

Sniffer RS-232

1.0

Generated by Doxygen 1.9.5

1 Module Index	1
1.1 Modules	1
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	5
3.1 File List	5
4 Module Documentation	7
4.1 BSP button	7
4.1.1 Detailed Description	8
4.1.2 Macro Definition Documentation	8
4.1.2.1 TIM_PERIOD_CALC	8
4.1.2.2 TIM_TICK_TO_MS	8
4.1.3 Enumeration Type Documentation	9
4.1.3.1 button_action	9
4.1.4 Function Documentation	9
4.1.4.1 __button_tim_is_started()	9
4.1.4.2 __button_tim_msp_deinit()	10
4.1.4.3 __button_tim_msp_init()	11
4.1.4.4 __button_tim_period_elapsed_callback()	11
4.1.4.5 __button_tim_start()	11
4.1.4.6 __button_tim_stop()	12
4.1.4.7 bsp_button_deinit()	12
4.1.4.8 bsp_button_init()	12
4.1.4.9 EXTI4_IRQHandler()	13
4.1.4.10 TIM7_IRQHandler()	13
4.2 BSP CRC	13
4.2.1 Detailed Description	13
4.2.2 Function Documentation	13
4.2.2.1 bsp_crc_calc()	13
4.2.2.2 bsp_crc_deinit()	14
4.2.2.3 bsp_crc_init()	14
4.2.2.4 HAL_CRC_MspDeInit()	14
4.2.2.5 HAL_CRC_MspInit()	15
4.3 BSP GPIO	15
4.3.1 Detailed Description	15
4.3.2 Macro Definition Documentation	15
4.3.2.1 BSP_GPIO_FORCE_OUTPUT_MODE	15
4.3.2.2 BSP_GPIO_PORT_READ	16
4.3.2.3 BSP_GPIO_PORT_WRITE	16
4.3.3 Function Documentation	16

4.3.3.1 bsp_gpio_bulk_read()	17
4.3.3.2 bsp_gpio_bulk_write()	17
4.4 BSP LCD1602	17
4.4.1 Detailed Description	20
4.4.2 Macro Definition Documentation	20
4.4.2.1 CURSOR_BLINK_STATE_IS_VALID	20
4.4.2.2 CURSOR_STATE_IS_VALID	20
4.4.2.3 DISP_STATE_IS_VALID	21
4.4.2.4 FONT_SIZE_IS_VALID	21
4.4.2.5 LCD1602_DATA_PINS	21
4.4.2.6 NUM_LINE_IS_VALID	22
4.4.2.7 SHIFT_ENTIRE_IS_VALID	22
4.4.2.8 TYPE_INTERFACE_IS_VALID	22
4.4.2.9 TYPE_MOVE_CURSOR_IS_VALID	23
4.4.2.10 TYPE_SHIFT_IS_VALID	23
4.4.3 Enumeration Type Documentation	23
4.4.3.1 lcd1602_cursor_blink_state	24
4.4.3.2 lcd1602_cursor_state	25
4.4.3.3 lcd1602_disp_state	25
4.4.3.4 lcd1602_font_size	25
4.4.3.5 lcd1602_num_line	26
4.4.3.6 lcd1602_shift_entire_disp	26
4.4.3.7 lcd1602_type_interface	26
4.4.3.8 lcd1602_type_move_cursor	27
4.4.3.9 lcd1602_type_shift	27
4.4.4 Function Documentation	27
4.4.4.1 __lcd1602_data_write()	27
4.4.4.2 __lcd1602_instruction_write()	28
4.4.4.3 __lcd1602_printf()	28
4.4.4.4 __lcd1602_read_busy_flag()	29
4.4.4.5 __lcd1602_read_write()	29
4.4.4.6 __lcd1602_wait()	29
4.4.4.7 bsp_lcd1602_cgram_address_set()	30
4.4.4.8 bsp_lcd1602_cprintf()	30
4.4.4.9 bsp_lcd1602_cursor_disp_shift()	31
4.4.4.10 bsp_lcd1602_ddram_address_set()	31
4.4.4.11 bsp_lcd1602_deinit()	31
4.4.4.12 bsp_lcd1602_display_clear()	32
4.4.4.13 bsp_lcd1602_display_on_off()	32
4.4.4.14 bsp_lcd1602_entry_mode_set()	32
4.4.4.15 bsp_lcd1602_function_set()	33
4.4.4.16 bsp_lcd1602_init()	33

4.4.4.17 bsp_lcd1602_printf()	34
4.4.4.18 bsp_lcd1602_return_home()	34
4.4.5 Variable Documentation	34
4.4.5.1 lcd1602_data_pins	35
4.5 BSP LED RGB	35
4.5.1 Detailed Description	36
4.5.2 Macro Definition Documentation	36
4.5.2.1 BSP_LED_RGB_HARDFAULT	36
4.5.3 Function Documentation	37
4.5.3.1 __led_rgb_blink_is_started()	37
4.5.3.2 __led_rgb_blink_start()	37
4.5.3.3 __led_rgb_blink_stop()	37
4.5.3.4 __led_rgb_blink_tim_period_elapsed_callback()	37
4.5.3.5 __led_rgb_blink_tim_pwm_pulse_finished_callback()	38
4.5.3.6 __led_rgb_tim_msp_post_init()	38
4.5.3.7 __led_rgb_tim_msp_prev_deinit()	38
4.5.3.8 __led_rgb_tim_pwm_msp_deinit()	38
4.5.3.9 __led_rgb_tim_pwm_msp_init()	39
4.5.3.10 bsp_led_rgb_blink_disable()	39
4.5.3.11 bsp_led_rgb_blink_enable()	39
4.5.3.12 bsp_led_rgb_calibrate()	40
4.5.3.13 bsp_led_rgb_deinit()	40
4.5.3.14 bsp_led_rgb_init()	41
4.5.3.15 bsp_led_rgb_set()	41
4.5.3.16 TIM2_IRQHandler()	41
4.6 BSP RCC	41
4.6.1 Detailed Description	42
4.6.2 Macro Definition Documentation	42
4.6.2.1 TIM_APB_NUM_CLOCK_GET	42
4.6.3 Function Documentation	42
4.6.3.1 bsp_rcc_apb_timer_freq_get()	42
4.6.3.2 bsp_rcc_main_config_init()	43
4.7 BSP	43
4.7.1 Detailed Description	43
4.8 BSP UART	44
4.8.1 Detailed Description	45
4.8.2 Macro Definition Documentation	45
4.8.2.1 HAL_UART_PARITY_TO	45
4.8.2.2 HAL_UART_STOPBITS_TO	46
4.8.2.3 HAL_UART_WORDLEN_TO	46
4.8.2.4 UART_PARITY_VALID	47
4.8.2.5 UART_STOPBITS_VALID	47

4.8.2.6 UART_TYPE_VALID	47
4.8.2.7 UART_WORDLEN_VALID	48
4.8.3 Enumeration Type Documentation	48
4.8.3.1 uart_parity	48
4.8.3.2 uart_stopbits	49
4.8.3.3 uart_type	49
4.8.3.4 uart_wordlen	49
4.8.4 Function Documentation	49
4.8.4.1 __uart_dma_deinit()	50
4.8.4.2 __uart_dma_init()	50
4.8.4.3 __uart_error_callback()	50
4.8.4.4 __uart_irq_handler()	51
4.8.4.5 __uart_msp_deinit()	51
4.8.4.6 __uart_msp_init()	51
4.8.4.7 __uart_rx_callback()	52
4.8.4.8 __uart_type_get()	52
4.8.4.9 bsp_uart_deinit()	53
4.8.4.10 bsp_uart_init()	53
4.8.4.11 bsp_uart_is_started()	53
4.8.4.12 bsp_uart_read()	54
4.8.4.13 bsp_uart_rx_buffer_is_empty()	54
4.8.4.14 bsp_uart_start()	55
4.8.4.15 bsp_uart_stop()	55
4.8.4.16 bsp_uart_write()	55
4.8.4.17 DMA1_Stream1_IRQHandler()	56
4.8.4.18 DMA1_Stream2_IRQHandler()	56
4.8.4.19 DMA1_Stream4_IRQHandler()	56
4.8.4.20 DMA1_Stream5_IRQHandler()	57
4.8.4.21 UART4_IRQHandler()	57
4.8.4.22 USART2_IRQHandler()	57
4.8.4.23 USART3_IRQHandler()	57
4.8.5 Variable Documentation	57
4.8.5.1	57
4.9 Common	58
4.9.1 Detailed Description	58
4.9.2 Macro Definition Documentation	58
4.9.2.1 ARRAY_SIZE	58
4.9.2.2 INSTR_DELAY_US	59
4.9.2.3 IS_PRINTABLE	59
4.9.2.4 MAX	60
4.9.2.5 MIN	60
4.10 Application layer of RGB LED	60

4.10.1 Detailed Description	61
4.10.2 Macro Definition Documentation	61
4.10.2.1 LED_EVENT_IS_VALID	61
4.10.3 Enumeration Type Documentation	62
4.10.3.1 led_event	62
4.10.4 Function Documentation	62
4.10.4.1 app_led_deinit()	62
4.10.4.2 app_led_init()	63
4.10.4.3 app_led_set()	63
4.11 Basic interrupts	63
4.11.1 Detailed Description	63
4.11.2 Function Documentation	64
4.11.2.1 BusFault_Handler()	64
4.11.2.2 DebugMon_Handler()	64
4.11.2.3 HardFault_Handler()	64
4.11.2.4 MemManage_Handler()	64
4.11.2.5 NMI_Handler()	64
4.11.2.6 PendSV_Handler()	65
4.11.2.7 SVC_Handler()	65
4.11.2.8 SysTick_Handler()	65
4.11.2.9 UsageFault_Handler()	65
4.12 CLI	65
4.12.1 Detailed Description	67
4.12.2 Function Documentation	67
4.12.2.1 __cli_menu_cfg_set()	67
4.12.2.2 __cli_menu_cfg_values_set()	68
4.12.2.3 __cli_menu_entry()	68
4.12.2.4 __cli_menu_exit()	68
4.12.2.5 __cli_menu_read_cb()	69
4.12.2.6 __cli_menu_set_defaults()	69
4.12.2.7 __cli_menu_write_cb()	70
4.12.2.8 __cli_prompt_generator()	70
4.12.2.9 __cli_uart_error_cb()	70
4.12.2.10 __cli_uart_overflow_cb()	71
4.12.2.11 cli_init()	71
4.12.2.12 cli_menu_exit()	72
4.12.2.13 cli_menu_is_started()	72
4.12.2.14 cli_menu_start()	72
4.12.2.15 cli_rs232_trace()	72
4.12.2.16 cli_terminal_reset()	73
4.12.2.17 cli_trace()	73
4.12.2.18 cli_welcome()	74

4.12.3 Variable Documentation	74
4.12.3.1 color_config_choose	74
4.12.3.2	75
4.12.3.3 rs232_channel_type_str	75
4.12.3.4 rs232_interspace_type_str	75
4.12.3.5 rs232_trace_type_str	76
4.12.3.6 uart_parity_str	76
4.13 Configuration	76
4.13.1 Detailed Description	77
4.13.2 Macro Definition Documentation	77
4.13.2.1 FLASH_CONFIG_DEFAULT	77
4.13.2.2 RS232_INTERSPACE_TYPE_VALID	77
4.13.2.3 RS232_TRACE_TYPE_VALID	78
4.13.2.4 UART_PRESETTINGS_DEFAULT	78
4.13.3 Enumeration Type Documentation	78
4.13.3.1 rs232_interspace_type	79
4.13.3.2 rs232_trace_type	80
4.13.4 Function Documentation	80
4.13.4.1 config_read()	80
4.13.4.2 config_save()	81
4.14 Application	81
4.14.1 Detailed Description	81
4.15 Main	81
4.15.1 Detailed Description	82
4.15.2 Macro Definition Documentation	82
4.15.2.1 IS_UART_ERROR	82
4.15.3 Function Documentation	83
4.15.3.1 button_cb()	83
4.15.3.2 button_wait_event()	83
4.15.3.3 internal_error()	84
4.15.3.4 main()	84
4.15.3.5 uart_error_cb()	84
4.15.3.6 uart_lin_break_cb()	85
4.15.3.7 uart_overflow_cb()	85
4.16 Menu library	85
4.16.1 Detailed Description	87
4.16.2 Macro Definition Documentation	87
4.16.2.1 MENU_COLOR_CONFIG_DEFAULT	88
4.16.2.2 MENU_NUM_TYPE_IS_VALID	88
4.16.2.3 MENU_PASS_TYPE_IS_VALID	88
4.16.3 Enumeration Type Documentation	89
4.16.3.1 menu_color_type	89

4.16.3.2 menu_num_type	89
4.16.3.3 menu_pass_type	90
4.16.4 Function Documentation	90
4.16.4.1 __menu_enumerator_inc()	90
4.16.4.2 __menu_get_last_item()	90
4.16.4.3 __menu_item_is_in_menu()	91
4.16.4.4 __menu_redraw()	91
4.16.4.5 __menu_strlen()	92
4.16.4.6 menu_all_destroy()	92
4.16.4.7 menu_by_label_get()	92
4.16.4.8 menu_create()	92
4.16.4.9 menu_current_item_get()	93
4.16.4.10 menu_entry()	93
4.16.4.11 menu_exit()	94
4.16.4.12 menu_is_started()	94
4.16.4.13 menu_item_add()	94
4.16.4.14 menu_item_by_label_get()	95
4.16.4.15 menu_item_by_label_only_get()	95
4.16.4.16 menu_item_label_get()	96
4.16.4.17 menu_item_value_set()	96
4.16.4.18 menu_start()	96
4.17 Algorithm of Sniffer RS-232	97
4.17.1 Detailed Description	98
4.17.2 Macro Definition Documentation	99
4.17.2.1 LIN_BREAK_MIN_LEN	99
4.17.2.2 RS232_CHANNEL_TYPE_VALID	99
4.17.2.3 SNIFFER_RS232_CFG_PARAM_IS_VALID	99
4.17.2.4 SNIFFER_RS232_CFG_PARAM_MAX	100
4.17.2.5 SNIFFER_RS232_CFG_PARAM_MIN	100
4.17.2.6 SNIFFER_RS232_CONFIG_DEFAULT	100
4.17.3 Enumeration Type Documentation	101
4.17.3.1 rs232_channel_type	101
4.17.4 Function Documentation	101
4.17.4.1 __sniffer_rs232_baudrate_calc()	101
4.17.4.2 __sniffer_rs232_baudrate_get()	102
4.17.4.3 __sniffer_rs232_line_baudrate_calc()	102
4.17.4.4 __sniffer_rs232_line_baudrate_calc_init()	103
4.17.4.5 __sniffer_rs232_params_calc()	103
4.17.4.6 __sniffer_rs232_tim_msp_deinit()	103
4.17.4.7 __sniffer_rs232_tim_msp_init()	105
4.17.4.8 __sniffer_rs232_uart_error_cb()	105
4.17.4.9 __sniffer_rs232_uart_overflow_cb()	105

4.17.4.10	EXTI3_IRQHandler()	106
4.17.4.11	EXTI9_5_IRQHandler()	106
4.17.4.12	sniffer_rs232_calc()	106
4.17.4.13	sniffer_rs232_config_check()	107
4.17.4.14	sniffer_rs232_config_item_range()	107
4.17.4.15	sniffer_rs232_deinit()	107
4.17.4.16	sniffer_rs232_init()	108
4.17.5	Variable Documentation	108
4.17.5.1	alg_tim	108
4.17.5.2	baudrates_list	108
4.17.5.3	hexti1	108
4.17.5.4	hexti2	108
4.17.5.5	hyp_seq	109
4.17.5.6	rx_buffer	109
4.17.5.7	tx_buffer	109
5	Data Structure Documentation	111
5.1	baud_calc_ctx Struct Reference	111
5.1.1	Detailed Description	111
5.2	bsp_led_pwm Struct Reference	112
5.2.1	Detailed Description	112
5.3	bsp_led_rgb Struct Reference	112
5.3.1	Detailed Description	112
5.4	button_init_ctx Struct Reference	113
5.4.1	Detailed Description	113
5.4.2	Field Documentation	113
5.4.2.1	button_isr_cb	113
5.4.2.2	long_press_dur_ms	113
5.4.2.3	press_delay_ms	113
5.4.2.4	press_min_dur_ms	114
5.5	flash_config Struct Reference	114
5.5.1	Detailed Description	114
5.5.2	Field Documentation	114
5.5.2.1	alg_config	114
5.5.2.2	crc	115
5.5.2.3	idle_presence	115
5.5.2.4	presettings	115
5.5.2.5	save_to_presettings	115
5.5.2.6	trace_type	115
5.5.2.7	txrx_delimiter	115
5.6	hyp_check_ctx Struct Reference	116
5.6.1	Detailed Description	116

5.6.2 Field Documentation	116
5.6.2.1 error_frame_cnt	116
5.6.2.2 error_parity_cnt	116
5.7 hyp_ctx Struct Reference	117
5.7.1 Detailed Description	117
5.7.2 Field Documentation	117
5.7.2.1 jump	117
5.7.2.2 parity	117
5.7.2.3 wordlen	117
5.8 lcd1602_settings Struct Reference	117
5.8.1 Detailed Description	118
5.9 menu_item::menu Struct Reference	118
5.9.1 Detailed Description	119
5.10 menu_color Struct Reference	119
5.10.1 Detailed Description	119
5.11 menu_color_config Struct Reference	119
5.11.1 Detailed Description	120
5.12 menu_config Struct Reference	120
5.12.1 Detailed Description	120
5.12.2 Field Documentation	120
5.12.2.1 indent	120
5.12.2.2 is_looped	120
5.12.2.3 num_delim	121
5.12.2.4 num_type	121
5.12.2.5 pass_type	121
5.12.2.6 read_callback	121
5.12.2.7 width	121
5.12.2.8 write_callback	121
5.13 menu_item Struct Reference	121
5.13.1 Detailed Description	122
5.14 sniffer_rs232_config Struct Reference	122
5.14.1 Detailed Description	123
5.15 uart_ctx Struct Reference	123
5.15.1 Detailed Description	124
5.16 uart_init_ctx Struct Reference	124
5.16.1 Detailed Description	124
5.17 uart_presettings Struct Reference	125
5.17.1 Detailed Description	125
6 File Documentation	127
6.1 app_led.h File Reference	127
6.1.1 Detailed Description	128

6.2 app_led.h	128
6.3 cli.h File Reference	128
6.3.1 Detailed Description	129
6.4 cli.h	129
6.5 config.h File Reference	130
6.5.1 Detailed Description	130
6.6 config.h	131
6.7 menu.h File Reference	131
6.7.1 Detailed Description	133
6.8 menu.h	133
6.9 sniffer_rs232.h File Reference	135
6.9.1 Detailed Description	136
6.10 sniffer_rs232.h	136
6.11 app_led.c File Reference	137
6.11.1 Detailed Description	137
6.12 basic_interrupts.c File Reference	138
6.12.1 Detailed Description	138
6.13 cli.c File Reference	138
6.13.1 Detailed Description	140
6.14 config.c File Reference	141
6.14.1 Detailed Description	141
6.15 main.c File Reference	141
6.15.1 Detailed Description	142
6.16 menu.c File Reference	143
6.16.1 Detailed Description	144
6.17 sniffer_rs232.c File Reference	144
6.17.1 Detailed Description	145
6.18 bsp_button.h File Reference	146
6.18.1 Detailed Description	146
6.19 bsp_button.h	146
6.20 bsp_crc.h File Reference	147
6.20.1 Detailed Description	147
6.21 bsp_crc.h	147
6.22 bsp_gpio.h File Reference	147
6.22.1 Detailed Description	148
6.23 bsp_gpio.h	148
6.24 bsp_lcd1602.h File Reference	148
6.24.1 Detailed Description	150
6.25 bsp_lcd1602.h	150
6.26 bsp_led_rgb.h File Reference	151
6.26.1 Detailed Description	152
6.27 bsp_led_rgb.h	152

6.28 bsp_rcc.h File Reference	153
6.28.1 Detailed Description	153
6.29 bsp_rcc.h	154
6.30 bsp_uart.h File Reference	154
6.30.1 Detailed Description	155
6.31 bsp_uart.h	155
6.32 bsp_button.c File Reference	156
6.32.1 Detailed Description	157
6.33 bsp_crc.c File Reference	158
6.33.1 Detailed Description	158
6.34 bsp_gpio.c File Reference	158
6.34.1 Detailed Description	159
6.35 bsp_lcd1602.c File Reference	159
6.35.1 Detailed Description	161
6.36 bsp_led_rgb.c File Reference	161
6.36.1 Detailed Description	162
6.37 bsp_rcc.c File Reference	162
6.37.1 Detailed Description	163
6.38 bsp_uart.c File Reference	163
6.38.1 Detailed Description	164
6.39 common.h File Reference	165
6.39.1 Detailed Description	165
6.40 common.h	166
Index	167

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

BSP	43
BSP button	7
BSP CRC	13
BSP GPIO	15
BSP LCD1602	17
BSP LED RGB	35
BSP RCC	41
BSP UART	44
Common	58
Application	81
Application layer of RGB LED	60
Basic interrupts	63
CLI	65
Configuration	76
Main	81
Menu library	85
Algorithm of Sniffer RS-232	97

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

baud_calc_ctx	111
bsp_led_pwm	
Parameters of RGB LED blinking	112
bsp_led_rgb	
RGB LED structure	112
button_init_ctx	
Initializing context of BSP button	113
flash_config	
Firmware configuration	114
hyp_check_ctx	116
hyp_ctx	117
lcd1602_settings	
Settings of BSP LCD1602	117
menu_item::menu	
Menu context	118
menu_color	
Menu color data	119
menu_color_config	
Menu color settings	119
menu_config	
Menu library settings	120
menu_item	
Menu item context	121
sniffer_rs232_config	
Algorithm settings	122
uart_ctx	
Context of the BSP UART instance	123
uart_init_ctx	
BSP UART initializing context	124
uart_presettings	
UART presettings	125

Chapter 3

File Index

3.1 File List

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

app_led.h	Header of application layer of RGB LED	127
cli.h	Header of command line interface	128
config.h	Header of flash configuration	130
menu.h	Header of menu library	131
sniffer_rs232.h	Header of algorithm of Sniffer RS-232	135
app_led.c	Application layer of RGB LED	137
basic_interrupts.c	File of handlers for basic interrupts	138
cli.c	Command line interface	138
config.c	Flash configuration	141
main.c	Main project file	141
menu.c	Menu library	143
sniffer_rs232.c	Algorithm of Sniffer RS-232	144
bsp_button.h	Header of BSP button module	146
bsp_crc.h	Header of BSP CRC module	147
bsp_gpio.h	Header of BSP GPIO module	147
bsp_lcd1602.h	Header of BSP LCD1602 module	148
bsp_led_rgb.h	Header of BSP LED RGB module	151
bsp_rcc.h	Header of BSP RCC module	153

bsp_uart.h	
Header of BSP UART module	154
bsp_button.c	
BSP button module	156
bsp_crc.c	
BSP CRC module	158
bsp_gpio.c	
BSP GPIO module	158
bsp_lcd1602.c	
BSP LCD1602 module	159
bsp_led_rgb.c	
BSP LED RGB module	161
bsp_rcc.c	
BSP RCC module	162
bsp_uart.c	
BSP UART module	163
common.h	
Common utils	165

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	<i>htim</i>	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	<i>htim</i>	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	<i>htim</i>	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	<i>period_ms</i>	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	<i>init_ctx</i>	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	<i>data</i>	data over which CRC is calculated
in	<i>len</i>	size of data within which CRC is calculated
out	<i>result</i>	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 * hcr )
```

STM32 HAL MSP CRC deinitialization

Parameters

in	<i>hcr</i>	STM32 HAL CRC instance, should equal to crc_module
----	------------	--

4.2.2.5 HAL_CRC_MspInit()

```
void HAL_CRC_MspInit (
    CRC_HandleTypeDef * hcrcc )
```

STM32 HAL MSP CRC initialization

Parameters

in	hcrcc	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	<i>GPIOX</i>	port of a pin
in	<i>GPIO_NUM</i>	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	<i>GPIOX</i>	port of read pin
in	<i>GPIO_PIN</i>	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	<i>GPIOX</i>	port of set pin
in	<i>GPIO_PIN</i>	set pin
in	<i>LEVEL</i>	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	<i>gpiox</i>	port of read pins
in	<i>gpio_pins</i>	GPIO pins levels on which should be read
out	<i>gpio_states</i>	array of levels read on <i>gpio_pins</i>

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	<i>gpiox</i>	port of written pins
in	<i>gpio_pins</i>	GPIO pins levels on which should be set
in	<i>gpio_states</i>	array of levels set on <i>gpio_pins</i>

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-)initializing 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	<code>X</code>	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	<code>X</code>	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	<i>X</i>	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	<i>X</i>	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[7])  
lcd1602_data_pins[4] | lcd1602_data_pins[5] | lcd1602_data_pins[6] |
```

All mixed GPIO pins from [lcd1602_data_pins](#), used for (de-)initializing 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	<i>X</i>	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	<i>X</i>	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	<i>X</i>	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	<code>X</code>	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	<code>X</code>	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`

Number 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	<i>data</i>	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	<i>instruction</i>	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	<i>line1</i>	formatted first line, if NULL - previous content remains
in	<i>line2</i>	formatted second line, if NULL - previous content remains
in	<i>is_centered</i>	flag whether content within each line should be centered
in	<i>argp</i>	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	<i>busy_flag</i>	read BUSY flag, if NULL - not returned
out	<i>address_counter</i>	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	<i>data</i>	read/written value from/to instruction/data register
in	<i>type_reg</i>	type of register, can be LCD1602_INSTR_REG or LCD1602_DATA_REG
in	<i>type_mode</i>	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	<i>timeout</i>	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	<i>address</i>	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	<i>line1</i>	formatted first line, if NULL - previous content remains
in	<i>line2</i>	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	<i>shift</i>	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	<i>address</i>	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	<i>disp_state</i>	display state
in	<i>cursor_state</i>	cursor state
in	<i>cursor_blink_state</i>	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	<i>cursor</i>	move type of cursor
in	<i>shift_entire</i>	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	<i>interface</i>	type of interface
in	<i>num_line</i>	1-line or 2-line mode of display
in	<i>font_size</i>	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	<i>init_settings</i>	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	<i>line1</i>	formatted first line, if NULL - previous content remains
in	<i>line2</i>	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 behaviour

The macro is used to activate specific RGB LED behaviour 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	<i>htim</i>	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	<i>htim</i>	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	<i>htim</i>	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	<i>htim</i>	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	<i>pwm</i>	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 coefficient is:
255 - appropriate color channel is with no corrections,
0 - color channel is not used in LED light

Parameters

in	<i>coef_rgb</i>	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	<i>rgb</i>	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	<i>INSTANCE</i>	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	<i>instance</i>	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 support 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 support 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	<i>type</i>	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	<i>type</i>	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 occurred

The function is the wrapper over user callback [uart_init_ctx::error_isr_cb](#)

Parameters

in	<i>type</i>	BSP UART type
in	<i>error</i>	mask of occurred 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 reception, errors and LIN break detection

Parameters

in	<i>type</i>	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	<i>type</i>	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	<i>type</i>	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 occurred

Parameters

in	<i>huart</i>	STM32 HAL UART instance
in	<i>pos</i>	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	<i>instance</i>	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 deinitialization of BSP UART instance

Parameters

in	<i>type</i>	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 initialization of BSP UART instance according to settings stored in *init*
if appropriate BSP UART instance is initialized it will be reinitialized

Parameters

in	<i>type</i>	BSP UART type
in	<i>init</i>	initializing 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	<i>type</i>	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	<i>type</i>	BSP UART type
out	<i>data</i>	received data
out	<i>len</i>	size of received data
in	<i>tmt_ms</i>	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	<i>type</i>	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	<i>type</i>	BSP UART type
in	<i>data</i>	sent data
in	<i>len</i>	size of sent data
in	<i>tmt_ms</i>	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 occurred.
- `#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	<i>X</i>	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 interrupts routine

Parameters

in	<i>DELAY</i>	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	<i>X</i>	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 behaviour)

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 behaviour)

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 behaviours.

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 behaviour

Parameters

in	<i>led_event</i>	type of LED behaviour
----	------------------	-----------------------

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 occurred.
 bool uart_overflow
 Flag whether UART receive buffer is overflowed.
} 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 without 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 | <i>input</i> | NOT used |
| in | <i>param</i> | 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 | <i>config</i> | 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 | <i>input</i> | NOT used |
| in | <i>param</i> | 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 | <i>input</i> | NOT used |
| in | <i>param</i> | 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 | <i>data</i> | 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 | <i>input</i> | NOT used |
| in | <i>param</i> | 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 | <i>data</i> | 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 | <i>menu_item_label</i> | 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 occurred on CLI  
BSP UART of CLI will be restarted

#### Parameters

|    |               |                                                        |
|----|---------------|--------------------------------------------------------|
| in | <i>type</i>   | UART type, should be <a href="#">BSP_UART_TYPE_CLI</a> |
| in | <i>error</i>  | mask of occurred UART errors                           |
| in | <i>params</i> | 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 occurred

#### Parameters

|    |               |                                                        |
|----|---------------|--------------------------------------------------------|
| in | <i>type</i>   | UART type, should be <a href="#">BSP_UART_TYPE_CLI</a> |
| in | <i>params</i> | 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

|                      |                     |                      |
|----------------------|---------------------|----------------------|
| <code>in, out</code> | <code>config</code> | 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 | <i>uart_type</i>  | channel type of traced data, should be <a href="#">BSP_UART_TYPE_RS232_TX</a> or <a href="#">BSP_UART_TYPE_RS232_RX</a> |
| in | <i>trace_type</i> | trace type                                                                                                              |
| in | <i>data</i>       | traced data                                                                                                             |
| in | <i>len</i>        | length of traced data                                                                                                   |
| in | <i>break_line</i> | 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 | <i>format</i> | formatted string                                    |
| in | ...           | variable argument list for formatting <i>format</i> |

#### 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 occurred the function is terminated with *is\_pressed* = false

##### Parameters

|     |                    |                                    |
|-----|--------------------|------------------------------------|
| in  | <i>welcome</i>     | welcome message                    |
| in  | <i>wait_time_s</i> | timeout in seconds                 |
| in  | <i>forced_exit</i> | flag of occurred external action   |
| out | <i>is_pressed</i>  | flag whether key pressing occurred |

##### 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\\_INTERSPACE\\_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_INTERSPACE_NONE` = 0 , `RS232_INTERSPACE_SPACE` , `RS232_INTERSPACE_NEW_LINE` , `RS232_INTERSPACE_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_INTERSPACE_NONE,\
 .txrx_delimiter = RS232_INTERSPACE_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_INTERSPACE_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 presetsings

**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_INTERSPACE_NONE     | No interspaces.                      |
| RS232_INTERSPACE_SPACE    | Space between RS-232 data.           |
| RS232_INTERSPACE_NEW_LINE | New line symbol between RS-232 data. |
| RS232_INTERSPACE_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 | <i>config</i> | 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 occurred.*
- ```

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 occurred.
} 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 occurred

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 occurred. Algorithm of the callback is the following:

1. Action [BUTTON_PRESSED](#) occurred:
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	<i>action</i>	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](#) occurred, using [press_event](#)
[press_event](#) is cleared if was set

Parameters

in	<i>tmt</i>	timeout in ms for event waiting
----	------------	---------------------------------

Returns

true if event occurred, 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 occurred 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	<i>led_event</i>	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 occurred

Parameters

in	<i>type</i>	UART type
in	<i>error</i>	mask of occurred UART errors
in	<i>params</i>	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	<i>type</i>	UART type
in	<i>params</i>	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 occurred

Parameters

in	<i>type</i>	UART type
in	<i>params</i>	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	<i>X</i>	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	<i>num_type</i>	numbering type
in, out	<i>enumerator</i>	enumerator as string
in	<i>enum_len</i>	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	<i>menu</i>	menu
in	<i>menu_item</i>	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	<i>prev_item_active</i>	previous menu item
in	<i>new_item_active</i>	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 strlen() guaranteeing return of string length less than [MENU_MAX_STR_LEN](#)

Parameters

in	<i>str</i>	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	<i>label</i>	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	<i>label</i>	label of menu
in	<i>filler</i>	filler for label of menu to fill rest part of menu_config::width
in	<i>color_config</i>	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	<i>menu</i>	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	<i>menu</i>	menu into which menu item is being added
in	<i>label</i>	label of menu item
in	<i>prompt</i>	prompt of menu item, NULL if no prompt
in	<i>value_border</i>	string border
in	<i>callback</i>	user callback by actions on menu item
in	<i>param</i>	optional parameters passed to <i>callback</i>
in	<i>menu_entry</i>	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	<i>menu</i>	menu from which menu item is got
in	<i>label</i>	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	<i>label</i>	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	<i>menu_item</i>	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	<i>menu_item</i>	menu item
in	<i>value</i>	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	<i>config</i>	menu library settings
in	<i>menu</i>	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	<i>TYPE</i>	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	<i>X</i>	parameter name
in	<i>V</i>	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 algorithm 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 algorithm 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	<i>channel_type</i>	RS-232 channel detection type
out	<i>baudrate</i>	calculated baudrate
out	<i>lin_detected</i>	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	<i>len_bit</i>	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	<i>ctx</i>	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	<i>gpiox</i>	GPIO port of <i>pin</i> used as EXTI
in	<i>pin</i>	GPIO pin used as EXTI
in	<i>irq_type</i>	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	<i>channel_type</i>	RS-232 channel detection type
in	<i>baudrate</i>	baudrate in bods on RS-232 lines
out	<i>hyp_num</i>	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	<i>htim</i>	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	<i>htim</i>	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 occurred

Callback counts UART errors into check context of the current hypothesis [hyp_check_ctx](#)

Parameters

in	<i>type</i>	UART type
in	<i>error</i>	mask of occurred UART errors
in	<i>params</i>	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 occurred

If call occurred the algorithm is terminated with fail

Parameters

in	<i>type</i>	UART type
in	<i>params</i>	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	<i>uart_params</i>	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	<i>__config</i>	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	<i>shift</i>	memory shift of an item over sniffer_rs232_config
in	<i>is_min</i>	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 occurred.

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 position in [rx_buff](#) used as ring buffer.
- uint16_t **rx_idx_set**
Write position in [rx_buff](#) used as ring buffer.
- bool **frame_error**
Flag whetner UART frame error is ocured, 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 behaviour)

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.](#)

```
1
2
3 8 #ifndef __APP_LED_H
4 9 #define __APP_LED_H
5 10
6 11 #include "common.h"
7 12 #include <stdint.h>
8 13 #include <stddef.h>
9 14
10 21 enum led_event {
11 22     LED_EVENT_NONE = 0,
12 23     LED_EVENT_COMMON_ERROR,
13 24     LED_EVENT_CRC_ERROR,
14 25     LED_EVENT_FLASH_ERROR,
15 26     LED_EVENT_LCD1602_ERROR,
16 27     LED_EVENT_IN_PROCESS,
17 28     LED_EVENT_SUCCESS,
18 29     LED_EVENT_FAILED,
19 30     LED_EVENT_UART_ERROR,
20 31     LED_EVENT_UART_OVERFLOW,
21 32     LED_EVENT_MAX
22 33 };
23 34
24 42 #define LED_EVENT_IS_VALID(X)      (((uint32_t)(X)) < LED_EVENT_MAX)
25 43
26 48 uint8_t app_led_init(void);
27 49
28 54 uint8_t app_led_deinit(void);
29 55
30 61 uint8_t app_led_set(enum led_event led_event);
31 62
32 65 #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.](#)

```
1
2
3 #ifndef __CLI_H__
4 #define __CLI_H__
5
6 #include "common.h"
7 #include <stdint.h>
8 #include <stdbool.h>
9 #include <stddef.h>
10 #include "config.h"
11
12 uint8_t cli_init(void);
13
14 uint8_t cli_menu_start(struct flash_config *config);
15
16 uint8_t cli_menu_exit(void);
17
18 bool cli_menu_is_started(void);
19
20 void cli_trace(const char *format, ...);
21
22 uint8_t cli_rs232_trace(enum uart_type uart_type,
23                        enum rs232_trace_type trace_type,
24                        uint8_t *data,
25                        uint32_t len,
26                        bool break_line);
27
28 uint8_t cli_welcome(const char *welcome, uint8_t wait_time_s, bool *forced_exit, bool *is_pressed);
29
30 void cli_terminal_reset(void);
31
32 #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_INTERSPACE_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_INTERSPACE_NONE](#) = 0 , [RS232_INTERSPACE_SPACE](#) , [RS232_INTERSPACE_NEW_LINE](#) , [RS232_INTERSPACE_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.](#)

```

1
2 #ifndef __CONFIG_H__
3 #define __CONFIG_H__
4
5 #include "common.h"
6 #include "stm32f4xx_hal.h"
7 #include "sniffer_rs232.h"
8
9 #define RS232_TRACE_TYPE_VALID(X) ((X) < RS232_TRACE_MAX)
10
11 #define RS232_INTERSPACE_TYPE_VALID(X) ((X) < RS232_INTERSPACE_MAX)
12
13 enum rs232_trace_type {
14     RS232_TRACE_HEX = 0,
15     RS232_TRACE_HYBRID,
16     RS232_TRACE_MAX
17 };
18
19 enum rs232_interspace_type {
20     RS232_INTERSPACE_NONE = 0,
21     RS232_INTERSPACE_SPACE,
22     RS232_INTERSPACE_NEW_LINE,
23     RS232_INTERSPACE_MAX
24 };
25
26 struct uart_presettings {
27     bool enable;
28     uint32_t baudrate;
29     enum uart_wordlen wordlen;
30     enum uart_parity parity;
31     enum uart_stopbits stopbits;
32     bool lin_enabled;
33 };
34
35 #pragma pack(1)
36 struct flash_config {
37     struct sniffer_rs232_config alg_config;
38     struct uart_presettings presettings;
39     enum rs232_trace_type trace_type;
40     enum rs232_interspace_type idle_presence;
41     enum rs232_interspace_type txrx_delimiter;
42     bool save_to_presettings;
43     uint32_t crc;
44 };
45
46 #pragma pack()
47
48 #define UART_PRESETTINGS_DEFAULT() {\
49     .enable = false,\
50     .parity = BSP_UART_PARITY_NONE,\
51     .baudrate = 0,\
52     .stopbits = BSP_UART_STOPBITS_1,\
53     .wordlen = BSP_UART_WORDLEN_8,\
54     .lin_enabled = false\
55 }
56
57 #define FLASH_CONFIG_DEFAULT() {\
58     .alg_config = SNIFFER_RS232_CONFIG_DEFAULT(),\
59     .presettings = UART_PRESETTINGS_DEFAULT(),\
60     .trace_type = RS232_TRACE_HEX,\
61     .idle_presence = RS232_INTERSPACE_NONE,\
62     .txrx_delimiter = RS232_INTERSPACE_NONE,\
63     .save_to_presettings = true\
64 }
65
66 uint8_t config_save(struct flash_config *config);
67
68 uint8_t config_read(struct flash_config *config);
69
70 #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.](#)

```

1
8 #ifndef __MENU_H__
9 #define __MENU_H__
10
11 #include "common.h"
12 #include <stdint.h>
13 #include <stdbool.h>
14 #include <stddef.h>
15
22 #define MENU_MAX_STR_LEN          256
23
25 #define MENU_COLOR_RESET          "\33[0;37;40m"
26
28 #define MENU_RETURN_HOME          "\33[H"
29
31 #define MENU_LINE_UP              "\33[A"
32
34 #define MENU_LINE_DOWN            "\33[B"
35
37 #define MENU_LINE_ERASE           "\33[2K"
38
40 #define MENU_SCREEN_ERASE         "\33[2J"
41
43 enum menu_color_type {
44     MENU_COLOR_BLACK = 0,
45     MENU_COLOR_RED,
46     MENU_COLOR_GREEN,
47     MENU_COLOR_YELLOW,
48     MENU_COLOR_BLUE,
49     MENU_COLOR_MAGENTA,

```

```

50     MENU_COLOR_CYAN,
51     MENU_COLOR_WHITE,
52     MENU_COLOR_MAX
53 };
54
55 enum menu_pass_type {
56     MENU_PASS_NONE = 0,
57     MENU_PASS_WITH_PROMPT,
58     MENU_PASS_ALWAYS,
59     MENU_PASS_MAX
60 };
61
62
63 enum menu_num_type {
64     MENU_NUM_NONE = 0,
65     MENU_NUM_DIGITAL,
66     MENU_NUM_UPPER_LETTER,
67     MENU_NUM_LOWER_LETTER,
68     MENU_NUM_MAX
69 };
70
71
72 struct menu_color {
73     enum menu_color_type foreground;
74     enum menu_color_type background;
75 };
76
77
78 struct menu_color_config {
79     struct menu_color active;
80     struct menu_color inactive;
81 };
82
83
84 struct menu_item {
85     struct menu_item *next;
86     struct menu_item *prev;
87
88     struct menu {
89         char *label;
90         char filler;
91         struct menu_color_config color_config;
92         struct menu_item *items;
93         struct menu *next;
94     } *menu_entry;
95
96     uint8_t (*callback) (char *input, void *param);
97     void *param;
98     char *prompt;
99     char *label;
100     char *value_left_border;
101     char *value_right_border;
102     char *value;
103     uint32_t value_len;
104 };
105
106
107 struct menu_config {
108     bool is_looped;
109     uint32_t width;
110     uint32_t indent;
111     enum menu_pass_type pass_type;
112     enum menu_num_type num_type;
113     char num_delim;
114     uint8_t (*read_callback) (char **read_str);
115     uint8_t (*write_callback) (char *write_str);
116 };
117
118 #define MENU_COLOR_CONFIG_DEFAULT() {\
119     .active = {.foreground = MENU_COLOR_BLUE, .background = MENU_COLOR_WHITE},\
120     .inactive = {.foreground = MENU_COLOR_WHITE, .background = MENU_COLOR_BLUE}\
121 }
122
123 void menu_all_destroy(void);
124
125 struct menu *menu_create(char *label, char filler, struct menu_color_config *color_config);
126
127 uint8_t menu_entry(struct menu *menu);
128
129 uint8_t menu_item_value_set(struct menu_item *menu_item, const char *value);
130
131 struct menu_item *menu_current_item_get(void);
132
133 char *menu_item_label_get(struct menu_item *menu_item);
134
135 struct menu *menu_by_label_get(const char *label);
136
137 struct menu_item *menu_item_by_label_get(struct menu *menu, const char *label);
138
139 struct menu_item *menu_item_by_label_only_get(const char *label);
140
141 bool menu_is_started(void);

```

```

218
228 uint8_t menu_start(struct menu_config *config, struct menu *menu);
229
236 uint8_t menu_exit(void);
237
257 struct menu_item * menu_item_add(struct menu *menu,
258                                 const char *label,
259                                 const char *prompt,
260                                 const char *value_border,
261                                 uint8_t (*callback) (char *input, void *param),
262                                 void *param,
263                                 struct menu *menu_entry);
264
267 #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.](#)

```

1
2 #ifndef __SNIFFER_RS232_H__
3 #define __SNIFFER_RS232_H__
4
5 #include "common.h"
6 #include "stm32f4xx_hal.h"
7 #include "bsp_uart.h"
8 #include <stdbool.h>
9
10 enum rs232_channel_type {
11     RS232_CHANNEL_TX = 0,
12     RS232_CHANNEL_RX,
13     RS232_CHANNEL_ANY,
14     RS232_CHANNEL_ALL,
15     RS232_CHANNEL_MAX
16 };
17
18 #define RS232_CHANNEL_TYPE_VALID(TYPE) (((uint32_t)(TYPE)) < RS232_CHANNEL_MAX)
19
20 struct sniffer_rs232_config {
21     enum rs232_channel_type channel_type;
22     uint32_t valid_packets_count;
23     uint32_t uart_error_count;
24     uint8_t baudrate_tolerance;
25     uint32_t min_detect_bits;
26     uint32_t exec_timeout;
27     uint32_t calc_attempts;
28     bool lin_detection;
29 };
30
31 #define SNIFFER_RS232_CFG_PARAM_MIN(X) sniffer_rs232_config_item_range((uint32_t)&((struct
32     sniffer_rs232_config*)0)->X, true)
33
34 #define SNIFFER_RS232_CFG_PARAM_MAX(X) sniffer_rs232_config_item_range((uint32_t)&((struct
35     sniffer_rs232_config*)0)->X, false)
36
37 #define SNIFFER_RS232_CFG_PARAM_IS_VALID(X, V) (((V) >= SNIFFER_RS232_CFG_PARAM_MIN(X)) && ((V) <=
38     SNIFFER_RS232_CFG_PARAM_MAX(X)))
39
40 #define SNIFFER_RS232_CONFIG_DEFAULT() {\
41     .channel_type = RS232_CHANNEL_ANY,\
42     .valid_packets_count = 20,\
43     .uart_error_count = 2,\
44     .baudrate_tolerance = 10,\
45     .min_detect_bits = 48,\
46     .exec_timeout = 600,\
47     .calc_attempts = 3,\
48     .lin_detection = false\
49 }
50
51 uint8_t sniffer_rs232_init(struct sniffer_rs232_config *__config);
52
53 uint8_t sniffer_rs232_deinit(void);
54
55 uint8_t sniffer_rs232_calc(struct uart_init_ctx *uart_params);
56
57 uint32_t sniffer_rs232_config_item_range(uint32_t shift, bool is_min);
58
59 bool sniffer_rs232_config_check(struct sniffer_rs232_config *__config);
60
61 #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 interrupts.

```
#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 interrupts.

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 occurred.
 - `bool uart_overflow`
Flag whether UART receive buffer is overflowed.
- `} 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 without 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 occurred.
- struct {
 uint32_t [error](#)
 Mask of UART errors.
 bool [overflow](#)
 Flag whether UART RX buffer is overflowed before call [bsp_uart_read](#).
 bool [lin_break](#)
 Flag whether LIN break detection is occurred.
} [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.](#)

```
1
2
3 8 #ifndef __BSP_BUTTON_H__
4 9 #define __BSP_BUTTON_H__
5 10
6 11 #include <stdint.h>
7 12 #include "stm32f4xx_hal.h"
8 13
9 20 enum button_action {
10 22     BUTTON_NONE = 0,
11 25     BUTTON_PRESSED,
12 27     BUTTON_LONG_PRESSED,
13 29     BUTTON_ACTION_MAX
14 30 };
15 31
16 33 struct button_init_ctx {
17 35     uint32_t press_delay_ms;
18 39     uint32_t press_min_dur_ms;
19 42     uint32_t long_press_dur_ms;
20 44     void (*button_isr_cb) (enum button_action action);
21 45 };
22 46
23 52 uint8_t bsp_button_init (struct button_init_ctx *init_ctx);
24 53
25 58 uint8_t bsp_button_deinit (void);
26 59
27 62 #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.](#)

```
1
8 #ifndef __BSP_CRC_H__
9 #define __BSP_CRC_H__
10
11 #include <stdint.h>
12 #include "stm32f4xx_hal.h"
13
23 uint8_t bsp_crc_init(void);
24
29 uint8_t bsp_crc_deinit(void);
30
38 uint8_t bsp_crc_calc(uint8_t *data, uint32_t len, uint32_t *result);
39
42 #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.](#)

```

1
2
3
4
5
6
7
8 #ifndef __BSP_GPIO_H__
9 #define __BSP_GPIO_H__
10
11 #include <stdint.h>
12 #include "stm32f4xx_hal.h"
13
14
15
16 uint8_t bsp_gpio_bulk_read(GPIO_TypeDef* gpiox, const uint16_t *gpio_pins, uint16_t *gpio_states);
17
18 uint8_t bsp_gpio_bulk_write(GPIO_TypeDef* gpiox, const uint16_t *gpio_pins, const uint16_t gpio_states);
19
20
21
22 #define BSP_GPIO_PORT_READ(GPIOX, GPIO_PIN)          (!(GPIOX->IDR & GPIO_PIN))
23
24 #define BSP_GPIO_PORT_WRITE(GPIOX, GPIO_PIN, LEVEL)   (GPIOX->BSRR = LEVEL ? GPIO_PIN :
25   ((uint32_t)GPIO_PIN << 16U))
26
27
28 #define BSP_GPIO_FORCE_OUTPUT_MODE(GPIOX, GPIO_NUM)  \
29 do {\
30   GPIOX->MODER = (GPIOX->MODER | (1 << (2 * GPIO_NUM))) & ~(1 << (2 * GPIO_NUM + 1));\
31 } while(0)
32
33 #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.](#)

```
1
2
3
4
5
6
7
8 #ifndef __BSP_LCD1602_H__
9 #define __BSP_LCD1602_H__
10
11 #include <stdint.h>
12
13
14
15
16
17
18
19 enum lcd1602_type_shift {
20     LCD1602_SHIFT_CURSOR_UNDEF = -1,
21     LCD1602_SHIFT_CURSOR_LEFT,
22     LCD1602_SHIFT_CURSOR_RIGHT,
23     LCD1602_SHIFT_DISPLAY_LEFT,
24     LCD1602_SHIFT_DISPLAY_RIGHT,
25     LCD1602_SHIFT_MAX
26 };
27
28
29 enum lcd1602_num_line {
30     LCD1602_NUM_LINE_UNDEF = -1,
31     LCD1602_NUM_LINE_1,
32     LCD1602_NUM_LINE_2,
33     LCD1602_NUM_LINE_MAX
34 };
35
36
37 enum lcd1602_font_size {
38     LCD1602_FONT_SIZE_UNDEF = -1,
39     LCD1602_FONT_SIZE_5X8,
40     LCD1602_FONT_SIZE_5X11,
41     LCD1602_FONT_SIZE_MAX
42 };
43
44
45 enum lcd1602_type_move_cursor {
46     LCD1602_CURSOR_MOVE_UNDEF = -1,
47     LCD1602_CURSOR_MOVE_LEFT,
48     LCD1602_CURSOR_MOVE_RIGHT,
49     LCD1602_CURSOR_MOVE_MAX
50 };
51
52
53 enum lcd1602_shift_entire_disp {
54     LCD1602_SHIFT_ENTIRE_UNDEF = -1,
55     LCD1602_SHIFT_ENTIRE_PERFORMED,
56     LCD1602_SHIFT_ENTIRE_NOT_PERFORMED,
57     LCD1602_SHIFT_ENTIRE_MAX
58 };
59
60
61 enum lcd1602_type_interface {
62     LCD1602_INTERFACE_UNDEF = -1,
63     LCD1602_INTERFACE_4BITS,
64     LCD1602_INTERFACE_8BITS,
65     LCD1602_INTERFACE_MAX
66 };
67
68
69
70 enum lcd1602_disp_state {
71     LCD1602_DISPLAY_UNDEF = -1,
72     LCD1602_DISPLAY_OFF,
73     LCD1602_DISPLAY_ON,
74     LCD1602_DISPLAY_MAX
75 };
76
77
78 enum lcd1602_cursor_state {
```

```

79     LCD1602_CURSOR_UNDEF = -1,
80     LCD1602_CURSOR_OFF,
81     LCD1602_CURSOR_ON,
82     LCD1602_CURSOR_MAX
83 };
84
85 enum lcd1602_cursor_blink_state {
86     LCD1602_CURSOR_BLINK_UNDEF = -1,
87     LCD1602_CURSOR_BLINK_OFF,
88     LCD1602_CURSOR_BLINK_ON,
89     LCD1602_CURSOR_BLINK_MAX
90 };
91
92
93 struct lcd1602_settings {
94     enum lcd1602_num_line          num_line;
95     enum lcd1602_font_size         font_size;
96     enum lcd1602_type_move_cursor type_move_cursor;
97     enum lcd1602_shift_entire_disp shift_entire_disp;
98     enum lcd1602_type_interface    type_interface;
99     enum lcd1602_disp_state         disp_state;
100    enum lcd1602_cursor_state        cursor_state;
101    enum lcd1602_cursor_blink_state  cursor_blink_state;
102 };
103
104
105 uint8_t bsp_lcd1602_init(struct lcd1602_settings *init_settings);
106
107 uint8_t bsp_lcd1602_deinit(void);
108
109 uint8_t bsp_lcd1602_printf(const char *line1, const char *line2, ...);
110
111 uint8_t bsp_lcd1602_cprintf(const char *line1, const char *line2, ...);
112
113 uint8_t bsp_lcd1602_ddram_address_set(const uint8_t address);
114
115 uint8_t bsp_lcd1602_cgram_address_set(const uint8_t address);
116
117 uint8_t bsp_lcd1602_function_set(const enum lcd1602_type_interface interface,
118                                  const enum lcd1602_num_line num_line,
119                                  const enum lcd1602_font_size font_size);
120
121 uint8_t bsp_lcd1602_cursor_disp_shift(const enum lcd1602_type_shift shift);
122
123 uint8_t bsp_lcd1602_display_on_off(const enum lcd1602_disp_state disp_state,
124                                    const enum lcd1602_cursor_state cursor_state,
125                                    const enum lcd1602_cursor_blink_state cursor_blink_state);
126
127 uint8_t bsp_lcd1602_entry_mode_set(const enum lcd1602_type_move_cursor cursor,
128                                    const enum lcd1602_shift_entire_disp shift_entire);
129
130 uint8_t bsp_lcd1602_return_home(void);
131
132 uint8_t bsp_lcd1602_display_clear(void);
133
134 #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.](#)

```
1
2
3
4
5
6
7
8 #ifndef __BSP_LED_RGB_H__
9 #define __BSP_LED_RGB_H__
10
11 #include <stdint.h>
12 #include <stdbool.h>
13 #include "stm32f4xx_hal.h"
14 #include "bsp_gpio.h"
15 #include "common.h"
16
17
18
19
20
21
22
23 struct bsp_led_rgb {
24     uint8_t r;
25     uint8_t g;
26     uint8_t b;
27 };
28
29
30 struct bsp_led_pwm {
31     uint32_t width_on_ms;
32     uint32_t width_off_ms;
33 };
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49 uint8_t bsp_led_rgb_calibrate(const struct bsp_led_rgb *coef_rgb);
50
51
52
53
54
55
56 uint8_t bsp_led_rgb_set(const struct bsp_led_rgb *rgb);
57
58
59
60
61
62
63
64
65 uint8_t bsp_led_rgb_init(void);
66
67
68
69
70
71
72
73 uint8_t bsp_led_rgb_deinit(void);
74
75
76
77
78
79
80
81
82
83
84
85
86 uint8_t bsp_led_rgb_blink_enable(const struct bsp_led_pwm *pwm);
87
88
89
90
91
92
93
94
95 #define BSP_LED_RGB_HARDFAULT() \
96 do {\
```

```
97 BSP_GPIO_FORCE_OUTPUT_MODE(GPIOA, 8);\
98 BSP_GPIO_FORCE_OUTPUT_MODE(GPIOA, 9);\
99 BSP_GPIO_FORCE_OUTPUT_MODE(GPIOA, 10);\
100 \
101 BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_8, false);\
102 BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_10, false);\
103 \
104 while (true) {\
105   BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_9, false);\
106   INSTR_DELAY_US(100000);\
107   BSP_GPIO_PORT_WRITE(GPIOA, GPIO_PIN_9, true);\
108   INSTR_DELAY_US(100000);\
109 }\
110 } while (0)
111
114 #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.](#)

```

1
8 #ifndef __BSP_RCC_H__
9 #define __BSP_RCC_H__
10
11 #include <stdint.h>
12 #include "stm32f4xx_hal.h"
13
26 #define TIM_APB_NUM_CLOCK_GET(INSTANCE) \
27 ((IS_TIM_INSTANCE(INSTANCE)) ? (\
28 ((INSTANCE) == TIM1) || \
29 ((INSTANCE) == TIM8) || \
30 ((INSTANCE) == TIM9) || \
31 ((INSTANCE) == TIM10) || \
32 ((INSTANCE) == TIM11)) ? 2 : 1) : 0)
33
40 uint8_t bsp_rcc_main_config_init(void);
41
47 uint32_t bsp_rcc_apb_timer_freq_get(TIM_TypeDef *instance);
48
51 #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.](#)

```

1
2
3
4
5
6
7
8 #ifndef __BSP_UART_H__
9 #define __BSP_UART_H__
10
11 #include <stdint.h>
12 #include <stdbool.h>
13 #include "stm32f4xx_hal.h"
14
15
16
17
18
19
20
21
22
23
24
25
26
27 #define UART_TYPE_VALID(X)      (((uint32_t)(X) < BSP_UART_TYPE_MAX))
28
29
30
31
32
33
34
35
36 #define UART_WORDLEN_VALID(X)  (((X) == BSP_UART_WORDLEN_8) || ((X) == BSP_UART_WORDLEN_9))
37
38
39
40
41
42
43
44
45 #define UART_PARITY_VALID(X)   (((X) == BSP_UART_PARITY_NONE) || ((X) == BSP_UART_PARITY_EVEN) || ((X) == BSP_UART_PARITY_ODD))
46
47
48
49
50
51
52
53
54 #define UART_STOPBITS_VALID(X) (((X) == BSP_UART_STOPBITS_1) || ((X) == BSP_UART_STOPBITS_2))
55

```

```

56 #define BSP_UART_ERROR_PE          HAL_UART_ERROR_PE
57 #define BSP_UART_ERROR_NE          HAL_UART_ERROR_NE
58 #define BSP_UART_ERROR_FE          HAL_UART_ERROR_FE
59 #define BSP_UART_ERROR_ORE          HAL_UART_ERROR_ORE
60 #define BSP_UART_ERROR_DMA          HAL_UART_ERROR_DMA
61
63 #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)
64
66 enum uart_type {
67     BSP_UART_TYPE_CLI = 0,
68     BSP_UART_TYPE_RS232_TX,
69     BSP_UART_TYPE_RS232_RX,
70     BSP_UART_TYPE_MAX
71 };
72
74 enum uart_wordlen {
75     BSP_UART_WORDLEN_8 = 8,
76     BSP_UART_WORDLEN_9 = 9
77 };
78
80 enum uart_parity {
81     BSP_UART_PARITY_NONE = 0,
82     BSP_UART_PARITY_EVEN = 1,
83     BSP_UART_PARITY_ODD = 2
84 };
85
87 enum uart_stopbits {
88     BSP_UART_STOPBITS_1 = 1,
89     BSP_UART_STOPBITS_2 = 2
90 };
91
93 struct uart_init_ctx {
94     uint32_t baudrate;
95     uint32_t tx_size;
96     uint32_t rx_size;
97     bool lin_enabled;
98     enum uart_wordlen wordlen;
99     enum uart_parity parity;
100     enum uart_stopbits stopbits;
101     void (*error_isr_cb)(enum uart_type type, uint32_t error, void *params);
102     void (*overflow_isr_cb)(enum uart_type type, void *params);
103     void (*lin_break_isr_cb)(enum uart_type type, void *params);
104     void *params;
105 };
106
117 uint8_t bsp_uart_init(enum uart_type type, struct uart_init_ctx *init);
118
126 uint8_t bsp_uart_deinit(enum uart_type type);
127
139 uint8_t bsp_uart_read(enum uart_type type, uint8_t *data, uint16_t *len, uint32_t tmt_ms);
140
153 uint8_t bsp_uart_write(enum uart_type type, uint8_t *data, uint16_t len, uint32_t tmt_ms);
154
162 uint8_t bsp_uart_start(enum uart_type type);
163
171 uint8_t bsp_uart_stop(enum uart_type type);
172
178 bool bsp_uart_is_started(enum uart_type type);
179
185 bool bsp_uart_rx_buffer_is_empty(enum uart_type type);
186
189 #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-)initializing 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 occurred.
- `#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.](#)

```
1
16 #ifndef __COMMON_H__
17 #define __COMMON_H__
18
19 #define RES_OK                0
20 #define RES_NOK               1
21 #define RES_INVALID_PAR      2
22 #define RES_MEMORY_ERR       3
23 #define RES_TIMEOUT          4
24 #define RES_NOT_SUPPORTED     5
25 #define RES_OVERFLOW          6
26 #define RES_NOT_INITIALIZED   7
27 #define RES_NOT_ALLOWED      8
28
36 #define ARRAY_SIZE(X)        (sizeof(X) / sizeof(X[0]))
37
44 #define MIN(X, Y)             ((X) < (Y)) ? (X) : (Y)
45
52 #define MAX(X, Y)             ((X) > (Y)) ? (X) : (Y)
53
61 #define IS_PRINTABLE(X)       (X >= ' ' && X <= '~')
62
73 #define INSTR_DELAY_US(Delay) \
74 do {\
75   __IO uint32_t clock_delay = Delay * (HAL_RCC_GetSysClockFreq() / 8 / 1000000);\
76   do {\
77     __NOP();\
78   } while (--clock_delay);\
79 } while (0)
80
83 #endif
```

Index

- `__button_tim_is_started`
 - BSP button, [9](#)
- `__button_tim_msp_deinit`
 - BSP button, [9](#)
- `__button_tim_msp_init`
 - BSP button, [11](#)
- `__button_tim_period_elapsed_callback`
 - BSP button, [11](#)
- `__button_tim_start`
 - BSP button, [11](#)
- `__button_tim_stop`
 - BSP button, [12](#)
- `__cli_menu_cfg_set`
 - CLI, [67](#)
- `__cli_menu_cfg_values_set`
 - CLI, [68](#)
- `__cli_menu_entry`
 - CLI, [68](#)
- `__cli_menu_exit`
 - CLI, [68](#)
- `__cli_menu_read_cb`
 - CLI, [69](#)
- `__cli_menu_set_defaults`
 - CLI, [69](#)
- `__cli_menu_write_cb`
 - CLI, [70](#)
- `__cli_prompt_generator`
 - CLI, [70](#)
- `__cli_uart_error_cb`
 - CLI, [70](#)
- `__cli_uart_overflow_cb`
 - CLI, [71](#)
- `__lcd1602_data_write`
 - BSP LCD1602, [27](#)
- `__lcd1602_instruction_write`
 - BSP LCD1602, [28](#)
- `__lcd1602_printf`
 - BSP LCD1602, [28](#)
- `__lcd1602_read_busy_flag`
 - BSP LCD1602, [29](#)
- `__lcd1602_read_write`
 - BSP LCD1602, [29](#)
- `__lcd1602_wait`
 - BSP LCD1602, [29](#)
- `__led_rgb_blink_is_started`
 - BSP LED RGB, [37](#)
- `__led_rgb_blink_start`
 - BSP LED RGB, [37](#)
- `__led_rgb_blink_stop`
 - BSP LED RGB, [37](#)
- `__led_rgb_blink_tim_period_elapsed_callback`
 - BSP LED RGB, [37](#)
- `__led_rgb_blink_tim_pwm_pulse_finished_callback`
 - BSP LED RGB, [38](#)
- `__led_rgb_tim_msp_post_init`
 - BSP LED RGB, [38](#)
- `__led_rgb_tim_msp_prev_deinit`
 - BSP LED RGB, [38](#)
- `__led_rgb_tim_pwm_msp_deinit`
 - BSP LED RGB, [38](#)
- `__led_rgb_tim_pwm_msp_init`
 - BSP LED RGB, [39](#)
- `__menu_enumerator_inc`
 - Menu library, [90](#)
- `__menu_get_last_item`
 - Menu library, [90](#)
- `__menu_item_is_in_menu`
 - Menu library, [90](#)
- `__menu_redraw`
 - Menu library, [91](#)
- `__menu_strlen`
 - Menu library, [91](#)
- `__sniffer_rs232_baudrate_calc`
 - Algorithm of Sniffer RS-232, [101](#)
- `__sniffer_rs232_baudrate_get`
 - Algorithm of Sniffer RS-232, [102](#)
- `__sniffer_rs232_line_baudrate_calc`
 - Algorithm of Sniffer RS-232, [102](#)
- `__sniffer_rs232_line_baudrate_calc_init`
 - Algorithm of Sniffer RS-232, [102](#)
- `__sniffer_rs232_params_calc`
 - Algorithm of Sniffer RS-232, [103](#)
- `__sniffer_rs232_tim_msp_deinit`
 - Algorithm of Sniffer RS-232, [103](#)
- `__sniffer_rs232_tim_msp_init`
 - Algorithm of Sniffer RS-232, [105](#)
- `__sniffer_rs232_uart_error_cb`
 - Algorithm of Sniffer RS-232, [105](#)
- `__sniffer_rs232_uart_overflow_cb`
 - Algorithm of Sniffer RS-232, [105](#)
- `__uart_dma_deinit`
 - BSP UART, [49](#)
- `__uart_dma_init`
 - BSP UART, [50](#)
- `__uart_error_callback`
 - BSP UART, [50](#)
- `__uart_irq_handler`
 - BSP UART, [51](#)

- __uart_msp_deinit
 - BSP UART, 51
- __uart_msp_init
 - BSP UART, 51
- __uart_rx_callback
 - BSP UART, 52
- __uart_type_get
 - BSP UART, 52
- alg_config
 - flash_config, 114
- alg_tim
 - Algorithm of Sniffer RS-232, 108
- Algorithm of Sniffer RS-232, 97
 - __sniffer_rs232_baudrate_calc, 101
 - __sniffer_rs232_baudrate_get, 102
 - __sniffer_rs232_line_baudrate_calc, 102
 - __sniffer_rs232_line_baudrate_calc_init, 102
 - __sniffer_rs232_params_calc, 103
 - __sniffer_rs232_tim_msp_deinit, 103
 - __sniffer_rs232_tim_msp_init, 105
 - __sniffer_rs232_uart_error_cb, 105
 - __sniffer_rs232_uart_overflow_cb, 105
- alg_tim, 108
- baudrates_list, 108
- EXTI3_IRQHandler, 106
- EXTI9_5_IRQHandler, 106
- hexi1, 108
- hexi2, 108
- hyp_seq, 108
- LIN_BREAK_MIN_LEN, 99
- RS232_CHANNEL_ALL, 101
- RS232_CHANNEL_ANY, 101
- RS232_CHANNEL_MAX, 101
- RS232_CHANNEL_RX, 101
- RS232_CHANNEL_TX, 101
- rs232_channel_type, 101
- RS232_CHANNEL_TYPE_VALID, 99
- rx_buffer, 109
- sniffer_rs232_calc, 106
- SNIFFER_RS232_CFG_PARAM_IS_VALID, 99
- SNIFFER_RS232_CFG_PARAM_MAX, 100
- SNIFFER_RS232_CFG_PARAM_MIN, 100
- sniffer_rs232_config_check, 106
- SNIFFER_RS232_CONFIG_DEFAULT, 100
- sniffer_rs232_config_item_range, 107
- sniffer_rs232_deinit, 107
- sniffer_rs232_init, 107
- tx_buffer, 109
- app_led.c, 137
- app_led.h, 127, 128
- app_led_deinit
 - Application layer of RGB LED, 62
- app_led_init
 - Application layer of RGB LED, 62
- app_led_set
 - Application layer of RGB LED, 63
- Application, 81
- Application layer of RGB LED, 60
 - app_led_deinit, 62
 - app_led_init, 62
 - app_led_set, 63
 - led_event, 62
 - LED_EVENT_COMMON_ERROR, 62
 - LED_EVENT_CRC_ERROR, 62
 - LED_EVENT_FAILED, 62
 - LED_EVENT_FLASH_ERROR, 62
 - LED_EVENT_IN_PROCESS, 62
 - LED_EVENT_IS_VALID, 61
 - LED_EVENT_LCD1602_ERROR, 62
 - LED_EVENT_MAX, 62
 - LED_EVENT_NONE, 62
 - LED_EVENT_SUCCESS, 62
 - LED_EVENT_UART_ERROR, 62
 - LED_EVENT_UART_OVERFLOW, 62
- ARRAY_SIZE
 - Common, 58
- Basic interrupts, 63
 - BusFault_Handler, 64
 - DebugMon_Handler, 64
 - HardFault_Handler, 64
 - MemManage_Handler, 64
 - NMI_Handler, 64
 - PendSV_Handler, 64
 - SVC_Handler, 65
 - SysTick_Handler, 65
 - UsageFault_Handler, 65
- basic_interrupts.c, 138
- baud_calc_ctx, 111
- baudrates_list
 - Algorithm of Sniffer RS-232, 108
- BSP, 43
- BSP button, 7
 - __button_tim_is_started, 9
 - __button_tim_msp_deinit, 9
 - __button_tim_msp_init, 11
 - __button_tim_period_elapsed_callback, 11
 - __button_tim_start, 11
 - __button_tim_stop, 12
 - bsp_button_deinit, 12
 - bsp_button_init, 12
 - button_action, 9
 - BUTTON_ACTION_MAX, 9
 - BUTTON_LONG_PRESSED, 9
 - BUTTON_NONE, 9
 - BUTTON_PRESSED, 9
 - EXTI4_IRQHandler, 13
 - TIM7_IRQHandler, 13
 - TIM_PERIOD_CALC, 8
 - TIM_TICK_TO_MS, 8
- BSP CRC, 13
 - bsp_crc_calc, 13
 - bsp_crc_deinit, 14
 - bsp_crc_init, 14
 - HAL_CRC_MspDeInit, 14
 - HAL_CRC_MspInit, 15
- BSP GPIO, 15

- bsp_gpio_bulk_read, 16
- bsp_gpio_bulk_write, 17
- BSP_GPIO_FORCE_OUTPUT_MODE, 15
- BSP_GPIO_PORT_READ, 16
- BSP_GPIO_PORT_WRITE, 16
- BSP LCD1602, 17
 - __lcd1602_data_write, 27
 - __lcd1602_instruction_write, 28
 - __lcd1602_printf, 28
 - __lcd1602_read_busy_flag, 29
 - __lcd1602_read_write, 29
 - __lcd1602_wait, 29
 - bsp_lcd1602_cgram_address_set, 30
 - bsp_lcd1602_cprintf, 30
 - bsp_lcd1602_cursor_disp_shift, 31
 - bsp_lcd1602_ddram_address_set, 31
 - bsp_lcd1602_deinit, 31
 - bsp_lcd1602_display_clear, 32
 - bsp_lcd1602_display_on_off, 32
 - bsp_lcd1602_entry_mode_set, 32
 - bsp_lcd1602_function_set, 33
 - bsp_lcd1602_init, 33
 - bsp_lcd1602_printf, 34
 - bsp_lcd1602_return_home, 34
 - CURSOR_BLINK_STATE_IS_VALID, 20
 - CURSOR_STATE_IS_VALID, 20
 - DISP_STATE_IS_VALID, 20
 - FONT_SIZE_IS_VALID, 21
 - LCD1602_CURSOR_BLINK_MAX, 25
 - LCD1602_CURSOR_BLINK_OFF, 25
 - LCD1602_CURSOR_BLINK_ON, 25
 - lcd1602_cursor_blink_state, 23
 - LCD1602_CURSOR_BLINK_UNDEF, 25
 - LCD1602_CURSOR_MAX, 25
 - LCD1602_CURSOR_MOVE_LEFT, 27
 - LCD1602_CURSOR_MOVE_MAX, 27
 - LCD1602_CURSOR_MOVE_RIGHT, 27
 - LCD1602_CURSOR_MOVE_UNDEF, 27
 - LCD1602_CURSOR_OFF, 25
 - LCD1602_CURSOR_ON, 25
 - lcd1602_cursor_state, 25
 - LCD1602_CURSOR_UNDEF, 25
 - LCD1602_DATA_PINS, 21
 - lcd1602_data_pins, 34
 - lcd1602_disp_state, 25
 - LCD1602_DISPLAY_MAX, 25
 - LCD1602_DISPLAY_OFF, 25
 - LCD1602_DISPLAY_ON, 25
 - LCD1602_DISPLAY_UNDEF, 25
 - lcd1602_font_size, 25
 - LCD1602_FONT_SIZE_5X11, 26
 - LCD1602_FONT_SIZE_5X8, 26
 - LCD1602_FONT_SIZE_MAX, 26
 - LCD1602_FONT_SIZE_UNDEF, 26
 - LCD1602_INTERFACE_4BITS, 27
 - LCD1602_INTERFACE_8BITS, 27
 - LCD1602_INTERFACE_MAX, 27
 - LCD1602_INTERFACE_UNDEF, 27
 - lcd1602_num_line, 26
 - LCD1602_NUM_LINE_1, 26
 - LCD1602_NUM_LINE_2, 26
 - LCD1602_NUM_LINE_MAX, 26
 - LCD1602_NUM_LINE_UNDEF, 26
 - LCD1602_SHIFT_CURSOR_LEFT, 27
 - LCD1602_SHIFT_CURSOR_RIGHT, 27
 - LCD1602_SHIFT_CURSOR_UNDEF, 27
 - LCD1602_SHIFT_DISPLAY_LEFT, 27
 - LCD1602_SHIFT_DISPLAY_RIGHT, 27
 - lcd1602_shift_entire_disp, 26
 - LCD1602_SHIFT_ENTIRE_MAX, 26
 - LCD1602_SHIFT_ENTIRE_NOT_PERFORMED, 26
 - LCD1602_SHIFT_ENTIRE_PERFORMED, 26
 - LCD1602_SHIFT_ENTIRE_UNDEF, 26
 - LCD1602_SHIFT_MAX, 27
 - lcd1602_type_interface, 26
 - lcd1602_type_move_cursor, 27
 - lcd1602_type_shift, 27
 - NUM_LINE_IS_VALID, 21
 - SHIFT_ENTIRE_IS_VALID, 22
 - TYPE_INTERFACE_IS_VALID, 22
 - TYPE_MOVE_CURSOR_IS_VALID, 23
 - TYPE_SHIFT_IS_VALID, 23
- BSP LED RGB, 35
 - __led_rgb_blink_is_started, 37
 - __led_rgb_blink_start, 37
 - __led_rgb_blink_stop, 37
 - __led_rgb_blink_tim_period_elapsed_callback, 37
 - __led_rgb_blink_tim_pwm_pulse_finished_callback, 38
 - __led_rgb_tim_msp_post_init, 38
 - __led_rgb_tim_msp_prev_deinit, 38
 - __led_rgb_tim_pwm_msp_deinit, 38
 - __led_rgb_tim_pwm_msp_init, 39
 - bsp_led_rgb_blink_disable, 39
 - bsp_led_rgb_blink_enable, 39
 - bsp_led_rgb_calibrate, 40
 - bsp_led_rgb_deinit, 40
 - BSP_LED_RGB_HARDFAULT, 36
 - bsp_led_rgb_init, 40
 - bsp_led_rgb_set, 41
 - TIM2_IRQHandler, 41
- BSP RCC, 41
 - bsp_rcc_apb_timer_freq_get, 42
 - bsp_rcc_main_config_init, 43
 - TIM_APB_NUM_CLOCK_GET, 42
- BSP UART, 44
 - __uart_dma_deinit, 49
 - __uart_dma_init, 50
 - __uart_error_callback, 50
 - __uart_irq_handler, 51
 - __uart_msp_deinit, 51
 - __uart_msp_init, 51
 - __uart_rx_callback, 52
 - __uart_type_get, 52
 - bsp_uart_deinit, 52

- bsp_uart_init, 53
- bsp_uart_is_started, 53
- BSP_UART_PARITY_EVEN, 48
- BSP_UART_PARITY_NONE, 48
- BSP_UART_PARITY_ODD, 48
- bsp_uart_read, 54
- bsp_uart_rx_buffer_is_empty, 54
- bsp_uart_start, 55
- bsp_uart_stop, 55
- BSP_UART_STOPBITS_1, 49
- BSP_UART_STOPBITS_2, 49
- BSP_UART_TYPE_CLI, 49
- BSP_UART_TYPE_MAX, 49
- BSP_UART_TYPE_RS232_RX, 49
- BSP_UART_TYPE_RS232_TX, 49
- BSP_UART_WORDLEN_8, 49
- BSP_UART_WORDLEN_9, 49
- bsp_uart_write, 55
- DMA1_Stream1_IRQHandler, 56
- DMA1_Stream2_IRQHandler, 56
- DMA1_Stream4_IRQHandler, 56
- DMA1_Stream5_IRQHandler, 56
- HAL_UART_PARITY_TO, 45
- HAL_UART_STOPBITS_TO, 46
- HAL_UART_WORDLEN_TO, 46
- UART4_IRQHandler, 57
- uart_obj, 57
- uart_parity, 48
- UART_PARITY_VALID, 46
- uart_stopbits, 48
- UART_STOPBITS_VALID, 47
- uart_type, 49
- UART_TYPE_VALID, 47
- uart_wordlen, 49
- UART_WORDLEN_VALID, 48
- USART2_IRQHandler, 57
- USART3_IRQHandler, 57
- bsp_button.c, 156
- bsp_button.h, 146
- bsp_button_deinit
 - BSP button, 12
- bsp_button_init
 - BSP button, 12
- bsp_crc.c, 158
- bsp_crc.h, 147
- bsp_crc_calc
 - BSP CRC, 13
- bsp_crc_deinit
 - BSP CRC, 14
- bsp_crc_init
 - BSP CRC, 14
- bsp_gpio.c, 158
- bsp_gpio.h, 147, 148
- bsp_gpio_bulk_read
 - BSP GPIO, 16
- bsp_gpio_bulk_write
 - BSP GPIO, 17
- BSP_GPIO_FORCE_OUTPUT_MODE
 - BSP GPIO, 15
 - BSP_GPIO_PORT_READ
 - BSP GPIO, 16
 - BSP_GPIO_PORT_WRITE
 - BSP GPIO, 16
 - bsp_lcd1602.c, 159
 - bsp_lcd1602.h, 148, 150
 - bsp_lcd1602_cgram_address_set
 - BSP LCD1602, 30
 - bsp_lcd1602_cprintf
 - BSP LCD1602, 30
 - bsp_lcd1602_cursor_disp_shift
 - BSP LCD1602, 31
 - bsp_lcd1602_ddram_address_set
 - BSP LCD1602, 31
 - bsp_lcd1602_deinit
 - BSP LCD1602, 31
 - bsp_lcd1602_display_clear
 - BSP LCD1602, 32
 - bsp_lcd1602_display_on_off
 - BSP LCD1602, 32
 - bsp_lcd1602_entry_mode_set
 - BSP LCD1602, 32
 - bsp_lcd1602_function_set
 - BSP LCD1602, 33
 - bsp_lcd1602_init
 - BSP LCD1602, 33
 - bsp_lcd1602_printf
 - BSP LCD1602, 34
 - bsp_lcd1602_return_home
 - BSP LCD1602, 34
 - bsp_led_pwm, 112
 - bsp_led_rgb, 112
 - bsp_led_rgb.c, 161
 - bsp_led_rgb.h, 151, 152
 - bsp_led_rgb_blink_disable
 - BSP LED RGB, 39
 - bsp_led_rgb_blink_enable
 - BSP LED RGB, 39
 - bsp_led_rgb_calibrate
 - BSP LED RGB, 40
 - bsp_led_rgb_deinit
 - BSP LED RGB, 40
 - BSP_LED_RGB_HARDFAULT
 - BSP LED RGB, 36
 - bsp_led_rgb_init
 - BSP LED RGB, 40
 - bsp_led_rgb_set
 - BSP LED RGB, 41
 - bsp_rcc.c, 162
 - bsp_rcc.h, 153, 154
 - bsp_rcc_apb_timer_freq_get
 - BSP RCC, 42
 - bsp_rcc_main_config_init
 - BSP RCC, 43
 - bsp_uart.c, 163
 - bsp_uart.h, 154, 155
 - bsp_uart_deinit

- BSP_UART, 52
- bsp_uart_init
 - BSP_UART, 53
- bsp_uart_is_started
 - BSP_UART, 53
- BSP_UART_PARITY_EVEN
 - BSP_UART, 48
- BSP_UART_PARITY_NONE
 - BSP_UART, 48
- BSP_UART_PARITY_ODD
 - BSP_UART, 48
- bsp_uart_read
 - BSP_UART, 54
- bsp_uart_rx_buffer_is_empty
 - BSP_UART, 54
- bsp_uart_start
 - BSP_UART, 55
- bsp_uart_stop
 - BSP_UART, 55
- BSP_UART_STOPBITS_1
 - BSP_UART, 49
- BSP_UART_STOPBITS_2
 - BSP_UART, 49
- BSP_UART_TYPE_CLI
 - BSP_UART, 49
- BSP_UART_TYPE_MAX
 - BSP_UART, 49
- BSP_UART_TYPE_RS232_RX
 - BSP_UART, 49
- BSP_UART_TYPE_RS232_TX
 - BSP_UART, 49
- BSP_UART_WORDLEN_8
 - BSP_UART, 49
- BSP_UART_WORDLEN_9
 - BSP_UART, 49
- bsp_uart_write
 - BSP_UART, 55
- BusFault_Handler
 - Basic interrupts, 64
- button_action
 - BSP button, 9
- BUTTON_ACTION_MAX
 - BSP button, 9
- button_cb
 - Main, 83
- button_init_ctx, 113
 - button_isr_cb, 113
 - long_press_dur_ms, 113
 - press_delay_ms, 113
 - press_min_dur_ms, 113
- button_isr_cb
 - button_init_ctx, 113
- BUTTON_LONG_PRESSED
 - BSP button, 9
- BUTTON_NONE
 - BSP button, 9
- BUTTON_PRESSED
 - BSP button, 9
- button_wait_event
 - Main, 83
- CLI, 65
 - __cli_menu_cfg_set, 67
 - __cli_menu_cfg_values_set, 68
 - __cli_menu_entry, 68
 - __cli_menu_exit, 68
 - __cli_menu_read_cb, 69
 - __cli_menu_set_defaults, 69
 - __cli_menu_write_cb, 70
 - __cli_prompt_generator, 70
 - __cli_uart_error_cb, 70
 - __cli_uart_overflow_cb, 71
 - cli_init, 71
 - cli_menu_exit, 71
 - cli_menu_is_started, 72
 - cli_menu_start, 72
 - cli_rs232_trace, 72
 - cli_terminal_reset, 73
 - cli_trace, 73
 - cli_welcome, 73
 - color_config_choose, 74
 - init_menus, 74
 - rs232_channel_type_str, 75
 - rs232_interspace_type_str, 75
 - rs232_trace_type_str, 75
 - uart_parity_str, 76
- cli.c, 138
- cli.h, 128, 129
- cli_init
 - CLI, 71
- cli_menu_exit
 - CLI, 71
- cli_menu_is_started
 - CLI, 72
- cli_menu_start
 - CLI, 72
- cli_rs232_trace
 - CLI, 72
- cli_terminal_reset
 - CLI, 73
- cli_trace
 - CLI, 73
- cli_welcome
 - CLI, 73
- color_config_choose
 - CLI, 74
- Common, 58
 - ARRAY_SIZE, 58
 - INSTR_DELAY_US, 59
 - IS_PRINTABLE, 59
 - MAX, 60
 - MIN, 60
- common.h, 165, 166
- config.c, 141
- config.h, 130, 131
- config_read
 - Configuration, 80

- config_save
 - Configuration, 80
- Configuration, 76
 - config_read, 80
 - config_save, 80
 - FLASH_CONFIG_DEFAULT, 77
 - rs232_interspace_type, 78
 - RS232_INTERSPACE_TYPE_VALID, 77
 - RS232_INTERSPACE_MAX, 80
 - RS232_INTERSPACE_NEW_LINE, 80
 - RS232_INTERSPACE_NONE, 80
 - RS232_INTERSPACE_SPACE, 80
 - RS232_TRACE_HEX, 80
 - RS232_TRACE_HYBRID, 80
 - RS232_TRACE_MAX, 80
 - rs232_trace_type, 80
 - RS232_TRACE_TYPE_VALID, 78
 - UART_PRESETTINGS_DEFAULT, 78
- crc
 - flash_config, 114
- CURSOR_BLINK_STATE_IS_VALID
 - BSP LCD1602, 20
- CURSOR_STATE_IS_VALID
 - BSP LCD1602, 20
- DebugMon_Handler
 - Basic interrupts, 64
- DISP_STATE_IS_VALID
 - BSP LCD1602, 20
- DMA1_Stream1_IRQHandler
 - BSP UART, 56
- DMA1_Stream2_IRQHandler
 - BSP UART, 56
- DMA1_Stream4_IRQHandler
 - BSP UART, 56
- DMA1_Stream5_IRQHandler
 - BSP UART, 56
- error_frame_cnt
 - hyp_check_ctx, 116
- error_parity_cnt
 - hyp_check_ctx, 116
- EXTI3_IRQHandler
 - Algorithm of Sniffer RS-232, 106
- EXTI4_IRQHandler
 - BSP button, 13
- EXTI9_5_IRQHandler
 - Algorithm of Sniffer RS-232, 106
- flash_config, 114
 - alg_config, 114
 - crc, 114
 - idle_presence, 115
 - presettings, 115
 - save_to_presettings, 115
 - trace_type, 115
 - txrx_delimiter, 115
- FLASH_CONFIG_DEFAULT
 - Configuration, 77
- FONT_SIZE_IS_VALID
 - BSP LCD1602, 21
- HAL_CRC_MspDeInit
 - BSP CRC, 14
- HAL_CRC_MspInit
 - BSP CRC, 15
- HAL_UART_PARITY_TO
 - BSP UART, 45
- HAL_UART_STOPBITS_TO
 - BSP UART, 46
- HAL_UART_WORDLEN_TO
 - BSP UART, 46
- HardFault_Handler
 - Basic interrupts, 64
- hexti1
 - Algorithm of Sniffer RS-232, 108
- hexti2
 - Algorithm of Sniffer RS-232, 108
- hyp_check_ctx, 116
 - error_frame_cnt, 116
 - error_parity_cnt, 116
- hyp_ctx, 117
 - jump, 117
 - parity, 117
 - wordlen, 117
- hyp_seq
 - Algorithm of Sniffer RS-232, 108
- idle_presence
 - flash_config, 115
- indent
 - menu_config, 120
- init_menus
 - CLI, 74
- INSTR_DELAY_US
 - Common, 59
- internal_error
 - Main, 84
- is_looped
 - menu_config, 120
- IS_PRINTABLE
 - Common, 59
- IS_UART_ERROR
 - Main, 82
- jump
 - hyp_ctx, 117
- LCD1602_CURSOR_BLINK_MAX
 - BSP LCD1602, 25
- LCD1602_CURSOR_BLINK_OFF
 - BSP LCD1602, 25
- LCD1602_CURSOR_BLINK_ON
 - BSP LCD1602, 25
- lcd1602_cursor_blink_state
 - BSP LCD1602, 23
- LCD1602_CURSOR_BLINK_UNDEF
 - BSP LCD1602, 25

LCD1602_CURSOR_MAX
BSP LCD1602, [25](#)

LCD1602_CURSOR_MOVE_LEFT
BSP LCD1602, [27](#)

LCD1602_CURSOR_MOVE_MAX
BSP LCD1602, [27](#)

LCD1602_CURSOR_MOVE_RIGHT
BSP LCD1602, [27](#)

LCD1602_CURSOR_MOVE_UNDEF
BSP LCD1602, [27](#)

LCD1602_CURSOR_OFF
BSP LCD1602, [25](#)

LCD1602_CURSOR_ON
BSP LCD1602, [25](#)

lcd1602_cursor_state
BSP LCD1602, [25](#)

LCD1602_CURSOR_UNDEF
BSP LCD1602, [25](#)

LCD1602_DATA_PINS
BSP LCD1602, [21](#)

lcd1602_data_pins
BSP LCD1602, [34](#)

lcd1602_disp_state
BSP LCD1602, [25](#)

LCD1602_DISPLAY_MAX
BSP LCD1602, [25](#)

LCD1602_DISPLAY_OFF
BSP LCD1602, [25](#)

LCD1602_DISPLAY_ON
BSP LCD1602, [25](#)

LCD1602_DISPLAY_UNDEF
BSP LCD1602, [25](#)

lcd1602_font_size
BSP LCD1602, [25](#)

LCD1602_FONT_SIZE_5X11
BSP LCD1602, [26](#)

LCD1602_FONT_SIZE_5X8
BSP LCD1602, [26](#)

LCD1602_FONT_SIZE_MAX
BSP LCD1602, [26](#)

LCD1602_FONT_SIZE_UNDEF
BSP LCD1602, [26](#)

LCD1602_INTERFACE_4BITS
BSP LCD1602, [27](#)

LCD1602_INTERFACE_8BITS
BSP LCD1602, [27](#)

LCD1602_INTERFACE_MAX
BSP LCD1602, [27](#)

LCD1602_INTERFACE_UNDEF
BSP LCD1602, [27](#)

lcd1602_num_line
BSP LCD1602, [26](#)

LCD1602_NUM_LINE_1
BSP LCD1602, [26](#)

LCD1602_NUM_LINE_2
BSP LCD1602, [26](#)

LCD1602_NUM_LINE_MAX
BSP LCD1602, [26](#)

LCD1602_NUM_LINE_UNDEF
BSP LCD1602, [26](#)

lcd1602_settings, [117](#)

LCD1602_SHIFT_CURSOR_LEFT
BSP LCD1602, [27](#)

LCD1602_SHIFT_CURSOR_RIGHT
BSP LCD1602, [27](#)

LCD1602_SHIFT_CURSOR_UNDEF
BSP LCD1602, [27](#)

LCD1602_SHIFT_DISPLAY_LEFT
BSP LCD1602, [27](#)

LCD1602_SHIFT_DISPLAY_RIGHT
BSP LCD1602, [27](#)

lcd1602_shift_entire_disp
BSP LCD1602, [26](#)

LCD1602_SHIFT_ENTIRE_MAX
BSP LCD1602, [26](#)

LCD1602_SHIFT_ENTIRE_NOT_PERFORMED
BSP LCD1602, [26](#)

LCD1602_SHIFT_ENTIRE_PERFORMED
BSP LCD1602, [26](#)

LCD1602_SHIFT_ENTIRE_UNDEF
BSP LCD1602, [26](#)

LCD1602_SHIFT_MAX
BSP LCD1602, [27](#)

lcd1602_type_interface
BSP LCD1602, [26](#)

lcd1602_type_move_cursor
BSP LCD1602, [27](#)

lcd1602_type_shift
BSP LCD1602, [27](#)

led_event
Application layer of RGB LED, [62](#)

LED_EVENT_COMMON_ERROR
Application layer of RGB LED, [62](#)

LED_EVENT_CRC_ERROR
Application layer of RGB LED, [62](#)

LED_EVENT_FAILED
Application layer of RGB LED, [62](#)

LED_EVENT_FLASH_ERROR
Application layer of RGB LED, [62](#)

LED_EVENT_IN_PROCESS
Application layer of RGB LED, [62](#)

LED_EVENT_IS_VALID
Application layer of RGB LED, [61](#)

LED_EVENT_LCD1602_ERROR
Application layer of RGB LED, [62](#)

LED_EVENT_MAX
Application layer of RGB LED, [62](#)

LED_EVENT_NONE
Application layer of RGB LED, [62](#)

LED_EVENT_SUCCESS
Application layer of RGB LED, [62](#)

LED_EVENT_UART_ERROR
Application layer of RGB LED, [62](#)

LED_EVENT_UART_OVERFLOW
Application layer of RGB LED, [62](#)

LIN_BREAK_MIN_LEN

- Algorithm of Sniffer RS-232, 99
- long_press_dur_ms
 - button_init_ctx, 113
- Main, 81
 - button_cb, 83
 - button_wait_event, 83
 - internal_error, 84
 - IS_UART_ERROR, 82
 - main, 84
 - uart_error_cb, 84
 - uart_lin_break_cb, 85
 - uart_overflow_cb, 85
- main
 - Main, 84
- main.c, 141
- MAX
 - Common, 60
- MemManage_Handler
 - Basic interrupts, 64
- Menu library, 85
 - __menu_enumerator_inc, 90
 - __menu_get_last_item, 90
 - __menu_item_is_in_menu, 90
 - __menu_redraw, 91
 - __menu_strlen, 91
 - menu_all_destroy, 92
 - menu_by_label_get, 92
 - MENU_COLOR_BLACK, 89
 - MENU_COLOR_BLUE, 89
 - MENU_COLOR_CONFIG_DEFAULT, 87
 - MENU_COLOR_CYAN, 89
 - MENU_COLOR_GREEN, 89
 - MENU_COLOR_MAGENTA, 89
 - MENU_COLOR_MAX, 89
 - MENU_COLOR_RED, 89
 - menu_color_type, 89
 - MENU_COLOR_WHITE, 89
 - MENU_COLOR_YELLOW, 89
 - menu_create, 92
 - menu_current_item_get, 93
 - menu_entry, 93
 - menu_exit, 94
 - menu_is_started, 94
 - menu_item_add, 94
 - menu_item_by_label_get, 95
 - menu_item_by_label_only_get, 95
 - menu_item_label_get, 96
 - menu_item_value_set, 96
 - MENU_NUM_DIGITAL, 89
 - MENU_NUM_LOWER_LETTER, 89
 - MENU_NUM_MAX, 89
 - MENU_NUM_NONE, 89
 - menu_num_type, 89
 - MENU_NUM_TYPE_IS_VALID, 88
 - MENU_NUM_UPPER_LETTER, 89
 - MENU_PASS_ALWAYS, 90
 - MENU_PASS_MAX, 90
 - MENU_PASS_NONE, 90
 - menu_pass_type, 89
 - MENU_PASS_TYPE_IS_VALID, 88
 - MENU_PASS_WITH_PROMPT, 90
 - menu_start, 96
 - menu.c, 143
 - menu.h, 131, 133
 - menu_all_destroy
 - Menu library, 92
 - menu_by_label_get
 - Menu library, 92
 - menu_color, 119
 - MENU_COLOR_BLACK
 - Menu library, 89
 - MENU_COLOR_BLUE
 - Menu library, 89
 - menu_color_config, 119
 - MENU_COLOR_CONFIG_DEFAULT
 - Menu library, 87
 - MENU_COLOR_CYAN
 - Menu library, 89
 - MENU_COLOR_GREEN
 - Menu library, 89
 - MENU_COLOR_MAGENTA
 - Menu library, 89
 - MENU_COLOR_MAX
 - Menu library, 89
 - MENU_COLOR_RED
 - Menu library, 89
 - menu_color_type
 - Menu library, 89
 - MENU_COLOR_WHITE
 - Menu library, 89
 - MENU_COLOR_YELLOW
 - Menu library, 89
 - menu_config, 120
 - indent, 120
 - is_looped, 120
 - num_delim, 120
 - num_type, 121
 - pass_type, 121
 - read_callback, 121
 - width, 121
 - write_callback, 121
 - menu_create
 - Menu library, 92
 - menu_current_item_get
 - Menu library, 93
 - menu_entry
 - Menu library, 93
 - menu_exit
 - Menu library, 94
 - menu_is_started
 - Menu library, 94
 - menu_item, 121
 - menu_item::menu, 118
 - menu_item_add
 - Menu library, 94
 - menu_item_by_label_get

- Menu library, [95](#)
- menu_item_by_label_only_get
 - Menu library, [95](#)
- menu_item_label_get
 - Menu library, [96](#)
- menu_item_value_set
 - Menu library, [96](#)
- MENU_NUM_DIGITAL
 - Menu library, [89](#)
- MENU_NUM_LOWER_LETTER
 - Menu library, [89](#)
- MENU_NUM_MAX
 - Menu library, [89](#)
- MENU_NUM_NONE
 - Menu library, [89](#)
- menu_num_type
 - Menu library, [89](#)
- MENU_NUM_TYPE_IS_VALID
 - Menu library, [88](#)
- MENU_NUM_UPPER_LETTER
 - Menu library, [89](#)
- MENU_PASS_ALWAYS
 - Menu library, [90](#)
- MENU_PASS_MAX
 - Menu library, [90](#)
- MENU_PASS_NONE
 - Menu library, [90](#)
- menu_pass_type
 - Menu library, [89](#)
- MENU_PASS_TYPE_IS_VALID
 - Menu library, [88](#)
- MENU_PASS_WITH_PROMPT
 - Menu library, [90](#)
- menu_start
 - Menu library, [96](#)
- MIN
 - Common, [60](#)
- NMI_Handler
 - Basic interrupts, [64](#)
- num_delim
 - menu_config, [120](#)
- NUM_LINE_IS_VALID
 - BSP LCD1602, [21](#)
- num_type
 - menu_config, [121](#)
- parity
 - hyp_ctx, [117](#)
- pass_type
 - menu_config, [121](#)
- PendSV_Handler
 - Basic interrupts, [64](#)
- presettings
 - flash_config, [115](#)
- press_delay_ms
 - button_init_ctx, [113](#)
- press_min_dur_ms
 - button_init_ctx, [113](#)
- read_callback
 - menu_config, [121](#)
- RS232_CHANNEL_ALL
 - Algorithm of Sniffer RS-232, [101](#)
- RS232_CHANNEL_ANY
 - Algorithm of Sniffer RS-232, [101](#)
- RS232_CHANNEL_MAX
 - Algorithm of Sniffer RS-232, [101](#)
- RS232_CHANNEL_RX
 - Algorithm of Sniffer RS-232, [101](#)
- RS232_CHANNEL_TX
 - Algorithm of Sniffer RS-232, [101](#)
- rs232_channel_type
 - Algorithm of Sniffer RS-232, [101](#)
- rs232_channel_type_str
 - CLI, [75](#)
- RS232_CHANNEL_TYPE_VALID
 - Algorithm of Sniffer RS-232, [99](#)
- rs232_interspace_type
 - Configuration, [78](#)
- rs232_interspace_type_str
 - CLI, [75](#)
- RS232_INTERSPACE_TYPE_VALID
 - Configuration, [77](#)
- RS232_INTERSPACE_MAX
 - Configuration, [80](#)
- RS232_INTERSPACE_NEW_LINE
 - Configuration, [80](#)
- RS232_INTERSPACE_NONE
 - Configuration, [80](#)
- RS232_INTERSPACE_SPACE
 - Configuration, [80](#)
- RS232_TRACE_HEX
 - Configuration, [80](#)
- RS232_TRACE_HYBRID
 - Configuration, [80](#)
- RS232_TRACE_MAX
 - Configuration, [80](#)
- rs232_trace_type
 - Configuration, [80](#)
- rs232_trace_type_str
 - CLI, [75](#)
- RS232_TRACE_TYPE_VALID
 - Configuration, [78](#)
- rx_buffer
 - Algorithm of Sniffer RS-232, [109](#)
- save_to_presettings
 - flash_config, [115](#)
- SHIFT_ENTIRE_IS_VALID
 - BSP LCD1602, [22](#)
- sniffer_rs232.c, [144](#)
- sniffer_rs232.h, [135](#), [136](#)
- sniffer_rs232_calc
 - Algorithm of Sniffer RS-232, [106](#)
- SNIFFER_RS232_CFG_PARAM_IS_VALID
 - Algorithm of Sniffer RS-232, [99](#)
- SNIFFER_RS232_CFG_PARAM_MAX
 - Algorithm of Sniffer RS-232, [100](#)

- SNIFFER_RS232_CFG_PARAM_MIN
 - Algorithm of Sniffer RS-232, [100](#)
 - sniffer_rs232_config, [122](#)
 - sniffer_rs232_config_check
 - Algorithm of Sniffer RS-232, [106](#)
 - SNIFFER_RS232_CONFIG_DEFAULT
 - Algorithm of Sniffer RS-232, [100](#)
 - sniffer_rs232_config_item_range
 - Algorithm of Sniffer RS-232, [107](#)
 - sniffer_rs232_deinit
 - Algorithm of Sniffer RS-232, [107](#)
 - sniffer_rs232_init
 - Algorithm of Sniffer RS-232, [107](#)
 - SVC_Handler
 - Basic interrupts, [65](#)
 - SysTick_Handler
 - Basic interrupts, [65](#)
 - TIM2_IRQHandler
 - BSP LED RGB, [41](#)
 - TIM7_IRQHandler
 - BSP button, [13](#)
 - TIM_APB_NUM_CLOCK_GET
 - BSP RCC, [42](#)
 - TIM_PERIOD_CALC
 - BSP button, [8](#)
 - TIM_TICK_TO_MS
 - BSP button, [8](#)
 - trace_type
 - flash_config, [115](#)
 - tx_buffer
 - Algorithm of Sniffer RS-232, [109](#)
 - txrx_delimiter
 - flash_config, [115](#)
 - TYPE_INTERFACE_IS_VALID
 - BSP LCD1602, [22](#)
 - TYPE_MOVE_CURSOR_IS_VALID
 - BSP LCD1602, [23](#)
 - TYPE_SHIFT_IS_VALID
 - BSP LCD1602, [23](#)
 - UART4_IRQHandler
 - BSP UART, [57](#)
 - uart_ctx, [123](#)
 - uart_error_cb
 - Main, [84](#)
 - uart_init_ctx, [124](#)
 - uart_lin_break_cb
 - Main, [85](#)
 - uart_obj
 - BSP UART, [57](#)
 - uart_overflow_cb
 - Main, [85](#)
 - uart_parity
 - BSP UART, [48](#)
 - uart_parity_str
 - CLI, [76](#)
 - UART_PARITY_VALID
 - BSP UART, [46](#)
 - uart_presettings, [125](#)
 - UART_PRESETTINGS_DEFAULT
 - Configuration, [78](#)
 - uart_stopbits
 - BSP UART, [48](#)
 - UART_STOPBITS_VALID
 - BSP UART, [47](#)
 - uart_type
 - BSP UART, [49](#)
 - UART_TYPE_VALID
 - BSP UART, [47](#)
 - uart_wordlen
 - BSP UART, [49](#)
 - UART_WORDLEN_VALID
 - BSP UART, [48](#)
 - UsageFault_Handler
 - Basic interrupts, [65](#)
 - USART2_IRQHandler
 - BSP UART, [57](#)
 - USART3_IRQHandler
 - BSP UART, [57](#)
 - width
 - menu_config, [121](#)
 - wordlen
 - hyp_ctx, [117](#)
 - write_callback
 - menu_config, [121](#)