

ПО для макета МПА (100 тема)

Создано системой Doxygen 1.8.20



1	Алфавитный указатель классов	1
1.1	Классы	1
2	Список файлов	3
2.1	Файлы	3
3	Классы	5
3.1	Структура <code>adc_config_data</code>	5
3.1.1	Подробное описание	6
3.1.2	Данные класса	6
3.1.2.1	<code>avg_num</code>	6
3.1.2.2	<code>ch_rx_num</code>	6
3.1.2.3	<code>init_flag</code>	6
3.1.2.4	<code>spi_struct</code>	6
3.1.2.5	<code>timer_n_capture</code>	6
3.2	Структура <code>ai_oper_mode_struct</code>	7
3.2.1	Подробное описание	7
3.2.2	Данные класса	7
3.2.2.1	<code>adc_chs_mode</code>	7
3.2.2.2	<code>reserv1</code>	7
3.3	Структура <code>bus_defect_struct</code>	7
3.3.1	Подробное описание	8
3.3.2	Данные класса	8
3.3.2.1	<code>fail_timeout</code>	8
3.3.2.2	<code>many_fail_packet</code>	8
3.3.2.3	<code>reserv</code>	8
3.4	Структура <code>cmd_header_struct</code>	8
3.4.1	Подробное описание	9
3.4.2	Данные класса	9
3.4.2.1	<code>cmd</code>	9
3.4.2.2	<code>length</code>	9
3.4.2.3	<code>result</code>	9
3.5	Структура <code>cmd_struct</code>	10
3.5.1	Подробное описание	10
3.5.2	Данные класса	10
3.5.2.1	<code>data</code>	10
3.5.2.2	<code>header</code>	11
3.6	Структура <code>common_register_space_ext_ram</code>	11
3.6.1	Подробное описание	12
3.6.2	Данные класса	12
3.6.2.1	<code>PLC_BusDefect_B1</code>	12
3.6.2.2	<code>PLC_BusDefect_B2</code>	12
3.6.2.3	<code>PLC_CM_State</code>	13
3.6.2.4	<code>PLC_CommonRomRegs</code>	13

3.6.2.5 PLC_Config	13
3.6.2.6 PLC_CorrPackFromDevice_B1	13
3.6.2.7 PLC_CorrPackFromDevice_B2	13
3.6.2.8 PLC_CorrPackToDevice_B1	13
3.6.2.9 PLC_CorrPackToDevice_B2	14
3.6.2.10 PLC_Durat	14
3.6.2.11 PLC_ErrPackFromDevice_B1	14
3.6.2.12 PLC_ErrPackFromDevice_B2	14
3.6.2.13 PLC_ErrPackToDevice_B1	14
3.6.2.14 PLC_ErrPackToDevice_B2	14
3.6.2.15 PLC_PMAAddr	15
3.6.2.16 PLC_PowerDefect	15
3.6.2.17 PLC_SelfDiagDefect	15
3.6.2.18 PLC_SoftVer	15
3.6.2.19 Reserv_1	15
3.7 Структура common_register_space_ext_rom	16
3.7.1 Подробное описание	17
3.7.2 Данные класса	17
3.7.2.1 PLC_BusConfig_B1	17
3.7.2.2 PLC_BusConfig_B2	17
3.7.2.3 PLC_DeviceInfo	17
3.7.2.4 PLC_DeviceType	17
3.7.2.5 PLC_DualControl	17
3.7.2.6 PLC_NumCrcErrorsForDefect_B1	18
3.7.2.7 PLC_NumCrcErrorsForDefect_B2	18
3.7.2.8 PLC_SerialNumber	18
3.7.2.9 PLC_TimeoutForDefect_B1	18
3.7.2.10 PLC_TimeoutForDefect_B2	18
3.7.2.11 PLC_TimeSoloWork	18
3.7.2.12 PLC_TimeToRepair	19
3.7.2.13 Reserv_2	19
3.8 Структура config_struct	19
3.8.1 Подробное описание	19
3.8.2 Данные класса	19
3.8.2.1 add_switch_1	20
3.8.2.2 add_switch_2	20
3.8.2.3 main_switch	20
3.8.2.4 reserv	20
3.9 Структура device_address_struct	20
3.9.1 Подробное описание	21
3.9.2 Данные класса	21
3.9.2.1 chassis_addr	21
3.9.2.2 module_addr	21

3.9.2.3 reserv	21
3.10 Структура device_type_struct	21
3.10.1 Подробное описание	22
3.10.2 Данные класса	22
3.10.2.1 batch	22
3.10.2.2 modification	22
3.10.2.3 reserv	22
3.10.2.4 revision	23
3.10.2.5 type	23
3.10.2.6 use_object	23
3.11 Структура heap_struct	23
3.11.1 Подробное описание	23
3.11.2 Данные класса	23
3.11.2.1 memory_page_space	24
3.11.2.2 memory_page_status	24
3.12 Структура list_head_struct	24
3.12.1 Подробное описание	24
3.12.2 Данные класса	25
3.12.2.1 next	25
3.12.2.2 prev	25
3.13 Структура mra_register_space_ext_ram	25
3.13.1 Подробное описание	26
3.13.2 Данные класса	26
3.13.2.1 AI_CodeADC	26
3.13.2.2 AI_DiagnosticChannel	26
3.13.2.3 AI_PhysQuantFloat	26
3.13.2.4 AI_RomRegs	27
3.13.2.5 AI_SignalChanged	27
3.13.2.6 Reserv_6	27
3.14 Структура mra_register_space_ext_rom	27
3.14.1 Подробное описание	28
3.14.2 Данные класса	28
3.14.2.1 AI_MaxCodeADC	28
3.14.2.2 AI_MetrologDat	29
3.14.2.3 AI_MinCodeADC	29
3.14.2.4 AI_NumForAverag	29
3.14.2.5 AI_OperMode	29
3.14.2.6 AI_PolynConst0	29
3.14.2.7 AI_PolynConst1	29
3.14.2.8 AI_PolynConst2	30
3.14.2.9 AI_PolynConst3	30
3.14.2.10 AI_PolynConst4	30
3.14.2.11 AI_PolynConst5	30

3.14.2.12 AI_PolynConst6 . . . . .	30
3.14.2.13 Reserv_3 . . . . .	30
3.14.2.14 Reserv_4 . . . . .	31
3.14.2.15 Reserv_5 . . . . .	31
3.15 Структура packet_header_struct . . . . .	31
3.15.1 Подробное описание . . . . .	31
3.15.2 Данные класса . . . . .	31
3.15.2.1 cmd_number . . . . .	32
3.15.2.2 header . . . . .	32
3.15.2.3 packet_length . . . . .	32
3.15.2.4 receiver_addr . . . . .	32
3.15.2.5 sender_addr . . . . .	32
3.15.2.6 service_byte . . . . .	32
3.16 Структура packet_tail_Struct . . . . .	33
3.16.1 Подробное описание . . . . .	33
3.16.2 Данные класса . . . . .	33
3.16.2.1 checksum . . . . .	33
3.16.2.2 end . . . . .	33
3.17 Структура plc_sof_ver_struct . . . . .	33
3.17.1 Подробное описание . . . . .	34
3.17.2 Данные класса . . . . .	34
3.17.2.1 add_info . . . . .	34
3.17.2.2 develop . . . . .	34
3.17.2.3 modification . . . . .	34
3.17.2.4 revision . . . . .	35
3.17.2.5 soft_ver . . . . .	35
3.17.2.6 type . . . . .	35
3.18 Структура power_failure_struct . . . . .	35
3.18.1 Подробное описание . . . . .	35
3.18.2 Данные класса . . . . .	36
3.18.2.1 reserv . . . . .	36
3.18.2.2 v1_3_3 . . . . .	36
3.18.2.3 v1_5 . . . . .	36
3.18.2.4 v2_3_3 . . . . .	36
3.18.2.5 v2_5 . . . . .	36
3.19 Структура ram_data_struct . . . . .	37
3.19.1 Подробное описание . . . . .	38
3.19.2 Данные класса . . . . .	38
3.19.2.1 common_ram_register_space . . . . .	38
3.19.2.2 crc_table . . . . .	38
3.19.2.3 mpa_ram_register_space . . . . .	38
3.19.2.4 packet_rx . . . . .	38
3.19.2.5 packet_tx . . . . .	39

3.19.2.6	Reserv	39
3.19.2.7	rx_packet_struct	39
3.19.2.8	service_byte_pm	39
3.19.2.9	service_byte_um	39
3.19.2.10	spi_1_rx_buffer	39
3.19.2.11	start_struct	40
3.19.2.12	tx_data	40
3.19.2.13	tx_packet_struct	40
3.19.2.14	uart1_rx_buffer	40
3.19.2.15	uart2_rx_buffer	40
3.20	Структура range_start_struct	40
3.20.1	Подробное описание	41
3.20.2	Данные класса	41
3.20.2.1	address	41
3.20.2.2	range_type	41
3.20.2.3	size	41
3.20.2.4	start_channel_num	42
3.21	Структура rom_data_struct	42
3.21.1	Подробное описание	42
3.21.2	Данные класса	43
3.21.2.1	common_rom_registers_space	43
3.21.2.2	mpa_rom_registers_space	43
3.22	Структура self_diag_struct	43
3.22.1	Подробное описание	44
3.22.2	Данные класса	44
3.22.2.1	fail_chanel	44
3.22.2.2	fail_crc_firmware	44
3.22.2.3	fail_download_rom	44
3.22.2.4	fail_firmware_1_bus	44
3.22.2.5	fail_firmware_2_bus	44
3.22.2.6	fail_firmware_ram	45
3.22.2.7	fail_soft_ver	45
3.22.2.8	power_fail	45
3.22.2.9	reserv	45
3.22.2.10	reserv1	45
3.23	Структура service_byte_struct_pm	45
3.23.1	Подробное описание	46
3.23.2	Данные класса	46
3.23.2.1	both_control	46
3.23.2.2	fail_bus_1	46
3.23.2.3	fail_bus_2	46
3.23.2.4	init	47
3.23.2.5	master	47

3.23.2.6 reserv_1 . . . . .	47
3.23.2.7 reserv_2 . . . . .	47
3.23.2.8 self_diagnostics_error . . . . .	47
3.24 Структура service_byte_struct_um . . . . .	47
3.24.1 Подробное описание . . . . .	48
3.24.2 Данные класса . . . . .	48
3.24.2.1 last_answer . . . . .	48
3.24.2.2 ready_to_control . . . . .	48
3.24.2.3 reserv . . . . .	48
3.25 Структура spi_config_data . . . . .	49
3.25.1 Подробное описание . . . . .	49
3.25.2 Данные класса . . . . .	49
3.25.2.1 buffer . . . . .	50
3.25.2.2 buffer_counter . . . . .	50
3.25.2.3 IRQn . . . . .	50
3.25.2.4 RST_CLK_PCLK_SPIIn . . . . .	50
3.25.2.5 SPI . . . . .	50
3.25.2.6 spi_dma_ch . . . . .	50
3.25.2.7 SSPx . . . . .	51
3.26 Структура spi_dma_params . . . . .	51
3.26.1 Подробное описание . . . . .	51
3.26.2 Данные класса . . . . .	51
3.26.2.1 dma_channel . . . . .	51
3.26.2.2 DMA_Channel_SPI_RX . . . . .	52
3.26.2.3 DMA_InitStructure_SPI_RX . . . . .	52
3.26.2.4 dma_irq_counter . . . . .	52
3.27 Структура start_struct_ext_ram . . . . .	52
3.27.1 Подробное описание . . . . .	53
3.27.2 Данные класса . . . . .	53
3.27.2.1 flag_change_struct . . . . .	53
3.27.2.2 length . . . . .	53
3.27.2.3 number_of_ranges . . . . .	53
3.27.2.4 ranges_in_start_struct . . . . .	54
3.27.2.5 text_info . . . . .	54
3.28 Структура timer_config_struct . . . . .	54
3.28.1 Подробное описание . . . . .	54
3.28.2 Данные класса . . . . .	54
3.28.2.1 sTIM_ChnInit . . . . .	55
3.28.2.2 timer_cnt . . . . .	55
3.28.2.3 TIMER_STATUS . . . . .	55
3.28.2.4 TIMERInitStruct . . . . .	55
3.28.2.5 TIMERx . . . . .	55
3.29 Структура timer_irq_list_struct . . . . .	56



3.29.1	Подробное описание	56
3.29.2	Данные класса	56
3.29.2.1	data	56
3.29.2.2	event	57
3.29.2.3	handler	57
3.29.2.4	list	57
3.30	Структура tx_rx_packet_struct	57
3.30.1	Подробное описание	58
3.30.2	Данные класса	58
3.30.2.1	cmd_with_data	58
3.30.2.2	packet_header	58
3.30.2.3	packet_tail	58
3.31	Структура uart_config_data	59
3.31.1	Подробное описание	60
3.31.2	Данные класса	60
3.31.2.1	buffer	60
3.31.2.2	buffer_count	60
3.31.2.3	IRQn	60
3.31.2.4	read_pos	60
3.31.2.5	RST_CLK_PCLK_UARTn	60
3.31.2.6	UART	61
3.31.2.7	uart_dma_ch	61
3.31.2.8	UART_HCLKdiv	61
3.31.2.9	uart_timeouts	61
3.31.2.10	UARTx	61
3.32	Структура uart_dma_params	61
3.32.1	Подробное описание	62
3.32.2	Данные класса	62
3.32.2.1	dma_channel	62
3.32.2.2	DMA_Channel_UART_RX	62
3.32.2.3	DMA_InitStructure_UART_RX	62
3.32.2.4	dma_irq_counter	63
3.33	Структура uart_timeouts	63
3.33.1	Подробное описание	63
3.33.2	Данные класса	64
3.33.2.1	read_timeout_flag	64
3.33.2.2	read_val_timeout	64
3.33.2.3	timer_n_timeout	64
3.33.2.4	write_timeout_flag	64
3.33.2.5	write_val_timeout	64
4	Файлы	65
4.1	Файл 1273pv19t.c	65

4.1.1	Подробное описание	66
4.1.2	Функции	66
4.1.2.1	adc_gpio_config()	66
4.1.2.2	adc_init()	67
4.1.2.3	adc_reset()	68
4.1.3	Переменные	68
4.1.3.1	adc_1	68
4.1.3.2	spi_1	68
4.1.3.3	timer_2	69
4.2	Файл 1273pv19t.h	69
4.2.1	Подробное описание	70
4.2.2	Макросы	70
4.2.2.1	PIN_ADC_MODE_A0	70
4.2.2.2	PIN_ADC_MODE_A1	71
4.2.2.3	PIN_ADC_NSS	71
4.2.2.4	PIN_ADC_RST	71
4.2.2.5	PIN_ADC_SDIFS_IRQ	71
4.2.2.6	PORT_ADC_MODE	71
4.2.2.7	PORT_ADC_NSS	71
4.2.2.8	PORT_ADC_RST	72
4.2.2.9	PORT_ADC_SDIFS_IRQ	72
4.2.3	Типы	72
4.2.3.1	adc_n	72
4.2.4	Функции	72
4.2.4.1	adc_init()	72
4.2.4.2	adc_reset()	73
4.3	Файл clock.c	74
4.3.1	Подробное описание	74
4.3.2	Функции	74
4.3.2.1	clock_init()	74
4.4	Файл clock.h	75
4.4.1	Подробное описание	75
4.4.2	Функции	75
4.4.2.1	clock_init()	76
4.5	Файл dma.c	76
4.5.1	Подробное описание	77
4.5.2	Функции	77
4.5.2.1	dma_common_init()	77
4.5.2.2	DMA_IRQHandler()	78
4.5.3	Переменные	79
4.5.3.1	adc_1	79
4.5.3.2	spi_1	79
4.5.3.3	spi_2	79

4.5.3.4 timer_1 . . . . .	79
4.5.3.5 timer_2 . . . . .	79
4.5.3.6 timer_3 . . . . .	79
4.5.3.7 uart_1 . . . . .	80
4.5.3.8 uart_2 . . . . .	80
4.6 Файл dma.h . . . . .	80
4.6.1 Подробное описание . . . . .	81
4.6.2 Функции . . . . .	81
4.6.2.1 dma_common_init() . . . . .	81
4.7 Файл ebc.c . . . . .	81
4.7.1 Подробное описание . . . . .	82
4.7.2 Функции . . . . .	82
4.7.2.1 ebc_gpio_config() . . . . .	82
4.7.2.2 ebc_init() . . . . .	83
4.8 Файл ebc.h . . . . .	84
4.8.1 Подробное описание . . . . .	85
4.8.2 Типы . . . . .	85
4.8.2.1 ebc_devices . . . . .	85
4.8.3 Перечисления . . . . .	85
4.8.3.1 devices . . . . .	85
4.8.4 Функции . . . . .	86
4.8.4.1 ebc_init() . . . . .	86
4.9 Файл external_ram.c . . . . .	87
4.9.1 Подробное описание . . . . .	87
4.9.2 Функции . . . . .	87
4.9.2.1 find_max_halfword() . . . . .	88
4.9.2.2 init_external_ram_space() . . . . .	88
4.9.3 Переменные . . . . .	89
4.9.3.1 polyn_ch_consts . . . . .	89
4.9.3.2 ram_space_pointer . . . . .	90
4.9.3.3 rom_space_pointer . . . . .	90
4.10 Файл external_ram.h . . . . .	90
4.10.1 Подробное описание . . . . .	93
4.10.2 Макросы . . . . .	94
4.10.2.1 EXT_RAM_START_ADDR . . . . .	94
4.10.2.2 PLC_CONFIG_ADD_SWITCH1 . . . . .	94
4.10.2.3 PLC_CONFIG_ADD_SWITCH2 . . . . .	94
4.10.2.4 PLC_CONFIG_MAIN_SWITCH . . . . .	94
4.10.2.5 PLC_CONFIG_RESERV . . . . .	94
4.10.2.6 PLC_PM_CHASSIS_ADDR . . . . .	94
4.10.2.7 PLC_PM_MODULE_ADDR . . . . .	95
4.10.2.8 PLC_SOFT_VER_ADD_INFO . . . . .	95
4.10.2.9 PLC_SOFT_VER_DEVELOP . . . . .	95

4.10.2.10 PLC_SOFT_VER_MODIFICATION . . . . .	95
4.10.2.11 PLC_SOFT_VER_REVISION . . . . .	95
4.10.2.12 PLC_SOFT_VER_SOFT_VER . . . . .	95
4.10.2.13 PLC_SOFT_VER_TYPE . . . . .	96
4.10.2.14 RAM_REGISTER_SPACE_START_ADDR . . . . .	96
4.10.2.15 RESET_BIT . . . . .	96
4.10.2.16 SET_BIT . . . . .	96
4.10.2.17 START_STRUCT_CHANGE_FLAG . . . . .	96
4.10.2.18 START_STRUCT_LENGTH . . . . .	96
4.10.2.19 START_STRUCT_NUMBER_OF_RANGES . . . . .	97
4.10.2.20 START_STRUCT_RANGE0_ADDR . . . . .	97
4.10.2.21 START_STRUCT_RANGE0_SIZE . . . . .	97
4.10.2.22 START_STRUCT_RANGE0_START_CH_NUM . . . . .	97
4.10.2.23 START_STRUCT_RANGE0_TYPE . . . . .	97
4.10.2.24 START_STRUCT_RANGE1_ADDR . . . . .	97
4.10.2.25 START_STRUCT_RANGE1_SIZE . . . . .	98
4.10.2.26 START_STRUCT_RANGE1_START_CH_NUM . . . . .	98
4.10.2.27 START_STRUCT_RANGE1_TYPE . . . . .	98
4.10.2.28 START_STRUCT_RANGE2_ADDR . . . . .	98
4.10.2.29 START_STRUCT_RANGE2_SIZE . . . . .	98
4.10.2.30 START_STRUCT_RANGE2_START_CH_NUM . . . . .	98
4.10.2.31 START_STRUCT_RANGE2_TYPE . . . . .	99
4.10.2.32 START_STRUCT_TEXT_INFO_ADDR . . . . .	99
4.10.2.33 TEST_BIT . . . . .	99
4.10.2.34 TIMER_NUM . . . . .	99
4.10.3 Типы . . . . .	99
4.10.3.1 bus_defect . . . . .	99
4.10.3.2 common_ram_registers . . . . .	99
4.10.3.3 device_address . . . . .	100
4.10.3.4 device_config . . . . .	100
4.10.3.5 module_type . . . . .	100
4.10.3.6 mpa_ram_registers . . . . .	100
4.10.3.7 plc_soft_ver . . . . .	100
4.10.3.8 power_failure . . . . .	100
4.10.3.9 ram_data . . . . .	101
4.10.3.10 ram_start_struct . . . . .	101
4.10.3.11 range . . . . .	101
4.10.3.12 self_diag . . . . .	101
4.10.3.13 service_struct_pm . . . . .	101
4.10.3.14 service_struct_um . . . . .	101
4.10.4 Перечисления . . . . .	101
4.10.4.1 type_of_module . . . . .	101
4.10.5 Функции . . . . .	102

4.10.5.1	<code>init_external_ram_space()</code>	102
4.11	Файл <code>external_rom.c</code>	104
4.11.1	Подробное описание	104
4.11.2	Макросы	105
4.11.2.1	<code>FIRST_TIME_INIT</code>	105
4.11.3	Функции	105
4.11.3.1	<code>erase_rom()</code>	105
4.11.3.2	<code>init_external_rom_space()</code>	106
4.11.3.3	<code>memcpy_to_rom()</code>	106
4.11.3.4	<code>read_byte_rom()</code>	107
4.11.3.5	<code>write_byte_rom()</code>	107
4.11.4	Переменные	107
4.11.4.1	<code>polyn_ch_consts</code>	108
4.12	Файл <code>external_rom.h</code>	108
4.12.1	Подробное описание	110
4.12.2	Макросы	110
4.12.2.1	<code>BATCH</code>	110
4.12.2.2	<code>DEV_INFO</code>	110
4.12.2.3	<code>DEV_TYPE</code>	111
4.12.2.4	<code>DEV_TYPE_RESERV</code>	111
4.12.2.5	<code>DUAL_CONTROL</code>	111
4.12.2.6	<code>EXT_ROM_START_ADDR</code>	111
4.12.2.7	<code>HWREG</code>	111
4.12.2.8	<code>MAX_CODE_ADC</code>	111
4.12.2.9	<code>METROLOG_DAT</code>	112
4.12.2.10	<code>MIN_CODE_ADC</code>	112
4.12.2.11	<code>MODIFICATION</code>	112
4.12.2.12	<code>NUM_CRC_ERRORS_FOR_DEFECT_B1</code>	112
4.12.2.13	<code>NUM_CRC_ERRORS_FOR_DEFECT_B2</code>	112
4.12.2.14	<code>NUM_FOR_AVERAGE</code>	112
4.12.2.15	<code>REVISION</code>	113
4.12.2.16	<code>ROM_REGISTER_SPACE_START_ADDR</code>	113
4.12.2.17	<code>SERIAL_NUMBER</code>	113
4.12.2.18	<code>TIME_SOLO_WORK</code>	113
4.12.2.19	<code>TIME_TO_REPAIR</code>	113
4.12.2.20	<code>TIMEOUT_FOR_DEFECT_B1</code>	113
4.12.2.21	<code>TIMEOUT_FOR_DEFECT_B2</code>	114
4.12.3	Типы	114
4.12.3.1	<code>ai_oper_mode</code>	114
4.12.3.2	<code>common_rom_registers</code>	114
4.12.3.3	<code>device_type</code>	114
4.12.3.4	<code>mpa_rom_registers</code>	114
4.12.3.5	<code>rom_data</code>	114

4.12.4	Функции	115
4.12.4.1	erase_rom()	115
4.12.4.2	init_external_rom_space()	115
4.12.4.3	memcpy_to_rom()	116
4.12.4.4	read_byte_rom()	116
4.12.4.5	write_byte_rom()	117
4.13	Файл internal_ram.c	118
4.13.1	Подробное описание	118
4.13.2	Функции	118
4.13.2.1	free_ram_pages()	118
4.13.2.2	malloc_ram_pages()	119
4.13.3	Переменные	119
4.13.3.1	heap_ptr	119
4.14	Файл internal_ram.h	119
4.14.1	Подробное описание	120
4.14.2	Макросы	121
4.14.2.1	PAGE_NUM	121
4.14.2.2	PAGE_SIZE	121
4.14.3	Типы	121
4.14.3.1	int_ram_heap	121
4.14.4	Функции	121
4.14.4.1	free_ram_pages()	121
4.14.4.2	malloc_ram_pages()	122
4.15	Файл leds.c	123
4.15.1	Подробное описание	123
4.15.2	Функции	123
4.15.2.1	leds_gpio_config()	123
4.16	Файл leds.h	124
4.16.1	Подробное описание	125
4.16.2	Макросы	125
4.16.2.1	CLOCK_LEDS	125
4.16.2.2	PIN_LED_ERROR_WORK	125
4.16.2.3	PIN_LED_OK_WORK	125
4.16.2.4	PORT_LEDS	125
4.16.2.5	RESET_LED_ERROR_WORK	126
4.16.2.6	RESET_LED_OK_WORK	126
4.16.2.7	SET_LED_ERROR_WORK	126
4.16.2.8	SET_LED_OK_WORK	126
4.16.3	Функции	126
4.16.3.1	leds_gpio_config()	126
4.17	Файл list.c	127
4.17.1	Подробное описание	127
4.17.2	Функции	127

4.17.2.1	<code>__list_add()</code>	128
4.17.2.2	<code>__list_del()</code>	128
4.17.2.3	<code>init_list_head()</code>	128
4.17.2.4	<code>list_add()</code>	128
4.17.2.5	<code>list_add_tail()</code>	129
4.17.2.6	<code>list_del()</code>	129
4.17.2.7	<code>list_empty()</code>	129
4.17.2.8	<code>list_is_last()</code>	129
4.18	Файл <code>list.h</code>	130
4.18.1	Подробное описание	131
4.18.2	Макросы	131
4.18.2.1	<code>container_of</code>	131
4.18.2.2	<code>list_entry</code>	132
4.18.2.3	<code>list_for_each</code>	132
4.18.2.4	<code>list_for_each_entry</code>	132
4.18.2.5	<code>LIST_HEAD</code>	132
4.18.2.6	<code>LIST_HEAD_INIT</code>	132
4.18.3	Типы	133
4.18.3.1	<code>list_head</code>	133
4.18.4	Функции	133
4.18.4.1	<code>__list_add()</code>	133
4.18.4.2	<code>__list_del()</code>	133
4.18.4.3	<code>init_list_head()</code>	134
4.18.4.4	<code>list_add()</code>	134
4.18.4.5	<code>list_add_tail()</code>	134
4.18.4.6	<code>list_del()</code>	135
4.18.4.7	<code>list_empty()</code>	135
4.18.4.8	<code>list_is_last()</code>	136
4.19	Файл <code>main.c</code>	136
4.19.1	Функции	137
4.19.1.1	<code>do_mpa_task()</code>	137
4.19.1.2	<code>main()</code>	138
4.19.1.3	<code>receive_adc_chanel_pack()</code>	140
4.19.1.4	<code>request_data()</code>	140
4.19.1.5	<code>sync_adc_chanel()</code>	141
4.19.2	Переменные	142
4.19.2.1	<code>adc_1</code>	142
4.19.2.2	<code>heap</code>	142
4.19.2.3	<code>heap_ptr</code>	142
4.19.2.4	<code>ram_space_pointer</code>	142
4.19.2.5	<code>rom_space_pointer</code>	143
4.19.2.6	<code>spi_1</code>	143
4.19.2.7	<code>spi_2</code>	143

4.19.2.8 timer_1	143
4.19.2.9 timer_2	143
4.19.2.10 timer_3	143
4.19.2.11 tmr_handler_head	144
4.19.2.12 uart_1	144
4.19.2.13 uart_2	144
4.20 Файл main.h	144
4.20.1 Подробное описание	145
4.20.2 Функции	145
4.20.2.1 do_mpa_task()	145
4.20.2.2 receive_adc_chanel_pack()	146
4.20.2.3 request_data()	147
4.20.2.4 sync_adc_channels()	148
4.21 Файл mdr32_drivers.h	148
4.21.1 Подробное описание	149
4.21.2 Макросы	149
4.21.2.1 CHANEL_NUMBER	150
4.21.2.2 HSE_OSC	150
4.21.2.3 K1986VE1T	150
4.21.2.4 MAX_CHANEL_NUMBER	150
4.21.2.5 PM_CHASSIS_ADDR	150
4.21.2.6 PM_DEV_ADDR	150
4.21.2.7 WORK_FREQ	151
4.22 Файл rs422_protocol.c	151
4.22.1 Подробное описание	152
4.22.2 Функции	152
4.22.2.1 crc32()	152
4.22.2.2 fill_crc32_table()	152
4.22.2.3 protocol_do_cmds()	152
4.22.2.4 receive_packet()	156
4.22.2.5 rx_error_handler()	158
4.22.2.6 transmit_packet()	159
4.22.2.7 um_service_byte_handler()	160
4.22.3 Переменные	161
4.22.3.1 ram_space_pointer	162
4.22.3.2 rom_space_pointer	162
4.23 Файл rs422_protocol.h	162
4.23.1 Подробное описание	164
4.23.2 Макросы	164
4.23.2.1 CONFIG	165
4.23.2.2 INIT_CMD	165
4.23.2.3 NUMBER_CMDS_IN_PACKET	165
4.23.2.4 PACKET_HEAD	165



4.23.2.5	PACKET_TAIL	165
4.23.2.6	PLC_CM_CRITICAL_FAULT	165
4.23.2.7	PLC_CM_INIT_1_BUS	166
4.23.2.8	PLC_CM_INIT_2_BUS	166
4.23.2.9	PLC_CM_NOT_INIT	166
4.23.2.10	PLC_CM_REMOVE_INIT	166
4.23.2.11	PLC_CM_UNKNOWN_STATE	166
4.23.2.12	READ_CMD	166
4.23.2.13	RESET_CMD	167
4.23.2.14	TYPE_CMD	167
4.23.2.15	WRITE_CMD	167
4.23.3	Типы	167
4.23.3.1	fields_cmd	167
4.23.3.2	fields_cmd_header	167
4.23.3.3	fields_packet	167
4.23.3.4	fields_packet_header	168
4.23.3.5	fields_packet_tail	168
4.23.3.6	protocol_error	168
4.23.4	Перечисления	168
4.23.4.1	protocol_errors	168
4.23.5	Функции	169
4.23.5.1	crc32()	169
4.23.5.2	fill_crc32_table()	169
4.23.5.3	protocol_do_cmds()	169
4.23.5.4	receive_packet()	173
4.23.5.5	rx_error_handler()	175
4.23.5.6	transmit_packet()	176
4.23.5.7	um_service_byte_handler()	177
4.24	Файл spi.c	179
4.24.1	Подробное описание	179
4.24.2	Функции	180
4.24.2.1	dma_spi_rx_init()	180
4.24.2.2	spi_clean_fifo_rx_buf()	180
4.24.2.3	spi_gpio_config()	181
4.24.2.4	spi_init()	181
4.24.2.5	spi_receive_halfword()	182
4.24.2.6	spi_transmit_halfword()	182
4.24.2.7	spi_transmit_message()	183
4.24.3	Переменные	183
4.24.3.1	spi_1	183
4.24.3.2	spi_2	183
4.25	Файл spi.h	184
4.25.1	Подробное описание	185

4.25.2	Макросы . . . . .	185
4.25.2.1	FIFO_SIZE . . . . .	185
4.25.2.2	PIN_SSP1_RX . . . . .	185
4.25.2.3	PIN_SSP1_SCK . . . . .	186
4.25.2.4	PIN_SSP1_SS . . . . .	186
4.25.2.5	PIN_SSP1_TX . . . . .	186
4.25.2.6	PORT_SSP1 . . . . .	186
4.25.2.7	SPI_BUFFER_SIZE . . . . .	186
4.25.3	Типы . . . . .	186
4.25.3.1	spi_n . . . . .	186
4.25.3.2	spi_n_dma_ch_params . . . . .	187
4.25.4	Функции . . . . .	187
4.25.4.1	dma_spi_rx_init() . . . . .	187
4.25.4.2	spi_clean_fifo_rx_buf() . . . . .	187
4.25.4.3	spi_init() . . . . .	188
4.25.4.4	spi_receive_halfword() . . . . .	188
4.25.4.5	spi_transmit_halfword() . . . . .	189
4.25.4.6	spi_transmit_message() . . . . .	189
4.26	Файл timers.c . . . . .	190
4.26.1	Подробное описание . . . . .	191
4.26.2	Функции . . . . .	191
4.26.2.1	delay_micro() . . . . .	191
4.26.2.2	delay_milli() . . . . .	191
4.26.2.3	list_tmr_handler_add_tail() . . . . .	192
4.26.2.4	list_tmr_handler_init() . . . . .	192
4.26.2.5	timer1_init() . . . . .	193
4.26.2.6	timer2_init() . . . . .	193
4.26.2.7	TIMER2_IRQHandler() . . . . .	194
4.26.2.8	timer3_init() . . . . .	194
4.26.2.9	timer_init() . . . . .	194
4.26.3	Переменные . . . . .	195
4.26.3.1	adc_1 . . . . .	195
4.26.3.2	ram_space_pointer . . . . .	195
4.26.3.3	spi_1 . . . . .	195
4.26.3.4	timer_1 . . . . .	195
4.26.3.5	timer_2 . . . . .	195
4.26.3.6	timer_3 . . . . .	196
4.26.3.7	tmr_handler_head . . . . .	196
4.27	Файл timers.h . . . . .	196
4.27.1	Подробное описание . . . . .	197
4.27.2	Типы . . . . .	197
4.27.2.1	timer_irq_list . . . . .	197
4.27.2.2	timer_n . . . . .	198

4.27.3	Функции	198
4.27.3.1	delay_micro()	198
4.27.3.2	delay_milli()	198
4.27.3.3	list_tmr_handler_add_tail()	198
4.27.3.4	list_tmr_handler_init()	199
4.27.3.5	timer_init()	199
4.28	Файл uart.c	200
4.28.1	Подробное описание	201
4.28.2	Функции	201
4.28.2.1	DMA_UART_RX_init()	201
4.28.2.2	UART1_IRQHandler()	202
4.28.2.3	UART2_IRQHandler()	202
4.28.2.4	uart_clean()	202
4.28.2.5	uart_get_buf_counter()	202
4.28.2.6	uart_gpio_config()	203
4.28.2.7	uart_init()	204
4.28.2.8	uart_read()	205
4.28.2.9	uart_read_pos()	206
4.28.2.10	uart_set_pos()	206
4.28.2.11	uart_set_read_timeout()	207
4.28.2.12	uart_set_write_timeout()	207
4.28.2.13	uart_write()	208
4.28.3	Переменные	209
4.28.3.1	uart_1	209
4.28.3.2	uart_2	209
4.29	Файл uart.h	209
4.29.1	Подробное описание	212
4.29.2	Макросы	212
4.29.2.1	BUFFER_MASK	212
4.29.2.2	GET_UART_BUF_PTR	212
4.29.2.3	PIN_UART1_EN	212
4.29.2.4	PIN_UART1_RX	212
4.29.2.5	PIN_UART1_TX	212
4.29.2.6	PIN_UART2_EN	213
4.29.2.7	PIN_UART2_RX	213
4.29.2.8	PIN_UART2_TX	213
4.29.2.9	PORT_UART1	213
4.29.2.10	PORT_UART1_EN	213
4.29.2.11	PORT_UART2	213
4.29.2.12	PORT_UART2_EN	214
4.29.2.13	RECOGNIZE_BUS	214
4.29.2.14	UART_BUFFER_SIZE	214
4.29.3	Типы	214

4.29.3.1	<code>uart_dma_ch_params</code>	214
4.29.3.2	<code>uart_errors</code>	214
4.29.3.3	<code>uart_n</code>	215
4.29.3.4	<code>uart_rx_tx_timeouts</code>	215
4.29.4	Перечисления	215
4.29.4.1	<code>errors</code>	215
4.29.5	Функции	215
4.29.5.1	<code>DMA_UART_RX_init()</code>	216
4.29.5.2	<code>uart_clean()</code>	216
4.29.5.3	<code>uart_get_buf_counter()</code>	217
4.29.5.4	<code>uart_init()</code>	217
4.29.5.5	<code>uart_read()</code>	218
4.29.5.6	<code>uart_read_pos()</code>	219
4.29.5.7	<code>uart_set_pos()</code>	220
4.29.5.8	<code>uart_set_read_timeout()</code>	220
4.29.5.9	<code>uart_set_write_timeout()</code>	221
4.29.5.10	<code>uart_write()</code>	221
	Предметный указатель	223

# Глава 1

## Алфавитный указатель классов

### 1.1 Классы

Классы с их кратким описанием.

<a href="#">adc_config_data</a>	Структура с конфигурационными параметрами АЦП . . . . .	5
<a href="#">ai_oper_mode_struct</a>	Структура с битовыми полями для регистра режим работы канала . . . . .	7
<a href="#">bus_defect_struct</a>	Структура с битовыми полями для регистра неисправность шины . . . . .	7
<a href="#">cmd_header_struct</a>	Структура с заголовком для каждой команды (субпакета) внутри одного пакета . . . . .	8
<a href="#">cmd_struct</a>	Структура с полями данных для каждой команды (субпакета) внутри одного пакета . . . . .	10
<a href="#">common_register_space_ext_ram</a>	Организация пространства общих регистров во внешнем ОЗУ . . . . .	11
<a href="#">common_register_space_ext_rom</a>	Организация пространства общих регистров для зеркализации во внешнем ПЗУ . . . . .	16
<a href="#">config_struct</a>	Структура с битовыми полями для регистра конфигурация . . . . .	19
<a href="#">device_address_struct</a>	Структура с битовыми полями для регистра адрес устройства . . . . .	20
<a href="#">device_type_struct</a>	Структура с битовыми полями для регистра тип устройства . . . . .	21
<a href="#">heap_struct</a>	Реализация "самодельной" кучи . . . . .	23
<a href="#">list_head_struct</a>	Структура с описанием двусвязанного списка . . . . .	24
<a href="#">mpa_register_space_ext_ram</a>	Организация пространства регистров МПА во внешнем ОЗУ . . . . .	25
<a href="#">mpa_register_space_ext_rom</a>	Организация пространства регистров МПА для зеркализации во внешнем ПЗУ . . . . .	27
<a href="#">packet_header_struct</a>	Структура с полями заголовка пакета . . . . .	31
<a href="#">packet_tail_Struct</a>	Структура с полями конца пакета . . . . .	33
<a href="#">plc_sof_ver_struct</a>	Структура с битовыми полями для регистра версия ПО . . . . .	33

<a href="#">power_failure_struct</a>	Структура с битовыми полями для регистра неисправность питания . . . . .	35
<a href="#">ram_data_struct</a>	Структура организующая память во внешнем ОЗУ . . . . .	37
<a href="#">range_start_struct</a>	Структура одного диапазона для стартовой структуры . . . . .	40
<a href="#">rom_data_struct</a>	Структура организующая память во внешнем ПЗУ . . . . .	42
<a href="#">self_diag_struct</a>	Структура с битовыми полями для регистра неисправность самодиагностики . .	43
<a href="#">service_byte_struct_pm</a>	Структура с битовыми полями для сервисного байта ПМ . . . . .	45
<a href="#">service_byte_struct_um</a>	Структура с битовыми полями для сервисного байта УМ . . . . .	47
<a href="#">spi_config_data</a>	Структура с конфигурационными параметрами SPI . . . . .	49
<a href="#">spi_dma_params</a>	Структура с параметрами DMA канала SPIn . . . . .	51
<a href="#">start_struct_ext_ram</a>	Структура, которая лежит в начале ОЗУ любого модуля . . . . .	52
<a href="#">timer_config_struct</a>	Структура с конфигурационными параметрами Таймеров . . . . .	54
<a href="#">timer_irq_list_struct</a>	Структура-реализация односвязанного списка обработчиков прерываний таймеров	56
<a href="#">tx_rx_packet_struct</a>	Структура пакета данных согласно утвержденному протоколу обмена данными .	57
<a href="#">uart_config_data</a>	Структура с конфигурационными параметрами UART и буфером приема . . . .	59
<a href="#">uart_dma_params</a>	Структура с параметрами DMA канала UARTn . . . . .	61
<a href="#">uart_timeouts</a>	Структура с таймаутами UARTn . . . . .	63

## Глава 2

# Список файлов

### 2.1 Файлы

Полный список файлов.

<a href="#">1273pv19t.c</a>	Файл с реализацией API для работы с АЦП 1273ПВ19Т . . . . .	65
<a href="#">1273pv19t.h</a>	Заголовочный файл с описанием API для работы с микросхемой АЦП 1273ПВ19Т	69
<a href="#">clock.c</a>	Файл с реализацией API для настройки тактирования МК . . . . .	74
<a href="#">clock.h</a>	Заголовочный файл с описанием API для настройки тактирования МК . . . . .	75
<a href="#">dma.c</a>	Файл с реализацией API для работы с DMA . . . . .	76
<a href="#">dma.h</a>	Заголовочный файл с описанием API для работы с DMA . . . . .	80
<a href="#">ebc.c</a>	Файл с реализацией API для работы с ЕВС . . . . .	81
<a href="#">ebc.h</a>	Заголовочный файл с описанием API для работы с ЕВС (контроллер внешней системной шины) . . . . .	84
<a href="#">external_ram.c</a>	Файл с реализацией API для работы с областью памяти внешнего ОЗУ . . . . .	87
<a href="#">external_ram.h</a>	Заголовочный файл с описанием API для работы с областью памяти внешнего ОЗУ	90
<a href="#">external_rom.c</a>	Файл с реализацией API для работы с областью памяти внешнего ПЗУ . . . . .	104
<a href="#">external_rom.h</a>	Заголовочный файл с описанием API для работы с областью памяти внешнего ПЗУ	108
<a href="#">internal_ram.c</a>	Файл с реализацией API для работы с областью памяти внутреннего ОЗУ . . . . .	118
<a href="#">internal_ram.h</a>	Заголовочный файл с описанием API для работы с областью памяти внутреннего ОЗУ . . . . .	119
<a href="#">leds.c</a>	Файл с реализацией API для работы со светодиодными индикаторами . . . . .	123
<a href="#">leds.h</a>	Заголовочный файл с описанием API для работы со светодиодными индикаторами	124

<a href="#">list.c</a>	Файл с реализацией API для работы со списками (реализация аналогична linux kernel) . . . . .	127
<a href="#">list.h</a>	Заголовочный файл с описанием API для работы со списками (реализация аналогична linux kernel) . . . . .	130
<a href="#">main.c</a>	. . . . .	136
<a href="#">main.h</a>	Заголовочный файл с реализацией основного функционала МПА . . . . .	144
<a href="#">mdr32_drivers.h</a>	Заголовочный файл с глобальными константами и подключаемыми библиотеками	148
<a href="#">rs422_protocol.c</a>	Файл с реализацией API протокола обмена данными по интерфейсу RS-422 . . .	151
<a href="#">rs422_protocol.h</a>	Заголовочный файл с описанием API протокола обмена данными по интерфейсу RS-422 . . . . .	162
<a href="#">spi.c</a>	Файл с реализацией API для работы с SPI . . . . .	179
<a href="#">spi.h</a>	Заголовочный файл с описанием API для работы с SPI . . . . .	184
<a href="#">timers.c</a>	Файл с реализацией API для работы с таймерами . . . . .	190
<a href="#">timers.h</a>	Заголовочный файл с описанием API для работы с таймерами . . . . .	196
<a href="#">uart.c</a>	Файл с реализацией API для работы с UART . . . . .	200
<a href="#">uart.h</a>	Заголовочный файл с описанием API для работы с UART . . . . .	209



## Глава 3

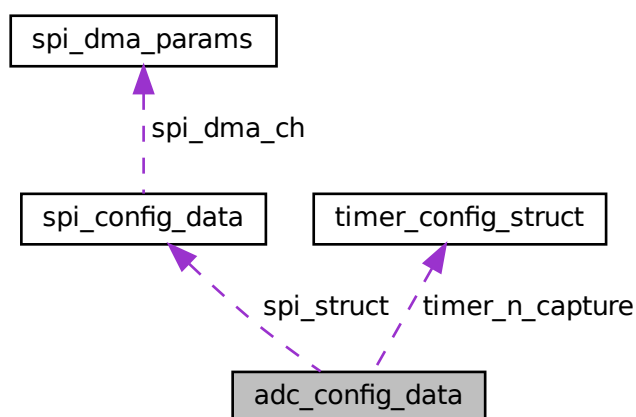
# Классы

### 3.1 Структура adc\_config\_data

Структура с конфигурационными параметрами АЦП

```
#include <1273pv19t.h>
```

Граф связей класса adc\_config\_data:



Открытые атрибуты

- `spi_n * spi_struct`  
SPI, по которому подключен АЦП
- `timer_n * timer_n_capture`  
Выбор таймера для режима захвата сигнала SDIFS/SDOFS.
- `uint8_t init_flag`  
Флаг инициализации АЦП
- `uint16_t avg_num`  
Кол-во выборок для усреднения
- `uint8_t ch_rx_num`  
Кол-во принятых каналов в одном пакете из CHANNEL\_NUMBER возможных

### 3.1.1 Подробное описание

Структура с конфигурационными параметрами АЦП

### 3.1.2 Данные класса

#### 3.1.2.1 avg\_num

`uint16_t adc_config_data::avg_num`

Кол-во выборок для усреднения

#### 3.1.2.2 ch\_rx\_num

`uint8_t adc_config_data::ch_rx_num`

Кол-во принятых каналов в одном пакете из CHANNEL\_NUMBER возможных

#### 3.1.2.3 init\_flag

`uint8_t adc_config_data::init_flag`

Флаг инициализации АЦП

#### 3.1.2.4 spi\_struct

`spi_n* adc_config_data::spi_struct`

SPI, по которому подключен АЦП

#### 3.1.2.5 timer\_n\_capture

`timer_n* adc_config_data::timer_n_capture`

Выбор таймера для режима захвата сигнала SDIFS/SDOFS.

Объявления и описания членов структуры находятся в файле:

- [1273pv19t.h](#)

## 3.2 Структура ai\_oper\_mode\_struct

Структура с битовыми полями для регистра режим работы канала

```
#include <external_rom.h>
```

Открытые атрибуты

- unsigned [adc\\_chs\\_mode](#): 8  
Режим работы каналов
- unsigned [reserv1](#): 8  
Резерв

### 3.2.1 Подробное описание

Структура с битовыми полями для регистра режим работы канала

### 3.2.2 Данные класса

#### 3.2.2.1 adc\_chs\_mode

```
unsigned ai_oper_mode_struct::adc_chs_mode
```

Режим работы каналов

#### 3.2.2.2 reserv1

```
unsigned ai_oper_mode_struct::reserv1
```

Резерв

Объявления и описания членов структуры находятся в файле:

- [external\\_rom.h](#)

## 3.3 Структура bus\_defect\_struct

Структура с битовыми полями для регистра неисправность шины

```
#include <external_ram.h>
```

## Открытые атрибуты

- unsigned [many\\_fail\\_packet](#): 1  
Количество битых пакетов подряд > уст. значения
- unsigned [fail\\_timeout](#): 1  
Неисправность по таймауту
- unsigned [reserv](#): 14  
Резерв

### 3.3.1 Подробное описание

Структура с битовыми полями для регистра неисправность шины

### 3.3.2 Данные класса

#### 3.3.2.1 fail\_timeout

unsigned bus\_defect\_struct::fail\_timeout

Неисправность по таймауту

#### 3.3.2.2 many\_fail\_packet

unsigned bus\_defect\_struct::many\_fail\_packet

Количество битых пакетов подряд > уст. значения

#### 3.3.2.3 reserv

unsigned bus\_defect\_struct::reserv

Резерв

Объявления и описания членов структуры находятся в файле:

- [external\\_ram.h](#)

## 3.4 Структура cmd\_header\_struct

Структура с заголовком для каждой команды (субпакета) внутри одного пакета

```
#include <rs422_protocol.h>
```

## Открытые атрибуты

- `uint8_t cmd`  
Команда
- `uint16_t result`  
Результат выполнения команды
- `uint16_t length`  
Длина команды

### 3.4.1 Подробное описание

Структура с заголовком для каждой команды (субпакета) внутри одного пакета

### 3.4.2 Данные класса

#### 3.4.2.1 cmd

`uint8_t cmd_header_struct::cmd`

Команда

#### 3.4.2.2 length

`uint16_t cmd_header_struct::length`

Длина команды

#### 3.4.2.3 result

`uint16_t cmd_header_struct::result`

Результат выполнения команды

Объявления и описания членов структуры находятся в файле:

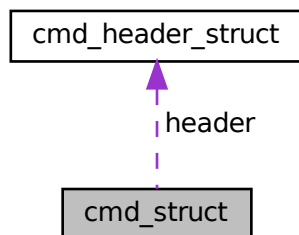
- [rs422\\_protocol.h](#)

## 3.5 Структура cmd\_struct

Структура с полями данных для каждой команды (субпакета) внутри одного пакета

```
#include <rs422_protocol.h>
```

Граф связей класса cmd\_struct:



Открытые атрибуты

- `fields_cmd_header header`  
Заголовок
- `uint8_t * data`  
Данные

### 3.5.1 Подробное описание

Структура с полями данных для каждой команды (субпакета) внутри одного пакета

### 3.5.2 Данные класса

#### 3.5.2.1 data

```
uint8_t* cmd_struct::data
```

Данные

## 3.5.2.2 header

`fields_cmd_header` `cmd_struct::header`

Заголовок

Объявления и описания членов структуры находятся в файле:

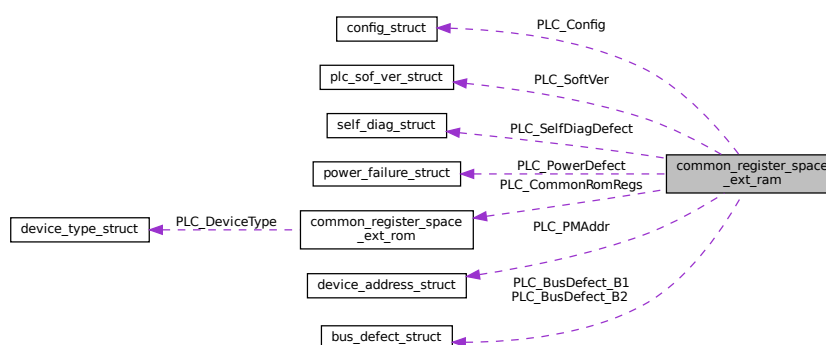
- `rs422_protocol.h`

## 3.6 Структура common\_register\_space\_ext\_ram

Организация пространства общих регистров во внешнем ОЗУ

```
#include <external_ram.h>
```

Граф связей класса `common_register_space_ext_ram`:



Открытые атрибуты

- `plc_soft_ver` `PLC_SoftVer`  
Версия ПО
- `device_config` `PLC_Config`  
Конфигурация устройства
- `device_address` `PLC_PMAAddr`  
Адрес устройства
- `uint32_t` `PLC_Durat`  
Время с момента запуска, с
- `uint32_t` `PLC_CM_State`  
Состояние автомата выбора УМ
- `uint32_t` `PLC_CorrPackFromDevice_B1`  
Корректных пакетов по Ш1, от устройства
- `uint32_t` `PLC_CorrPackToDevice_B1`  
Корректных пакетов по Ш1, к устройству
- `uint32_t` `PLC_ErrPackToDevice_B1`

- Ошибок приема пакета по Ш1.
- uint32\_t [PLC\\_ErrPackFromDevice\\_B1](#)
- Ошибок отправки пакета по Ш1.
- uint32\_t [PLC\\_CorrPackFromDevice\\_B2](#)
- Корректных пакетов по Ш2, от устройства
- uint32\_t [PLC\\_CorrPackToDevice\\_B2](#)
- Корректных пакетов по Ш2, к устройству
- uint32\_t [PLC\\_ErrPackToDevice\\_B2](#)
- Ошибок приема пакета по Ш2.
- uint32\_t [PLC\\_ErrPackFromDevice\\_B2](#)
- Ошибок отправки пакета по Ш2.
- [power\\_failure PLC\\_PowerDefect](#)
- Неиспр питания
- [bus\\_defect PLC\\_BusDefect\\_B1](#)
- Неиспр 1 шины
- [bus\\_defect PLC\\_BusDefect\\_B2](#)
- Неиспр 2 шины
- [self\\_diag PLC\\_SelfDiagDefect](#)
- Неиспр самодиагностики
- uint8\_t [Reserv\\_1](#) [68]
- РЕЗЕРВ
- [common\\_rom\\_registers PLC\\_CommonRomRegs](#)
- Общие регистры, которые хранятся в ПЗУ

### 3.6.1 Подробное описание

Организация пространства общих регистров во внешнем ОЗУ

### 3.6.2 Данные класса

#### 3.6.2.1 PLC\_BusDefect\_B1

[bus\\_defect](#) common\_register\_space\_ext\_ram::PLC\_BusDefect\_B1

Неиспр 1 шины

#### 3.6.2.2 PLC\_BusDefect\_B2

[bus\\_defect](#) common\_register\_space\_ext\_ram::PLC\_BusDefect\_B2

Неиспр 2 шины



### 3.6.2.3 PLC\_CM\_State

uint32\_t common\_register\_space\_ext\_ram::PLC\_CM\_State

Состояние автомата выбора УМ

### 3.6.2.4 PLC\_CommonRomRegs

common\_rom\_registers common\_register\_space\_ext\_ram::PLC\_CommonRomRegs

Общие регистры, которые хранятся в ПЗУ

### 3.6.2.5 PLC\_Config

device\_config common\_register\_space\_ext\_ram::PLC\_Config

Конфигурация устройства

### 3.6.2.6 PLC\_CorrPackFromDevice\_B1

uint32\_t common\_register\_space\_ext\_ram::PLC\_CorrPackFromDevice\_B1

Корректных пакетов по Ш1, от устройства

### 3.6.2.7 PLC\_CorrPackFromDevice\_B2

uint32\_t common\_register\_space\_ext\_ram::PLC\_CorrPackFromDevice\_B2

Корректных пакетов по Ш2, от устройства

### 3.6.2.8 PLC\_CorrPackToDevice\_B1

uint32\_t common\_register\_space\_ext\_ram::PLC\_CorrPackToDevice\_B1

Корректных пакетов по Ш1, к устройству

### 3.6.2.9 PLC\_CorrPackToDevice\_B2

uint32\_t common\_register\_space\_ext\_ram::PLC\_CorrPackToDevice\_B2

Корректных пакетов по Ш2, к устройству

### 3.6.2.10 PLC\_Durat

uint32\_t common\_register\_space\_ext\_ram::PLC\_Durat

Время с момента запуска, с

### 3.6.2.11 PLC\_ErrPackFromDevice\_B1

uint32\_t common\_register\_space\_ext\_ram::PLC\_ErrPackFromDevice\_B1

Ошибок отправки пакета по Ш1.

### 3.6.2.12 PLC\_ErrPackFromDevice\_B2

uint32\_t common\_register\_space\_ext\_ram::PLC\_ErrPackFromDevice\_B2

Ошибок отправки пакета по Ш2.

### 3.6.2.13 PLC\_ErrPackToDevice\_B1

uint32\_t common\_register\_space\_ext\_ram::PLC\_ErrPackToDevice\_B1

Ошибок приема пакета по Ш1.

### 3.6.2.14 PLC\_ErrPackToDevice\_B2

uint32\_t common\_register\_space\_ext\_ram::PLC\_ErrPackToDevice\_B2

Ошибок приема пакета по Ш2.

## 3.6.2.15 PLC\_PMAAddr

[device\\_address](#) common\_register\_space\_ext\_ram::PLC\_PMAAddr

Адрес устройства

## 3.6.2.16 PLC\_PowerDefect

[power\\_failure](#) common\_register\_space\_ext\_ram::PLC\_PowerDefect

Неиспр питания

## 3.6.2.17 PLC\_SelfDiagDefect

[self\\_diag](#) common\_register\_space\_ext\_ram::PLC\_SelfDiagDefect

Неиспр самодиагностики

## 3.6.2.18 PLC\_SoftVer

[plc\\_soft\\_ver](#) common\_register\_space\_ext\_ram::PLC\_SoftVer

Версия ПО

## 3.6.2.19 Reserv\_1

uint8\_t common\_register\_space\_ext\_ram::Reserv\_1[68]

РЕЗЕРВ

Объявления и описания членов структуры находятся в файле:

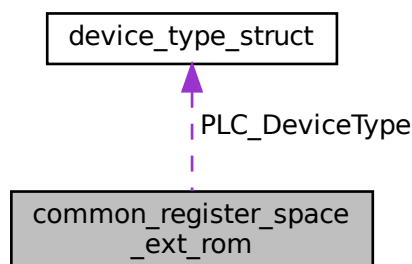
- [external\\_ram.h](#)

### 3.7 Структура common\_register\_space\_ext\_rom

Организация пространства общих регистров для зеркализации во внешнем ПЗУ

```
#include <external_rom.h>
```

Граф связей класса common\_register\_space\_ext\_rom:



Открытые атрибуты

- uint8\_t [PLC\\_DeviceInfo](#) [1024]  
Текстовая информация о модуле
- [device\\_type](#) [PLC\\_DeviceType](#)  
Тип устройства
- uint32\_t [PLC\\_SerialNumber](#)  
Серийный номер устройства
- uint32\_t [PLC\\_BusConfig\\_B1](#)  
Конфигурация шины 1.
- uint32\_t [PLC\\_BusConfig\\_B2](#)  
Конфигурация шины 2.
- uint32\_t [PLC\\_TimeoutForDefect\\_B1](#)  
Время без связи до неискр шины 1, мс
- uint32\_t [PLC\\_TimeoutForDefect\\_B2](#)  
Время без связи до неискр шины 2, мс
- uint16\_t [PLC\\_NumCrcErrorsForDefect\\_B1](#)  
Количество ошибок приема пакета до неисправности шины 1.
- uint16\_t [PLC\\_NumCrcErrorsForDefect\\_B2](#)  
Количество ошибок приема пакета до неисправности шины 2.
- uint16\_t [PLC\\_TimeToRepair](#)  
Максимально время переключения на резервный УМ, x10мс
- uint16\_t [PLC\\_TimeSoloWork](#)  
Время работы без резервирующего УМ, с
- uint16\_t [PLC\\_DualControl](#)  
Реакция при одновременном управлении
- uint8\_t [Reserv\\_2](#) [64]  
РЕЗЕРВ

### 3.7.1 Подробное описание

Организация пространства общих регистров для зеркализации во внешнем ПЗУ

### 3.7.2 Данные класса

#### 3.7.2.1 PLC\_BusConfig\_B1

```
uint32_t common_register_space_ext_rom::PLC_BusConfig_B1
```

Конфигурация шины 1.

#### 3.7.2.2 PLC\_BusConfig\_B2

```
uint32_t common_register_space_ext_rom::PLC_BusConfig_B2
```

Конфигурация шины 2.

#### 3.7.2.3 PLC\_DeviceInfo

```
uint8_t common_register_space_ext_rom::PLC_DeviceInfo[1024]
```

Текстовая информация о модуле

#### 3.7.2.4 PLC\_DeviceType

```
device_type common_register_space_ext_rom::PLC_DeviceType
```

Тип устройства

#### 3.7.2.5 PLC\_DualControl

```
uint16_t common_register_space_ext_rom::PLC_DualControl
```

Реакция при одновременном управлении

### 3.7.2.6 PLC\_NumCrcErrorsForDefect\_B1

uint16\_t common\_register\_space\_ext\_rom::PLC\_NumCrcErrorsForDefect\_B1

Количество ошибок приема пакета до неисправности шины 1.

### 3.7.2.7 PLC\_NumCrcErrorsForDefect\_B2

uint16\_t common\_register\_space\_ext\_rom::PLC\_NumCrcErrorsForDefect\_B2

Количество ошибок приема пакета до неисправности шины 2.

### 3.7.2.8 PLC\_SerialNumber

uint32\_t common\_register\_space\_ext\_rom::PLC\_SerialNumber

Серийный номер устройства

### 3.7.2.9 PLC\_TimeoutForDefect\_B1

uint32\_t common\_register\_space\_ext\_rom::PLC\_TimeoutForDefect\_B1

Время без связи до неспр шины 1, мс

### 3.7.2.10 PLC\_TimeoutForDefect\_B2

uint32\_t common\_register\_space\_ext\_rom::PLC\_TimeoutForDefect\_B2

Время без связи до неспр шины 2, мс

### 3.7.2.11 PLC\_TimeSoloWork

uint16\_t common\_register\_space\_ext\_rom::PLC\_TimeSoloWork

Время работы без резервирующего УМ, с

### 3.7.2.12 PLC\_TimeToRepair

uint16\_t common\_register\_space\_ext\_rom::PLC\_TimeToRepair

Максимально время переключения на резервный УМ, x10мс

### 3.7.2.13 Reserv\_2

uint8\_t common\_register\_space\_ext\_rom::Reserv\_2[64]

## РЕЗЕРВ

Объявления и описания членов структуры находятся в файле:

- [external\\_rom.h](#)

## 3.8 Структура config\_struct

Структура с битовыми полями для регистра конфигурация

```
#include <external_ram.h>
```

### Открытые атрибуты

- unsigned [main\\_switch](#): 4  
Основной свитч
- unsigned [add\\_switch\\_1](#): 4  
Доп свитч 1.
- unsigned [add\\_switch\\_2](#): 4  
Доп свитч 2.
- unsigned [reserv](#): 4  
Резерв

### 3.8.1 Подробное описание

Структура с битовыми полями для регистра конфигурация

### 3.8.2 Данные класса

### 3.8.2.1 add\_switch\_1

`unsigned config_struct::add_switch_1`

Доп свитч 1.

### 3.8.2.2 add\_switch\_2

`unsigned config_struct::add_switch_2`

Доп свитч 2.

### 3.8.2.3 main\_switch

`unsigned config_struct::main_switch`

Основной свитч

### 3.8.2.4 reserv

`unsigned config_struct::reserv`

Резерв

Объявления и описания членов структуры находятся в файле:

- [external\\_ram.h](#)

## 3.9 Структура device\_address\_struct

Структура с битовыми полями для регистра адрес устройства

```
#include <external_ram.h>
```

Открытые атрибуты

- unsigned `module_addr`: 4  
Адрес модуля
- unsigned `chassis_addr`: 4  
Адрес шасси
- unsigned `reserv`: 8  
Резерв



### 3.9.1 Подробное описание

Структура с битовыми полями для регистра адрес устройства

### 3.9.2 Данные класса

#### 3.9.2.1 chassis\_addr

unsigned device\_address\_struct::chassis\_addr

Адрес шасси

#### 3.9.2.2 module\_addr

unsigned device\_address\_struct::module\_addr

Адрес модуля

#### 3.9.2.3 reserv

unsigned device\_address\_struct::reserv

Резерв

Объявления и описания членов структуры находятся в файле:

- [external\\_ram.h](#)

## 3.10 Структура device\_type\_struct

Структура с битовыми полями для регистра тип устройства

```
#include <external_rom.h>
```

## Открытые атрибуты

- unsigned [revision](#): 4  
Ревизия модуля
- unsigned [modification](#): 4  
Модификация модуля
- unsigned [type](#): 9  
Тип модуля
- unsigned [batch](#): 5  
Партия
- unsigned [use\\_object](#): 5  
Объект применения
- unsigned [reserv](#): 5  
Резерв

### 3.10.1 Подробное описание

Структура с битовыми полями для регистра тип устройства

### 3.10.2 Данные класса

#### 3.10.2.1 batch

unsigned device\_type\_struct::batch

Партия

#### 3.10.2.2 modification

unsigned device\_type\_struct::modification

Модификация модуля

#### 3.10.2.3 reserv

unsigned device\_type\_struct::reserv

Резерв

#### 3.10.2.4 revision

unsigned device\_type\_struct::revision

Ревизия модуля

#### 3.10.2.5 type

unsigned device\_type\_struct::type

Тип модуля

#### 3.10.2.6 use\_object

unsigned device\_type\_struct::use\_object

Объект применения

Объявления и описания членов структуры находятся в файле:

- [external\\_rom.h](#)

### 3.11 Структура heap\_struct

Реализация "самодельной" кучи

```
#include <internal_ram.h>
```

Открытые атрибуты

- uint8\_t [memory\\_page\\_status](#) [[PAGE\\_NUM](#)]  
Статус каждой страница 0-свободна, 1-занята
- uint8\_t [memory\\_page\\_space](#) [[PAGE\\_NUM](#)][[PAGE\\_SIZE](#)]  
Место памяти для кучи разбитой на 32 страницы по 64 байта

#### 3.11.1 Подробное описание

Реализация "самодельной" кучи

#### 3.11.2 Данные класса

### 3.11.2.1 memory\_page\_space

```
uint8_t heap_struct::memory_page_space[PAGE_NUM][PAGE_SIZE]
```

Место памяти для кучи разбитой на 32 страницы по 64 байта

### 3.11.2.2 memory\_page\_status

```
uint8_t heap_struct::memory_page_status[PAGE_NUM]
```

Статус каждой страница 0-свободна, 1-занята

Объявления и описания членов структуры находятся в файле:

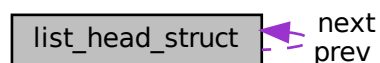
- [internal\\_ram.h](#)

## 3.12 Структура list\_head\_struct

Структура с описанием двусвязанного списка

```
#include <list.h>
```

Граф связей класса list\_head\_struct:



### Открытые атрибуты

- struct [list\\_head](#) \* [next](#)  
Указатель на следующий элемент списка
- struct [list\\_head](#) \* [prev](#)  
Указатель на предыдущий элемент списка

### 3.12.1 Подробное описание

Структура с описанием двусвязанного списка

### 3.12.2 Данные класса

#### 3.12.2.1 next

```
struct list\_head* list_head_struct::next
```

Указатель на следующий элемент списка

#### 3.12.2.2 prev

```
struct list\_head* list_head_struct::prev
```

Указатель на предыдущий элемент списка

Объявления и описания членов структуры находятся в файле:

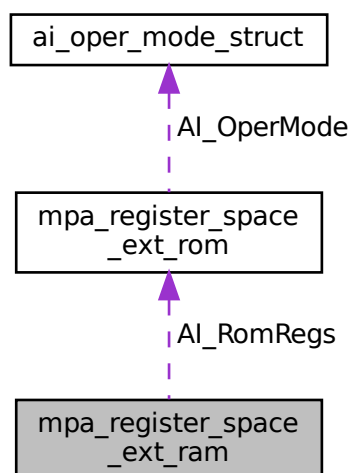
- [list.h](#)

## 3.13 Структура mpa\_register\_space\_ext\_ram

Организация пространства регистров МПА во внешнем ОЗУ

```
#include <external_ram.h>
```

Граф связей класса mpa\_register\_space\_ext\_ram:



## Открытые атрибуты

- [mpa\\_rom\\_registers AI\\_RomRegs](#)  
Регистры МПА, которые хранятся в ПЗУ
- [uint16\\_t AI\\_SignalChanged](#)  
Изменялся ли сигнал с последнего опроса
- [int16\\_t AI\\_CodeADC \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Сырые данные, код АЦП
- [float AI\\_PhysQuantFloat \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Физическая величина (число с плавающей точкой)
- [uint8\\_t AI\\_DiagnosticChannel \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Самодиагностика каналов
- [uint8\\_t Reserv\\_6 \[2\]](#)  
Резерв

### 3.13.1 Подробное описание

Организация пространства регистров МПА во внешнем ОЗУ

### 3.13.2 Данные класса

#### 3.13.2.1 AI\_CodeADC

```
int16_t mpa_register_space_ext_ram::AI_CodeADC[MAX_CHANNEL_NUMBER]
```

Сырые данные, код АЦП

#### 3.13.2.2 AI\_DiagnosticChannel

```
uint8_t mpa_register_space_ext_ram::AI_DiagnosticChannel[MAX_CHANNEL_NUMBER]
```

Самодиагностика каналов

#### 3.13.2.3 AI\_PhysQuantFloat

```
float mpa_register_space_ext_ram::AI_PhysQuantFloat[MAX_CHANNEL_NUMBER]
```

Физическая величина (число с плавающей точкой)

## 3.13.2.4 AI\_RomRegs

```
mpa_rom_registers mpa_register_space_ext_ram::AI_RomRegs
```

Регистры МПА, которые хранятся в ПЗУ

## 3.13.2.5 AI\_SignalChanged

```
uint16_t mpa_register_space_ext_ram::AI_SignalChanged
```

Изменялся ли сигнал с последнего опроса

## 3.13.2.6 Reserv\_6

```
uint8_t mpa_register_space_ext_ram::Reserv_6[2]
```

Резерв

Объявления и описания членов структуры находятся в файле:

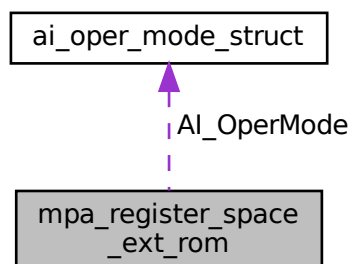
- [external\\_ram.h](#)

## 3.14 Структура mpa\_register\_space\_ext\_rom

Организация пространства регистров МПА для зеркализации во внешнем ПЗУ

```
#include <external_rom.h>
```

Граф связей класса mpa\_register\_space\_ext\_rom:



## Открытые атрибуты

- [ai\\_oper\\_mode AI\\_OperMode](#)  
Режим работы канала
- [uint16\\_t AI\\_NumForAverag \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Кол-во выборок для усреднения
- [int16\\_t AI\\_MinCodeADC \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Константа В - минимум кода АЦП
- [int16\\_t AI\\_MaxCodeADC \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Константа А - максимум кода АЦП
- [uint8\\_t Reserv\\_3 \[32\]](#)  
Резерв
- [float AI\\_PolynConst0 \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Константа A0 аппрокс. полинома
- [float AI\\_PolynConst1 \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Константа A1 аппрокс. полинома
- [float AI\\_PolynConst2 \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Константа A2 аппрокс. полинома
- [float AI\\_PolynConst3 \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Константа A3 аппрокс. полинома
- [float AI\\_PolynConst4 \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Константа A4 аппрокс. полинома
- [float AI\\_PolynConst5 \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Константа A5 аппрокс. полинома
- [float AI\\_PolynConst6 \[MAX\\_CHANNEL\\_NUMBER\]](#)  
Константа A6 аппрокс. полинома
- [uint8\\_t AI\\_MetrologDat \[32\]](#)  
Сведения о метрологии
- [uint8\\_t Reserv\\_4 \[32\]](#)  
Резерв
- [uint8\\_t Reserv\\_5 \[32\]](#)  
Резерв

### 3.14.1 Подробное описание

Организация пространства регистров МПА для зеркализации во внешнем ПЗУ

### 3.14.2 Данные класса

#### 3.14.2.1 AI\_MaxCodeADC

```
int16_t mpa_register_space_ext_rom::AI_MaxCodeADC[MAX_CHANNEL_NUMBER]
```

Константа А - максимум кода АЦП



### 3.14.2.2 AI\_MetrologDat

uint8\_t mpa\_register\_space\_ext\_rom::AI\_MetrologDat[32]

Сведения о метрологии

### 3.14.2.3 AI\_MinCodeADC

int16\_t mpa\_register\_space\_ext\_rom::AI\_MinCodeADC[MAX\_CHANEL\_NUMBER]

Константа В - минимум кода АЦП

### 3.14.2.4 AI\_NumForAverag

uint16\_t mpa\_register\_space\_ext\_rom::AI\_NumForAverag[MAX\_CHANEL\_NUMBER]

Кол-во выборок для усреднения

### 3.14.2.5 AI\_OperMode

ai\_oper\_mode mpa\_register\_space\_ext\_rom::AI\_OperMode

Режим работы канала

### 3.14.2.6 AI\_PolynConst0

float mpa\_register\_space\_ext\_rom::AI\_PolynConst0[MAX\_CHANEL\_NUMBER]

Константа A0 аппрокс. полинома

### 3.14.2.7 AI\_PolynConst1

float mpa\_register\_space\_ext\_rom::AI\_PolynConst1[MAX\_CHANEL\_NUMBER]

Константа A1 аппрокс. полинома

#### 3.14.2.8 AI\_PolynConst2

float mpa\_register\_space\_ext\_rom::AI\_PolynConst2[[MAX\\_CHANEL\\_NUMBER](#)]

Константа A2 аппрокс. полинома

#### 3.14.2.9 AI\_PolynConst3

float mpa\_register\_space\_ext\_rom::AI\_PolynConst3[[MAX\\_CHANEL\\_NUMBER](#)]

Константа A3 аппрокс. полинома

#### 3.14.2.10 AI\_PolynConst4

float mpa\_register\_space\_ext\_rom::AI\_PolynConst4[[MAX\\_CHANEL\\_NUMBER](#)]

Константа A4 аппрокс. полинома

#### 3.14.2.11 AI\_PolynConst5

float mpa\_register\_space\_ext\_rom::AI\_PolynConst5[[MAX\\_CHANEL\\_NUMBER](#)]

Константа A5 аппрокс. полинома

#### 3.14.2.12 AI\_PolynConst6

float mpa\_register\_space\_ext\_rom::AI\_PolynConst6[[MAX\\_CHANEL\\_NUMBER](#)]

Константа A6 аппрокс. полинома

#### 3.14.2.13 Reserv\_3

uint8\_t mpa\_register\_space\_ext\_rom::Reserv\_3[32]

Резерв

## 3.14.2.14 Reserv\_4

```
uint8_t mpa_register_space_ext_rom::Reserv_4[32]
```

Резерв

## 3.14.2.15 Reserv\_5

```
uint8_t mpa_register_space_ext_rom::Reserv_5[32]
```

Резерв

Объявления и описания членов структуры находятся в файле:

- [external\\_rom.h](#)

## 3.15 Структура packet\_header\_struct

Структура с полями заголовка пакета

```
#include <rs422_protocol.h>
```

Открытые атрибуты

- uint8\_t [header](#)  
Заголовок
- uint8\_t [receiver\\_addr](#)  
Адрес получателя
- uint8\_t [sender\\_addr](#)  
Адрес отправителя
- uint16\_t [packet\\_length](#)  
Длина пакета
- uint8\_t [service\\_byte](#)  
Сервисный байт
- uint8\_t [cmd\\_number](#)  
Количество команд

## 3.15.1 Подробное описание

Структура с полями заголовка пакета

## 3.15.2 Данные класса

#### 3.15.2.1 cmd\_number

`uint8_t packet_header_struct::cmd_number`

Количество команд

#### 3.15.2.2 header

`uint8_t packet_header_struct::header`

Заголовок

#### 3.15.2.3 packet\_length

`uint16_t packet_header_struct::packet_length`

Длина пакета

#### 3.15.2.4 receiver\_addr

`uint8_t packet_header_struct::receiver_addr`

Адрес получателя

#### 3.15.2.5 sender\_addr

`uint8_t packet_header_struct::sender_addr`

Адрес отправителя

#### 3.15.2.6 service\_byte

`uint8_t packet_header_struct::service_byte`

Сервисный байт

Объявления и описания членов структуры находятся в файле:

- [rs422\\_protocol.h](#)

## 3.16 Структура packet\_tail\_Struct

Структура с полями конца пакета

```
#include <rs422_protocol.h>
```

Открытые атрибуты

- uint32\_t [checksum](#)  
Контрольная сумма (CRC32)
- uint16\_t [end](#)  
Конец пакета

### 3.16.1 Подробное описание

Структура с полями конца пакета

### 3.16.2 Данные класса

#### 3.16.2.1 checksum

```
uint32_t packet_tail_Struct::checksum
```

Контрольная сумма (CRC32)

#### 3.16.2.2 end

```
uint16_t packet_tail_Struct::end
```

Конец пакета

Объявления и описания членов структуры находятся в файле:

- [rs422\\_protocol.h](#)

## 3.17 Структура plc\_sof\_ver\_struct

Структура с битовыми полями для регистра версия ПО

```
#include <external_ram.h>
```

## Открытые атрибуты

- unsigned `revision`: 4  
Ревизия модуля
- unsigned `modification`: 4  
Модификация модуля
- unsigned `type`: 9  
Тип модуля
- unsigned `soft_ver`: 10  
Версия ПО
- unsigned `add_info`: 4  
Дополнительная информация
- unsigned `develop`: 1  
1=ПО в процессе разработки

### 3.17.1 Подробное описание

Структура с битовыми полями для регистра версия ПО

### 3.17.2 Данные класса

#### 3.17.2.1 `add_info`

`unsigned plc_sof_ver_struct::add_info`

Дополнительная информация

#### 3.17.2.2 `develop`

`unsigned plc_sof_ver_struct::develop`

1=ПО в процессе разработки

#### 3.17.2.3 `modification`

`unsigned plc_sof_ver_struct::modification`

Модификация модуля

#### 3.17.2.4 revision

unsigned plc\_sof\_ver\_struct::revision

Ревизия модуля

#### 3.17.2.5 soft\_ver

unsigned plc\_sof\_ver\_struct::soft\_ver

Версия ПО

#### 3.17.2.6 type

unsigned plc\_sof\_ver\_struct::type

Тип модуля

Объявления и описания членов структуры находятся в файле:

- [external\\_ram.h](#)

### 3.18 Структура power\_failure\_struct

Структура с битовыми полями для регистра неисправность питания

```
#include <external_ram.h>
```

Открытые атрибуты

- unsigned [v2\\_3\\_3](#): 1  
2V3.3
- unsigned [v2\\_5](#): 1  
2V5
- unsigned [v1\\_3\\_3](#): 1  
1V3.3
- unsigned [v1\\_5](#): 1  
1V5
- unsigned [reserv](#): 12  
Резерв

#### 3.18.1 Подробное описание

Структура с битовыми полями для регистра неисправность питания

### 3.18.2 Данные класса

#### 3.18.2.1 reserv

`unsigned power_failure_struct::reserv`

Резерв

#### 3.18.2.2 v1\_3\_3

`unsigned power_failure_struct::v1_3_3`

1V3.3

#### 3.18.2.3 v1\_5

`unsigned power_failure_struct::v1_5`

1V5

#### 3.18.2.4 v2\_3\_3

`unsigned power_failure_struct::v2_3_3`

2V3.3

#### 3.18.2.5 v2\_5

`unsigned power_failure_struct::v2_5`

2V5

Объявления и описания членов структуры находятся в файле:

- [external\\_ram.h](#)

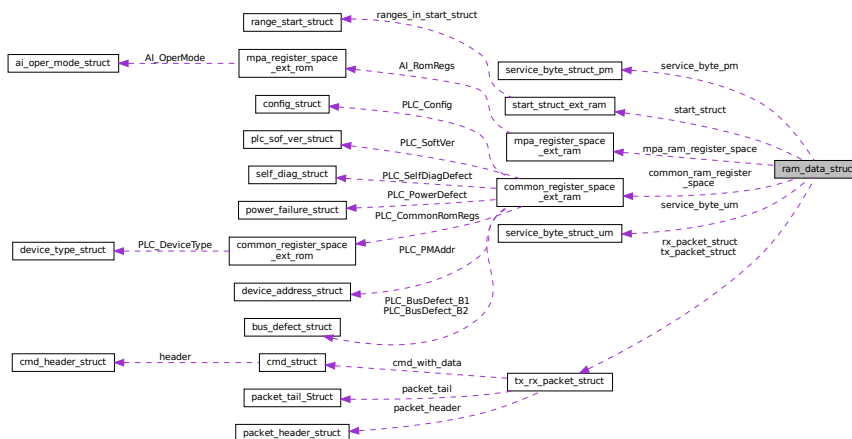


## 3.19 Структура ram\_data\_struct

Структура организующая память во внешнем ОЗУ

```
#include <external_ram.h>
```

Граф связей класса ram\_data\_struct:



### Открытые атрибуты

- [ram\\_start\\_struct start\\_struct](#)  
Структура по умолчанию, которая должна находиться в начале ОЗУ
- [uint8\\_t Reserv \[RAM\\_REGISTER\\_SPACE\\_START\\_ADDR - sizeof\(ram\\_start\\_struct\)\]](#)  
Зарезервированное место
- [common\\_ram\\_registers common\\_ram\\_register\\_space](#)  
Карта общих регистров начиная с адреса REGISTER\_SPACE\_START\_ADDR.
- [mpa\\_ram\\_registers mpa\\_ram\\_register\\_space](#)  
Карта регистров МПА начиная с адреса REGISTER\_SPACE\_START\_ADDR.
- [fields\\_packet rx\\_packet\\_struct](#)  
Принятый пакет с идентифицированными полями
- [fields\\_packet tx\\_packet\\_struct](#)  
Отправляемый пакет с идентифицированными полями
- [uint8\\_t tx\\_data \[UART\\_BUFFER\\_SIZE\]](#)  
Выделенное место для данных tx\_packet.
- [uint8\\_t packet\\_rx \[UART\\_BUFFER\\_SIZE\]](#)  
Буфер с принятым пакетом
- [uint8\\_t packet\\_tx \[UART\\_BUFFER\\_SIZE\]](#)  
Буфер с отправленным пакетом
- [service\\_struct\\_pm service\\_byte\\_pm](#)  
Структура сервисного байта ПМ
- [service\\_struct\\_um service\\_byte\\_um](#)  
Структура сервисного байта УМ
- [uint\\_least32\\_t crc\\_table \[256\]](#)  
Таблица для вычисления контрольной суммы
- [uint8\\_t uart1\\_rx\\_buffer \[UART\\_BUFFER\\_SIZE\]](#)

- Буфер приемника UART1.
- `uint8_t uart2_rx_buffer [UART_BUFFER_SIZE]`
- Буфер приемника UART2.
- `uint16_t spi_1_rx_buffer [SPI_BUFFER_SIZE]`
- Буфер приемника SPI1.

### 3.19.1 Подробное описание

Структура организующая память во внешнем ОЗУ

### 3.19.2 Данные класса

#### 3.19.2.1 `common_ram_register_space`

`common_ram_registers` `ram_data_struct::common_ram_register_space`

Карта общих регистров начиная с адреса `REGISTER_SPACE_START_ADDR`.

#### 3.19.2.2 `crc_table`

`uint_least32_t` `ram_data_struct::crc_table[256]`

Таблица для вычисления контрольной суммы

#### 3.19.2.3 `mpa_ram_register_space`

`mpa_ram_registers` `ram_data_struct::mpa_ram_register_space`

Карта регистров МПА начиная с адреса `REGISTER_SPACE_START_ADDR`.

#### 3.19.2.4 `packet_rx`

`uint8_t` `ram_data_struct::packet_rx[UART_BUFFER_SIZE]`

Буфер с принятым пакетом

## 3.19.2.5 packet\_tx

```
uint8_t ram_data_struct::packet_tx[UART_BUFFER_SIZE]
```

Буфер с отправленным пакетом

## 3.19.2.6 Reserv

```
uint8_t ram_data_struct::Reserv[RAM_REGISTER_SPACE_START_ADDR - sizeof(ram_start_struct)]
```

Зарезервированное место

## 3.19.2.7 rx\_packet\_struct

```
fields_packet ram_data_struct::rx_packet_struct
```

Принятый пакет с идентифицированными полями

## 3.19.2.8 service\_byte\_pm

```
service_struct_pm ram_data_struct::service_byte_pm
```

Структура сервисного байта ПМ

## 3.19.2.9 service\_byte\_um

```
service_struct_um ram_data_struct::service_byte_um
```

Структура сервисного байта УМ

## 3.19.2.10 spi\_1\_rx\_buffer

```
uint16_t ram_data_struct::spi_1_rx_buffer[SPI_BUFFER_SIZE]
```

Буфер приемника SPI1.

### 3.19.2.11 start\_struct

`ram_start_struct` `ram_data_struct::start_struct`

Структура по умолчанию, которая должна находиться в начале ОЗУ

### 3.19.2.12 tx\_data

`uint8_t` `ram_data_struct::tx_data[UART_BUFFER_SIZE]`

Выделенное место для данных `tx_packet`.

### 3.19.2.13 tx\_packet\_struct

`fields_packet` `ram_data_struct::tx_packet_struct`

Отправляемый пакет с идентифицированными полями

### 3.19.2.14 uart1\_rx\_buffer

`uint8_t` `ram_data_struct::uart1_rx_buffer[UART_BUFFER_SIZE]`

Буфер приемника UART1.

### 3.19.2.15 uart2\_rx\_buffer

`uint8_t` `ram_data_struct::uart2_rx_buffer[UART_BUFFER_SIZE]`

Буфер приемника UART2.

Объявления и описания членов структуры находятся в файле:

- `external_ram.h`

## 3.20 Структура range\_start\_struct

Структура одного диапазона для стартовой структуры

```
#include <external_ram.h>
```

## Открытые атрибуты

- uint16\_t [range\\_type](#)  
Тип диапазона (h) и тип операции (l)
- uint16\_t [start\\_channel\\_num](#)  
Стартовый номер канала (h) и номер диапазона (l)
- uint16\_t [address](#)  
Адрес
- uint16\_t [size](#)  
Количество байт

### 3.20.1 Подробное описание

Структура одного диапазона для стартовой структуры

### 3.20.2 Данные класса

#### 3.20.2.1 address

```
uint16_t range_start_struct::address
```

Адрес

#### 3.20.2.2 range\_type

```
uint16_t range_start_struct::range_type
```

Тип диапазона (h) и тип операции (l)

#### 3.20.2.3 size

```
uint16_t range_start_struct::size
```

Количество байт

### 3.20.2.4 start\_channel\_num

```
uint16_t range_start_struct::start_channel_num
```

Стартовый номер канала (h) и номер диапазона (l)

Объявления и описания членов структуры находятся в файле:

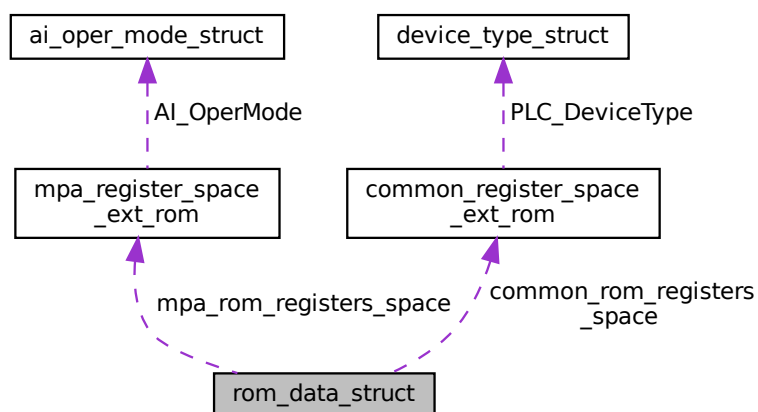
- [external\\_ram.h](#)

## 3.21 Структура rom\_data\_struct

Структура организующая память во внешнем ПЗУ

```
#include <external_rom.h>
```

Граф связей класса rom\_data\_struct:



Открытые атрибуты

- [common\\_rom\\_registers](#) [common\\_rom\\_registers\\_space](#)  
Общие регистры
- [mpa\\_rom\\_registers](#) [mpa\\_rom\\_registers\\_space](#)  
Регистры МПА

### 3.21.1 Подробное описание

Структура организующая память во внешнем ПЗУ

### 3.21.2 Данные класса

#### 3.21.2.1 common\_rom\_registers\_space

`common_rom_registers` rom\_data\_struct::common\_rom\_registers\_space

Общие регистры

#### 3.21.2.2 mpa\_rom\_registers\_space

`mpa_rom_registers` rom\_data\_struct::mpa\_rom\_registers\_space

Регистры МПА

Объявления и описания членов структуры находятся в файле:

- `external_rom.h`

## 3.22 Структура self\_diag\_struct

Структура с битовыми полями для регистра неисправность самодиагностики

```
#include <external_ram.h>
```

Открытые атрибуты

- unsigned `fail_crc_firmware`: 1  
Некорректная ЦРЦ прошивки
- unsigned `power_fail`: 1  
Неисправность питания
- unsigned `fail_download_rom`: 1  
Ошибка при загрузке данных из ПЗУ
- unsigned `fail_soft_ver`: 1  
Версия ПО и тип модуля не совпадают
- unsigned `fail_firmware_ram`: 1  
Неисправность в ПО арбитра ОЗУ
- unsigned `fail_firmware_2_bus`: 1  
Неисправность в ПО 2ой шины
- unsigned `fail_firmware_1_bus`: 1  
Неисправность в ПО 1ой шины
- unsigned `reserv`: 1  
Резерв
- uint8\_t `fail_chanel` [8]  
Неисправность в каналах
- unsigned `reserv1`: 8  
Резерв

### 3.22.1 Подробное описание

Структура с битовыми полями для регистра неисправность самодиагностики

### 3.22.2 Данные класса

#### 3.22.2.1 fail\_chanel

```
uint8_t self_diag_struct::fail_chanel[8]
```

Неисправность в каналах

#### 3.22.2.2 fail\_crc\_firmware

```
unsigned self_diag_struct::fail_crc_firmware
```

Некорректная ЦРЦ прошивки

#### 3.22.2.3 fail\_download\_rom

```
unsigned self_diag_struct::fail_download_rom
```

Ошибка при загрузке данных из ПЗУ

#### 3.22.2.4 fail\_firmware\_1\_bus

```
unsigned self_diag_struct::fail_firmware_1_bus
```

Неисправность в ПО 1ой шины

#### 3.22.2.5 fail\_firmware\_2\_bus

```
unsigned self_diag_struct::fail_firmware_2_bus
```

Неисправность в ПО 2ой шины



## 3.22.2.6 fail\_firmware\_ram

unsigned self\_diag\_struct::fail\_firmware\_ram

Неисправность в ПО арбитра ОЗУ

## 3.22.2.7 fail\_soft\_ver

unsigned self\_diag\_struct::fail\_soft\_ver

Версия ПО и тип модуля не совпадают

## 3.22.2.8 power\_fail

unsigned self\_diag\_struct::power\_fail

Неисправность питания

## 3.22.2.9 reserv

unsigned self\_diag\_struct::reserv

Резерв

## 3.22.2.10 reserv1

unsigned self\_diag\_struct::reserv1

Резерв

Объявления и описания членов структуры находятся в файле:

- [external\\_ram.h](#)

## 3.23 Структура service\_byte\_struct\_pm

Структура с битовыми полями для сервисного байта ПМ

```
#include <external_ram.h>
```

## Открытые атрибуты

- unsigned `init`: 1  
Инициализация (0 – не инициализирован, 1 – инициализирован)
- unsigned `self_diagnostics_error`: 1  
Неисправность самодиагностики
- unsigned `reserv_1`: 1  
Резерв
- unsigned `reserv_2`: 1  
Резерв
- unsigned `fail_bus_1`: 1  
Неисправность (1 шина)
- unsigned `fail_bus_2`: 1  
Неисправность (2 шина)
- unsigned `both_control`: 1  
Признак одновременного управления
- unsigned `master`: 1  
Ведущий мастер (0 – первый, 1 – второй)

### 3.23.1 Подробное описание

Структура с битовыми полями для сервисного байта ПМ

### 3.23.2 Данные класса

#### 3.23.2.1 `both_control`

`unsigned service_byte_struct_pm::both_control`

Признак одновременного управления

#### 3.23.2.2 `fail_bus_1`

`unsigned service_byte_struct_pm::fail_bus_1`

Неисправность (1 шина)

#### 3.23.2.3 `fail_bus_2`

`unsigned service_byte_struct_pm::fail_bus_2`

Неисправность (2 шина)

#### 3.23.2.4 init

unsigned service\_byte\_struct\_pm::init

Инициализация (0 – не инициализирован, 1 – инициализирован)

#### 3.23.2.5 master

unsigned service\_byte\_struct\_pm::master

Ведущий мастер (0 – первый, 1 – второй)

#### 3.23.2.6 reserv\_1

unsigned service\_byte\_struct\_pm::reserv\_1

Резерв

#### 3.23.2.7 reserv\_2

unsigned service\_byte\_struct\_pm::reserv\_2

Резерв

#### 3.23.2.8 self\_diagnostics\_error

unsigned service\_byte\_struct\_pm::self\_diagnostics\_error

Неисправность самодиагностики

Объявления и описания членов структуры находятся в файле:

- [external\\_ram.h](#)

### 3.24 Структура service\_byte\_struct\_um

Структура с битовыми полями для сервисного байта УМ

```
#include <external_ram.h>
```

## Открытые атрибуты

- unsigned [last\\_answer](#): 1  
1= не получен ответ на предыдущий пакет
- unsigned [reserv](#): 6  
Резерв
- unsigned [ready\\_to\\_control](#): 1  
Флаг «Готовность выполнять управление»

### 3.24.1 Подробное описание

Структура с битовыми полями для сервисного байта УМ

### 3.24.2 Данные класса

#### 3.24.2.1 last\_answer

unsigned service\_byte\_struct\_um::last\_answer

1= не получен ответ на предыдущий пакет

#### 3.24.2.2 ready\_to\_control

unsigned service\_byte\_struct\_um::ready\_to\_control

Флаг «Готовность выполнять управление»

#### 3.24.2.3 reserv

unsigned service\_byte\_struct\_um::reserv

Резерв

Объявления и описания членов структуры находятся в файле:

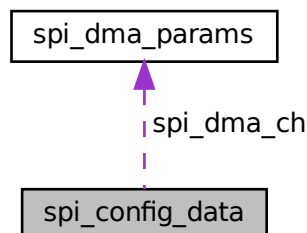
- [external\\_ram.h](#)

## 3.25 Структура spi\_config\_data

Структура с конфигурационными параметрами SPI.

```
#include <spi.h>
```

Граф связей класса spi\_config\_data:



### Открытые атрибуты

- MDR\_SSP\_TypeDef \* SSPx  
Библиотечная структура с периферийными регистрами блока SPI.
- spi\_n\_dma\_ch\_params spi\_dma\_ch  
Структура с параметрами канала DMA для SPI.
- uint32\_t RST\_CLK\_PCLK\_SPIin  
Включение тактирования для SPIin.
- SSP\_InitTypeDef SPI  
Библиотечная структура с конфигурационными параметрами SPI.
- IRQn\_Type IRQn  
Выбор обработчика прерываний блока SPI.
- uint8\_t buffer\_counter  
Счетчик слов в приемнике SPI.
- uint16\_t \* buffer  
Указатель на буфер приемника SPI.

### 3.25.1 Подробное описание

Структура с конфигурационными параметрами SPI.

### 3.25.2 Данные класса

### 3.25.2.1 buffer

`uint16_t* spi_config_data::buffer`

Указатель на буфер приемника SPI.

### 3.25.2.2 buffer\_counter

`uint8_t spi_config_data::buffer_counter`

Счетчик слов в приемнике SPI.

### 3.25.2.3 IRQn

`IRQn_Type spi_config_data::IRQn`

Выбор обработчика прерываний блока SPI.

### 3.25.2.4 RST\_CLK\_PCLK\_SPIn

`uint32_t spi_config_data::RST_CLK_PCLK_SPIn`

Включение тактирования для SPIn.

### 3.25.2.5 SPI

`SSP_InitTypeDef spi_config_data::SPI`

Библиотечная структура с конфигурационными параметрами SPI.

### 3.25.2.6 spi\_dma\_ch

`spi_n_dma_ch_params spi_config_data::spi_dma_ch`

Структура с параметрами канала DMA для SPI.

### 3.25.2.7 SSPx

MDR\_SSP\_TypeDef\* spi\_config\_data::SSPx

Библиотечная структура с периферийными регистрами блока SPI.

Объявления и описания членов структуры находятся в файле:

- [spi.h](#)

## 3.26 Структура spi\_dma\_params

Структура с параметрами DMA канала SPIn.

```
#include <spi.h>
```

### Открытые атрибуты

- uint8\_t [dma\\_channel](#)  
Выбор канала DMA для SPIn.
- uint32\_t [dma\\_irq\\_counter](#)  
Счетчик прерываний DMA.
- DMA\_CtrlDataInitTypeDef [DMA\\_InitStructure\\_SPI\\_RX](#)  
Структура с настройками DMA в целом
- DMA\_ChannelInitTypeDef [DMA\\_Channel\\_SPI\\_RX](#)  
Структура с настройками канала DMA.

### 3.26.1 Подробное описание

Структура с параметрами DMA канала SPIn.

### 3.26.2 Данные класса

#### 3.26.2.1 dma\_channel

uint8\_t spi\_dma\_params::dma\_channel

Выбор канала DMA для SPIn.

### 3.26.2.2 DMA\_Channel\_SPI\_RX

`DMA_ChannelInitTypeDef spi_dma_params::DMA_Channel_SPI_RX`

Структура с настройками канала DMA.

### 3.26.2.3 DMA\_InitStructure\_SPI\_RX

`DMA_CtrlDataInitTypeDef spi_dma_params::DMA_InitStructure_SPI_RX`

Структура с настройками DMA в целом

### 3.26.2.4 dma\_irq\_counter

`uint32_t spi_dma_params::dma_irq_counter`

Счетчик прерываний DMA.

Объявления и описания членов структуры находятся в файле:

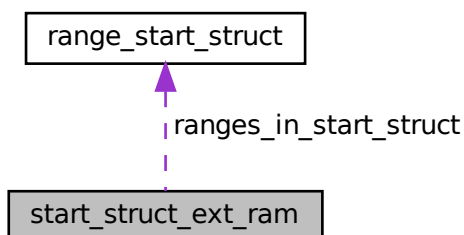
- [spi.h](#)

## 3.27 Структура start\_struct\_ext\_ram

Структура, которая лежит в начале ОЗУ любого модуля

```
#include <external_ram.h>
```

Граф связей класса `start_struct_ext_ram`:





## Открытые атрибуты

- `uint16_t length`  
Длина структуры
- `uint16_t text_info`  
Ссылка на текстовую информацию о модуле
- `uint16_t flag_change_struct`  
Флаг изменения структуры
- `uint16_t number_of_ranges`  
Количество диапазонов
- `range ranges_in_start_struct [START_STRUCT_NUMBER_OF_RANGES]`  
Диапазоны в стартовой структуре

### 3.27.1 Подробное описание

Структура, которая лежит в начале ОЗУ любого модуля

### 3.27.2 Данные класса

#### 3.27.2.1 `flag_change_struct`

`uint16_t start_struct_ext_ram::flag_change_struct`

Флаг изменения структуры

#### 3.27.2.2 `length`

`uint16_t start_struct_ext_ram::length`

Длина структуры

#### 3.27.2.3 `number_of_ranges`

`uint16_t start_struct_ext_ram::number_of_ranges`

Количество диапазонов

#### 3.27.2.4 ranges\_in\_start\_struct

`range` start\_struct\_ext\_ram::ranges\_in\_start\_struct[START\_STRUCT\_NUMBER\_OF\_RANGES]

Диапазоны в стартовой структуре

#### 3.27.2.5 text\_info

`uint16_t` start\_struct\_ext\_ram::text\_info

Ссылка на текстовую информацию о модуле

Объявления и описания членов структуры находятся в файле:

- [external\\_ram.h](#)

### 3.28 Структура timer\_config\_struct

Структура с конфигурационными параметрами Таймеров

`#include <timers.h>`

Открытые атрибуты

- `MDR_TIMER_TypeDef * TIMERx`  
Библиотечная структура с периферийными регистрами блока TIMER.
- `TIMER_CntInitTypeDef TIMERInitStruct`  
Библиотечная структура с конфигурационными параметрами блока TIMER.
- `TIMER_ChnInitTypeDef sTIM\_ChnInit`  
Библиотечная структура с конфигурационными параметрами каналов блока TIMER (используются например для режима захвата)
- `TIMER_Status_Flags_TypeDef TIMER\_STATUS`  
Настрока событий, по которому происходит прерывание блока TIMER.
- `uint32_t timer\_cnt`  
Счетчик для TIMER (может быть использован для разных целей)

#### 3.28.1 Подробное описание

Структура с конфигурационными параметрами Таймеров

#### 3.28.2 Данные класса

### 3.28.2.1 sTIM\_ChnInit

TIMER\_ChnInitTypeDef timer\_config\_struct::sTIM\_ChnInit

Библиотечная структура с конфигурационными параметрами каналов блока TIMER (используются например для режима захвата)

### 3.28.2.2 timer\_cnt

uint32\_t timer\_config\_struct::timer\_cnt

Счетчик для TIMER (может быть использован для разных целей)

### 3.28.2.3 TIMER\_STATUS

TIMER\_Status\_Flags\_TypeDef timer\_config\_struct::TIMER\_STATUS

Настройка событий, по которому происходит прерывание блока TIMER.

### 3.28.2.4 TIMERInitStruct

TIMER\_CntInitTypeDef timer\_config\_struct::TIMERInitStruct

Библиотечная структура с конфигурационными параметрами блока TIMER.

### 3.28.2.5 TIMEx

MDR\_TIMER\_TypeDef\* timer\_config\_struct::TIMEx

Библиотечная структура с периферийными регистрами блока TIMER.

Объявления и описания членов структуры находятся в файле:

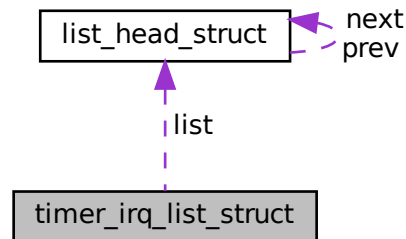
- [timers.h](#)

### 3.29 Структура timer\_irq\_list\_struct

Структура-реализация односвязанного списка обработчиков прерываний таймеров

```
#include <timers.h>
```

Граф связей класса timer\_irq\_list\_struct:



#### Открытые атрибуты

- void(\* [handler](#) )(void \*)  
Указатель на функцию-обработчик прерывания
- TIMER\_Status\_Flags\_TypeDef [event](#)  
Событие, по которому срабатывает прерывание
- void \* [data](#)  
Указатель на данные, передаваемые в обработчик (при необходимости)
- [list\\_head list](#)  
Список прерываний

#### 3.29.1 Подробное описание

Структура-реализация односвязанного списка обработчиков прерываний таймеров

#### 3.29.2 Данные класса

##### 3.29.2.1 data

```
void* timer_irq_list_struct::data
```

Указатель на данные, передаваемые в обработчик (при необходимости)

## 3.29.2.2 event

TIMER\_Status\_Flags\_TypeDef timer\_irq\_list\_struct::event

Событие, по которому срабатывает прерывание

## 3.29.2.3 handler

void(\* timer\_irq\_list\_struct::handler) (void \*)

Указатель на функцию-обработчик прерывания

## 3.29.2.4 list

list\_head timer\_irq\_list\_struct::list

Список прерываний

Объявления и описания членов структуры находятся в файле:

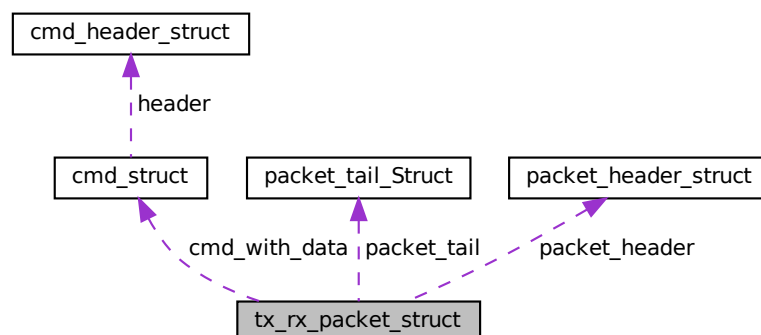
- [timers.h](#)

## 3.30 Структура tx\_rx\_packet\_struct

Структура пакета данных согласно утвержденному протоколу обмена данными

```
#include <rs422_protocol.h>
```

Граф связей класса tx\_rx\_packet\_struct:



## Открытые атрибуты

- [fields\\_packet\\_header packet\\_header](#)  
Заголовочные поля
- [fields\\_cmd cmd\\_with\\_data \[NUMBER\\_CMDS\\_IN\\_PACKET\]](#)  
Массив полей (субпакетов) с содержимым каждой команды
- [fields\\_packet\\_tail packet\\_tail](#)  
Поля хвоста пакета

### 3.30.1 Подробное описание

Структура пакета данных согласно утвержденному протоколу обмена данными

### 3.30.2 Данные класса

#### 3.30.2.1 cmd\_with\_data

[fields\\_cmd tx\\_rx\\_packet\\_struct::cmd\\_with\\_data\[NUMBER\\_CMDS\\_IN\\_PACKET\]](#)

Массив полей (субпакетов) с содержимым каждой команды

#### 3.30.2.2 packet\_header

[fields\\_packet\\_header tx\\_rx\\_packet\\_struct::packet\\_header](#)

Заголовочные поля

#### 3.30.2.3 packet\_tail

[fields\\_packet\\_tail tx\\_rx\\_packet\\_struct::packet\\_tail](#)

Поля хвоста пакета

Объявления и описания членов структуры находятся в файле:

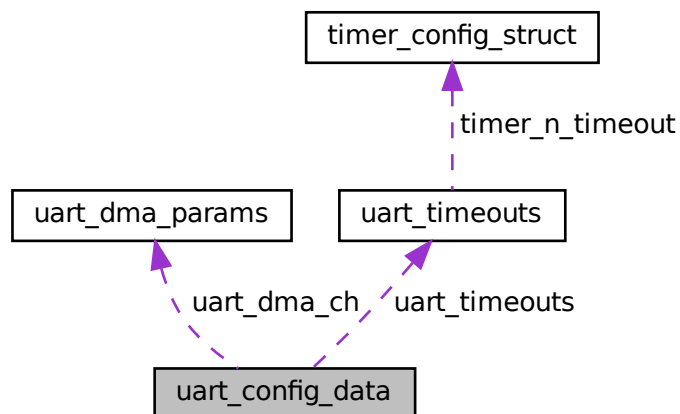
- [rs422\\_protocol.h](#)

### 3.31 Структура `uart_config_data`

Структура с конфигурационными параметрами UART и буфером приема

```
#include <uart.h>
```

Граф связей класса `uart_config_data`:



#### Открытые атрибуты

- `MDR_UART_TypeDef * UARTx`  
Библиотечная структура с периферийными регистрами блока UART.
- `uart_rx_tx_timeouts uart_timeouts`  
Структура с таймаутами UARTn.
- `uart_dma_ch_params uart_dma_ch`  
Структура с параметрами канала DMA для UART.
- `IRQn_Type IRQn`  
Выбор обработчика прерываний UARTn.
- `uint32_t RST_CLK_PCLK_UARTn`  
Включение тактирования для UARTn.
- `UART_InitTypeDef UART`  
Библиотечная структура с конфигурационными параметрами UART.
- `uint32_t UART_HCLKdiv`  
Выбор делителя тактовой частоты для тактирования блока UARTn
- `uint8_t * buffer`  
Указатель на буфер приемника UART.
- `uint32_t buffer_count`  
Счетчик элементов буфера
- `uint32_t read_pos`  
Текущая позиция курсора чтения в буфере

### 3.31.1 Подробное описание

Структура с конфигурационными параметрами UART и буфером приема

### 3.31.2 Данные класса

#### 3.31.2.1 buffer

```
uint8_t* uart_config_data::buffer
```

Указатель на буфер приемника UART.

#### 3.31.2.2 buffer\_count

```
uint32_t uart_config_data::buffer_count
```

Счетчик элементов буфера

#### 3.31.2.3 IRQn

```
IRQn_Type uart_config_data::IRQn
```

Выбор обработчика прерываний UARTn.

#### 3.31.2.4 read\_pos

```
uint32_t uart_config_data::read_pos
```

Текущая позиция курсора чтения в буфере

#### 3.31.2.5 RST\_CLK\_PCLK\_UARTn

```
uint32_t uart_config_data::RST_CLK_PCLK_UARTn
```

Включение тактирования для UARTn.



### 3.31.2.6 UART

`UART_InitTypeDef` `uart_config_data::UART`

Библиотечная структура с конфигурационными параметрами UART.

### 3.31.2.7 `uart_dma_ch`

`uart_dma_ch_params` `uart_config_data::uart_dma_ch`

Структура с параметрами канала DMA для UART.

### 3.31.2.8 `UART_HCLKdiv`

`uint32_t` `uart_config_data::UART_HCLKdiv`

Выбор делителя тактовой частоты для тактирования блока UARTn

### 3.31.2.9 `uart_timeouts`

`uart_rx_tx_timeouts` `uart_config_data::uart_timeouts`

Структура с таймаутами UARTn.

### 3.31.2.10 `UARTx`

`MDR_UART_TypeDef*` `uart_config_data::UARTx`

Библиотечная структура с периферийными регистрами блока UART.

Объявления и описания членов структуры находятся в файле:

- [uart.h](#)

## 3.32 Структура `uart_dma_params`

Структура с параметрами DMA канала UARTn.

```
#include <uart.h>
```

## Открытые атрибуты

- `uint8_t dma_channel`  
Выбор канала DMA для UARTn.
- `uint32_t dma_irq_counter`  
Счетчик прерываний DMA.
- `DMA_CtrlDataInitTypeDef DMA_InitStructure_UART_RX`  
Структура с настройками DMA в целом
- `DMA_ChannelInitTypeDef DMA_Channel_UART_RX`  
Структура с настройками канала DMA.

### 3.32.1 Подробное описание

Структура с параметрами DMA канала UARTn.

### 3.32.2 Данные класса

#### 3.32.2.1 dma\_channel

```
uint8_t uart_dma_params::dma_channel
```

Выбор канала DMA для UARTn.

#### 3.32.2.2 DMA\_Channel\_UART\_RX

```
DMA_ChannelInitTypeDef uart_dma_params::DMA_Channel_UART_RX
```

Структура с настройками канала DMA.

#### 3.32.2.3 DMA\_InitStructure\_UART\_RX

```
DMA_CtrlDataInitTypeDef uart_dma_params::DMA_InitStructure_UART_RX
```

Структура с настройками DMA в целом

3.32.2.4 `dma_irq_counter`

```
uint32_t uart_dma_params::dma_irq_counter
```

Счетчик прерываний DMA.

Объявления и описания членов структуры находятся в файле:

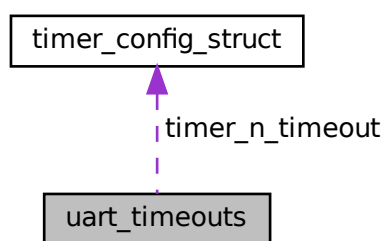
- [uart.h](#)

3.33 Структура `uart_timeouts`

Структура с таймаутами UARTn.

```
#include <uart.h>
```

Граф связей класса `uart_timeouts`:



## Открытые атрибуты

- `timer_n * timer_n_timeout`  
Выбор таймера для отслеживания таймаутов
- `uint8_t read_timeout_flag`  
Флаг таймаута на чтение
- `uint8_t write_timeout_flag`  
Флаг таймаута на запись
- `uint32_t read_val_timeout`  
Таймаут на чтение
- `uint32_t write_val_timeout`  
Таймаут на запись

## 3.33.1 Подробное описание

Структура с таймаутами UARTn.

### 3.33.2 Данные класса

#### 3.33.2.1 read\_timeout\_flag

`uint8_t uart_timeouts::read_timeout_flag`

Флаг таймаута на чтение

#### 3.33.2.2 read\_val\_timeout

`uint32_t uart_timeouts::read_val_timeout`

Таймаут на чтение

#### 3.33.2.3 timer\_n\_timeout

`timer_n* uart_timeouts::timer_n_timeout`

Выбор таймера для отслеживания таймаутов

#### 3.33.2.4 write\_timeout\_flag

`uint8_t uart_timeouts::write_timeout_flag`

Флаг таймаута на запись

#### 3.33.2.5 write\_val\_timeout

`uint32_t uart_timeouts::write_val_timeout`

Таймаут на запись

Объявления и описания членов структуры находятся в файле:

- [uart.h](#)

## Глава 4

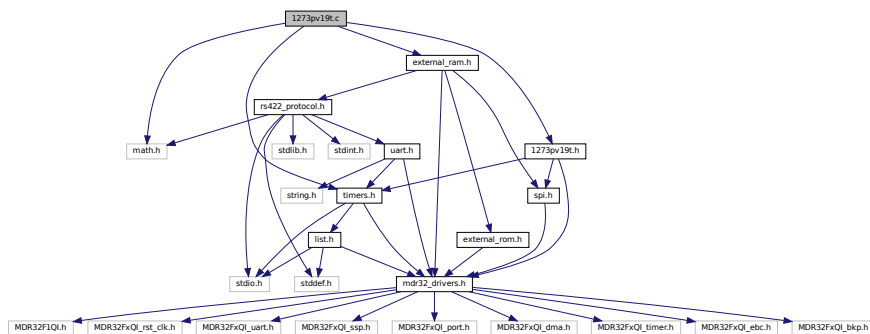
# Файлы

### 4.1 Файл 1273pv19t.c

Файл с реализацией API для работы с АЦП 1273ПВ19Т

```
#include "math.h"  
#include "1273pv19t.h"  
#include "timers.h"  
#include "external_ram.h"
```

Граф включаемых заголовочных файлов для 1273pv19t.c:



## Функции

- void `adc_gpio_config` (void)  
Конфигурирует выводы МК для АЦП
- void `adc_init` (adc\_n \*adc\_struct)  
Инициализирует микросхему АЦП 1273ПВ19Т
- void `adc_reset` (void)  
Аппаратный сброс микросхемы АЦП 1273ПВ19Т

## Переменные

- `spi_n spi_1`  
Структура с конфигурационными параметрами блока SPI1 МК
- `timer_n timer_2`  
Структура с конфигурационными параметрами блока Timer2 МК
- `adc_n adc_1`  
Структура с конфигурационными параметрами микросхемы АЦП

### 4.1.1 Подробное описание

Файл с реализацией API для работы с АЦП 1273ПВ19Т

### 4.1.2 Функции

#### 4.1.2.1 `adc_gpio_config()`

```
void adc_gpio_config (
    void )
```

Конфигурирует выводы МК для АЦП

```
36 {
37     // Включение тактирования портов
38     RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK|RST_CLK_PCLK_PORTD|RST_CLK_PCLK_PORTE|RST_CLK_PCLK_PORTF) = ENABLE;
39
40     PORT_InitTypeDef GPIO_init_structADC;
41     PORT_StructInit(&GPIO_init_structADC);
42
43     // Инициализация RESET PD0 для аппаратного сброса ацп
44     GPIO_init_structADC.PORT_Pin = PIN_ADC_RST;
45     GPIO_init_structADC.PORT_FUNC = PORT_FUNC_PORT;
46     GPIO_init_structADC.PORT_OE = PORT_OE_OUT;
47     GPIO_init_structADC.PORT_MODE = PORT_MODE_DIGITAL;
48     GPIO_init_structADC.PORT_SPEED = PORT_SPEED_MAXFAST;
49     PORT_Init(PIN_ADC_RST, &GPIO_init_structADC);
50     // Установка RESET в лог единицу
51     PORT_SetBits(PIN_ADC_RST, PIN_ADC_RST);
52
53     // Инициализация ножек выбора режима работы (0-10В, 4-20мА)
54     GPIO_init_structADC.PORT_Pin = PIN_ADC_MODE_A0;
55     GPIO_init_structADC.PORT_FUNC = PORT_FUNC_PORT;
56     GPIO_init_structADC.PORT_OE = PORT_OE_OUT;
57     PORT_Init(PIN_ADC_MODE, &GPIO_init_structADC);
58
59     GPIO_init_structADC.PORT_Pin = PIN_ADC_MODE_A1;
60     GPIO_init_structADC.PORT_FUNC = PORT_FUNC_PORT;
61     GPIO_init_structADC.PORT_OE = PORT_OE_OUT;
62     PORT_Init(PIN_ADC_MODE, &GPIO_init_structADC);
63
64     // Выбор однополярного режима на мультиплексоре A0=1; A1=0
65     PORT_SetBits(PIN_ADC_MODE, PIN_ADC_MODE_A0);
66     PORT_ResetBits(PIN_ADC_MODE, PIN_ADC_MODE_A1);
67
68     // Инициализация вывода NSS
69     GPIO_init_structADC.PORT_Pin = PIN_ADC_NSS;
70     GPIO_init_structADC.PORT_FUNC = PORT_FUNC_PORT;
71     GPIO_init_structADC.PORT_OE = PORT_OE_OUT;
72     PORT_Init(PIN_ADC_NSS, &GPIO_init_structADC);
73
74     // Установка NSS в лог ноль
75     PORT_ResetBits(PIN_ADC_NSS, PIN_ADC_NSS);
76 }
```

```

77 // Инициализация порта E для канала 2 таймера 2 на вход прерывания SDIFS/SDOFS от АЦП (канал захвата для
78 таймера)
79 GPIO_init_structADC.PORT_FUNC = PORT_FUNC_MAIN;
80 GPIO_init_structADC.PORT_OE = PORT_OE_IN;
81 GPIO_init_structADC.PORT_SPEED = PORT_SPEED_MAXFAST;
82 GPIO_init_structADC.PORT_MODE = PORT_MODE_DIGITAL;
83 GPIO_init_structADC.PORT_Pin = PIN_ADC_SDIFS_IRQ;
84 GPIO_init_structADC.PORT_PULL_DOWN = PORT_PULL_DOWN_ON;
85 PORT_Init (PORT_ADC_SDIFS_IRQ, &GPIO_init_structADC);
86 }

```

#### 4.1.2.2 adc\_init()

```

void adc_init (
    adc_n * adc_struct )

```

Инициализирует микросхему АЦП 1273ПВ19Т

Аргументы

*adc_struct	- Структура с конфиг. параметрами АЦП
-------------	---------------------------------------

```

90 {
91     adc_gpio_config();
92
93     // Активация микросхемы АЦП
94     PORT_SetBits(PORT_ADC_NSS, PIN_ADC_NSS);
95     delay_milli(1);
96
97     adc_reset();
98     delay_milli(1);
99
100     switch (CHANNEL_NUMBER)
101     {
102     case 1:
103         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
104         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8308);
105         delay_milli(1);
106         break;
107     case 2:
108         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
109         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
110         delay_milli(1);
111         break;
112     case 3:
113         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
114         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
115         delay_milli(1);
116         // Режим control, запись в регистр E управление питанием АЦП3 и АЦП4
117         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8408);
118         delay_milli(1);
119         break;
120     case 4:
121         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
122         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
123         delay_milli(1);
124         // Режим control, запись в регистр E управление питанием АЦП3 и АЦП4
125         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8488);
126         delay_milli(1);
127         break;
128     case 5:
129         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
130         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
131         delay_milli(1);
132         // Режим control, запись в регистр E управление питанием АЦП3 и АЦП4
133         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8488);
134         delay_milli(1);
135         // Режим control, запись в регистр F управление питанием АЦП5 и АЦП6
136         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8508);
137         delay_milli(1);
138         break;
139     case 6:
140         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
141         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
142         delay_milli(1);

```

```

143          // Режим control, запись в регистр E управление питанием АЦП3 и АЦП4
144          SSP_SendData(adc_struct->spi_struct->SSPx, 0x8488);
145          delay_milli(1);
146          // Режим control, запись в регистр F управление питанием АЦП5 и АЦП6
147          SSP_SendData(adc_struct->spi_struct->SSPx, 0x8588);
148          delay_milli(1);
149          break;
150      }
151  }
152  // Режим control, запись в регистр C вкл. 5В режима,использ. вывода REFOUT, вкл. опорное напряж.
153  SSP_SendData(adc_struct->spi_struct->SSPx, 0x82E0);
154  delay_milli(1);
155
156  // Режим control, запись в регистр A - перевод в режим данных
157  SSP_SendData(adc_struct->spi_struct->SSPx, 0x8001);
158  delay_micro(10);
159
160  // Очистка буфера FIFO SPI передатчика
161  spi_clean_fifo_rx_buf(adc_struct->spi_struct);
162
163  adc_struct->init_flag = 1;
164 }

```

#### 4.1.2.3 adc\_reset()

```

void adc_reset (
    void )

```

Аппаратный сброс микросхемы АЦП 1273ПВ19Т

```

169 {
170     PORT_ResetBits(PORT_ADC_RST,PIN_ADC_RST);
171     delay_milli(100);
172     PORT_SetBits(PORT_ADC_RST,PIN_ADC_RST);
173 }

```

### 4.1.3 Переменные

#### 4.1.3.1 adc\_1

adc\_n adc\_1

Структура с конфигурационными параметрами микросхемы АЦП

#### 4.1.3.2 spi\_1

spi\_n spi\_1 [extern]

Структура с конфигурационными параметрами блока SPI1 МК



## 4.1.3.3 timer\_2

`timer_n` timer\_2 [extern]

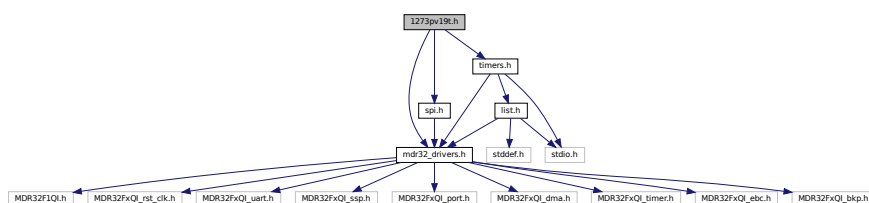
Структура с конфигурационными параметрами блока Timer2 МК

## 4.2 Файл 1273pv19t.h

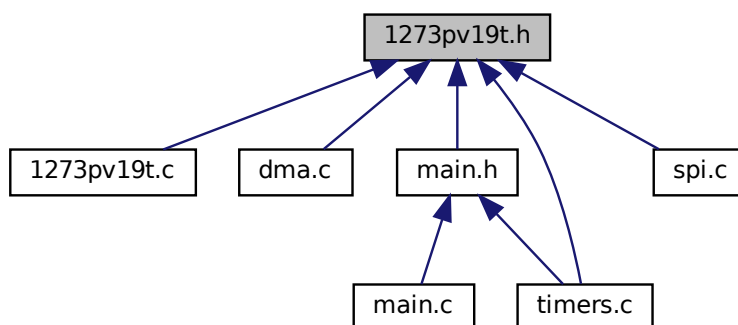
Заголовочный файл с описанием API для работы с микросхемой АЦП 1273ПВ19Т

```
#include "mdr32_drivers.h"
#include "spi.h"
#include "timers.h"
```

Граф включаемых заголовочных файлов для 1273pv19t.h:



Граф файлов, в которые включается этот файл:



## Классы

- struct `adc_config_data`

Структура с конфигурационными параметрами АЦП

## Макросы

- `#define PORT_ADC_RST MDR_PORTD`  
Порт линии аппаратного сброса микросхемы АЦП
- `#define PIN_ADC_RST PORT_Pin_0`  
Вывод линии аппаратного сброса микросхемы АЦП
- `#define PORT_ADC_MODE MDR_PORTE`  
Порт линии выбора режима работы АЦП
- `#define PIN_ADC_MODE_A0 PORT_Pin_11`  
Вывод линии выбора режима работы АЦП A0.
- `#define PIN_ADC_MODE_A1 PORT_Pin_15`  
Вывод линии выбора режима работы АЦП A1.
- `#define PORT_ADC_NSS MDR_PORTD`  
Порт линии активации микросхемы АЦП
- `#define PIN_ADC_NSS PORT_Pin_1`  
Вывод линии активации микросхемы АЦП
- `#define PORT_ADC_SDIFS_IRQ MDR_PORTE`  
Порт линии входа прерывания SDIFS/SDOFS от АЦП (канал захвата для таймера)
- `#define PIN_ADC_SDIFS_IRQ PORT_Pin_10`  
Вывод линии входа прерывания SDIFS/SDOFS от АЦП (канал захвата для таймера)

## Определения типов

- `typedef struct adc_config_data adc_n`  
Структура с конфигурационными параметрами АЦП

## Функции

- `void adc_init (adc_n *adc_struct)`  
Инициализирует микросхему АЦП 1273ПВ19Т
- `void adc_reset (void)`  
Аппаратный сброс микросхемы АЦП 1273ПВ19Т

### 4.2.1 Подробное описание

Заголовочный файл с описанием API для работы с микросхемой АЦП 1273ПВ19Т

### 4.2.2 Макросы

#### 4.2.2.1 PIN\_ADC\_MODE\_A0

```
#define PIN_ADC_MODE_A0 PORT_Pin_11
```

Вывод линии выбора режима работы АЦП A0.

#### 4.2.2.2 PIN\_ADC\_MODE\_A1

```
#define PIN_ADC_MODE_A1 PORT_Pin_15
```

Вывод линии выбора режима работы АЦП А1.

#### 4.2.2.3 PIN\_ADC\_NSS

```
#define PIN_ADC_NSS PORT_Pin_1
```

Вывод линии активации микросхемы АЦП

#### 4.2.2.4 PIN\_ADC\_RST

```
#define PIN_ADC_RST PORT_Pin_0
```

Вывод линии аппаратного сброса микросхемы АЦП

#### 4.2.2.5 PIN\_ADC\_SDIFS\_IRQ

```
#define PIN_ADC_SDIFS_IRQ PORT_Pin_10
```

Вывод линии входа прерывания SDIFS/SDOFS от АЦП (канал захвата для таймера)

#### 4.2.2.6 PORT\_ADC\_MODE

```
#define PORT_ADC_MODE MDR_PORTE
```

Порт линии выбора режима работы АЦП

#### 4.2.2.7 PORT\_ADC\_NSS

```
#define PORT_ADC_NSS MDR_PORTD
```

Порт линии активации микросхемы АЦП

#### 4.2.2.8 PORT\_ADC\_RST

```
#define PORT_ADC_RST MDR_PORTD
```

Порт линии аппаратного сброса микросхемы АЦП

#### 4.2.2.9 PORT\_ADC\_SDIFS\_IRQ

```
#define PORT_ADC_SDIFS_IRQ MDR_PORTE
```

Порт линии входа прерывания SDIFS/SDOFS от АЦП (канал захвата для таймера)

### 4.2.3 Типы

#### 4.2.3.1 adc\_n

```
typedef struct adc_config_data adc_n
```

Структура с конфигурационными параметрами АЦП

### 4.2.4 Функции

#### 4.2.4.1 adc\_init()

```
void adc_init (
    adc_n * adc_struct )
```

Инициализирует микросхему АЦП 1273ПВ19Т

Аргументы

*adc_struct	- Структура с конфиг. параметрами АЦП
-------------	---------------------------------------

```
90 {
91     adc_gpio_config();
92
93     // Активация микросхемы АЦП
94     PORT_SetBits(PORT_ADC_NSS, PIN_ADC_NSS);
95     delay_milli(1);
96
97     adc_reset();
98     delay_milli(1);
99
100     switch (CHANEL_NUMBER)
101     {
102         case 1:
```

```

103         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
104         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8308);
105         delay_milli(1);
106         break;
107     case 2:
108         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
109         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
110         delay_milli(1);
111         break;
112     case 3:
113         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
114         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
115         delay_milli(1);
116         // Режим control, запись в регистр E управление питанием АЦП3 и АЦП4
117         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8408);
118         delay_milli(1);
119         break;
120     case 4:
121         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
122         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
123         delay_milli(1);
124         // Режим control, запись в регистр E управление питанием АЦП3 и АЦП4
125         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8488);
126         delay_milli(1);
127         break;
128     case 5:
129         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
130         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
131         delay_milli(1);
132         // Режим control, запись в регистр E управление питанием АЦП3 и АЦП4
133         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8488);
134         delay_milli(1);
135         // Режим control, запись в регистр F управление питанием АЦП5 и АЦП6
136         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8508);
137         delay_milli(1);
138         break;
139     case 6:
140         // Режим control, запись в регистр D управление питанием АЦП1 и АЦП2
141         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8388);
142         delay_milli(1);
143         // Режим control, запись в регистр E управление питанием АЦП3 и АЦП4
144         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8488);
145         delay_milli(1);
146         // Режим control, запись в регистр F управление питанием АЦП5 и АЦП6
147         SSP_SendData(adc_struct->spi_struct->SSPx, 0x8588);
148         delay_milli(1);
149         break;
150 }
151 // Режим control, запись в регистр C вкл. 5В режима,использ. вывода REFOUT, вкл. опорное напряж.
152 SSP_SendData(adc_struct->spi_struct->SSPx, 0x82E0);
153 delay_milli(1);
154 // Режим control, запись в регистр A - перевод в режим данных
155 SSP_SendData(adc_struct->spi_struct->SSPx, 0x8001);
156 delay_micro(10);
157 // Очистка буфера FIFO SPI передатчика
158 spi_clean_fifo_rx_buf(adc_struct->spi_struct);
159
160 adc_struct->init_flag = 1;
161 }

```

#### 4.2.4.2 adc\_reset()

```

void adc_reset (
    void )

```

Аппаратный сброс микросхемы АЦП 1273ПВ19Т

```

169 {
170     PORT_ResetBits(PORT_ADC_RST,PIN_ADC_RST);
171     delay_milli(100);
172     PORT_SetBits(PORT_ADC_RST,PIN_ADC_RST);
173 }

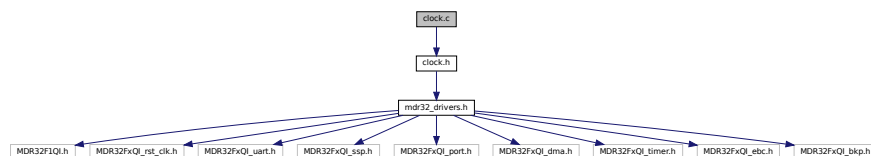
```

## 4.3 Файл clock.c

Файл с реализацией API для настройки тактирования МК

```
#include "clock.h"
```

Граф включаемых заголовочных файлов для clock.c:



### Функции

- void `clock_init` (void)  
Настраивает тактирование МК

### 4.3.1 Подробное описание

Файл с реализацией API для настройки тактирования МК

### 4.3.2 Функции

#### 4.3.2.1 clock\_init()

```
void clock_init (
    void )
```

Настраивает тактирование МК

```

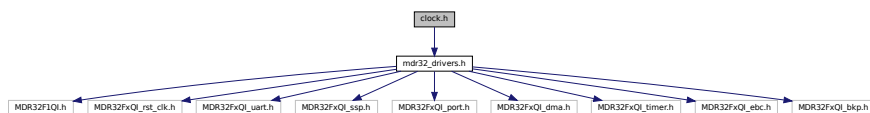
12 {
13     // Сброс настроек системы тактирования
14     RST_CLK_DeInit();
15
16     // Настройка тактирования от внешнего кварца HSE_OSC МГц
17     // Включаем генератор на внешнем кварце
18     RST_CLK_HSEconfig (RST_CLK_HSE_ON);
19     while (RST_CLK_HSEstatus () != SUCCESS);
20
21     // Настраиваем источник и коэффициент умножения PLL
22     //(CPU_C1_SEL = HSE / 1 * 12 = 144 МГц )
23     RST_CLK_CPU_PLLconfig ( RST_CLK_CPU_PLSrcHSEdiv1,11);
24
25     // Включаем PLL, но еще не подключаем к кристаллу (PLL умножает частоту тактирования)
26     RST_CLK_CPU_PLLcmd (ENABLE);
27     while (RST_CLK_CPU_PLLstatus () != SUCCESS);
28
29     // Делитель C3 ( CPU_C3_SEL = CPU_C2_SEL )
30     RST_CLK_CPUclkPrescaler (RST_CLK_CPUclkDIV1);
31
32     // На C2 идет с PLL, а не напрямую с C1 (CPU_C2_SEL = PLL)
33     RST_CLK_CPU_PLUse (ENABLE);
34     // CPU берет тактирование с выхода C3
35     //( HCLK_SEL = CPU_C3_SEL = 128 МГц)
36     RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
37
38     SystemCoreClockUpdate();
39 }
```

## 4.4 Файл clock.h

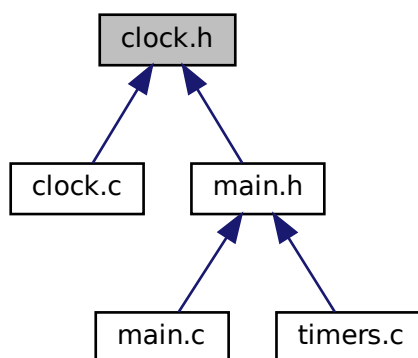
Заголовочный файл с описанием API для настройки тактирования МК

```
#include "mdr32_drivers.h"
```

Граф включаемых заголовочных файлов для clock.h:



Граф файлов, в которые включается этот файл:



### Функции

- void `clock_init` (void)  
Настраивает тактирование МК

#### 4.4.1 Подробное описание

Заголовочный файл с описанием API для настройки тактирования МК

#### 4.4.2 Функции

#### 4.4.2.1 clock\_init()

```
void clock_init (
    void )
```

Настраивает тактирование МК

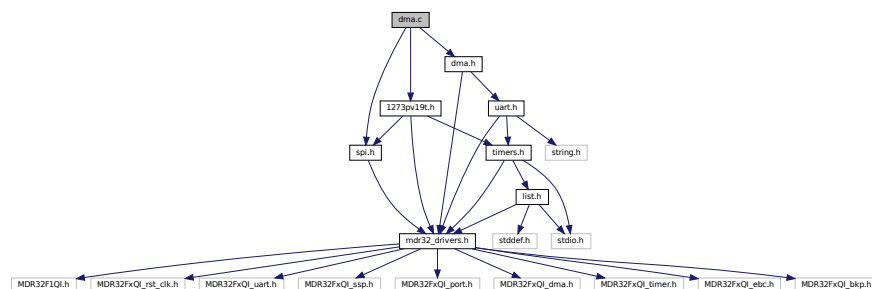
```
12 {
13     // Сброс настроек системы тактирования
14     RST_CLK_DeInit();
15
16     // Настройка тактирования от внешнего кварца HSE_OSC МГц
17     // Включаем генератор на внешнем кварце
18     RST_CLK_HSEconfig (RST_CLK_HSE_ON);
19     while (RST_CLK_HSEstatus () != SUCCESS);
20
21     // Настраиваем источник и коэффициент умножения PLL
22     //(CPU_C1_SEL = HSE / 1 * 12 = 144 МГц)
23     RST_CLK_CPU_PLLconfig ( RST_CLK_CPU_PLLsrcHSEdiv1,11);
24
25     // Включаем PLL, но еще не подключаем к кристаллу (PLL умножает частоту тактирования)
26     RST_CLK_CPU_PLLcmd (ENABLE);
27     while (RST_CLK_CPU_PLLstatus () != SUCCESS);
28
29     // Делитель C3 ( CPU_C3_SEL = CPU_C2_SEL )
30     RST_CLK_CPUclkPrescaler (RST_CLK_CPUclkDIV1);
31
32     // На C2 идет с PLL, а не напрямую с C1 (CPU_C2_SEL = PLL)
33     RST_CLK_CPU_PLLuse (ENABLE);
34     // CPU берет тактирование с выхода C3
35     //( HCLK_SEL = CPU_C3_SEL = 128 МГц)
36     RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
37
38     SystemCoreClockUpdate();
39 }
```

## 4.5 Файл dma.c

Файл с реализацией API для работы с DMA.

```
#include "dma.h"
#include "spi.h"
#include "1273pv19t.h"
```

Граф включаемых заголовочных файлов для dma.c:



### Функции

- void [dma\\_common\\_init](#) (void)  
Выполняет общую инициализацию DMA.
- void [DMA\\_IRQHandler](#) (void)  
Обработчик прерываний DMA.



## Переменные

- [spi\\_n spi\\_1](#)  
Структура с конфигурационными параметрами блока SPI1 МК
- [spi\\_n spi\\_2](#)  
Структура с конфигурационными параметрами блока SPI2 МК
- [adc\\_n adc\\_1](#)  
Структура с конфигурационными параметрами микросхемы АЦП
- [timer\\_n timer\\_1](#)  
Структура с конфигурационными параметрами блока Timer1 МК
- [timer\\_n timer\\_2](#)  
Структура с конфигурационными параметрами блока Timer2 МК
- [timer\\_n timer\\_3](#)  
Структура с конфигурационными параметрами блока Timer3 МК
- [uart\\_n uart\\_1](#)  
Структура с конфигурационными параметрами блока UART1 МК
- [uart\\_n uart\\_2](#)  
Структура с конфигурационными параметрами блока UART2 МК

### 4.5.1 Подробное описание

Файл с реализацией API для работы с DMA.

### 4.5.2 Функции

#### 4.5.2.1 dma\_common\_init()

```
void dma_common_init (
    void )
```

Выполняет общую инициализацию DMA.

```
44 {
45     // Разрешить тактирование DMA
46     RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK | RST_CLK_PCLK_DMA | RST_CLK_PCLK_SSP1
47                     | RST_CLK_PCLK_SSP2 | RST_CLK_PCLK_SSP3, ENABLE);
48     #ifdef K1986VE3T
49     RST_CLK_PCLKcmd(RST_CLK_PCLK_SSP4, ENABLE);
50     #endif
51     // Запретить все прерывания, в том числе от SSP
52     NVIC->ICPR[0] = 0xFFFFFFFF;
53     NVIC->ICER[0] = 0xFFFFFFFF;
54     MDR_SSP1->DMACR=0;
55     MDR_SSP2->DMACR=0;
56     MDR_SSP3->DMACR=0;
57     #ifdef K1986VE3T
58     MDR_SSP4->DMACR=0;
59     #endif
60     // Сбросить все настройки DMA
61     DMA_DeInit();
62 }
```

## 4.5.2.2 DMA\_IRQHandler()

```
void DMA_IRQHandler (
    void )
```

Обработчик прерываний DMA.

```
68 {
69     // Если сработало прерывание при заполнении буфера приемника UART1
70     if(DMA_GetFlagStatus(DMA_Channel_REQ_UART1_RX, DMA_FLAG_CHNL_ENA) == 0)
71     {
72         // Если скопирован последний буфер UART
73         if (uart_1.uart_dma_ch.dma_irq_counter == ((UART_BUFFER_SIZE/1024) - 1))
74         {
75             DMA_UART_RX_init(&uart_1);
76             uart_1.uart_dma_ch.dma_irq_counter = 0;
77             uart_1.buffer_count = 0;
78         }
79         else
80         {
81             uart_1.uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestBaseAddr +=
uart_1.uart_dma_ch.DMA_InitStructure_UART_RX.DMA_CycleSize;
82             // Инициализировать канал снова
83             DMA_Init(uart_1.uart_dma_ch.dma_channel, &uart_1.uart_dma_ch.DMA_Channel_UART_RX);
84             uart_1.uart_dma_ch.dma_irq_counter++;
85         }
86     }
87     // Если сработало прерывание при заполнении буфера приемника UART2
88     if(DMA_GetFlagStatus(DMA_Channel_REQ_UART2_RX, DMA_FLAG_CHNL_ENA) == 0)
89     {
90         // Если скопирован последний буфер UART
91         if (uart_2.uart_dma_ch.dma_irq_counter == ((UART_BUFFER_SIZE/1024) - 1))
92         {
93             DMA_UART_RX_init(&uart_2);
94             uart_2.uart_dma_ch.dma_irq_counter = 0;
95         }
96         else
97         {
98             uart_2.uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestBaseAddr +=
uart_2.uart_dma_ch.DMA_InitStructure_UART_RX.DMA_CycleSize;
99             // Инициализировать канал снова
100             DMA_Init(uart_2.uart_dma_ch.dma_channel, &uart_2.uart_dma_ch.DMA_Channel_UART_RX);
101             uart_2.uart_dma_ch.dma_irq_counter++;
102         }
103     }
104     #ifdef K1986VE3T
105     // Если сработало прерывание при заполнении буфера приемника UART3
106     if(DMA_GetFlagStatus(DMA_Channel_SW12, DMA_FLAG_CHNL_ENA) == RESET)
107     {
108         // Если скопирован последний буфер UART
109         if (uart_3.uart_dma_ch.dma_irq_counter == ((UART_BUFFER_SIZE/1024) - 1))
110         {
111             DMA_UART_RX_init(&uart_3);
112             uart_3.uart_dma_ch.dma_irq_counter = 0;
113             uart_3.buffer_count = 0;
114         }
115         else
116         {
117             uart_3.uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestBaseAddr +=
uart_3.uart_dma_ch.DMA_InitStructure_UART_RX.DMA_CycleSize;
118             // Инициализировать канал снова
119             DMA_Init(uart_3.uart_dma_ch.dma_channel, &uart_3.uart_dma_ch.DMA_Channel_UART_RX);
120             uart_3.uart_dma_ch.dma_irq_counter++;
121         }
122     }
123     // Если сработало прерывание при заполнении буфера приемника UART4
124     if(DMA_GetFlagStatus(DMA_Channel_SW14, DMA_FLAG_CHNL_ENA) == RESET)
125     {
126         // Если скопирован последний буфер UART
127         if (uart_4.uart_dma_ch.dma_irq_counter == ((UART_BUFFER_SIZE/1024) - 1))
128         {
129             DMA_UART_RX_init(&uart_4);
130             uart_4.uart_dma_ch.dma_irq_counter = 0;
131             uart_4.buffer_count = 0;
132         }
133         else
134         {
135             uart_4.uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestBaseAddr +=
uart_4.uart_dma_ch.DMA_InitStructure_UART_RX.DMA_CycleSize;
136             // Инициализировать канал снова
137             DMA_Init(uart_4.uart_dma_ch.dma_channel, &uart_4.uart_dma_ch.DMA_Channel_UART_RX);
138             uart_4.uart_dma_ch.dma_irq_counter++;
139         }
140     }
141     #endif
142 }
```

### 4.5.3 Переменные

#### 4.5.3.1 adc\_1

`adc_n` adc\_1 [extern]

Структура с конфигурационными параметрами микросхемы АЦП

#### 4.5.3.2 spi\_1

`spi_n` spi\_1 [extern]

Структура с конфигурационными параметрами блока SPI1 МК

#### 4.5.3.3 spi\_2

`spi_n` spi\_2 [extern]

Структура с конфигурационными параметрами блока SPI2 МК

#### 4.5.3.4 timer\_1

`timer_n` timer\_1 [extern]

Структура с конфигурационными параметрами блока Timer1 МК

#### 4.5.3.5 timer\_2

`timer_n` timer\_2 [extern]

Структура с конфигурационными параметрами блока Timer2 МК

#### 4.5.3.6 timer\_3

`timer_n` timer\_3 [extern]

Структура с конфигурационными параметрами блока Timer3 МК

#### 4.5.3.7 uart\_1

`uart_n` `uart_1` [extern]

Структура с конфигурационными параметрами блока UART1 МК

#### 4.5.3.8 uart\_2

`uart_n` `uart_2` [extern]

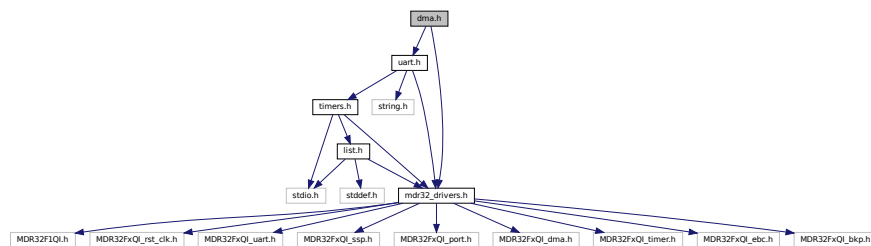
Структура с конфигурационными параметрами блока UART2 МК

### 4.6 Файл dma.h

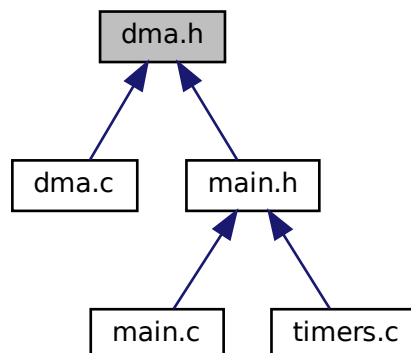
Заголовочный файл с описанием API для работы с DMA.

```
#include "uart.h"
#include "mdr32_drivers.h"
```

Граф включаемых заголовочных файлов для dma.h:



Граф файлов, в которые включается этот файл:



## Функции

- void `dma_common_init` (void)  
Выполняет общую инициализацию DMA.

### 4.6.1 Подробное описание

Заголовочный файл с описанием API для работы с DMA.

### 4.6.2 Функции

#### 4.6.2.1 `dma_common_init()`

```
void dma_common_init (
    void )
```

Выполняет общую инициализацию DMA.

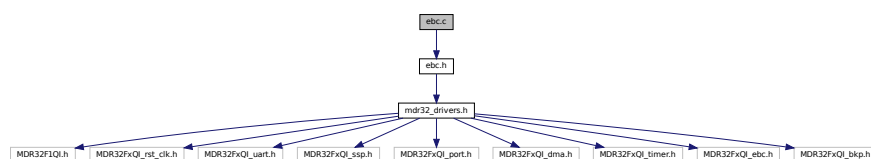
```
44 {
45     // Разрешить тактирование DMA
46     RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK | RST_CLK_PCLK_DMA | RST_CLK_PCLK_SSP1
47         | RST_CLK_PCLK_SSP2 | RST_CLK_PCLK_SSP3, ENABLE);
48     #ifdef K1986VE3T
49     RST_CLK_PCLKcmd(RST_CLK_PCLK_SSP4, ENABLE);
50     #endif
51     // Запретить все прерывания, в том числе от SSP
52     NVIC->ICPR[0] = 0xFFFFFFFF;
53     NVIC->ICER[0] = 0xFFFFFFFF;
54     MDR_SSP1->DMACR=0;
55     MDR_SSP2->DMACR=0;
56     MDR_SSP3->DMACR=0;
57     #ifdef K1986VE3T
58     MDR_SSP4->DMACR=0;
59     #endif
60     // Сбросить все настройки DMA
61     DMA_DeInit();
62 }
```

## 4.7 Файл ebc.c

Файл с реализацией API для работы с ЕВС.

```
#include "ebc.h"
```

Граф включаемых заголовочных файлов для ebc.c:



## Функции

- void `ebc_gpio_config` (`ebc_devices` device)  
Конфигурирует выводы МК ЕВС для внешнего ОЗУ или ПЗУ
- void `ebc_init` (`ebc_devices` device)  
Инициализирует ЕВС для внешнего ОЗУ или ПЗУ

### 4.7.1 Подробное описание

Файл с реализацией API для работы с ЕВС.

### 4.7.2 Функции

#### 4.7.2.1 `ebc_gpio_config()`

```
void ebc_gpio_config (
    ebc_devices device )
```

Конфигурирует выводы МК ЕВС для внешнего ОЗУ или ПЗУ

```
12 {
13     // Структура для инициализации линий ввода-вывода системной шины
14     PORT_InitTypeDef ExtBusInitStruct;
15
16     // Включение тактирования портов
17     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTA, ENABLE);
18     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTB, ENABLE);
19     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
20     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
21     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTE, ENABLE);
22     RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTF, ENABLE);
23
24     // Конфигурация линии данных
25     if (device == EBC_RAM)
26     {
27         // 32-битный режим данных
28         ExtBusInitStruct.PORT_Pin = (PORT_Pin_0 | PORT_Pin_1 | PORT_Pin_2 | PORT_Pin_3 | PORT_Pin_4 |
PORT_Pin_5 | PORT_Pin_6 | PORT_Pin_7 | PORT_Pin_8 | PORT_Pin_9 | PORT_Pin_10 | PORT_Pin_11 |
PORT_Pin_12 | PORT_Pin_13 | PORT_Pin_14 | PORT_Pin_15);
29     }
30     else if (device == EBC_ROM)
31     {
32         // 8-битный режим данных
33         ExtBusInitStruct.PORT_Pin = (PORT_Pin_0 | PORT_Pin_1 | PORT_Pin_2 | PORT_Pin_3 | PORT_Pin_4 |
PORT_Pin_5 | PORT_Pin_6 | PORT_Pin_7);
34     }
35     ExtBusInitStruct.PORT_FUNC = PORT_FUNC_MAIN;
36     ExtBusInitStruct.PORT_MODE = PORT_MODE_DIGITAL;
37     ExtBusInitStruct.PORT_SPEED = PORT_SPEED_FAST;
38
39     PORT_Init(MDR_PORTA, &ExtBusInitStruct);
40
41     ExtBusInitStruct.PORT_Pin = (PORT_Pin_0 | PORT_Pin_1 | PORT_Pin_2 | PORT_Pin_3 | PORT_Pin_4 |
PORT_Pin_5 | PORT_Pin_6 | PORT_Pin_7 | PORT_Pin_8 | PORT_Pin_9 | PORT_Pin_10 | PORT_Pin_11 |
PORT_Pin_12 | PORT_Pin_13 | PORT_Pin_14 | PORT_Pin_15);
42     if (device == EBC_RAM)
43     {
44         // 32-битный режим данных
45         ExtBusInitStruct.PORT_FUNC = PORT_FUNC_MAIN;
46     }
47     else if (device == EBC_ROM)
48     {
49         // 8-битный режим данных
50         ExtBusInitStruct.PORT_FUNC = PORT_FUNC_PORT;
51     }
52     ExtBusInitStruct.PORT_MODE = PORT_MODE_DIGITAL;
53     ExtBusInitStruct.PORT_SPEED = PORT_SPEED_FAST;
54 }
```

```

55     PORT_Init(MDR_PORTB, &ExtBusInitStruct);
56
57     // Конфигурация OE, WE
58     ExtBusInitStruct.PORT_Pin = (PORT_Pin_0 | PORT_Pin_1);
59     ExtBusInitStruct.PORT_FUNC = PORT_FUNC_MAIN;
60     ExtBusInitStruct.PORT_MODE = PORT_MODE_DIGITAL;
61     ExtBusInitStruct.PORT_SPEED = PORT_SPEED_FAST;
62
63     PORT_Init(MDR_PORTC, &ExtBusInitStruct);
64
65     // Конфигурация BE3, BE2, BE1, BE0
66     ExtBusInitStruct.PORT_Pin = (PORT_Pin_9 | PORT_Pin_10 | PORT_Pin_11 | PORT_Pin_12);
67     ExtBusInitStruct.PORT_FUNC = PORT_FUNC_ALTER;
68     ExtBusInitStruct.PORT_MODE = PORT_MODE_DIGITAL;
69     ExtBusInitStruct.PORT_SPEED = PORT_SPEED_FAST;
70
71     PORT_Init(MDR_PORTC, &ExtBusInitStruct);
72
73     // Конфигурация линии адреса
74     if (device == EBC_RAM)
75     {
76         // Сдвиг адресов A2->A0
77         ExtBusInitStruct.PORT_Pin = (PORT_Pin_5 | PORT_Pin_6 | PORT_Pin_7 | PORT_Pin_8 | PORT_Pin_9 |
PORT_Pin_10 | PORT_Pin_11 | PORT_Pin_12 | PORT_Pin_13 | PORT_Pin_14 | PORT_Pin_15);
78     }
79     else if (device == EBC_ROM)
80     {
81         // Нет сдвига адресов A0->A0
82         ExtBusInitStruct.PORT_Pin = (PORT_Pin_3 | PORT_Pin_4 | PORT_Pin_5 | PORT_Pin_6 | PORT_Pin_7 |
PORT_Pin_8 | PORT_Pin_9 | PORT_Pin_10 | PORT_Pin_11 | PORT_Pin_12 | PORT_Pin_13 | PORT_Pin_14 |
PORT_Pin_15);
83     }
84     ExtBusInitStruct.PORT_FUNC = PORT_FUNC_ALTER;
85     ExtBusInitStruct.PORT_MODE = PORT_MODE_DIGITAL;
86     ExtBusInitStruct.PORT_SPEED = PORT_SPEED_FAST;
87
88     PORT_Init(MDR_PORTF, &ExtBusInitStruct);
89
90     // Конфигурация линии адреса A13
91     ExtBusInitStruct.PORT_Pin = PORT_Pin_15;
92     ExtBusInitStruct.PORT_FUNC = PORT_FUNC_ALTER;
93     ExtBusInitStruct.PORT_MODE = PORT_MODE_DIGITAL;
94     ExtBusInitStruct.PORT_SPEED = PORT_SPEED_FAST;
95
96     PORT_Init(MDR_PORTD, &ExtBusInitStruct);
97
98     // Конфигурация линии адреса A14-A19
99     ExtBusInitStruct.PORT_Pin = (PORT_Pin_0 | PORT_Pin_1 | PORT_Pin_2 | PORT_Pin_3 | PORT_Pin_4 |
PORT_Pin_5);
100     ExtBusInitStruct.PORT_FUNC = PORT_FUNC_ALTER;
101     ExtBusInitStruct.PORT_MODE = PORT_MODE_DIGITAL;
102     ExtBusInitStruct.PORT_SPEED = PORT_SPEED_FAST;
103
104     PORT_Init(MDR_PORTE, &ExtBusInitStruct);
105
106     // Конфигурация CS1-PE7(A21) и CS0-PE6(A20)(Ножки выбора ОЗУ)
107     ExtBusInitStruct.PORT_Pin = (PORT_Pin_7 | PORT_Pin_6);
108     ExtBusInitStruct.PORT_FUNC = PORT_FUNC_ALTER;
109     ExtBusInitStruct.PORT_MODE = PORT_MODE_DIGITAL;
110     ExtBusInitStruct.PORT_SPEED = PORT_SPEED_FAST;
111
112     PORT_Init(MDR_PORTE, &ExtBusInitStruct);
113
114     // Конфигурация CS2-PE8(A22)(Ножка выбора ПЗУ)*/
115     ExtBusInitStruct.PORT_Pin = PORT_Pin_8;
116     ExtBusInitStruct.PORT_OE = PORT_OE_OUT;
117     ExtBusInitStruct.PORT_FUNC = PORT_FUNC_ALTER;
118     ExtBusInitStruct.PORT_MODE = PORT_MODE_DIGITAL;
119     ExtBusInitStruct.PORT_PD = PORT_PD_DRIVER;
120     ExtBusInitStruct.PORT_SPEED = PORT_SPEED_SLOW;
121     ExtBusInitStruct.PORT_GFEN = PORT_GFEN_OFF;
122     ExtBusInitStruct.PORT_PULL_UP = PORT_PULL_UP_OFF;
123     ExtBusInitStruct.PORT_PULL_DOWN = PORT_PULL_DOWN_OFF;
124
125     PORT_Init(MDR_PORTE, &ExtBusInitStruct);
126 }

```

#### 4.7.2.2 ebc\_init()

```

void ebc_init (
    ebc_devices device )

```

Инициализирует ЕВС для внешнего ОЗУ или ПЗУ

Аргументы

device	0-RAM, 1-ROM
--------	--------------

```

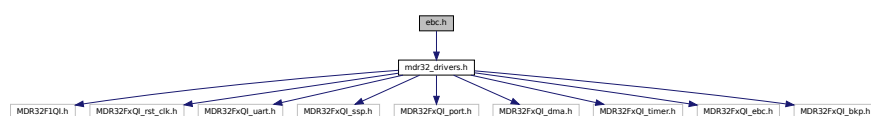
131 {
132     ebc_gpio_config(device);
133
134     EBC_InitTypeDef EBC_InitStruct;
135     EBC_MemRegionInitTypeDef EBC_MemRegionInitStruct;
136
137     // Включение тактирования ЕВС
138     RST_CLK_PCLKcmd(RST_CLK_PCLK_EBC, ENABLE);
139
140     EBC_StructInit(&EBC_InitStruct);
141
142     if (device == EBC_RAM)
143     {
144         EBC_InitStruct.EBC_Mode = EBC_MODE_RAM;
145     }
146     else if (device == EBC_ROM)
147     {
148         EBC_InitStruct.EBC_Mode = EBC_MODE_ROM;
149     }
150
151     EBC_Init(&EBC_InitStruct);
152
153     MDR_EBC->CONTROL = 0x00008001;
154     if (device == EBC_RAM)
155     {
156         MDR_EBC->CONTROL &= ~(1<5); // 32-битный режим данных
157     }
158     else if (device == EBC_ROM)
159     {
160         MDR_EBC->CONTROL |= (1<5); // 8-битный режим данных
161     }
162
163     EBC_MemRegionStructInit(&EBC_MemRegionInitStruct);
164     EBC_MemRegionInitStruct.WS_Setup = 3;
165     EBC_MemRegionInitStruct.WS_Active = 9;
166     EBC_MemRegionInitStruct.WS_Hold = 3;
167     EBC_MemRegionInitStruct.Enable_Tune = ENABLE;
168
169     EBC_MemRegionInit(&EBC_MemRegionInitStruct, EBC_MEM_REGION_50000000);
170 }
```

## 4.8 Файл ebc.h

Заголовочный файл с описанием API для работы с ЕВС (контроллер внешней системной шины)

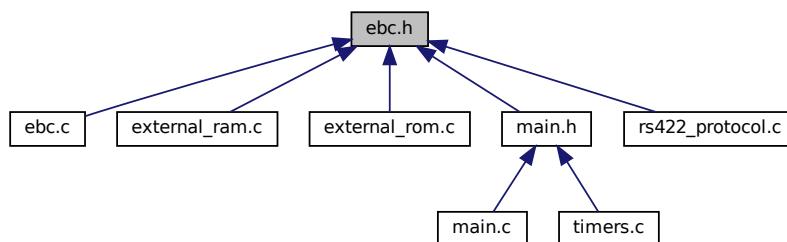
```
#include "mdr32_drivers.h"
```

Граф включаемых заголовочных файлов для ebc.h:





Граф файлов, в которые включается этот файл:



## Определения типов

- typedef enum `devices` `ebc_devices`  
Типы устройств для подключение по EBC.

## Перечисления

- enum `devices` { `EBC_RAM`, `EBC_ROM` }  
Типы устройств для подключение по EBC.

## Функции

- void `ebc_init` (`ebc_devices` device)  
Инициализирует EBC для внешнего ОЗУ или ПЗУ

### 4.8.1 Подробное описание

Заголовочный файл с описанием API для работы с EBC (контроллер внешней системной шины)

### 4.8.2 Типы

#### 4.8.2.1 ebc\_devices

```
typedef enum devices ebc_devices
```

Типы устройств для подключение по EBC.

### 4.8.3 Перечисления

#### 4.8.3.1 devices

```
enum devices
```

Типы устройств для подключение по EBC.

Элементы перечислений

EBC_RAM	RAM.
EBC_ROM	ROM.

```
16 {
17     EBC_RAM,
18     EBC_ROM
19 } ebc_devices;
```

#### 4.8.4 Функции

##### 4.8.4.1 ebc\_init()

```
void ebc_init (
    ebc_devices device )
```

Инициализирует ЕВС для внешнего ОЗУ или ПЗУ

Аргументы

device	0-RAM, 1-ROM
--------	--------------

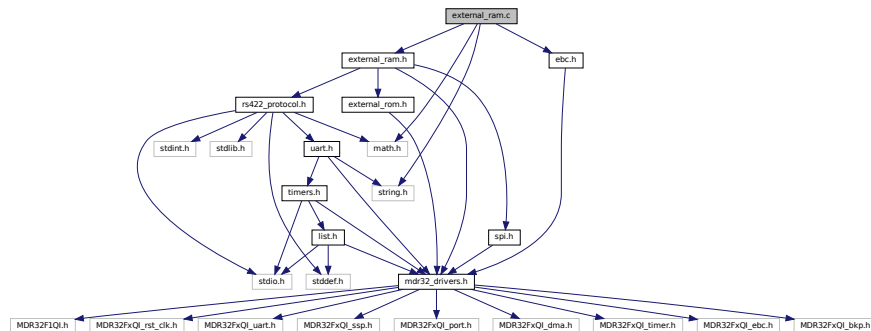
```
131 {
132     ebc_gpio_config(device);
133
134     EBC_InitTypeDef EBC_InitStruct;
135     EBC_MemRegionInitTypeDef EBC_MemRegionInitStruct;
136
137     // Включение тактирования ЕВС
138     RST_CLK_PCLKcmd(RST_CLK_PCLK_EBC, ENABLE);
139
140     EBC_StructInit(&EBC_InitStruct);
141
142     if (device == EBC_RAM)
143     {
144         EBC_InitStruct.EBC_Mode = EBC_MODE_RAM;
145     }
146     else if (device == EBC_ROM)
147     {
148         EBC_InitStruct.EBC_Mode = EBC_MODE_ROM;
149     }
150
151     EBC_Init(&EBC_InitStruct);
152
153     MDR_EBC->CONTROL = 0x00008001;
154     if (device == EBC_RAM)
155     {
156         MDR_EBC->CONTROL &= ~(1<5); // 32-битный режим данных
157     }
158     else if (device == EBC_ROM)
159     {
160         MDR_EBC->CONTROL |= (1<5); // 8-битный режим данных
161     }
162
163     EBC_MemRegionStructInit(&EBC_MemRegionInitStruct);
164     EBC_MemRegionInitStruct.WS_Setup = 3;
165     EBC_MemRegionInitStruct.WS_Active = 9;
166     EBC_MemRegionInitStruct.WS_Hold = 3;
167     EBC_MemRegionInitStruct.Enable_Tune = ENABLE;
168
169     EBC_MemRegionInit(&EBC_MemRegionInitStruct, EBC_MEM_REGION_50000000);
170 }
```

## 4.9 Файл external\_ram.c

Файл с реализацией API для работы с областью памяти внешнего ОЗУ

```
#include "external_ram.h"
#include "ebc.h"
#include <string.h>
#include <math.h>
```

Граф включаемых заголовочных файлов для external\_ram.c:



### Функции

- void `init_external_ram_space` (void)  
Инициализирует область памяти внешнего ОЗУ
- uint16\_t `find_max_halfword` (uint16\_t \*array, uint32\_t array\_size)  
Находит максимальный элемент массива

### Переменные

- ram\_data \* ram\_space\_pointer  
Указатель для обращения к внешнему ОЗУ
- rom\_data \* rom\_space\_pointer  
Указатель для обращения к внешнему ПЗУ
- float `polyn_ch_consts` [MAX\_CHANEL\_NUMBER][7]  
Константы полиномов

#### 4.9.1 Подробное описание

Файл с реализацией API для работы с областью памяти внешнего ОЗУ

#### 4.9.2 Функции

## 4.9.2.1 find\_max\_halfword()

```
uint16_t find_max_halfword (
    uint16_t * array,
    uint32_t array_size )
```

Находит максимальный элемент массива

```
147 {
148     uint16_t result = 0;
149     uint16_t next_item;
150
151     for (int i = 0; i < array_size; i++)
152     {
153         memcpy(&next_item, (uint16_t*)(array + i*sizeof(result)), sizeof(result));
154         if (next_item > result)
155         {
156             result = next_item;
157         }
158     }
159
160     return result;
161 }
```

## 4.9.2.2 init\_external\_ram\_space()

```
void init_external_ram_space (
    void )
```

Инициализирует область памяти внешнего ОЗУ

Функция инициализации области памяти внешнего ОЗУ

```
36 {
37     ram_space_pointer = (ram_data*)EXT_RAM_START_ADDR;
38     rom_space_pointer = (rom_data*)EXT_ROM_START_ADDR;
39
40     ram_start_struct* start_struct_ptr = &ram_space_pointer->start_struct;
41     common_ram_registers* common_ram_reg_ptr = &ram_space_pointer->common_ram_register_space;
42     mpa_rom_registers* mpa_rom_reg_ptr = &ram_space_pointer->mpa_rom_register_space.AI_RomRegs;
43     common_rom_registers* common_rom_reg_ptr = &ram_space_pointer-
        >common_ram_register_space.PLC_CommonRomRegs;
44
45     // Первичная очистка используемого куска памяти ОЗУ
46     memset(ram_space_pointer, 0, sizeof(ram_data));
47
48     // Инициализации стартовой структуры, которая лежит в начале ОЗУ
49     start_struct_ptr->length = START_STRUCT_LENGTH;
50     start_struct_ptr->text_info = START_STRUCT_TEXT_INFO_ADDR;
51     start_struct_ptr->flag_change_struct = START_STRUCT_CHANGE_FLAG;
52     start_struct_ptr->number_of_ranges = START_STRUCT_NUMBER_OF_RANGES;
53     start_struct_ptr->ranges_in_start_struct[0].range_type = START_STRUCT_RANGE0_TYPE;
54     start_struct_ptr->ranges_in_start_struct[0].start_channel_num = START_STRUCT_RANGE0_START_CH_NUM;
55     start_struct_ptr->ranges_in_start_struct[0].address = START_STRUCT_RANGE0_ADDR;
56     start_struct_ptr->ranges_in_start_struct[0].size = START_STRUCT_RANGE0_SIZE;
57     start_struct_ptr->ranges_in_start_struct[1].range_type = START_STRUCT_RANGE1_TYPE;
58     start_struct_ptr->ranges_in_start_struct[1].start_channel_num = START_STRUCT_RANGE1_START_CH_NUM;
59     start_struct_ptr->ranges_in_start_struct[1].address = START_STRUCT_RANGE1_ADDR;
60     start_struct_ptr->ranges_in_start_struct[1].size = START_STRUCT_RANGE1_SIZE;
61     start_struct_ptr->ranges_in_start_struct[2].range_type = START_STRUCT_RANGE2_TYPE;
62     start_struct_ptr->ranges_in_start_struct[2].start_channel_num = START_STRUCT_RANGE2_START_CH_NUM;
63     start_struct_ptr->ranges_in_start_struct[2].address = START_STRUCT_RANGE2_ADDR;
64     start_struct_ptr->ranges_in_start_struct[2].size = START_STRUCT_RANGE2_SIZE;
65
66     // Кладем карту регистров по адресу 200 во внешней ОЗУ и инициализируем ее
67     // Кладем регистры, которые инициализируются в ПО
68     common_ram_reg_ptr->PLC_SoftVer.revision = PLC_SOFT_VER_REVISION;
69     common_ram_reg_ptr->PLC_SoftVer.modification = PLC_SOFT_VER_MODIFICATION;
70     common_ram_reg_ptr->PLC_SoftVer.type = PLC_SOFT_VER_TYPE;
71     common_ram_reg_ptr->PLC_SoftVer.soft_ver = PLC_SOFT_VER_SOFT_VER;
72     common_ram_reg_ptr->PLC_SoftVer.add_info = PLC_SOFT_VER_ADD_INFO;
73     common_ram_reg_ptr->PLC_SoftVer.develop = PLC_SOFT_VER_DEVELOP;
74     common_ram_reg_ptr->PLC_PMAAddr.module_addr = PLC_PM_MODULE_ADDR;
75     common_ram_reg_ptr->PLC_PMAAddr.chassis_addr = PLC_PM_CHASSIS_ADDR;
76     common_ram_reg_ptr->PLC_Config.main_switch = PLC_CONFIG_MAIN_SWITCH;
77     common_ram_reg_ptr->PLC_Config.add_switch_1 = PLC_CONFIG_ADD_SWITCH1;
```

```

78 common_ram_reg_ptr->PLC_Config.add_switch_2 = PLC_CONFIG_ADD_SWITCH2;
79 common_ram_reg_ptr->PLC_Config.reserv = PLC_CONFIG_RESERV;
80 common_ram_reg_ptr->PLC_CM_State = PLC_CM_NOT_INIT;
81
82 // Кладем регистры, которые берутся из ПЗУ
83 // Если используется внешнее ПЗУ (в общем то как и должно быть) зеркализируем данные из ПЗУ в ОЗУ
84 #ifdef ROM_IS_USED
85 common_rom_registers common_regs;
86 mpa_rom_registers mpa_regs;
87
88 ebc_init(EBC_ROM);
89 memcpy(&common_regs, &rom_space_pointer->common_rom_registers_space, sizeof(common_regs));
90 memcpy(&mpa_regs, &rom_space_pointer->mpa_rom_registers_space, sizeof(mpa_regs));
91 ebc_init(EBC_RAM);
92
93 memcpy(&ram_space_pointer->common_ram_register_space.PLC_CommonRomRegs, &common_regs,
sizeof(common_regs));
94 memcpy(&ram_space_pointer->mpa_ram_register_space.AI_RomRegs, &mpa_regs, sizeof(mpa_regs));
95 #endif
96 #ifndef ROM_IS_USED
97 // Если не используется внешнее ПЗУ, то самостоятельно инициализируем регистры в ОЗУ (данный вариант
временный и используется когда нет рабочей ПЗУ)
98 // Инициализация общих регистров
99 strncpy(&common_rom_reg_ptr->PLC_DeviceInfo.DEV_INFO, sizeof(DEV_INFO));
100 common_rom_reg_ptr->PLC_DeviceType.revision = REVISION;
101 common_rom_reg_ptr->PLC_DeviceType.modification = MODIFICATION;
102 common_rom_reg_ptr->PLC_DeviceType.type = MPA;
103 common_rom_reg_ptr->PLC_DeviceType.batch = DEV_TYPE;
104 common_rom_reg_ptr->PLC_DeviceType.reserv = DEV_TYPE_RESERV;
105 common_rom_reg_ptr->PLC_SerialNumber = SERIAL_NUMBER;
106 common_rom_reg_ptr->PLC_TimeoutForDefect_B1 = TIMEOUT_FOR_DEFECT_B1;
107 common_rom_reg_ptr->PLC_TimeoutForDefect_B2 = TIMEOUT_FOR_DEFECT_B2;
108 common_rom_reg_ptr->PLC_NumCrcErrorsForDefect_B1 = NUM_CRC_ERRORS_FOR_DEFECT_B1;
109 common_rom_reg_ptr->PLC_NumCrcErrorsForDefect_B2 = NUM_CRC_ERRORS_FOR_DEFECT_B2;
110 common_rom_reg_ptr->PLC_TimeToRepair = TIME_TO_REPAIR;
111 common_rom_reg_ptr->PLC_TimeSoloWork = TIME_SOLO_WORK;
112 common_rom_reg_ptr->PLC_DualControl = DUAL_CONTROL;
113 memset(&common_rom_reg_ptr->Reserv_2, 0, sizeof(common_rom_reg_ptr->Reserv_2));
114
115 // Инициализация регистров МПА
116 for (uint8_t i = 0; i < MAX_CHANEL_NUMBER; i++)
117 {
118     RESET_BIT(i, mpa_rom_reg_ptr->AI_OperMode.adc_chs_mode);
119     mpa_rom_reg_ptr->AI_NumForAverag[i] = NUM_FOR_AVERAGE;
120     mpa_rom_reg_ptr->AI_MinCodeADC[i] = MIN_CODE_ADC;
121     mpa_rom_reg_ptr->AI_MaxCodeADC[i] = MAX_CODE_ADC;
122     // Такие значения коэф. полиномов только для напряжения 0-10В
123     mpa_rom_reg_ptr->AI_PolynConst0[i] = polyn_ch_consts[i][0];
124     mpa_rom_reg_ptr->AI_PolynConst1[i] = polyn_ch_consts[i][1];
125     mpa_rom_reg_ptr->AI_PolynConst2[i] = polyn_ch_consts[i][2];
126     mpa_rom_reg_ptr->AI_PolynConst3[i] = polyn_ch_consts[i][3];
127     mpa_rom_reg_ptr->AI_PolynConst4[i] = polyn_ch_consts[i][4];
128     mpa_rom_reg_ptr->AI_PolynConst5[i] = polyn_ch_consts[i][5];
129     mpa_rom_reg_ptr->AI_PolynConst6[i] = polyn_ch_consts[i][6];
130     mpa_rom_reg_ptr->AI_MetrologDat[i] = METROLOG_DAT;
131 }
132 memset(mpa_rom_reg_ptr->AI_MetrologDat, 0, sizeof(mpa_rom_reg_ptr->AI_MetrologDat));
133 memset(mpa_rom_reg_ptr->Reserv_4, 0, sizeof(mpa_rom_reg_ptr->Reserv_4));
134 memset(mpa_rom_reg_ptr->Reserv_5, 0, sizeof(mpa_rom_reg_ptr->Reserv_5));
135 #endif
136
137 // Заполняем таблицу CRC32
138 fill_crc32_table();
139 }

```

## 4.9.3 Переменные

### 4.9.3.1 polyn\_ch\_consts

```
float polyn_ch_consts[MAX_CHANEL_NUMBER][7]
```

Инициализатор

```
= {
    {4.94072982f, 0.00015744f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f},
    {4.92697692f, 0.00015771f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f},

```

```

{0.0f,      0.0f,      0.0f, 0.0f, 0.0f, 0.0f},
{0.0f,      0.0f,      0.0f, 0.0f, 0.0f, 0.0f},
{0.0f,      0.0f,      0.0f, 0.0f, 0.0f, 0.0f},
{0.0f,      0.0f,      0.0f, 0.0f, 0.0f, 0.0f},
{0.0f,      0.0f,      0.0f, 0.0f, 0.0f, 0.0f}
}

```

Константы полиномов

#### 4.9.3.2 ram\_space\_pointer

```
ram_data* ram_space_pointer [extern]
```

Указатель для обращения к внешнему ОЗУ

#### 4.9.3.3 rom\_space\_pointer

```
rom_data* rom_space_pointer [extern]
```

Указатель для обращения к внешнему ПЗУ

## 4.10 Файл external\_ram.h

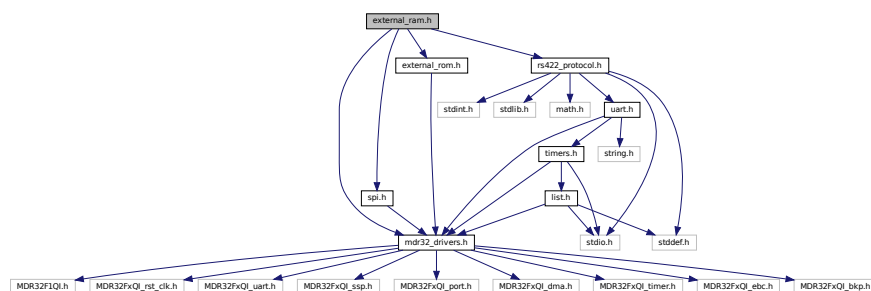
Заголовочный файл с описанием API для работы с областью памяти внешнего ОЗУ

```

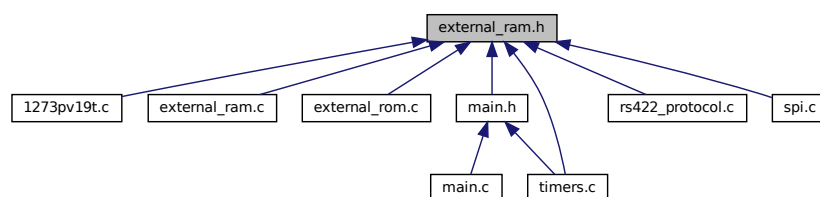
#include "mdr32_drivers.h"
#include "rs422_protocol.h"
#include "spi.h"
#include "external_rom.h"

```

Граф включаемых заголовочных файлов для external\_ram.h:



Граф файлов, в которые включается этот файл:



## Классы

- struct [plc\\_sof\\_ver\\_struct](#)  
Структура с битовыми полями для регистра версия ПО
- struct [device\\_address\\_struct](#)  
Структура с битовыми полями для регистра адрес устройства
- struct [config\\_struct](#)  
Структура с битовыми полями для регистра конфигурация
- struct [power\\_failure\\_struct](#)  
Структура с битовыми полями для регистра неисправность питания
- struct [bus\\_defect\\_struct](#)  
Структура с битовыми полями для регистра неисправность шины
- struct [self\\_diag\\_struct](#)  
Структура с битовыми полями для регистра неисправность самодиагностики
- struct [range\\_start\\_struct](#)  
Структура одного диапазона для стартовой структуры
- struct [start\\_struct\\_ext\\_ram](#)  
Структура, которая лежит в начале ОЗУ любого модуля
- struct [common\\_register\\_space\\_ext\\_ram](#)  
Организация пространства общих регистров во внешнем ОЗУ
- struct [mpa\\_register\\_space\\_ext\\_ram](#)  
Организация пространства регистров МПА во внешнем ОЗУ
- struct [service\\_byte\\_struct\\_pm](#)  
Структура с битовыми полями для сервисного байта ПМ
- struct [service\\_byte\\_struct\\_um](#)  
Структура с битовыми полями для сервисного байта УМ
- struct [ram\\_data\\_struct](#)  
Структура организующая память во внешнем ОЗУ

## Макросы

- #define [EXT\\_RAM\\_START\\_ADDR](#) 0x50200000  
Физический адрес в памяти МК, с которой начинается обращение к внешней ОЗУ
- #define [RAM\\_REGISTER\\_SPACE\\_START\\_ADDR](#) 200  
Стартовый адрес карты регистров в ОЗУ
- #define [TIMER\\_NUM](#) 4  
Кол-во таймеров в МК
- #define [TEST\\_BIT](#)(num, value) ((value>>num)&0x1)  
Макрос проверки бита (0 или 1) в байте
- #define [SET\\_BIT](#)(num, value) (value |= (1<<num))  
Макрос установки бита в байте в 1.
- #define [RESET\\_BIT](#)(num, value) (value &= ~(1<<num))  
Макрос сброса бита в байте в 0.
- #define [START\\_STRUCT\\_LENGTH](#) 32  
Длина структуры
- #define [START\\_STRUCT\\_TEXT\\_INFO\\_ADDR](#) 332  
Ссылка на текстовую информацию о модуле
- #define [START\\_STRUCT\\_CHANGE\\_FLAG](#) 0  
Флаг изменения структуры
- #define [START\\_STRUCT\\_NUMBER\\_OF\\_RANGES](#) 3

- Кол-во диапазонов в стартовой структуре
- #define `START_STRUCT_RANGE0_TYPE` 0x0100  
Тип диапазона (h) и тип операции (l)
- #define `START_STRUCT_RANGE0_START_CH_NUM` 0x0000  
Стартовый номер канала (h) и номер диапазона (l)
- #define `START_STRUCT_RANGE0_ADDR` 1874  
Адрес
- #define `START_STRUCT_RANGE0_SIZE` 32  
Количество байт
- #define `START_STRUCT_RANGE1_TYPE` 0x0400  
Тип диапазона (h) и тип операции (l)
- #define `START_STRUCT_RANGE1_START_CH_NUM` 0x0001  
Стартовый номер канала (h) и номер диапазона (l)
- #define `START_STRUCT_RANGE1_ADDR` 254  
Адрес
- #define `START_STRUCT_RANGE1_SIZE` 10  
Количество байт
- #define `START_STRUCT_RANGE2_TYPE` 0x0500  
Тип диапазона (h) и тип операции (l)
- #define `START_STRUCT_RANGE2_START_CH_NUM` 0x0002  
Стартовый номер канала (h) и номер диапазона (l)
- #define `START_STRUCT_RANGE2_ADDR` 1906  
Адрес
- #define `START_STRUCT_RANGE2_SIZE` 8  
Количество байт
- #define `PLC_SOFT_VER_REVISION` 1  
Ревизия модуля
- #define `PLC_SOFT_VER_MODIFICATION` 2  
Модификация модуля
- #define `PLC_SOFT_VER_TYPE` MPA  
Тип модуля
- #define `PLC_SOFT_VER_SOFT_VER` 1  
Версия ПО
- #define `PLC_SOFT_VER_ADD_INFO` 4  
Доп.информация
- #define `PLC_SOFT_VER_DEVELOP` 1  
1=ПО в процессе разработки
- #define `PLC_PM_MODULE_ADDR` PM\_DEV\_ADDR  
адрес модуля
- #define `PLC_PM_CHASSIS_ADDR` PM\_CHASSIS\_ADDR  
адрес шасси
- #define `PLC_CONFIG_MAIN_SWITCH` 4  
Основной свитч
- #define `PLC_CONFIG_ADD_SWITCH1` 3  
1 Доп свитч (при наличии)
- #define `PLC_CONFIG_ADD_SWITCH2` 2  
2 Доп свитч (при наличии)
- #define `PLC_CONFIG_RESERV` 1  
резерв



## Определения типов

- typedef enum [type\\_of\\_module](#) [module\\_type](#)  
Типы модулей
- typedef struct [plc\\_soft\\_ver\\_struct](#) [plc\\_soft\\_ver](#)  
Структура с битовыми полями для регистра версия ПО
- typedef struct [device\\_address\\_struct](#) [device\\_address](#)  
Структура с битовыми полями для регистра адрес устройства
- typedef struct [config\\_struct](#) [device\\_config](#)  
Структура с битовыми полями для регистра конфигурация
- typedef struct [power\\_failure\\_struct](#) [power\\_failure](#)  
Структура с битовыми полями для регистра неисправность питания
- typedef struct [bus\\_defect\\_struct](#) [bus\\_defect](#)  
Структура с битовыми полями для регистра неисправность шины
- typedef struct [self\\_diag\\_struct](#) [self\\_diag](#)  
Структура с битовыми полями для регистра неисправность самодиагностики
- typedef struct [range\\_start\\_struct](#) [range](#)  
Структура одного диапазона для стартовой структуры
- typedef struct [start\\_struct\\_ext\\_ram](#) [ram\\_start\\_struct](#)  
Структура, которая лежит в начале ОЗУ любого модуля
- typedef struct [common\\_register\\_space\\_ext\\_ram](#) [common\\_ram\\_registers](#)  
Организация пространства общих регистров во внешнем ОЗУ
- typedef struct [mpa\\_register\\_space\\_ext\\_ram](#) [mpa\\_ram\\_registers](#)  
Организация пространства регистров МПА во внешнем ОЗУ
- typedef struct [service\\_byte\\_struct\\_pm](#) [service\\_struct\\_pm](#)  
Структура с битовыми полями для сервисного байта ПМ
- typedef struct [service\\_byte\\_struct\\_um](#) [service\\_struct\\_um](#)  
Структура с битовыми полями для сервисного байта УМ
- typedef struct [ram\\_data\\_struct](#) [ram\\_data](#)  
Структура организующая память во внешнем ОЗУ

## Перечисления

- enum [type\\_of\\_module](#) {  
[MPD](#) = 1, [MVD](#), [MVD\\_U](#), [MPT](#),  
[MVA](#), [MPA](#), [MPI\\_U](#), [MSR](#),  
[MPCH](#), [MPPT](#), [MPI](#) }  
 Типы модулей

## Функции

- void [init\\_external\\_ram\\_space](#) (void)  
Инициализирует область памяти внешнего ОЗУ

### 4.10.1 Подробное описание

Заголовочный файл с описанием API для работы с областью памяти внешнего ОЗУ

## 4.10.2 Макросы

### 4.10.2.1 EXT\_RAM\_START\_ADDR

```
#define EXT_RAM_START_ADDR 0x50200000
```

Физический адрес в памяти МК, с которой начинается обращение к внешней ОЗУ

### 4.10.2.2 PLC\_CONFIG\_ADD\_SWITCH1

```
#define PLC_CONFIG_ADD_SWITCH1 3
```

1 Доп свитч (при наличии)

### 4.10.2.3 PLC\_CONFIG\_ADD\_SWITCH2

```
#define PLC_CONFIG_ADD_SWITCH2 2
```

2 Доп свитч (при наличии)

### 4.10.2.4 PLC\_CONFIG\_MAIN\_SWITCH

```
#define PLC_CONFIG_MAIN_SWITCH 4
```

Основной свитч

### 4.10.2.5 PLC\_CONFIG\_RESERV

```
#define PLC_CONFIG_RESERV 1
```

резерв

### 4.10.2.6 PLC\_PM\_CHASSIS\_ADDR

```
#define PLC_PM_CHASSIS_ADDR PM\_CHASSIS\_ADDR
```

адрес шасси

## 4.10.2.7 PLC\_PM\_MODULE\_ADDR

```
#define PLC_PM_MODULE_ADDR PM_DEV_ADDR
```

адрес модуля

## 4.10.2.8 PLC\_SOFT\_VER\_ADD\_INFO

```
#define PLC_SOFT_VER_ADD_INFO 4
```

Доп. информация

## 4.10.2.9 PLC\_SOFT\_VER\_DEVELOP

```
#define PLC_SOFT_VER_DEVELOP 1
```

1=ПО в процессе разработки

## 4.10.2.10 PLC\_SOFT\_VER\_MODIFICATION

```
#define PLC_SOFT_VER_MODIFICATION 2
```

Модификация модуля

## 4.10.2.11 PLC\_SOFT\_VER\_REVISION

```
#define PLC_SOFT_VER_REVISION 1
```

Ревизия модуля

## 4.10.2.12 PLC\_SOFT\_VER\_SOFT\_VER

```
#define PLC_SOFT_VER_SOFT_VER 1
```

Версия ПО

#### 4.10.2.13 PLC\_SOFT\_VER\_TYPE

```
#define PLC_SOFT_VER_TYPE MPA
```

Тип модуля

#### 4.10.2.14 RAM\_REGISTER\_SPACE\_START\_ADDR

```
#define RAM_REGISTER_SPACE_START_ADDR 200
```

Стартовый адрес карты регистров в ОЗУ

#### 4.10.2.15 RESET\_BIT

```
#define RESET_BIT(  
    num,  
    value ) (value &= ~(1<<num))
```

Макрос сброса бита в байте в 0.

#### 4.10.2.16 SET\_BIT

```
#define SET_BIT(  
    num,  
    value ) (value |= (1<<num))
```

Макрос установки бита в байте в 1.

#### 4.10.2.17 START\_STRUCT\_CHANGE\_FLAG

```
#define START_STRUCT_CHANGE_FLAG 0
```

Флаг изменения структуры

#### 4.10.2.18 START\_STRUCT\_LENGTH

```
#define START_STRUCT_LENGTH 32
```

Длина структуры

## 4.10.2.19 START\_STRUCT\_NUMBER\_OF\_RANGES

```
#define START_STRUCT_NUMBER_OF_RANGES 3
```

Кол-во диапазонов в стартовой структуре

## 4.10.2.20 START\_STRUCT\_RANGE0\_ADDR

```
#define START_STRUCT_RANGE0_ADDR 1874
```

Адрес

## 4.10.2.21 START\_STRUCT\_RANGE0\_SIZE

```
#define START_STRUCT_RANGE0_SIZE 32
```

Количество байт

## 4.10.2.22 START\_STRUCT\_RANGE0\_START\_CH\_NUM

```
#define START_STRUCT_RANGE0_START_CH_NUM 0x0000
```

Стартовый номер канала (h) и номер диапазона (l)

## 4.10.2.23 START\_STRUCT\_RANGE0\_TYPE

```
#define START_STRUCT_RANGE0_TYPE 0x0100
```

Тип диапазона (h) и тип операции (l)

## 4.10.2.24 START\_STRUCT\_RANGE1\_ADDR

```
#define START_STRUCT_RANGE1_ADDR 254
```

Адрес

#### 4.10.2.25 START\_STRUCT\_RANGE1\_SIZE

```
#define START_STRUCT_RANGE1_SIZE 10
```

Количество байт

#### 4.10.2.26 START\_STRUCT\_RANGE1\_START\_CH\_NUM

```
#define START_STRUCT_RANGE1_START_CH_NUM 0x0001
```

Стартовый номер канала (h) и номер диапазона (l)

#### 4.10.2.27 START\_STRUCT\_RANGE1\_TYPE

```
#define START_STRUCT_RANGE1_TYPE 0x0400
```

Тип диапазона (h) и тип операции (l)

#### 4.10.2.28 START\_STRUCT\_RANGE2\_ADDR

```
#define START_STRUCT_RANGE2_ADDR 1906
```

Адрес

#### 4.10.2.29 START\_STRUCT\_RANGE2\_SIZE

```
#define START_STRUCT_RANGE2_SIZE 8
```

Количество байт

#### 4.10.2.30 START\_STRUCT\_RANGE2\_START\_CH\_NUM

```
#define START_STRUCT_RANGE2_START_CH_NUM 0x0002
```

Стартовый номер канала (h) и номер диапазона (l)

## 4.10.2.31 START\_STRUCT\_RANGE2\_TYPE

```
#define START_STRUCT_RANGE2_TYPE 0x0500
```

Тип диапазона (h) и тип операции (l)

## 4.10.2.32 START\_STRUCT\_TEXT\_INFO\_ADDR

```
#define START_STRUCT_TEXT_INFO_ADDR 332
```

Ссылка на текстовую информацию о модуле

## 4.10.2.33 TEST\_BIT

```
#define TEST_BIT(  
    num,  
    value ) ((value>>num)&0x1)
```

Макрос проверки бита (0 или 1) в байте

## 4.10.2.34 TIMER\_NUM

```
#define TIMER_NUM 4
```

Кол-во таймеров в МК

## 4.10.3 Типы

## 4.10.3.1 bus\_defect

```
typedef struct bus_defect_struct bus_defect
```

Структура с битовыми полями для регистра неисправность шины

## 4.10.3.2 common\_ram\_registers

```
typedef struct common_register_space_ext_ram common_ram_registers
```

Организация пространства общих регистров во внешнем ОЗУ

#### 4.10.3.3 device\_address

```
typedef struct device_address_struct device_address
```

Структура с битовыми полями для регистра адрес устройства

#### 4.10.3.4 device\_config

```
typedef struct config_struct device_config
```

Структура с битовыми полями для регистра конфигурация

#### 4.10.3.5 module\_type

```
typedef enum type_of_module module_type
```

Типы модулей

#### 4.10.3.6 mpa\_ram\_registers

```
typedef struct mpa_register_space_ext_ram mpa_ram_registers
```

Организация пространства регистров МПА во внешнем ОЗУ

#### 4.10.3.7 plc\_soft\_ver

```
typedef struct plc_soft_ver_struct plc_soft_ver
```

Структура с битовыми полями для регистра версия ПО

#### 4.10.3.8 power\_failure

```
typedef struct power_failure_struct power_failure
```

Структура с битовыми полями для регистра неисправность питания



#### 4.10.3.9 ram\_data

```
typedef struct ram_data_struct ram_data
```

Структура организующая память во внешнем ОЗУ

#### 4.10.3.10 ram\_start\_struct

```
typedef struct start_struct_ext_ram ram_start_struct
```

Структура, которая лежит в начале ОЗУ любого модуля

#### 4.10.3.11 range

```
typedef struct range_start_struct range
```

Структура одного диапазона для стартовой структуры

#### 4.10.3.12 self\_diag

```
typedef struct self_diag_struct self_diag
```

Структура с битовыми полями для регистра неисправность самодиагностики

#### 4.10.3.13 service\_struct\_pm

```
typedef struct service_byte_struct_pm service_struct_pm
```

Структура с битовыми полями для сервисного байта ПМ

#### 4.10.3.14 service\_struct\_um

```
typedef struct service_byte_struct_um service_struct_um
```

Структура с битовыми полями для сервисного байта УМ

### 4.10.4 Перечисления

#### 4.10.4.1 type\_of\_module

```
enum type_of_module
```

Типы модулей

Элементы перечислений

MPD	
MVD	
MVD_U	
MPT	
MVA	
MPA	
MPI_U	
MSR	
MPCH	
MPPT	
MPI	

```

58 {
59     MPD = 1,
60     MVD,
61     MVD_U,
62     MPT,
63     MVA,
64     MPA,
65     MPI_U,
66     MSR,
67     MPCH,
68     MPPT,
69     MPI
70 } module_type;
```

#### 4.10.5 Функции

##### 4.10.5.1 init\_external\_ram\_space()

```

void init_external_ram_space (
    void )
```

Инициализирует область памяти внешнего ОЗУ

Функция инициализации области памяти внешнего ОЗУ

```

36 {
37     ram_space_pointer = (ram_data*)EXT_RAM_START_ADDR;
38     rom_space_pointer = (rom_data*)EXT_ROM_START_ADDR;
39
40     ram_start_struct* start_struct_ptr = &ram_space_pointer->start_struct;
41     common_ram_registers* common_ram_reg_ptr = &ram_space_pointer->common_ram_register_space;
42     mpa_rom_registers* mpa_rom_reg_ptr = &ram_space_pointer->mpa_rom_register_space.AI_RomRegs;
43     common_rom_registers* common_rom_reg_ptr = &ram_space_pointer-
44         >common_ram_register_space.PLC_CommonRomRegs;
45
46     // Первичная очистка используемого куска памяти ОЗУ
47     memset(ram_space_pointer, 0, sizeof(ram_data));
48
49     // Инициализации стартовой структуры, которая лежит в начале ОЗУ
50     start_struct_ptr->length = START_STRUCT_LENGTH;
51     start_struct_ptr->text_info = START_STRUCT_TEXT_INFO_ADDR;
52     start_struct_ptr->flag_change_struct = START_STRUCT_CHANGE_FLAG;
53     start_struct_ptr->number_of_ranges = START_STRUCT_NUMBER_OF_RANGES;
54     start_struct_ptr->ranges_in_start_struct[0].range_type = START_STRUCT_RANGE0_TYPE;
55     start_struct_ptr->ranges_in_start_struct[0].start_channel_num = START_STRUCT_RANGE0_START_CH_NUM;
56     start_struct_ptr->ranges_in_start_struct[0].address = START_STRUCT_RANGE0_ADDR;
57     start_struct_ptr->ranges_in_start_struct[0].size = START_STRUCT_RANGE0_SIZE;
58     start_struct_ptr->ranges_in_start_struct[1].range_type = START_STRUCT_RANGE1_TYPE;
59     start_struct_ptr->ranges_in_start_struct[1].start_channel_num = START_STRUCT_RANGE1_START_CH_NUM;
60     start_struct_ptr->ranges_in_start_struct[1].address = START_STRUCT_RANGE1_ADDR;
```

```

60 start_struct_ptr->ranges_in_start_struct[1].size = START_STRUCT_RANGE1_SIZE;
61 start_struct_ptr->ranges_in_start_struct[2].range_type = START_STRUCT_RANGE2_TYPE;
62 start_struct_ptr->ranges_in_start_struct[2].start_channel_num = START_STRUCT_RANGE2_START_CH_NUM;
63 start_struct_ptr->ranges_in_start_struct[2].address = START_STRUCT_RANGE2_ADDR;
64 start_struct_ptr->ranges_in_start_struct[2].size = START_STRUCT_RANGE2_SIZE;
65
66 // Кладем карту регистров по адресу 200 во внешней ОЗУ и инициализируем ее
67 // Кладем регистры, которые инициализируются в ПО
68 common_ram_reg_ptr->PLC_SoftVer.revision = PLC_SOFT_VER_REVISION;
69 common_ram_reg_ptr->PLC_SoftVer.modification = PLC_SOFT_VER_MODIFICATION;
70 common_ram_reg_ptr->PLC_SoftVer.type = PLC_SOFT_VER_TYPE;
71 common_ram_reg_ptr->PLC_SoftVer.soft_ver = PLC_SOFT_VER_SOFT_VER;
72 common_ram_reg_ptr->PLC_SoftVer.add_info = PLC_SOFT_VER_ADD_INFO;
73 common_ram_reg_ptr->PLC_SoftVer.develop = PLC_SOFT_VER_DEVELOP;
74 common_ram_reg_ptr->PLC_PMAAddr.module_addr = PLC_PM_MODULE_ADDR;
75 common_ram_reg_ptr->PLC_PMAAddr.chassis_addr = PLC_PM_CHASSIS_ADDR;
76 common_ram_reg_ptr->PLC_Config.main_switch = PLC_CONFIG_MAIN_SWITCH;
77 common_ram_reg_ptr->PLC_Config.add_switch_1 = PLC_CONFIG_ADD_SWITCH1;
78 common_ram_reg_ptr->PLC_Config.add_switch_2 = PLC_CONFIG_ADD_SWITCH2;
79 common_ram_reg_ptr->PLC_Config.reserv = PLC_CONFIG_RESERV;
80 common_ram_reg_ptr->PLC_CM_State = PLC_CM_NOT_INIT;
81
82 // Кладем регистры, которые берутся из ПЗУ
83 // Если используется внешнее ПЗУ (в общем то как и должно быть) зеркалируем данные из ПЗУ в ОЗУ
84 #ifdef ROM_IS_USED
85 common_rom_registers common_regs;
86 mpa_rom_registers mpa_regs;
87
88 ebc_init(EBC_ROM);
89 memcpy(&common_regs, &rom_space_pointer->common_rom_registers_space, sizeof(common_regs));
90 memcpy(&mpa_regs, &rom_space_pointer->mpa_rom_registers_space, sizeof(mpa_regs));
91 ebc_init(EBC_RAM);
92
93 memcpy(&ram_space_pointer->common_ram_register_space.PLC_CommonRomRegs, &common_regs,
94 sizeof(common_regs));
95 memcpy(&ram_space_pointer->mpa_ram_register_space.AI_RomRegs, &mpa_regs, sizeof(mpa_regs));
96 #endif
97 #ifndef ROM_IS_USED
98 // Если не используется внешнее ПЗУ, то самостоятельно инициализируем регистры в ОЗУ (данный вариант
99 // временный и используется когда нет рабочей ПЗУ)
100 // Инициализация общих регистров
101 structcpy(&common_rom_reg_ptr->PLC_DeviceInfo, DEV_INFO, sizeof(DEV_INFO));
102 common_rom_reg_ptr->PLC_DeviceType.revision = REVISION;
103 common_rom_reg_ptr->PLC_DeviceType.modification = MODIFICATION;
104 common_rom_reg_ptr->PLC_DeviceType.type = MPA;
105 common_rom_reg_ptr->PLC_DeviceType.batch = DEV_TYPE;
106 common_rom_reg_ptr->PLC_DeviceType.reserv = DEV_TYPE_RESERV;
107 common_rom_reg_ptr->PLC_SerialNumber = SERIAL_NUMBER;
108 common_rom_reg_ptr->PLC_TimeoutForDefect_B1 = TIMEOUT_FOR_DEFECT_B1;
109 common_rom_reg_ptr->PLC_TimeoutForDefect_B2 = TIMEOUT_FOR_DEFECT_B2;
110 common_rom_reg_ptr->PLC_NumCrcErrorsForDefect_B1 = NUM_CRC_ERRORS_FOR_DEFECT_B1;
111 common_rom_reg_ptr->PLC_NumCrcErrorsForDefect_B2 = NUM_CRC_ERRORS_FOR_DEFECT_B2;
112 common_rom_reg_ptr->PLC_TimeToRepair = TIME_TO_REPAIR;
113 common_rom_reg_ptr->PLC_TimeSoloWork = TIME_SOLO_WORK;
114 common_rom_reg_ptr->PLC_DualControl = DUAL_CONTROL;
115 memset(&common_rom_reg_ptr->Reserv_2, 0, sizeof(common_rom_reg_ptr->Reserv_2));
116
117 // Инициализация регистров МПА
118 for (uint8_t i = 0; i < MAX_CHANEL_NUMBER; i++)
119 {
120     RESET_BIT(i, mpa_rom_reg_ptr->AI_OperMode.adc_chs_mode);
121     mpa_rom_reg_ptr->AI_NumForAverag[i] = NUM_FOR_AVERAGE;
122     mpa_rom_reg_ptr->AI_MinCodeADC[i] = MIN_CODE_ADC;
123     mpa_rom_reg_ptr->AI_MaxCodeADC[i] = MAX_CODE_ADC;
124     // Такие значения коэф. полиномов только для напряжения 0-10В
125     mpa_rom_reg_ptr->AI_PolynConst0[i] = polyn_ch_consts[i][0];
126     mpa_rom_reg_ptr->AI_PolynConst1[i] = polyn_ch_consts[i][1];
127     mpa_rom_reg_ptr->AI_PolynConst2[i] = polyn_ch_consts[i][2];
128     mpa_rom_reg_ptr->AI_PolynConst3[i] = polyn_ch_consts[i][3];
129     mpa_rom_reg_ptr->AI_PolynConst4[i] = polyn_ch_consts[i][4];
130     mpa_rom_reg_ptr->AI_PolynConst5[i] = polyn_ch_consts[i][5];
131     mpa_rom_reg_ptr->AI_PolynConst6[i] = polyn_ch_consts[i][6];
132     mpa_rom_reg_ptr->AI_MetrologDat[i] = METROLOG_DAT;
133 }
134 memset(mpa_rom_reg_ptr->AI_MetrologDat, 0, sizeof(mpa_rom_reg_ptr->AI_MetrologDat));
135 memset(mpa_rom_reg_ptr->Reserv_4, 0, sizeof(mpa_rom_reg_ptr->Reserv_4));
136 memset(mpa_rom_reg_ptr->Reserv_5, 0, sizeof(mpa_rom_reg_ptr->Reserv_5));
137 #endif
138 // Заполняем таблицу CRC32
139 fill_crc32_table();
140 }

```



## 4.11.2 Макросы

### 4.11.2.1 FIRST\_TIME\_INIT

```
#define FIRST_TIME_INIT
```

Макрос для записи данных (по умолчанию) в ПЗУ в первый раз

## 4.11.3 Функции

### 4.11.3.1 erase\_rom()

```
void erase_rom (  
    void )
```

Очищает память в ПЗУ

Функция очистки памяти ПЗУ

```
142 {  
143     uint8_t status;  
144  
145     // Команда Reset  
146     HWREG(EXT_ROM_START_ADDR) = 0xF0;  
147  
148     // Конфигурирование выводов EBC на выход  
149     MDR_PORTA->OE = 0x0000FFFF;  
150  
151     // Отправка соответствующей команды  
152     HWREG(EXT_ROM_START_ADDR + 0x555) = 0xAA;  
153     HWREG(EXT_ROM_START_ADDR + 0x2AA) = 0x55;  
154     HWREG(EXT_ROM_START_ADDR + 0x555) = 0x80;  
155     HWREG(EXT_ROM_START_ADDR + 0x555) = 0xAA;  
156     HWREG(EXT_ROM_START_ADDR + 0x2AA) = 0x55;  
157     HWREG(EXT_ROM_START_ADDR + 0x555) = 0x10;  
158  
159     // Конфигурирование выводов EBC на вход  
160     MDR_PORTA->OE = 0x00000000;  
161  
162     do status = HWREG(EXT_ROM_START_ADDR);  
163     while ((status & 0x80) != 0x80);  
164 }
```

#### 4.11.3.2 init\_external\_rom\_space()

```
void init_external_rom_space (
    void )
```

Инициализирует область памяти внешнего ПЗУ

Функция инициализации области памяти внешнего ПЗУ

```
23 {
24     #ifdef FIRST_TIME_INIT
25         common_rom_registers    common_regs;
26         mpa_rom_registers       mpa_regs;
27
28         // Очистка ПЗУ
29         erase_rom();
30
31         // Инициализация общих регистров
32         strncpy(&common_regs.PLC_DeviceInfo.DEV_INFO,sizeof(DEV_INFO));
33         common_regs.PLC_DeviceType.revision = REVISION;
34         common_regs.PLC_DeviceType.modification = MODIFICATION;
35         common_regs.PLC_DeviceType.type = DEV_TYPE;
36         common_regs.PLC_DeviceType.batch = BATCH;
37         common_regs.PLC_DeviceType.reserv = DEV_TYPE_RESERV;
38         common_regs.PLC_SerialNumber = SERIAL_NUMBER;
39         common_regs.PLC_TimeoutForDefect_B1 = TIMEOUT_FOR_DEFECT_B1;
40         common_regs.PLC_TimeoutForDefect_B2 = TIMEOUT_FOR_DEFECT_B2;
41         common_regs.PLC_NumCrcErrorsForDefect_B1 = NUM_CRC_ERRORS_FOR_DEFECT_B1;
42         common_regs.PLC_NumCrcErrorsForDefect_B2 = NUM_CRC_ERRORS_FOR_DEFECT_B2;
43         common_regs.PLC_TimeToRepair = TIME_TO_REPAIR;
44         common_regs.PLC_TimeSoloWork = TIME_SOLO_WORK;
45         common_regs.PLC_DualControl = DUAL_CONTROL;
46         memset(&common_regs.Reserv_2, 0, sizeof(common_regs.Reserv_2));
47
48         // Инициализация регистров МПА
49         for (uint8_t i = 0; i < MAX_CHANEL_NUMBER; i++)
50         {
51             RESET_BIT(i, mpa_regs.AI_OperMode.adc_chs_mode);
52             mpa_regs.AI_NumForAverag[i] = NUM_FOR_AVERAGE;
53             mpa_regs.AI_MinCodeADC[i] = MIN_CODE_ADC;
54             mpa_regs.AI_MaxCodeADC[i] = MAX_CODE_ADC;
55             // Такие значения коэф. полиномов только для напряжения 0-10В
56             mpa_regs.AI_PolynConst0[i] = polyn_ch_consts[i][0];
57             mpa_regs.AI_PolynConst1[i] = polyn_ch_consts[i][1];
58             mpa_regs.AI_PolynConst2[i] = polyn_ch_consts[i][2];
59             mpa_regs.AI_PolynConst3[i] = polyn_ch_consts[i][3];
60             mpa_regs.AI_PolynConst4[i] = polyn_ch_consts[i][4];
61             mpa_regs.AI_PolynConst5[i] = polyn_ch_consts[i][5];
62             mpa_regs.AI_PolynConst6[i] = polyn_ch_consts[i][6];
63             mpa_regs.AI_MetrologDat[i] = METROLOG_DAT;
64         }
65         memset(mpa_regs.AI_MetrologDat, 0, sizeof(mpa_regs.AI_MetrologDat));
66         memset(mpa_regs.Reserv_4, 0, sizeof(mpa_regs.Reserv_4));
67         memset(mpa_regs.Reserv_5, 0, sizeof(mpa_regs.Reserv_5));
68
69         // Копирование данных в ПЗУ
70         memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR, &common_regs, sizeof(common_regs));
71         memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR + sizeof(common_regs), &mpa_regs,
72             sizeof(mpa_regs));
73     #endif
74 }
```

#### 4.11.3.3 memcpy\_to\_rom()

```
void memcpy_to_rom (
    uint32_t dest_addr,
    void * src_addr,
    uint32_t size )
```

Копирует область памяти в ПЗУ

Функция копирования области памяти в ПЗУ

```
131 {
132     for (uint32_t i = 0; i < size; i++)
133     {
134         write_byte_rom(dest_addr + i, *((uint8_t*)src_addr+i));
135     }
136 }
```

## 4.11.3.4 read\_byte\_rom()

```
uint8_t read_byte_rom (
    uint32_t dest_addr )
```

Читает байт из ПЗУ

Функция чтения байта из ПЗУ

```
121 {
122     // Конфигурирование выводов ЕВС на вход
123     MDR_PORTA->OE = 0x00000000;
124     return (HWREG(dest_addr));
125 }
```

## 4.11.3.5 write\_byte\_rom()

```
uint8_t write_byte_rom (
    uint32_t dest_addr,
    uint8_t byte )
```

Записывает байт в ПЗУ

Функция записи байта в ПЗУ

```
80 {
81     uint8_t status;
82
83     // Конфигурирование выводов ЕВС на выход
84     MDR_PORTA->OE = 0x0000FFFF;
85
86     HWREG(EXT_ROM_START_ADDR + 0x555) = 0xAA;
87     HWREG(EXT_ROM_START_ADDR + 0x2AA) = 0x55;
88     HWREG(EXT_ROM_START_ADDR + 0x555) = 0xA0;
89     HWREG(dest_addr + EXT_ROM_START_ADDR) = byte;
90
91     // Конфигурирование выводов ЕВС на вход
92     MDR_PORTA->OE = 0x00000000;
93
94     // Теперь проверяем, выполнялась ли запись успешно (успех = бит D7 == data[7])
95     do status = HWREG(dest_addr + EXT_ROM_START_ADDR);
96     while(((status & 0x80) != (byte & 0x80)) & ((status & 0x20) == 0));
97
98     if((status & 0x20) != 0)
99     {
100         status = HWREG(dest_addr + EXT_ROM_START_ADDR);
101         if((status & 0x80) == (byte & 0x80)) // Успех
102         {
103             return 0;
104         }
105         else
106         {
107             return 1; // Ошибка записи
108         }
109     }
110     else if((status & 0x80) == (byte & 0x80)) // Успех
111     {
112         return 0;
113     }
114     return 2;
115 }
```

## 4.11.4 Переменные

#### 4.11.4.1 polyn\_ch\_consts

```
float polyn_ch_consts[MAX_CHANEL_NUMBER][7] [extern]
```

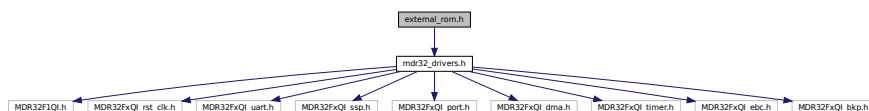
Константы полиномов

## 4.12 Файл external\_rom.h

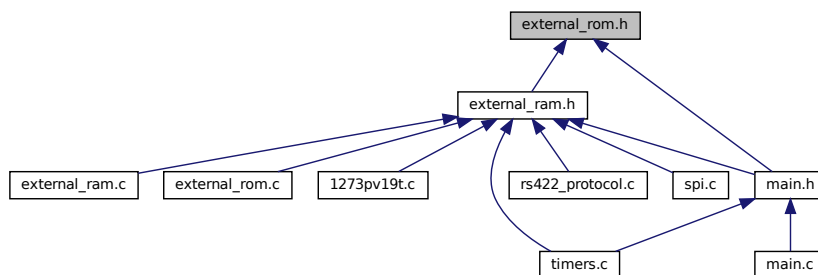
Заголовочный файл с описанием API для работы с областью памяти внешнего ПЗУ

```
#include "mdr32_drivers.h"
```

Граф включаемых заголовочных файлов для external\_rom.h:



Граф файлов, в которые включается этот файл:



## Классы

- struct [device\\_type\\_struct](#)  
Структура с битовыми полями для регистра тип устройства
- struct [ai\\_oper\\_mode\\_struct](#)  
Структура с битовыми полями для регистра режим работы канала
- struct [common\\_register\\_space\\_ext\\_rom](#)  
Организация пространства общих регистров для зеркализации во внешнем ПЗУ
- struct [mpa\\_register\\_space\\_ext\\_rom](#)  
Организация пространства регистров МПА для зеркализации во внешнем ПЗУ
- struct [rom\\_data\\_struct](#)  
Структура организующая память во внешнем ПЗУ



## Макросы

- `#define HWREG(x) *(( unsigned char *)(x))`  
Макрос чтения байта по адресу
- `#define EXT_ROM_START_ADDR 0x00100000`  
Адрес в памяти МК, с которой начинается обращение к внешней ПЗУ
- `#define ROM_REGISTER_SPACE_START_ADDR 0`  
Стартовый адрес карты регистров в ПЗУ
- `#define DEV_TYPE MPA`  
Тип устройства (значение для инициализации по умолчанию)
- `#define DEV_INFO "MPA"`  
Текстовая информация о модуле (значение для инициализации по умолчанию)
- `#define REVISION 1`  
Ревизия модуля (значение для инициализации по умолчанию)
- `#define MODIFICATION 2`  
Модификация модуля (значение для инициализации по умолчанию)
- `#define BATCH 1`  
Партия (значение для инициализации по умолчанию)
- `#define DEV_TYPE_RESERV 20`  
Резерв (значение для инициализации по умолчанию)
- `#define SERIAL_NUMBER 1717986918`  
Серийный номер устройства (значение для инициализации по умолчанию)
- `#define TIMEOUT_FOR_DEFECT_B1 200`  
Время без связи до неспр шины 1, мс (значение для инициализации по умолчанию)
- `#define TIMEOUT_FOR_DEFECT_B2 200`  
Время без связи до неспр шины 2, мс (значение для инициализации по умолчанию)
- `#define NUM_CRC_ERRORS_FOR_DEFECT_B1 6`  
Количество ошибок приема пакета до неисправности шины 1 (значение для инициализации по умолчанию)
- `#define NUM_CRC_ERRORS_FOR_DEFECT_B2 6`  
Количество ошибок приема пакета до неисправности шины 2 (значение для инициализации по умолчанию)
- `#define TIME_TO_REPAIR 65535`  
Максимально время переключения на резервный УМ, x10мс (значение для инициализации по умолчанию)
- `#define TIME_SOLO_WORK 61166`  
Время работы без резервирующего УМ, с (значение для инициализации по умолчанию)
- `#define DUAL_CONTROL 56797`  
Реакция при одновременном управлении (значение для инициализации по умолчанию)
- `#define NUM_FOR_AVERAGE 10`  
Кол-во выборок для усреднения (значение для инициализации по умолчанию)
- `#define MIN_CODE_ADC 0`  
Минимальный код АЦП (значение для инициализации по умолчанию)
- `#define MAX_CODE_ADC 65535`  
Максимальный код АЦП (значение для инициализации по умолчанию)
- `#define METROLOG_DAT 0.0f`  
Сведения о метрологии (значение для инициализации по умолчанию)

## Определения типов

- typedef struct [device\\_type\\_struct](#) [device\\_type](#)  
Структура с битовыми полями для регистра тип устройства
- typedef struct [ai\\_oper\\_mode\\_struct](#) [ai\\_oper\\_mode](#)  
Структура с битовыми полями для регистра режим работы канала
- typedef struct [common\\_register\\_space\\_ext\\_rom](#) [common\\_rom\\_registers](#)  
Организация пространства общих регистров для зеркализации во внешнем ПЗУ
- typedef struct [mpa\\_register\\_space\\_ext\\_rom](#) [mpa\\_rom\\_registers](#)  
Организация пространства регистров МПА для зеркализации во внешнем ПЗУ
- typedef struct [rom\\_data\\_struct](#) [rom\\_data](#)  
Структура организующая память во внешнем ПЗУ

## Функции

- void [init\\_external\\_rom\\_space](#) (void)  
Инициализирует область памяти внешнего ПЗУ
- uint8\_t [write\\_byte\\_rom](#) (uint32\_t dest\_addr, uint8\_t byte)  
Записывает байт в ПЗУ
- uint8\_t [read\\_byte\\_rom](#) (uint32\_t dest\_addr)  
Читает байт из ПЗУ
- void [memcpy\\_to\\_rom](#) (uint32\_t dest\_addr, void \*src\_addr, uint32\_t size)  
Копирует область памяти в ПЗУ
- void [erase\\_rom](#) (void)  
Очищает память в ПЗУ

### 4.12.1 Подробное описание

Заголовочный файл с описанием API для работы с областью памяти внешнего ПЗУ

### 4.12.2 Макросы

#### 4.12.2.1 BATCH

```
#define BATCH 1
```

Партия (значение для инициализации по умолчанию)

#### 4.12.2.2 DEV\_INFO

```
#define DEV_INFO "МПА"
```

Текстовая информация о модуле (значение для инициализации по умолчанию)

## 4.12.2.3 DEV\_TYPE

```
#define DEV_TYPE MPA
```

Тип устройства (значение для инициализации по умолчанию)

## 4.12.2.4 DEV\_TYPE\_RESERV

```
#define DEV_TYPE_RESERV 20
```

Резерв (значение для инициализации по умолчанию)

## 4.12.2.5 DUAL\_CONTROL

```
#define DUAL_CONTROL 56797
```

Реакция при одновременном управлении (значение для инициализации по умолчанию)

## 4.12.2.6 EXT\_ROM\_START\_ADDR

```
#define EXT_ROM_START_ADDR 0x00100000
```

Адрес в памяти МК, с которой начинается обращение к внешней ПЗУ

## 4.12.2.7 HWREG

```
#define HWREG(  
    x ) ((( unsigned char *)(x)))
```

Макрос чтения байта по адресу

## 4.12.2.8 MAX\_CODE\_ADC

```
#define MAX_CODE_ADC 65535
```

Максимальный код АЦП (значение для инициализации по умолчанию)

#### 4.12.2.9 METROLOG\_DAT

```
#define METROLOG_DAT 0.0f
```

Сведения о метрологии (значение для инициализации по умолчанию)

#### 4.12.2.10 MIN\_CODE\_ADC

```
#define MIN_CODE_ADC 0
```

Минимальный код АЦП (значение для инициализации по умолчанию)

#### 4.12.2.11 MODIFICATION

```
#define MODIFICATION 2
```

Модификация модуля (значение для инициализации по умолчанию)

#### 4.12.2.12 NUM\_CRC\_ERRORS\_FOR\_DEFECT\_B1

```
#define NUM_CRC_ERRORS_FOR_DEFECT_B1 6
```

Количество ошибок приема пакета до неисправности шины 1 (значение для инициализации по умолчанию)

#### 4.12.2.13 NUM\_CRC\_ERRORS\_FOR\_DEFECT\_B2

```
#define NUM_CRC_ERRORS_FOR_DEFECT_B2 6
```

Количество ошибок приема пакета до неисправности шины 2 (значение для инициализации по умолчанию)

#### 4.12.2.14 NUM\_FOR\_AVERAGE

```
#define NUM_FOR_AVERAGE 10
```

Кол-во выборок для усреднения (значение для инициализации по умолчанию)

## 4.12.2.15 REVISION

```
#define REVISION 1
```

Ревизия модуля (значение для инициализации по умолчанию)

## 4.12.2.16 ROM\_REGISTER\_SPACE\_START\_ADDR

```
#define ROM_REGISTER_SPACE_START_ADDR 0
```

Стартовый адрес карты регистров в ПЗУ

## 4.12.2.17 SERIAL\_NUMBER

```
#define SERIAL_NUMBER 1717986918
```

Серийный номер устройства (значение для инициализации по умолчанию)

## 4.12.2.18 TIME\_SOLO\_WORK

```
#define TIME_SOLO_WORK 61166
```

Время работы без резервирующего УМ, с (значение для инициализации по умолчанию)

## 4.12.2.19 TIME\_TO\_REPAIR

```
#define TIME_TO_REPAIR 65535
```

Максимально время переключения на резервный УМ, x10мс (значение для инициализации по умолчанию)

## 4.12.2.20 TIMEOUT\_FOR\_DEFECT\_B1

```
#define TIMEOUT_FOR_DEFECT_B1 200
```

Время без связи до неиспр шины 1, мс (значение для инициализации по умолчанию)

#### 4.12.2.21 TIMEOUT\_FOR\_DEFECT\_B2

```
#define TIMEOUT_FOR_DEFECT_B2 200
```

Время без связи до неиспр шины 2, мс (значение для инициализации по умолчанию)

### 4.12.3 Типы

#### 4.12.3.1 ai\_oper\_mode

```
typedef struct ai_oper_mode_struct ai_oper_mode
```

Структура с битовыми полями для регистра режим работы канала

#### 4.12.3.2 common\_rom\_registers

```
typedef struct common_register_space_ext_rom common_rom_registers
```

Организация пространства общих регистров для зеркализации во внешнем ПЗУ

#### 4.12.3.3 device\_type

```
typedef struct device_type_struct device_type
```

Структура с битовыми полями для регистра тип устройства

#### 4.12.3.4 mpa\_rom\_registers

```
typedef struct mpa_register_space_ext_rom mpa_rom_registers
```

Организация пространства регистров МПА для зеркализации во внешнем ПЗУ

#### 4.12.3.5 rom\_data

```
typedef struct rom_data_struct rom_data
```

Структура организующая память во внешнем ПЗУ

## 4.12.4 Функции

### 4.12.4.1 erase\_rom()

```
void erase_rom (
    void )
```

Очищает память в ПЗУ

Функция очистки памяти ПЗУ

```
142 {
143     uint8_t status;
144
145     // Команда Reset
146     HWREG(EXT_ROM_START_ADDR) = 0xF0;
147
148     // Конфигурирование выводов EBC на выход
149     MDR_PORTA->OE = 0x0000FFFF;
150
151     // Отправка соответствующей команды
152     HWREG(EXT_ROM_START_ADDR + 0x555) = 0xAA;
153     HWREG(EXT_ROM_START_ADDR + 0x2AA) = 0x55;
154     HWREG(EXT_ROM_START_ADDR + 0x555) = 0x80;
155     HWREG(EXT_ROM_START_ADDR + 0x555) = 0xAA;
156     HWREG(EXT_ROM_START_ADDR + 0x2AA) = 0x55;
157     HWREG(EXT_ROM_START_ADDR + 0x555) = 0x10;
158
159     // Конфигурирование выводов EBC на вход
160     MDR_PORTA->OE = 0x00000000;
161
162     do status = HWREG(EXT_ROM_START_ADDR);
163     while ((status & 0x80) != 0x80);
164 }
```

### 4.12.4.2 init\_external\_rom\_space()

```
void init_external_rom_space (
    void )
```

Инициализирует область памяти внешнего ПЗУ

Функция инициализации области памяти внешнего ПЗУ

```
23 {
24     #ifdef FIRST_TIME_INIT
25         common_rom_registers    common_regs;
26         mpa_rom_registers       mpa_regs;
27
28         // Очистка ПЗУ
29         erase_rom();
30
31         // Инициализация общих регистров
32         strncpy(&common_regs.PLC_DeviceInfo, DEV_INFO, sizeof(DEV_INFO));
33         common_regs.PLC_DeviceType.revision = REVISION;
34         common_regs.PLC_DeviceType.modification = MODIFICATION;
35         common_regs.PLC_DeviceType.type = DEV_TYPE;
36         common_regs.PLC_DeviceType.batch = BATCH;
37         common_regs.PLC_DeviceType.reserv = DEV_TYPE_RESERV;
38         common_regs.PLC_SerialNumber = SERIAL_NUMBER;
39         common_regs.PLC_TimeoutForDefect_B1 = TIMEOUT_FOR_DEFECT_B1;
40         common_regs.PLC_TimeoutForDefect_B2 = TIMEOUT_FOR_DEFECT_B2;
41         common_regs.PLC_NumCrcErrorsForDefect_B1 = NUM_CRC_ERRORS_FOR_DEFECT_B1;
42         common_regs.PLC_NumCrcErrorsForDefect_B2 = NUM_CRC_ERRORS_FOR_DEFECT_B2;
43         common_regs.PLC_TimeToRepair = TIME_TO_REPAIR;
44         common_regs.PLC_TimeSoloWork = TIME_SOLO_WORK;
45         common_regs.PLC_DualControl = DUAL_CONTROL;
46         memset(&common_regs.Reserv_2, 0, sizeof(common_regs.Reserv_2));
47     }
```

```

48 // Инициализация регистров МПА
49 for (uint8_t i = 0; i < MAX_CHANEL_NUMBER; i++)
50 {
51     RESET_BIT(i, mpa_regs.AI_OperMode.adc_chs_mode);
52     mpa_regs.AI_NumForAverag[i] = NUM_FOR_AVERAGE;
53     mpa_regs.AI_MinCodeADC[i] = MIN_CODE_ADC;
54     mpa_regs.AI_MaxCodeADC[i] = MAX_CODE_ADC;
55     // Такие значения коэф. полиномов только для напряжения 0-10В
56     mpa_regs.AI_PolynConst0[i] = polyn_ch_consts[i][0];
57     mpa_regs.AI_PolynConst1[i] = polyn_ch_consts[i][1];
58     mpa_regs.AI_PolynConst2[i] = polyn_ch_consts[i][2];
59     mpa_regs.AI_PolynConst3[i] = polyn_ch_consts[i][3];
60     mpa_regs.AI_PolynConst4[i] = polyn_ch_consts[i][4];
61     mpa_regs.AI_PolynConst5[i] = polyn_ch_consts[i][5];
62     mpa_regs.AI_PolynConst6[i] = polyn_ch_consts[i][6];
63     mpa_regs.AI_MetrologDat[i] = METROLOG_DAT;
64 }
65 memset(mpa_regs.AI_MetrologDat, 0, sizeof(mpa_regs.AI_MetrologDat));
66 memset(mpa_regs.Reserv_4, 0, sizeof(mpa_regs.Reserv_4));
67 memset(mpa_regs.Reserv_5, 0, sizeof(mpa_regs.Reserv_5));
68
69 // Копирование данных в ПЗУ
70 memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR, &common_regs, sizeof(common_regs));
71 memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR + sizeof(common_regs), &mpa_regs,
72 sizeof(mpa_regs));
73 #endif
74 }

```

#### 4.12.4.3 memcpy\_to\_rom()

```

void memcpy_to_rom (
    uint32_t dest_addr,
    void * src_addr,
    uint32_t size )

```

Копирует область памяти в ПЗУ

Аргументы

dest_addr	- адрес назначения
src_addr	- адрес источника
size	- размер данных

Функция копирования области памяти в ПЗУ

```

131 {
132     for (uint32_t i = 0; i < size; i++)
133     {
134         write_byte_rom(dest_addr + i, *((uint8_t*)src_addr+i));
135     }
136 }

```

#### 4.12.4.4 read\_byte\_rom()

```

uint8_t read_byte_rom (
    uint32_t dest_addr )

```

Читает байт из ПЗУ



Аргументы

dest_addr	- адрес чтения
-----------	----------------

Возвращает

Байт

Функция чтения байта из ПЗУ

```

121 {
122     // Конфигурирование выводов ЕВС на вход
123     MDR_PORTA->OE = 0x00000000;
124     return (HWREG(dest_addr));
125 }
```

#### 4.12.4.5 write\_byte\_rom()

```

uint8_t write_byte_rom (
    uint32_t dest_addr,
    uint8_t byte )
```

Записывает байт в ПЗУ

Аргументы

dest_addr	- адрес назначения
byte	- записываемый байт

Функция записи байта в ПЗУ

```

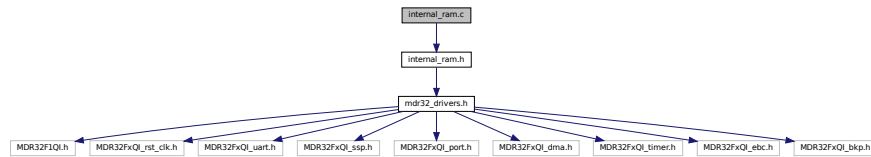
80 {
81     uint8_t status;
82
83     // Конфигурирование выводов ЕВС на выход
84     MDR_PORTA->OE = 0x0000FFFF;
85
86     HWREG(EXT_ROM_START_ADDR + 0x555) = 0xAA;
87     HWREG(EXT_ROM_START_ADDR + 0x2AA) = 0x55;
88     HWREG(EXT_ROM_START_ADDR + 0x555) = 0xA0;
89     HWREG(dest_addr + EXT_ROM_START_ADDR) = byte;
90
91     // Конфигурирование выводов ЕВС на вход
92     MDR_PORTA->OE = 0x00000000;
93
94     // Теперь проверяем, выполнялась ли запись успешно (успех = бит D7 == data[7])
95     do status = HWREG(dest_addr + EXT_ROM_START_ADDR);
96     while(((status & 0x80) != (byte & 0x80)) & ((status & 0x20) == 0));
97
98     if((status & 0x20) != 0)
99     {
100         status = HWREG(dest_addr + EXT_ROM_START_ADDR);
101         if((status & 0x80) == (byte & 0x80)) // Успех
102         {
103             return 0;
104         }
105         else
106         {
107             return 1; // Ошибка записи
108         }
109     }
110     else if((status & 0x80) == (byte & 0x80)) // Успех
111     {
112         return 0;
113     }
114     return 2;
115 }
```

## 4.13 Файл internal\_ram.c

Файл с реализацией API для работы с областью памяти внутреннего ОЗУ

```
#include "internal_ram.h"
```

Граф включаемых заголовочных файлов для internal\_ram.c:



### Функции

- `uint8_t * malloc_ram_pages (uint32_t size)`  
Выделяет свободную память во внутреннем ОЗУ (страницы памяти) - необходима для реализации кучи
- `void free_ram_pages (uint8_t *page_addr, uint32_t size)`  
Освобождает память во внутреннем ОЗУ (страницы памяти) - необходима для реализации кучи

### Переменные

- `int_ram_heap * heap_ptr`  
Указатель на "самодельную" кучу

#### 4.13.1 Подробное описание

Файл с реализацией API для работы с областью памяти внутреннего ОЗУ

#### 4.13.2 Функции

##### 4.13.2.1 free\_ram\_pages()

```
void free_ram_pages (
    uint8_t * page_addr,
    uint32_t size )
```

Освобождает память во внутреннем ОЗУ (страницы памяти) - необходима для реализации кучи

Функция освобождения памяти во внешнем ОЗУ (страниц памяти)

```

49 {
50     uint32_t page_num = size/PAGE_SIZE + 1; // Кол-во освобождаемых страниц
51     uint32_t page_cnt; // Счетчик страниц
52
53     for (page_cnt = 0; (page_cnt + page_num) < PAGE_SIZE; page_cnt++)
54     {
55         if (page_addr == &heap_ptr->memory_page_space[page_cnt][PAGE_SIZE])
56         {
57             // Обновляем статус страниц
58             for (uint32_t i = 0; i < page_num; i++)
59             {
60                 memset(&heap_ptr->memory_page_status[page_cnt + i], 0, sizeof(heap_ptr->memory_page_status[page_cnt
61                 + i]));
62             }
63         }
64     }
```

## 4.13.2.2 malloc\_ram\_pages()

```
uint8_t* malloc_ram_pages (
    uint32_t size )
```

Выделяет свободную память во внутреннем ОЗУ (страницы памяти) - необходима для реализации кучи

Функция выделения свободной памяти во внешнем ОЗУ (страниц памяти)

```
15 {
16     uint32_t page_num = size/PAGE_SIZE + 1; // Кол-во выделяемых страниц
17     uint32_t page_cnt; // Счетчик страниц
18     uint32_t consecutive_page_cnt; // Счетчик подряд идущих пустых страниц
19
20     for (page_cnt = 0; (page_cnt + page_num) < PAGE_SIZE; page_cnt++)
21     {
22         // Проверяем, что подряд идут пустые страницы памяти
23         for (consecutive_page_cnt = 0; consecutive_page_cnt < page_num; consecutive_page_cnt++)
24         {
25             if (heap_ptr->memory_page_status[page_cnt + consecutive_page_cnt] != 0)
26             {
27                 page_cnt += consecutive_page_cnt;
28                 break;
29             }
30         }
31         // Если удалось найти требуемое кол-во подряд идущих пустых страниц, то возвращаем указатель
32         if (consecutive_page_cnt == page_num)
33         {
34             // Обновляем статус страниц
35             for (consecutive_page_cnt = 0; consecutive_page_cnt < page_num; consecutive_page_cnt++)
36             {
37                 memset(&heap_ptr->memory_page_status[page_cnt + consecutive_page_cnt], 1, sizeof(heap_ptr-
38 >memory_page_status[page_cnt + consecutive_page_cnt]));
39             }
40             return &heap_ptr->memory_page_space[page_cnt][0];
41         }
42     }
43     return 0;
44 }
```

## 4.13.3 Переменные

## 4.13.3.1 heap\_ptr

```
int_ram_heap* heap_ptr [extern]
```

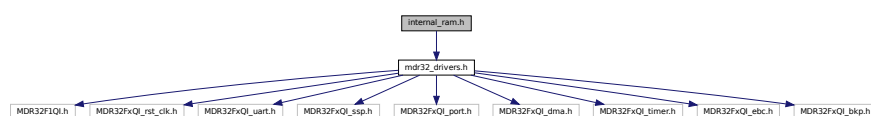
Указатель на "самодельную" кучу

## 4.14 Файл internal\_ram.h

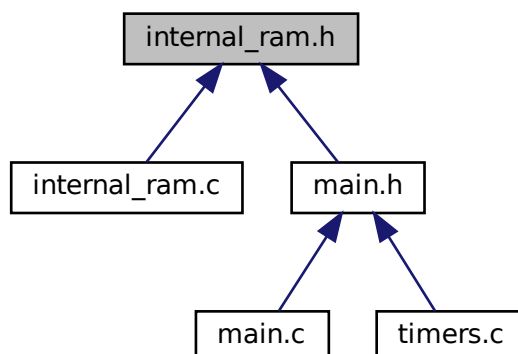
Заголовочный файл с описанием API для работы с областью памяти внутреннего ОЗУ

```
#include "mdr32_drivers.h"
```

Граф включаемых заголовочных файлов для internal\_ram.h:



Граф файлов, в которые включается этот файл:



## Классы

- struct `heap_struct`  
Реализация "самодельной" кучи

## Макросы

- `#define PAGE_NUM 256`  
Кол-во страниц памяти для кучи
- `#define PAGE_SIZE 64`  
Размер страницы памяти для кучи

## Определения типов

- `typedef struct heap_struct int_ram_heap`  
Реализация "самодельной" кучи

## Функции

- `uint8_t * malloc_ram_pages (uint32_t size)`  
Выделяет свободную память во внутреннем ОЗУ (страницы памяти) - необходима для реализации кучи
- `void free_ram_pages (uint8_t *page_addr, uint32_t size)`  
Освобождает память во внутреннем ОЗУ (страницы памяти) - необходима для реализации кучи

### 4.14.1 Подробное описание

Заголовочный файл с описанием API для работы с областью памяти внутреннего ОЗУ

### 4.14.2 Макросы

#### 4.14.2.1 PAGE\_NUM

```
#define PAGE_NUM 256
```

Кол-во страниц памяти для кучи

#### 4.14.2.2 PAGE\_SIZE

```
#define PAGE_SIZE 64
```

Размер страницы памяти для кучи

### 4.14.3 Типы

#### 4.14.3.1 int\_ram\_heap

```
typedef struct heap_struct int_ram_heap
```

Реализация "самодельной" кучи

### 4.14.4 Функции

#### 4.14.4.1 free\_ram\_pages()

```
void free_ram_pages (
    uint8_t * page_addr,
    uint32_t size )
```

Освобождает память во внутреннем ОЗУ (страницы памяти) - необходима для реализации кучи

Аргументы

page_addr	- адрес стартовой страницы
size	- размер освобождаемой памяти

Функция освобождения памяти во внешнем ОЗУ (страниц памяти)

```

49 {
50     uint32_t page_num = size/PAGE_SIZE + 1; // Кол-во освобождаемых страниц
51     uint32_t page_cnt; // Счетчик страниц
52
53     for (page_cnt = 0; (page_cnt + page_num) < PAGE_SIZE; page_cnt++)
54     {
55         if (page_addr == &heap_ptr->memory_page_space[page_cnt][PAGE_SIZE])
56         {
57             // Обновляем статус страниц
58             for (uint32_t i = 0; i < page_num; i++)
59             {
60                 memset(&heap_ptr->memory_page_status[page_cnt + i], 0, sizeof(heap_ptr->memory_page_status[page_cnt
61                     + i]));
62             }
63         }
64     }

```

#### 4.14.4.2 malloc\_ram\_pages()

```

uint8_t* malloc_ram_pages (
    uint32_t size )

```

Выделяет свободную память во внутреннем ОЗУ (страницы памяти) - необходима для реализации кучи

Аргументы

size	- размер выделяемой памяти
------	----------------------------

Возвращает

Указатель на начало выделяемой памяти или 0, если нет свободного места

Функция выделения свободной памяти во внешнем ОЗУ (страниц памяти)

```

15 {
16     uint32_t page_num = size/PAGE_SIZE + 1; // Кол-во выделяемых страниц
17     uint32_t page_cnt; // Счетчик страниц
18     uint32_t consecutive_page_cnt; // Счетчик подряд идущих пустых страниц
19
20     for (page_cnt = 0; (page_cnt + page_num) < PAGE_SIZE; page_cnt++)
21     {
22         // Проверяем, что подряд идут пустые страницы памяти
23         for (consecutive_page_cnt = 0; consecutive_page_cnt < page_num; consecutive_page_cnt++)
24         {
25             if (heap_ptr->memory_page_status[page_cnt + consecutive_page_cnt] != 0)
26             {
27                 page_cnt += consecutive_page_cnt;
28                 break;
29             }
30         }
31         // Если удалось найти требуемое кол-во подряд идущих пустых страницы, то возвращаем указатель
32         if (consecutive_page_cnt == page_num)
33         {
34             // Обновляем статус страниц
35             for (consecutive_page_cnt = 0; consecutive_page_cnt < page_num; consecutive_page_cnt++)
36             {
37                 memset(&heap_ptr->memory_page_status[page_cnt + consecutive_page_cnt], 1, sizeof(heap_ptr-
38                     >memory_page_status[page_cnt + consecutive_page_cnt]));
39             }
40             return &heap_ptr->memory_page_space[page_cnt][0];
41         }
42     }
43     return 0;
44 }

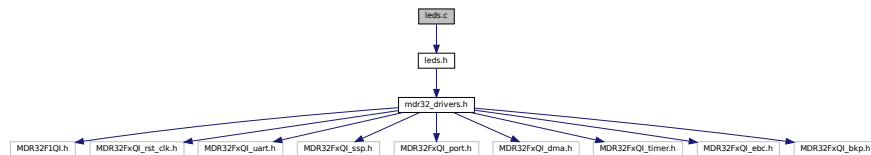
```

## 4.15 Файл leds.c

Файл с реализацией API для работы со светодиодными индикаторами

```
#include "leds.h"
```

Граф включаемых заголовочных файлов для leds.c:



### Функции

- void `leds_gpio_config` (void)

Конфигурирует выводы МК для светодиодных индикаторов

#### 4.15.1 Подробное описание

Файл с реализацией API для работы со светодиодными индикаторами

#### 4.15.2 Функции

##### 4.15.2.1 leds\_gpio\_config()

```
void leds_gpio_config (
    void )
```

Конфигурирует выводы МК для светодиодных индикаторов

```

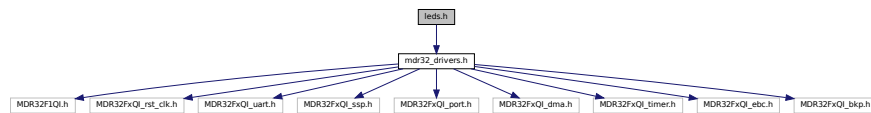
12 {
13     // Включение тактирования портов
14     RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK|CLOCK_LEDS, ENABLE);
15
16     PORT_InitTypeDef gpio_init_struct_leds;
17     PORT_StructInit(&gpio_init_struct_leds);
18
19     // Инициализация светодиода-индикатора корректной работы
20     gpio_init_struct_leds.PORT_Pin = PIN_LED_OK_WORK;
21     gpio_init_struct_leds.PORT_FUNC = PORT_FUNC_PORT;
22     gpio_init_struct_leds.PORT_OE = PORT_OE_OUT;
23     gpio_init_struct_leds.PORT_MODE = PORT_MODE_DIGITAL;
24     gpio_init_struct_leds.PORT_SPEED = PORT_SPEED_MAXFAST;
25     PORT_Init(PORT_LEDS, &gpio_init_struct_leds);
26
27     // Инициализация светодиода-индикатора неисправности
28     gpio_init_struct_leds.PORT_Pin = PIN_LED_ERROR_WORK;
29     gpio_init_struct_leds.PORT_FUNC = PORT_FUNC_PORT;
30     gpio_init_struct_leds.PORT_OE = PORT_OE_OUT;
31     gpio_init_struct_leds.PORT_MODE = PORT_MODE_DIGITAL;
32     gpio_init_struct_leds.PORT_SPEED = PORT_SPEED_MAXFAST;
33     PORT_Init(PORT_LEDS, &gpio_init_struct_leds);
34 }
```

## 4.16 Файл leds.h

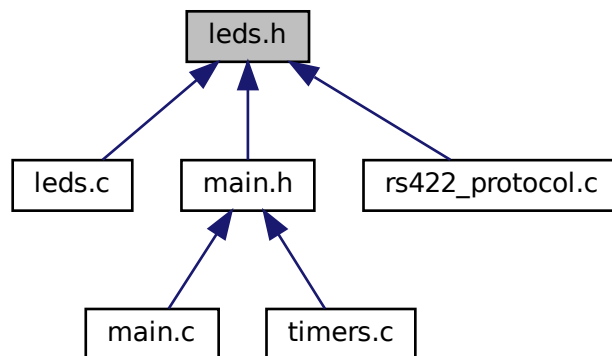
Заголовочный файл с описанием API для работы со светодиодными индикаторами

```
#include "mdr32_drivers.h"
```

Граф включаемых заголовочных файлов для leds.h:



Граф файлов, в которые включается этот файл:



## Макросы

- `#define PORT_LEDS MDR_PORTD`  
Порт выводов на светодиоды
- `#define CLOCK_LEDS RST_CLK_PCLK_PORTD`  
Выбор тактирования выводов для светодиодов
- `#define PIN_LED_OK_WORK PORT_Pin_7`  
Светодиод-индикатор корректной работы
- `#define PIN_LED_ERROR_WORK PORT_Pin_8`  
Светодиод-индикатор неисправности
- `#define SET_LED_OK_WORK() {PORT_LEDS->SETTX = PIN_LED_OK_WORK;}`  
Макрос включения светодиода корректной работы
- `#define RESET_LED_OK_WORK() {PORT_LEDS->CLR_TX = PIN_LED_OK_WORK;}`  
Макрос выключения светодиода корректной работы
- `#define SET_LED_ERROR_WORK() {PORT_LEDS->SETTX = PIN_LED_ERROR_WORK;}`  
Макрос включения светодиода некорректной работы
- `#define RESET_LED_ERROR_WORK() {PORT_LEDS->CLR_TX = PIN_LED_ERROR_WORK;}`  
Макрос выключения светодиода некорректной работы



## Функции

- void `leds_gpio_config` (void)  
Конфигурирует выводы МК для светодиодных индикаторов

### 4.16.1 Подробное описание

Заголовочный файл с описанием API для работы со светодиодными индикаторами

### 4.16.2 Макросы

#### 4.16.2.1 CLOCK\_LEDS

```
#define CLOCK_LEDS RST_CLK_PCLK_PORTD
```

Выбор тактирования выводов для светодиодов

#### 4.16.2.2 PIN\_LED\_ERROR\_WORK

```
#define PIN_LED_ERROR_WORK PORT_Pin_8
```

Светодиод-индикатор неисправности

#### 4.16.2.3 PIN\_LED\_OK\_WORK

```
#define PIN_LED_OK_WORK PORT_Pin_7
```

Светодиод-индикатор корректной работы

#### 4.16.2.4 PORT\_LEDS

```
#define PORT_LEDS MDR_PORTD
```

Порт выводов на светодиоды

#### 4.16.2.5 RESET\_LED\_ERROR\_WORK

```
#define RESET_LED_ERROR_WORK( ) {PORT_LEDS->CLRTX = PIN_LED_ERROR_WORK;}
```

Макрос выключения светодиода некорректной работы

#### 4.16.2.6 RESET\_LED\_OK\_WORK

```
#define RESET_LED_OK_WORK( ) {PORT_LEDS->CLRTX = PIN_LED_OK_WORK;}
```

Макрос выключения светодиода корректной работы

#### 4.16.2.7 SET\_LED\_ERROR\_WORK

```
#define SET_LED_ERROR_WORK( ) {PORT_LEDS->SETTX = PIN_LED_ERROR_WORK;}
```

Макрос включения светодиода некорректной работы

#### 4.16.2.8 SET\_LED\_OK\_WORK

```
#define SET_LED_OK_WORK( ) {PORT_LEDS->SETTX = PIN_LED_OK_WORK;}
```

Макрос включения светодиода корректной работы

### 4.16.3 Функции

#### 4.16.3.1 leds\_gpio\_config()

```
void leds_gpio_config (
    void )
```

Конфигурирует выводы МК для светодиодных индикаторов

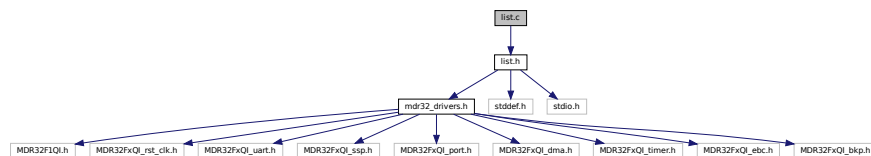
```
12 {
13     // Включение тактирования портов
14     RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK|CLOCK_LEDS, ENABLE);
15
16     PORT_InitTypeDef gpio_init_struct_leds;
17     PORT_StructInit(&gpio_init_struct_leds);
18
19     // Инициализация светодиода-индикатора корректной работы
20     gpio_init_struct_leds.PORT_Pin = PIN_LED_OK_WORK;
21     gpio_init_struct_leds.PORT_FUNC = PORT_FUNC_PORT;
22     gpio_init_struct_leds.PORT_OE = PORT_OE_OUT;
23     gpio_init_struct_leds.PORT_MODE = PORT_MODE_DIGITAL;
24     gpio_init_struct_leds.PORT_SPEED = PORT_SPEED_MAXFAST;
25     PORT_Init(PORT_LEDS, &gpio_init_struct_leds);
26
27     // Инициализация светодиода-индикатора неисправности
28     gpio_init_struct_leds.PORT_Pin = PIN_LED_ERROR_WORK;
29     gpio_init_struct_leds.PORT_FUNC = PORT_FUNC_PORT;
30     gpio_init_struct_leds.PORT_OE = PORT_OE_OUT;
31     gpio_init_struct_leds.PORT_MODE = PORT_MODE_DIGITAL;
32     gpio_init_struct_leds.PORT_SPEED = PORT_SPEED_MAXFAST;
33     PORT_Init(PORT_LEDS, &gpio_init_struct_leds);
34 }
```

## 4.17 Файл list.c

Файл с реализацией API для работы со списками (реализация аналогична linux kernel)

```
#include "list.h"
```

Граф включаемых заголовочных файлов для list.c:



### Функции

- void `init_list_head` (`list_head *list`)  
Инициализирует двусвязанный список
- void `__list_add` (`list_head *new`, `list_head *prev`, `list_head *next`)  
Вставляет элемент в связанный список (между предыдущими и следующими элементами)
- void `list_add` (`list_head *new`, `list_head *head`)  
Вставляет элемент в заголовок связанного списка
- void `list_add_tail` (`list_head *new`, `list_head *head`)  
Вставляет элемент в конец связанного списка
- void `__list_del` (`list_head *prev`, `list_head *next`)  
Удаляет элемент списка
- void `list_del` (`list_head *entry`)  
Удаляет элемент списка и очищает его
- int `list_is_last` (`list_head *list`, `list_head *head`)  
Определяет, является ли текущий узел списка последним узлом в списке
- int `list_empty` (`list_head *head`)  
Определяет, является ли связанный список пустым

#### 4.17.1 Подробное описание

Файл с реализацией API для работы со списками (реализация аналогична linux kernel)

#### 4.17.2 Функции

## 4.17.2.1 \_\_list\_add()

```
void __list_add (
    list_head * new,
    list_head * prev,
    list_head * next )
```

Вставляет элемент в связанный список (между предыдущими и следующими элементами)

Функция вставки элемента в связанный список (между предыдущими и следующими элементами)

```
20 {
21     next->prev = new;
22     new->next = next;
23     new->prev = prev;
24     prev->next = new;
25 }
```

## 4.17.2.2 \_\_list\_del()

```
void __list_del (
    list_head * prev,
    list_head * next )
```

Удаляет элемент списка

Функция удаления элемента списка

```
44 {
45     next->prev = prev;
46     prev->next = next;
47 }
```

## 4.17.2.3 init\_list\_head()

```
void init_list_head (
    list_head * list )
```

Инициализирует двусвязанный список

Функция инициализации двусвязного списка

```
12 {
13     list->next = list;
14     list->prev = list;
15 }
```

## 4.17.2.4 list\_add()

```
void list_add (
    list_head * new,
    list_head * head )
```

Вставляет элемент в заголовок связанного списка

Функция вставки элемента сразу после head

```
30 {
31     __list_add(new, head, head->next);
32 }
```

## 4.17.2.5 list\_add\_tail()

```
void list_add_tail (
    list_head * new,
    list_head * head )
```

Вставляет элемент в конец связанного списка

Функция вставки элемента перед head (т.е. в хвост узла)

```
37 {
38     __list_add(new, head->prev, head);
39 }
```

## 4.17.2.6 list\_del()

```
void list_del (
    list_head * entry )
```

Удаляет элемент списка и очищает его

Функция удаления элемента списка и его очистки

```
52 {
53     __list_del(entry->prev, entry->next);
54     entry->next = NULL;
55     entry->prev = NULL;
56 }
```

## 4.17.2.7 list\_empty()

```
int list_empty (
    list_head * head )
```

Определяет, является ли связанный список пустым

Функция, определяющая, является ли связанный список пустым, возвращает true, если пуст, в противном случае возвращает false

```
68 {
69     return head->next == head;
70 }
```

## 4.17.2.8 list\_is\_last()

```
int list_is_last (
    list_head * list,
    list_head * head )
```

Определяет, является ли текущий узел списка последним узлом в списке

Функция, определяющая, является ли текущий узел списка последним узлом в списке

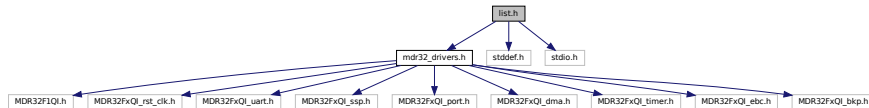
```
61 {
62     return list->next == head;
63 }
```

## 4.18 Файл list.h

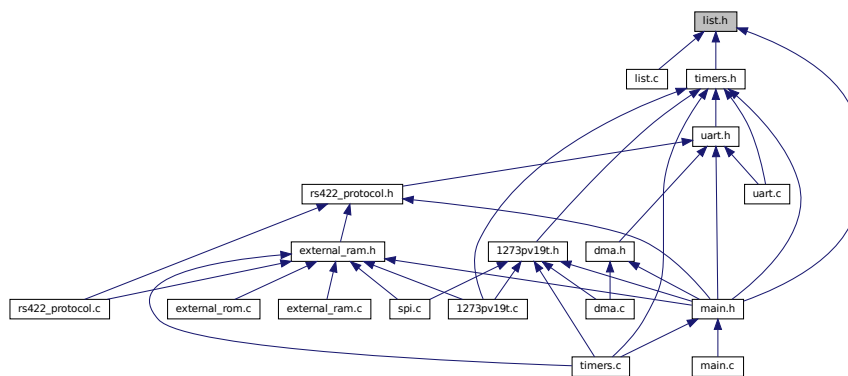
Заголовочный файл с описанием API для работы со списками (реализация аналогична linux kernel)

```
#include "mdr32_drivers.h"
#include <stddef.h>
#include <stdio.h>
```

Граф включаемых заголовочных файлов для list.h:



Граф файлов, в которые включается этот файл:



## Классы

- struct `list_head_struct`  
Структура с описанием двусвязного списка

## Макросы

- #define `list_for_each_entry`(pos, head, member)  
Макрос для обхода двусвязного списка (более оптимальный вариант)
- #define `list_for_each`(pos, head) for (pos = (head)->next; pos != (head); pos = pos->next)  
Макрос для обхода двусвязного списка (head - указывает на список, pos - курсор, используемый для обхода)
- #define `container_of`(ptr, type, member)  
Макрос определяет адрес структуры по известному адресу члена этой структуры (ptr-указатель на член структуры, type-тип структуры, member-имя члена структуры)
- #define `list_entry`(ptr, type, member) `container_of`(ptr, type, member)  
Макрос обертка container\_of.
- #define `LIST_HEAD_INIT`(name) { &(name), &(name) }  
Макрос инициализации двусвязного списка
- #define `LIST_HEAD`(name) `list_head` name = `LIST_HEAD_INIT`(name)  
Макрос создания двусвязного списка с именем name.

## Определения типов

- typedef struct list\_head\_struct list\_head  
Структура с описанием двусвязанного списка

## Функции

- void init\_list\_head (list\_head \*list)  
Инициализирует двусвязанный список
- void \_\_list\_add (list\_head \*new, list\_head \*prev, list\_head \*next)  
Вставляет элемент в связанный список (между предыдущими и следующими элементами)
- void list\_add (list\_head \*new, list\_head \*head)  
Вставляет элемент в заголовок связанного списка
- void list\_add\_tail (list\_head \*new, list\_head \*head)  
Вставляет элемент в конец связанного списка
- void \_\_list\_del (list\_head \*prev, list\_head \*next)  
Удаляет элемент списка
- void list\_del (list\_head \*entry)  
Удаляет элемент списка и очищает его
- int list\_is\_last (list\_head \*list, list\_head \*head)  
Определяет, является ли текущий узел списка последним узлом в списке
- int list\_empty (list\_head \*head)  
Определяет, является ли связанный список пустым

### 4.18.1 Подробное описание

Заголовочный файл с описанием API для работы со списками (реализация аналогична linux kernel)

### 4.18.2 Макросы

#### 4.18.2.1 container\_of

```
#define container_of(
    ptr,
    type,
    member )
```

Макроопределение:

```
{
    const typeof( ((type *)0)->member ) *__mptr = (ptr); \
    (type *) ( (char *)__mptr - offsetof(type,member) );}
```

Макрос определяет адрес структуры по известному адресу члена этой структуры (ptr-указатель на член структуры, type-тип структуры, member-имя члена структуры)

## 4.18.2.2 list\_entry

```
#define list_entry(
    ptr,
    type,
    member ) container_of(ptr, type, member)
```

Макрос обертка container\_of.

## 4.18.2.3 list\_for\_each

```
#define list_for_each(
    pos,
    head ) for (pos = (head)->next; pos != (head); pos = pos->next)
```

Макрос для обхода двусвязного списка (head - указывает на список, pos - курсор, используемый для обхода)

## 4.18.2.4 list\_for\_each\_entry

```
#define list_for_each_entry(
    pos,
    head,
    member )
```

Макроопределение:

```
for (pos = list_entry((head)->next, typeof(*pos), member); \
    &pos->member != (head); \
    pos = list_entry(pos->member.next, typeof(*pos), member))
```

Макрос для обхода двусвязного списка (более оптимальный вариант)

## 4.18.2.5 LIST\_HEAD

```
#define LIST_HEAD(
    name ) list_head name = LIST_HEAD_INIT(name)
```

Макрос создания двусвязного списка с именем name.

## 4.18.2.6 LIST\_HEAD\_INIT

```
#define LIST_HEAD_INIT(
    name ) { &(name), &(name) }
```

Макрос инициализации двусвязного списка



### 4.18.3 Типы

#### 4.18.3.1 list\_head

```
typedef struct list_head_struct list_head
```

Структура с описанием двусвязанного списка

### 4.18.4 Функции

#### 4.18.4.1 \_\_list\_add()

```
void __list_add (
    list_head * new,
    list_head * prev,
    list_head * next )
```

Вставляет элемент в связанный список (между предыдущими и следующими элементами)

Аргументы

new	- указатель на новый элемент списка
prev	- указатель на предыдущий элемент списка
next	- указатель на следующий элемент списка

Функция вставки элемента в связанный список (между предыдущими и следующими элементами)

```
20 {
21     next->prev = new;
22     new->next = next;
23     new->prev = prev;
24     prev->next = new;
25 }
```

#### 4.18.4.2 \_\_list\_del()

```
void __list_del (
    list_head * prev,
    list_head * next )
```

Удаляет элемент списка

Аргументы

prev	- указатель на предыдущий элемент
next	- указатель на следующий элемент

Функция удаления элемента списка

```
44 {  
45     next->prev = prev;  
46     prev->next = next;  
47 }
```

#### 4.18.4.3 init\_list\_head()

```
void init_list_head (  
    list_head * list )
```

Инициализирует двусвязанный список

Аргументы

list	- указатель на заголовок списка
------	---------------------------------

Функция инициализации двусвязного списка

```
12 {  
13     list->next = list;  
14     list->prev = list;  
15 }
```

#### 4.18.4.4 list\_add()

```
void list_add (  
    list_head * new,  
    list_head * head )
```

Вставляет элемент в заголовок связанного списка

Аргументы

new	- указатель на новый элемент списка
head	- указатель на заголовок списка

Функция вставки элемента сразу после head

```
30 {  
31     __list_add(new, head, head->next);  
32 }
```

#### 4.18.4.5 list\_add\_tail()

```
void list_add_tail (  
    list_head * new,  
    list_head * head )
```

Вставляет элемент в конец связанного списка

## Аргументы

new	- указатель на новый элемент списка
head	- указатель на заголовок списка

Функция вставки элемента перед head (т.е. в хвост узла)

```
37 {  
38     __list_add(new, head->prev, head);  
39 }
```

## 4.18.4.6 list\_del()

```
void list_del (  
    list_head * entry )
```

Удаляет элемент списка и очищает его

## Аргументы

entry	- указатель на удаляемый элемент
-------	----------------------------------

Функция удаления элемента списка и его очистки

```
52 {  
53     __list_del(entry->prev, entry->next);  
54     entry->next = NULL;  
55     entry->prev = NULL;  
56 }
```

## 4.18.4.7 list\_empty()

```
int list_empty (  
    list_head * head )
```

Определяет, является ли связанный список пустым

## Аргументы

head	- указатель на заголовок списка
------	---------------------------------

Возвращает

Сообщение с результатом (1 - список пуст, 0 - не пуст)

Функция, определяющая, является ли связанный список пустым, возвращает true, если пуст, в противном случае возвращает false

```
68 {  
69     return head->next == head;  
70 }
```

## 4.18.4.8 list\_is\_last()

```
int list_is_last (
    list_head * list,
    list_head * head )
```

Определяет, является ли текущий узел списка последним узлом в списке

Аргументы

list	- текущий элемент списка
head	- указатель на заголовок списка

Возвращает

Сообщение с результатом (1 - является, 0 - не является)

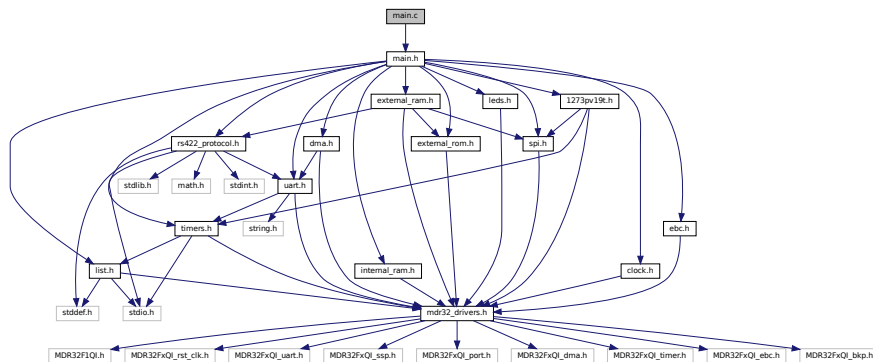
Функция, определяющая, является ли текущий узел списка последним узлом в списке

```
61 {
62     return list->next == head;
63 }
```

## 4.19 Файл main.c

```
#include "main.h"
```

Граф включаемых заголовочных файлов для main.c:



## Функции

- int `main` (void)
- uint8\_t `request_data` (uart\_n \*uart\_struct)  
Запрашивает данные на выбранной шине
- void `do_mpa_task` (adc\_n \*adc\_struct)  
Выполняет периферийные задачи МПА
- void `sync_adc_chanel` (void \*data)  
Синхронизирует каналы АЦП (выполняется при срабатывании прерывания Timer2 по переполнению счетчика CNT)
- void `receive_adc_chanel_pack` (void \*data)  
Принимает пакет с результатами измерений одного канала (выполняется при срабатывании прерывания Timer2 по захвату)

## Переменные

- `adc_n adc_1`  
Структура с конфигурационными параметрами микросхемы АЦП
- `timer_n timer_1`  
Структуры с конфигурационными параметрами блоков таймеров МК
- `timer_n timer_2`  
Структура с конфигурационными параметрами блока Timer2 МК
- `timer_n timer_3`  
Структура с конфигурационными параметрами блока Timer3 МК
- `list_head * tmr_handler_head [TIMER_NUM]`  
Массив указателей на head списков обработчиков прерываний таймеров
- `spi_n spi_1`  
Структуры с конфигурационными параметрами блоков SPI МК
- `spi_n spi_2`  
Структура с конфигурационными параметрами блока SPI2 МК
- `uart_n uart_1`  
Структуры с конфигурационными параметрами блоков UART МК
- `uart_n uart_2`  
Структура с конфигурационными параметрами блока UART2 МК
- `ram_data * ram_space_pointer`  
Указатель для обращения к внешнему ОЗУ
- `rom_data * rom_space_pointer`  
Указатель для обращения к внешнему ПЗУ
- `int_ram_heap heap`  
Выделение памяти во внутреннем ОЗУ МК для "самодельной" кучи
- `int_ram_heap * heap_ptr = &heap`  
Указатель на "самодельную" кучу

### 4.19.1 Функции

#### 4.19.1.1 do\_mpa\_task()

```
void do_mpa_task (
    adc_n * adc_struct )
```

Выполняет периферийные задачи МПА

Аргументы

<code>*adc_struct</code>	- Структура с конфигурационными параметрами микросхемы АЦП
--------------------------	--

```
197 {
198     // Темповые переменные, необходимые для обработки данных АЦП
199     int adc_code[MAX_CHANEL_NUMBER] = {0};
200     int16_t adc_value;
201
202     // TODO: на текущий момент данная ф-ция обрабатывает данные каналов МПА только для напряжений 0-10В
203     // (для тока в карту регистров надо добавлять свои полиномы, т.к. они отличаются)
204     // Указатель на пространство регистров МПА
```

```

205 mpa_ram_registers *ptr = &ram_space_pointer->mpa_ram_register_space;
206
207 // Ниже приведены аппроксимирующие полиномы, полученные экспериментально для всех режимов работы
    отладочной платы АЦП
208 /*
209     для двуполярного вх напр (-5.4 ... 5.4 В ) на мультиплексоре A0=A1=0
210     U = 1.6474f*pow(10,-4)*adc_code;
211     delta = 6.5627f*pow(10,-6)*adc_code + 0.00039f;
212     для однополярного вх напр (0 ... 10.8 В ) на мультиплексоре A0=1;A1=0
213     U = 1.6474f*pow(10,-4)*adc_code + 5.398f;
214     delta = 6.6962f*pow(10,-6)*adc_code + 0.4252307f;
215     для однополярного вх тока (0 ... 21.6 мА ) на мультиплексоре A0=1;A1=0
216     I = 3.052f*pow(10,-4)*adc_code + 10.0f;
217     delta = -11.9006f*pow(10,-6)*adc_code + 0.03072506f;
218     самодиагностика для двуполярного случая на мультиплексоре A0=1;A1=1 (на выходе должно быть 0В)
219     U = 1.6474f*pow(10,-4)*adc_code;
220     delta = 6.5627f*pow(10,-6)*adc_code + 0.00039f;
221     самодиагностика для однополярного случая на мультиплексоре A0=0;A1=1 (на выходе должно быть 0В)
222     U = 1.6474f*pow(10,-4)*adc_code + 5.398f;
223     delta = 6.6962f*pow(10,-6)*adc_code + 0.4252307f;
224
225     Результирующее напряжение после аппроксимации U = U - delta;
226 */
227
228 // Производим усреднение по максим кол-ву выборок, если кол-во выборок различается для разных каналов МПА
229 adc_1.avg_num = find_max_halfword(ptr->AI_RomRegs.AI_NumForAverag, CHANNEL_NUMBER);
230
231 for (uint8_t k = 0; k < CHANNEL_NUMBER; k++)
232 {
233     for (uint8_t i = 0; i < ptr->AI_RomRegs.AI_NumForAverag[k]; i++)
234     {
235         memcpy(&adc_value, adc_struct->spi_struct->buffer + (i*CHANNEL_NUMBER) + k, sizeof(adc_value));
236         adc_code[k] += (int16_t)(^adc_value + 1);
237     }
238     adc_code[k] /= ptr->AI_RomRegs.AI_NumForAverag[k];
239     // Кладем в соответствующий регистр МПА полученные код АЦП для текущего канала МПА
240     memcpy(&(ptr->AI_CodeADC[k]), &adc_code[k], sizeof(adc_code[k]));
241     // В зависимости от режима работы канала МПА (ток/напряжение) вычисляем по аппроксимирующему
    полиному значение напряжения/тока и кладем
242     // результат в соответствующий регистр МПА для текущего канала МПА
243     switch ( TEST_BIT(k, ptr->AI_RomRegs.AI_OperMode.adc_chs_mode))
244     {
245     case 0:
246         // Напряжение 0-10В
247         ptr->AI_PhysQuantFloat[k] = ptr->AI_RomRegs.AI_PolynConst0[k] + (ptr-
    >AI_RomRegs.AI_PolynConst1[k])*(ptr->AI_CodeADC[k]);
248         break;
249
250     case 1:
251         // Ток 0-20мА
252         ptr->AI_PhysQuantFloat[k] = 4.004f*(ptr->AI_RomRegs.AI_PolynConst0[k] + (ptr-
    >AI_RomRegs.AI_PolynConst1[k])*(ptr->AI_CodeADC[k]));
253         break;
254
255     default:
256         break;
257     }
258 }
259 }

```

#### 4.19.1.2 main()

```

int main (
    void )
{
37 {
38     // Инициализация тактирования МК
39     clock_init();
40     // Общая инициализация блока DMA МК
41     dma_common_init();
42     // Прошивка внешней микросхемы ПЗУ (делается только 1 раз)
43     #ifdef ROM_IS_USED
44         ebc_init(EBC_ROM);
45         init_external_rom_space();
46     #endif
47     // Инициализация внешней микросхемы ОЗУ
48     ebc_init(EBC_RAM);
49     init_external_ram_space();
50     // Инициализация светодиодных индикаторов
51     leds_gpio_config();

```

```

52
53 // Инициализация и создание списков обработчиков прерываний таймеров
54 list_tmr_handler_init(1);
55 list_tmr_handler_add_tail(1, sync_adc_chanel, NULL, TIMER_STATUS_CNT_ARR);
56 list_tmr_handler_add_tail(1, receive_adc_chanel_pack, NULL, TIMER_STATUS_CCR_CAP1_CH4);
57
58 // Инициализация блока SSI1 МК
59 spi_1.SSPx = MDR_SSP1;
60 spi_1.RST_CLK_PCLK_SPIn = RST_CLK_PCLK_SSP1;
61 spi_1.SPI.SSP_WordLength = SSP_WordLength16b;
62 spi_1.SPI.SSP_Mode = SSP_ModeSlave;
63 spi_1.SPI.SSP_SPH = SSP_SPH_2Edge;
64 spi_1.SPI.SSP_FRF = SSP_FRF_SSI_TI;
65 spi_1.SPI.SSP_HardwareFlowControl = SSP_HardwareFlowControl_SSE;
66 spi_1.SPI.SSP_CPSDVSR = WORK_FREQ/2;
67 spi_1.IRQn = SSP1_IRQn;
68 spi_1.spi_dma_ch.dma_channel = DMA_Channel_REQ_SSP1_RX;
69 spi_1.buffer = ram_space_pointer->spi_1_rx_buffer;
70
71 spi_init(&spi_1);
72
73 // Инициализация блока Timer1 МК (настроен на период 10мс)
74 timer_1.TIMERInitStruct.TIMER_Period = 0x270F;//10000-1
75 timer_1.TIMERInitStruct.TIMER_Prescaler = WORK_FREQ - 1;//128-1
76 timer_1.TIMERx = MDR_TIMER1;
77
78 timer_init(&timer_1);
79
80 // Инициализация блока Timer3 МК (настроен на период 1сек)
81 timer_3.TIMERInitStruct.TIMER_Period = 0xC34F;//50000-1
82 timer_3.TIMERInitStruct.TIMER_Prescaler = WORK_FREQ*20 - 1;//2560-1
83 timer_3.TIMERx = MDR_TIMER3;
84
85 timer_init(&timer_3);
86
87 // Инициализация блока Timer2 МК (настроен для режима захвата по 2 каналу таймера - для нужд АЦП, также
настроен на период 10мкс)
88 timer_2.TIMERInitStruct.TIMER_Period = 10 + 1;
89 timer_2.TIMERInitStruct.TIMER_Prescaler = WORK_FREQ + 1;
90 timer_2.sTIM_ChnInit.TIMER_CH_Number = TIMER_CHANNEL4;
91 timer_2.sTIM_ChnInit.TIMER_CH_Mode = TIMER_CH_MODE_CAPTURE;
92 timer_2.sTIM_ChnInit.TIMER_CH_EventSource = TIMER_CH_EvSrc_PE;
93 timer_2.TIMER_STATUS = TIMER_STATUS_CCR_CAP1_CH4 | TIMER_STATUS_CNT_ARR;
94 timer_2.TIMERx = MDR_TIMER2;
95
96 timer_init(&timer_2);
97
98 // Инициализация 6-ти канальной микросхемы АЦП1
99 adc_1.spi_struct = &spi_1;
100 adc_1.spi_struct->buffer_counter = 0;
101 adc_1.timer_n_capture = &timer_2;
102 adc_1.avg_num = find_max_halfword(ram_space_pointer-
->mpa_ram_register_space.AI_RomRegs.AI_NumForAverag, CHANEL_NUMBER);
103 adc_1.ch_rx_num = 0;
104 adc_1.init_flag = 0;
105
106 adc_init(&adc_1);
107
108 // Инициализация блоков UART1-2 МК:
109 uart_1.UARTx = MDR_UART1;
110 uart_1.uart_dma_ch.dma_channel = DMA_Channel_REQ_UART1_RX;
111 uart_1.IRQn = UART1_IRQn;
112 uart_1.UART.UART_BaudRate = 115200;
113 uart_1.UART.UART_WordLength = UART_WordLength8b;
114 uart_1.UART.UART_StopBits = UART_StopBits1;
115 uart_1.UART.UART_Parity = UART_Parity_No;
116 uart_1.UART.UART_FIFOMode = UART_FIFO_OFF;
117 uart_1.UART.UART_HardwareFlowControl = UART_HardwareFlowControl_RXE |
UART_HardwareFlowControl_TXE;
118 uart_1.buffer = (uint8_t*)(ram_space_pointer->uart1_rx_buffer);
119 uart_1.buffer_count = 0;
120 uart_1.read_pos = 0;
121 uart_1.uart_timeouts.timer_n_timeout = &timer_3;
122
123 uart_2.UARTx = MDR_UART2;
124 uart_2.uart_dma_ch.dma_channel = DMA_Channel_REQ_UART2_RX;
125 uart_2.IRQn = UART2_IRQn;
126 uart_2.UART.UART_BaudRate = 921600;
127 uart_2.UART.UART_WordLength = UART_WordLength8b;
128 uart_2.UART.UART_StopBits = UART_StopBits1;
129 uart_2.UART.UART_Parity = UART_Parity_No;
130 uart_2.UART.UART_FIFOMode = UART_FIFO_OFF;
131 uart_2.UART.UART_HardwareFlowControl = UART_HardwareFlowControl_RXE |
UART_HardwareFlowControl_TXE;
132 uart_2.buffer = (uint8_t*)(ram_space_pointer->uart2_rx_buffer);
133 uart_2.buffer_count = 0;
134 uart_2.read_pos = 0;

```

```

135  uart_2.uart_timeouts.timer_n_timeout = &timer_3;
136
137  // Установка таймаутов на ШИНАХ1-2
138  //uart_set_read_timeout(&uart_1, 300);
139  uart_set_read_timeout(&uart_2, 300);
140
141  // Инициализация ШИНЫ1
142  //uart_init(&uart_1);
143  // Инициализация DMA для UART1
144  //DMA_UART_RX_init(&UART1);
145
146  // Инициализация ШИНЫ2
147  uart_init(&uart_2);
148  // Инициализация DMA для UART2
149  DMA_UART_RX_init(&uart_2);
150
151  while(1)
152  {
153      //запрос пакета по ШИНЕ1
154      //request_data(&uart_1);
155      //запрос пакета по ШИНЕ2
156      request_data(&uart_2);
157  }
158 }

```

#### 4.19.1.3 receive\_adc\_chanel\_pack()

```

void receive_adc_chanel_pack (
    void * data )

```

Принимает пакет с результатами измерений одного канала (выполняется при срабатывании прерывания Timer2 по захвату)

Аргументы

*data	- Указатель на передаваемые при необходимости данные (не используется)
-------	--

```

296 {
297     // Только, если инициализирована микросхема АЦП
298     if ((adc_1.init_flag == 1))
299     {
300         // Выключаем прерывания таймера, срабатываемое при переполнении счетчика таймера
301         TIMER_ITConfig(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR, DISABLE);
302         // Ведем счетчик принятых данных каналов АЦП
303         adc_1.ch_rx_num++;
304         // Если счетчик превышает максимальное число каналов АЦП, значит принятые данные соответствуют
           от микросхемы АЦП
305         if (adc_1.ch_rx_num == (CHANEL_NUMBER+1))
306         {
307             adc_1.ch_rx_num = 1;
308         }
309         // Сброс счетчика таймера, тем самым устанавливается таймаут на временной интервал между пакетами данных
           от микросхемы АЦП
310         TIMER_SetCounter(adc_1.timer_n_capture->TIMERx, 0);
311         TIMER_ClearITPendingBit(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR);
312         // Включаем прерывания таймера, срабатываемое при переполнении счетчика таймера
313         TIMER_ITConfig(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR, ENABLE);
314     }
315     TIMER_ClearITPendingBit(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CCR_CAP1_CH4);
316 }

```

#### 4.19.1.4 request\_data()

```

uint8_t request_data (
    uart_n * uart_struct )

```

Запрашивает данные на выбранной шине



## Аргументы

*uart_struct	- Выбранный UART МК
--------------	---------------------

## Возвращает

Код возврата: 0 - пакет данных получен, обработан и ответный пакет отправлен; 1- нет принятого пакета, ошибка

```

163 {
164     // Номер шины, по которой запрашиваются данные
165     uint8_t ext_bus;
166
167     // Определение шины, по которой идет обмен данными
168     RECOGNIZE_BUS(ext_bus, uart_struct);
169
170     // Прием пакета данных по шине
171     if(receive_packet(uart_struct, ext_bus) != NO_ERROR)
172     {
173         return 1;
174     }
175
176     // Выполнение команды периферией (для МПА - опрос каналов АЦП)
177     do_mpa_task(&adc_1);
178
179     // Выполнение соответствующих протокольных команд
180     if(protocol_do_cmds(ext_bus) != 0)
181     {
182         return 1;
183     }
184
185     // Передача ответного пакета
186     if(transmit_packet(uart_struct, ext_bus) != NO_ERROR)
187     {
188         return 1;
189     }
190
191     return 0;
192 }
```

## 4.19.1.5 sync\_adc\_channels()

```

void sync_adc_channels (
    void * data )
```

Синхронизирует каналы АЦП (выполняется при срабатывании прерывания Timer2 по переполнению счетчика CNT)

## Аргументы

*data	- Указатель на передаваемые при необходимости данные (не используется)
-------	--

```

264 {
265     // Только, если инициализирована микросхема АЦП
266     if ((adc_1.init_flag == 1))
267     {
268         // Выключаем прерывания таймера, срабатываемое при переполнении счетчика таймера
269         TIMER_ITConfig(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR, DISABLE);
270
271         // Считываем FIFO буфер SPI
272         uint16_t spi_rx_value[FIFO_SIZE];
273         for (uint8_t i = 0; i < FIFO_SIZE; i++)
274         {
275             spi_rx_value[i] = adc_1.spi_struct->SSPx->DR;
276         }
277
278         // Только, если получили данные всех каналов микросхемы АЦП, то переносим данные из FIFO буфера SPI в
        буфер SPI, расположенный во внешней ОЗУ

```

```

279     if (adc_1.ch_rx_num == CHANEL_NUMBER)
280     {
281         memcpy(ram_space_pointer->spi_1_rx_buffer + spi_1.buffer_counter, spi_rx_value,
CHANEL_NUMBER*sizeof(spi_rx_value[0]));
282         spi_1.buffer_counter += CHANEL_NUMBER;
283         if (adc_1.spi_struct->buffer_counter >= (CHANEL_NUMBER*adc_1.avg_num))
284         {
285             adc_1.spi_struct->buffer_counter = 0;
286         }
287     }
288     adc_1.ch_rx_num = 0;
289 }
290 TIMER_ClearITPendingBit(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR);
291 }

```

## 4.19.2 Переменные

### 4.19.2.1 adc\_1

`adc_n` `adc_1` [extern]

Структура с конфигурационными параметрами микросхемы АЦП

### 4.19.2.2 heap

`int_ram_heap` `heap`

Выделение памяти во внутреннем ОЗУ МК для "самодельной" кучи

### 4.19.2.3 heap\_ptr

`int_ram_heap*` `heap_ptr` = `&heap`

Указатель на "самодельную" кучу

### 4.19.2.4 ram\_space\_pointer

`ram_data*` `ram_space_pointer`

Указатель для обращения к внешнему ОЗУ

## 4.19.2.5 rom\_space\_pointer

```
rom_data* rom_space_pointer
```

Указатель для обращения к внешнему ПЗУ

## 4.19.2.6 spi\_1

```
spi_n spi_1 [extern]
```

Структуры с конфигурационными параметрами блоков SPI МК

Структуры с конфигурационными параметрами блоков SPI МК

## 4.19.2.7 spi\_2

```
spi_n spi_2 [extern]
```

Структура с конфигурационными параметрами блока SPI2 МК

## 4.19.2.8 timer\_1

```
timer_n timer_1 [extern]
```

Структуры с конфигурационными параметрами блоков таймеров МК

Структуры с конфигурационными параметрами блоков таймеров МК

## 4.19.2.9 timer\_2

```
timer_n timer_2 [extern]
```

Структура с конфигурационными параметрами блока Timer2 МК

## 4.19.2.10 timer\_3

```
timer_n timer_3 [extern]
```

Структура с конфигурационными параметрами блока Timer3 МК

## 4.19.2.11 tmr\_handler\_head

```
list_head* tmr_handler_head[TIMER_NUM] [extern]
```

Массив указателей на head списков обработчиков прерываний таймеров

Массив указателей на head списков обработчиков прерываний таймеров

## 4.19.2.12 uart\_1

```
uart_n uart_1 [extern]
```

Структуры с конфигурационными параметрами блоков UART МК

Структуры с конфигурационными параметрами блоков UART МК

## 4.19.2.13 uart\_2

```
uart_n uart_2 [extern]
```

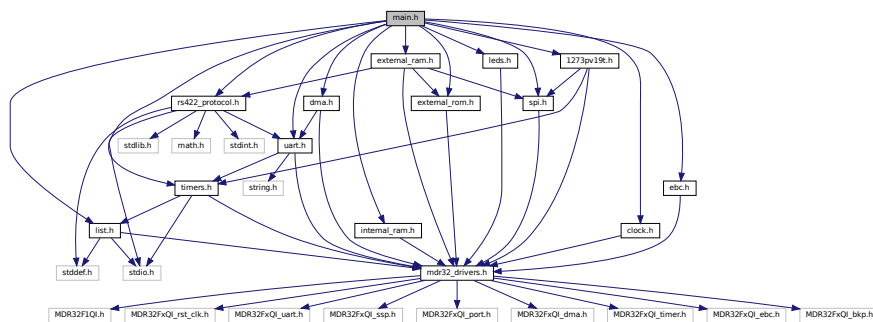
Структура с конфигурационными параметрами блока UART2 МК

## 4.20 Файл main.h

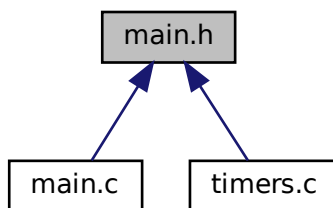
Заголовочный файл с реализацией основного функционала МПА

```
#include "1273pv19t.h"
#include "timers.h"
#include "spi.h"
#include "uart.h"
#include "rs422_protocol.h"
#include "clock.h"
#include "dma.h"
#include "ebc.h"
#include "internal_ram.h"
#include "external_ram.h"
#include "external_rom.h"
#include "leds.h"
#include "list.h"
```

Граф включаемых заголовочных файлов для main.h:



Граф файлов, в которые включается этот файл:



## Функции

- `uint8_t request_data (uart_n *uart_struct)`  
Запрашивает данные на выбранной шине
- `void do_mpa_task (adc_n *adc_struct)`  
Выполняет периферийные задачи МПА
- `void sync_adc_channels (void *data)`  
Синхронизирует каналы АЦП (выполняется при срабатывании прерывания Timer2 по переполнению счетчика CNT)
- `void receive_adc_channel_pack (void *data)`  
Принимает пакет с результатами измерений одного канала (выполняется при срабатывании прерывания Timer2 по захвату)

### 4.20.1 Подробное описание

Заголовочный файл с реализацией основного функционала МПА

### 4.20.2 Функции

#### 4.20.2.1 do\_mpa\_task()

```
void do_mpa_task (
    adc_n * adc_struct )
```

Выполняет периферийные задачи МПА

Аргументы

<code>*adc_struct</code>	- Структура с конфигурационными параметрами микросхемы АЦП
--------------------------	--

197 {

```

198 // Темповые переменные, необходимые для обработки данных АЦП
199 int adc_code[MAX_CHANEL_NUMBER] = {0};
200 int16_t adc_value;
201
202 // TODO: на текущий момент данная ф-ция обрабатывает данные каналов МПА только для напряжений 0-10В
203 // (для тока в карту регистров надо добавлять свои полиномы, т.к. они отличаются)
204
205 // Указатель на пространство регистров МПА
206 mpa_ram_registers *ptr = &ram_space_pointer->mpa_ram_register_space;
207
208 // Ниже приведены аппроксимирующие полиномы, полученные экспериментально для всех режимов работы
209 // отладочной платы АЦП
210 /*
211 для двуполярного вх напр (-5.4 ... 5.4 В ) на мультиплексоре A0=A1=0
212     U = 1.6474f*pow(10,-4)*adc_code;
213     delta = 6.5627f*pow(10,-6)*adc_code + 0.00039f;
214 для однополярного вх напр (0 ... 10.8 В ) на мультиплексоре A0=1;A1=0
215     U = 1.6474f*pow(10,-4)*adc_code + 5.398f;
216     delta = 6.6962f*pow(10,-6)*adc_code + 0.4252307f;
217 для однополярного вх тока (0 ... 21.6 мА ) на мультиплексоре A0=1;A1=0
218     I = 3.052f*pow(10,-4)*adc_code + 10.0f;
219     delta = -11.9006f*pow(10,-6)*adc_code + 0.03072506f;
220 самодиагностика для двуполярного случая на мультиплексоре A0=1;A1=1 (на выходе должно быть 0В)
221     U = 1.6474f*pow(10,-4)*adc_code;
222     delta = 6.5627f*pow(10,-6)*adc_code + 0.00039f;
223 самодиагностика для однополярного случая на мультиплексоре A0=0;A1=1 (на выходе должно быть 0В)
224     U = 1.6474f*pow(10,-4)*adc_code + 5.398f;
225     delta = 6.6962f*pow(10,-6)*adc_code + 0.4252307f;
226
227 Результирующее напряжение после аппроксимации U = U - delta;
228 */
229 // Производим усреднение по максим кол-ву выборок, если кол-во выборок различается для разных каналов МПА
230 adc_1.avg_num = find_max_halfword(ptr->AI_RomRegs.AI_NumForAverag, CHANEL_NUMBER);
231
232 for (uint8_t k = 0; k < CHANEL_NUMBER; k++)
233 {
234     for (uint8_t i = 0; i < ptr->AI_RomRegs.AI_NumForAverag[k]; i++)
235     {
236         memcpy(&adc_value, adc_struct->spi_struct->buffer + (i*CHANEL_NUMBER) + k, sizeof(adc_value));
237         adc_code[k] += (int16_t)(-adc_value + 1);
238     }
239     adc_code[k] /= ptr->AI_RomRegs.AI_NumForAverag[k];
240     // Кладем в соответствующий регистр МПА полученные код АЦП для текущего канала МПА
241     memcpy(&(ptr->AI_CodeADC[k]), &adc_code[k], sizeof(adc_code[k]));
242     // В зависимости от режима работы канала МПА (ток/напряжение) вычисляем по аппроксимирующему
243     // полиному значение напряжения/тока и кладем
244     // результат в соответствующий регистр МПА для текущего канала МПА
245     switch ( TEST_BIT(k, ptr->AI_RomRegs.AI_OperMode.adc_chs_mode))
246     {
247     case 0:
248         // Напряжение 0-10В
249         ptr->AI_PhysQuantFloat[k] = ptr->AI_RomRegs.AI_PolynConst0[k] + (ptr-
250 >AI_RomRegs.AI_PolynConst1[k])*(ptr->AI_CodeADC[k]);
251         break;
252     case 1:
253         // Ток 0-20мА
254         ptr->AI_PhysQuantFloat[k] = 4.004f*(ptr->AI_RomRegs.AI_PolynConst0[k] + (ptr-
255 >AI_RomRegs.AI_PolynConst1[k])*(ptr->AI_CodeADC[k]));
256         break;
257     default:
258         break;
259     }
260 }

```

#### 4.20.2.2 receive\_adc\_channel\_pack()

```

void receive_adc_channel_pack (
    void * data )

```

Принимает пакет с результатами измерений одного канала (выполняется при срабатывании прерывания Timer2 по захвату)

## Аргументы

*data	- Указатель на передаваемые при необходимости данные (не используется)
-------	--

```

296 {
297     // Только, если инициализирована микросхема АЦП
298     if ((adc_1.init_flag == 1))
299     {
300         // Выключаем прерывания таймера, срабатываемое при переполнении счетчика таймера
301         TIMER_ITConfig(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR, DISABLE);
302         // Ведем счетчик принятых данных каналов АЦП
303         adc_1.ch_rx_num++;
304         // Если счетчик переваливает максимальное число каналов АЦП, значит принятые данные соответствуют
           первому каналу АЦП
305         if (adc_1.ch_rx_num == (CHANEL_NUMBER+1))
306         {
307             adc_1.ch_rx_num = 1;
308         }
309         // Сброс счетчика таймера, тем самым устанавливается таймаут на временной интервал между пакетами данных
           от микросхемы АЦП
310         TIMER_SetCounter(adc_1.timer_n_capture->TIMERx, 0);
311         TIMER_ClearITPendingBit(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR);
312         // Включаем прерывания таймера, срабатываемое при переполнении счетчика таймера
313         TIMER_ITConfig(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR, ENABLE);
314     }
315     TIMER_ClearITPendingBit(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CCR_CAP1_CH4);
316 }

```

## 4.20.2.3 request\_data()

```

uint8_t request_data (
    uart_n * uart_struct )

```

Запрашивает данные на выбранной шине

## Аргументы

*uart_struct	- Выбранный UART МК
--------------	---------------------

## Возвращает

Код возврата: 0 - пакет данных получен, обработан и ответный пакет отправлен; 1- нет принятого пакета, ошибка

```

163 {
164     // Номер шины, по которой запрашиваются данные
165     uint8_t ext_bus;
166
167     // Определение шины, по которой идет обмен данными
168     RECOGNIZE_BUS(ext_bus, uart_struct);
169
170     // Прием пакета данных по шине
171     if(receive_packet(uart_struct, ext_bus) != NO_ERROR)
172     {
173         return 1;
174     }
175
176     // Выполнение команды периферией (для МПА - опрос каналов АЦП)
177     do_mpa_task(&adc_1);
178
179     // Выполнение соответствующих протокольных команд
180     if(protocol_do_cmds(ext_bus) != 0)
181     {
182         return 1;
183     }
184
185     // Передача ответного пакета
186     if(transmit_packet(uart_struct, ext_bus) != NO_ERROR)

```

```

187 {
188     return 1;
189 }
190
191 return 0;
192 }

```

#### 4.20.2.4 sync\_adc\_channels()

```

void sync_adc_channels (
    void * data )

```

Синхронизирует каналы АЦП (выполняется при срабатывании прерывания Timer2 по переполнению счетчика CNT)

Аргументы

*data	- Указатель на передаваемые при необходимости данные (не используется)
-------	--

```

264 {
265     // Только, если инициализирована микросхема АЦП
266     if ((adc_1.init_flag == 1))
267     {
268         // Выключаем прерывания таймера, срабатываемое при переполнении счетчика таймера
269         TIMER_ITConfig(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR, DISABLE);
270
271         // Считываем FIFO буфер SPI
272         uint16_t spi_rx_value[FIFO_SIZE];
273         for (uint8_t i = 0; i < FIFO_SIZE; i++)
274         {
275             spi_rx_value[i] = adc_1.spi_struct->SSPx->DR;
276         }
277
278         // Только, если получили данные всех каналов микросхемы АЦП, то переносим данные из FIFO буфера SPI в
        // буфер SPI, расположенный во внешней ОЗУ
279         if (adc_1.ch_rx_num == CHANEL_NUMBER)
280         {
281             memcpy(ram_space_pointer->spi_1_rx_buffer + spi_1.buffer_counter, spi_rx_value,
282                 CHANEL_NUMBER*sizeof(spi_rx_value[0]));
283             spi_1.buffer_counter += CHANEL_NUMBER;
284             if (adc_1.spi_struct->buffer_counter >= (CHANEL_NUMBER*adc_1.avg_num))
285             {
286                 adc_1.spi_struct->buffer_counter = 0;
287             }
288             adc_1.ch_rx_num = 0;
289         }
290         TIMER_ClearITPendingBit(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR);
291     }

```

## 4.21 Файл mdr32\_drivers.h

Заголовочный файл с глобальными константами и подключаемыми библиотеками

```

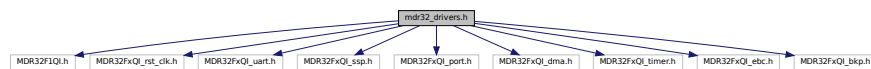
#include <MDR32F1QI.h>
#include <MDR32FxQI_rst_clk.h>
#include <MDR32FxQI_uart.h>
#include <MDR32FxQI_ssp.h>
#include <MDR32FxQI_port.h>
#include <MDR32FxQI_dma.h>
#include <MDR32FxQI_timer.h>
#include <MDR32FxQI_ebc.h>

```

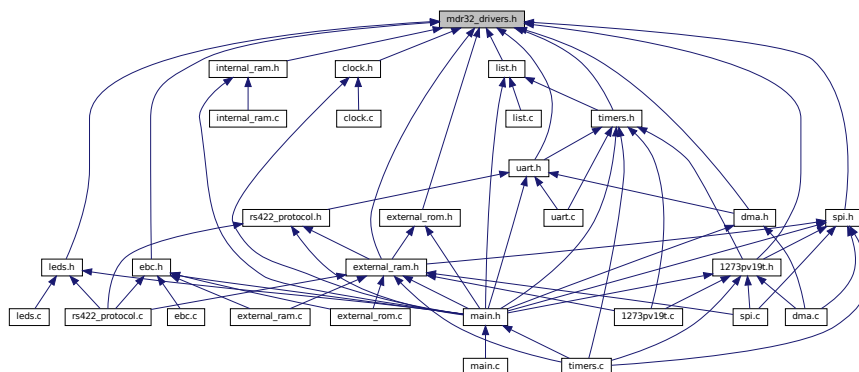


```
#include <MDR32FxDI_bkp.h>
```

Граф включаемых заголовочных файлов для mdr32\_drivers.h:



Граф файлов, в которые включается этот файл:



## Макросы

- `#define K1986VE1T`  
Выбор МК K1986VE1T
- `#define HSE_OSC ((uint32_t)12000000)`  
Частота внешнего тактового генератора (!также изменить в MDR32\_config.h)
- `#define WORK_FREQ 144`  
Рабочая частота в МГц
- `#define CHANEL_NUMBER 6`  
Кол-во каналов в МПА
- `#define MAX_CHANEL_NUMBER 8`  
Максимальное кол-во каналов в МПА
- `#define PM_DEV_ADDR 0`  
Адрес модуля
- `#define PM_CHASSIS_ADDR 0`  
Адрес шасси

### 4.21.1 Подробное описание

Заголовочный файл с глобальными константами и подключаемыми библиотеками

### 4.21.2 Макросы

#### 4.21.2.1 CHANEL\_NUMBER

```
#define CHANEL_NUMBER 6
```

Кол-во каналов в МПА

#### 4.21.2.2 HSE\_OSC

```
#define HSE_OSC ((uint32_t)12000000)
```

Частота внешнего тактового генератора (!также изменить в MDR32\_config.h)

#### 4.21.2.3 K1986VE1T

```
#define K1986VE1T
```

Выбор МК K1986BE1T

#### 4.21.2.4 MAX\_CHANEL\_NUMBER

```
#define MAX_CHANEL_NUMBER 8
```

Максимальное кол-во каналов в МПА

#### 4.21.2.5 PM\_CHASSIS\_ADDR

```
#define PM_CHASSIS_ADDR 0
```

Адрес шасси

#### 4.21.2.6 PM\_DEV\_ADDR

```
#define PM_DEV_ADDR 0
```

Адрес модуля

## 4.21.2.7 WORK\_FREQ

```
#define WORK_FREQ 144
```

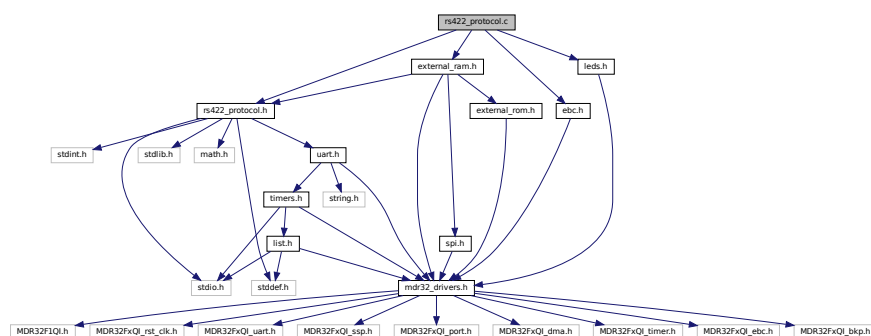
Рабочая частота в МГц

## 4.22 Файл rs422\_protocol.c

Файл с реализацией API протокола обмена данными по интерфейсу RS-422.

```
#include "rs422_protocol.h"
#include "external_ram.h"
#include "ebc.h"
#include "leds.h"
```

Граф включаемых заголовочных файлов для rs422\_protocol.c:



## Функции

- `protocol_error transmit_packet (uart_n *uart_struct, uint8_t ext_bus)`  
Отправляет пакет данных
- `protocol_error receive_packet (uart_n *uart_struct, uint8_t ext_bus)`  
Читает пакет данных
- `uint8_t protocol_do_cmds (uint8_t ext_bus)`  
Выполняет требуемые команды
- `uint_least32_t crc32 (unsigned char *buf, size_t len)`
- `void fill_crc32_table (void)`  
Заполняет таблицу CRC32.
- `void rx_error_handler (protocol_error error, uint8_t ext_bus)`  
Обрабатывает ошибки приема пакетов
- `void um_service_byte_handler (uint8_t ext_bus)`  
Обрабатывает сервисный байт УМ

## Переменные

- `ram_data * ram_space_pointer`  
Указатель для обращения к внешнему ОЗУ
- `rom_data * rom_space_pointer`  
Указатель для обращения к внешнему ПЗУ

### 4.22.1 Подробное описание

Файл с реализацией API протокола обмена данными по интерфейсу RS-422.

### 4.22.2 Функции

#### 4.22.2.1 crc32()

```
uint_least32_t crc32 (
    unsigned char * buf,
    size_t len )
438 {
439     uint_least32_t crc;
440
441     crc = 0xFFFFFFFFUL;
442
443     while (len--)
444         crc = ram_space_pointer->crc_table[(crc ^ *buf++) & 0xFF] ^ (crc >> 8);
445
446     return crc ^ 0xFFFFFFFFUL;
447 }
```

#### 4.22.2.2 fill\_crc32\_table()

```
void fill_crc32_table (
    void )
```

Заполняет таблицу CRC32.

```
452 {
453     uint_least32_t crc; int i, j;
454     for (i = 0; i < 256; i++)
455     {
456         crc = i;
457         for (j = 0; j < 8; j++)
458             crc = crc & 1 ? (crc >> 1) ^ 0xEDB88320UL : crc >> 1;
459
460         ram_space_pointer->crc_table[i] = crc;
461     }
462 }
```

#### 4.22.2.3 protocol\_do\_cmds()

```
uint8_t protocol_do_cmds (
    uint8_t ext_bus )
```

Выполняет требуемые команды

Аргументы

ext_bus	- Номер шины
---------	--------------

Возвращает

Код ошибки protocol\_error

```

205 {
206     common_ram_registers *common_ram_reg_space_ptr = &ram_space_pointer->common_ram_register_space; //
        // Указатель на область памяти внешнего ОЗУ с общими регистрами
207     fields_cmd *tx_pack_cmd_with_data_ptr = &ram_space_pointer->tx_packet_struct.cmd_with_data[0]; //
        // Указатель на массив команд с данными для отправляемого пакета
208     fields_cmd *rx_pack_cmd_with_data_ptr = &ram_space_pointer->rx_packet_struct.cmd_with_data[0]; //
        // Указатель на массив команд с данными для принимаемого пакета
209     fields_packet_header *tx_pack_header = &ram_space_pointer->tx_packet_struct.packet_header; // Указатель на
        // заголовок отправляемого пакета
210     fields_packet_header *rx_pack_header = &ram_space_pointer->rx_packet_struct.packet_header; // Указатель на
        // заголовок принимаемого пакета
211     fields_packet_tail *tx_pack_tail = &ram_space_pointer->tx_packet_struct.packet_tail; // Указатель на хвост
        // отправляемого пакета
212     fields_packet_tail *rx_pack_tail = &ram_space_pointer->rx_packet_struct.packet_tail; // Указатель на хвост
        // принятого пакета
213     uint32_t offset = 0; // Указатель на данные для каждой команды
214     uint16_t start_addr = 0; // Начальный адрес для чтения/записи регистров
215     uint16_t size = 0; // Кол-во байт для чтения/записи
216
217     for (uint8_t i = 0; i < rx_pack_header->cmd_number; i++)
218     {
219         switch ((rx_pack_cmd_with_data_ptr + i)->header.cmd)
220         {
221             case TYPE_CMD:
222                 // По команде TYPE кладем в поле data регистры PLC_SoftVer, PLC_Config, PLC_DeviceType,
                // PLC_SerialNumber
223                 (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
224
225                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->PLC_SoftVer),
226                     sizeof(common_ram_reg_space_ptr->PLC_SoftVer));
227                 offset += sizeof(common_ram_reg_space_ptr->PLC_SoftVer);
228                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->PLC_Config),
229                     sizeof(common_ram_reg_space_ptr->PLC_Config));
230                 offset += sizeof(common_ram_reg_space_ptr->PLC_Config);
231                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->
                >PLC_CommonRomRegs.PLC_DeviceType), sizeof(common_ram_reg_space_ptr-
                >PLC_CommonRomRegs.PLC_DeviceType));
232                 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_DeviceType);
233                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->
                >PLC_CommonRomRegs.PLC_SerialNumber), sizeof(common_ram_reg_space_ptr-
                >PLC_CommonRomRegs.PLC_SerialNumber));
234                 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
235
236                 (tx_pack_cmd_with_data_ptr + i)->header.cmd = TYPE_CMD;
237                 // TODO:разобраться для чего поле result
238                 (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
239                 (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) +
                sizeof(common_ram_reg_space_ptr->PLC_SoftVer) +
240                 sizeof(ram_space_pointer->common_ram_register_space.PLC_Config) +
241                 sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_DeviceType) +
242                 sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
243                 break;
244
245             case INIT_CMD:
246                 // По команде INIT кладем в поле data регистр PLC_SerialNumber, если выполнен ряд условий
247                 switch (ext_bus)
248                 {
249                     case 1:
250                         if((ram_space_pointer->service_byte_um.ready_to_control == 1) &&
251                             (ram_space_pointer->service_byte_pm.fail_bus_1 == 0) && (common_ram_reg_space_ptr->PLC_CM_State !=
252                             PLC_CM_INIT_2_BUS))
253                         {
254                             // Заносим инфу в сервисный байт ПМ
255                             ram_space_pointer->service_byte_pm.init = 1;
256                             ram_space_pointer->service_byte_pm.master = 0;
257                             // Заносим инфу в регистры
258                             common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_INIT_1_BUS;
259                         }
260                     else
261                     {
262                         continue;
263                     }
264                     break;
265
266                     case 2:
267                         if((ram_space_pointer->service_byte_um.ready_to_control == 1) &&
268                             (ram_space_pointer->service_byte_pm.fail_bus_2 == 0) && (common_ram_reg_space_ptr->PLC_CM_State !=
269                             PLC_CM_INIT_1_BUS))
270                         {
271                             // Заносим инфу в сервисный байт ПМ
272                             ram_space_pointer->service_byte_pm.init = 1;
273                             ram_space_pointer->service_byte_pm.master = 1;
274                             // Заносим инфу в регистры

```

```

267         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_INIT_2_BUS;
268     }
269     else
270     {
271         continue;
272     }
273     break;
274
275     default:
276         break;
277 }
278 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr-
>PLC_CommonRomRegs.PLC_SerialNumber), sizeof(common_ram_reg_space_ptr-
>PLC_CommonRomRegs.PLC_SerialNumber));
279 (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
280 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
281 (tx_pack_cmd_with_data_ptr + i)->header.cmd = INIT_CMD;
282 // TODO:разобраться для чего поле result
283 (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
284 (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) +
sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
285 break;
286
287 case READ_CMD:
288     memcpy(&start_addr, (rx_pack_cmd_with_data_ptr + i)->data, sizeof(start_addr));
289     memcpy(&size, (rx_pack_cmd_with_data_ptr + i)->data + sizeof(start_addr), sizeof(size));
290
291     // По команде READ кладем в поле data size байт начиная с start_addr
292     memcpy((ram_space_pointer->tx_data) + offset, (void*)&(ram_space_pointer->start_struct)) +
start_addr, size);
293     (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
294     offset += size;
295     (tx_pack_cmd_with_data_ptr + i)->header.cmd = READ_CMD;
296     // TODO:разобраться для чего поле result
297     (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
298     (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + size;
299     break;
300
301 case WRITE_CMD:
302     // Проверка того, что установлено соединение по выбранной шине
303     switch (ext_bus)
304     {
305         case 1:
306             if(ram_space_pointer->common_ram_register_space.PLC_CM_State !=
PLC_CM_INIT_1_BUS)
307             {
308                 continue;
309             }
310             break;
311
312         case 2:
313             if(ram_space_pointer->common_ram_register_space.PLC_CM_State !=
PLC_CM_INIT_2_BUS)
314             {
315                 continue;
316             }
317             break;
318
319         default:
320             break;
321     }
322     memcpy(&start_addr, (rx_pack_cmd_with_data_ptr + i)->data, sizeof(start_addr));
323     memcpy(&size, ((rx_pack_cmd_with_data_ptr + i)->data) + sizeof(start_addr), sizeof(size));
324     // По команде WRITE кладем по адресу start_addr size принятых байт для записи и в поле данных
кладем код команды
325     memcpy((void*)&(ram_space_pointer->start_struct)) + start_addr, ((rx_pack_cmd_with_data_ptr
+ i)->data) + sizeof(start_addr) + sizeof(size), size);
326     memset((ram_space_pointer->tx_data) + offset, WRITE_CMD, 1);
327     (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
328     offset++;
329     (tx_pack_cmd_with_data_ptr + i)->header.cmd = WRITE_CMD;
330     // TODO:разобраться для чего поле result
331     (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
332     (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + 1;
333     break;
334
335 case RESET_CMD:
336     // Проверка того, что установлено соединение по выбранной шине
337     switch (ext_bus)
338     {
339         case 1:
340             if(common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_1_BUS)
341             {
342                 continue;
343             }
344             break;
345

```

```

346         case 2:
347             if(common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_2_BUS)
348             {
349                 continue;
350             }
351             break;
352
353         default:
354             break;
355     }
356     // По команде RESET сбрасываем регистры и в поле данных кладем код команды
357     init_external_ram_space();
358     memset((ram_space_pointer->tx_data) + offset, RESET, 1);
359     (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
360     offset++;
361     (tx_pack_cmd_with_data_ptr + i)->header.cmd = RESET;
362     // TODO:разобраться для чего поле result
363     (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
364     (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + 1;
365     break;
366
367 case CONFIG:
368     // Проверка того, что установлено соединение по выбранной шине
369     switch (ext_bus)
370     {
371         case 1:
372             if(common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_1_BUS)
373             {
374                 continue;
375             }
376             break;
377
378         case 2:
379             if(common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_2_BUS)
380             {
381                 continue;
382             }
383             break;
384
385         default:
386             break;
387     }
388     // Проверка на то, что ПМ находится в сервисном режиме
389     if (common_ram_reg_space_ptr->PLC_PMAAddr.module_addr != 0x00)
390     {
391         break;
392     }
393     // По команде CONFIG регистры зеркализируются в ПЗУ и в поле данных кладем код команды
394     memcpy(&start_addr, (rx_pack_cmd_with_data_ptr + i)->data, sizeof(start_addr));
395     memcpy(&size, (rx_pack_cmd_with_data_ptr + i)->data + sizeof(start_addr), sizeof(size));
396     // Если используется внешнее ПЗУ, то записываем данные в ПЗУ
397     #ifdef ROM_IS_USED
398     //запись данных в ПЗУ
399     memcpy(&common_regs, &common_ram_reg_space_ptr->PLC_CommonRomRegs,
400 sizeof(common_regs));
401     memcpy(&mpa_regs, &ram_space_pointer->mpa_ram_register_space.AI_RomRegs, sizeof(mpa_regs));
402     ebc_init(EBC_ROM);
403     memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR, &common_regs, sizeof(common_regs));
404     memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR + sizeof(common_regs), &mpa_regs,
405 sizeof(mpa_regs));
406     ebc_init(EBC_RAM);
407     #endif
408
409     memset((ram_space_pointer->tx_data) + offset, CONFIG, 1);
410     (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
411     offset++;
412     (tx_pack_cmd_with_data_ptr + i)->header.cmd = CONFIG;
413     // TODO:разобраться для чего поле result
414     (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
415     (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + 1;
416     break;
417
418     default:
419         break;
420     }
421     tx_pack_header->packet_length += (tx_pack_cmd_with_data_ptr + i)->header.length;
422
423 // Заполнение остальных полей
424 tx_pack_header->header = rx_pack_header->header;
425 tx_pack_header->receiver_addr = rx_pack_header->sender_addr;
426 tx_pack_header->sender_addr = rx_pack_header->receiver_addr;
427 tx_pack_header->packet_length += sizeof(ram_space_pointer->tx_packet_struct.packet_header) -
428 sizeof(tx_pack_header->header) +
429 sizeof(tx_pack_tail->checksum);
430 memcpy(&(tx_pack_header->service_byte), &(ram_space_pointer->service_byte_pm), sizeof(ram_space_pointer->service_byte_pm));
431 tx_pack_header->cmd_number = rx_pack_header->cmd_number;

```

```

429
430     tx_pack_tail->end = rx_pack_tail->end;
431
432     return 0;
433 }

```

#### 4.22.2.4 receive\_packet()

```

protocol_error receive_packet (
    uart_n * uart_struct,
    uint8_t ext_bus )

```

Читает пакет данных

Аргументы

*uart_struct	- Выбранный UART МК
ext_bus	- Номер шины

Возвращает

Код ошибки protocol\_error

```

62 {
63     fields_packet *rx_pack_ptr = &ram_space_pointer->rx_packet_struct; // Указатель на структуру с принимаемым
        пакетом
64     fields_packet_header *rx_pack_header_ptr = &ram_space_pointer->rx_packet_struct.packet_header; // Указатель
        на заголовок принимаемого пакета
65     fields_packet_tail *rx_pack_tail_ptr = &ram_space_pointer->rx_packet_struct.packet_tail; // Указатель на
        заголовок принимаемого пакета
66     fields_cmd *rx_pack_cmd_with_data_ptr = &ram_space_pointer->rx_packet_struct.cmd_with_data[0]; //
        Указатель на массив команд с данными для принимаемого пакета
67     uint32_t current_rx_packet = 0;
68
69     // Код последней ошибки, возникшей в процессе обмена данными
70     protocol_error error;
71     // Код последней ошибки, связанной с работой блока UART МК
72     uart_errors uart_error;
73
74     uint8_t packet_head;
75
76     // Очистка всех структур данных связанных с принимаемым и отправляемым пакетом
77     memset(rx_pack_ptr, 0, sizeof(ram_space_pointer->rx_packet_struct) + sizeof(ram_space_pointer->tx_packet_struct)
        +
78         sizeof(ram_space_pointer->tx_data) + sizeof(ram_space_pointer->packet_tx) + sizeof(ram_space_pointer-
        >packet_rx));
79
80     // Здесь в цикле ловится и обрабатывается последний пакет в буфере приемника UART
81     while (1)
82     {
83         if (uart_read_pos(uart_struct) < UART_BUFFER_SIZE)
84         {
85             memcpy(&packet_head, (uint8_t*)(GET_UART_BUF_PTR + uart_read_pos(uart_struct)),
                sizeof(packet_head));
86         }
87         else
88         {
89             memcpy(&packet_head, (uint8_t*)(GET_UART_BUF_PTR), sizeof(packet_head));
90         }
91
92         // Если уже были разобраны какие то пакеты, а начала следующего пакета (0x55) в буфере нет, это означает что
        крайний разобранный пакет оказался последним и выходим из цикла
93         if ((current_rx_packet != 0) && (packet_head != PACKET_HEAD))
94         {
95             uart_clean(uart_struct);
96             break;
97         }
98
99         // Считываем первый байт и проверяем на 0x55
100         uart_error = uart_read(uart_struct, sizeof(rx_pack_header_ptr->header), ram_space_pointer->packet_rx);

```



```

101     if (uart_error != 0)
102     {
103         error = UART_ERROR;
104         rx_error_handler(error, ext_bus);
105         return error;
106     }
107
108     memcpy(&(rx_pack_header_ptr->header), ram_space_pointer->packet_rx, sizeof(rx_pack_header_ptr->header));
109     if (rx_pack_header_ptr->header != PACKET_HEAD)
110     {
111         error = PACKET_ERROR;
112         rx_error_handler(error, ext_bus);
113         return error;
114     }
115
116     // Считываем заголовок телеграммы
117     uart_error = uart_read(uart_struct, sizeof(rx_pack_ptr->packet_header) - sizeof(rx_pack_header_ptr->header),
118 (ram_space_pointer->packet_rx) + sizeof(rx_pack_header_ptr->header));
119     if (uart_error != 0)
120     {
121         error = UART_ERROR;
122         rx_error_handler(error, ext_bus);
123         return error;
124     }
125     memcpy(rx_pack_header_ptr, ram_space_pointer->packet_rx, sizeof(rx_pack_ptr->packet_header));
126
127     // Анализ длины пакета
128     if (rx_pack_header_ptr->packet_length > UART_BUFFER_SIZE)
129     {
130         error = PACKET_ERROR;
131         rx_error_handler(error, ext_bus);
132         return error;
133     }
134
135     // Считываем весь пакет вычисленной длины
136     uart_error = uart_read(uart_struct, (rx_pack_header_ptr->packet_length) - sizeof(rx_pack_ptr->packet_header)
137 + sizeof(rx_pack_header_ptr->header) + sizeof(rx_pack_tail_ptr->end),
138 (ram_space_pointer->packet_rx) + sizeof(fields_packet_header));
139     if (uart_error != 0)
140     {
141         error = UART_ERROR;
142         rx_error_handler(error, ext_bus);
143         return error;
144     }
145
146     // Считываем хвост пакета
147     memcpy(&(rx_pack_ptr->packet_tail), ram_space_pointer->packet_rx + (rx_pack_header_ptr->packet_length) -
148 sizeof(rx_pack_tail_ptr->checksum) + sizeof(rx_pack_header_ptr->header), sizeof(rx_pack_ptr->packet_tail));
149
150     // Вторая обязательная проверка - последние 2 байта должны быть 0xAA
151     if (rx_pack_tail_ptr->end != PACKET_TAIL)
152     {
153         error = PACKET_ERROR;
154         rx_error_handler(error, ext_bus);
155         return error;
156     }
157
158     current_rx_packet++;
159 }
160
161 // После того, как распознали последний принятый пакет, продолжаем его обработку
162 // Распознавание сервисного байта УМ
163 memcpy(&(ram_space_pointer->service_byte_um), &(rx_pack_header_ptr->service_byte),
164 sizeof(rx_pack_header_ptr->service_byte));
165 // Обработка сервисного байта УМ
166 um_service_byte_handler(ext_bus);
167 // Считываем массив команд (команда - данные)
168 uint16_t buffer_offset = 0; // перемещение по буферу
169 for (uint32_t i = 0; i < (rx_pack_header_ptr->cmd_number); i++)
170 {
171     // Считываем заголовок - команда, результат, данные
172     memcpy(&(rx_pack_cmd_with_data_ptr + i)->header, ram_space_pointer->packet_rx + sizeof(rx_pack_ptr-
173 >packet_header) + buffer_offset, sizeof((rx_pack_cmd_with_data_ptr + i)->header));
174     // Анализ длины команды
175     if ((rx_pack_cmd_with_data_ptr + i)->header.length > UART_BUFFER_SIZE)
176     {
177         error = PACKET_ERROR;
178         return error;
179     }
180     // Считываем указатель на данные
181     rx_pack_cmd_with_data_ptr + i->data = ram_space_pointer->packet_rx + sizeof(rx_pack_ptr-
182 >packet_header) + buffer_offset + sizeof((rx_pack_cmd_with_data_ptr + i)->header);
183     buffer_offset += (rx_pack_cmd_with_data_ptr + i)->header.length;
184 }
185
186 // Вычисление контрольной суммы и сравнение ее с той, что в телеграмме
187 uint32_t real_checksum = crc32((ram_space_pointer->packet_rx) + sizeof(rx_pack_header_ptr->header),
188 rx_pack_header_ptr->packet_length - sizeof(rx_pack_tail_ptr->checksum));

```

```

181 if (real_checksum != (rx_pack_tail_ptr->checksum))
182 {
183     error = CRC_ERROR;
184     rx_error_handler(error, ext_bus);
185     return error;
186 }
187
188 // Проверка адресации
189 if(rx_pack_header_ptr->receiver_addr != ram_space_pointer-
>common_ram_register_space.PLC_PMAAddr.module_addr)
190 {
191     error = PM_ADDR_ERROR;
192     return error;
193 }
194
195 // Если нет ошибок
196 error = NO_ERROR;
197 rx_error_handler(error, ext_bus);
198
199 return error;
200 }

```

#### 4.22.2.5 rx\_error\_handler()

```

void rx_error_handler (
    protocol_error error,
    uint8_t ext_bus )

```

Обрабатывает ошибки приема пакетов

Аргументы

error	- Код ошибки
ext_bus	- Номер шины

```

467 {
468     common_ram_registers *common_ram_reg_space_ptr = &ram_space_pointer->common_ram_register_space;
469     //указатель на область памяти внешнего ОЗУ с общими регистрами
470     switch ((uint8_t)error)
471     {
472         case NO_ERROR:
473             SET_LED_OK_WORK()
474             RESET_LED_ERROR_WORK()
475             switch (ext_bus)
476             {
477                 case 1:
478                     common_ram_reg_space_ptr->PLC_CorrPackToDevice_B1++; // Увеличиваем счетчик
479                     корректно принятых пакетов
480                     common_ram_reg_space_ptr->PLC_ErrPackToDevice_B1 = 0; // Кол-во поврежденных
481                     пакетов подряд
482                     ram_space_pointer->service_byte_pm.fail_bus_1 = 0; // Снятие в сервисном байте ПИМ бита
483                     неисправности шины
484                     common_ram_reg_space_ptr->PLC_BusDefect_B1.many_fail_packet = 0; // Снятие в
485                     регистре неисправности шины бита "кол-во битых пакетов больше установленного"
486                     common_ram_reg_space_ptr->PLC_BusDefect_B1.fail_timeout = 0; // Снятие в регистре
487                     неисправности шины бита "неисправность по таймауту"
488                     break;
489                 case 2:
490                     common_ram_reg_space_ptr->PLC_CorrPackToDevice_B2++; // Увеличиваем счетчик
491                     корректно принятых пакетов
492                     common_ram_reg_space_ptr->PLC_ErrPackToDevice_B2 = 0; // Кол-во поврежденных
493                     пакетов подряд
494                     ram_space_pointer->service_byte_pm.fail_bus_2 = 0; // Снятие в сервисном байте ПИМ
495                     бита неисправности шины
496                     common_ram_reg_space_ptr->PLC_BusDefect_B2.many_fail_packet = 0; // Снятие в
497                     регистре неисправности шины бита "кол-во битых пакетов больше установленного"
498                     common_ram_reg_space_ptr->PLC_BusDefect_B2.fail_timeout = 0; // Снятие в регистре
499                     неисправности шины бита "неисправность по таймауту"
500                     break;
501                 default:

```

```

494             break;
495         }
496         break;
497     case UART_ERROR:
498         RESET_LED_OK_WORK()
499         SET_LED_ERROR_WORK()
500     switch (ext_bus)
501     {
502     case 1:
503         ram_space_pointer->service_byte_pm.fail_bus_1 = 1; // Запись в сервисный байт ПМ
504         бита неисправности шины, если превышен таймаут
505         common_ram_reg_space_ptr->PLC_BusDefect_B1.fail_timeout = 1; // Запись в регистр
506         неисправности шины бита "неисправность по таймауту"
507         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT; // Снятие
508         инициализации
509         break;
510     case 2:
511         ram_space_pointer->service_byte_pm.fail_bus_2 = 1; // Запись в сервисный байт ПМ
512         бита неисправности шины, если превышен таймаут
513         common_ram_reg_space_ptr->PLC_BusDefect_B2.fail_timeout = 1; // Запись в регистр
514         неисправности шины бита "неисправность по таймауту"
515         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT; // Снятие
516         инициализации
517         break;
518     default:
519         break;
520     }
521     break;
522 case CRC_ERROR: case PACKET_ERROR:
523     RESET_LED_OK_WORK()
524     SET_LED_ERROR_WORK()
525     switch (ext_bus)
526     {
527     case 1:
528         common_ram_reg_space_ptr->PLC_ErrPackToDevice_B1++; // Кол-во поврежденных
529         пакетов подряд
530         if (common_ram_reg_space_ptr->PLC_ErrPackToDevice_B1 >=
531             common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_NumCrcErrorsForDefect_B1)
532         {
533             ram_space_pointer->service_byte_pm.fail_bus_1 = 1; // Запись в сервисный байт
534             ПМ бита неисправности шины, если кол-во подряд поврежденных пакетов больше установленного
535             common_ram_reg_space_ptr->PLC_BusDefect_B1.many_fail_packet = 1; // Запись в
536             регистр неисправности шины бита "кол-во битых пакетов больше установленного"
537             common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT; //
538             Снятие инициализации
539             }
540             break;
541     case 2:
542         common_ram_reg_space_ptr->PLC_ErrPackToDevice_B2++; // Кол-во поврежденных
543         пакетов подряд
544         if (common_ram_reg_space_ptr->PLC_ErrPackToDevice_B2 >=
545             common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_NumCrcErrorsForDefect_B2)
546         {
547             ram_space_pointer->service_byte_pm.fail_bus_2 = 1; // Запись в сервисный байт
548             ПМ бита неисправности шины, если кол-во подряд поврежденных пакетов больше установленного
549             common_ram_reg_space_ptr->PLC_BusDefect_B2.many_fail_packet = 1; // Запись в
550             регистр неисправности шины бита "кол-во битых пакетов больше установленного"
551             common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT; //
552             Снятие инициализации
553             }
554             break;
555     default:
556         break;
557     }
558     break;
559 }
560 }

```

#### 4.22.2.6 transmit\_packet()

protocol\_error transmit\_packet (

```
uart_n * uart_struct,
uint8_t ext_bus )
```

Отправляет пакет данных

Аргументы

*uart_struct	- Выбранный UART МК
ext_bus	- Номер шины

Возвращает

Код ошибки protocol\_error

```
20 {
21     protocol_error error;
22     fields_packet *tx_pack_ptr = &ram_space_pointer->tx_packet_struct; // Указатель на структуру с отправляемым
    пакетом
23     fields_cmd *tx_pack_cmd_with_data_ptr = &ram_space_pointer->tx_packet_struct.cmd_with_data[0]; //
    Указатель на массив команд с данными для принимаемого пакета
24     fields_packet_header *tx_pack_header_ptr = &ram_space_pointer->tx_packet_struct.packet_header; // Указатель
    на заголовок принимаемого пакета
25     fields_packet_tail *tx_pack_tail_ptr = &ram_space_pointer->tx_packet_struct.packet_tail; // Указатель на хвост
    принимаемого пакета
26
27     // Записываем в буфер заголовок пакета
28     memcpy(&(ram_space_pointer->packet_tx), &(tx_pack_ptr->packet_header), sizeof(tx_pack_ptr->packet_header));
29     // Записываем в буфер массив команд (команда - данные)
30     uint16_t buffer_offset = 0; //перемещение по буферу
31     for (uint32_t i = 0; i < (tx_pack_header_ptr->cmd_number); i++)
32     {
33         // Записываем команда, результат, данные
34         memcpy((ram_space_pointer->packet_tx) + sizeof(tx_pack_ptr->packet_header) + buffer_offset,
            &((tx_pack_cmd_with_data_ptr + i)->header), sizeof((tx_pack_cmd_with_data_ptr + i)->header));
35         memcpy((ram_space_pointer->packet_tx) + sizeof(tx_pack_ptr->packet_header) + buffer_offset +
            sizeof(fields_cmd_header), (tx_pack_cmd_with_data_ptr + i)->data,
36             ((tx_pack_cmd_with_data_ptr + i)->header.length) - sizeof(fields_cmd_header));
37         buffer_offset += (tx_pack_cmd_with_data_ptr + i)->header.length;
38     }
39
40     // Вычисление контрольной суммы
41     tx_pack_tail_ptr->checksum = crc32((ram_space_pointer->packet_tx) + sizeof(tx_pack_header_ptr->header),
        tx_pack_header_ptr->packet_length - sizeof(tx_pack_tail_ptr->checksum));
42
43     // Записываем в буфер хвост пакета
44     memcpy((ram_space_pointer->packet_tx) + sizeof(tx_pack_ptr->packet_header) + buffer_offset, &(tx_pack_ptr-
        >packet_tail), sizeof(tx_pack_ptr->packet_tail));
45
46     // Отправляем данные по UART
47     if (uart_write(uart_struct, ram_space_pointer->packet_tx, tx_pack_header_ptr->packet_length +
        sizeof(tx_pack_header_ptr->header) + sizeof(tx_pack_tail_ptr->end)) != 0)
48     {
49         error = UART_ERROR;
50         return error;
51     }
52
53     // Если нет ошибок
54     error = NO_ERROR;
55
56     return error;
57 }
```

#### 4.22.2.7 um\_service\_byte\_handler()

```
void um_service_byte_handler (
    uint8_t ext_bus )
```

Обрабатывает сервисный байт УМ

## Аргументы

ext_bus	- Номер шины
---------	--------------

```

560 {
561     common_ram_registers *common_ram_reg_space_ptr = &ram_space_pointer->common_ram_register_space;
562     //указатель на область памяти внешнего ОЗУ с общими регистрами
563     switch (ram_space_pointer->service_byte_um.last_answer)
564     {
565         case 0:
566             switch (ext_bus)
567             {
568                 case 1:
569                     if((ram_space_pointer->service_byte_um.ready_to_control == 0) &&
570 (common_ram_reg_space_ptr->PLC_CM_State == PLC_CM_INIT_1_BUS))
571                     {
572                         // Сброс инициализации
573                         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT;
574                         ram_space_pointer->service_byte_pm.init = 0;
575                     }
576                     common_ram_reg_space_ptr->PLC_CorrPackFromDevice_B1++; // Увеличиваем счетчик
577                     // корректно отправленных пакетов
578                     break;
579                 case 2:
580                     if((ram_space_pointer->service_byte_um.ready_to_control == 0) &&
581 (common_ram_reg_space_ptr->PLC_CM_State == PLC_CM_INIT_2_BUS))
582                     {
583                         // Сброс инициализации
584                         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT;
585                         ram_space_pointer->service_byte_pm.init = 0;
586                     }
587                     common_ram_reg_space_ptr->PLC_CorrPackFromDevice_B2++; // Увеличиваем счетчик
588                     // корректно отправленных пакетов
589                     break;
590                 default:
591                     break;
592             }
593             break;
594         case 1:
595             switch (ext_bus)
596             {
597                 case 1:
598                     if((ram_space_pointer->service_byte_um.ready_to_control == 0) &&
599 (common_ram_reg_space_ptr->PLC_CM_State == PLC_CM_INIT_1_BUS))
600                     {
601                         // Сброс инициализации
602                         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT;
603                         ram_space_pointer->service_byte_pm.init = 0;
604                     }
605                     common_ram_reg_space_ptr->PLC_ErrPackFromDevice_B1++; // Увеличиваем счетчик
606                     // ошибочно отправленных пакетов
607                     break;
608                 case 2:
609                     if((ram_space_pointer->service_byte_um.ready_to_control == 0) &&
610 (common_ram_reg_space_ptr->PLC_CM_State == PLC_CM_INIT_2_BUS))
611                     {
612                         // Сброс инициализации
613                         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT;
614                         ram_space_pointer->service_byte_pm.init = 0;
615                     }
616                     common_ram_reg_space_ptr->PLC_ErrPackFromDevice_B2++; // Увеличиваем счетчик
617                     // ошибочно отправленных пакетов
618                     break;
619                 default:
620                     break;
621             }
622             break;
623         default:
624             break;
625     }

```

## 4.22.3 Переменные

#### 4.22.3.1 ram\_space\_pointer

`ram_data* ram_space_pointer [extern]`

Указатель для обращения к внешнему ОЗУ

#### 4.22.3.2 rom\_space\_pointer

`rom_data* rom_space_pointer [extern]`

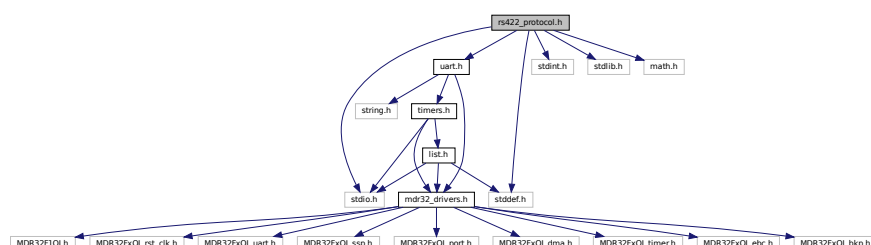
Указатель для обращения к внешнему ПЗУ

### 4.23 Файл rs422\_protocol.h

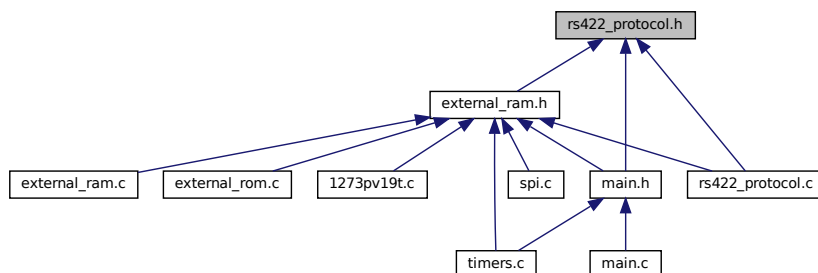
Заголовочный файл с описанием API протокола обмена данными по интерфейсу RS-422.

```
#include "uart.h"
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <stdlib.h>
#include <math.h>
```

Граф включаемых заголовочных файлов для rs422\_protocol.h:



Граф файлов, в которые включается этот файл:



## Классы

- struct `cmd_header_struct`  
Структура с заголовком для каждой команды (субпакета) внутри одного пакета
- struct `cmd_struct`  
Структура с полями данных для каждой команды (субпакета) внутри одного пакета
- struct `packet_header_struct`  
Структура с полями заголовка пакета
- struct `packet_tail_Struct`  
Структура с полями конца пакета
- struct `tx_rx_packet_struct`  
Структура пакета данных согласно утвержденному протоколу обмена данными

## Макросы

- #define `NUMBER_CMDS_IN_PACKET` 255  
Максимально возможное число команд в одном пакете
- #define `PACKET_HEAD` 0x55  
Заголовок пакета
- #define `PACKET_TAIL` 0xAAAA  
Хвост пакета
- #define `PLC_CM_UNKNOWN_STATE` 0x10  
Неизвестное состояние
- #define `PLC_CM_INIT_2_BUS` 0x09  
Модуль инициализирован, управление по шине 2.
- #define `PLC_CM_CRITICAL_FAULT` 0x06  
Критическая неисправность
- #define `PLC_CM_REMOVE_INIT` 0x05  
Управление не осуществляется (снятие инициализации)
- #define `PLC_CM_INIT_1_BUS` 0x04  
Модуль инициализирован, управление по шине 1.
- #define `PLC_CM_NOT_INIT` 0x01  
Модуль не инициализирован
- #define `TYPE_CMD` 0x00  
Код команды TYPE.
- #define `INIT_CMD` 0x01  
Код команды INIT.
- #define `READ_CMD` 0x02  
Код команды READ.
- #define `WRITE_CMD` 0x03  
Код команды WRITE.
- #define `RESET_CMD` 0x04  
Код команды RESET.
- #define `CONFIG` 0x05  
Код команды CONFIG.

## Определения типов

- typedef enum [protocol\\_errors](#) [protocol\\_error](#)  
Коды ошибок, которые могут возникать в процессе обмена данными по шине
- typedef struct [cmd\\_header\\_struct](#) [fields\\_cmd\\_header](#)  
Структура с заголовком для каждой команды (субпакета) внутри одного пакета
- typedef struct [cmd\\_struct](#) [fields\\_cmd](#)  
Структура с полями данных для каждой команды (субпакета) внутри одного пакета
- typedef struct [packet\\_header\\_struct](#) [fields\\_packet\\_header](#)  
Структура с полями заголовка пакета
- typedef struct [packet\\_tail\\_struct](#) [fields\\_packet\\_tail](#)  
Структура с полями конца пакета
- typedef struct [tx\\_rx\\_packet\\_struct](#) [fields\\_packet](#)  
Структура пакета данных согласно утвержденному протоколу обмена данными

## Перечисления

- enum [protocol\\_errors](#) {  
    [NO\\_ERROR](#), [UART\\_ERROR](#), [CRC\\_ERROR](#), [PM\\_ADDR\\_ERROR](#),  
    [PACKET\\_ERROR](#) }  
Коды ошибок, которые могут возникать в процессе обмена данными по шине

## Функции

- [protocol\\_error](#) [transmit\\_packet](#) ([uart\\_n](#) \*[uart\\_struct](#), [uint8\\_t](#) [ext\\_bus](#))  
Отправляет пакет данных
- [protocol\\_error](#) [receive\\_packet](#) ([uart\\_n](#) \*[uart\\_struct](#), [uint8\\_t](#) [ext\\_bus](#))  
Читает пакет данных
- [uint8\\_t](#) [protocol\\_do\\_cmds](#) ([uint8\\_t](#) [ext\\_bus](#))  
Выполняет требуемые команды
- [uint\\_least32\\_t](#) [crc32](#) ([uint8\\_t](#) \*[buf](#), [size\\_t](#) [len](#))  
Вычисляет контрольную сумму по алгоритму CRC32.
- void [fill\\_crc32\\_table](#) (void)  
Заполняет таблицу CRC32.
- void [rx\\_error\\_handler](#) ([protocol\\_error](#) [error](#), [uint8\\_t](#) [ext\\_bus](#))  
Обрабатывает ошибки приема пакетов
- void [um\\_service\\_byte\\_handler](#) ([uint8\\_t](#) [ext\\_bus](#))  
Обрабатывает сервисный байт УМ

### 4.23.1 Подробное описание

Заголовочный файл с описанием API протокола обмена данными по интерфейсу RS-422.

### 4.23.2 Макросы



#### 4.23.2.1 CONFIG

```
#define CONFIG 0x05
```

Код команды CONFIG.

#### 4.23.2.2 INIT\_CMD

```
#define INIT_CMD 0x01
```

Код команды INIT.

#### 4.23.2.3 NUMBER\_CMDS\_IN\_PACKET

```
#define NUMBER_CMDS_IN_PACKET 255
```

Максимально возможное число команд в одном пакете

#### 4.23.2.4 PACKET\_HEAD

```
#define PACKET_HEAD 0x55
```

Заголовок пакета

#### 4.23.2.5 PACKET\_TAIL

```
#define PACKET_TAIL 0xAAAA
```

Хвост пакета

#### 4.23.2.6 PLC\_CM\_CRITICAL\_FAULT

```
#define PLC_CM_CRITICAL_FAULT 0x06
```

Критическая неисправность

#### 4.23.2.7 PLC\_CM\_INIT\_1\_BUS

```
#define PLC_CM_INIT_1_BUS 0x04
```

Модуль инициализирован, управление по шине 1.

#### 4.23.2.8 PLC\_CM\_INIT\_2\_BUS

```
#define PLC_CM_INIT_2_BUS 0x09
```

Модуль инициализирован, управление по шине 2.

#### 4.23.2.9 PLC\_CM\_NOT\_INIT

```
#define PLC_CM_NOT_INIT 0x01
```

Модуль не инициализирован

#### 4.23.2.10 PLC\_CM\_REMOVE\_INIT

```
#define PLC_CM_REMOVE_INIT 0x05
```

Управление не осуществляется (снятие инициализации)

#### 4.23.2.11 PLC\_CM\_UNKNOWN\_STATE

```
#define PLC_CM_UNKNOWN_STATE 0x10
```

Неизвестное состояние

#### 4.23.2.12 READ\_CMD

```
#define READ_CMD 0x02
```

Код команды READ.

#### 4.23.2.13 RESET\_CMD

```
#define RESET_CMD 0x04
```

Код команды RESET.

#### 4.23.2.14 TYPE\_CMD

```
#define TYPE_CMD 0x00
```

Код команды TYPE.

#### 4.23.2.15 WRITE\_CMD

```
#define WRITE_CMD 0x03
```

Код команды WRITE.

### 4.23.3 Типы

#### 4.23.3.1 fields\_cmd

```
typedef struct cmd_struct fields_cmd
```

Структура с полями данных для каждой команды (субпакета) внутри одного пакета

#### 4.23.3.2 fields\_cmd\_header

```
typedef struct cmd_header_struct fields_cmd_header
```

Структура с заголовком для каждой команды (субпакета) внутри одного пакета

#### 4.23.3.3 fields\_packet

```
typedef struct tx_rx_packet_struct fields_packet
```

Структура пакета данных согласно утвержденному протоколу обмена данными

#### 4.23.3.4 fields\_packet\_header

```
typedef struct packet_header_struct fields_packet_header
```

Структура с полями заголовка пакета

#### 4.23.3.5 fields\_packet\_tail

```
typedef struct packet_tail_Struct fields_packet_tail
```

Структура с полями конца пакета

#### 4.23.3.6 protocol\_error

```
typedef enum protocol_errors protocol_error
```

Коды ошибок, которые могут возникать в процессе обмена данными по шине

### 4.23.4 Перечисления

#### 4.23.4.1 protocol\_errors

```
enum protocol_errors
```

Коды ошибок, которые могут возникать в процессе обмена данными по шине

Элементы перечислений

NO_ERROR	Нет ошибок
UART_ERROR	Ошибка работы UART.
CRC_ERROR	Ошибка контрольной суммы
PM_ADDR_ERROR	Ошибка адресации
PACKET_ERROR	Ошибка структуры пакета

```
44 {
45     NO_ERROR,
46     UART_ERROR,
47     CRC_ERROR,
48     PM_ADDR_ERROR,
49     PACKET_ERROR
50 } protocol_error;
```

### 4.23.5 Функции

#### 4.23.5.1 crc32()

```
uint_least32_t crc32 (
    uint8_t * buf,
    size_t len )
```

Вычисляет контрольную сумму по алгоритму CRC32.

Аргументы

*buf	- Буфер с данными для расчета контрольной суммы
len	- Длина буфера

Возвращает

Контрольная сумма

#### 4.23.5.2 fill\_crc32\_table()

```
void fill_crc32_table (
    void )
```

Заполняет таблицу CRC32.

```
452 {
453     uint_least32_t crc; int i, j;
454     for (i = 0; i < 256; i++)
455     {
456         crc = i;
457         for (j = 0; j < 8; j++)
458             crc = crc & 1 ? (crc » 1) ^ 0xEDB88320UL : crc » 1;
459
460         ram_space_pointer->crc_table[i] = crc;
461     }
462 }
```

#### 4.23.5.3 protocol\_do\_cmds()

```
uint8_t protocol_do_cmds (
    uint8_t ext_bus )
```

Выполняет требуемые команды

Аргументы

ext_bus	- Номер шины
---------	--------------

Возвращает

Код ошибки protocol\_error

```

205 {
206     common_ram_registers *common_ram_reg_space_ptr = &ram_space_pointer->common_ram_register_space; //
        // Указатель на область памяти внешнего ОЗУ с общими регистрами
207     fields_cmd *tx_pack_cmd_with_data_ptr = &ram_space_pointer->tx_packet_struct.cmd_with_data[0]; //
        // Указатель на массив команд с данными для отправляемого пакета
208     fields_cmd *rx_pack_cmd_with_data_ptr = &ram_space_pointer->rx_packet_struct.cmd_with_data[0]; //
        // Указатель на массив команд с данными для принимаемого пакета
209     fields_packet_header *tx_pack_header = &ram_space_pointer->tx_packet_struct.packet_header; // Указатель на
        // заголовок отправляемого пакета
210     fields_packet_header *rx_pack_header = &ram_space_pointer->rx_packet_struct.packet_header; // Указатель на
        // заголовок принимаемого пакета
211     fields_packet_tail *tx_pack_tail = &ram_space_pointer->tx_packet_struct.packet_tail; // Указатель на хвост
        // отправляемого пакета
212     fields_packet_tail *rx_pack_tail = &ram_space_pointer->rx_packet_struct.packet_tail; // Указатель на хвост
        // принятого пакета
213     uint32_t offset = 0; // Указатель на данные для каждой команды
214     uint16_t start_addr = 0; // Начальный адрес для чтения/записи регистров
215     uint16_t size = 0; // Кол-во байт для чтения/записи
216
217     for (uint8_t i = 0; i < rx_pack_header->cmd_number; i++)
218     {
219         switch ((rx_pack_cmd_with_data_ptr + i)->header.cmd)
220         {
221             case TYPE_CMD:
222                 // По команде TYPE кладем в поле data регистры PLC_SoftVer, PLC_Config, PLC_DeviceType,
                // PLC_SerialNumber
223                 (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
224
225                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->PLC_SoftVer),
226                     sizeof(common_ram_reg_space_ptr->PLC_SoftVer));
227                 offset += sizeof(common_ram_reg_space_ptr->PLC_SoftVer);
228                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->PLC_Config),
229                     sizeof(common_ram_reg_space_ptr->PLC_Config));
230                 offset += sizeof(common_ram_reg_space_ptr->PLC_Config);
231                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->
                >PLC_CommonRomRegs.PLC_DeviceType), sizeof(common_ram_reg_space_ptr-
                >PLC_CommonRomRegs.PLC_DeviceType));
232                 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_DeviceType);
233                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->
                >PLC_CommonRomRegs.PLC_SerialNumber), sizeof(common_ram_reg_space_ptr-
                >PLC_CommonRomRegs.PLC_SerialNumber));
234                 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
235
236                 (tx_pack_cmd_with_data_ptr + i)->header.cmd = TYPE_CMD;
237                 // TODO:разобраться для чего поле result
238                 (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
239                 (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) +
                sizeof(common_ram_reg_space_ptr->PLC_SoftVer) +
240                 sizeof(ram_space_pointer->common_ram_register_space.PLC_Config) +
241                 sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_DeviceType) +
242                 sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
243                 break;
244
245             case INIT_CMD:
246                 // По команде INIT кладем в поле data регистр PLC_SerialNumber, если выполнен ряд условий
247                 switch (ext_bus)
248                 {
249                     case 1:
250                         if((ram_space_pointer->service_byte_um.ready_to_control == 1) &&
251                             (ram_space_pointer->service_byte_pm.fail_bus_1 == 0) && (common_ram_reg_space_ptr->PLC_CM_State !=
252                             PLC_CM_INIT_2_BUS))
253                         {
254                             // Заносим инфу в сервисный байт ПМ
255                             ram_space_pointer->service_byte_pm.init = 1;
256                             ram_space_pointer->service_byte_pm.master = 0;
257                             // Заносим инфу в регистры
258                             common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_INIT_1_BUS;
259                         }
260                     else
261                     {
262                         continue;
263                     }
264                 }
265                 break;
266
267             case 2:
268                 if((ram_space_pointer->service_byte_um.ready_to_control == 1) &&
269                     (ram_space_pointer->service_byte_pm.fail_bus_2 == 0) && (common_ram_reg_space_ptr->PLC_CM_State !=
270                     PLC_CM_INIT_1_BUS))
271                 {
272                     // Заносим инфу в сервисный байт ПМ
273                     ram_space_pointer->service_byte_pm.init = 1;
274                     ram_space_pointer->service_byte_pm.master = 1;
275                     // Заносим инфу в регистры

```

```

267         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_INIT_2_BUS;
268     }
269     else
270     {
271         continue;
272     }
273     break;
274
275     default:
276         break;
277 }
278 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr-
>PLC_CommonRomRegs.PLC_SerialNumber), sizeof(common_ram_reg_space_ptr-
>PLC_CommonRomRegs.PLC_SerialNumber));
279 (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
280 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
281 (tx_pack_cmd_with_data_ptr + i)->header.cmd = INIT_CMD;
282 // TODO:разобраться для чего поле result
283 (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
284 (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) +
sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
285 break;
286
287 case READ_CMD:
288     memcpy(&start_addr, (rx_pack_cmd_with_data_ptr + i)->data, sizeof(start_addr));
289     memcpy(&size, (rx_pack_cmd_with_data_ptr + i)->data + sizeof(start_addr), sizeof(size));
290
291     // По команде READ кладем в поле data size байт начиная с start_addr
292     memcpy((ram_space_pointer->tx_data) + offset, (void*)&(ram_space_pointer->start_struct)) +
start_addr, size);
293     (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
294     offset += size;
295     (tx_pack_cmd_with_data_ptr + i)->header.cmd = READ_CMD;
296     // TODO:разобраться для чего поле result
297     (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
298     (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + size;
299     break;
300
301 case WRITE_CMD:
302     // Проверка того, что установлено соединение по выбранной шине
303     switch (ext_bus)
304     {
305         case 1:
306             if(ram_space_pointer->common_ram_register_space.PLC_CM_State !=
PLC_CM_INIT_1_BUS)
307             {
308                 continue;
309             }
310             break;
311
312         case 2:
313             if(ram_space_pointer->common_ram_register_space.PLC_CM_State !=
PLC_CM_INIT_2_BUS)
314             {
315                 continue;
316             }
317             break;
318
319         default:
320             break;
321     }
322     memcpy(&start_addr, (rx_pack_cmd_with_data_ptr + i)->data, sizeof(start_addr));
323     memcpy(&size, ((rx_pack_cmd_with_data_ptr + i)->data) + sizeof(start_addr), sizeof(size));
324     // По команде WRITE кладем по адресу start_addr size принятых байт для записи и в поле данных
кладем код команды
325     memcpy((void*)&(ram_space_pointer->start_struct)) + start_addr, ((rx_pack_cmd_with_data_ptr
+ i)->data) + sizeof(start_addr) + sizeof(size), size);
326     memset((ram_space_pointer->tx_data) + offset, WRITE_CMD, 1);
327     (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
328     offset++;
329     (tx_pack_cmd_with_data_ptr + i)->header.cmd = WRITE_CMD;
330     // TODO:разобраться для чего поле result
331     (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
332     (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + 1;
333     break;
334
335 case RESET_CMD:
336     // Проверка того, что установлено соединение по выбранной шине
337     switch (ext_bus)
338     {
339         case 1:
340             if(common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_1_BUS)
341             {
342                 continue;
343             }
344             break;
345

```

```

346         case 2:
347             if(common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_2_BUS)
348             {
349                 continue;
350             }
351             break;
352
353         default:
354             break;
355     }
356     // По команде RESET сбрасываем регистры и в поле данных кладем код команды
357     init_external_ram_space();
358     memset((ram_space_pointer->tx_data) + offset, RESET, 1);
359     (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
360     offset++;
361     (tx_pack_cmd_with_data_ptr + i)->header.cmd = RESET;
362     // TODO:разобраться для чего поле result
363     (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
364     (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + 1;
365     break;
366
367 case CONFIG:
368     // Проверка того, что установлено соединение по выбранной шине
369     switch (ext_bus)
370     {
371         case 1:
372             if(common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_1_BUS)
373             {
374                 continue;
375             }
376             break;
377
378         case 2:
379             if(common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_2_BUS)
380             {
381                 continue;
382             }
383             break;
384
385         default:
386             break;
387     }
388     // Проверка на то, что ПМ находится в сервисном режиме
389     if (common_ram_reg_space_ptr->PLC_PMAAddr.module_addr != 0x00)
390     {
391         break;
392     }
393     // По команде CONFIG регистры зеркализируются в ПЗУ и в поле данных кладем код команды
394     memcpy(&start_addr, (rx_pack_cmd_with_data_ptr + i)->data, sizeof(start_addr));
395     memcpy(&size, (rx_pack_cmd_with_data_ptr + i)->data + sizeof(start_addr), sizeof(size));
396     // Если используется внешнее ПЗУ, то записываем данные в ПЗУ
397     #ifdef ROM_IS_USED
398     //запись данных в ПЗУ
399     memcpy(&common_regs, &common_ram_reg_space_ptr->PLC_CommonRomRegs,
400 sizeof(common_regs));
401     memcpy(&mpa_regs, &ram_space_pointer->mpa_ram_register_space.AI_RomRegs, sizeof(mpa_regs));
402     ebc_init(EBC_ROM);
403     memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR, &common_regs, sizeof(common_regs));
404     memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR + sizeof(common_regs), &mpa_regs,
405 sizeof(mpa_regs));
406     ebc_init(EBC_RAM);
407     #endif
408
409     memset((ram_space_pointer->tx_data) + offset, CONFIG, 1);
410     (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
411     offset++;
412     (tx_pack_cmd_with_data_ptr + i)->header.cmd = CONFIG;
413     // TODO:разобраться для чего поле result
414     (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
415     (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + 1;
416     break;
417
418     default:
419         break;
420     }
421     tx_pack_header->packet_length += (tx_pack_cmd_with_data_ptr + i)->header.length;
422 }
423 // Заполнение остальных полей
424 tx_pack_header->header = rx_pack_header->header;
425 tx_pack_header->receiver_addr = rx_pack_header->sender_addr;
426 tx_pack_header->sender_addr = rx_pack_header->receiver_addr;
427 tx_pack_header->packet_length += sizeof(ram_space_pointer->tx_packet_struct.packet_header) -
428 sizeof(tx_pack_header->header) +
429 sizeof(tx_pack_tail->checksum);
430 memcpy(&(tx_pack_header->service_byte), &(ram_space_pointer->service_byte_pm), sizeof(ram_space_pointer-
431 >service_byte_pm));
432 tx_pack_header->cmd_number = rx_pack_header->cmd_number;

```



```

429
430     tx_pack_tail->end = rx_pack_tail->end;
431
432     return 0;
433 }

```

#### 4.23.5.4 receive\_packet()

```

protocol_error receive_packet (
    uart_n * uart_struct,
    uint8_t ext_bus )

```

Читает пакет данных

Аргументы

*uart_struct	- Выбранный UART МК
ext_bus	- Номер шины

Возвращает

Код ошибки protocol\_error

```

62 {
63     fields_packet *rx_pack_ptr = &ram_space_pointer->rx_packet_struct; // Указатель на структуру с принимаемым
        пакетом
64     fields_packet_header *rx_pack_header_ptr = &ram_space_pointer->rx_packet_struct.packet_header; // Указатель
        на заголовок принимаемого пакета
65     fields_packet_tail *rx_pack_tail_ptr = &ram_space_pointer->rx_packet_struct.packet_tail; // Указатель на
        заголовок принимаемого пакета
66     fields_cmd *rx_pack_cmd_with_data_ptr = &ram_space_pointer->rx_packet_struct.cmd_with_data[0]; //
        Указатель на массив команд с данными для принимаемого пакета
67     uint32_t current_rx_packet = 0;
68
69     // Код последней ошибки, возникшей в процессе обмена данными
70     protocol_error error;
71     // Код последней ошибки, связанной с работой блока UART МК
72     uart_errors uart_error;
73
74     uint8_t packet_head;
75
76     // Очистка всех структур данных связанных с принимаемым и отправляемым пакетом
77     memset(rx_pack_ptr, 0, sizeof(ram_space_pointer->rx_packet_struct) + sizeof(ram_space_pointer->tx_packet_struct)
        +
78         sizeof(ram_space_pointer->tx_data) + sizeof(ram_space_pointer->packet_tx) + sizeof(ram_space_pointer-
        >packet_rx));
79
80     // Здесь в цикле ловится и обрабатывается последний пакет в буфере приемника UART
81     while (1)
82     {
83         if (uart_read_pos(uart_struct) < UART_BUFFER_SIZE)
84         {
85             memcpy(&packet_head, (uint8_t*)(GET_UART_BUF_PTR + uart_read_pos(uart_struct)),
                sizeof(packet_head));
86         }
87         else
88         {
89             memcpy(&packet_head, (uint8_t*)(GET_UART_BUF_PTR), sizeof(packet_head));
90         }
91
92         // Если уже были разобраны какие то пакеты, а начала следующего пакета (0x55) в буфере нет, это означает что
        крайний разобранный пакет оказался последним и выходим из цикла
93         if ((current_rx_packet != 0) && (packet_head != PACKET_HEAD))
94         {
95             uart_clean(uart_struct);
96             break;
97         }
98
99         // Считываем первый байт и проверяем на 0x55
100         uart_error = uart_read(uart_struct, sizeof(rx_pack_header_ptr->header), ram_space_pointer->packet_rx);

```

```

101     if (uart_error != 0)
102     {
103         error = UART_ERROR;
104         rx_error_handler(error, ext_bus);
105         return error;
106     }
107
108     memcpy(&(rx_pack_header_ptr->header), ram_space_pointer->packet_rx, sizeof(rx_pack_header_ptr->header));
109     if (rx_pack_header_ptr->header != PACKET_HEAD)
110     {
111         error = PACKET_ERROR;
112         rx_error_handler(error, ext_bus);
113         return error;
114     }
115
116     // Считываем заголовок телеграммы
117     uart_error = uart_read(uart_struct, sizeof(rx_pack_ptr->packet_header) - sizeof(rx_pack_header_ptr->header),
118 (ram_space_pointer->packet_rx) + sizeof(rx_pack_header_ptr->header));
119     if (uart_error != 0)
120     {
121         error = UART_ERROR;
122         rx_error_handler(error, ext_bus);
123         return error;
124     }
125     memcpy(rx_pack_header_ptr, ram_space_pointer->packet_rx, sizeof(rx_pack_ptr->packet_header));
126
127     // Анализ длины пакета
128     if (rx_pack_header_ptr->packet_length > UART_BUFFER_SIZE)
129     {
130         error = PACKET_ERROR;
131         rx_error_handler(error, ext_bus);
132         return error;
133     }
134
135     // Считываем весь пакет вычисленной длины
136     uart_error = uart_read(uart_struct, (rx_pack_header_ptr->packet_length) - sizeof(rx_pack_ptr->packet_header)
137 + sizeof(rx_pack_header_ptr->header) + sizeof(rx_pack_tail_ptr->end),
138 (ram_space_pointer->packet_rx) + sizeof(fields_packet_header));
139     if (uart_error != 0)
140     {
141         error = UART_ERROR;
142         rx_error_handler(error, ext_bus);
143         return error;
144     }
145
146     // Считываем хвост пакета
147     memcpy(&(rx_pack_ptr->packet_tail), ram_space_pointer->packet_rx + (rx_pack_header_ptr->packet_length) -
148 sizeof(rx_pack_tail_ptr->checksum) + sizeof(rx_pack_header_ptr->header), sizeof(rx_pack_ptr->packet_tail));
149
150     // Вторая обязательная проверка - последние 2 байта должны быть 0xAA
151     if (rx_pack_tail_ptr->end != PACKET_TAIL)
152     {
153         error = PACKET_ERROR;
154         rx_error_handler(error, ext_bus);
155         return error;
156     }
157
158     current_rx_packet++;
159 }
160
161 // После того, как распознали последний принятый пакет, продолжаем его обработку
162 // Распознавание сервисного байта УМ
163 memcpy(&(ram_space_pointer->service_byte_um), &(rx_pack_header_ptr->service_byte),
164 sizeof(rx_pack_header_ptr->service_byte));
165 // Обработка сервисного байта УМ
166 um_service_byte_handler(ext_bus);
167
168 // Считываем массив команд (команда - данные)
169 uint16_t buffer_offset = 0; // перемещение по буферу
170 for (uint32_t i = 0; i < (rx_pack_header_ptr->cmd_number); i++)
171 {
172     // Считываем заголовок - команда, результат, данные
173     memcpy(&(rx_pack_cmd_with_data_ptr + i)->header, ram_space_pointer->packet_rx + sizeof(rx_pack_ptr-
174 >packet_header) + buffer_offset, sizeof((rx_pack_cmd_with_data_ptr + i)->header));
175
176     // Анализ длины команды
177     if ((rx_pack_cmd_with_data_ptr + i)->header.length > UART_BUFFER_SIZE)
178     {
179         error = PACKET_ERROR;
180         rx_error_handler(error, ext_bus);
181         return error;
182     }
183
184     // Считываем указатель на данные
185     rx_pack_cmd_with_data_ptr + i->data = ram_space_pointer->packet_rx + sizeof(rx_pack_ptr-
186 >packet_header) + buffer_offset + sizeof((rx_pack_cmd_with_data_ptr + i)->header);
187     buffer_offset += (rx_pack_cmd_with_data_ptr + i)->header.length;
188 }
189
190 // Вычисление контрольной суммы и сравнение ее с той, что в телеграмме
191 uint32_t real_checksum = crc32((ram_space_pointer->packet_rx) + sizeof(rx_pack_header_ptr->header),
192 rx_pack_header_ptr->packet_length - sizeof(rx_pack_tail_ptr->checksum));

```

```

181 if (real_checksum != (rx_pack_tail_ptr->checksum))
182 {
183     error = CRC_ERROR;
184     rx_error_handler(error, ext_bus);
185     return error;
186 }
187
188 // Проверка адресации
189 if(rx_pack_header_ptr->receiver_addr != ram_space_pointer-
>common_ram_register_space.PLC_PMAAddr.module_addr)
190 {
191     error = PM_ADDR_ERROR;
192     return error;
193 }
194
195 // Если нет ошибок
196 error = NO_ERROR;
197 rx_error_handler(error, ext_bus);
198
199 return error;
200 }

```

#### 4.23.5.5 rx\_error\_handler()

```

void rx_error_handler (
    protocol_error error,
    uint8_t ext_bus )

```

Обрабатывает ошибки приема пакетов

Аргументы

error	- Код ошибки
ext_bus	- Номер шины

```

467 {
468     common_ram_registers *common_ram_reg_space_ptr = &ram_space_pointer->common_ram_register_space;
469     //указатель на область памяти внешнего ОЗУ с общими регистрами
470     switch ((uint8_t)error)
471     {
472         case NO_ERROR:
473             SET_LED_OK_WORK()
474             RESET_LED_ERROR_WORK()
475             switch (ext_bus)
476             {
477                 case 1:
478                     common_ram_reg_space_ptr->PLC_CorrPackToDevice_B1++; // Увеличиваем счетчик
479                     корректно принятых пакетов
480                     common_ram_reg_space_ptr->PLC_ErrPackToDevice_B1 = 0; // Кол-во поврежденных
481                     пакетов подряд
482                     ram_space_pointer->service_byte_pm.fail_bus_1 = 0; // Снятие в сервисном байте ПИМ бита
483                     неисправности шины
484                     common_ram_reg_space_ptr->PLC_BusDefect_B1.many_fail_packet = 0; // Снятие в
485                     регистре неисправности шины бита "кол-во битых пакетов больше установленного"
486                     common_ram_reg_space_ptr->PLC_BusDefect_B1.fail_timeout = 0; // Снятие в регистре
487                     неисправности шины бита "неисправность по таймауту"
488                     break;
489                 case 2:
490                     common_ram_reg_space_ptr->PLC_CorrPackToDevice_B2++; // Увеличиваем счетчик
491                     корректно принятых пакетов
492                     common_ram_reg_space_ptr->PLC_ErrPackToDevice_B2 = 0; // Кол-во поврежденных
493                     пакетов подряд
494                     ram_space_pointer->service_byte_pm.fail_bus_2 = 0; // Снятие в сервисном байте ПИМ
495                     бита неисправности шины
496                     common_ram_reg_space_ptr->PLC_BusDefect_B2.many_fail_packet = 0; // Снятие в
497                     регистре неисправности шины бита "кол-во битых пакетов больше установленного"
498                     common_ram_reg_space_ptr->PLC_BusDefect_B2.fail_timeout = 0; // Снятие в регистре
499                     неисправности шины бита "неисправность по таймауту"
500                     break;
501                 default:

```

```

494             break;
495         }
496         break;
497     case UART_ERROR:
498         RESET_LED_OK_WORK()
499         SET_LED_ERROR_WORK()
500     switch (ext_bus)
501     {
502     case 1:
503         ram_space_pointer->service_byte_pm.fail_bus_1 = 1; // Запись в сервисный байт ПМ
504         бита неисправности шины, если превышен таймаут
505         common_ram_reg_space_ptr->PLC_BusDefect_B1.fail_timeout = 1; // Запись в регистр
506         неисправности шины бита "неисправность по таймауту"
507         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT; // Снятие
508         инициализации
509         break;
510     case 2:
511         ram_space_pointer->service_byte_pm.fail_bus_2 = 1; // Запись в сервисный байт ПМ
512         бита неисправности шины, если превышен таймаут
513         common_ram_reg_space_ptr->PLC_BusDefect_B2.fail_timeout = 1; // Запись в регистр
514         неисправности шины бита "неисправность по таймауту"
515         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT; // Снятие
516         инициализации
517         break;
518     default:
519         break;
520     }
521     break;
522 case CRC_ERROR: case PACKET_ERROR:
523     RESET_LED_OK_WORK()
524     SET_LED_ERROR_WORK()
525     switch (ext_bus)
526     {
527     case 1:
528         common_ram_reg_space_ptr->PLC_ErrPackToDevice_B1++; // Кол-во поврежденных
529         пакетов подряд
530         if (common_ram_reg_space_ptr->PLC_ErrPackToDevice_B1 >=
531             common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_NumCrcErrorsForDefect_B1)
532         {
533             ram_space_pointer->service_byte_pm.fail_bus_1 = 1; // Запись в сервисный байт
534             ПМ бита неисправности шины, если кол-во подряд поврежденных пакетов больше установленного
535             common_ram_reg_space_ptr->PLC_BusDefect_B1.many_fail_packet = 1; // Запись в
536             регистр неисправности шины бита "кол-во битых пакетов больше установленного"
537             common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT; //
538             Снятие инициализации
539             }
540             break;
541     case 2:
542         common_ram_reg_space_ptr->PLC_ErrPackToDevice_B2++; // Кол-во поврежденных
543         пакетов подряд
544         if (common_ram_reg_space_ptr->PLC_ErrPackToDevice_B2 >=
545             common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_NumCrcErrorsForDefect_B2)
546         {
547             ram_space_pointer->service_byte_pm.fail_bus_2 = 1; // Запись в сервисный байт
548             ПМ бита неисправности шины, если кол-во подряд поврежденных пакетов больше установленного
549             common_ram_reg_space_ptr->PLC_BusDefect_B2.many_fail_packet = 1; // Запись в
550             регистр неисправности шины бита "кол-во битых пакетов больше установленного"
551             common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT; //
552             Снятие инициализации
553             }
554             break;
555     default:
556         break;
557     }
558     break;
559 }
560 }

```

#### 4.23.5.6 transmit\_packet()

protocol\_error transmit\_packet (

```
uart_n * uart_struct,
uint8_t ext_bus )
```

Отправляет пакет данных

Аргументы

*uart_struct	- Выбранный UART МК
ext_bus	- Номер шины

Возвращает

Код ошибки protocol\_error

```
20 {
21     protocol_error error;
22     fields_packet *tx_pack_ptr = &ram_space_pointer->tx_packet_struct; // Указатель на структуру с отправляемым
        пакетом
23     fields_cmd *tx_pack_cmd_with_data_ptr = &ram_space_pointer->tx_packet_struct.cmd_with_data[0]; //
        Указатель на массив команд с данными для принимаемого пакета
24     fields_packet_header *tx_pack_header_ptr = &ram_space_pointer->tx_packet_struct.packet_header; // Указатель
        на заголовок принимаемого пакета
25     fields_packet_tail *tx_pack_tail_ptr = &ram_space_pointer->tx_packet_struct.packet_tail; // Указатель на хвост
        принимаемого пакета
26
27     // Записываем в буфер заголовок пакета
28     memcpy(&(ram_space_pointer->packet_tx), &(tx_pack_ptr->packet_header), sizeof(tx_pack_ptr->packet_header));
29     // Записываем в буфер массив команд (команда - данные)
30     uint16_t buffer_offset = 0; //перемещение по буферу
31     for (uint32_t i = 0; i < (tx_pack_header_ptr->cmd_number); i++)
32     {
33         // Записываем команда, результат, данные
34         memcpy((ram_space_pointer->packet_tx) + sizeof(tx_pack_ptr->packet_header) + buffer_offset,
            &((tx_pack_cmd_with_data_ptr + i)->header), sizeof((tx_pack_cmd_with_data_ptr + i)->header));
35         memcpy((ram_space_pointer->packet_tx) + sizeof(tx_pack_ptr->packet_header) + buffer_offset +
            sizeof(fields_cmd_header), (tx_pack_cmd_with_data_ptr + i)->data,
36             ((tx_pack_cmd_with_data_ptr + i)->header.length) - sizeof(fields_cmd_header));
37         buffer_offset += (tx_pack_cmd_with_data_ptr + i)->header.length;
38     }
39
40     // Вычисление контрольной суммы
41     tx_pack_tail_ptr->checksum = crc32((ram_space_pointer->packet_tx) + sizeof(tx_pack_header_ptr->header),
        tx_pack_header_ptr->packet_length - sizeof(tx_pack_tail_ptr->checksum));
42
43     // Записываем в буфер хвост пакета
44     memcpy((ram_space_pointer->packet_tx) + sizeof(tx_pack_ptr->packet_header) + buffer_offset, &(tx_pack_ptr-
        >packet_tail), sizeof(tx_pack_ptr->packet_tail));
45
46     // Отправляем данные по UART
47     if (uart_write(uart_struct, ram_space_pointer->packet_tx, tx_pack_header_ptr->packet_length +
        sizeof(tx_pack_header_ptr->header) + sizeof(tx_pack_tail_ptr->end)) != 0)
48     {
49         error = UART_ERROR;
50         return error;
51     }
52
53     // Если нет ошибок
54     error = NO_ERROR;
55
56     return error;
57 }
```

#### 4.23.5.7 um\_service\_byte\_handler()

```
void um_service_byte_handler (
    uint8_t ext_bus )
```

Обрабатывает сервисный байт УМ

## Аргументы

ext_bus	- Номер шины
---------	--------------

```

560 {
561     common_ram_registers *common_ram_reg_space_ptr = &ram_space_pointer->common_ram_register_space;
562     //указатель на область памяти внешнего ОЗУ с общими регистрами
563     switch (ram_space_pointer->service_byte_um.last_answer)
564     {
565         case 0:
566             switch (ext_bus)
567             {
568                 case 1:
569                     if((ram_space_pointer->service_byte_um.ready_to_control == 0) &&
570 (common_ram_reg_space_ptr->PLC_CM_State == PLC_CM_INIT_1_BUS))
571                     {
572                         // Сброс инициализации
573                         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT;
574                         ram_space_pointer->service_byte_pm.init = 0;
575                     }
576                     common_ram_reg_space_ptr->PLC_CorrPackFromDevice_B1++; // Увеличиваем счетчик
577                     корректно отправленных пакетов
578                     break;
579                 case 2:
580                     if((ram_space_pointer->service_byte_um.ready_to_control == 0) &&
581 (common_ram_reg_space_ptr->PLC_CM_State == PLC_CM_INIT_2_BUS))
582                     {
583                         // Сброс инициализации
584                         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT;
585                         ram_space_pointer->service_byte_pm.init = 0;
586                     }
587                     common_ram_reg_space_ptr->PLC_CorrPackFromDevice_B2++; // Увеличиваем счетчик
588                     корректно отправленных пакетов
589                     break;
590                 default:
591                     break;
592             }
593             break;
594         case 1:
595             switch (ext_bus)
596             {
597                 case 1:
598                     if((ram_space_pointer->service_byte_um.ready_to_control == 0) &&
599 (common_ram_reg_space_ptr->PLC_CM_State == PLC_CM_INIT_1_BUS))
600                     {
601                         // Сброс инициализации
602                         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT;
603                         ram_space_pointer->service_byte_pm.init = 0;
604                     }
605                     common_ram_reg_space_ptr->PLC_ErrPackFromDevice_B1++; // Увеличиваем счетчик
606                     ошибочно отправленных пакетов
607                     break;
608                 case 2:
609                     if((ram_space_pointer->service_byte_um.ready_to_control == 0) &&
610 (common_ram_reg_space_ptr->PLC_CM_State == PLC_CM_INIT_2_BUS))
611                     {
612                         // Сброс инициализации
613                         common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_REMOVE_INIT;
614                         ram_space_pointer->service_byte_pm.init = 0;
615                     }
616                     common_ram_reg_space_ptr->PLC_ErrPackFromDevice_B2++; // Увеличиваем счетчик
617                     ошибочно отправленных пакетов
618                     break;
619                 default:
620                     break;
621             }
622             break;
623         default:
624             break;
625     }

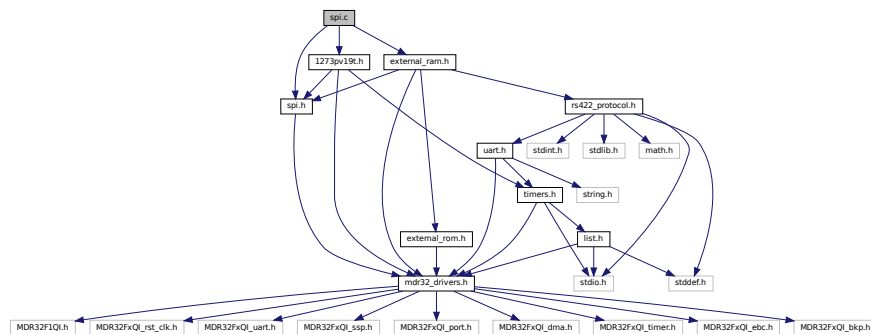
```

## 4.24 Файл spi.c

Файл с реализацией API для работы с SPI.

```
#include "spi.h"
#include "1273pv19t.h"
#include "external_ram.h"
```

Граф включаемых заголовочных файлов для spi.c:



### Функции

- void `spi_gpio_config` (void)  
Конфигурирует выводы МК для SPI.
- void `spi_init` (spi\_n \*spi\_struct)  
Инициализирует выбранный SPIn.
- void `spi_transmit_halfword` (spi\_n \*spi\_struct, uint16\_t half\_word)  
Отправляет полуслово по SPIn.
- void `spi_transmit_message` (spi\_n \*spi\_struct, uint16\_t message[], uint32\_t length)  
Отправляет массив полуслов по SPIn.
- uint16\_t `spi_receive_halfword` (spi\_n \*spi\_struct)  
Принимает полуслово по SPIn.
- void `spi_clean_fifo_rx_buf` (spi\_n \*spi\_struct)  
Очищает FIFO буфер приемника SPIn.
- void `dma_spi_rx_init` (spi\_n \*spi\_struct)  
Инициализирует n-ый канал DMA на запрос от приемника SPIn.

### Переменные

- `spi_n spi_1`  
Структура с конфигурационными параметрами блока SPI1 МК
- `spi_n spi_2`  
Структура с конфигурационными параметрами блока SPI2 МК

#### 4.24.1 Подробное описание

Файл с реализацией API для работы с SPI.

## 4.24.2 Функции

### 4.24.2.1 dma\_spi\_rx\_init()

```
void dma_spi_rx_init (
    spi_n * spi_struct )
```

Инициализирует n-ый канал DMA на запрос от приемника SPIn.

Аргументы

*spi_struct	- Выбранный SPI
-------------	-----------------

```
129 {
130
131     DMA_StructInit(&spi_struct->spi_dma_ch.DMA_Channel_SPI_RX);
132     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_SourceBaseAddr = (uint32_t)(&(spi_struct->SSPx-
133     >DR));
134     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_DestBaseAddr = (uint32_t)(spi_struct->buffer);
135     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_CycleSize = 1;
136     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_SourceIncSize = DMA_SourceIncNo;
137     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_DestIncSize = DMA_DestIncHalfword;
138     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
139     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_NumContinuous = DMA_Transfers_1;
140     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_SourceProtCtrl = DMA_SourcePrivileged;
141     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_DestProtCtrl = DMA_DestPrivileged;
142     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_Mode = DMA_Mode_Basic;
143
144     // Задать структуру канала
145     spi_struct->spi_dma_ch.DMA_Channel_SPI_RX.DMA_PriCtrlData = &spi_struct-
146     >spi_dma_ch.DMA_InitStructure_SPI_RX;
147     spi_struct->spi_dma_ch.DMA_Channel_SPI_RX.DMA_Priority = DMA_Priority_High;
148     spi_struct->spi_dma_ch.DMA_Channel_SPI_RX.DMA_UseBurst = DMA_BurstClear;
149     spi_struct->spi_dma_ch.DMA_Channel_SPI_RX.DMA_SelectDataStructure = DMA_CTRL_DATA_PRIMARY;
150
151     // Инициализировать канал
152     DMA_Init(spi_struct->spi_dma_ch.dma_channel, &spi_struct->spi_dma_ch.DMA_Channel_SPI_RX);
153
154     MDR_DMA->CHNL_REQ_MASK_CLR = 1 « spi_struct->spi_dma_ch.dma_channel;
155     MDR_DMA->CHNL_USEBURST_CLR = 1 « spi_struct->spi_dma_ch.dma_channel;
156
157     SSP_DMACmd(spi_struct->SSPx, SSP_DMA_RXE, DISABLE);
158     // Разрешить работу DMA с SPI
159     DMA_Cmd (spi_struct->spi_dma_ch.dma_channel, DISABLE);
160
161     //NVIC_SetPriority (DMA_IRQn, 0);
162     //NVIC_EnableIRQ(DMA_IRQn);
163 }
```

### 4.24.2.2 spi\_clean\_fifo\_rx\_buf()

```
void spi_clean_fifo_rx_buf (
    spi_n * spi_struct )
```

Очищает FIFO буфер приемника SPIn.

Аргументы

*spi_struct	- Выбранный SPI
-------------	-----------------

```
118 {
```



```

119     uint16_t a;
120     while(SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_RNE) == SET)
121     {
122         a = spi_struct->SSPx->DR;
123     }
124 }

```

#### 4.24.2.3 spi\_gpio\_config()

```

void spi_gpio_config (
    void )

```

Конфигурирует выводы МК для SPI.

```

19 {
20     // Включение тактирования портов
21     RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK|RST_CLK_PCLK_PORTD, ENABLE);
22
23     // Инициализация портов SSP1
24     PORT_InitTypeDef GPIO_init_struct_SPI1;
25
26     PORT_StructInit(&GPIO_init_struct_SPI1);
27     GPIO_init_struct_SPI1.PORT_FUNC = PORT_FUNC_ALTER;
28     GPIO_init_struct_SPI1.PORT_MODE = PORT_MODE_DIGITAL;
29     GPIO_init_struct_SPI1.PORT_SPEED = PORT_SPEED_MAXFAST;
30     // Инициализация вывода SCK
31     GPIO_init_struct_SPI1.PORT_Pin = PIN_SSP1_SCK;
32     GPIO_init_struct_SPI1.PORT_OE = PORT_OE_IN;
33     PORT_Init(PORT_SSP1, &GPIO_init_struct_SPI1);
34     // Инициализация вывода SSP_RX
35     GPIO_init_struct_SPI1.PORT_Pin = PIN_SSP1_RX;
36     GPIO_init_struct_SPI1.PORT_OE = PORT_OE_IN;
37     PORT_Init(PORT_SSP1, &GPIO_init_struct_SPI1);
38     // Инициализация вывода SSP_TX
39     GPIO_init_struct_SPI1.PORT_Pin = PIN_SSP1_TX;
40     GPIO_init_struct_SPI1.PORT_OE = PORT_OE_OUT;
41     PORT_Init(PORT_SSP1, &GPIO_init_struct_SPI1);
42     PORT_ResetBits(PORT_SSP1, PIN_SSP1_TX);
43     // Инициализация вывода SS (вход SDIFS)
44     GPIO_init_struct_SPI1.PORT_Pin = PIN_SSP1_SS;
45     GPIO_init_struct_SPI1.PORT_FUNC = PORT_FUNC_ALTER;
46     GPIO_init_struct_SPI1.PORT_OE = PORT_OE_IN;
47     PORT_Init(PORT_SSP1, &GPIO_init_struct_SPI1);
48 }

```

#### 4.24.2.4 spi\_init()

```

void spi_init (
    spi_n * spi_struct )

```

Инициализирует выбранный SPIn.

Аргументы

*spi_struct	- Выбранный SPI для инициализации
-------------	-----------------------------------

```

53 {
54     spi_gpio_config();
55
56     // Структура для инициализации SPI
57     SSP_InitTypeDef SSP_InitStruct;
58
59     // Включение тактирования SPI
60     RST_CLK_PCLKcmd(spi_struct->RST_CLK_PCLK_SPIn, ENABLE);
61
62     SSP_StructInit(&SSP_InitStruct);

```

```

63
64     SSP_BRGInit(spi_struct->SSPx, SSP_HCLKdiv1);
65
66     SSP_InitStruct.SSP_WordLength = spi_struct->SPI.SSP_WordLength;
67     SSP_InitStruct.SSP_Mode = spi_struct->SPI.SSP_Mode;
68     SSP_InitStruct.SSP_SPH = spi_struct->SPI.SSP_SPH;
69     SSP_InitStruct.SSP_FRF = spi_struct->SPI.SSP_FRF;
70     SSP_InitStruct.SSP_HardwareFlowControl = spi_struct->SPI.SSP_HardwareFlowControl;
71     //SSP_InitStruct.SSP_SCR = 0x10;
72     SSP_InitStruct.SSP_CPSDVR = spi_struct->SPI.SSP_CPSDVR; // Частота обмена 2МГц
73     SSP_Init(spi_struct->SSPx, &SSP_InitStruct);
74
75     NVIC_DisableIRQ(spi_struct->IRQn);
76     // Выбор источников прерываний (прием и передача данных)
77     SSP_ITConfig (spi_struct->SSPx, SSP_IT_RX, DISABLE);
78
79     SSP_Cmd(spi_struct->SSPx, ENABLE);
80 }

```

#### 4.24.2.5 spi\_receive\_halfword()

```

uint16_t spi_receive_halfword (
    spi_n * spi_struct )

```

Принимает полуслово по SPIн.

Аргументы

*spi_struct	- Выбранный SPI
-------------	-----------------

```

105 {
106     uint16_t tmpVar;
107     // Обработка прерывания от приемника данных
108     while (SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_RNE) != SET) {} // Ждем, пока появится байт
109     // Получаем данные
110     tmpVar = SSP_ReceiveData(spi_struct->SSPx);
111
112     return tmpVar;
113 }

```

#### 4.24.2.6 spi\_transmit\_halfword()

```

void spi_transmit_halfword (
    spi_n * spi_struct,
    uint16_t half_word )

```

Отправляет полуслово по SPIн.

Аргументы

*spi_struct	- Выбранный SPI
half_word	- Полуслово для отправки

```

85 {
86     SSP_SendData(spi_struct->SSPx, half_word);
87     while ( SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_TFE) != SET);
88 }

```

## 4.24.2.7 spi\_transmit\_message()

```
void spi_transmit_message (
    spi_n * spi_struct,
    uint16_t message[],
    uint32_t length )
```

Отправляет массив полуслов по SPIn.

Аргументы

*spi_struct	- Выбранный SPI
message	- Массив для отправки
length	- Размер массива

```
93 {
94     for(uint32_t i = 0; i < length; i++)
95     {
96         while(SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_TNF) != SET) {} // Ждем, когда в буфере появится
           место и затем записываем следующий байт
97         SSP_SendData(spi_struct->SSPx, message[i]);
98     }
99     while (SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_TFE) != SET) {} // Ждем, пока байт уйдет
100 }
```

## 4.24.3 Переменные

## 4.24.3.1 spi\_1

```
spi_n spi_1
```

Структура с конфигурационными параметрами блока SPI1 МК

Структура с конфигурационными параметрами блока SPI МК

Структуры с конфигурационными параметрами блоков SPI МК

## 4.24.3.2 spi\_2

```
spi_n spi_2
```

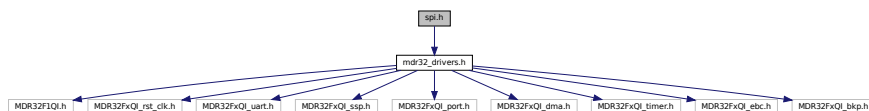
Структура с конфигурационными параметрами блока SPI2 МК

## 4.25 Файл spi.h

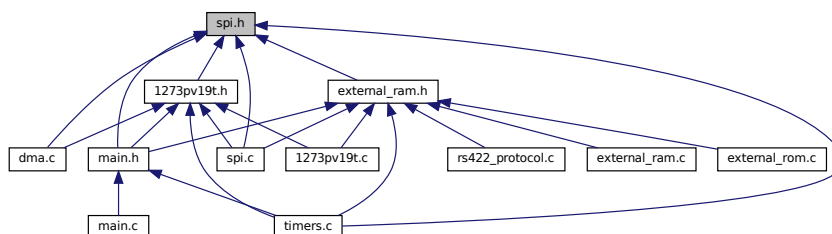
Заголовочный файл с описанием API для работы с SPI.

```
#include "mdr32_drivers.h"
```

Граф включаемых заголовочных файлов для spi.h:



Граф файлов, в которые включается этот файл:



## Классы

- struct `spi_dma_params`  
Структура с параметрами DMA канала SPIn.
- struct `spi_config_data`  
Структура с конфигурационными параметрами SPI.

## Макросы

- `#define PORT_SSP1 MDR_PORTD`  
Порт SSP1.
- `#define PIN_SSP1_SCK PORT_Pin_4`  
Линия SCK порта SSP1.
- `#define PIN_SSP1_RX PORT_Pin_3`  
Линия RX порта SSP1.
- `#define PIN_SSP1_TX PORT_Pin_2`  
Линия TX порта SSP1.
- `#define PIN_SSP1_SS PORT_Pin_5`  
Линия SS порта SSP1.
- `#define FIFO_SIZE 8`  
Размер буфера FIFO SSP в полусловах (16 бит)
- `#define SPI_BUFFER_SIZE 512`  
Буфер приемника SSP.

## Определения типов

- typedef struct [spi\\_dma\\_params](#) [spi\\_n\\_dma\\_ch\\_params](#)  
Структура с параметрами DMA канала SPIn.
- typedef struct [spi\\_config\\_data](#) [spi\\_n](#)  
Структура с конфигурационными параметрами SPI.

## Функции

- void [spi\\_init](#) ([spi\\_n](#) \*spi\_struct)  
Инициализирует выбранный SPIn.
- void [spi\\_transmit\\_halfword](#) ([spi\\_n](#) \*spi\_struct, uint16\_t half\_word)  
Отправляет полуслово по SPIn.
- void [spi\\_transmit\\_message](#) ([spi\\_n](#) \*spi\_struct, uint16\_t message[], uint32\_t length)  
Отправляет массив полуслов по SPIn.
- uint16\_t [spi\\_receive\\_halfword](#) ([spi\\_n](#) \*spi\_struct)  
Принимает полуслово по SPIn.
- void [spi\\_clean\\_fifo\\_rx\\_buf](#) ([spi\\_n](#) \*spi\_struct)  
Очищает FIFO буфер приемника SPIn.
- void [dma\\_spi\\_rx\\_init](#) ([spi\\_n](#) \*spi\_struct)  
Инициализирует n-ый канал DMA на запрос от приемника SPIn.

### 4.25.1 Подробное описание

Заголовочный файл с описанием API для работы с SPI.

### 4.25.2 Макросы

#### 4.25.2.1 FIFO\_SIZE

```
#define FIFO_SIZE 8
```

Размер буфера FIFO SSP в полусловах (16 бит)

#### 4.25.2.2 PIN\_SSP1\_RX

```
#define PIN_SSP1_RX PORT_Pin_3
```

Линия RX порта SSP1.

#### 4.25.2.3 PIN\_SSP1\_SCK

```
#define PIN_SSP1_SCK PORT_Pin_4
```

Линия SCK порта SSP1.

#### 4.25.2.4 PIN\_SSP1\_SS

```
#define PIN_SSP1_SS PORT_Pin_5
```

Линия SS порта SSP1.

#### 4.25.2.5 PIN\_SSP1\_TX

```
#define PIN_SSP1_TX PORT_Pin_2
```

Линия TX порта SSP1.

#### 4.25.2.6 PORT\_SSP1

```
#define PORT_SSP1 MDR_PORTD
```

Порт SSP1.

#### 4.25.2.7 SPI\_BUFFER\_SIZE

```
#define SPI_BUFFER_SIZE 512
```

Буфер приемника SSP.

### 4.25.3 Типы

#### 4.25.3.1 spi\_n

```
typedef struct spi_config_data spi_n
```

Структура с конфигурационными параметрами SPI.

## 4.25.3.2 spi\_n\_dma\_ch\_params

```
typedef struct spi_dma_params spi_n_dma_ch_params
```

Структура с параметрами DMA канала SPIn.

## 4.25.4 Функции

## 4.25.4.1 dma\_spi\_rx\_init()

```
void dma_spi_rx_init (
    spi_n * spi_struct )
```

Инициализирует n-ый канал DMA на запрос от приемника SPIn.

Аргументы

*spi_struct	- Выбранный SPI
-------------	-----------------

```
129 {
130
131     DMA_StructInit(&spi_struct->spi_dma_ch.DMA_Channel_SPI_RX);
132     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_SourceBaseAddr = (uint32_t)(&(spi_struct->SSPx-
133     >DR));
134     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_DestBaseAddr = (uint32_t)(spi_struct->buffer);
135     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_CycleSize = 1;
136     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_SourceIncSize = DMA_SourceIncNo;
137     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_DestIncSize = DMA_DestIncHalfword;
138     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
139     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_NumContinuous = DMA_Transfers_1;
140     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_SourceProtCtrl = DMA_SourcePrivileged;
141     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_DestProtCtrl = DMA_DestPrivileged;
142     spi_struct->spi_dma_ch.DMA_InitStructure_SPI_RX.DMA_Mode = DMA_Mode_Basic;
143
144     // Задать структуру канала
145     spi_struct->spi_dma_ch.DMA_Channel_SPI_RX.DMA_PriCtrlData = &spi_struct-
146     >spi_dma_ch.DMA_InitStructure_SPI_RX;
147     spi_struct->spi_dma_ch.DMA_Channel_SPI_RX.DMA_Priority = DMA_Priority_High;
148     spi_struct->spi_dma_ch.DMA_Channel_SPI_RX.DMA_UseBurst = DMA_BurstClear;
149     spi_struct->spi_dma_ch.DMA_Channel_SPI_RX.DMA_SelectDataStructure = DMA_CTRL_DATA_PRIMARY;
150
151     // Инициализировать канал
152     DMA_Init(spi_struct->spi_dma_ch.dma_channel, &spi_struct->spi_dma_ch.DMA_Channel_SPI_RX);
153
154     MDR_DMA->CHNL_REQ_MASK_CLR = 1 « spi_struct->spi_dma_ch.dma_channel;
155     MDR_DMA->CHNL_USEBURST_CLR = 1 « spi_struct->spi_dma_ch.dma_channel;
156
157     SSP_DMACmd(spi_struct->SSPx, SSP_DMA_RXE, DISABLE);
158     // Разрешить работу DMA с SPI
159     DMA_Cmd (spi_struct->spi_dma_ch.dma_channel, DISABLE);
160
161     //NVIC_SetPriority (DMA_IRQn, 0);
162     //NVIC_EnableIRQ(DMA_IRQn);
163 }
```

## 4.25.4.2 spi\_clean\_fifo\_rx\_buf()

```
void spi_clean_fifo_rx_buf (
    spi_n * spi_struct )
```

Очищает FIFO буфер приемника SPIn.

## Аргументы

*spi_struct	- Выбранный SPI
-------------	-----------------

```

118 {
119     uint16_t a;
120     while(SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_RNE) == SET)
121     {
122         a = spi_struct->SSPx->DR;
123     }
124 }

```

## 4.25.4.3 spi\_init()

```

void spi_init (
    spi_n * spi_struct )

```

Инициализирует выбранный SPI.

## Аргументы

*spi_struct	- Выбранный SPI для инициализации
-------------	-----------------------------------

```

53 {
54     spi_gpio_config();
55
56     // Структура для инициализации SPI
57     SSP_InitTypeDef SSP_InitStruct;
58
59     // Включение тактирования SPI
60     RST_CLK_PCLKcmd(spi_struct->RST_CLK_PCLK_SPIn, ENABLE);
61
62     SSP_StructInit(&SSP_InitStruct);
63
64     SSP_BRGInit(spi_struct->SSPx, SSP_HCLKdiv1);
65
66     SSP_InitStruct.SSP_WordLength = spi_struct->SPI.SSP_WordLength;
67     SSP_InitStruct.SSP_Mode = spi_struct->SPI.SSP_Mode;
68     SSP_InitStruct.SSP_SPH = spi_struct->SPI.SSP_SPH;
69     SSP_InitStruct.SSP_FRF = spi_struct->SPI.SSP_FRF;
70     SSP_InitStruct.SSP_HardwareFlowControl = spi_struct->SPI.SSP_HardwareFlowControl;
71     //SSP_InitStruct.SSP_SCR = 0x10;
72     SSP_InitStruct.SSP_CPSPDVR = spi_struct->SPI.SSP_CPSPDVR; // Частота обмена 2МГц
73     SSP_Init(spi_struct->SSPx, &SSP_InitStruct);
74
75     NVIC_DisableIRQ(spi_struct->IRQn);
76     // Выбор источников прерываний (прием и передача данных)
77     SSP_ITConfig (spi_struct->SSPx, SSP_IT_RX, DISABLE);
78
79     SSP_Cmd(spi_struct->SSPx, ENABLE);
80 }

```

## 4.25.4.4 spi\_receive\_halfword()

```

uint16_t spi_receive_halfword (
    spi_n * spi_struct )

```

Принимает полуслово по SPI.

## Аргументы

*spi_struct	- Выбранный SPI
-------------	-----------------



```

105 {
106     uint16_t tmpVar;
107     // Обработка прерывания от приемника данных
108     while (SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_RNE) != SET) {} // Ждем, пока появится байт
109     // Получаем данные
110     tmpVar = SSP_ReceiveData(spi_struct->SSPx);
111
112     return tmpVar;
113 }

```

#### 4.25.4.5 spi\_transmit\_halfword()

```

void spi_transmit_halfword (
    spi_n * spi_struct,
    uint16_t half_word )

```

Отправляет полуслово по SPIn.

Аргументы

*spi_struct	- Выбранный SPI
half_word	- Полуслово для отправки

```

85 {
86     SSP_SendData(spi_struct->SSPx, half_word);
87     while ( SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_TFE) != SET);
88 }

```

#### 4.25.4.6 spi\_transmit\_message()

```

void spi_transmit_message (
    spi_n * spi_struct,
    uint16_t message[],
    uint32_t length )

```

Отправляет массив полуслов по SPIn.

Аргументы

*spi_struct	- Выбранный SPI
message	- Массив для отправки
length	- Размер массива

```

93 {
94     for(uint32_t i = 0; i < length; i++)
95     {
96         while(SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_TNF) != SET) {} // Ждем, когда в буфере появится
           место и затем записываем следующий байт
97         SSP_SendData(spi_struct->SSPx, message[i]);
98     }
99     while (SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_TFE) != SET) {} // Ждем, пока байт уйдет
100 }

```



## Переменные

- `adc_n adc_1`  
Структура с конфигурационными параметрами микросхемы АЦП
- `ram_data * ram_space_pointer`  
Указатель для обращения к внешнему ОЗУ
- `list_head tmr_handler_head [TIMER_NUM]`  
Указатели на списки обработчиков таймеров
- `timer_n timer_1`  
Структура с конфигурационными параметрами блока Timer1 МК
- `timer_n timer_2`  
Структура с конфигурационными параметрами блока Timer2 МК
- `timer_n timer_3`  
Структура с конфигурационными параметрами блока Timer3 МК
- `spi_n spi_1`  
Структура с конфигурационными параметрами блока SPI МК

### 4.26.1 Подробное описание

Файл с реализацией API для работы с таймерами

### 4.26.2 Функции

#### 4.26.2.1 `delay_micro()`

```
void delay_micro (
    uint32_t time_micro )
```

Реализует задержку в мкс

Аргументы

<code>time_micro</code>	- Задержка в мкс
-------------------------	------------------

```
162 {
163     TIMER_SetCounter(MDR_TIMER1, 0);
164     while (TIMER_GetCounter(MDR_TIMER1) <= time_micro);
165 }
```

#### 4.26.2.2 `delay_milli()`

```
void delay_milli (
    uint32_t time_milli )
```

Реализует задержку в мс

## Аргументы

time_milli	- Задержка в мс
------------	-----------------

```

154 {
155     TIMER_SetCounter(MDR_TIMER3, 0);
156     while (TIMER_GetCounter(MDR_TIMER3) <=(time_milli*50));
157 }

```

## 4.26.2.3 list\_tmr\_handler\_add\_tail()

```

void list_tmr_handler_add_tail (
    uint8_t tmr_num,
    void (*)(void *) func_ptr,
    void * data,
    TIMER_Status_Flags_TypeDef event )

```

Добавляет обработчик прерывания таймера в список обработчиков

## Аргументы

tmr_num	- Номер таймера
func_ptr	- Указатель на функцию-обработчик прерывания
data	- Указатель на данные в обработчике прерывания
event	- Событие, по которому вызывается данное прерывание

```

170 {
171     // Выделяем память для нового указателя
172     timer_irq_list *ptr = (timer_irq_list*)malloc_ram_pages(sizeof(timer_irq_list));
173     // Инициализируем элемент списка
174     ptr->data = data;
175     ptr->event = event;
176     ptr->handler = func_ptr;
177
178     list_add_tail(&ptr->list, &tmr_handler_head[tmr_num]);
179 }

```

## 4.26.2.4 list\_tmr\_handler\_init()

```

void list_tmr_handler_init (
    uint8_t tmr_num )

```

Инициализирует список обработчиков прерываний таймера

## Аргументы

tmr_num	- Номер таймера, для которого инициализируется список
---------	---

```

184 {
185     init_list_head(&tmr_handler_head[tmr_num]);
186 }

```

## 4.26.2.5 timer1\_init()

```
void timer1_init (
    timer_n * timer_struct )
```

Инициализирует Timer1.

```
36 {
37     TIMER_CntInitTypeDef TIMER1InitStruct;
38
39     RST_CLK_PCLKcmd (RST_CLK_PCLK_TIMER1, ENABLE);
40
41     TIMER_CntStructInit(&TIMER1InitStruct);
42     TIMER1InitStruct.TIMER_Period = timer_struct->TIMERInitStruct.TIMER_Period;
43     TIMER1InitStruct.TIMER_Prescaler = timer_struct->TIMERInitStruct.TIMER_Prescaler;
44
45     TIMER_CntInit(MDR_TIMER1, &TIMER1InitStruct);
46
47     TIMER_BRGInit(MDR_TIMER1, TIMER_HCLKdiv1);
48
49     TIMER_Cmd(MDR_TIMER1, ENABLE);
50     TIMER_SetCounter(MDR_TIMER1, 0);
51 }
```

## 4.26.2.6 timer2\_init()

```
void timer2_init (
    timer_n * timer_struct )
```

Инициализирует Timer2.

```
78 {
79     TIMER_CntInitTypeDef TIMER2InitStruct;
80     TIMER_ChnInitTypeDef sTIM2_ChnInit;
81
82     RST_CLK_PCLKcmd (RST_CLK_PCLK_TIMER2, ENABLE);
83
84     TIMER_CntStructInit(&TIMER2InitStruct);
85     TIMER2InitStruct.TIMER_Prescaler = timer_struct->TIMERInitStruct.TIMER_Prescaler;
86     TIMER2InitStruct.TIMER_Period = timer_struct->TIMERInitStruct.TIMER_Period;
87     TIMER2InitStruct.TIMER_CounterDirection = timer_struct->TIMERInitStruct.TIMER_CounterDirection;
88     TIMER2InitStruct.TIMER_CounterMode = timer_struct->TIMERInitStruct.TIMER_CounterMode;
89     TIMER2InitStruct.TIMER_EventSource = timer_struct->TIMERInitStruct.TIMER_EventSource;
90     TIMER2InitStruct.TIMER_ARR_UpdateMode = timer_struct->TIMERInitStruct.TIMER_ARR_UpdateMode;
91     TIMER2InitStruct.TIMER_FilterSampling = timer_struct->TIMERInitStruct.TIMER_FilterSampling;
92     TIMER2InitStruct.TIMER_ETR_FilterConf = timer_struct->TIMERInitStruct.TIMER_ETR_FilterConf;
93     TIMER2InitStruct.TIMER_ETR_Prescaler = timer_struct->TIMERInitStruct.TIMER_ETR_Prescaler;
94     TIMER2InitStruct.TIMER_ETR_Polarity = timer_struct->TIMERInitStruct.TIMER_ETR_Polarity;
95     TIMER2InitStruct.TIMER_BRK_Polarity = timer_struct->TIMERInitStruct.TIMER_BRK_Polarity;
96     TIMER_CntInit (MDR_TIMER2, &TIMER2InitStruct);
97
98     TIMER_ChnStructInit (&sTIM2_ChnInit);
99     sTIM2_ChnInit.TIMER_CH_Number = timer_struct->sTIM_ChnInit.TIMER_CH_Number;
100     sTIM2_ChnInit.TIMER_CH_Mode = timer_struct->sTIM_ChnInit.TIMER_CH_Mode;
101     sTIM2_ChnInit.TIMER_CH_EventSource = timer_struct->sTIM_ChnInit.TIMER_CH_EventSource;
102     TIMER_ChnInit (MDR_TIMER2, &sTIM2_ChnInit);
103
104     TIMER_ChnCCR1_Cmd(MDR_TIMER2, sTIM2_ChnInit.TIMER_CH_Number, ENABLE);
105
106     TIMER_ITConfig(MDR_TIMER2, timer_struct->TIMER_STATUS, ENABLE);
107     TIMER_ClearITPendingBit(MDR_TIMER2, TIMER_STATUS_CNT_ARR);
108     TIMER_ITConfig(MDR_TIMER2, TIMER_STATUS_CNT_ARR, DISABLE);
109     NVIC_EnableIRQ(TIMER2_IRQn);
110     //NVIC_SetPriority(timer_struct->IRQn, 0);
111
112     TIMER_BRGInit(MDR_TIMER2, TIMER_HCLKdiv1);
113
114     TIMER_Cmd(MDR_TIMER2, ENABLE);
115     TIMER_SetCounter(MDR_TIMER2, 0);
116 }
```

## 4.26.2.7 TIMER2\_IRQHandler()

```
void TIMER2_IRQHandler (
    void )
```

Обработчик прерываний Timer2 (Обработка прерываний путем обхода двусвязного списка реализована аналогично linux kernel)

```
140 {
141     timer_irq_list *ep;
142     list_for_each_entry(ep, &tmr_handler_head[1], list)
143     {
144         if (TIMER_GetITStatus(timer_2.TIMERx, ep->event) == SET)
145         {
146             ep->handler(ep->data);
147         }
148     }
149 }
```

## 4.26.2.8 timer3\_init()

```
void timer3_init (
    timer_n * timer_struct )
```

Инициализирует Timer3.

```
57 {
58     TIMER_CntInitTypeDef TIMER3InitStruct;
59
60     RST_CLK_PCLKcmd (RST_CLK_PCLK_TIMER3, ENABLE);
61
62     TIMER_CntStructInit(&TIMER3InitStruct);
63     TIMER3InitStruct.TIMER_Period = timer_struct->TIMERInitStruct.TIMER_Period;
64     TIMER3InitStruct.TIMER_Prescaler = timer_struct->TIMERInitStruct.TIMER_Prescaler;
65
66     TIMER_CntInit(MDR_TIMER3, &TIMER3InitStruct);
67
68     TIMER_BRGInit(MDR_TIMER3, TIMER_HCLKdiv1);
69
70     TIMER_Cmd(MDR_TIMER3, ENABLE);
71     TIMER_SetCounter(MDR_TIMER3, 0);
72 }
```

## 4.26.2.9 timer\_init()

```
void timer_init (
    timer_n * timer_struct )
```

Инициализирует выбранный Timer.

Аргументы

*timer_struct	- Таймер для инициализации
---------------	----------------------------

```
121 {
122     if (timer_struct->TIMERx == MDR_TIMER1)
123     {
124         timer1_init(timer_struct);
125     }
126     else if (timer_struct->TIMERx == MDR_TIMER2)
127     {
128         timer2_init(timer_struct);
129     }
```

```
130     else if (timer_struct->TIMERx == MDR_TIMER3)
131     {
132         timer3_init(timer_struct);
133     }
134 }
```

### 4.26.3 Переменные

#### 4.26.3.1 adc\_1

`adc_n` adc\_1 [extern]

Структура с конфигурационными параметрами микросхемы АЦП

#### 4.26.3.2 ram\_space\_pointer

`ram_data*` ram\_space\_pointer [extern]

Указатель для обращения к внешнему ОЗУ

#### 4.26.3.3 spi\_1

`spi_n` spi\_1 [extern]

Структура с конфигурационными параметрами блока SPI МК

Структура с конфигурационными параметрами блока SPI МК

Структуры с конфигурационными параметрами блоков SPI МК

#### 4.26.3.4 timer\_1

`timer_n` timer\_1

Структура с конфигурационными параметрами блока Timer1 МК

Структуры с конфигурационными параметрами блоков таймеров МК

#### 4.26.3.5 timer\_2

`timer_n` timer\_2

Структура с конфигурационными параметрами блока Timer2 МК

#### 4.26.3.6 timer\_3

`timer_n timer_3`

Структура с конфигурационными параметрами блока Timer3 МК

#### 4.26.3.7 tmr\_handler\_head

`list_head tmr_handler_head[TIMER_NUM]`

Указатели на списки обработчиков таймеров

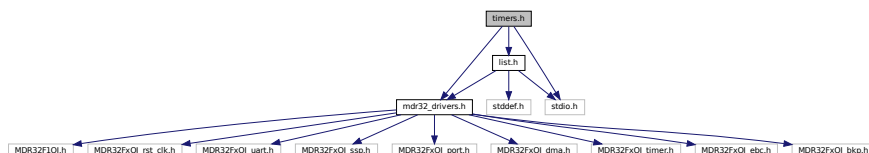
Массив указателей на head списков обработчиков прерываний таймеров

### 4.27 Файл timers.h

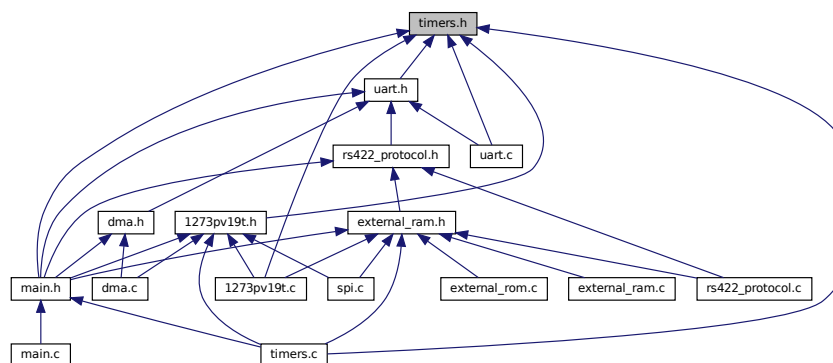
Заголовочный файл с описанием API для работы с таймерами

```
#include "mdr32_drivers.h"
#include <stdio.h>
#include "list.h"
```

Граф включаемых заголовочных файлов для timers.h:



Граф файлов, в которые включается этот файл:





## Классы

- struct `timer_config_struct`  
Структура с конфигурационными параметрами Таймеров
- struct `timer_irq_list_struct`  
Структура-реализация односвязанного списка обработчиков прерываний таймеров

## Определения типов

- typedef struct `timer_config_struct` `timer_n`  
Структура с конфигурационными параметрами Таймеров
- typedef struct `timer_irq_list_struct` `timer_irq_list`  
Структура-реализация односвязанного списка обработчиков прерываний таймеров

## Функции

- void `timer_init` (`timer_n` \*`timer_struct`)  
Инициализирует выбранный Timer.
- void `delay_milli` (`uint32_t` `time_milli`)  
Реализует задержку в мс
- void `delay_micro` (`uint32_t` `time_micro`)  
Реализует задержку в мкс
- void `list_tmr_handler_add_tail` (`uint8_t` `tmr_num`, void(\*`func_ptr`)(void \*), void \*`data`, TIM←  
ER\_Status\_Flags\_TypeDef `event`)  
Добавляет обработчик прерывания таймера в список обработчиков
- void `list_tmr_handler_init` (`uint8_t` `tmr_num`)  
Инициализирует список обработчиков прерываний таймера

### 4.27.1 Подробное описание

Заголовочный файл с описанием API для работы с таймерами

### 4.27.2 Типы

#### 4.27.2.1 `timer_irq_list`

typedef struct `timer_irq_list_struct` `timer_irq_list`

Структура-реализация односвязанного списка обработчиков прерываний таймеров

#### 4.27.2.2 timer\_n

```
typedef struct timer_config_struct timer_n
```

Структура с конфигурационными параметрами Таймеров

### 4.27.3 Функции

#### 4.27.3.1 delay\_micro()

```
void delay_micro (
    uint32_t time_micro )
```

Реализует задержку в мкс

Аргументы

time_micro	- Задержка в мкс
------------	------------------

```
162 {
163     TIMER_SetCounter(MDR_TIMER1, 0);
164     while (TIMER_GetCounter(MDR_TIMER1) <= time_micro);
165 }
```

#### 4.27.3.2 delay\_milli()

```
void delay_milli (
    uint32_t time_milli )
```

Реализует задержку в мс

Аргументы

time_milli	- Задержка в мс
------------	-----------------

```
154 {
155     TIMER_SetCounter(MDR_TIMER3, 0);
156     while (TIMER_GetCounter(MDR_TIMER3) <=(time_milli*50));
157 }
```

#### 4.27.3.3 list\_tmr\_handler\_add\_tail()

```
void list_tmr_handler_add_tail (
    uint8_t tmr_num,
    void(*)(void *) func_ptr,
```

```
void * data,
TIMER_Status_Flags_TypeDef event )
```

Добавляет обработчик прерывания таймера в список обработчиков

Аргументы

tmr_num	- Номер таймера
func_ptr	- Указатель на функцию-обработчик прерывания
data	- Указатель на данные в обработчике прерывания
event	- Событие, по которому вызывается данное прерывание

```
170 {
171     // Выделяем память для нового указателя
172     timer_irq_list *ptr = (timer_irq_list*) malloc_ram_pages(sizeof(timer_irq_list));
173     // Инициализируем элемент списка
174     ptr->data = data;
175     ptr->event = event;
176     ptr->handler = func_ptr;
177
178     list_add_tail(&ptr->list, &tmr_handler_head[tmr_num]);
179 }
```

#### 4.27.3.4 list\_tmr\_handler\_init()

```
void list_tmr_handler_init (
    uint8_t tmr_num )
```

Инициализирует список обработчиков прерываний таймера

Аргументы

tmr_num	- Номер таймера, для которого инициализируется список
---------	---

```
184 {
185     init_list_head(&tmr_handler_head[tmr_num]);
186 }
```

#### 4.27.3.5 timer\_init()

```
void timer_init (
    timer_n * timer_struct )
```

Инициализирует выбранный Timer.

Аргументы

*timer_struct	- Таймер для инициализации
---------------	----------------------------

```
121 {
122     if (timer_struct->TIMERx == MDR_TIMER1)
123     {
124         timer1_init(timer_struct);
125     }
```

```

126     else if (timer_struct->TIMERx == MDR_TIMER2)
127     {
128         timer2_init(timer_struct);
129     }
130     else if (timer_struct->TIMERx == MDR_TIMER3)
131     {
132         timer3_init(timer_struct);
133     }
134 }

```

## 4.28 Файл uart.c

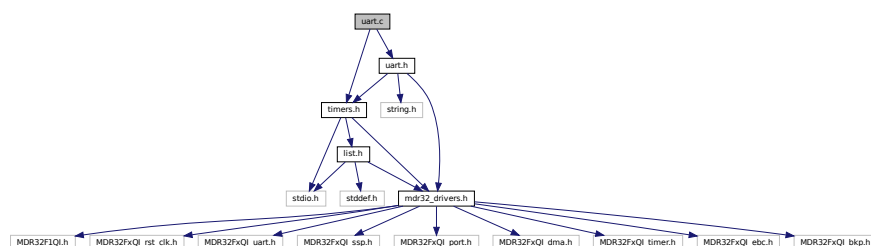
Файл с реализацией API для работы с UART.

```

#include "uart.h"
#include "timers.h"

```

Граф включаемых заголовочных файлов для uart.c:



## Функции

- void `UART1_IRQHandler` (void)  
Обработчик прерываний UART1.
- void `UART2_IRQHandler` (void)  
Обработчик прерываний UART2.
- void `uart_gpio_config` (void)  
Конфигурирует выводы МК для UART.
- `uart_errors` `uart_init` (`uart_n` \*`uart_struct`)  
Инициализирует выбранный UARTn.
- `uart_errors` `uart_write` (`uart_n` \*`uart_struct`, `uint8_t` \*`data`, `uint32_t` `data_size`)  
Передаёт данные по UART.
- `uart_errors` `uart_read` (`uart_n` \*`uart_struct`, `uint32_t` `len`, `uint8_t` \*`data`)  
Читает данные из буфера UART.
- `uart_errors` `uart_set_pos` (`uart_n` \*`uart_struct`, `uint32_t` `pos`)  
Устанавливает курсор чтения в кольцевом буфере
- `uint32_t` `uart_read_pos` (`uart_n` \*`uart_struct`)  
Читает текущую позицию курсора чтения в кольцевом буфере
- `uint32_t` `uart_get_buf_counter` (`uart_n` \*`uart_struct`)  
Читает кол-во байт в кольцевом буфере UART.
- void `uart_clean` (`uart_n` \*`uart_struct`)  
Очищает кольцевой буфер приемника UART.
- void `DMA_UART_RX_init` (`uart_n` \*`uart_struct`)  
Инициализирует n-ый канал DMA на запрос от приемника UARTn.
- void `uart_set_read_timeout` (`uart_n` \*`uart_struct`, `uint32_t` `read_timeout`)  
Устанавливает таймаут UARTn на чтение
- void `uart_set_write_timeout` (`uart_n` \*`uart_struct`, `uint32_t` `write_timeout`)  
Устанавливает таймаут UARTn на запись

## Переменные

- `uart_n uart_1`  
Структура с конфигурационными параметрами блока UART1 МК
- `uart_n uart_2`  
Структура с конфигурационными параметрами блока UART2 МК

### 4.28.1 Подробное описание

Файл с реализацией API для работы с UART.

### 4.28.2 Функции

#### 4.28.2.1 DMA\_UART\_RX\_init()

```
void DMA_UART_RX_init (
    uart_n * uart_struct )
```

Инициализирует n-ый канал DMA на запрос от приемника UARTn.

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

```
379 {
380
381     DMA_StructInit(&uart_struct->uart_dma_ch.DMA_Channel_UART_RX);
382     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_SourceBaseAddr = (uint32_t)(&(uart_struct-
>UARTx->DR));
383     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestBaseAddr = (uint32_t)(uart_struct->buffer);
384     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_CycleSize = 1024;
385     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_SourceIncSize = DMA_SourceIncNo;
386     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestIncSize = DMA_DestIncByte;
387     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
388     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_NumContinuous = DMA_Transfers_1;
389     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_SourceProtCtrl = DMA_SourcePrivileged;
390     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestProtCtrl = DMA_DestPrivileged;
391     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_Mode = DMA_Mode_Basic;
392
393     // Задать структуру канала
394     uart_struct->uart_dma_ch.DMA_Channel_UART_RX.DMA_PriCtrlData = &uart_struct-
>uart_dma_ch.DMA_InitStructure_UART_RX;
395     uart_struct->uart_dma_ch.DMA_Channel_UART_RX.DMA_Priority = DMA_Priority_High;
396     uart_struct->uart_dma_ch.DMA_Channel_UART_RX.DMA_UseBurst = DMA_BurstClear;
397     uart_struct->uart_dma_ch.DMA_Channel_UART_RX.DMA_SelectDataStructure =
DMA_CTRL_DATA_PRIMARY;
398
399     // Инициализировать канал
400     DMA_Init(uart_struct->uart_dma_ch.dma_channel, &uart_struct->uart_dma_ch.DMA_Channel_UART_RX);
401
402     MDR_DMA->CHNL_REQ_MASK_CLR = 1 « uart_struct->uart_dma_ch.dma_channel;
403     MDR_DMA->CHNL_USEBURST_CLR = 1 « uart_struct->uart_dma_ch.dma_channel;
404
405     UART_DMACmd(uart_struct->UARTx, UART_DMA_RXE, ENABLE);
406     // Разрешить работу DMA с UART
407     DMA_Cmd (uart_struct->uart_dma_ch.dma_channel, ENABLE);
408
409     NVIC_SetPriority (DMA_IRQn, 1);
410     NVIC_EnableIRQ(DMA_IRQn);
411 }
```

## 4.28.2.2 UART1\_IRQHandler()

```
void UART1_IRQHandler (
    void )
```

Обработчик прерываний UART1.

```
26 {
27     uart_1.buffer_count &= BUFFER_MASK;
28     while (UART_GetFlagStatus(uart_1.UARTx, UART_FLAG_TXFE) != SET){}
29     uart_1.buffer_count++;
30     UART_ReceiveData(uart_1.UARTx);
31     UART_ClearITPendingBit(uart_1.UARTx, UART_IT_RX);
32 }
```

## 4.28.2.3 UART2\_IRQHandler()

```
void UART2_IRQHandler (
    void )
```

Обработчик прерываний UART2.

```
38 {
39     uart_2.buffer_count &= BUFFER_MASK;
40     while (UART_GetFlagStatus(uart_2.UARTx, UART_FLAG_TXFE) != SET){}
41     uart_2.buffer_count++;
42     UART_ReceiveData(uart_2.UARTx);
43     UART_ClearITPendingBit(uart_2.UARTx, UART_IT_RX);
44 }
```

## 4.28.2.4 uart\_clean()

```
void uart_clean (
    uart_n * uart_struct )
```

Очищает кольцевой буфер приемника UART.

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

```
372 {
373     memset(uart_struct->buffer, 0, UART_BUFFER_SIZE);
374 }
```

## 4.28.2.5 uart\_get\_buf\_counter()

```
uint32_t uart_get_buf_counter (
    uart_n * uart_struct )
```

Читает кол-во байт в кольцевом буфере UART.

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

Возвращает

Кол-во байт в кольцевом буфере UART

```

365 {
366     return uart_struct->buffer_count;
367 }

```

## 4.28.2.6 uart\_gpio\_config()

```

void uart_gpio_config (
    void )

```

Конфигурирует выводы МК для UART.

```

76 {
77     // Включение тактирования портов
78     RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK|RST_CLK_PCLK_PORTD|RST_CLK_PCLK_PORTC,
79                     ENABLE);
80
81     // Инициализации портов UART1
82     PORT_InitTypeDef GPIO_init_structUART1;
83     PORT_StructInit(&GPIO_init_structUART1);
84     GPIO_init_structUART1.PORT_FUNC = PORT_FUNC_MAIN;
85     GPIO_init_structUART1.PORT_SPEED = PORT_SPEED_MAXFAST;
86     GPIO_init_structUART1.PORT_MODE = PORT_MODE_DIGITAL;
87     GPIO_init_structUART1.PORT_PULL_UP = PORT_PULL_UP_OFF;
88     GPIO_init_structUART1.PORT_PULL_DOWN = PORT_PULL_DOWN_ON;
89     GPIO_init_structUART1.PORT_PD_SHM = PORT_PD_SHM_OFF;
90     GPIO_init_structUART1.PORT_PD = PORT_PD_DRIVER;
91     GPIO_init_structUART1.PORT_GFEN = PORT_GFEN_OFF;
92     // Инициализация вывода PC3 как UART_TX (передача)
93     GPIO_init_structUART1.PORT_Pin = PIN_UART1_TX;
94     GPIO_init_structUART1.PORT_OE = PORT_OE_OUT;
95     PORT_Init(PORT_UART1, &GPIO_init_structUART1);
96     // Инициализация вывода PC4 как UART_RX (прием)
97     GPIO_init_structUART1.PORT_Pin = PIN_UART1_RX;
98     GPIO_init_structUART1.PORT_OE = PORT_OE_IN;
99     PORT_Init(PORT_UART1, &GPIO_init_structUART1);
100
101     // Инициализация ножки разрешения записи данных по UART1 (для микросхемы rs485)
102     GPIO_init_structUART1.PORT_Pin = PIN_UART1_EN;
103     GPIO_init_structUART1.PORT_FUNC = PORT_FUNC_PORT;
104     GPIO_init_structUART1.PORT_OE = PORT_OE_OUT;
105     PORT_Init(PORT_UART1_EN, &GPIO_init_structUART1);
106     // Деактивирование микросхемы RS485 на выдачу данных
107     PORT_WriteBit(PORT_UART1_EN, PIN_UART1_EN, 0);
108
109
110     // Инициализация портов UART2
111     PORT_InitTypeDef GPIO_init_structUART2;
112
113     PORT_StructInit(&GPIO_init_structUART2);
114     GPIO_init_structUART2.PORT_FUNC = PORT_FUNC_MAIN;
115     GPIO_init_structUART2.PORT_SPEED = PORT_SPEED_MAXFAST;
116     GPIO_init_structUART2.PORT_MODE = PORT_MODE_DIGITAL;
117     GPIO_init_structUART2.PORT_PULL_UP = PORT_PULL_UP_OFF;
118     GPIO_init_structUART2.PORT_PULL_DOWN = PORT_PULL_DOWN_ON;
119     GPIO_init_structUART2.PORT_PD_SHM = PORT_PD_SHM_OFF;
120     GPIO_init_structUART2.PORT_PD = PORT_PD_DRIVER;
121     GPIO_init_structUART2.PORT_GFEN = PORT_GFEN_OFF;
122     // Инициализация вывода PD13 как UART_TX (передача)
123     GPIO_init_structUART2.PORT_Pin = PIN_UART2_TX;
124     GPIO_init_structUART2.PORT_OE = PORT_OE_OUT;
125     PORT_Init(PORT_UART2, &GPIO_init_structUART2);
126     // Инициализация вывода PD14 как UART_RX (прием)
127     GPIO_init_structUART2.PORT_Pin = PIN_UART2_RX;
128     GPIO_init_structUART2.PORT_OE = PORT_OE_IN;
129     PORT_Init(PORT_UART2, &GPIO_init_structUART2);

```

```

130
131 // Инициализация ножки разрешения записи данных по UART2 (для микросхемы rs485)
132 GPIO_init_structUART2.PORT_Pin = PIN_UART2_EN;
133 GPIO_init_structUART2.PORT_FUNC = PORT_FUNC_PORT;
134 GPIO_init_structUART2.PORT_OE = PORT_OE_OUT;
135 PORT_Init(PORT_UART2_EN, &GPIO_init_structUART2);
136 // Деактивирование микросхемы RS485 на выдачу данных
137 PORT_WriteBit(PORT_UART2_EN, PIN_UART2_EN, 0);
138 }

```

#### 4.28.2.7 uart\_init()

```

uart_errors uart_init (
    uart_n * uart_struct )

```

Инициализирует выбранный UARTn.

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

Возвращает

Код ошибки uart\_errors

```

143 {
144     uart_gpio_config();
145
146     uart_errors error;
147
148     // Объявление структуры для инициализации контроллера UART
149     UART_InitTypeDef UART_InitStructure;
150     uint8_t res = 0;
151
152     // Включение тактирования UART
153     if(uart_struct->UARTx == MDR_UART1)
154     {
155         uart_struct->RST_CLK_PCLK_UARTn = RST_CLK_PCLK_UART1;
156     }
157     else if(uart_struct->UARTx == MDR_UART2)
158     {
159         uart_struct->RST_CLK_PCLK_UARTn = RST_CLK_PCLK_UART2;
160     }
161     RST_CLK_PCLKcmd(uart_struct->RST_CLK_PCLK_UARTn, ENABLE);
162
163     // Делитель тактовой частоты UART
164     uart_struct->UART_HCLKdiv = UART_HCLKdiv1;
165     UART_BRGInit(uart_struct->UARTx, uart_struct->UART_HCLKdiv);
166
167     NVIC_EnableIRQ(uart_struct->IRQn);
168
169     // Конфигурация UART
170     UART_InitStructure.UART_BaudRate = uart_struct->UART.UART_BaudRate;
171     UART_InitStructure.UART_WordLength = uart_struct->UART.UART_WordLength;
172     UART_InitStructure.UART_StopBits = uart_struct->UART.UART_StopBits;
173     UART_InitStructure.UART_Parity = uart_struct->UART.UART_Parity;
174     UART_InitStructure.UART_FIFOMode = uart_struct->UART.UART_FIFOMode;
175     UART_InitStructure.UART_HardwareFlowControl = uart_struct->UART.UART_HardwareFlowControl;
176
177     // Инициализация UART с заданными параметрами
178     res = UART_Init(uart_struct->UARTx, &UART_InitStructure);
179     if (res != SUCCESS)
180     {
181         error = INIT_ERROR;
182         return error;
183     }
184
185     // Включение прерываний UART
186     NVIC_SetPriority (uart_struct->IRQn, 0);
187     UART_ITConfig( uart_struct->UARTx, UART_IT_RX, ENABLE );
188

```



```

189 // Включение сконфигурированного UART
190 UART_Cmd(uart_struct->UARTx, ENABLE);
191
192 return NO_ERRORS;
193 }

```

#### 4.28.2.8 uart\_read()

```

uart_errors uart_read (
    uart_n * uart_struct,
    uint32_t len,
    uint8_t * data )

```

Читает данные из буфера UART.

Аргументы

*uart_struct	- Выбранный UART для инициализации
len	- Кол-во байт для чтения
*data	- Указатель на массив, куда считываются байты из буфера UART

Возвращает

Код ошибки uart\_errors

```

255 {
256     uart_errors error;
257     uint32_t timer_cnt = 0;
258
259     // Если длина превышает размер буфера
260     if (len > UART_BUFFER_SIZE)
261     {
262         error = SIZE_ERROR;
263         uart_struct->read_pos++;
264         return error;
265     }
266     // Если последний принятый байт перевалил границу буфера и байты будут перезаписываться в буфере с самого
    начала
267     if (((uart_struct->read_pos)+len) > UART_BUFFER_SIZE)
268     {
269         // Если задан таймаут
270         if (uart_struct->uart_timeouts.read_timeout_flag == 1)
271         {
272             TIMER_SetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx, 0);
273             while ((int)((uart_struct->buffer_count) - (uart_struct->read_pos)) >= 0)
274             {
275                 timer_cnt = TIMER_GetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx);
276                 if (timer_cnt >= (uart_struct->uart_timeouts.read_val_timeout*50))
277                 {
278                     error = READ_TIMEOUT_ERROR;
279                     return error;
280                 }
281             }
282             while ((UART_BUFFER_SIZE - (uart_struct->read_pos) + (uart_struct->buffer_count)) < len)
283             {
284                 timer_cnt = TIMER_GetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx);
285                 if (timer_cnt >= (uart_struct->uart_timeouts.read_val_timeout*50))
286                 {
287                     error = READ_TIMEOUT_ERROR;
288                     return error;
289                 }
290             }
291         }
292
293         if ((int)((uart_struct->buffer_count) - (uart_struct->read_pos)) >= 0)
294         {
295             error = SIZE_ERROR;
296             return error;
297         }

```

```

298     else
299     {
300         if (((UART_BUFFER_SIZE + (uart_struct->buffer_count) - (uart_struct->read_pos)) < len)
301         {
302             error = SIZE_ERROR;
303             return error;
304         }
305         memcpy(data, (uart_struct->buffer) + (uart_struct->read_pos), UART_BUFFER_SIZE-(uart_struct-
>read_pos));
306         memcpy(data + UART_BUFFER_SIZE-(uart_struct->read_pos), uart_struct->buffer, len+(uart_struct-
>read_pos)-UART_BUFFER_SIZE);
307         uart_struct->read_pos = (uart_struct->read_pos) + len-UART_BUFFER_SIZE;
308     }
309 }
310 // Если последний принятый байт не перевалил границу буфера
311 else
312 {
313     // Если задан таймаут
314     if (uart_struct->uart_timeouts.read_timeout_flag == 1)
315     {
316         TIMER_SetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx, 0);
317         while (((uart_struct->buffer_count) - (uart_struct->read_pos)) < len)
318         {
319             timer_cnt = TIMER_GetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx);
320             if (timer_cnt >= (uart_struct->uart_timeouts.read_val_timeout*50))
321             {
322                 error = READ_TIMEOUT_ERROR;
323                 return error;
324             }
325         }
326     }
327
328     if (((uart_struct->buffer_count) - (uart_struct->read_pos)) < len)
329     {
330         error = SIZE_ERROR;
331         return error;
332     }
333     memcpy(data, (uart_struct->buffer) + (uart_struct->read_pos), len);
334     uart_struct->read_pos = (uart_struct->read_pos) + len;
335 }
336
337 return NO_ERRORS;
338 }

```

#### 4.28.2.9 uart\_read\_pos()

```

uint32_t uart_read_pos (
    uart_n * uart_struct )

```

Читает текущую позицию курсора чтения в кольцевом буфере

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

Возвращает

Текущая позиция курсора

```

358 {
359     return uart_struct->read_pos;
360 }

```

#### 4.28.2.10 uart\_set\_pos()

```

uart_errors uart_set_pos (
    uart_n * uart_struct,
    uint32_t pos )

```

Устанавливает курсор чтения в кольцевом буфере

Аргументы

*uart_struct	- Выбранный UART для инициализации
pos	- Позиция курсора в буфере

Возвращает

Код ошибки uart\_errors

```

343 {
344     uart_errors error;
345
346     if (pos > UART_BUFFER_SIZE)
347     {
348         error = POSITION_ERROR;
349         return error;
350     }
351     uart_struct->read_pos = pos;
352     return NO_ERRORS;
353 }
```

#### 4.28.2.11 uart\_set\_read\_timeout()

```

void uart_set_read_timeout (
    uart_n * uart_struct,
    uint32_t read_timeout )
```

Устанавливает таймаут UARTn на чтение

Аргументы

*uart_struct	- Выбранный UART для инициализации
read_timeout	- Таймаут на чтение (в мс)

```

416 {
417     uart_struct->uart_timeouts.read_timeout_flag = 1;
418     uart_struct->uart_timeouts.read_val_timeout = read_timeout;
419 }
```

#### 4.28.2.12 uart\_set\_write\_timeout()

```

void uart_set_write_timeout (
    uart_n * uart_struct,
    uint32_t write_timeout )
```

Устанавливает таймаут UARTn на запись

Аргументы

*uart_struct	- Выбранный UART для инициализации
write_timeout	- Таймаут на запись (в мс)

```

424 {
425     uart_struct->uart_timeouts.write_timeout_flag = 1;
426     uart_struct->uart_timeouts.write_val_timeout = write_timeout;
427 }

```

#### 4.28.2.13 uart\_write()

```

uart_errors uart_write (
    uart_n * uart_struct,
    uint8_t * data,
    uint32_t data_size )

```

Передаёт данные по UART.

Аргументы

*uart_struct	- Выбранный UART для инициализации
*data	- Указатель на массив данных
data_size	- Размер данных в байтах

Возвращает

Код ошибки uart\_errors

```

198 {
199     uart_errors error;
200     uint32_t timer_cnt = 0;
201
202     // Активирование микросхемы RS485 на выдачу данных
203     if (uart_struct->UARTx == MDR_UART1)
204     {
205         PORT_WriteBit(PORT_UART1_EN, PIN_UART1_EN, 1);
206     }
207     else if (uart_struct->UARTx == MDR_UART2)
208     {
209         PORT_WriteBit(PORT_UART2_EN, PIN_UART2_EN, 1);
210     }
211
212     if (uart_struct->uart_timeouts.write_timeout_flag == 1)
213     {
214         TIMER_SetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx, 0);
215     }
216     if (data_size > UART_BUFFER_SIZE)
217     {
218         error = SIZE_ERROR;
219         return error;
220     }
221     for (int i = 0; i < data_size; i++)
222     {
223         UART_SendData(uart_struct->UARTx, data[i]);
224         while (UART_GetFlagStatus(uart_struct->UARTx, UART_FLAG_TXFF) == SET)
225         {
226             if (uart_struct->uart_timeouts.write_timeout_flag == 1)
227             {
228                 timer_cnt = TIMER_GetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx);
229                 if (timer_cnt >= (uart_struct->uart_timeouts.write_val_timeout*50))
230                 {
231                     error = WRITE_TIMEOUT_ERROR;
232                     return error;
233                 }
234             }
235         }
236     }
237     // Небольшая задержка для микросхемы RS-485
238     delay_micro(10);
239     // Деактивирование микросхемы RS485 на прием данных
240     if (uart_struct->UARTx == MDR_UART1)
241     {
242         PORT_WriteBit(PORT_UART1_EN, PIN_UART1_EN, 0);

```

```

243     }
244     else if (uart_struct->UARTx == MDR_UART2)
245     {
246         PORT_WriteBit(PORT_UART2_EN, PIN_UART2_EN, 0);
247     }
248
249     return NO_ERRORS;
250 }

```

### 4.28.3 Переменные

#### 4.28.3.1 uart\_1

```
uart_n uart_1
```

Структура с конфигурационными параметрами блока UART1 МК

Структуры с конфигурационными параметрами блоков UART МК

#### 4.28.3.2 uart\_2

```
uart_n uart_2
```

Структура с конфигурационными параметрами блока UART2 МК

## 4.29 Файл uart.h

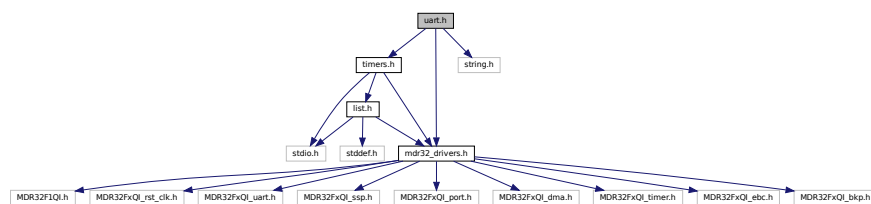
Заголовочный файл с описанием API для работы с UART.

```

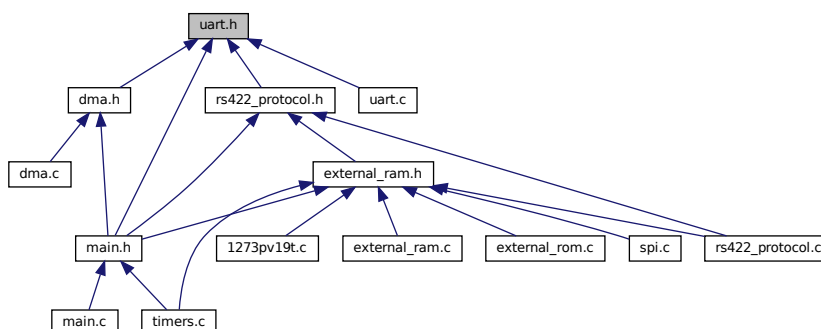
#include "mdr32_drivers.h"
#include "timers.h"
#include "string.h"

```

Граф включаемых заголовочных файлов для uart.h:



Граф файлов, в которые включается этот файл:



## Классы

- `struct uart_timeouts`  
Структура с таймаутами UARTn.
- `struct uart_dma_params`  
Структура с параметрами DMA канала UARTn.
- `struct uart_config_data`  
Структура с конфигурационными параметрами UART и буфером приема

## Макросы

- `#define PORT_UART1 MDR_PORTC`  
Порт UART1.
- `#define PIN_UART1_RX PORT_Pin_4`  
Линия RX порта UART1.
- `#define PIN_UART1_TX PORT_Pin_3`  
Линия TX порта UART1.
- `#define PORT_UART2 MDR_PORTD`  
Порт UART2.
- `#define PIN_UART2_RX PORT_Pin_14`  
Линия RX порта UART2.
- `#define PIN_UART2_TX PORT_Pin_13`  
Линия TX порта UART2.
- `#define PORT_UART1_EN MDR_PORTD`  
Порт линии EN микросхемы RS485 для UART1.
- `#define PORT_UART2_EN MDR_PORTD`  
Порт линии EN микросхемы RS485 для UART2.
- `#define PIN_UART1_EN PORT_Pin_10`  
Линия EN микросхемы RS485 для UART1.
- `#define PIN_UART2_EN PORT_Pin_12`  
Линия EN микросхемы RS485 для UART2.
- `#define UART_BUFFER_SIZE 16384`  
Размер кольцевого буфера UARTn (в кБайтах)
- `#define BUFFER_MASK (UART_BUFFER_SIZE-1)`

- Маска, необходимая для корректной работы кольцевого буфера
- `#define RECOGNIZE_BUS(ext_bus, uart_struct)`  
Макрос для определения по какой шине идет обмен данными
- `#define GET_UART_BUF_PTR uart_struct->buffer`  
Макрос возвращающий указатель на буфер UART.

## Определения типов

- `typedef enum errors uart_errors`  
Коды ошибок работы UART.
- `typedef struct uart_timeouts uart_rx_tx_timeouts`  
Структура с таймаутами UARTn.
- `typedef struct uart_dma_params uart_dma_ch_params`  
Структура с параметрами DMA канала UARTn.
- `typedef struct uart_config_data uart_n`  
Структура с конфигурационными параметрами UART и буфером приема

## Перечисления

- `enum errors {`  
`NO_ERRORS, INIT_ERROR, WRITE_TIMEOUT_ERROR, READ_TIMEOUT_ERROR,`  
`SIZE_ERROR, POSITION_ERROR }`  
 Коды ошибок работы UART.

## Функции

- `uart_errors uart_init (uart_n *uart_struct)`  
Инициализирует выбранный UARTn.
- `uart_errors uart_write (uart_n *uart_struct, uint8_t *data, uint32_t data_size)`  
Передаёт данные по UART.
- `uart_errors uart_read (uart_n *uart_struct, uint32_t len, uint8_t *data)`  
Читает данные из буфера UART.
- `void uart_clean (uart_n *uart_struct)`  
Очищает кольцевой буфер приемника UART.
- `uart_errors uart_set_pos (uart_n *uart_struct, uint32_t pos)`  
Устанавливает курсор чтения в кольцевом буфере
- `uint32_t uart_read_pos (uart_n *uart_struct)`  
Читает текущую позицию курсора чтения в кольцевом буфере
- `uint32_t uart_get_buf_counter (uart_n *uart_struct)`  
Читает кол-во байт в кольцевом буфере UART.
- `void DMA_UART_RX_init (uart_n *uart_struct)`  
Инициализирует n-ый канал DMA на запрос от приемника UARTn.
- `void uart_set_read_timeout (uart_n *uart_struct, uint32_t read_timeout)`  
Устанавливает таймаут UARTn на чтение
- `void uart_set_write_timeout (uart_n *uart_struct, uint32_t write_timeout)`  
Устанавливает таймаут UARTn на запись

### 4.29.1 Подробное описание

Заголовочный файл с описанием API для работы с UART.

### 4.29.2 Макросы

#### 4.29.2.1 BUFFER\_MASK

```
#define BUFFER_MASK (UART_BUFFER_SIZE-1)
```

Маска, необходимая для корректной работы кольцевого буфера

#### 4.29.2.2 GET\_UART\_BUF\_PTR

```
#define GET_UART_BUF_PTR uart_struct->buffer
```

Макрос возвращающий указатель на буфер UART.

#### 4.29.2.3 PIN\_UART1\_EN

```
#define PIN_UART1_EN PORT_Pin_10
```

Линия EN микросхемы RS485 для UART1.

#### 4.29.2.4 PIN\_UART1\_RX

```
#define PIN_UART1_RX PORT_Pin_4
```

Линия RX порта UART1.

#### 4.29.2.5 PIN\_UART1\_TX

```
#define PIN_UART1_TX PORT_Pin_3
```

Линия TX порта UART1.



## 4.29.2.6 PIN\_UART2\_EN

```
#define PIN_UART2_EN PORT_Pin_12
```

Линия EN микросхемы RS485 для UART2.

## 4.29.2.7 PIN\_UART2\_RX

```
#define PIN_UART2_RX PORT_Pin_14
```

Линия RX порта UART2.

## 4.29.2.8 PIN\_UART2\_TX

```
#define PIN_UART2_TX PORT_Pin_13
```

Линия TX порта UART2.

## 4.29.2.9 PORT\_UART1

```
#define PORT_UART1 MDR_PORTC
```

Порт UART1.

## 4.29.2.10 PORT\_UART1\_EN

```
#define PORT_UART1_EN MDR_PORTD
```

Порт линии EN микросхемы RS485 для UART1.

## 4.29.2.11 PORT\_UART2

```
#define PORT_UART2 MDR_PORTD
```

Порт UART2.

#### 4.29.2.12 PORT\_UART2\_EN

```
#define PORT_UART2_EN MDR_PORTD
```

Порт линии EN микросхемы RS485 для UART2.

#### 4.29.2.13 RECOGNIZE\_BUS

```
#define RECOGNIZE_BUS(  
    ext_bus,  
    uart_struct )
```

Макроопределение:

```
{  
    {\br/>    if(uart_struct->UARTx == MDR_UART1) ext_bus = 1;\br/>    else if(uart_struct->UARTx == MDR_UART2) ext_bus = 2;\br/>}
```

Макрос для определения по какой шине идет обмен данными

#### 4.29.2.14 UART\_BUFFER\_SIZE

```
#define UART_BUFFER_SIZE 16384
```

Размер кольцевого буфера UARTn (в кБайтах)

### 4.29.3 Типы

#### 4.29.3.1 uart\_dma\_ch\_params

```
typedef struct uart_dma_params uart_dma_ch_params
```

Структура с параметрами DMA канала UARTn.

#### 4.29.3.2 uart\_errors

```
typedef enum errors uart_errors
```

Коды ошибок работы UART.

4.29.3.3 `uart_n`

```
typedef struct uart_config_data uart_n
```

Структура с конфигурационными параметрами UART и буфером приема

4.29.3.4 `uart_rx_tx_timeouts`

```
typedef struct uart_timeouts uart_rx_tx_timeouts
```

Структура с таймаутами UARTn.

## 4.29.4 Перечисления

4.29.4.1 `errors`

```
enum errors
```

Коды ошибок работы UART.

Элементы перечислений

<code>NO_ERRORS</code>	Нет ошибок
<code>INIT_ERROR</code>	Ошибка инициализации UART.
<code>WRITE_TIMEOUT_ERROR</code>	Ошибка сигнализирует о том, что превышен таймаут на запись
<code>READ_TIMEOUT_ERROR</code>	Ошибка сигнализирует о том, что превышен таймаут на чтение
<code>SIZE_ERROR</code>	Ошибка сигнализирует о том, что либо в буфере недостаточно данных для чтения, либо размер записываемых/считываемых данных больше размера самого буфера
<code>POSITION_ERROR</code>	Ошибка установки курсора чтения

```
33 {
34     NO_ERRORS,
35     INIT_ERROR,
36     WRITE_TIMEOUT_ERROR,
37     READ_TIMEOUT_ERROR,
38     SIZE_ERROR,
39     POSITION_ERROR
40 } uart_errors;
```

## 4.29.5 Функции

## 4.29.5.1 DMA\_UART\_RX\_init()

```
void DMA_UART_RX_init (
    uart_n * uart_struct )
```

Инициализирует n-ый канал DMA на запрос от приемника UARTn.

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

```
379 {
380     DMA_StructInit(&uart_struct->uart_dma_ch.DMA_Channel_UART_RX);
381     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_SourceBaseAddr = (uint32_t)(&(uart_struct->
382     >UARTx->DR));
383     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestBaseAddr = (uint32_t)(uart_struct->buffer);
384     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_CycleSize = 1024;
385     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_SourceIncSize = DMA_SourceIncNo;
386     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestIncSize = DMA_DestIncByte;
387     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
388     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_NumContinuous = DMA_Transfers_1;
389     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_SourceProtCtrl = DMA_SourcePrivileged;
390     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_DestProtCtrl = DMA_DestPrivileged;
391     uart_struct->uart_dma_ch.DMA_InitStructure_UART_RX.DMA_Mode = DMA_Mode_Basic;
392
393     // Задать структуру канала
394     uart_struct->uart_dma_ch.DMA_Channel_UART_RX.DMA_PriCtrlData = &uart_struct->
395     >uart_dma_ch.DMA_InitStructure_UART_RX;
396     uart_struct->uart_dma_ch.DMA_Channel_UART_RX.DMA_Priority = DMA_Priority_High;
397     uart_struct->uart_dma_ch.DMA_Channel_UART_RX.DMA_UseBurst = DMA_BurstClear;
398     DMA_CTRL_DATA_PRIMARY;
399
400     // Инициализировать канал
401     DMA_Init(uart_struct->uart_dma_ch.dma_channel, &uart_struct->uart_dma_ch.DMA_Channel_UART_RX);
402
403     MDR_DMA->CHNL_REQ_MASK_CLR = 1 << uart_struct->uart_dma_ch.dma_channel;
404     MDR_DMA->CHNL_USEBURST_CLR = 1 << uart_struct->uart_dma_ch.dma_channel;
405
406     UART_DMACmd(uart_struct->UARTx, UART_DMA_RXE, ENABLE);
407     // Разрешить работу DMA с UART
408     DMA_Cmd (uart_struct->uart_dma_ch.dma_channel, ENABLE);
409
410     NVIC_SetPriority (DMA_IRQn, 1);
411     NVIC_EnableIRQ(DMA_IRQn);
412 }
```

## 4.29.5.2 uart\_clean()

```
void uart_clean (
    uart_n * uart_struct )
```

Очищает кольцевой буфер приемника UART.

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

```
372 {
373     memset(uart_struct->buffer, 0, UART_BUFFER_SIZE);
374 }
```

## 4.29.5.3 uart\_get\_buf\_counter()

```
uint32_t uart_get_buf_counter (
    uart_n * uart_struct )
```

Читает кол-во байт в кольцевом буфере UART.

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

Возвращает

Кол-во байт в кольцевом буфере UART

```
365 {
366     return uart_struct->buffer_count;
367 }
```

## 4.29.5.4 uart\_init()

```
uart_errors uart_init (
    uart_n * uart_struct )
```

Инициализирует выбранный UARTn.

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

Возвращает

Код ошибки uart\_errors

```
143 {
144     uart_gpio_config();
145
146     uart_errors error;
147
148     // Объявление структуры для инициализации контроллера UART
149     UART_InitTypeDef UART_InitStructure;
150     uint8_t res = 0;
151
152     // Включение тактирования UART
153     if(uart_struct->UARTx == MDR_UART1)
154     {
155         uart_struct->RST_CLK_PCLK_UARTn = RST_CLK_PCLK_UART1;
156     }
157     else if(uart_struct->UARTx == MDR_UART2)
158     {
159         uart_struct->RST_CLK_PCLK_UARTn = RST_CLK_PCLK_UART2;
160     }
161     RST_CLK_PCLKcmd(uart_struct->RST_CLK_PCLK_UARTn, ENABLE);
162
163     // Делитель тактовой частоты UART
164     uart_struct->UART_HCLKdiv = UART_HCLKdiv1;
165     UART_BRGInit(uart_struct->UARTx, uart_struct->UART_HCLKdiv);
166
167     NVIC_EnableIRQ(uart_struct->IRQn);
168
169     // Конфигурация UART
```

```

170  UART_InitStructure.UART_BaudRate = uart_struct->UART.UART_BaudRate;
171  UART_InitStructure.UART_WordLength = uart_struct->UART.UART_WordLength;
172  UART_InitStructure.UART_StopBits = uart_struct->UART.UART_StopBits;
173  UART_InitStructure.UART_Parity = uart_struct->UART.UART_Parity;
174  UART_InitStructure.UART_FIFOMode = uart_struct->UART.UART_FIFOMode;
175  UART_InitStructure.UART_HardwareFlowControl = uart_struct->UART.UART_HardwareFlowControl;
176
177  // Инициализация UART с заданными параметрами
178  res = UART_Init(uart_struct->UARTx, &UART_InitStructure);
179  if (res != SUCCESS)
180  {
181      error = INIT_ERROR;
182      return error;
183  }
184
185  // Включение прерываний UART
186  NVIC_SetPriority (uart_struct->IRQn, 0);
187  UART_ITConfig( uart_struct->UARTx, UART_IT_RX, ENABLE );
188
189  // Включение сконфигурированного UART
190  UART_Cmd(uart_struct->UARTx, ENABLE);
191
192  return NO_ERRORS;
193 }

```

#### 4.29.5.5 uart\_read()

```

uart_errors uart_read (
    uart_n * uart_struct,
    uint32_t len,
    uint8_t * data )

```

Читает данные из буфера UART.

Аргументы

*uart_struct	- Выбранный UART для инициализации
len	- Кол-во байт для чтения
*data	- Указатель на массив, куда считываются байты из буфера UART

Возвращает

Код ошибки uart\_errors

```

255 {
256     uart_errors error;
257     uint32_t timer_cnt = 0;
258
259     // Если длина превышает размер буфера
260     if (len > UART_BUFFER_SIZE)
261     {
262         error = SIZE_ERROR;
263         uart_struct->read_pos++;
264         return error;
265     }
266     // Если последний принятый байт перевалил границу буфера и байты будут перезаписываться в буфере с самого
    начала
267     if (((uart_struct->read_pos)+len) > UART_BUFFER_SIZE)
268     {
269         // Если задан таймаут
270         if (uart_struct->uart_timeouts.read_timeout_flag == 1)
271         {
272             TIMER_SetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx, 0);
273             while ((int)((uart_struct->buffer_count) - (uart_struct->read_pos)) >= 0)
274             {
275                 timer_cnt = TIMER_GetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx);
276                 if (timer_cnt >= (uart_struct->uart_timeouts.read_val_timeout*50))
277                 {
278                     error = READ_TIMEOUT_ERROR;

```

```

279         return error;
280     }
281 }
282 while ((UART_BUFFER_SIZE - (uart_struct->read_pos) + (uart_struct->buffer_count)) < len)
283 {
284     timer_cnt = TIMER_GetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx);
285     if (timer_cnt >= (uart_struct->uart_timeouts.read_val_timeout*50))
286     {
287         error = READ_TIMEOUT_ERROR;
288         return error;
289     }
290 }
291 }
292
293 if ((int)((uart_struct->buffer_count) - (uart_struct->read_pos)) >= 0)
294 {
295     error = SIZE_ERROR;
296     return error;
297 }
298 else
299 {
300     if ((UART_BUFFER_SIZE + (uart_struct->buffer_count) - (uart_struct->read_pos)) < len)
301     {
302         error = SIZE_ERROR;
303         return error;
304     }
305     memcpy(data, (uart_struct->buffer) + (uart_struct->read_pos), UART_BUFFER_SIZE-(uart_struct-
>read_pos));
306     memcpy(data + UART_BUFFER_SIZE-(uart_struct->read_pos), uart_struct->buffer, len+(uart_struct-
>read_pos)-UART_BUFFER_SIZE);
307     uart_struct->read_pos = (uart_struct->read_pos) + len-UART_BUFFER_SIZE;
308 }
309 }
310 // Если последний принятый байт не перевалил границу буфера
311 else
312 {
313     // Если задан таймаут
314     if (uart_struct->uart_timeouts.read_timeout_flag == 1)
315     {
316         TIMER_SetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx, 0);
317         while (((uart_struct->buffer_count) - (uart_struct->read_pos)) < len)
318         {
319             timer_cnt = TIMER_GetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx);
320             if (timer_cnt >= (uart_struct->uart_timeouts.read_val_timeout*50))
321             {
322                 error = READ_TIMEOUT_ERROR;
323                 return error;
324             }
325         }
326     }
327
328     if (((uart_struct->buffer_count) - (uart_struct->read_pos)) < len)
329     {
330         error = SIZE_ERROR;
331         return error;
332     }
333     memcpy(data, (uart_struct->buffer) + (uart_struct->read_pos), len);
334     uart_struct->read_pos = (uart_struct->read_pos) + len;
335 }
336
337 return NO_ERRORS;
338 }

```

#### 4.29.5.6 uart\_read\_pos()

```

uint32_t uart_read_pos (
    uart_n * uart_struct )

```

Читает текущую позицию курсора чтения в кольцевом буфере

Аргументы

*uart_struct	- Выбранный UART для инициализации
--------------	------------------------------------

Возвращает

Текущая позиция курсора

```
358 {
359     return uart_struct->read_pos;
360 }
```

#### 4.29.5.7 uart\_set\_pos()

```
uart_errors uart_set_pos (
    uart_n * uart_struct,
    uint32_t pos )
```

Устанавливает курсор чтения в кольцевом буфере

Аргументы

*uart_struct	- Выбранный UART для инициализации
pos	- Позиция курсора в буфере

Возвращает

Код ошибки uart\_errors

```
343 {
344     uart_errors error;
345
346     if (pos > UART_BUFFER_SIZE)
347     {
348         error = POSITION_ERROR;
349         return error;
350     }
351     uart_struct->read_pos = pos;
352     return NO_ERRORS;
353 }
```

#### 4.29.5.8 uart\_set\_read\_timeout()

```
void uart_set_read_timeout (
    uart_n * uart_struct,
    uint32_t read_timeout )
```

Устанавливает таймаут UARTn на чтение

Аргументы

*uart_struct	- Выбранный UART для инициализации
read_timeout	- Таймаут на чтение (в мс)

```
416 {
417     uart_struct->uart_timeouts.read_timeout_flag = 1;
418     uart_struct->uart_timeouts.read_val_timeout = read_timeout;
419 }
```



## 4.29.5.9 uart\_set\_write\_timeout()

```
void uart_set_write_timeout (
    uart_n * uart_struct,
    uint32_t write_timeout )
```

Устанавливает таймаут UARTn на запись

Аргументы

*uart_struct	- Выбранный UART для инициализации
write_timeout	- Таймаут на запись (в мс)

```
424 {
425     uart_struct->uart_timeouts.write_timeout_flag = 1;
426     uart_struct->uart_timeouts.write_val_timeout = write_timeout;
427 }
```

## 4.29.5.10 uart\_write()

```
uart_errors uart_write (
    uart_n * uart_struct,
    uint8_t * data,
    uint32_t data_size )
```

Передаёт данные по UART.

Аргументы

*uart_struct	- Выбранный UART для инициализации
*data	- Указатель на массив данных
data_size	- Размер данных в байтах

Возвращает

Код ошибки uart\_errors

```
198 {
199     uart_errors error;
200     uint32_t timer_cnt = 0;
201
202     // Активирование микросхемы RS485 на выдачу данных
203     if (uart_struct->UARTx == MDR_UART1)
204     {
205         PORT_WriteBit(PORT_UART1_EN, PIN_UART1_EN, 1);
206     }
207     else if (uart_struct->UARTx == MDR_UART2)
208     {
209         PORT_WriteBit(PORT_UART2_EN, PIN_UART2_EN, 1);
210     }
211
212     if (uart_struct->uart_timeouts.write_timeout_flag == 1)
213     {
214         TIMER_SetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx, 0);
215     }
```

```
216 if (data_size > UART_BUFFER_SIZE)
217 {
218     error = SIZE_ERROR;
219     return error;
220 }
221 for (int i = 0; i < data_size; i++)
222 {
223     UART_SendData(uart_struct->UARTx, data[i]);
224     while (UART_GetFlagStatus(uart_struct->UARTx, UART_FLAG_TXFF) == SET)
225     {
226         if (uart_struct->uart_timeouts.write_timeout_flag == 1)
227         {
228             timer_cnt = TIMER_GetCounter(uart_struct->uart_timeouts.timer_n_timeout->TIMERx);
229             if (timer_cnt >= (uart_struct->uart_timeouts.write_val_timeout*50))
230             {
231                 error = WRITE_TIMEOUT_ERROR;
232                 return error;
233             }
234         }
235     }
236 }
237 // Небольшая задержка для микросхемы RS-485
238 delay_micro(10);
239 // Деактивирование микросхемы RS485 на прием данных
240 if (uart_struct->UARTx == MDR_UART1)
241 {
242     PORT_WriteBit(PORT_UART1_EN, PIN_UART1_EN, 0);
243 }
244 else if (uart_struct->UARTx == MDR_UART2)
245 {
246     PORT_WriteBit(PORT_UART2_EN, PIN_UART2_EN, 0);
247 }
248
249 return NO_ERRORS;
250 }
```

# Предметный указатель

- \_\_list\_add
    - list.c, [127](#)
    - list.h, [133](#)
  - \_\_list\_del
    - list.c, [128](#)
    - list.h, [133](#)
- 1273pv19t.c, [65](#)
  - adc\_1, [68](#)
  - adc\_gpio\_config, [66](#)
  - adc\_init, [67](#)
  - adc\_reset, [68](#)
  - spi\_1, [68](#)
  - timer\_2, [68](#)
- 1273pv19t.h, [69](#)
  - adc\_init, [72](#)
  - adc\_n, [72](#)
  - adc\_reset, [73](#)
  - PIN\_ADC\_MODE\_A0, [70](#)
  - PIN\_ADC\_MODE\_A1, [70](#)
  - PIN\_ADC\_NSS, [71](#)
  - PIN\_ADC\_RST, [71](#)
  - PIN\_ADC\_SDIFS\_IRQ, [71](#)
  - PORT\_ADC\_MODE, [71](#)
  - PORT\_ADC\_NSS, [71](#)
  - PORT\_ADC\_RST, [71](#)
  - PORT\_ADC\_SDIFS\_IRQ, [72](#)
- adc\_1
  - 1273pv19t.c, [68](#)
  - dma.c, [79](#)
  - main.c, [142](#)
  - timers.c, [195](#)
- adc\_chs\_mode
  - ai\_oper\_mode\_struct, [7](#)
- adc\_config\_data, [5](#)
  - avg\_num, [6](#)
  - ch\_rx\_num, [6](#)
  - init\_flag, [6](#)
  - spi\_struct, [6](#)
  - timer\_n\_capture, [6](#)
- adc\_gpio\_config
  - 1273pv19t.c, [66](#)
- adc\_init
  - 1273pv19t.c, [67](#)
  - 1273pv19t.h, [72](#)
- adc\_n
  - 1273pv19t.h, [72](#)
- adc\_reset
  - 1273pv19t.c, [68](#)
- 1273pv19t.h, [73](#)
- add\_info
  - plc\_sof\_ver\_struct, [34](#)
- add\_switch\_1
  - config\_struct, [19](#)
- add\_switch\_2
  - config\_struct, [20](#)
- address
  - range\_start\_struct, [41](#)
- AI\_CodeADC
  - mpa\_register\_space\_ext\_ram, [26](#)
- AI\_DiagnosticChannel
  - mpa\_register\_space\_ext\_ram, [26](#)
- AI\_MaxCodeADC
  - mpa\_register\_space\_ext\_rom, [28](#)
- AI\_MetrologDat
  - mpa\_register\_space\_ext\_rom, [28](#)
- AI\_MinCodeADC
  - mpa\_register\_space\_ext\_rom, [29](#)
- AI\_NumForAverag
  - mpa\_register\_space\_ext\_rom, [29](#)
- ai\_oper\_mode
  - external\_rom.h, [114](#)
- ai\_oper\_mode\_struct, [7](#)
  - adc\_chs\_mode, [7](#)
  - reserv1, [7](#)
- AI\_OperMode
  - mpa\_register\_space\_ext\_rom, [29](#)
- AI\_PhysQuantFloat
  - mpa\_register\_space\_ext\_ram, [26](#)
- AI\_PolynConst0
  - mpa\_register\_space\_ext\_rom, [29](#)
- AI\_PolynConst1
  - mpa\_register\_space\_ext\_rom, [29](#)
- AI\_PolynConst2
  - mpa\_register\_space\_ext\_rom, [29](#)
- AI\_PolynConst3
  - mpa\_register\_space\_ext\_rom, [30](#)
- AI\_PolynConst4
  - mpa\_register\_space\_ext\_rom, [30](#)
- AI\_PolynConst5
  - mpa\_register\_space\_ext\_rom, [30](#)
- AI\_PolynConst6
  - mpa\_register\_space\_ext\_rom, [30](#)
- AI\_RomRegs
  - mpa\_register\_space\_ext\_ram, [26](#)
- AI\_SignalChanged
  - mpa\_register\_space\_ext\_ram, [27](#)
- avg\_num

- adc\_config\_data, 6
- BATCH
  - external\_rom.h, 110
- batch
  - device\_type\_struct, 22
- both\_control
  - service\_byte\_struct\_pm, 46
- buffer
  - spi\_config\_data, 49
  - uart\_config\_data, 60
- buffer\_count
  - uart\_config\_data, 60
- buffer\_counter
  - spi\_config\_data, 50
- BUFFER\_MASK
  - uart.h, 212
- bus\_defect
  - external\_ram.h, 99
- bus\_defect\_struct, 7
  - fail\_timeout, 8
  - many\_fail\_packet, 8
  - reserv, 8
- ch\_rx\_num
  - adc\_config\_data, 6
- CHANEL\_NUMBER
  - mdr32\_drivers.h, 149
- chassis\_addr
  - device\_address\_struct, 21
- checksum
  - packet\_tail\_Struct, 33
- clock.c, 74
  - clock\_init, 74
- clock.h, 75
  - clock\_init, 75
- clock\_init
  - clock.c, 74
  - clock.h, 75
- CLOCK\_LEDS
  - leds.h, 125
- cmd
  - cmd\_header\_struct, 9
- cmd\_header\_struct, 8
  - cmd, 9
  - length, 9
  - result, 9
- cmd\_number
  - packet\_header\_struct, 31
- cmd\_struct, 10
  - data, 10
  - header, 10
- cmd\_with\_data
  - tx\_rx\_packet\_struct, 58
- common\_ram\_register\_space
  - ram\_data\_struct, 38
- common\_ram\_registers
  - external\_ram.h, 99
- common\_register\_space\_ext\_ram, 11
  - PLC\_BusDefect\_B1, 12
  - PLC\_BusDefect\_B2, 12
  - PLC\_CM\_State, 12
  - PLC\_CommonRomRegs, 13
  - PLC\_Config, 13
  - PLC\_CorrPackFromDevice\_B1, 13
  - PLC\_CorrPackFromDevice\_B2, 13
  - PLC\_CorrPackToDevice\_B1, 13
  - PLC\_CorrPackToDevice\_B2, 13
  - PLC\_Durat, 14
  - PLC\_ErrPackFromDevice\_B1, 14
  - PLC\_ErrPackFromDevice\_B2, 14
  - PLC\_ErrPackToDevice\_B1, 14
  - PLC\_ErrPackToDevice\_B2, 14
  - PLC\_PMAAddr, 14
  - PLC\_PowerDefect, 15
  - PLC\_SelfDiagDefect, 15
  - PLC\_SoftVer, 15
  - Reserv\_1, 15
- common\_register\_space\_ext\_rom, 16
  - PLC\_BusConfig\_B1, 17
  - PLC\_BusConfig\_B2, 17
  - PLC\_DeviceInfo, 17
  - PLC\_DeviceType, 17
  - PLC\_DualControl, 17
  - PLC\_NumCrcErrorsForDefect\_B1, 17
  - PLC\_NumCrcErrorsForDefect\_B2, 18
  - PLC\_SerialNumber, 18
  - PLC\_TimeoutForDefect\_B1, 18
  - PLC\_TimeoutForDefect\_B2, 18
  - PLC\_TimeSoloWork, 18
  - PLC\_TimeToRepair, 18
  - Reserv\_2, 19
- common\_rom\_registers
  - external\_rom.h, 114
- common\_rom\_registers\_space
  - rom\_data\_struct, 43
- CONFIG
  - rs422\_protocol.h, 164
- config\_struct, 19
  - add\_switch\_1, 19
  - add\_switch\_2, 20
  - main\_switch, 20
  - reserv, 20
- container\_of
  - list.h, 131
- crc32
  - rs422\_protocol.c, 152
  - rs422\_protocol.h, 169
- CRC\_ERROR
  - rs422\_protocol.h, 168
- crc\_table
  - ram\_data\_struct, 38
- data
  - cmd\_struct, 10
  - timer\_irq\_list\_struct, 56
- delay\_micro
  - timers.c, 191

- timers.h, 198
- delay\_milli
  - timers.c, 191
  - timers.h, 198
- DEV\_INFO
  - external\_rom.h, 110
- DEV\_TYPE
  - external\_rom.h, 110
- DEV\_TYPE\_RESERV
  - external\_rom.h, 111
- develop
  - plc\_sof\_ver\_struct, 34
- device\_address
  - external\_ram.h, 99
- device\_address\_struct, 20
  - chassis\_addr, 21
  - module\_addr, 21
  - reserv, 21
- device\_config
  - external\_ram.h, 100
- device\_type
  - external\_rom.h, 114
- device\_type\_struct, 21
  - batch, 22
  - modification, 22
  - reserv, 22
  - revision, 22
  - type, 23
  - use\_object, 23
- devices
  - ebc.h, 85
- dma.c, 76
  - adc\_1, 79
  - dma\_common\_init, 77
  - DMA\_IRQHandler, 77
  - spi\_1, 79
  - spi\_2, 79
  - timer\_1, 79
  - timer\_2, 79
  - timer\_3, 79
  - uart\_1, 79
  - uart\_2, 80
- dma.h, 80
  - dma\_common\_init, 81
- dma\_channel
  - spi\_dma\_params, 51
  - uart\_dma\_params, 62
- DMA\_Channel\_SPI\_RX
  - spi\_dma\_params, 51
- DMA\_Channel\_UART\_RX
  - uart\_dma\_params, 62
- dma\_common\_init
  - dma.c, 77
  - dma.h, 81
- DMA\_InitStructure\_SPI\_RX
  - spi\_dma\_params, 52
- DMA\_InitStructure\_UART\_RX
  - uart\_dma\_params, 62
- dma\_irq\_counter
  - spi\_dma\_params, 52
  - uart\_dma\_params, 62
- DMA\_IRQHandler
  - dma.c, 77
- dma\_spi\_rx\_init
  - spi.c, 180
  - spi.h, 187
- DMA\_UART\_RX\_init
  - uart.c, 201
  - uart.h, 215
- do\_mpa\_task
  - main.c, 137
  - main.h, 145
- DUAL\_CONTROL
  - external\_rom.h, 111
- ebc.c, 81
  - ebc\_gpio\_config, 82
  - ebc\_init, 83
- ebc.h, 84
  - devices, 85
  - ebc\_devices, 85
  - ebc\_init, 86
  - EBC\_RAM, 86
  - EBC\_ROM, 86
- ebc\_devices
  - ebc.h, 85
- ebc\_gpio\_config
  - ebc.c, 82
- ebc\_init
  - ebc.c, 83
  - ebc.h, 86
- EBC\_RAM
  - ebc.h, 86
- EBC\_ROM
  - ebc.h, 86
- end
  - packet\_tail\_Struct, 33
- erase\_rom
  - external\_rom.c, 105
  - external\_rom.h, 115
- errors
  - uart.h, 215
- event
  - timer\_irq\_list\_struct, 56
- EXT\_RAM\_START\_ADDR
  - external\_ram.h, 94
- EXT\_ROM\_START\_ADDR
  - external\_rom.h, 111
- external\_ram.c, 87
  - find\_max\_halfword, 87
  - init\_external\_ram\_space, 88
  - polyn\_ch\_consts, 89
  - ram\_space\_pointer, 90
  - rom\_space\_pointer, 90
- external\_ram.h, 90
  - bus\_defect, 99
  - common\_ram\_registers, 99

- device\_address, 99
- device\_config, 100
- EXT\_RAM\_START\_ADDR, 94
- init\_external\_ram\_space, 102
- module\_type, 100
- MPA, 102
- mpa\_ram\_registers, 100
- MPCH, 102
- MPD, 102
- MPI, 102
- MPI\_U, 102
- MPPT, 102
- MPT, 102
- MSR, 102
- MVA, 102
- MVD, 102
- MVD\_U, 102
- PLC\_CONFIG\_ADD\_SWITCH1, 94
- PLC\_CONFIG\_ADD\_SWITCH2, 94
- PLC\_CONFIG\_MAIN\_SWITCH, 94
- PLC\_CONFIG\_RESERV, 94
- PLC\_PM\_CHASSIS\_ADDR, 94
- PLC\_PM\_MODULE\_ADDR, 94
- plc\_soft\_ver, 100
- PLC\_SOFT\_VER\_ADD\_INFO, 95
- PLC\_SOFT\_VER\_DEVELOP, 95
- PLC\_SOFT\_VER\_MODIFICATION, 95
- PLC\_SOFT\_VER\_REVISION, 95
- PLC\_SOFT\_VER\_SOFT\_VER, 95
- PLC\_SOFT\_VER\_TYPE, 95
- power\_failure, 100
- ram\_data, 100
- RAM\_REGISTER\_SPACE\_START\_ADDR, 96
- ram\_start\_struct, 101
- range, 101
- RESET\_BIT, 96
- self\_diag, 101
- service\_struct\_pm, 101
- service\_struct\_um, 101
- SET\_BIT, 96
- START\_STRUCT\_CHANGE\_FLAG, 96
- START\_STRUCT\_LENGTH, 96
- START\_STRUCT\_NUMBER\_OF\_RANGES, 96
- START\_STRUCT\_RANGE0\_ADDR, 97
- START\_STRUCT\_RANGE0\_SIZE, 97
- START\_STRUCT\_RANGE0\_START\_CH\_NUM, 97
- START\_STRUCT\_RANGE0\_TYPE, 97
- START\_STRUCT\_RANGE1\_ADDR, 97
- START\_STRUCT\_RANGE1\_SIZE, 97
- START\_STRUCT\_RANGE1\_START\_CH\_NUM, 98
- START\_STRUCT\_RANGE1\_TYPE, 98
- START\_STRUCT\_RANGE2\_ADDR, 98
- START\_STRUCT\_RANGE2\_SIZE, 98
- START\_STRUCT\_RANGE2\_START\_CH\_NUM, 98
- START\_STRUCT\_RANGE2\_TYPE, 98
- START\_STRUCT\_TEXT\_INFO\_ADDR, 99
- TEST\_BIT, 99
- TIMER\_NUM, 99
- type\_of\_module, 101
- external\_rom.c, 104
- erase\_rom, 105
- FIRST\_TIME\_INIT, 105
- init\_external\_rom\_space, 105
- memcpy\_to\_rom, 106
- polyn\_ch\_consts, 107
- read\_byte\_rom, 106
- write\_byte\_rom, 107
- external\_rom.h, 108
- ai\_oper\_mode, 114
- BATCH, 110
- common\_rom\_registers, 114
- DEV\_INFO, 110
- DEV\_TYPE, 110
- DEV\_TYPE\_RESERV, 111
- device\_type, 114
- DUAL\_CONTROL, 111
- erase\_rom, 115
- EXT\_ROM\_START\_ADDR, 111
- HWREG, 111
- init\_external\_rom\_space, 115
- MAX\_CODE\_ADC, 111
- memcpy\_to\_rom, 116
- METROLOG\_DAT, 111
- MIN\_CODE\_ADC, 112
- MODIFICATION, 112
- mpa\_rom\_registers, 114
- NUM\_CRC\_ERRORS\_FOR\_DEFECT\_B1, 112
- NUM\_CRC\_ERRORS\_FOR\_DEFECT\_B2, 112
- NUM\_FOR\_AVERAGE, 112
- read\_byte\_rom, 116
- REVISION, 112
- rom\_data, 114
- ROM\_REGISTER\_SPACE\_START\_ADDR, 113
- SERIAL\_NUMBER, 113
- TIME\_SOLO\_WORK, 113
- TIME\_TO\_REPAIR, 113
- TIMEOUT\_FOR\_DEFECT\_B1, 113
- TIMEOUT\_FOR\_DEFECT\_B2, 113
- write\_byte\_rom, 117
- fail\_bus\_1
- service\_byte\_struct\_pm, 46
- fail\_bus\_2
- service\_byte\_struct\_pm, 46
- fail\_chanel
- self\_diag\_struct, 44
- fail\_crc\_firmware

- self\_diag\_struct, [44](#)
- fail\_download\_rom
  - self\_diag\_struct, [44](#)
- fail\_firmware\_1\_bus
  - self\_diag\_struct, [44](#)
- fail\_firmware\_2\_bus
  - self\_diag\_struct, [44](#)
- fail\_firmware\_ram
  - self\_diag\_struct, [44](#)
- fail\_soft\_ver
  - self\_diag\_struct, [45](#)
- fail\_timeout
  - bus\_defect\_struct, [8](#)
- fields\_cmd
  - rs422\_protocol.h, [167](#)
- fields\_cmd\_header
  - rs422\_protocol.h, [167](#)
- fields\_packet
  - rs422\_protocol.h, [167](#)
- fields\_packet\_header
  - rs422\_protocol.h, [167](#)
- fields\_packet\_tail
  - rs422\_protocol.h, [168](#)
- FIFO\_SIZE
  - spi.h, [185](#)
- fill\_crc32\_table
  - rs422\_protocol.c, [152](#)
  - rs422\_protocol.h, [169](#)
- find\_max\_halfword
  - external\_ram.c, [87](#)
- FIRST\_TIME\_INIT
  - external\_rom.c, [105](#)
- flag\_change\_struct
  - start\_struct\_ext\_ram, [53](#)
- free\_ram\_pages
  - internal\_ram.c, [118](#)
  - internal\_ram.h, [121](#)
- GET\_UART\_BUF\_PTR
  - uart.h, [212](#)
- handler
  - timer\_irq\_list\_struct, [57](#)
- header
  - cmd\_struct, [10](#)
  - packet\_header\_struct, [32](#)
- heap
  - main.c, [142](#)
- heap\_ptr
  - internal\_ram.c, [119](#)
  - main.c, [142](#)
- heap\_struct, [23](#)
  - memory\_page\_space, [23](#)
  - memory\_page\_status, [24](#)
- HSE\_OSC
  - mdr32\_drivers.h, [150](#)
- HWREG
  - external\_rom.h, [111](#)
- init
  - service\_byte\_struct\_pm, [46](#)
- INIT\_CMD
  - rs422\_protocol.h, [165](#)
- INIT\_ERROR
  - uart.h, [215](#)
- init\_external\_ram\_space
  - external\_ram.c, [88](#)
  - external\_ram.h, [102](#)
- init\_external\_rom\_space
  - external\_rom.c, [105](#)
  - external\_rom.h, [115](#)
- init\_flag
  - adc\_config\_data, [6](#)
- init\_list\_head
  - list.c, [128](#)
  - list.h, [134](#)
- int\_ram\_heap
  - internal\_ram.h, [121](#)
- internal\_ram.c, [118](#)
  - free\_ram\_pages, [118](#)
  - heap\_ptr, [119](#)
  - malloc\_ram\_pages, [118](#)
- internal\_ram.h, [119](#)
  - free\_ram\_pages, [121](#)
  - int\_ram\_heap, [121](#)
  - malloc\_ram\_pages, [122](#)
  - PAGE\_NUM, [121](#)
  - PAGE\_SIZE, [121](#)
- IRQn
  - spi\_config\_data, [50](#)
  - uart\_config\_data, [60](#)
- K1986VE1T
  - mdr32\_drivers.h, [150](#)
- last\_answer
  - service\_byte\_struct\_um, [48](#)
- leds.c, [123](#)
  - leds\_gpio\_config, [123](#)
- leds.h, [124](#)
  - CLOCK\_LEDS, [125](#)
  - leds\_gpio\_config, [126](#)
  - PIN\_LED\_ERROR\_WORK, [125](#)
  - PIN\_LED\_OK\_WORK, [125](#)
  - PORT\_LEDS, [125](#)
  - RESET\_LED\_ERROR\_WORK, [125](#)
  - RESET\_LED\_OK\_WORK, [126](#)
  - SET\_LED\_ERROR\_WORK, [126](#)
  - SET\_LED\_OK\_WORK, [126](#)
- leds\_gpio\_config
  - leds.c, [123](#)
  - leds.h, [126](#)
- length
  - cmd\_header\_struct, [9](#)
  - start\_struct\_ext\_ram, [53](#)
- list
  - timer\_irq\_list\_struct, [57](#)
- list.c, [127](#)

- \_\_list\_add, 127
- \_\_list\_del, 128
- init\_list\_head, 128
- list\_add, 128
- list\_add\_tail, 128
- list\_del, 129
- list\_empty, 129
- list\_is\_last, 129
- list.h, 130
  - \_\_list\_add, 133
  - \_\_list\_del, 133
  - container\_of, 131
  - init\_list\_head, 134
  - list\_add, 134
  - list\_add\_tail, 134
  - list\_del, 135
  - list\_empty, 135
  - list\_entry, 131
  - list\_for\_each, 132
  - list\_for\_each\_entry, 132
  - LIST\_HEAD, 132
  - list\_head, 133
  - LIST\_HEAD\_INIT, 132
  - list\_is\_last, 135
- list\_add
  - list.c, 128
  - list.h, 134
- list\_add\_tail
  - list.c, 128
  - list.h, 134
- list\_del
  - list.c, 129
  - list.h, 135
- list\_empty
  - list.c, 129
  - list.h, 135
- list\_entry
  - list.h, 131
- list\_for\_each
  - list.h, 132
- list\_for\_each\_entry
  - list.h, 132
- LIST\_HEAD
  - list.h, 132
- list\_head
  - list.h, 133
- LIST\_HEAD\_INIT
  - list.h, 132
- list\_head\_struct, 24
  - next, 25
  - prev, 25
- list\_is\_last
  - list.c, 129
  - list.h, 135
- list\_tmr\_handler\_add\_tail
  - timers.c, 192
  - timers.h, 198
- list\_tmr\_handler\_init
  - timers.c, 192
  - timers.h, 199
- main
  - main.c, 138
- main.c, 136
  - adc\_1, 142
  - do\_mpa\_task, 137
  - heap, 142
  - heap\_ptr, 142
  - main, 138
  - ram\_space\_pointer, 142
  - receive\_adc\_chanel\_pack, 140
  - request\_data, 140
  - rom\_space\_pointer, 142
  - spi\_1, 143
  - spi\_2, 143
  - sync\_adc\_chanel, 141
  - timer\_1, 143
  - timer\_2, 143
  - timer\_3, 143
  - tmr\_handler\_head, 143
  - uart\_1, 144
  - uart\_2, 144
- main.h, 144
  - do\_mpa\_task, 145
  - receive\_adc\_chanel\_pack, 146
  - request\_data, 147
  - sync\_adc\_chanel, 148
- main\_switch
  - config\_struct, 20
- malloc\_ram\_pages
  - internal\_ram.c, 118
  - internal\_ram.h, 122
- many\_fail\_packet
  - bus\_defect\_struct, 8
- master
  - service\_byte\_struct\_pm, 47
- MAX\_CHANEL\_NUMBER
  - mdr32\_drivers.h, 150
- MAX\_CODE\_ADC
  - external\_rom.h, 111
- mdr32\_drivers.h, 148
  - CHANEL\_NUMBER, 149
  - HSE\_OSC, 150
  - K1986VE1T, 150
  - MAX\_CHANEL\_NUMBER, 150
  - PM\_CHASSIS\_ADDR, 150
  - PM\_DEV\_ADDR, 150
  - WORK\_FREQ, 150
- memcpy\_to\_rom
  - external\_rom.c, 106
  - external\_rom.h, 116
- memory\_page\_space
  - heap\_struct, 23
- memory\_page\_status
  - heap\_struct, 24
- METROLOG\_DAT
  - external\_rom.h, 111



- MIN\_CODE\_ADC
  - external\_rom.h, [112](#)
- MODIFICATION
  - external\_rom.h, [112](#)
- modification
  - device\_type\_struct, [22](#)
  - plc\_sof\_ver\_struct, [34](#)
- module\_addr
  - device\_address\_struct, [21](#)
- module\_type
  - external\_ram.h, [100](#)
- MPA
  - external\_ram.h, [102](#)
- mpa\_ram\_register\_space
  - ram\_data\_struct, [38](#)
- mpa\_ram\_registers
  - external\_ram.h, [100](#)
- mpa\_register\_space\_ext\_ram, [25](#)
  - AI\_CodeADC, [26](#)
  - AI\_DiagnosticChannel, [26](#)
  - AI\_PhysQuantFloat, [26](#)
  - AI\_RomRegs, [26](#)
  - AI\_SignalChanged, [27](#)
  - Reserv\_6, [27](#)
- mpa\_register\_space\_ext\_rom, [27](#)
  - AI\_MaxCodeADC, [28](#)
  - AI\_MetrologDat, [28](#)
  - AI\_MinCodeADC, [29](#)
  - AI\_NumForAverag, [29](#)
  - AI\_OperMode, [29](#)
  - AI\_PolynConst0, [29](#)
  - AI\_PolynConst1, [29](#)
  - AI\_PolynConst2, [29](#)
  - AI\_PolynConst3, [30](#)
  - AI\_PolynConst4, [30](#)
  - AI\_PolynConst5, [30](#)
  - AI\_PolynConst6, [30](#)
  - Reserv\_3, [30](#)
  - Reserv\_4, [30](#)
  - Reserv\_5, [31](#)
- mpa\_rom\_registers
  - external\_rom.h, [114](#)
- mpa\_rom\_registers\_space
  - rom\_data\_struct, [43](#)
- MPCH
  - external\_ram.h, [102](#)
- MPD
  - external\_ram.h, [102](#)
- MPI
  - external\_ram.h, [102](#)
- MPI\_U
  - external\_ram.h, [102](#)
- MPPT
  - external\_ram.h, [102](#)
- MPT
  - external\_ram.h, [102](#)
- MSR
  - external\_ram.h, [102](#)
- MVA
  - external\_ram.h, [102](#)
- MVD
  - external\_ram.h, [102](#)
- MVD\_U
  - external\_ram.h, [102](#)
- next
  - list\_head\_struct, [25](#)
- NO\_ERROR
  - rs422\_protocol.h, [168](#)
- NO\_ERRORS
  - uart.h, [215](#)
- NUM\_CRC\_ERRORS\_FOR\_DEFECT\_B1
  - external\_rom.h, [112](#)
- NUM\_CRC\_ERRORS\_FOR\_DEFECT\_B2
  - external\_rom.h, [112](#)
- NUM\_FOR\_AVERAGE
  - external\_rom.h, [112](#)
- NUMBER\_CMDS\_IN\_PACKET
  - rs422\_protocol.h, [165](#)
- number\_of\_ranges
  - start\_struct\_ext\_ram, [53](#)
- PACKET\_ERROR
  - rs422\_protocol.h, [168](#)
- PACKET\_HEAD
  - rs422\_protocol.h, [165](#)
- packet\_header
  - tx\_rx\_packet\_struct, [58](#)
- packet\_header\_struct, [31](#)
  - cmd\_number, [31](#)
  - header, [32](#)
  - packet\_length, [32](#)
  - receiver\_addr, [32](#)
  - sender\_addr, [32](#)
  - service\_byte, [32](#)
- packet\_length
  - packet\_header\_struct, [32](#)
- packet\_rx
  - ram\_data\_struct, [38](#)
- PACKET\_TAIL
  - rs422\_protocol.h, [165](#)
- packet\_tail
  - tx\_rx\_packet\_struct, [58](#)
- packet\_tail\_Struct, [33](#)
  - checksum, [33](#)
  - end, [33](#)
- packet\_tx
  - ram\_data\_struct, [38](#)
- PAGE\_NUM
  - internal\_ram.h, [121](#)
- PAGE\_SIZE
  - internal\_ram.h, [121](#)
- PIN\_ADC\_MODE\_A0
  - 1273pv19t.h, [70](#)
- PIN\_ADC\_MODE\_A1
  - 1273pv19t.h, [70](#)
- PIN\_ADC\_NSS

- 1273pv19t.h, [71](#)
- PIN\_ADC\_RST
  - 1273pv19t.h, [71](#)
- PIN\_ADC\_SDIFS\_IRQ
  - 1273pv19t.h, [71](#)
- PIN\_LED\_ERROR\_WORK
  - leds.h, [125](#)
- PIN\_LED\_OK\_WORK
  - leds.h, [125](#)
- PIN\_SSP1\_RX
  - spi.h, [185](#)
- PIN\_SSP1\_SCK
  - spi.h, [185](#)
- PIN\_SSP1\_SS
  - spi.h, [186](#)
- PIN\_SSP1\_TX
  - spi.h, [186](#)
- PIN\_UART1\_EN
  - uart.h, [212](#)
- PIN\_UART1\_RX
  - uart.h, [212](#)
- PIN\_UART1\_TX
  - uart.h, [212](#)
- PIN\_UART2\_EN
  - uart.h, [212](#)
- PIN\_UART2\_RX
  - uart.h, [213](#)
- PIN\_UART2\_TX
  - uart.h, [213](#)
- PLC\_BusConfig\_B1
  - common\_register\_space\_ext\_rom, [17](#)
- PLC\_BusConfig\_B2
  - common\_register\_space\_ext\_rom, [17](#)
- PLC\_BusDefect\_B1
  - common\_register\_space\_ext\_ram, [12](#)
- PLC\_BusDefect\_B2
  - common\_register\_space\_ext\_ram, [12](#)
- PLC\_CM\_CRITICAL\_FAULT
  - rs422\_protocol.h, [165](#)
- PLC\_CM\_INIT\_1\_BUS
  - rs422\_protocol.h, [165](#)
- PLC\_CM\_INIT\_2\_BUS
  - rs422\_protocol.h, [166](#)
- PLC\_CM\_NOT\_INIT
  - rs422\_protocol.h, [166](#)
- PLC\_CM\_REMOVE\_INIT
  - rs422\_protocol.h, [166](#)
- PLC\_CM\_State
  - common\_register\_space\_ext\_ram, [12](#)
- PLC\_CM\_UNKNOWN\_STATE
  - rs422\_protocol.h, [166](#)
- PLC\_CommonRomRegs
  - common\_register\_space\_ext\_ram, [13](#)
- PLC\_Config
  - common\_register\_space\_ext\_ram, [13](#)
- PLC\_CONFIG\_ADD\_SWITCH1
  - external\_ram.h, [94](#)
- PLC\_CONFIG\_ADD\_SWITCH2
  - external\_ram.h, [94](#)
- PLC\_CONFIG\_MAIN\_SWITCH
  - external\_ram.h, [94](#)
- PLC\_CONFIG\_RESERV
  - external\_ram.h, [94](#)
- PLC\_CorrPackFromDevice\_B1
  - common\_register\_space\_ext\_ram, [13](#)
- PLC\_CorrPackFromDevice\_B2
  - common\_register\_space\_ext\_ram, [13](#)
- PLC\_CorrPackToDevice\_B1
  - common\_register\_space\_ext\_ram, [13](#)
- PLC\_CorrPackToDevice\_B2
  - common\_register\_space\_ext\_ram, [13](#)
- PLC\_DeviceInfo
  - common\_register\_space\_ext\_rom, [17](#)
- PLC\_DeviceType
  - common\_register\_space\_ext\_rom, [17](#)
- PLC\_DualControl
  - common\_register\_space\_ext\_rom, [17](#)
- PLC\_Durat
  - common\_register\_space\_ext\_ram, [14](#)
- PLC\_ErrPackFromDevice\_B1
  - common\_register\_space\_ext\_ram, [14](#)
- PLC\_ErrPackFromDevice\_B2
  - common\_register\_space\_ext\_ram, [14](#)
- PLC\_ErrPackToDevice\_B1
  - common\_register\_space\_ext\_ram, [14](#)
- PLC\_ErrPackToDevice\_B2
  - common\_register\_space\_ext\_ram, [14](#)
- PLC\_NumCrcErrorsForDefect\_B1
  - common\_register\_space\_ext\_rom, [17](#)
- PLC\_NumCrcErrorsForDefect\_B2
  - common\_register\_space\_ext\_rom, [18](#)
- PLC\_PM\_CHASSIS\_ADDR
  - external\_ram.h, [94](#)
- PLC\_PM\_MODULE\_ADDR
  - external\_ram.h, [94](#)
- PLC\_PMAAddr
  - common\_register\_space\_ext\_ram, [14](#)
- PLC\_PowerDefect
  - common\_register\_space\_ext\_ram, [15](#)
- PLC\_SelfDiagDefect
  - common\_register\_space\_ext\_ram, [15](#)
- PLC\_SerialNumber
  - common\_register\_space\_ext\_rom, [18](#)
- plc\_sof\_ver\_struct, [33](#)
  - add\_info, [34](#)
  - develop, [34](#)
  - modification, [34](#)
  - revision, [34](#)
  - soft\_ver, [35](#)
  - type, [35](#)
- plc\_soft\_ver
  - external\_ram.h, [100](#)
- PLC\_SOFT\_VER\_ADD\_INFO
  - external\_ram.h, [95](#)
- PLC\_SOFT\_VER\_DEVELOP
  - external\_ram.h, [95](#)

- PLC\_SOFT\_VER\_MODIFICATION
  - external\_ram.h, [95](#)
- PLC\_SOFT\_VER\_REVISION
  - external\_ram.h, [95](#)
- PLC\_SOFT\_VER\_SOFT\_VER
  - external\_ram.h, [95](#)
- PLC\_SOFT\_VER\_TYPE
  - external\_ram.h, [95](#)
- PLC\_SoftVer
  - common\_register\_space\_ext\_ram, [15](#)
- PLC\_TimeoutForDefect\_B1
  - common\_register\_space\_ext\_rom, [18](#)
- PLC\_TimeoutForDefect\_B2
  - common\_register\_space\_ext\_rom, [18](#)
- PLC\_TimeSoloWork
  - common\_register\_space\_ext\_rom, [18](#)
- PLC\_TimeToRepair
  - common\_register\_space\_ext\_rom, [18](#)
- PM\_ADDR\_ERROR
  - rs422\_protocol.h, [168](#)
- PM\_CHASSIS\_ADDR
  - mdr32\_drivers.h, [150](#)
- PM\_DEV\_ADDR
  - mdr32\_drivers.h, [150](#)
- polyn\_ch\_consts
  - external\_ram.c, [89](#)
  - external\_rom.c, [107](#)
- PORT\_ADC\_MODE
  - 1273pv19t.h, [71](#)
- PORT\_ADC\_NSS
  - 1273pv19t.h, [71](#)
- PORT\_ADC\_RST
  - 1273pv19t.h, [71](#)
- PORT\_ADC\_SDIFS\_IRQ
  - 1273pv19t.h, [72](#)
- PORT\_LEDS
  - leds.h, [125](#)
- PORT\_SSP1
  - spi.h, [186](#)
- PORT\_UART1
  - uart.h, [213](#)
- PORT\_UART1\_EN
  - uart.h, [213](#)
- PORT\_UART2
  - uart.h, [213](#)
- PORT\_UART2\_EN
  - uart.h, [213](#)
- POSITION\_ERROR
  - uart.h, [215](#)
- power\_fail
  - self\_diag\_struct, [45](#)
- power\_failure
  - external\_ram.h, [100](#)
- power\_failure\_struct, [35](#)
  - reserv, [36](#)
  - v1\_3\_3, [36](#)
  - v1\_5, [36](#)
  - v2\_3\_3, [36](#)
- v2\_5, [36](#)
- prev
  - list\_head\_struct, [25](#)
- protocol\_do\_cmds
  - rs422\_protocol.c, [152](#)
  - rs422\_protocol.h, [169](#)
- protocol\_error
  - rs422\_protocol.h, [168](#)
- protocol\_errors
  - rs422\_protocol.h, [168](#)
- ram\_data
  - external\_ram.h, [100](#)
- ram\_data\_struct, [37](#)
  - common\_ram\_register\_space, [38](#)
  - crc\_table, [38](#)
  - mpa\_ram\_register\_space, [38](#)
  - packet\_rx, [38](#)
  - packet\_tx, [38](#)
  - Reserv, [39](#)
  - rx\_packet\_struct, [39](#)
  - service\_byte\_pm, [39](#)
  - service\_byte\_um, [39](#)
  - spi\_1\_rx\_buffer, [39](#)
  - start\_struct, [39](#)
  - tx\_data, [40](#)
  - tx\_packet\_struct, [40](#)
  - uart1\_rx\_buffer, [40](#)
  - uart2\_rx\_buffer, [40](#)
- RAM\_REGISTER\_SPACE\_START\_ADDR
  - external\_ram.h, [96](#)
- ram\_space\_pointer
  - external\_ram.c, [90](#)
  - main.c, [142](#)
  - rs422\_protocol.c, [161](#)
  - timers.c, [195](#)
- ram\_start\_struct
  - external\_ram.h, [101](#)
- range
  - external\_ram.h, [101](#)
- range\_start\_struct, [40](#)
  - address, [41](#)
  - range\_type, [41](#)
  - size, [41](#)
  - start\_channel\_num, [41](#)
- range\_type
  - range\_start\_struct, [41](#)
- ranges\_in\_start\_struct
  - start\_struct\_ext\_ram, [53](#)
- read\_byte\_rom
  - external\_rom.c, [106](#)
  - external\_rom.h, [116](#)
- READ\_CMD
  - rs422\_protocol.h, [166](#)
- read\_pos
  - uart\_config\_data, [60](#)
- READ\_TIMEOUT\_ERROR
  - uart.h, [215](#)
- read\_timeout\_flag

- uart\_timeouts, 64
- read\_val\_timeout
  - uart\_timeouts, 64
- ready\_to\_control
  - service\_byte\_struct\_um, 48
- receive\_adc\_chanel\_pack
  - main.c, 140
  - main.h, 146
- receive\_packet
  - rs422\_protocol.c, 156
  - rs422\_protocol.h, 173
- receiver\_addr
  - packet\_header\_struct, 32
- RECOGNIZE\_BUS
  - uart.h, 214
- request\_data
  - main.c, 140
  - main.h, 147
- Reserv
  - ram\_data\_struct, 39
- reserv
  - bus\_defect\_struct, 8
  - config\_struct, 20
  - device\_address\_struct, 21
  - device\_type\_struct, 22
  - power\_failure\_struct, 36
  - self\_diag\_struct, 45
  - service\_byte\_struct\_um, 48
- reserv1
  - ai\_oper\_mode\_struct, 7
  - self\_diag\_struct, 45
- Reserv\_1
  - common\_register\_space\_ext\_ram, 15
- reserv\_1
  - service\_byte\_struct\_pm, 47
- Reserv\_2
  - common\_register\_space\_ext\_rom, 19
- reserv\_2
  - service\_byte\_struct\_pm, 47
- Reserv\_3
  - mpa\_register\_space\_ext\_rom, 30
- Reserv\_4
  - mpa\_register\_space\_ext\_rom, 30
- Reserv\_5
  - mpa\_register\_space\_ext\_rom, 31
- Reserv\_6
  - mpa\_register\_space\_ext\_rom, 27
- RESET\_BIT
  - external\_ram.h, 96
- RESET\_CMD
  - rs422\_protocol.h, 166
- RESET\_LED\_ERROR\_WORK
  - leds.h, 125
- RESET\_LED\_OK\_WORK
  - leds.h, 126
- result
  - cmd\_header\_struct, 9
- REVISION
  - external\_rom.h, 112
- revision
  - device\_type\_struct, 22
  - plc\_sof\_ver\_struct, 34
- rom\_data
  - external\_rom.h, 114
- rom\_data\_struct, 42
  - common\_rom\_registers\_space, 43
  - mpa\_rom\_registers\_space, 43
- ROM\_REGISTER\_SPACE\_START\_ADDR
  - external\_rom.h, 113
- rom\_space\_pointer
  - external\_ram.c, 90
  - main.c, 142
  - rs422\_protocol.c, 162
- rs422\_protocol.c, 151
  - crc32, 152
  - fill\_crc32\_table, 152
  - protocol\_do\_cmds, 152
  - ram\_space\_pointer, 161
  - receive\_packet, 156
  - rom\_space\_pointer, 162
  - rx\_error\_handler, 158
  - transmit\_packet, 159
  - um\_service\_byte\_handler, 160
- rs422\_protocol.h, 162
  - CONFIG, 164
  - crc32, 169
  - CRC\_ERROR, 168
  - fields\_cmd, 167
  - fields\_cmd\_header, 167
  - fields\_packet, 167
  - fields\_packet\_header, 167
  - fields\_packet\_tail, 168
  - fill\_crc32\_table, 169
  - INIT\_CMD, 165
  - NO\_ERROR, 168
  - NUMBER\_CMDS\_IN\_PACKET, 165
  - PACKET\_ERROR, 168
  - PACKET\_HEAD, 165
  - PACKET\_TAIL, 165
  - PLC\_CM\_CRITICAL\_FAULT, 165
  - PLC\_CM\_INIT\_1\_BUS, 165
  - PLC\_CM\_INIT\_2\_BUS, 166
  - PLC\_CM\_NOT\_INIT, 166
  - PLC\_CM\_REMOVE\_INIT, 166
  - PLC\_CM\_UNKNOWN\_STATE, 166
  - PM\_ADDR\_ERROR, 168
  - protocol\_do\_cmds, 169
  - protocol\_error, 168
  - protocol\_errors, 168
  - READ\_CMD, 166
  - receive\_packet, 173
  - RESET\_CMD, 166
  - rx\_error\_handler, 175
  - transmit\_packet, 176
  - TYPE\_CMD, 167
  - UART\_ERROR, 168

- um\_service\_byte\_handler, 177
- WRITE\_CMD, 167
- RST\_CLK\_PCLK\_SPIn
  - spi\_config\_data, 50
- RST\_CLK\_PCLK\_UARTn
  - uart\_config\_data, 60
- rx\_error\_handler
  - rs422\_protocol.c, 158
  - rs422\_protocol.h, 175
- rx\_packet\_struct
  - ram\_data\_struct, 39
- self\_diag
  - external\_ram.h, 101
- self\_diag\_struct, 43
  - fail\_chanel, 44
  - fail\_crc\_firmware, 44
  - fail\_download\_rom, 44
  - fail\_firmware\_1\_bus, 44
  - fail\_firmware\_2\_bus, 44
  - fail\_firmware\_ram, 44
  - fail\_soft\_ver, 45
  - power\_fail, 45
  - reserv, 45
  - reserv1, 45
- self\_diagnostics\_error
  - service\_byte\_struct\_pm, 47
- sender\_addr
  - packet\_header\_struct, 32
- SERIAL\_NUMBER
  - external\_rom.h, 113
- service\_byte
  - packet\_header\_struct, 32
- service\_byte\_pm
  - ram\_data\_struct, 39
- service\_byte\_struct\_pm, 45
  - both\_control, 46
  - fail\_bus\_1, 46
  - fail\_bus\_2, 46
  - init, 46
  - master, 47
  - reserv\_1, 47
  - reserv\_2, 47
  - self\_diagnostics\_error, 47
- service\_byte\_struct\_um, 47
  - last\_answer, 48
  - ready\_to\_control, 48
  - reserv, 48
- service\_byte\_um
  - ram\_data\_struct, 39
- service\_struct\_pm
  - external\_ram.h, 101
- service\_struct\_um
  - external\_ram.h, 101
- SET\_BIT
  - external\_ram.h, 96
- SET\_LED\_ERROR\_WORK
  - leds.h, 126
- SET\_LED\_OK\_WORK
  - leds.h, 126
- size
  - range\_start\_struct, 41
- SIZE\_ERROR
  - uart.h, 215
- soft\_ver
  - plc\_soft\_ver\_struct, 35
- SPI
  - spi\_config\_data, 50
- spi.c, 179
  - dma\_spi\_rx\_init, 180
  - spi\_1, 183
  - spi\_2, 183
  - spi\_clean\_fifo\_rx\_buf, 180
  - spi\_gpio\_config, 181
  - spi\_init, 181
  - spi\_receive\_halfword, 182
  - spi\_transmit\_halfword, 182
  - spi\_transmit\_message, 182
- spi.h, 184
  - dma\_spi\_rx\_init, 187
  - FIFO\_SIZE, 185
  - PIN\_SSP1\_RX, 185
  - PIN\_SSP1\_SCK, 185
  - PIN\_SSP1\_SS, 186
  - PIN\_SSP1\_TX, 186
  - PORT\_SSP1, 186
  - SPI\_BUFFER\_SIZE, 186
  - spi\_clean\_fifo\_rx\_buf, 187
  - spi\_init, 188
  - spi\_n, 186
  - spi\_n\_dma\_ch\_params, 186
  - spi\_receive\_halfword, 188
  - spi\_transmit\_halfword, 189
  - spi\_transmit\_message, 189
- spi\_1
  - 1273pv19t.c, 68
  - dma.c, 79
  - main.c, 143
  - spi.c, 183
  - timers.c, 195
- spi\_1\_rx\_buffer
  - ram\_data\_struct, 39
- spi\_2
  - dma.c, 79
  - main.c, 143
  - spi.c, 183
- SPI\_BUFFER\_SIZE
  - spi.h, 186
- spi\_clean\_fifo\_rx\_buf
  - spi.c, 180
  - spi.h, 187
- spi\_config\_data, 49
  - buffer, 49
  - buffer\_counter, 50
  - IRQn, 50
  - RST\_CLK\_PCLK\_SPIn, 50
  - SPI, 50

- spi\_dma\_ch, 50
- SSPx, 50
- spi\_dma\_ch
  - spi\_config\_data, 50
- spi\_dma\_params, 51
  - dma\_channel, 51
  - DMA\_Channel\_SPI\_RX, 51
  - DMA\_InitStructure\_SPI\_RX, 52
  - dma\_irq\_counter, 52
- spi\_gpio\_config
  - spi.c, 181
- spi\_init
  - spi.c, 181
  - spi.h, 188
- spi\_n
  - spi.h, 186
- spi\_n\_dma\_ch\_params
  - spi.h, 186
- spi\_receive\_halfword
  - spi.c, 182
  - spi.h, 188
- spi\_struct
  - adc\_config\_data, 6
- spi\_transmit\_halfword
  - spi.c, 182
  - spi.h, 189
- spi\_transmit\_message
  - spi.c, 182
  - spi.h, 189
- SSPx
  - spi\_config\_data, 50
- start\_channel\_num
  - range\_start\_struct, 41
- start\_struct
  - ram\_data\_struct, 39
- START\_STRUCT\_CHANGE\_FLAG
  - external\_ram.h, 96
- start\_struct\_ext\_ram, 52
  - flag\_change\_struct, 53
  - length, 53
  - number\_of\_ranges, 53
  - ranges\_in\_start\_struct, 53
  - text\_info, 54
- START\_STRUCT\_LENGTH
  - external\_ram.h, 96
- START\_STRUCT\_NUMBER\_OF\_RANGES
  - external\_ram.h, 96
- START\_STRUCT\_RANGE0\_ADDR
  - external\_ram.h, 97
- START\_STRUCT\_RANGE0\_SIZE
  - external\_ram.h, 97
- START\_STRUCT\_RANGE0\_START\_CH\_NUM
  - external\_ram.h, 97
- START\_STRUCT\_RANGE0\_TYPE
  - external\_ram.h, 97
- START\_STRUCT\_RANGE1\_ADDR
  - external\_ram.h, 97
- START\_STRUCT\_RANGE1\_SIZE
  - external\_ram.h, 97
- START\_STRUCT\_RANGE1\_START\_CH\_NUM
  - external\_ram.h, 98
- START\_STRUCT\_RANGE1\_TYPE
  - external\_ram.h, 98
- START\_STRUCT\_RANGE2\_ADDR
  - external\_ram.h, 98
- START\_STRUCT\_RANGE2\_SIZE
  - external\_ram.h, 98
- START\_STRUCT\_RANGE2\_START\_CH\_NUM
  - external\_ram.h, 98
- START\_STRUCT\_RANGE2\_TYPE
  - external\_ram.h, 98
- START\_STRUCT\_TEXT\_INFO\_ADDR
  - external\_ram.h, 99
- sTIM\_ChnInit
  - timer\_config\_struct, 54
- sync\_adc\_channels
  - main.c, 141
  - main.h, 148
- TEST\_BIT
  - external\_ram.h, 99
- text\_info
  - start\_struct\_ext\_ram, 54
- TIME\_SOLO\_WORK
  - external\_rom.h, 113
- TIME\_TO\_REPAIR
  - external\_rom.h, 113
- TIMEOUT\_FOR\_DEFECT\_B1
  - external\_rom.h, 113
- TIMEOUT\_FOR\_DEFECT\_B2
  - external\_rom.h, 113
- timer1\_init
  - timers.c, 192
- timer2\_init
  - timers.c, 193
- TIMER2\_IRQHandler
  - timers.c, 193
- timer3\_init
  - timers.c, 194
- timer\_1
  - dma.c, 79
  - main.c, 143
  - timers.c, 195
- timer\_2
  - 1273pv19t.c, 68
  - dma.c, 79
  - main.c, 143
  - timers.c, 195
- timer\_3
  - dma.c, 79
  - main.c, 143
  - timers.c, 195
- timer\_cnt
  - timer\_config\_struct, 55
- timer\_config\_struct, 54
  - sTIM\_ChnInit, 54
  - timer\_cnt, 55

- TIMER\_STATUS, 55
- TIMERInitStruct, 55
- TIMERx, 55
- timer\_init
  - timers.c, 194
  - timers.h, 199
- timer\_irq\_list
  - timers.h, 197
- timer\_irq\_list\_struct, 56
  - data, 56
  - event, 56
  - handler, 57
  - list, 57
- timer\_n
  - timers.h, 197
- timer\_n\_capture
  - adc\_config\_data, 6
- timer\_n\_timeout
  - uart\_timeouts, 64
- TIMER\_NUM
  - external\_ram.h, 99
- TIMER\_STATUS
  - timer\_config\_struct, 55
- TIMERInitStruct
  - timer\_config\_struct, 55
- timers.c, 190
  - adc\_1, 195
  - delay\_micro, 191
  - delay\_milli, 191
  - list\_tmr\_handler\_add\_tail, 192
  - list\_tmr\_handler\_init, 192
  - ram\_space\_pointer, 195
  - spi\_1, 195
  - timer1\_init, 192
  - timer2\_init, 193
  - TIMER2\_IRQHandler, 193
  - timer3\_init, 194
  - timer\_1, 195
  - timer\_2, 195
  - timer\_3, 195
  - timer\_init, 194
  - tmr\_handler\_head, 196
- timers.h, 196
  - delay\_micro, 198
  - delay\_milli, 198
  - list\_tmr\_handler\_add\_tail, 198
  - list\_tmr\_handler\_init, 199
  - timer\_init, 199
  - timer\_irq\_list, 197
  - timer\_n, 197
- TIMERx
  - timer\_config\_struct, 55
- tmr\_handler\_head
  - main.c, 143
  - timers.c, 196
- transmit\_packet
  - rs422\_protocol.c, 159
  - rs422\_protocol.h, 176
- tx\_data
  - ram\_data\_struct, 40
- tx\_packet\_struct
  - ram\_data\_struct, 40
- tx\_rx\_packet\_struct, 57
  - cmd\_with\_data, 58
  - packet\_header, 58
  - packet\_tail, 58
- type
  - device\_type\_struct, 23
  - plc\_sof\_ver\_struct, 35
- TYPE\_CMD
  - rs422\_protocol.h, 167
- type\_of\_module
  - external\_ram.h, 101
- UART
  - uart\_config\_data, 60
- uart.c, 200
  - DMA\_UART\_RX\_init, 201
  - UART1\_IRQHandler, 201
  - UART2\_IRQHandler, 202
  - uart\_1, 209
  - uart\_2, 209
  - uart\_clean, 202
  - uart\_get\_buf\_counter, 202
  - uart\_gpio\_config, 203
  - uart\_init, 204
  - uart\_read, 205
  - uart\_read\_pos, 206
  - uart\_set\_pos, 206
  - uart\_set\_read\_timeout, 207
  - uart\_set\_write\_timeout, 207
  - uart\_write, 208
- uart.h, 209
  - BUFFER\_MASK, 212
  - DMA\_UART\_RX\_init, 215
  - errors, 215
  - GET\_UART\_BUF\_PTR, 212
  - INIT\_ERROR, 215
  - NO\_ERRORS, 215
  - PIN\_UART1\_EN, 212
  - PIN\_UART1\_RX, 212
  - PIN\_UART1\_TX, 212
  - PIN\_UART2\_EN, 212
  - PIN\_UART2\_RX, 213
  - PIN\_UART2\_TX, 213
  - PORT\_UART1, 213
  - PORT\_UART1\_EN, 213
  - PORT\_UART2, 213
  - PORT\_UART2\_EN, 213
  - POSITION\_ERROR, 215
  - READ\_TIMEOUT\_ERROR, 215
  - RECOGNIZE\_BUS, 214
  - SIZE\_ERROR, 215
  - UART\_BUFFER\_SIZE, 214
  - uart\_clean, 216
  - uart\_dma\_ch\_params, 214
  - uart\_errors, 214

- uart\_get\_buf\_counter, 216
- uart\_init, 217
- uart\_n, 214
- uart\_read, 218
- uart\_read\_pos, 219
- uart\_rx\_tx\_timeouts, 215
- uart\_set\_pos, 220
- uart\_set\_read\_timeout, 220
- uart\_set\_write\_timeout, 221
- uart\_write, 221
- WRITE\_TIMEOUT\_ERROR, 215
- UART1\_IRQHandler
  - uart.c, 201
- uart1\_rx\_buffer
  - ram\_data\_struct, 40
- UART2\_IRQHandler
  - uart.c, 202
- uart2\_rx\_buffer
  - ram\_data\_struct, 40
- uart\_1
  - dma.c, 79
  - main.c, 144
  - uart.c, 209
- uart\_2
  - dma.c, 80
  - main.c, 144
  - uart.c, 209
- UART\_BUFFER\_SIZE
  - uart.h, 214
- uart\_clean
  - uart.c, 202
  - uart.h, 216
- uart\_config\_data, 59
  - buffer, 60
  - buffer\_count, 60
  - IRQn, 60
  - read\_pos, 60
  - RST\_CLK\_PCLK\_UARTn, 60
  - UART, 60
  - uart\_dma\_ch, 61
  - UART\_HCLKdiv, 61
  - uart\_timeouts, 61
  - UARTx, 61
- uart\_dma\_ch
  - uart\_config\_data, 61
- uart\_dma\_ch\_params
  - uart.h, 214
- uart\_dma\_params, 61
  - dma\_channel, 62
  - DMA\_Channel\_UART\_RX, 62
  - DMA\_InitStructure\_UART\_RX, 62
  - dma\_irq\_counter, 62
- UART\_ERROR
  - rs422\_protocol.h, 168
- uart\_errors
  - uart.h, 214
- uart\_get\_buf\_counter
  - uart.c, 202
- uart.h, 216
- uart\_gpio\_config
  - uart.c, 203
- UART\_HCLKdiv
  - uart\_config\_data, 61
- uart\_init
  - uart.c, 204
  - uart.h, 217
- uart\_n
  - uart.h, 214
- uart\_read
  - uart.c, 205
  - uart.h, 218
- uart\_read\_pos
  - uart.c, 206
  - uart.h, 219
- uart\_rx\_tx\_timeouts
  - uart.h, 215
- uart\_set\_pos
  - uart.c, 206
  - uart.h, 220
- uart\_set\_read\_timeout
  - uart.c, 207
  - uart.h, 220
- uart\_set\_write\_timeout
  - uart.c, 207
  - uart.h, 221
- uart\_timeouts, 63
  - read\_timeout\_flag, 64
  - read\_val\_timeout, 64
  - timer\_n\_timeout, 64
  - uart\_config\_data, 61
  - write\_timeout\_flag, 64
  - write\_val\_timeout, 64
- uart\_write
  - uart.c, 208
  - uart.h, 221
- UARTx
  - uart\_config\_data, 61
- um\_service\_byte\_handler
  - rs422\_protocol.c, 160
  - rs422\_protocol.h, 177
- use\_object
  - device\_type\_struct, 23
- v1\_3\_3
  - power\_failure\_struct, 36
- v1\_5
  - power\_failure\_struct, 36
- v2\_3\_3
  - power\_failure\_struct, 36
- v2\_5
  - power\_failure\_struct, 36
- WORK\_FREQ
  - mdr32\_drivers.h, 150
- write\_byte\_rom
  - external\_rom.c, 107
  - external\_rom.h, 117



WRITE\_CMD  
    rs422\_protocol.h, [167](#)  
WRITE\_TIMEOUT\_ERROR  
    uart.h, [215](#)  
write\_timeout\_flag  
    uart\_timeouts, [64](#)  
write\_val\_timeout  
    uart\_timeouts, [64](#)