { LCD1602_CURSOR_MOVE_UNDEF = -1 , LCD1602_CURSOR_MOVE_LEFT , LCD1602_CURSOR_MOVE_RIGHT , LCD1602_CURSOR_MOVE_MAX }

    Move types of cursor.
- enum lcd1602_shift_entire_disp { LCD1602_SHIFT_ENTIRE_UNDEF = -1 , LCD1602_SHIFT_ENTIRE_PERFORMED , LCD1602_SHIFT_ENTIRE_NOT_PERFORMED , LCD1602_SHIFT_ENTIRE_MAX }

    Shift types of entire display.
- enum lcd1602_type_interface { LCD1602_INTERFACE_UNDEF = -1 , LCD1602_INTERFACE_4BITS , LCD1602_INTERFACE_8BITS , LCD1602_INTERFACE_MAX }

    Type of LCD1602 interfaces.
- enum lcd1602_disp_state { LCD1602_DISPLAY_UNDEF = -1 , LCD1602_DISPLAY_OFF , LCD1602_DISPLAY_ON , LCD1602_DISPLAY_MAX }

    Display states.
- enum lcd1602_cursor_state { LCD1602_CURSOR_UNDEF = -1 , LCD1602_CURSOR_OFF , LCD1602_CURSOR_ON , LCD1602_CURSOR_MAX }

    Cursor states.
- enum lcd1602_cursor_blink_state { LCD1602_CURSOR_BLINK_UNDEF = -1 , LCD1602_CURSOR_BLINK_OFF , LCD1602_CURSOR_BLINK_ON , LCD1602_CURSOR_BLINK_MAX }

    Cursor blink states.

Functions

- uint8_t bsp_lcd1602_init (struct lcd1602_settings ∗init_settings)
- uint8_t bsp_lcd1602_deinit (void)
- uint8_t bsp_lcd1602_printf (const char ∗line1, const char ∗line2,...)
- uint8_t bsp_lcd1602_cprintf (const char ∗line1, const char ∗line2,...)
- uint8_t bsp_lcd1602_ddram_address_set (const uint8_t address)
- uint8_t bsp_lcd1602_cgram_address_set (const uint8_t address)
- uint8_t bsp_lcd1602_function_set (const enum lcd1602_type_interface interface, const enum lcd1602_num_line num_line, const enum lcd1602_font_size font_size)
- uint8_t bsp_lcd1602_cursor_disp_shift (const enum lcd1602_type_shift shift)
- uint8_t bsp_lcd1602_display_on_off (const enum lcd1602_disp_state disp_state, const enum lcd1602_cursor_state cursor_state, const enum lcd1602_cursor_blink_state cursor_blink_state)
- uint8_t bsp_lcd1602_entry_mode_set (const enum lcd1602_type_move_cursor cursor, const enum lcd1602_shift_entire_disp shift_entire)
- uint8_t bsp_lcd1602_return_home (void)
- uint8_t bsp_lcd1602_display_clear (void)

6.24.1 Detailed Description

Header of BSP LCD1602 module.

Author

JavaLandau

Copyright

MIT License

6.25 bsp_lcd1602.h

Go to the documentation of this file.

```
#ifndef __BSP_LCD1602_H__
#define __BSP_LCD1602_H__

#include <stdint.h>

enum lcd1602_type_shift {
    LCD1602_SHIFT_CURSOR_UNDEF = -1,
    LCD1602_SHIFT_CURSOR_LEFT,
    LCD1602_SHIFT_CURSOR_RIGHT,
    LCD1602_SHIFT_DISPLAY_LEFT,
    LCD1602_SHIFT_DISPLAY_RIGHT,
    LCD1602_SHIFT_MAX
};

enum lcd1602_num_line {
    LCD1602_NUM_LINE_UNDEF = -1,
    LCD1602_NUM_LINE_1,
    LCD1602_NUM_LINE_2,
    LCD1602_NUM_LINE_MAX
};

enum lcd1602_font_size {
    LCD1602_FONT_SIZE_UNDEF = -1,
    LCD1602_FONT_SIZE_5X8,
    LCD1602_FONT_SIZE_5X11,
    LCD1602_FONT_SIZE_MAX
};

enum lcd1602_type_move_cursor {
    LCD1602_CURSOR_MOVE_UNDEF = -1,
    LCD1602_CURSOR_MOVE_LEFT,
    LCD1602_CURSOR_MOVE_RIGHT,
    LCD1602_CURSOR_MOVE_MAX
};

enum lcd1602_shift_entire_disp {
    LCD1602_SHIFT_ENTIRE_UNDEF = -1,
    LCD1602_SHIFT_ENTIRE_PERFORMED,
    LCD1602_SHIFT_ENTIRE_NOT_PERFORMED,
    LCD1602_SHIFT_ENTIRE_MAX
};

enum lcd1602_type_interface {
    LCD1602_INTERFACE_UNDEF = -1,
    LCD1602_INTERFACE_4BITS,
    LCD1602_INTERFACE_8BITS,
    LCD1602_INTERFACE_MAX
};


enum lcd1602_disp_state {
    LCD1602_DISPLAY_UNDEF = -1,
    LCD1602_DISPLAY_OFF,
    LCD1602_DISPLAY_ON,
    LCD1602_DISPLAY_MAX
};

enum lcd1602_cursor_state {
    LCD1602_CURSOR_UNDEF = -1,
    LCD1602_CURSOR_OFF,
    LCD1602_CURSOR_ON,
    LCD1602_CURSOR_MAX
};

enum lcd1602_cursor_blink_state {
    LCD1602_CURSOR_BLINK_UNDEF = -1,
    LCD1602_CURSOR_BLINK_OFF,
    LCD1602_CURSOR_BLINK_ON,
    LCD1602_CURSOR_BLINK_MAX
};

struct lcd1602_settings {
    enum lcd1602_num_line              num_line;
    enum lcd1602_font_size             font_size;
    enum lcd1602_type_move_cursor      type_move_cursor;
    enum lcd1602_shift_entire_disp     shift_entire_disp;
    enum lcd1602_type_interface        type_interface;
    enum lcd1602_disp_state            disp_state;
    enum lcd1602_cursor_state          cursor_state;
    enum lcd1602_cursor_blink_state    cursor_blink_state;
};

uint8_t bsp_lcd1602_init(struct lcd1602_settings *init_settings);

uint8_t bsp_lcd1602_deinit(void);

uint8_t bsp_lcd1602_printf(const char *line1, const char *line2, ...);

uint8_t bsp_lcd1602_cprintf(const char *line1, const char *line2, ...);

uint8_t bsp_lcd1602_ddram_address_set(const uint8_t address);

uint8_t bsp_lcd1602_cgram_address_set(const uint8_t address);

uint8_t bsp_lcd1602_function_set(const enum lcd1602_type_interface interface,
                                 const enum lcd1602_num_line num_line,
                                 const enum lcd1602_font_size font_size);

uint8_t bsp_lcd1602_cursor_disp_shift(const enum lcd1602_type_shift shift);

uint8_t bsp_lcd1602_display_on_off(const enum lcd1602_disp_state disp_state,
                                   const enum lcd1602_cursor_state cursor_state,
                                   const enum lcd1602_cursor_blink_state cursor_blink_state);

uint8_t bsp_lcd1602_entry_mode_set(const enum lcd1602_type_move_cursor cursor,
                                   const enum lcd1602_shift_entire_disp shift_entire);

uint8_t bsp_lcd1602_return_home(void);

uint8_t bsp_lcd1602_display_clear(void);

#endif //__BSP_LCD1602_H__
```

6.26 bsp_led_rgb.h File Reference

Header of BSP LED RGB module.

```
#include <stdint.h>
#include <stdbool.h>
#include "stm32f4xx_hal.h"
#include "bsp_gpio.h"
#include "common.h"
```

Data Structures

- struct bsp_led_rgb

  RGB LED structure.
- struct bsp_led_pwm

  Parameters of RGB LED blinking.

Macros

- #define BSP_LED_RGB_HARDFAULT()

Functions

- uint8_t bsp_led_rgb_calibrate (const struct bsp_led_rgb ∗coef_rgb)
- uint8_t bsp_led_rgb_set (const struct bsp_led_rgb ∗rgb)
- uint8_t bsp_led_rgb_init (void)
- uint8_t bsp_led_rgb_deinit (void)
- uint8_t bsp_led_rgb_blink_enable (const struct bsp_led_pwm ∗pwm)
- uint8_t bsp_led_rgb_blink_disable (void)

6.26.1 Detailed Description

Header of BSP LED RGB module.

Author

JavaLandau

Copyright

MIT License

6.27 bsp_led_rgb.h

Go to the documentation of this file.

```
#ifndef __BSP_LED_RGB_H__
#define __BSP_LED_RGB_H__

 #include <stdint.h>
 #include <stdbool.h>
 #include "stm32f4xx_hal.h"
 #include "bsp_gpio.h"
 #include "common.h"

 struct bsp_led_rgb {
     uint8_t r;
     uint8_t g;
     uint8_t b;
 };

 struct bsp_led_pwm {
     uint32_t width_on_ms;
     uint32_t width_off_ms;
 };

 uint8_t bsp_led_rgb_calibrate(const struct bsp_led_rgb *coef_rgb);

 uint8_t bsp_led_rgb_set(const struct bsp_led_rgb *rgb);

 uint8_t bsp_led_rgb_init(void);

 uint8_t bsp_led_rgb_deinit(void);

 uint8_t bsp_led_rgb_blink_enable(const struct bsp_led_pwm *pwm);

 uint8_t bsp_led_rgb_blink_disable(void);

 #define BSP_LED_RGB_HARDFAULT() \
 do {\
 BSP_GPIO_FORCE_OUTPUT_MODE(GPIOA, 8);\
 BSP_GPIO_FORCE_OUTPUT_MODE(GPIOA, 9);\
 BSP_GPIO_FORCE_OUTPUT_MODE(GPIOA, 10);\
 \
 BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_8, false);\
 BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_10, false);\
 \
 while (true) {\
 BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_9, false);\
 INSTR_DELAY_US(100000);\
 BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_9, true);\
 INSTR_DELAY_US(100000);\
 }\
 } while (0)

 #endif //__BSP_LED_RGB_H__
```

6.28 bsp_rcc.h File Reference

Header of BSP RCC module.

```
#include <stdint.h>
#include "stm32f4xx_hal.h"
```

Macros

- #define TIM_APB_NUM_CLOCK_GET(INSTANCE)

Functions

- uint8_t bsp_rcc_main_config_init (void)
- uint32_t bsp_rcc_apb_timer_freq_get (TIM_TypeDef ∗instance)

6.28.1 Detailed Description

Header of BSP RCC module.

Author

JavaLandau

Copyright

MIT License

6.29 bsp_rcc.h

Go to the documentation of this file.

```
#ifndef __BSP_RCC_H__
#define __BSP_RCC_H__

#include <stdint.h>
#include "stm32f4xx_hal.h"

#define TIM_APB_NUM_CLOCK_GET(INSTANCE)  \
((IS_TIM_INSTANCE(INSTANCE)) ? (\
(((INSTANCE) == TIM1) || \
((INSTANCE) == TIM8) || \
((INSTANCE) == TIM9) || \
((INSTANCE) == TIM10) || \
((INSTANCE) == TIM11)) ?  2 :  1) :  0)

uint8_t bsp_rcc_main_config_init(void);

uint32_t bsp_rcc_apb_timer_freq_get(TIM_TypeDef *instance);

#endif //__BSP_RCC_H__
```

6.30 bsp_uart.h File Reference

Header of BSP UART module.

```
#include <stdint.h>
#include <stdbool.h>
#include "stm32f4xx_hal.h"
```

Data Structures

- struct uart_init_ctx

  BSP UART initializing context.

Macros

- #define UART_TYPE_VALID(X) (((uint32_t)(X) < BSP_UART_TYPE_MAX))
- #define UART_WORDLEN_VALID(X) (((X) == BSP_UART_WORDLEN_8) || ((X) == BSP_UART_WORDLEN_9))
- #define UART_PARITY_VALID(X) (((X) == BSP_UART_PARITY_NONE) || ((X) == BSP_UART_PARITY_EVEN) || ((X) == BSP_UART_PARITY_ODD))
- #define UART_STOPBITS_VALID(X) (((X) == BSP_UART_STOPBITS_1) || ((X) == BSP_UART_STOPBITS_2))
- #define **BSP_UART_ERROR_PE** HAL_UART_ERROR_PE

  BSP UART parity error.
- #define **BSP_UART_ERROR_NE** HAL_UART_ERROR_NE

  BSP UART noise error.
- #define **BSP_UART_ERROR_FE** HAL_UART_ERROR_FE

  BSP UART frame error.
- #define **BSP_UART_ERROR_ORE** HAL_UART_ERROR_ORE

  BSP UART overrun error.
- #define **BSP_UART_ERROR_DMA** HAL_UART_ERROR_DMA

  BSP UART DMA error.
- #define **BSP_UART_ERRORS_ALL** (BSP_UART_ERROR_PE | BSP_UART_ERROR_NE | BSP_UART_ERROR_FE | BSP_UART_ERROR_ORE | BSP_UART_ERROR_DMA)

  Mask including all possible BSP UART errors.

Enumerations

- enum uart_type { BSP_UART_TYPE_CLI = 0 , BSP_UART_TYPE_RS232_TX , BSP_UART_TYPE_RS232_RX , BSP_UART_TYPE_MAX }

  Types of BSP UART instances.
- enum uart_wordlen { BSP_UART_WORDLEN_8 = 8 , BSP_UART_WORDLEN_9 = 9 }

  BSP UART word length.
- enum uart_parity { BSP_UART_PARITY_NONE = 0 , BSP_UART_PARITY_EVEN = 1 , BSP_UART_PARITY_ODD = 2 }

  BSP UART parity types.
- enum uart_stopbits { BSP_UART_STOPBITS_1 = 1 , BSP_UART_STOPBITS_2 = 2 }

  BSP UART stop bits count.

Functions

- uint8_t bsp_uart_init (enum uart_type type, struct uart_init_ctx ∗init)
- uint8_t bsp_uart_deinit (enum uart_type type)
- uint8_t bsp_uart_read (enum uart_type type, uint8_t ∗data, uint16_t ∗len, uint32_t tmt_ms)
- uint8_t bsp_uart_write (enum uart_type type, uint8_t ∗data, uint16_t len, uint32_t tmt_ms)
- uint8_t bsp_uart_start (enum uart_type type)
- uint8_t bsp_uart_stop (enum uart_type type)
- bool bsp_uart_is_started (enum uart_type type)
- bool bsp_uart_rx_buffer_is_empty (enum uart_type type)

6.30.1 Detailed Description

Header of BSP UART module.

Author

JavaLandau

Copyright

MIT License

6.31 bsp_uart.h

Go to the documentation of this file.

```
#ifndef __BSP_UART_H__
#define __BSP_UART_H__

#include <stdint.h>
#include <stdbool.h>
#include "stm32f4xx_hal.h"

#define UART_TYPE_VALID(X)       (((uint32_t)(X) < BSP_UART_TYPE_MAX))

#define UART_WORDLEN_VALID(X)    (((X) == BSP_UART_WORDLEN_8) || ((X) == BSP_UART_WORDLEN_9))

#define UART_PARITY_VALID(X)     (((X) == BSP_UART_PARITY_NONE) || ((X) == BSP_UART_PARITY_EVEN) || ((X)
      == BSP_UART_PARITY_ODD))

#define UART_STOPBITS_VALID(X)   (((X) == BSP_UART_STOPBITS_1) || ((X) == BSP_UART_STOPBITS_2))

```

```
#define BSP_UART_ERROR_PE       HAL_UART_ERROR_PE
#define BSP_UART_ERROR_NE       HAL_UART_ERROR_NE
#define BSP_UART_ERROR_FE       HAL_UART_ERROR_FE
#define BSP_UART_ERROR_ORE      HAL_UART_ERROR_ORE
#define BSP_UART_ERROR_DMA      HAL_UART_ERROR_DMA

#define BSP_UART_ERRORS_ALL     (BSP_UART_ERROR_PE | BSP_UART_ERROR_NE | BSP_UART_ERROR_FE |
   BSP_UART_ERROR_ORE | BSP_UART_ERROR_DMA)

enum uart_type {
    BSP_UART_TYPE_CLI = 0,
    BSP_UART_TYPE_RS232_TX,
    BSP_UART_TYPE_RS232_RX,
    BSP_UART_TYPE_MAX
};

enum uart_wordlen {
    BSP_UART_WORDLEN_8 = 8,
    BSP_UART_WORDLEN_9 = 9
};

enum uart_parity {
    BSP_UART_PARITY_NONE = 0,
    BSP_UART_PARITY_EVEN = 1,
    BSP_UART_PARITY_ODD = 2
};

enum uart_stopbits {
    BSP_UART_STOPBITS_1 = 1,
    BSP_UART_STOPBITS_2 = 2
};

struct uart_init_ctx {
    uint32_t baudrate;
    uint32_t tx_size;
    uint32_t rx_size;
    bool lin_enabled;
    enum uart_wordlen wordlen;
    enum uart_parity parity;
    enum uart_stopbits stopbits;
    void (*error_isr_cb)(enum uart_type type, uint32_t error, void *params);
    void (*overflow_isr_cb)(enum uart_type type, void *params);
    void (*lin_break_isr_cb)(enum uart_type type, void *params);
    void *params;
};

uint8_t bsp_uart_init(enum uart_type type, struct uart_init_ctx *init);

uint8_t bsp_uart_deinit(enum uart_type type);

uint8_t bsp_uart_read(enum uart_type type, uint8_t *data, uint16_t *len, uint32_t tmt_ms);

uint8_t bsp_uart_write(enum uart_type type, uint8_t *data, uint16_t len, uint32_t tmt_ms);

uint8_t bsp_uart_start(enum uart_type type);

uint8_t bsp_uart_stop(enum uart_type type);

bool bsp_uart_is_started(enum uart_type type);

bool bsp_uart_rx_buffer_is_empty(enum uart_type type);

#endif //__BSP_UART_H__
```

6.32 bsp_button.c File Reference

BSP button module.

```
#include "common.h"
#include "bsp_button.h"
#include "bsp_rcc.h"
#include "bsp_gpio.h"
#include "stm32f4xx_ll_tim.h"
#include <stdbool.h>
```

Macros

- #define **BUTTON_TIM_FREQ** (10000)

  Frequency of htim.
- #define TIM_TICK_TO_MS(X) ((1000 ∗ (X)) / BUTTON_TIM_FREQ)
- #define TIM_PERIOD_CALC(X) ((BUTTON_TIM_FREQ ∗ (X)) / 1000)

Functions

- static void __button_tim_msp_init (TIM_HandleTypeDef ∗htim)
- static void __button_tim_msp_deinit (TIM_HandleTypeDef ∗htim)
- static void __button_tim_period_elapsed_callback (TIM_HandleTypeDef ∗htim)
- static bool __button_tim_is_started (void)
- static uint8_t __button_tim_stop (void)
- static uint8_t __button_tim_start (uint32_t period_ms)
- uint8_t bsp_button_init (struct button_init_ctx ∗init_ctx)
- uint8_t bsp_button_deinit (void)
- void EXTI4_IRQHandler (void)
- void TIM7_IRQHandler (void)

Variables

- static EXTI_HandleTypeDef **hexti** = {.Line = EXTI_LINE_4}

  STM32 HAL EXTI instance, used to detect pushing and releasing actions on the button.
- static TIM_HandleTypeDef **htim** = {.Instance = TIM7}

  STM32 HAL TIM instance, used to detect long pressing and filter contact bounce.
- static struct button_init_ctx **ctx** = {0}

  BSP button context.
- static bool **button_pressed** = false

  Current state of the button: true - pressed, false - not.
- static bool **is_long_action** = false

  Flag whether button timer is checking of long press action on the button.

6.32.1 Detailed Description

BSP button module.

Author

JavaLandau

Copyright

MIT License

The file includes implementation of BSP layer of the button

6.33 bsp_crc.c File Reference

BSP CRC module.

```
#include "common.h"
#include "bsp_crc.h"
#include <string.h>
```

Functions

- void HAL_CRC_MspInit (CRC_HandleTypeDef ∗hcrc)
- void HAL_CRC_MspDeInit (CRC_HandleTypeDef ∗hcrc)
- uint8_t bsp_crc_init (void)
- uint8_t bsp_crc_deinit (void)
- uint8_t bsp_crc_calc (uint8_t ∗data, uint32_t len, uint32_t ∗result)

Variables

- static CRC_HandleTypeDef **crc_module** = {.Instance = CRC}

  STM32 HAL CRC instance.

6.33.1 Detailed Description

BSP CRC module.

Author

JavaLandau

Copyright

MIT License

The file includes implementation of BSP layer of the CRC

6.34 bsp_gpio.c File Reference

BSP GPIO module.

```
#include "common.h"
#include "bsp_gpio.h"
```

Functions

- uint8_t bsp_gpio_bulk_read (GPIO_TypeDef ∗gpiox, const uint16_t ∗gpio_pins, uint16_t ∗gpio_states)
- uint8_t bsp_gpio_bulk_write (GPIO_TypeDef ∗gpiox, const uint16_t ∗gpio_pins, const uint16_t gpio_states)

6.34.1 Detailed Description

BSP GPIO module.

Author

JavaLandau

Copyright

MIT License

The file includes implementation of BSP layer of the GPIO

6.35 bsp_lcd1602.c File Reference

BSP LCD1602 module.

```
#include <stdarg.h>
#include <string.h>
#include <stdio.h>
#include "bsp_lcd1602.h"
#include "bsp_gpio.h"
#include "common.h"
#include "stm32f4xx_hal.h"
#include <stdbool.h>
```

Macros

- #define **MAX_CGRAM_ADDRESS** 0x3F

  Maximum address of CGRAM memory.
- #define **MAX_DDRAM_ADDRESS** 0x7F

  Maximum address of DDRAM memory.
- #define **LCD1602_LENGTH_LINE** 16

  Length of the line of LCD1602 in symbols.
- #define **LCD1602_MAX_STR_LEN** (4 $*$ LCD1602_LENGTH_LINE)

  Maximum length of buffered string used within the module.
- #define **LCD1602_DDRAM_START_LINE1** 0x00

  DDRAM address of start of first line.
- #define **LCD1602_DDRAM_END_LINE1** 0x27

  DDRAM address of end of first line (display is used in 2-line mode)
- #define **LCD1602_DDRAM_START_LINE2** 0x40

  DDRAM address of start of second line.
- #define **LCD1602_DDRAM_END_LINE2** 0x67

  DDRAM address of end of second line.
- #define **LCD1602_INSTR_REG** 0x0

  Level on signal RS to choose instruction register.
- #define **LCD1602_DATA_REG** 0x1

Level on signal RS to choose data register.

- #define **LCD1602_READ_MODE** 0x1

  Level on signal R/W to set read mode.
- #define **LCD1602_WRITE_MODE** 0x0

  Level on signal R/W to set write mode.
- #define **TIME_FOR_DELAY** 1

  Time delay in us while waiting for BUSY flag, used in __lcd1602_wait.
- #define **WAIT_TMT** 500

  Timeout in ms for waiting for BUSY flag.
- #define TYPE_SHIFT_IS_VALID(X) (((uint8_t)(X)) < LCD1602_SHIFT_MAX)
- #define NUM_LINE_IS_VALID(X) (((uint8_t)(X)) < LCD1602_NUM_LINE_MAX)
- #define FONT_SIZE_IS_VALID(X) (((uint8_t)(X)) < LCD1602_FONT_SIZE_MAX)
- #define TYPE_MOVE_CURSOR_IS_VALID(X) (((uint8_t)(X)) < LCD1602_CURSOR_MOVE_MAX)
- #define SHIFT_ENTIRE_IS_VALID(X) (((uint8_t)(X)) < LCD1602_SHIFT_ENTIRE_MAX)
- #define TYPE_INTERFACE_IS_VALID(X) (((uint8_t)(X)) < LCD1602_INTERFACE_MAX)
- #define DISP_STATE_IS_VALID(X) (((uint8_t)(X)) < LCD1602_DISPLAY_MAX)
- #define CURSOR_STATE_IS_VALID(X) (((uint8_t)(X)) < LCD1602_CURSOR_MAX)
- #define CURSOR_BLINK_STATE_IS_VALID(X) (((uint8_t)(X)) < LCD1602_CURSOR_BLINK_MAX)
- #define LCD1602_DATA_PINS

  All mixed GPIO pins from lcd1602_data_pins, used for (de-)initalizating purposes.

Functions

- static uint8_t __lcd1602_read_write (uint8_t ∗data, uint8_t type_reg, uint8_t type_mode)
- static uint8_t __lcd1602_instruction_write (uint8_t instruction)
- static uint8_t __lcd1602_read_busy_flag (uint8_t ∗busy_flag, uint8_t ∗address_counter)
- static uint8_t __lcd1602_data_write (uint8_t data)
- static uint8_t __lcd1602_wait (const uint32_t timeout)
- uint8_t bsp_lcd1602_function_set (const enum lcd1602_type_interface interface, const enum lcd1602_num_line num_line, const enum lcd1602_font_size font_size)
- uint8_t bsp_lcd1602_init (struct lcd1602_settings ∗init_settings)
- uint8_t bsp_lcd1602_deinit (void)
- uint8_t bsp_lcd1602_display_clear (void)
- uint8_t bsp_lcd1602_return_home (void)
- uint8_t bsp_lcd1602_entry_mode_set (const enum lcd1602_type_move_cursor cursor, const enum lcd1602_shift_entire_disp shift_entire)
- uint8_t bsp_lcd1602_display_on_off (const enum lcd1602_disp_state disp_state, const enum lcd1602_cursor_state cursor_state, const enum lcd1602_cursor_blink_state cursor_blink_state)
- uint8_t bsp_lcd1602_cursor_disp_shift (const enum lcd1602_type_shift shift)
- uint8_t bsp_lcd1602_cgram_address_set (const uint8_t address)
- uint8_t bsp_lcd1602_ddram_address_set (const uint8_t address)
- static uint8_t __lcd1602_printf (const char ∗line1, const char ∗line2, bool is_centered, va_list argp)
- uint8_t bsp_lcd1602_printf (const char ∗line1, const char ∗line2,...)
- uint8_t bsp_lcd1602_cprintf (const char ∗line1, const char ∗line2,...)

Variables

- static const uint16_t lcd1602_data_pins [ ]

  Array of GPIO pins used for 8-bit parallel interface.
- static struct lcd1602_settings **settings**

  Local copy of display settings.

6.35.1 Detailed Description

BSP LCD1602 module.

Author

JavaLandau

Copyright

MIT License

The file includes implementation of BSP layer of the LCD1602 display

6.36 bsp_led_rgb.c File Reference

BSP LED RGB module.

```
#include "common.h"
#include "bsp_led_rgb.h"
#include "bsp_rcc.h"
#include "stm32f4xx_ll_tim.h"
#include <stdbool.h>
```

Macros

- #define **RGB_TIM_FREQ** 1000

  Frequency of RGB timer in Hz.
- #define **RGB_TIM_PERIOD** UINT16_MAX

  Value of period register of RGB timer.
- #define **BLINK_TIM_FREQ** 2000

  Frequency of blink timer in Hz.

Functions

- static void __led_rgb_tim_pwm_msp_init (TIM_HandleTypeDef ∗htim)
- static void __led_rgb_tim_pwm_msp_deinit (TIM_HandleTypeDef ∗htim)
- static void __led_rgb_tim_msp_post_init (void)
- static void __led_rgb_tim_msp_prev_deinit (void)
- static void __led_rgb_blink_tim_period_elapsed_callback (TIM_HandleTypeDef ∗htim)
- static void __led_rgb_blink_tim_pwm_pulse_finished_callback (TIM_HandleTypeDef ∗htim)
- static uint8_t __led_rgb_blink_start (void)
- static uint8_t __led_rgb_blink_stop (void)
- static bool __led_rgb_blink_is_started (void)
- uint8_t bsp_led_rgb_init (void)
- uint8_t bsp_led_rgb_deinit (void)
- uint8_t bsp_led_rgb_calibrate (const struct bsp_led_rgb ∗coef_rgb)
- uint8_t bsp_led_rgb_set (const struct bsp_led_rgb ∗rgb)
- uint8_t bsp_led_rgb_blink_enable (const struct bsp_led_pwm ∗pwm)
- uint8_t bsp_led_rgb_blink_disable (void)
- void TIM2_IRQHandler (void)

Variables

- static TIM_HandleTypeDef **htim_rgb** = {.Instance = TIM1}

  STM32 HAL TIM instance of RGB timer.
- static TIM_HandleTypeDef **htim_blink** = {.Instance = TIM2}

  STM32 HAL TIM instance of blink timer.
- static uint32_t **led_rgb_tim_channels** [ ] = {TIM_CHANNEL_1, TIM_CHANNEL_2, TIM_CHANNEL_3}

  Array of STM32 HAL TIM channels.
- static float **coef_r** = 1.0f

  Corrective coefficient for red channel, set by bsp_led_rgb_calibrate.
- static float **coef_g** = 1.0f

  Corrective coefficient for green channel, set by bsp_led_rgb_calibrate.
- static float **coef_b** = 1.0f

  Corrective coefficient for blue channel, set by bsp_led_rgb_calibrate.

6.36.1 Detailed Description

BSP LED RGB module.

Author

JavaLandau

Copyright

MIT License

The file includes implementation of BSP layer of the LED RGB

6.37 bsp_rcc.c File Reference

BSP RCC module.

```
#include "common.h"
#include "bsp_rcc.h"
#include "stm32f4xx_ll_rcc.h"
```

Functions

- uint8_t bsp_rcc_main_config_init (void)
- uint32_t bsp_rcc_apb_timer_freq_get (TIM_TypeDef ∗instance)

6.37.1 Detailed Description

BSP RCC module.

Author

JavaLandau

Copyright

MIT License

The file includes implementation of BSP layer of the RCC

6.38 bsp_uart.c File Reference

BSP UART module.

```
#include "common.h"
#include "bsp_uart.h"
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "stm32f4xx_ll_usart.h"
```

Data Structures

- struct uart_ctx

  Context of the BSP UART instance.

Macros

- #define HAL_UART_WORDLEN_TO(X) (((X) == BSP_UART_WORDLEN_8) ? UART_WORDLENGTH_8B : UART_WORDLENGTH_9B)
- #define HAL_UART_STOPBITS_TO(X) (((X) == BSP_UART_STOPBITS_1) ? UART_STOPBITS_1 ↩ : UART_STOPBITS_2)
- #define HAL_UART_PARITY_TO(X)

Functions

- static enum uart_type __uart_type_get (USART_TypeDef ∗instance)
- static uint8_t __uart_dma_deinit (enum uart_type type)
- static uint8_t __uart_msp_deinit (enum uart_type type)
- static uint8_t __uart_dma_init (enum uart_type type)
- static uint8_t __uart_msp_init (enum uart_type type)
- static void __uart_rx_callback (UART_HandleTypeDef ∗huart, uint16_t pos)
- static void __uart_error_callback (enum uart_type type, uint32_t error)
- uint8_t bsp_uart_start (enum uart_type type)
- uint8_t bsp_uart_stop (enum uart_type type)
- bool bsp_uart_is_started (enum uart_type type)
- uint8_t bsp_uart_write (enum uart_type type, uint8_t ∗data, uint16_t len, uint32_t tmt_ms)
- bool bsp_uart_rx_buffer_is_empty (enum uart_type type)
- uint8_t bsp_uart_read (enum uart_type type, uint8_t ∗data, uint16_t ∗len, uint32_t tmt_ms)
- uint8_t bsp_uart_init (enum uart_type type, struct uart_init_ctx ∗init)
- uint8_t bsp_uart_deinit (enum uart_type type)
- static void __uart_irq_handler (enum uart_type type)
- void UART4_IRQHandler (void)
- void USART2_IRQHandler (void)
- void USART3_IRQHandler (void)
- void DMA1_Stream1_IRQHandler (void)
- void DMA1_Stream2_IRQHandler (void)
- void DMA1_Stream4_IRQHandler (void)
- void DMA1_Stream5_IRQHandler (void)

Variables

- struct {
  UART_HandleTypeDef **uart**
  STM32 HAL UART instance.
  struct uart_ctx ∗ **ctx**
  Context of the instance.
  } uart_obj [BSP_UART_TYPE_MAX]

6.38.1 Detailed Description

BSP UART module.

Author

JavaLandau

Copyright

MIT License

The file includes implementation of BSP layer of the UART

6.39 common.h File Reference

Common utils.

Macros

- #define **RES_OK** 0

  Return code: Success.
- #define **RES_NOK** 1

  Return code: Not specified error.
- #define **RES_INVALID_PAR** 2

  Return code: Invalid input parameter(-s)
- #define **RES_MEMORY_ERR** 3

  Return code: Memory allocation error.
- #define **RES_TIMEOUT** 4

  Return code: Timeout occured.
- #define **RES_NOT_SUPPORTED** 5

  Return code: Some feature is not supported.
- #define **RES_OVERFLOW** 6

  Return code: Overflow of an object.
- #define **RES_NOT_INITIALIZED** 7

  Return code: An object is not initialized.
- #define **RES_NOT_ALLOWED** 8

  Return code: An object/feature is not allowed.
- #define ARRAY_SIZE(X) (sizeof(X) / sizeof(X[0]))
- #define MIN(X, Y) (((X) < (Y)) ? (X) : (Y))
- #define MAX(X, Y) (((X) > (Y)) ? (X) : (Y))
- #define IS_PRINTABLE(X) (X >= ' ' && X <= '~')
- #define INSTR_DELAY_US(DELAY)

6.39.1 Detailed Description

Common utils.

Author

JavaLandau

Copyright

MIT License

The file includes basic utils, defines and macros

6.40 common.h

Go to the documentation of this file.

```
#ifndef __COMMON_H__
#define __COMMON_H__

#define RES_OK                  0
#define RES_NOK                 1
#define RES_INVALID_PAR         2
#define RES_MEMORY_ERR          3
#define RES_TIMEOUT             4
#define RES_NOT_SUPPORTED       5
#define RES_OVERFLOW            6
#define RES_NOT_INITIALIZED     7
#define RES_NOT_ALLOWED         8

#define ARRAY_SIZE(X)       (sizeof(X) / sizeof(X[0]))

#define MIN(X, Y)           (((X) < (Y)) ? (X) : (Y))

#define MAX(X, Y)           (((X) > (Y)) ? (X) : (Y))

#define IS_PRINTABLE(X)     (X >= ' ' && X <= '~')

#define INSTR_DELAY_US(DELAY)   \
do {\
__IO uint32_t clock_delay = DELAY * (HAL_RCC_GetSysClockFreq() / 8 / 1000000);\
do {\
__NOP();\
} while (--clock_delay);\
} while (0)

#endif
```

Index