ПО для макета МПА (100 тема)

Создано системой Doxygen 1.8.20

Глава 1

Алфавитный указатель классов

1.1 Классы

Классы с их кратким описанием.

Глава 2

Список файлов

2.1 Файлы

Полный список файлов.

Глава 3

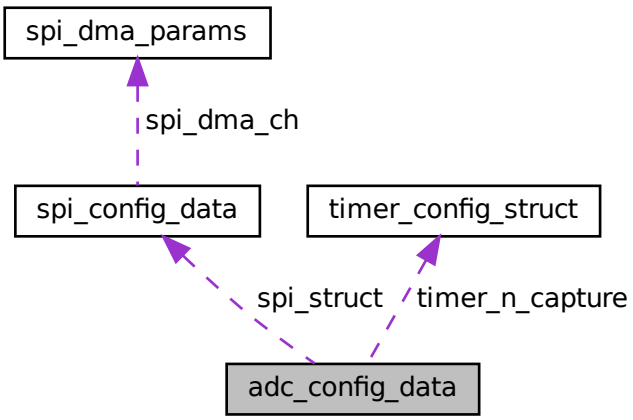
Классы

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 CHANEL_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 CHANEL_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:

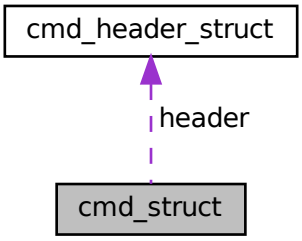
- rs422_protocol.h

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:

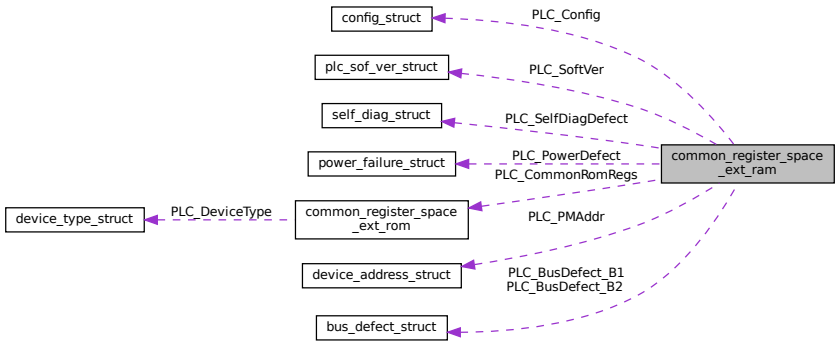
- rs422_protocol.h

3.6 Структура common_register_space_ext_ram

Органиация 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_PMAddr
  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 храняться в ПЗУ

3.6.1 Подробное описание

Органиация 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 храняться 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_PMAddr

device_address common_register_space_ext_ram::PLC_PMAddr

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:

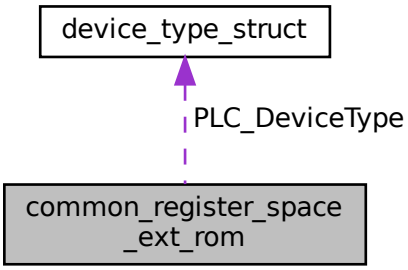
- external_ram.h

3.7 Структура common_register_space_ext_rom

Органиация 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 неиспр шины 1, мс
- uint32_t PLC_TimeoutForDefect_B2
  Время без связи до неиспр 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, х10мс
- 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

Органиация 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 неиспр шины 1, мс

3.7.2.10 PLC_TimeoutForDefect_B2

uint32_t common_register_space_ext_rom::PLC_TimeoutForDefect_B2

Время без связи до неиспр 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, х10мс

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:

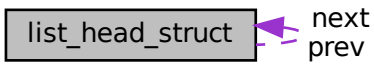
- internal_ram.h

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:

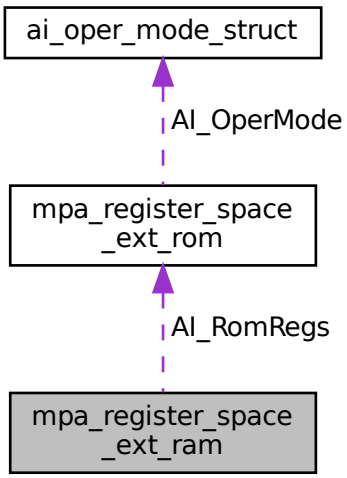
- list.h

3.13 Структура mpa_register_space_ext_ram

Органиация 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 храняться в ПЗУ
- uint16_t AI_SignalChanged
  Изменялся ли сигнал с последнего опроса
- int16_t AI_CodeADC [MAX_CHANEL_NUMBER]
  Сырые данные, код АЦП
- float AI_PhysQuantFloat [MAX_CHANEL_NUMBER]
  Физическая величина (число с плавающей точкой)
- uint8_t AI_DiagnosticChannel [MAX_CHANEL_NUMBER]
  Самодиагностика каналов
- uint8_t Reserv_6 [2]
  Резерв

3.13.1 Подробное описание

Органиация 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_CHANEL_NUMBER]

Сырые данные, код АЦП

3.13.2.2 AI_DiagnosticChannel

uint8_t mpa_register_space_ext_ram::AI_DiagnosticChannel[MAX_CHANEL_NUMBER]

Самодиагностика каналов

3.13.2.3 AI_PhysQuantFloat

float mpa_register_space_ext_ram::AI_PhysQuantFloat[MAX_CHANEL_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 храняться 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:

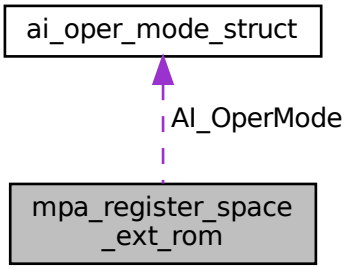
- external_ram.h

3.14 Структура mpa_register_space_ext_rom

Органиация 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_CHANEL_NUMBER]
  Кол-во выборок для усреднения
- int16_t AI_MinCodeADC [MAX_CHANEL_NUMBER]
  Константа B - минимум кода АЦП
- int16_t AI_MaxCodeADC [MAX_CHANEL_NUMBER]
  Константа A - максимум кода АЦП
- uint8_t Reserv_3 [32]
  Резерв
- float AI_PolynConst0 [MAX_CHANEL_NUMBER]
  Константа A0 апрокс. полинома
- float AI_PolynConst1 [MAX_CHANEL_NUMBER]
  Константа A1 апрокс. полинома
- float AI_PolynConst2 [MAX_CHANEL_NUMBER]
  Константа A2 апрокс. полинома
- float AI_PolynConst3 [MAX_CHANEL_NUMBER]
  Константа A3 апрокс. полинома
- float AI_PolynConst4 [MAX_CHANEL_NUMBER]
  Константа A4 апрокс. полинома
- float AI_PolynConst5 [MAX_CHANEL_NUMBER]
  Константа A5 апрокс. полинома
- float AI_PolynConst6 [MAX_CHANEL_NUMBER]
  Константа A6 апрокс. 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

Органиация 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_CHANEL_NUMBER]

Константа A - 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 B - 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 апрокс. полинома

3.14.2.7 AI_PolynConst1

float mpa_register_space_ext_rom::AI_PolynConst1[MAX_CHANEL_NUMBER]

Константа A1 апрокс. полинома

3.14.2.8 AI_PolynConst2

float mpa_register_space_ext_rom::AI_PolynConst2[MAX_CHANEL_NUMBER]

Константа A2 апрокс. полинома

3.14.2.9 AI_PolynConst3

float mpa_register_space_ext_rom::AI_PolynConst3[MAX_CHANEL_NUMBER]

Константа A3 апрокс. полинома

3.14.2.10 AI_PolynConst4

float mpa_register_space_ext_rom::AI_PolynConst4[MAX_CHANEL_NUMBER]

Константа A4 апрокс. полинома

3.14.2.11 AI_PolynConst5

float mpa_register_space_ext_rom::AI_PolynConst5[MAX_CHANEL_NUMBER]

Константа A5 апрокс. полинома

3.14.2.12 AI_PolynConst6

float mpa_register_space_ext_rom::AI_PolynConst6[MAX_CHANEL_NUMBER]

Константа A6 апрокс. 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:

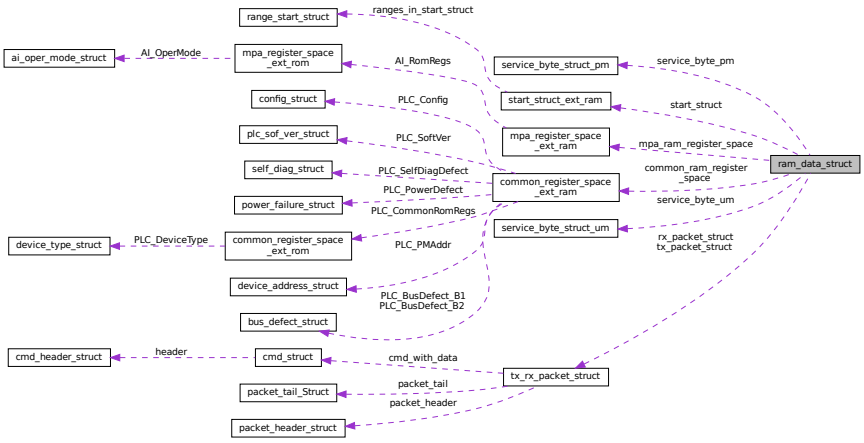
- external_ram.h

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:

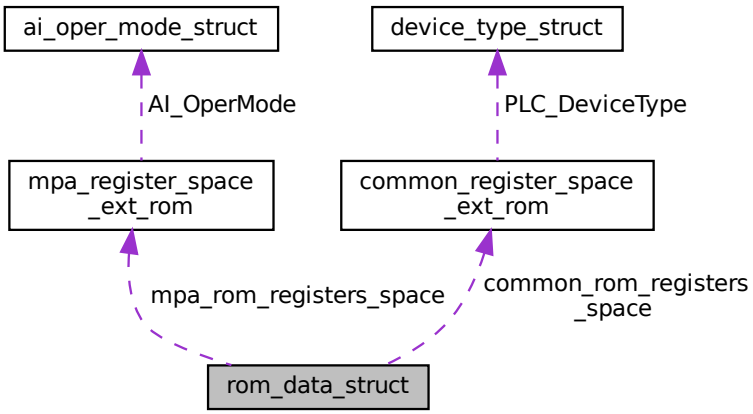
- external_ram.h

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_chanels [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_chanels

uint8_t self_diag_struct::fail_chanels[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:

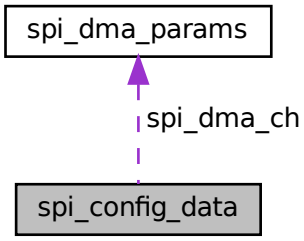
- external_ram.h

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_SPIn
  Включение тактирования для SPIn.
- 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:

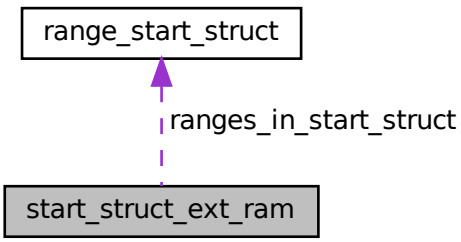
- spi.h

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 текстувую 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 текстувую 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

Настрока 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 TIMERx

MDR_TIMER_TypeDef* timer_config_struct::TIMERx

Библиотечная структура с периферийными регистрами блока TIMER.

Объявления и описания членов структуры находятся в файле:

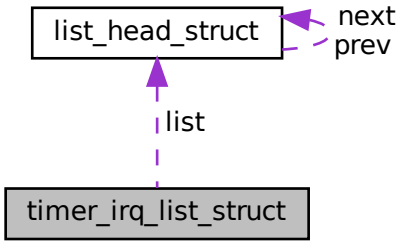
- timers.h

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:

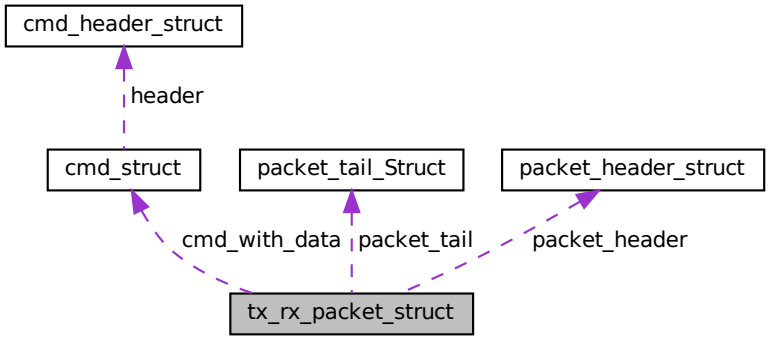
- timers.h

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:

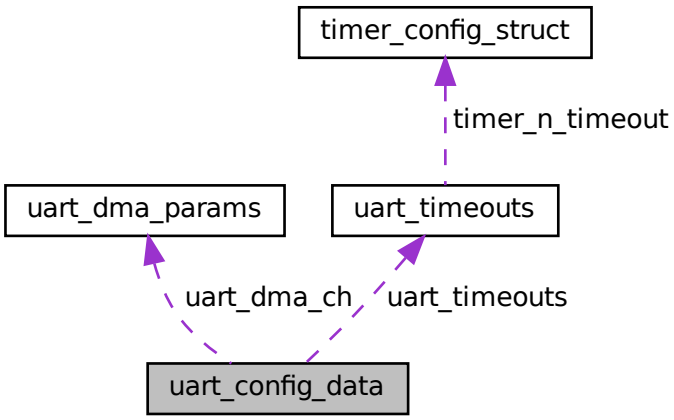
- rs422_protocol.h

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 преимника 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 преимника 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:

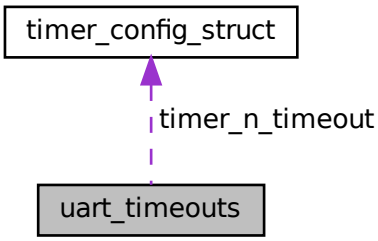
- uart.h

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

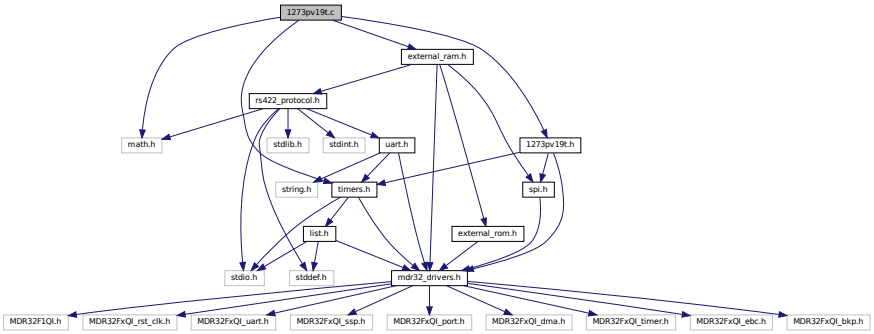
Глава 4

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_PORT
       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(PORT_ADC_RST, &GPIO_init_structADC);
50     // Установка RESET в лог единицу
51     PORT_SetBits(PORT_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(PORT_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(PORT_ADC_MODE, &GPIO_init_structADC);
63
64     // Выбор однополярного режима на мультиплексоре A0=1;A1=0
65     PORT_SetBits(PORT_ADC_MODE,PIN_ADC_MODE_A0);
66     PORT_ResetBits(PORT_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(PORT_ADC_NSS, &GPIO_init_structADC);
73
74     // Установка NSS в лог ноль
75     PORT_ResetBits(PORT_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
        таймера)
78    GPIO_init_structADC.PORT_FUNC = PORT_FUNC_MAIN;
79    GPIO_init_structADC.PORT_OE = PORT_OE_IN;
80    GPIO_init_structADC.PORT_SPEED = PORT_SPEED_MAXFAST;
81    GPIO_init_structADC.PORT_MODE = PORT_MODE_DIGITAL;
82    GPIO_init_structADC.PORT_Pin = PIN_ADC_SDIFS_IRQ;
83    GPIO_init_structADC.PORT_PULL_DOWN = PORT_PULL_DOWN_ON;
84    PORT_Init (PORT_ADC_SDIFS_IRQ, &GPIO_init_structADC);
85 }
```

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 (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     }
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

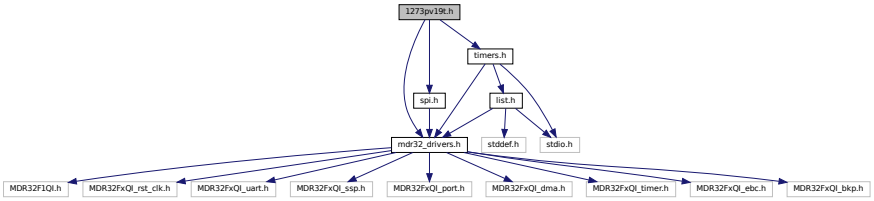
timer_n timer_2 [extern]

Структура с конфигурационными параметрами блока Timer2 МК

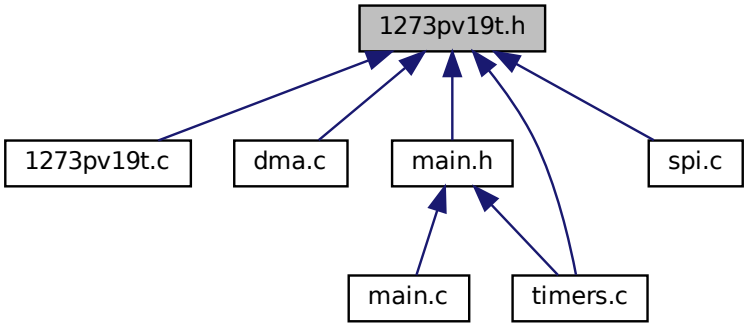
4.2 Файл 1273pv19t.h

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 A1.

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     }
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 }
```

4.2.4.2 adc_reset()

```
void adc_reset (
                void  )
```

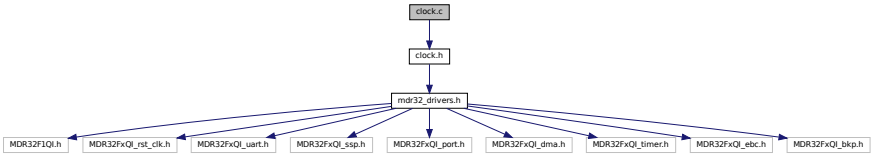
Аппаратный сброс микросхемы АЦП 1273ПВ19Т

```
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  )
```

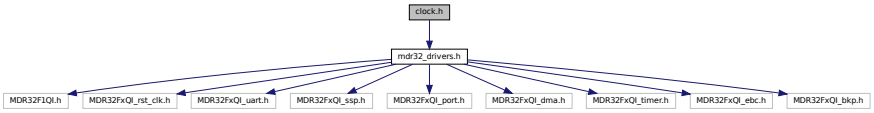
Настраивает тактирование МК

```
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_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 }
```

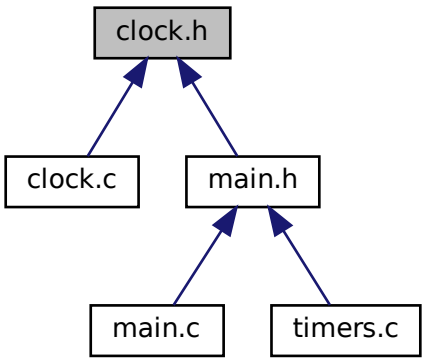
4.4 Файл clock.h

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  )
```

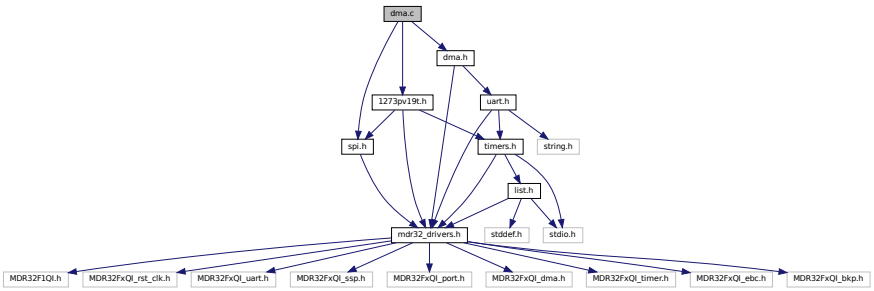
Настраивает тактирование МК

```
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
       |RST_CLK_PCLK_SSP2 |RST_CLK_PCLK_SSP3, ENABLE);
47   #ifdef K1986VE3T
48   RST_CLK_PCLKcmd(RST_CLK_PCLK_SSP4, ENABLE);
49   #endif
50   // Запретить все прерывания, в том числе от SSP
51   NVIC->ICPR[0] = 0xFFFFFFFF;
52   NVIC->ICER[0] = 0xFFFFFFFF;
53
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

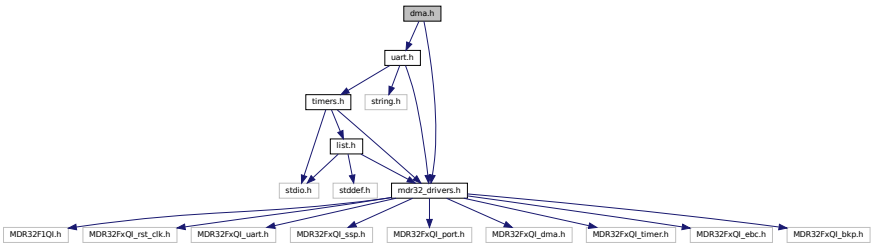
uart_n uart_2 [extern]

Структура с конфигурационными параметрами блока UART2 МК

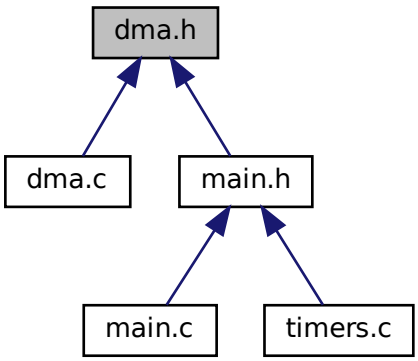
4.6 Файл dma.h

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  )
```

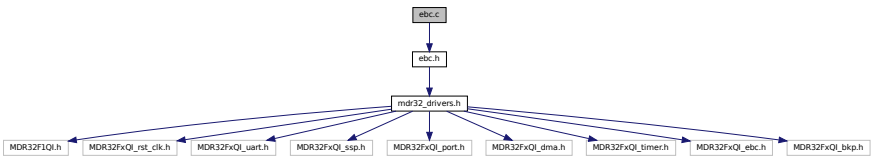
Выполняет общую инициализацию DMA.

```
44 {
45     // Разрешить тактирование DMA
46     RST_CLK_PCLKcmd(RST_CLK_PCLK_RST_CLK | RST_CLK_PCLK_DMA | RST_CLK_PCLK_SSP1
       |RST_CLK_PCLK_SSP2 |RST_CLK_PCLK_SSP3, ENABLE);
47     #ifdef K1986VE3T
48     RST_CLK_PCLKcmd(RST_CLK_PCLK_SSP4, ENABLE);
49     #endif
50     // Запретить все прерывания, в том числе от SSP
51     NVIC->ICPR[0] = 0xFFFFFFFF;
52     NVIC->ICER[0] = 0xFFFFFFFF;
53
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 EBC.

#include "ebc.h"
Граф включаемых заголовочных файлов для ebc.c:



Функции

- void ebc_gpio_config (ebc_devices device)

  Конфигурует выводы МК EBC для внешнего ОЗУ или ПЗУ
- void ebc_init (ebc_devices device)

  Инициализирует EBC для внешнего ОЗУ или ПЗУ

4.7.1 Подробное описание

Файл с реализацией API для работы с EBC.

4.7.2 Функции

4.7.2.1 ebc_gpio_config()

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

Конфигурует выводы МК EBC 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 EBC для внешнего ОЗУ или ПЗУ

Аргументы

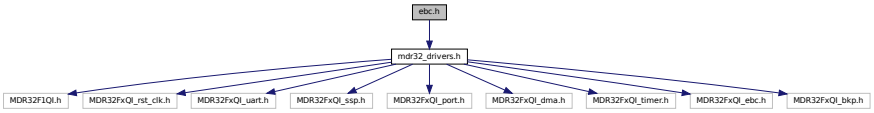
device	0-RAM, 1-ROM

```
131 {
132     ebc_gpio_config(device);
133 
134     EBC_InitTypeDef EBC_InitStruct;
135   EBC_MemRegionInitTypeDef EBC_MemRegionInitStruct;
136 
137   // Включение тактирования EBC
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 }
```

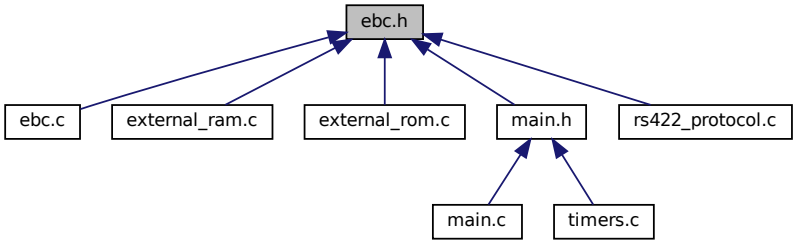
4.8 Файл ebc.h

Заголовочный файл с описанием API для работы с EBC (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 EBC для внешнего ОЗУ или ПЗУ

Аргументы

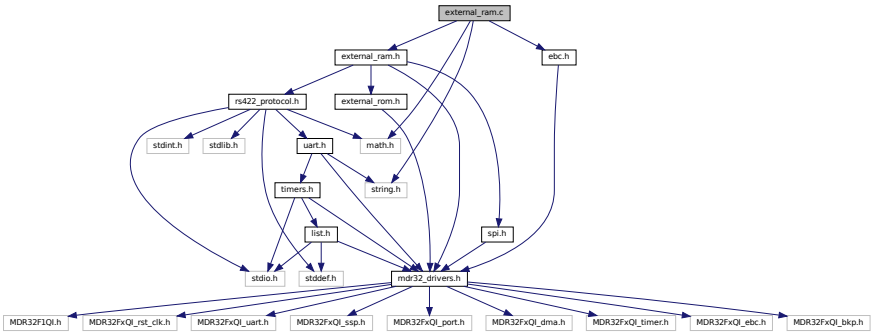
device	0-RAM, 1-ROM

```
131 {
132     ebc_gpio_config(device);
133 
134     EBC_InitTypeDef EBC_InitStruct;
135   EBC_MemRegionInitTypeDef EBC_MemRegionInitStruct;
136 
137   // Включение тактирования EBC
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)

  Инициализацирует 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  )
```

Инициализацирует 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_ram_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_PMAddr.module_addr = PLC_PM_MODULE_ADDR;
75     common_ram_reg_ptr->PLC_PMAddr.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       // Инциализация 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    // Заполянем 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,          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]

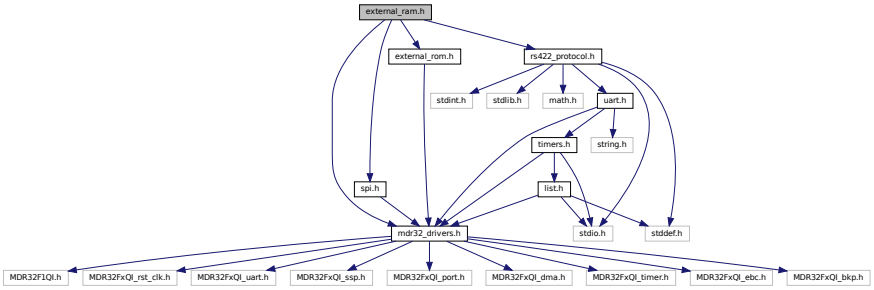
Указатель для обращения к внешнему ПЗУ

4.10 Файл external_ram.h

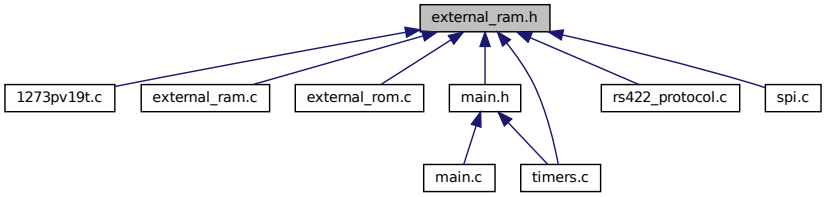
Заголовочный файл с описанием API для работы с областью памяти внешнего ОЗУ

```
#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
  Органиация пространства общих регистров во внешнем ОЗУ
- struct mpa_register_space_ext_ram
  Органиация 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_sof_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

  Органиация пространства общих регистров во внешнем ОЗУ
- typedef struct mpa_register_space_ext_ram mpa_ram_registers

  Органиация 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)

  Инициализацирует 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
```

Органиация 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

Органиация пространства регистров МПА во внешнем ОЗУ

4.10.3.7 plc_soft_ver

typedef struct plc_sof_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  )
```

Инициализацирует область памяти внешнего ОЗУ

Функция инициализации области памяти внешнего ОЗУ

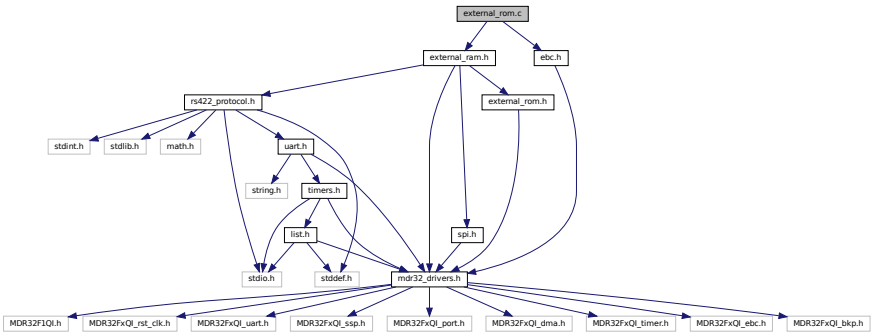
```
36 {
37     ram_space_pointer = (ram_data*)EXT_RAM_START_ADDR;
38     rom_space_pointer = (rom_data*)EXT_ROM_START_ADDR;
39 
40     ram_start_struct* start_struct_ptr = &ram_space_pointer->start_struct;
41     common_ram_registers* common_ram_reg_ptr = &ram_space_pointer->common_ram_register_space;
42     mpa_rom_registers* mpa_rom_reg_ptr = &ram_space_pointer->mpa_ram_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_PMAddr.module_addr = PLC_PM_MODULE_ADDR;
75    common_ram_reg_ptr->PLC_PMAddr.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       // Инциализация 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    // Заполянем таблицу CRC32
138    fill_crc32_table();
139 }
```

4.11 Файл external_rom.c

Файл с реализацией API для работы с областью памяти внешнего ПЗУ

#include "external_ram.h"
#include "ebc.h"
Граф включаемых заголовочных файлов для external_rom.c:



Макросы

- #define FIRST_TIME_INIT

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

Функции

- void init_external_rom_space (void)

  Инициализацирует 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

Переменные

- float polyn_ch_consts [MAX_CHANEL_NUMBER][7]

  Константы полиномов

4.11.1 Подробное описание

Файл с реализацией API для работы с областью памяти внешнего ПЗУ

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  )
```

Инициализацирует 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,
     sizeof(mpa_regs));
72 
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 EBC 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 EBC 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 EBC на вход
92     MDR_PORTA->OE = 0x00000000;
93
94     // Теперь проверяем, ывполнилась 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

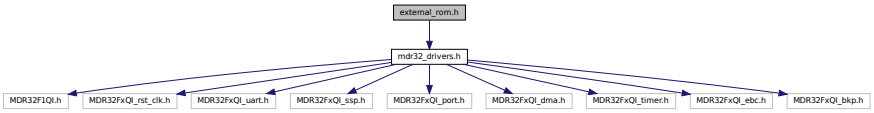
float polyn_ch_consts[MAX_CHANEL_NUMBER][7] [extern]

Константы полиномов

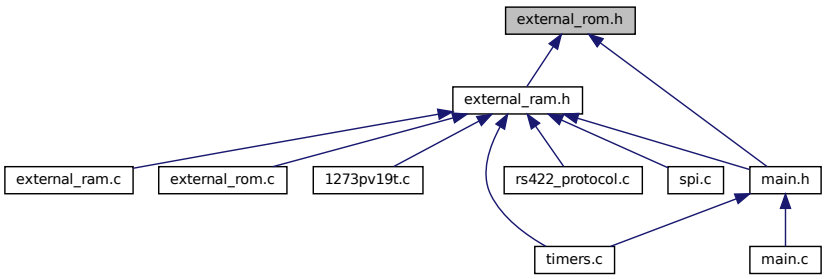
4.12 Файл external_rom.h

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

  Органиация пространства общих регистров для зеркализации во внешнем ПЗУ
- struct mpa_register_space_ext_rom

  Органиация 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 неиспр шины 1, мс (значение для инициализации по умолчанию)
- #define TIMEOUT_FOR_DEFECT_B2 200

  Время без связи до неиспр 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, х10мс (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
  Органиация пространства общих регистров для зеркализации во внешнем ПЗУ
- typedef struct mpa_register_space_ext_rom mpa_rom_registers
  Органиация 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)
  Инициализацирует 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 "MPA"

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, х10мс (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

Органиация 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

Органиация 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  )
```

Инициализацирует 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,
      sizeof(mpa_regs));
72
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 EBC 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

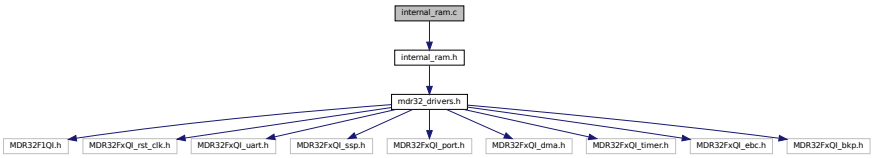
Функция записи байта в ПЗУ

```
80 {
81     uint8_t status;
82 
83     // Конфигурирование выводов EBC 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 EBC на вход
92     MDR_PORTA->OE = 0x00000000;
93 
94     // Теперь проверяем, ывполнилась 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
    + i]));
61             }
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 страницы, то возвращаяем 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-
    >memory_page_status[page_cnt + consecutive_page_cnt]));
38             }
39             return &heap_ptr->memory_page_space[page_cnt][0];
40         }
41     }
42 
43     return 0;
44 }
```

4.13.3 Переменные

4.13.3.1 heap_ptr

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

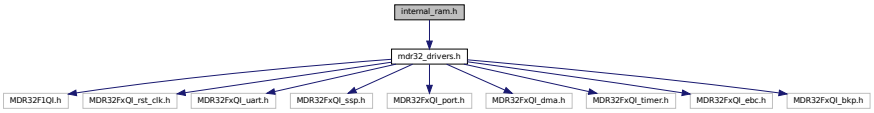
Указатель на "самодельную" кучу

4.14 Файл internal_ram.h

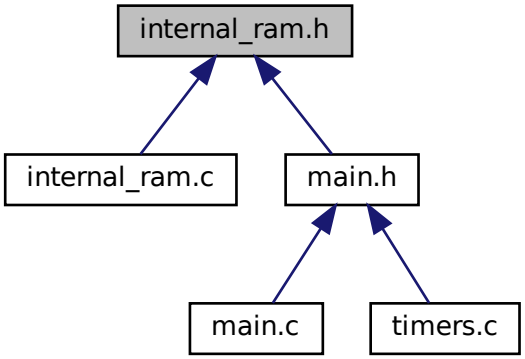
Заголовочный файл с описанием API для работы с областью памяти внутреннего ОЗУ

```
#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
    + i]));
61             }
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

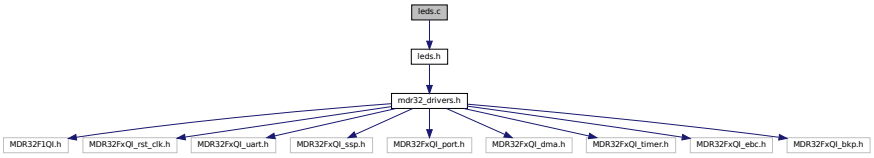
Функция выделения свободной памяти во внешнем ОЗУ (страниц памяти)

```
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-
    >memory_page_status[page_cnt + consecutive_page_cnt]));
38             }
39             return &heap_ptr->memory_page_space[page_cnt][0];
40         }
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  )
```

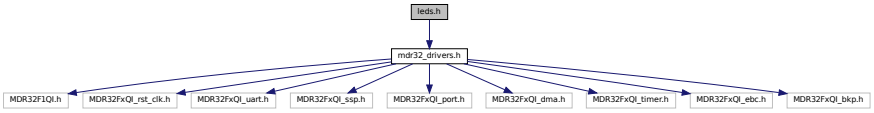
Конфигурирует выводы МК для светодиодных индикаторов

```
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 }
```

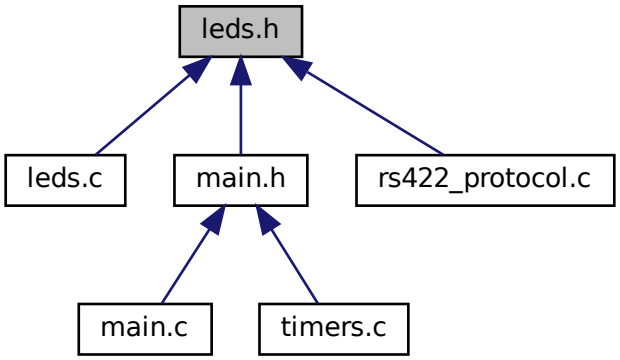
4.16 Файл leds.h

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->CLRTX = 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->CLRTX = 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  )
```

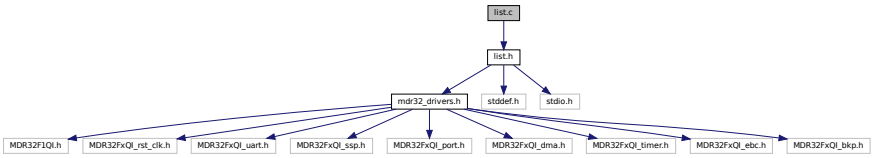
Конфигурирует выводы МК для светодиодных индикаторов

```
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

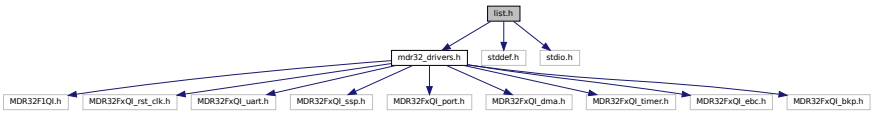
Функция, определяющая, является ли текущий узел списка последним узлом в списке

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

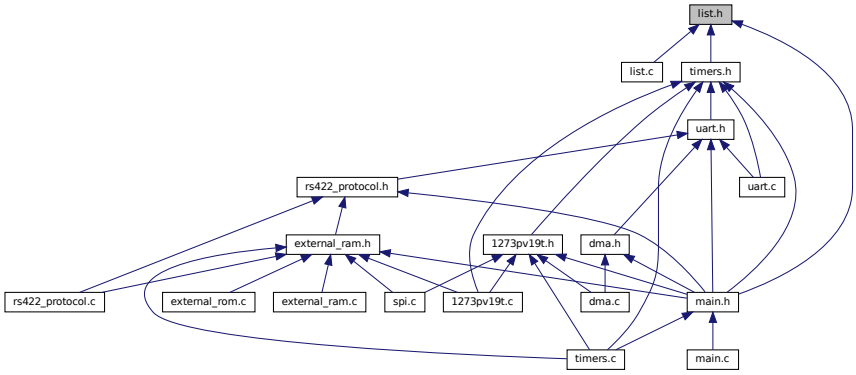
4.18 Файл list.h

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 двусвязанного 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 двусвязанного 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 двусвязанного 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

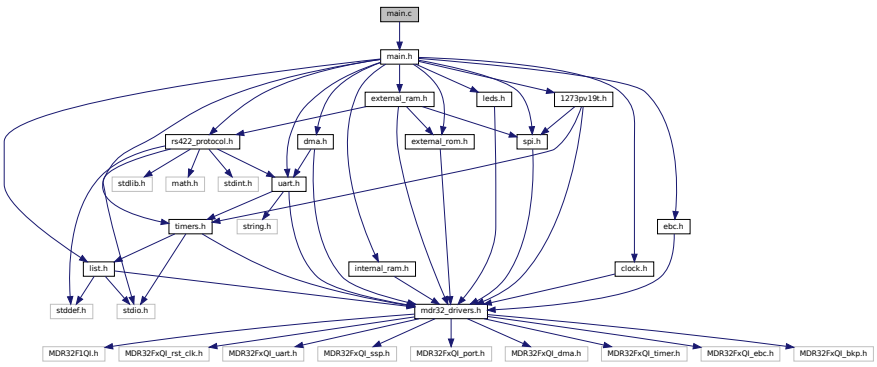
Сообщение с результатом (1 - является, 0 - не является)

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_chanels (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

*adc_struct	- Структура с конфигурационными параметрами микросхемы АЦП

```
197 {
198     // Темповые переменные, необходиме 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
        (для тока в карту регистров надо добавлять свои полиномы, т.к. они отличаются)
203 
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, CHANEL_NUMBER);
230
231     for (uint8_t k = 0; k < CHANEL_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*CHANEL_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 соответсвующий 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 соответсвующий 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_chanels, 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 переваливает максимальное число каналов АЦП, значит принятые данные соответсвуют
        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 }
```

4.19.1.4 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 ответноно 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_chanels()

```
void sync_adc_chanels (
                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]

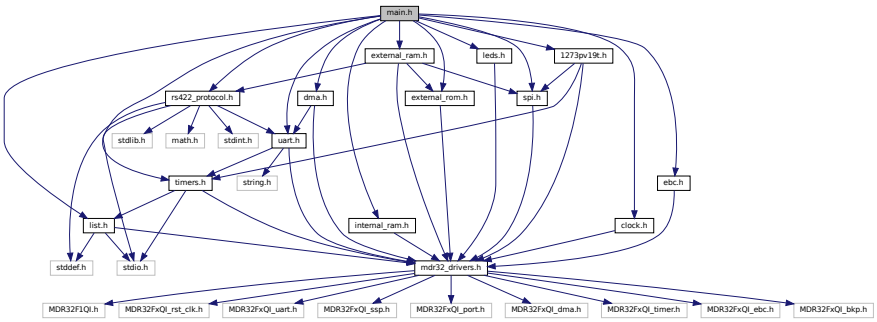
Структура с конфигурационными параметрами блока UART2 МК

4.20 Файл main.h

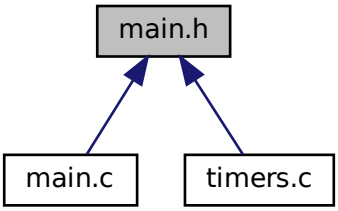
Заголовочный файл с реализацией основного функционала МПА

```
#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_chanels (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)

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

*adc_struct	- Структура с конфигурационными параметрами микросхемы АЦП

197 {

```
198     // Темповые переменные, необходиме 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
        (для тока в карту регистров надо добавлять свои полиномы, т.к. они отличаются)
203 
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, CHANEL_NUMBER);
230 
231     for (uint8_t k = 0; k < CHANEL_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*CHANEL_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 соответсвующий 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 соответсвующий 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 }
```

4.20.2.2 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 переваливает максимальное число каналов АЦП, значит принятые данные соответсвуют
        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 ответноно 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_chanels()

```
void sync_adc_chanels (
                void * data )
```

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

Аргументы

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

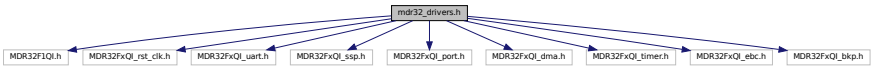
```
264 {
265     // Только, если инициализирована микросхема АЦП
266     if ((adc_1.init_flag == 1))
267     {
268         // Выключаем прерывания таймера, срабатываемое при переполнении счетчика таймера
269         TIMER_ITConfig(adc_1.timer_n_capture->TIMERx, TIMER_STATUS_CNT_ARR, DISABLE);
270 
271         // 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 }
```

4.21 Файл mdr32_drivers.h

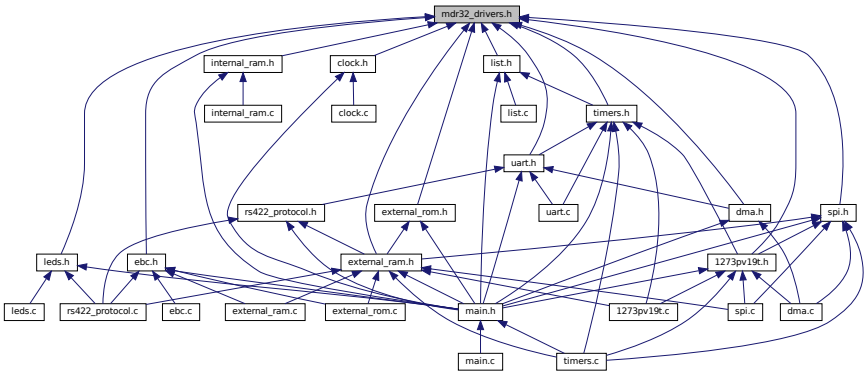
Заголовочный файл с глобальными константами и подключаемыми библиотеками

```
#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 <MDR32FxQI_bkp.h>
Граф включаемых заголовочных файлов для mdr32_drivers.h:



Граф файлов, в которые включается этот файл:



Макросы

- #define K1986VE1T
  Выбор МК K1986BE1T
- #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

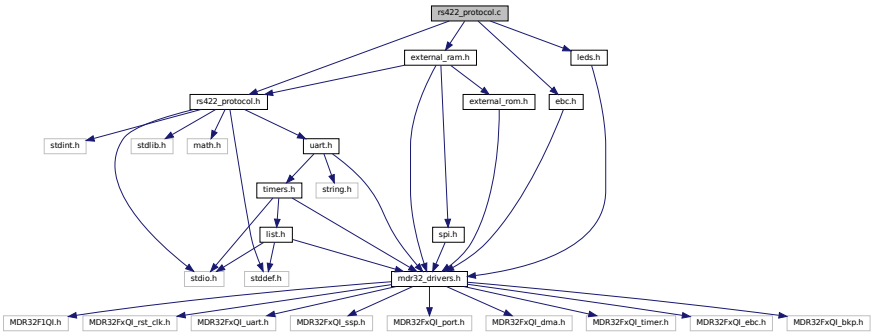
#define WORK_FREQ 144

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 дата 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),
        sizeof(common_ram_reg_space_ptr->PLC_SoftVer));
226                 offset += sizeof(common_ram_reg_space_ptr->PLC_SoftVer);
227                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->PLC_Config),
        sizeof(common_ram_reg_space_ptr->PLC_Config));
228                 offset += sizeof(common_ram_reg_space_ptr->PLC_Config);
229                 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));
230                 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_DeviceType);
231                 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));
232                 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
233 
234                 (tx_pack_cmd_with_data_ptr + i)->header.cmd = TYPE_CMD;
235                 // TODO:разобраться для чего поле result
236                 (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
237                 (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) +
        sizeof(common_ram_reg_space_ptr->PLC_SoftVer) +
238                     sizeof(ram_space_pointer->common_ram_register_space.PLC_Config) +
        sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_DeviceType) +
        sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
239                 break;
240 
241             case INIT_CMD:
242                 // По команде INIT кладем в поле дата регистр PLC_SerialNumber, если выполнен ряд условий
243                 switch (ext_bus)
244                 {
245                     case 1:
246                         if((ram_space_pointer->service_byte_um.ready_to_control == 1) &&
        (ram_space_pointer->service_byte_pm.fail_bus_1 == 0) && (common_ram_reg_space_ptr->PLC_CM_State !=
        PLC_CM_INIT_2_BUS))
247                         {
248                             // Заносим инфу в сервисный байт ПМ
249                             ram_space_pointer->service_byte_pm.init = 1;
250                             ram_space_pointer->service_byte_pm.master = 0;
251                             // Заносим инфу в регистры
252                             common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_INIT_1_BUS;
253                         }
254                         else
255                         {
256                             continue;
257                         }
258                         break;
259 
260                     case 2:
261                         if((ram_space_pointer->service_byte_um.ready_to_control == 1) &&
        (ram_space_pointer->service_byte_pm.fail_bus_2 == 0) && (common_ram_reg_space_ptr->PLC_CM_State !=
        PLC_CM_INIT_1_BUS))
262                         {
263                             // Заносим инфу в сервисный байт ПМ
264                             ram_space_pointer->service_byte_pm.init = 1;
265                             ram_space_pointer->service_byte_pm.master = 1;
266                             // 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 дата 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_PMAddr.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,
    sizeof(common_regs));
400                     memcpy(&mpa_regs, &ram_space_pointer->mpa_ram_register_space.AI_RomRegs, sizeof(mpa_regs));
401                     ebc_init(EBC_ROM);
402                     memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR, &common_regs, sizeof(common_regs));
403                     memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR + sizeof(common_regs), &mpa_regs,
    sizeof(mpa_regs));
404                     ebc_init(EBC_RAM);
405                 #endif
406 
407                 memset((ram_space_pointer->tx_data) + offset, CONFIG, 1);
408                 (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
409                 offset++;
410                 (tx_pack_cmd_with_data_ptr + i)->header.cmd = CONFIG;
411                 // TODO:разобраться для чего поле result
412                 (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
413                 (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + 1;
414                 break;
415 
416             default:
417                 break;
418         }
419         tx_pack_header->packet_length += (tx_pack_cmd_with_data_ptr + i)->header.length;
420     }
421     // Заполнение остальных полей
422     tx_pack_header->header = rx_pack_header->header;
423     tx_pack_header->receiver_addr = rx_pack_header->sender_addr;
424     tx_pack_header->sender_addr = rx_pack_header->receiver_addr;
425     tx_pack_header->packet_length += sizeof(ram_space_pointer->tx_packet_struct.packet_header) -
     sizeof(tx_pack_header->header) +
426         sizeof(tx_pack_tail->checksum);
427     memcpy(&(tx_pack_header->service_byte), &(ram_space_pointer->service_byte_pm), sizeof(ram_space_pointer-
    >service_byte_pm));
428     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),
    (ram_space_pointer->packet_rx) + sizeof(rx_pack_header_ptr->header));
118         if (uart_error != 0)
119         {
120             error = UART_ERROR;
121             rx_error_handler(error, ext_bus);
122             return error;
123         }
124         memcpy(rx_pack_header_ptr, ram_space_pointer->packet_rx, sizeof(rx_pack_ptr->packet_header));
125 
126         // Анализ длины пакета
127         if (rx_pack_header_ptr->packet_length > UART_BUFFER_SIZE)
128         {
129             error = PACKET_ERROR;
130             rx_error_handler(error, ext_bus);
131             return error;
132         }
133 
134         // Считываем весь пакет вычисленной длины
135         uart_error = uart_read(uart_struct, (rx_pack_header_ptr->packet_length) - sizeof(rx_pack_ptr->packet_header)
    + sizeof(rx_pack_header_ptr->header) + sizeof(rx_pack_tail_ptr->end),
136             (ram_space_pointer->packet_rx) + sizeof(fields_packet_header));
137         if (uart_error != 0)
138         {
139             error = UART_ERROR;
140             rx_error_handler(error, ext_bus);
141             return error;
142         }
143 
144         // Считываем хвост пакета
145         memcpy(&(rx_pack_ptr->packet_tail), ram_space_pointer->packet_rx + (rx_pack_header_ptr->packet_length) -
    sizeof(rx_pack_tail_ptr->checksum) + sizeof(rx_pack_header_ptr->header), sizeof(rx_pack_ptr->packet_tail));
146 
147         // Вторая обязательная проверка - последние 2 байта должны быть 0xAA
148         if (rx_pack_tail_ptr->end != PACKET_TAIL)
149         {
150             error = PACKET_ERROR;
151             rx_error_handler(error, ext_bus);
152             return error;
153         }
154 
155         current_rx_packet++;
156     }
157     // После того, как распознали последниий принятый пакет, продолжаем его обработку
158     // Распознавание сервисного байта УМ
159     memcpy(&(ram_space_pointer->service_byte_um), &(rx_pack_header_ptr->service_byte),
    sizeof(rx_pack_header_ptr->service_byte));
160     // Обработка сервисного байта УМ
161     um_service_byte_handler(ext_bus);
162     // Считываем массив команд (команда - данные)
163     uint16_t buffer_offset = 0; //перемещение по буферу
164     for (uint32_t i = 0; i < (rx_pack_header_ptr->cmd_number); i++)
165     {
166         // Считываем заголовок - команда, результат, данные
167         memcpy(&((rx_pack_cmd_with_data_ptr + i)->header), ram_space_pointer->packet_rx + sizeof(rx_pack_ptr-
    >packet_header) + buffer_offset, sizeof((rx_pack_cmd_with_data_ptr + i)->header));
168         // Анализ длины команды
169         if ((rx_pack_cmd_with_data_ptr + i)->header.length > UART_BUFFER_SIZE)
170         {
171             error = PACKET_ERROR;
172             return error;
173         }
174         // Считываем указатель на данные
175         (rx_pack_cmd_with_data_ptr + i)->data = ram_space_pointer->packet_rx + sizeof(rx_pack_ptr-
    >packet_header) + buffer_offset + sizeof((rx_pack_cmd_with_data_ptr + i)->header);
176         buffer_offset += (rx_pack_cmd_with_data_ptr + i)->header.length;
177     }
178 
179     // Вычисление контрольной суммы и сравнение ее с той, что в телеграмме
180     uint32_t real_checksum = crc32((ram_space_pointer->packet_rx) + sizeof(rx_pack_header_ptr->header),
    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_PMAddr.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;
        //указатель на область памяти внешнего ОЗУ с общими регистрами
469 
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
        корректно принятых пакетов
479                     common_ram_reg_space_ptr->PLC_ErrPackToDevice_B1 = 0;// Кол-во поврежденных
        пакетов подряд
480                     ram_space_pointer->service_byte_pm.fail_bus_1 = 0; // Снятие в сервисном байте ПМ бита
        несиправности шины
481                     common_ram_reg_space_ptr->PLC_BusDefect_B1.many_fail_packet = 0; // Снятие в
        регистре неисправности шины бита "кол-во битых пакетов больше установленного"
482                     common_ram_reg_space_ptr->PLC_BusDefect_B1.fail_timeout = 0; // Снятие в регистре
        неисправности шины бита "неисправность по таймауту"
483                     break;
484 
485                 case 2:
486                     common_ram_reg_space_ptr->PLC_CorrPackToDevice_B2++; // Увеличиваем счетчик
        корректно принятых пакетов
487                     common_ram_reg_space_ptr->PLC_ErrPackToDevice_B2 = 0;// Кол-во поврежденных
        пакетов подряд
488                     ram_space_pointer->service_byte_pm.fail_bus_2 = 0;    // Снятие в сервисном байте ПМ
        бита несиправности шины
489                     common_ram_reg_space_ptr->PLC_BusDefect_B2.many_fail_packet = 0; // Снятие в
        регистре неисправности шины бита "кол-во битых пакетов больше установленного"
490                     common_ram_reg_space_ptr->PLC_BusDefect_B2.fail_timeout = 0; // Снятие в регистре
        неисправности шины бита "неисправность по таймауту"
491                     break;
492 
493                 default:
```

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

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

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]

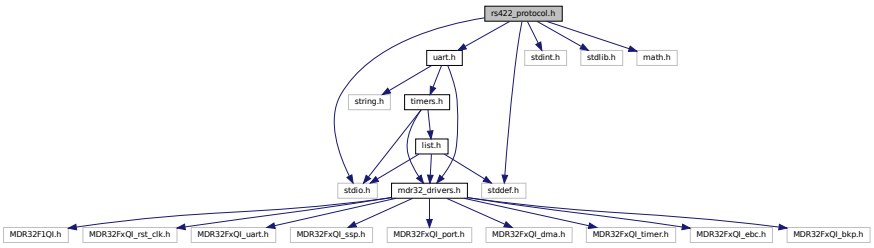
Указатель для обращения к внешнему ПЗУ

4.23 Файл rs422_protocol.h

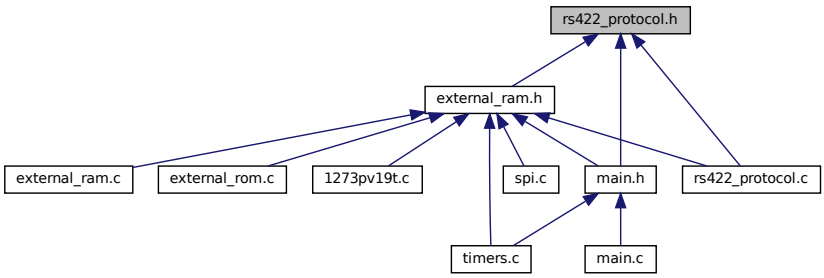
Заголовочный файл с описанием API протокола обмена данными по интерфейсу RS-422.

```
#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 дата 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), sizeof(common_ram_reg_space_ptr->PLC_SoftVer));
226                 offset += sizeof(common_ram_reg_space_ptr->PLC_SoftVer);
227                 memcpy((ram_space_pointer->tx_data) + offset, &(common_ram_reg_space_ptr->PLC_Config), sizeof(common_ram_reg_space_ptr->PLC_Config));
228                 offset += sizeof(common_ram_reg_space_ptr->PLC_Config);
229                 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));
230                 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_DeviceType);
231                 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));
232                 offset += sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
233 
234                 (tx_pack_cmd_with_data_ptr + i)->header.cmd = TYPE_CMD;
235                 // TODO:разобраться для чего поле result
236                 (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
237                 (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + sizeof(common_ram_reg_space_ptr->PLC_SoftVer) +
238                     sizeof(ram_space_pointer->common_ram_register_space.PLC_Config) + sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_DeviceType) + sizeof(common_ram_reg_space_ptr->PLC_CommonRomRegs.PLC_SerialNumber);
239                 break;
240 
241             case INIT_CMD:
242                 // По команде INIT кладем в поле дата регистр PLC_SerialNumber, если выполнен ряд условий
243                 switch (ext_bus)
244                 {
245                     case 1:
246                         if((ram_space_pointer->service_byte_um.ready_to_control == 1) && (ram_space_pointer->service_byte_pm.fail_bus_1 == 0) && (common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_2_BUS))
247                         {
248                             // Заносим инфу в сервисный байт ПМ
249                             ram_space_pointer->service_byte_pm.init = 1;
250                             ram_space_pointer->service_byte_pm.master = 0;
251                             // Заносим инфу в регистры
252                             common_ram_reg_space_ptr->PLC_CM_State = PLC_CM_INIT_1_BUS;
253                         }
254                         else
255                         {
256                             continue;
257                         }
258                         break;
259 
260                     case 2:
261                         if((ram_space_pointer->service_byte_um.ready_to_control == 1) && (ram_space_pointer->service_byte_pm.fail_bus_2 == 0) && (common_ram_reg_space_ptr->PLC_CM_State != PLC_CM_INIT_1_BUS))
262                         {
263                             // Заносим инфу в сервисный байт ПМ
264                             ram_space_pointer->service_byte_pm.init = 1;
265                             ram_space_pointer->service_byte_pm.master = 1;
266                             // 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 дата 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_PMAddr.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,
    sizeof(common_regs));
400                     memcpy(&mpa_regs, &ram_space_pointer->mpa_ram_register_space.AI_RomRegs, sizeof(mpa_regs));
401                     ebc_init(EBC_ROM);
402                     memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR, &common_regs, sizeof(common_regs));
403                     memcpy_to_rom(ROM_REGISTER_SPACE_START_ADDR + sizeof(common_regs), &mpa_regs,
    sizeof(mpa_regs));
404                     ebc_init(EBC_RAM);
405                 #endif
406 
407                 memset((ram_space_pointer->tx_data) + offset, CONFIG, 1);
408                 (tx_pack_cmd_with_data_ptr + i)->data = (ram_space_pointer->tx_data) + offset;
409                 offset++;
410                 (tx_pack_cmd_with_data_ptr + i)->header.cmd = CONFIG;
411                 // TODO:разобраться для чего поле result
412                 (tx_pack_cmd_with_data_ptr + i)->header.result = 0;
413                 (tx_pack_cmd_with_data_ptr + i)->header.length = sizeof(fields_cmd_header) + 1;
414                 break;
415 
416         default:
417                 break;
418     }
419     tx_pack_header->packet_length += (tx_pack_cmd_with_data_ptr + i)->header.length;
420 }
421 // Заполнение остальных полей
422 tx_pack_header->header = rx_pack_header->header;
423 tx_pack_header->receiver_addr = rx_pack_header->sender_addr;
424 tx_pack_header->sender_addr = rx_pack_header->receiver_addr;
425 tx_pack_header->packet_length += sizeof(ram_space_pointer->tx_packet_struct.packet_header) -
    sizeof(tx_pack_header->header) +
426     sizeof(tx_pack_tail->checksum);
427 memcpy(&(tx_pack_header->service_byte), &(ram_space_pointer->service_byte_pm), sizeof(ram_space_pointer-
    >service_byte_pm));
428 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),
    (ram_space_pointer->packet_rx) + sizeof(rx_pack_header_ptr->header));
118         if (uart_error != 0)
119         {
120             error = UART_ERROR;
121             rx_error_handler(error, ext_bus);
122             return error;
123         }
124         memcpy(rx_pack_header_ptr, ram_space_pointer->packet_rx, sizeof(rx_pack_ptr->packet_header));
125 
126         // Анализ длины пакета
127         if (rx_pack_header_ptr->packet_length > UART_BUFFER_SIZE)
128         {
129             error = PACKET_ERROR;
130             rx_error_handler(error, ext_bus);
131             return error;
132         }
133 
134         // Считываем весь пакет вычисленной длины
135         uart_error = uart_read(uart_struct, (rx_pack_header_ptr->packet_length) - sizeof(rx_pack_ptr->packet_header)
    + sizeof(rx_pack_header_ptr->header) + sizeof(rx_pack_tail_ptr->end),
136             (ram_space_pointer->packet_rx) + sizeof(fields_packet_header));
137         if (uart_error != 0)
138         {
139             error = UART_ERROR;
140             rx_error_handler(error, ext_bus);
141             return error;
142         }
143 
144         // Считываем хвост пакета
145         memcpy(&(rx_pack_ptr->packet_tail), ram_space_pointer->packet_rx + (rx_pack_header_ptr->packet_length) -
    sizeof(rx_pack_tail_ptr->checksum) + sizeof(rx_pack_header_ptr->header), sizeof(rx_pack_ptr->packet_tail));
146 
147         // Вторая обязательная проверка - последние 2 байта должны быть 0xAA
148         if (rx_pack_tail_ptr->end != PACKET_TAIL)
149         {
150             error = PACKET_ERROR;
151             rx_error_handler(error, ext_bus);
152             return error;
153         }
154 
155         current_rx_packet++;
156     }
157     // После того, как распознали последниий принятый пакет, продолжаем его обработку
158     // Распознавание сервисного байта УМ
159     memcpy(&(ram_space_pointer->service_byte_um), &(rx_pack_header_ptr->service_byte),
    sizeof(rx_pack_header_ptr->service_byte));
160     // Обработка сервисного байта УМ
161     um_service_byte_handler(ext_bus);
162     // Считываем массив команд (команда - данные)
163     uint16_t buffer_offset = 0; //перемещение по буферу
164     for (uint32_t i = 0; i < (rx_pack_header_ptr->cmd_number); i++)
165     {
166         // Считываем заголовок - команда, результат, данные
167         memcpy(&((rx_pack_cmd_with_data_ptr + i)->header), ram_space_pointer->packet_rx + sizeof(rx_pack_ptr-
    >packet_header) + buffer_offset, sizeof((rx_pack_cmd_with_data_ptr + i)->header));
168         // Анализ длины команды
169         if ((rx_pack_cmd_with_data_ptr + i)->header.length > UART_BUFFER_SIZE)
170         {
171             error = PACKET_ERROR;
172             return error;
173         }
174         // Считываем указатель на данные
175         (rx_pack_cmd_with_data_ptr + i)->data = ram_space_pointer->packet_rx + sizeof(rx_pack_ptr-
    >packet_header) + buffer_offset + sizeof((rx_pack_cmd_with_data_ptr + i)->header);
176         buffer_offset += (rx_pack_cmd_with_data_ptr + i)->header.length;
177     }
178 
179     // Вычисление контрольной суммы и сравнение ее с той, что в телеграмме
180     uint32_t real_checksum = crc32((ram_space_pointer->packet_rx) + sizeof(rx_pack_header_ptr->header),
    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_PMAddr.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;
        //указатель на область памяти внешнего ОЗУ с общими регистрами
469 
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
        корректно принятых пакетов
479                     common_ram_reg_space_ptr->PLC_ErrPackToDevice_B1 = 0;// Кол-во поврежденных
        пакетов подряд
480                     ram_space_pointer->service_byte_pm.fail_bus_1 = 0; // Снятие в сервисном байте ПМ бита
        несиправности шины
481                     common_ram_reg_space_ptr->PLC_BusDefect_B1.many_fail_packet = 0; // Снятие в
        регистре неисправности шины бита "кол-во битых пакетов больше установленного"
482                     common_ram_reg_space_ptr->PLC_BusDefect_B1.fail_timeout = 0; // Снятие в регистре
        неисправности шины бита "неисправность по таймауту"
483                     break;
484 
485                 case 2:
486                     common_ram_reg_space_ptr->PLC_CorrPackToDevice_B2++; // Увеличиваем счетчик
        корректно принятых пакетов
487                     common_ram_reg_space_ptr->PLC_ErrPackToDevice_B2 = 0;// Кол-во поврежденных
        пакетов подряд
488                     ram_space_pointer->service_byte_pm.fail_bus_2 = 0;    // Снятие в сервисном байте ПМ
        бита несиправности шины
489                     common_ram_reg_space_ptr->PLC_BusDefect_B2.many_fail_packet = 0; // Снятие в
        регистре неисправности шины бита "кол-во битых пакетов больше установленного"
490                     common_ram_reg_space_ptr->PLC_BusDefect_B2.fail_timeout = 0; // Снятие в регистре
        неисправности шины бита "неисправность по таймауту"
491                     break;
492 
493                 default:
```

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

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

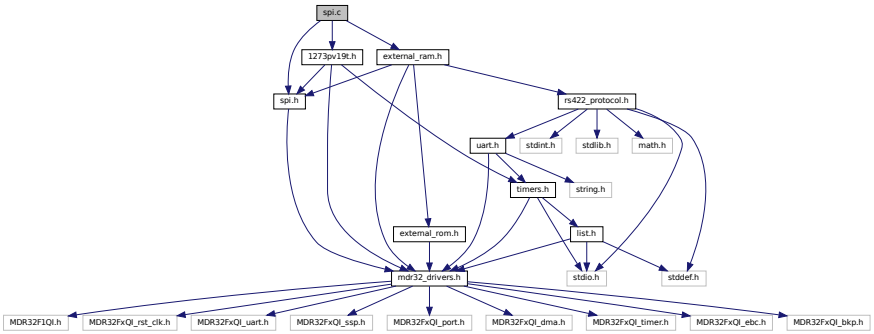
ext_bus	- Номер шины

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

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

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 )
```

Инициализацирует 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_CPSDVSR = spi_struct->SPI.SSP_CPSDVSR; // 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 SPIn.

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 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 }
```

4.24.2.7 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

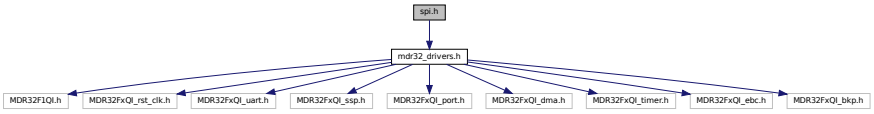
spi_n spi_2

Структура с конфигурационными параметрами блока SPI2 МК

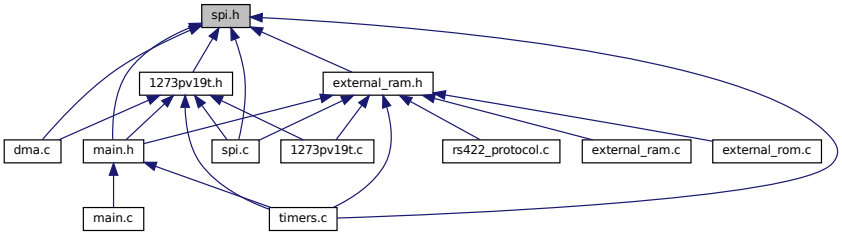
4.25 Файл spi.h

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

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 )
```

Инициализацирует 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_CPSDVSR = spi_struct->SPI.SSP_CPSDVSR; // 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 SPIn.

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

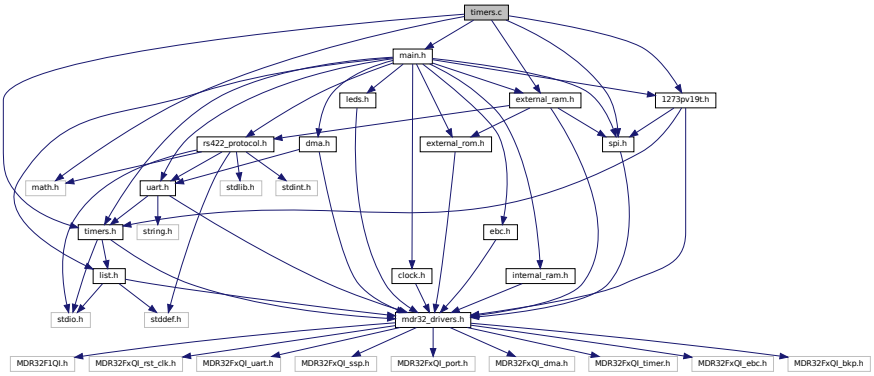
```
93 {
94     for(uint32_t i = 0; i < length; i++)
95     {
96         while(SSP_GetFlagStatus(spi_struct->SSPx, SSP_FLAG_TNF) != SET) {} // Ждем, когда в буфере появится
       место и затем записываем следующий байт
97         SSP_SendData(spi_struct->SSPx, message[i]);
98     }
99     while (SSP_GetFlagStatus(spi_struct->SSPx,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

4.26 Файл timers.c

Файл с реализацией API для работы с таймерами

```
#include "timers.h"
#include "spi.h"
#include "1273pv19t.h"
#include "external_ram.h"
#include <math.h>
#include "main.h"
```

Граф включаемых заголовочных файлов для timers.c:



Функции

- void timer1_init (timer_n *timer_struct)

  Инициализирует Timer1.
- void timer3_init (timer_n *timer_struct)

  Инициализирует Timer3.
- void timer2_init (timer_n *timer_struct)

  Инициализирует Timer2.
- void timer_init (timer_n *timer_struct)

  Инициализацирует выбранный Timer.
- void TIMER2_IRQHandler (void)

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

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

time_micro	- 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 )
```

Инициализацирует 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]

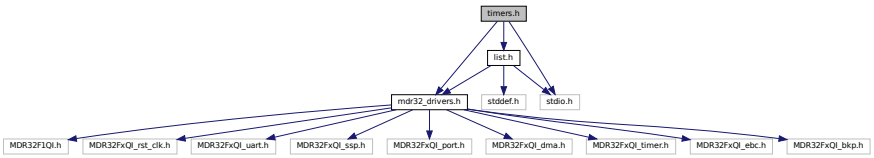
Указатели на списки обработчиков таймеров

Массив указателей на head списков обработчиков прерываний таймеров

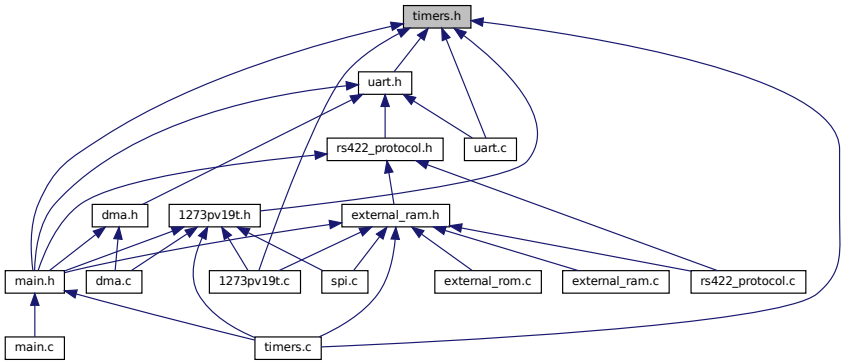
4.27 Файл timers.h

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)

  Инициализацирует 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 )
```

Инициализацирует выбранный Timer.

Аргументы

*timer_struct	- Таймер для инициализации

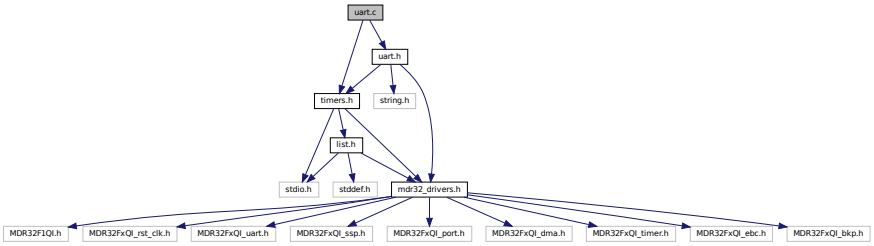
```
121 {
122     if (timer_struct->TIMERx == MDR_TIMER1)
123     {
124         timer1_init(timer_struct);
125     }
```

```
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)
  Инициализацирует 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 MK
- uart_n uart_2

  Структура с конфигурационными параметрами блока UART2 MK

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,
         ENABLE);
79
80     // Инициализации портов UART1
81     PORT_InitTypeDef GPIO_init_structUART1;
82
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     // Дезактивирование 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     // Дезактивирование 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 )
```

Инициализацирует 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 перевалит 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 перевалит 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     // Дезактивирование 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 MK

Структуры с конфигурационными параметрами блоков UART MK

4.28.3.2 uart_2

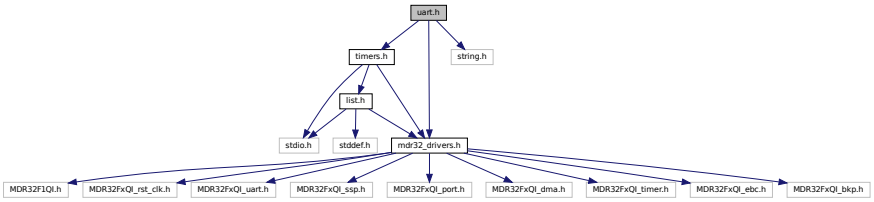
uart_n uart_2

Структура с конфигурационными параметрами блока UART2 MK

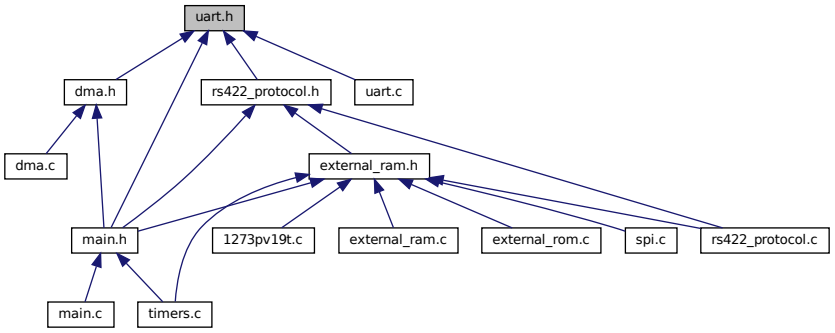
4.29 Файл uart.h

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)

  Инициализацирует 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:

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

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

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

4.29.5.2 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 )
```

Инициализацирует 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 перевалит 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 перевалит 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     // Дезактивирование 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