МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Информационная безопасность систем и технологий»

Отчет
по практической работе №5
на тему «Работа с последовательными интерфейсами в режиме DMA в среде STM32CubeIDE»

Дисциплина: ПМК
Группа: 21ПИ1
Выполнил: Гусев Д. А.
Количество баллов:
Дата сдачи:
Принял: Хворостухин С. П.

2024

1 Цель работы: изучить средства конфигурирования последовательных интерфейсов и контроллера DMA в среде разработки STM32CubeIDE.

2 Задание на практическую работу.

2.1 Выполнить настройку последовательного интерфейса обмена данными в соответствии с вариантом в графическом интерфейсе среды разработки STM32CubeIDE. Варианты приведен на рисунке 1.

8	*USART1*	*Asynchronous*

Рисунок 1 — Вариант задания

2.2 Выполнить подключение приемника и передатчика последовательного интерфейса обмена данными в настройках контроллера DMA (номер контроллера DMA определяется типом подключаемого интерфейса).

2.3 Выполнить подключение прерываний контроллера последовательного интерфейса и контроллера DMA в настройках контроллера прерываний NVIC.

2.4 Сгенерировать программный код, соответствующий заданной конфигурации.

2.5 Проанализировать функции настройки контроллеров последовательного интерфейса и DMA - файл main.c.

2.6 Проанализировать программный код библиотеки HAL, приведенный в файлах: stm32f4xx_hal_dma_ex.c; stm32f4xx_hal_dma.c; stm32f4xx_it.c; stm32f4xx_hal_msp.c; stm32f4xx_hal_uart.c — согласно варианту задания.

2.7 Сформировать описание проанализированного программного кода.

3 Выполнение практической работы:

3.1 Была выполнена настройка последовательного интерфейса обмена данными в соответствии с вариантом 8.

3.2 Была выполнена настройка последовательного интерфейса обмена данными в соответствии с вариантом 8.

3.3 Было выполнено подключение прерываний контроллера последовательного интерфейса и контроллера DMA в настройках контроллера прерываний NVIC. Результат представлен на рисунке 2.

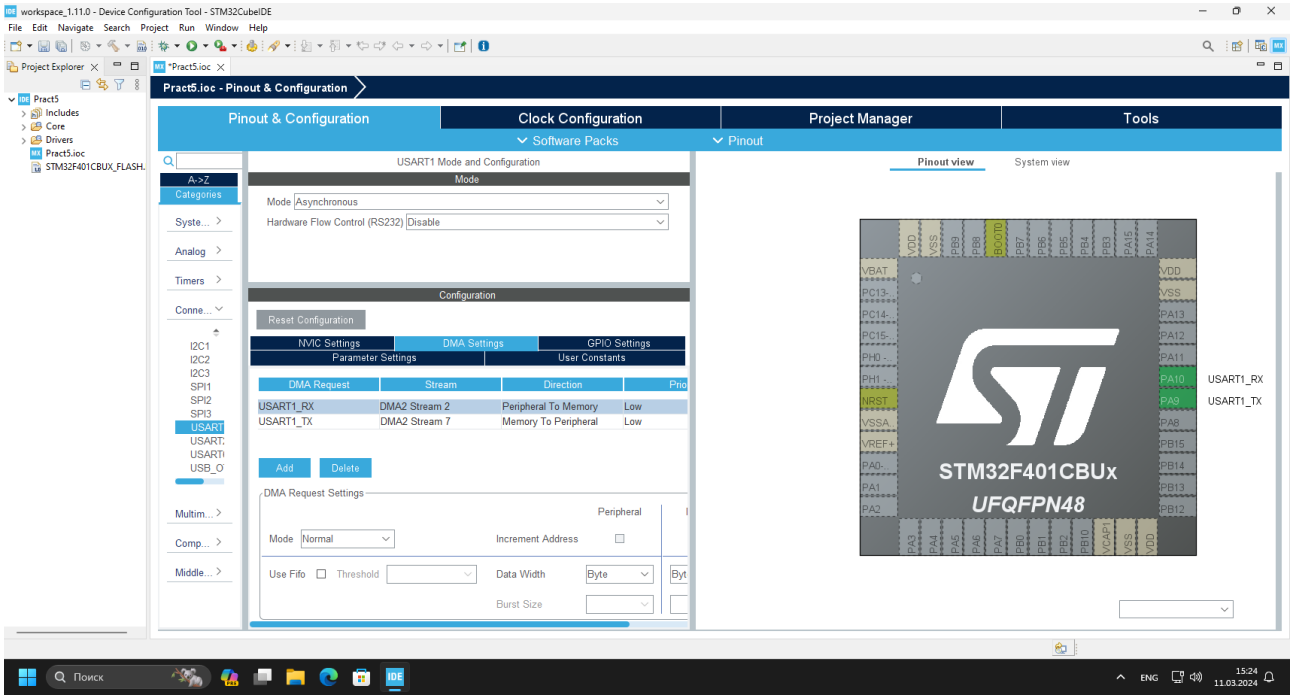

Рисунок 2 — Настройка портов

3.4 Был сгенерирован программный код, соответствующий заданной конфигурации.

3.5 Были проанализированы функции настройки контроллеров последовательного интерфейса и DMA в файле main.c.

3.5.1 Функция настройки контроллеров последовательного интерфейса MX_USART1_UART_Init находится в файле main.c

Возвращаемое значение функции: void - нет возвращаемого значения.

Параметры функции: нет параметров.

Краткий алгоритм работы функции:

1) USART1 настраивается на скорость 115200 бит/с, формат кадра 8 бит данных, 1 стоп-бит, без бита четности, в режиме передачи и приема (TX_RX mode).

2) Дополнительные параметры настройки UART, такие как контроль потоком и оверсемплинг, также устанавливаются.

3) Если инициализация UART завершается неудачно, программа переходит к функции Error_Handler, которая в данном случае просто отключает прерывания и переводит систему в бесконечный цикл.

3.5.2 Функция настройки DMA MX_DMA_Init находится в файле main.c.

Возвращаемое значение функции: void - нет возвращаемого значения.

Параметры функции: нет параметров.

Краткий алгоритм работы функции:

1) Устанавливается DMA2 контроллер.

2) Настройка прерываний для потока 2 и потока 7 DMA.

3.6 Был проанализирован программный код библиотеки HAL, приведенный в файлах: stm32f4xx_hal_dma_ex.c; stm32f4xx_hal_dma.c; stm32f4xx_it.c; stm32f4xx_hal_msp.c; stm32f4xx_hal_uart.c.

3.6.1 stm32f4xx_hal_dma_ex.c: Расширенные функции и настройки для управления прямым доступом к памяти (DMA) в STM32F4xx. Этот файл может содержать дополнительные функции и конфигурации, связанные с DMA.

3.6.2 stm32f4xx_hal_dma.c: Основные функции и настройки для управления прямым доступом к памяти (DMA) в STM32F4xx. Содержатся функции и структуры данных для инициализации, настройки и использования DMA.

3.6.3 stm32f4xx_it.c: Файл прерываний, который содержит обработчики прерываний (IRQ handlers) для обработки прерываний от различных периферийных устройств или событий на микроконтроллере STM32F4xx.

3.6.4 stm32f4xx_hal_msp.c: Файл для управления периферийными функциями микроконтроллера STM32F4xx. В нем часто содержатся функции обратного вызова (callback functions), которые позволяют настроить или переопределить стандартное поведение библиотеки.

3.6.5 stm32f4xx_hal_uart.c: Файл, содержащие функции для работы с UART (USART), SPI и I2C интерфейсами соответственно. В файле

реализованы функции инициализации, передачи данных, приема данных и другие операции для соответствующих периферийных устройств.

3.7 Был описан программный код библиотеки HAL, приведенный в файлах: stm32f4xx_hal_dma_ex.c; stm32f4xx_hal_dma.c; stm32f4xx_it.c; stm32f4xx_hal_msp.c; stm32f4xx_hal_uart.c. Описание кода представлено в приложениях А, Б, В, Г и Д.

4 Вывод: были изучены средства конфигурирования последовательных интерфейсов и контроллера DMA в среде разработки STM32CubeIDE.

Приложение А

Описание файла stm32f4xx_hal_dma_ex.c

```
/**
******************************************************************************
***
  * @file stm32f4xx_hal_dma_ex.c
  * @Author MCD -команда приложений
  * @brief DMA Extension Hal Driver
  * Этот файл предоставляет функции прошивки для управления следующим
  * Функциональные возможности периферийного устройства расширения DMA:
  * + Функции расширенных функций
  *
  @verbatim

==============================================================================
=================
                     ##### Как использовать этот драйвер #####
==============================================================================
=================
  [..]
  Драйвер удлинения DMA можно использовать следующим образом:
   (#) Запустите много буферного переноса, используя функцию hal_dma_multibufferstart
()
       для режима опроса или hal_dma_multibufferstart_it () для режима прерывания.
     -@-В режиме передачи памяти в память, много (двойной) буферный режим не
допускается.
     -@- Когда много (двойной) буферный режим включен, передача по умолчанию по
умолчанию.
     -@- В режиме с несколькими (двойными) буфером можно обновить базовый адрес для
          Порт памяти AHB на лету (DMA_SXM0AR или DMA_SXM1AR) при включении потока.
  @endverbatim

******************************************************************************
***
  * @внимание
  *
  * Copyright (C) 2017 Stmicroelectronics.
```

```
  * Все права защищены.
  *
  * Это программное обеспечение лицензировано в соответствии с условиями, которые можно
найти в файле лицензии в
  * Корневой каталог этого программного компонента.
  * Если нет лицензионного файла с этим программным обеспечением, оно предоставляется
как есть.
  *

******************************************************************************
***
  */
/* Включает ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
/** @addtogroup stm32f4xx_hal_driver
  * @{
  */
/** @defgroup dmaex dmaex
  * @brief DMA расширенный драйвер модуля HAL
  * @{
  */
#ifdef HAL_DMA_MODULE_ENABLED
/* Частные типы
------------------------------------------------------------------------*/
/* Приватные переменные ---------------------------------------------------------*/
/* Частные константы ---------------------------------------------------------*/
/* Приватные макросы ---------------------------------------------------------*//
/* Приватные функции
-----------------------------------------------------------------*/
/** @addtogroup dmaex_private_functions
  * @{
  */
static void DMA_MultiBufferSetConfig(DMA_HandleTypeDef *hdma, uint32_t SrcAddress,
uint32_t DstAddress, uint32_t DataLength);
/**
  * @}
  */
/* Экспортируемые функции ---------------------------------------------------------*/
/** @addtogroup dmaex_exported_functions
  * @{
```

```
  */

/** @addtogroup dmaex_exported_functions_group1
  *
@verbatim

==============================================================================
=================
                ##### Функции расширенных функций #####

==============================================================================
=================
    [..] В этом разделе представлены функции, позволяющие:
      (+) Настройте источник, адрес назначения и длину данных и
          Начните многобуфере DMA Transfer
      (+) Настройте источник, адрес назначения и длину данных и
          Начните многопользовательский DMA Transfer с прерыванием
      (+) Измените на Fly на адрес Memory0 или Memory1.
@endverbatim
  * @{
  */

/**
  * @brief запускает Multi_buffer DMA Transfer.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param srcaddress Адрес буфера памяти источника
  * @param dstaddress Адрес буфера памяти назначения
  * @param secondmemaddress Второй адрес буфера памяти в случае многократного буфера
передачи
  * @param DataLength продолжительность передачи данных из источника в назначение
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart(DMA_HandleTypeDef *hdma, uint32_t
SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)
{
  HAL_StatusTypeDef status = HAL_OK;
  / * Проверьте параметры */
  assert_param(IS_DMA_BUFFER_SIZE(DataLength));
  / * Передача памяти в память не поддерживается в режиме двойной буферизации */
```

```
  if (hdma->Init.Direction == DMA_MEMORY_TO_MEMORY)
  {
    hdma->ErrorCode = HAL_DMA_ERROR_NOT_SUPPORTED;
    status = HAL_ERROR;
  }
  else
  {
    / * Процесс заблокирован */
    __HAL_LOCK(hdma);
    if(HAL_DMA_STATE_READY == hdma->State)
    {
      / * Изменить периферическое состояние DMA */
      hdma->State = HAL_DMA_STATE_BUSY;
      / * Включить двойной буферный режим */
      hdma->Instance->CR |= (uint32_t)DMA_SxCR_DBM;
      / * Настроить адрес назначения потока DMA */
      hdma->Instance->M1AR = SecondMemAddress;
      / * Настройте источник, адрес назначения и длину данных */
      DMA_MultiBufferSetConfig(hdma, SrcAddress, DstAddress, DataLength);
      / * Включить периферическое */
      __HAL_DMA_ENABLE(hdma);
    }
    else
    {
      / * Статус ошибки возврата */
      status = HAL_BUSY;
    }
  }
  return status;
}
/**
  * @brief начинает передачу Multi_buffer DMA с включенным прерыванием.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param srcaddress Адрес буфера памяти источника
  * @param dstaddress Адрес буфера памяти назначения
  * @param secondmemaddress Второй адрес буфера памяти в случае многократного буфера
передачи
  * @param DataLength продолжительность передачи данных из источника в назначение
  * @retval hal status
```

```
  */
HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart_IT(DMA_HandleTypeDef *hdma, uint32_t
SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)
{
  HAL_StatusTypeDef status = HAL_OK;
  / * Проверьте параметры */
  assert_param(IS_DMA_BUFFER_SIZE(DataLength));
  / * Передача памяти в память не поддерживается в режиме двойной буферизации */
  if (hdma->Init.Direction == DMA_MEMORY_TO_MEMORY)
  {
    hdma->ErrorCode = HAL_DMA_ERROR_NOT_SUPPORTED;
    return HAL_ERROR;
  }
  / * Проверьте функции обратного вызова */
  if ((NULL == hdma->XferCpltCallback) || (NULL == hdma->XferM1CpltCallback) || (NULL
== hdma->XferErrorCallback))
  {
    hdma->ErrorCode = HAL_DMA_ERROR_PARAM;
    return HAL_ERROR;
  }
  / * Процесс заблокирован */
  __HAL_LOCK(hdma);
  if(HAL_DMA_STATE_READY == hdma->State)
  {
    / * Изменить периферическое состояние DMA */
    hdma->State = HAL_DMA_STATE_BUSY;
    / * Инициализировать код ошибки */
    hdma->ErrorCode = HAL_DMA_ERROR_NONE;
    / * Включить двойной буферный режим */
    hdma->Instance->CR |= (uint32_t)DMA_SxCR_DBM;
    / * Настроить адрес назначения потока DMA */
    hdma->Instance->M1AR = SecondMemAddress;
    / * Настройте источник, адрес назначения и длину данных */
    DMA_MultiBufferSetConfig(hdma, SrcAddress, DstAddress, DataLength);
    / * Очистить все флаги */
    __HAL_DMA_CLEAR_FLAG (hdma, __HAL_DMA_GET_TC_FLAG_INDEX(hdma));
    __HAL_DMA_CLEAR_FLAG (hdma, __HAL_DMA_GET_HT_FLAG_INDEX(hdma));
    __HAL_DMA_CLEAR_FLAG (hdma, __HAL_DMA_GET_TE_FLAG_INDEX(hdma));
    __HAL_DMA_CLEAR_FLAG (hdma, __HAL_DMA_GET_DME_FLAG_INDEX(hdma));
    __HAL_DMA_CLEAR_FLAG (hdma, __HAL_DMA_GET_FE_FLAG_INDEX(hdma));
```

```
    /* Включить общие прерывания*/
    hdma->Instance->CR  |= DMA_IT_TC | DMA_IT_TE | DMA_IT_DME;
    hdma->Instance->FCR |= DMA_IT_FE;
    if((hdma->XferHalfCpltCallback != NULL) || (hdma->XferM1HalfCpltCallback != NULL))
    {
      hdma->Instance->CR  |= DMA_IT_HT;
    }
    / * Включить периферическое */
    __HAL_DMA_ENABLE(hdma);
  }
  else
  {
    / * Процесс разблокирован */
    __HAL_UNLOCK(hdma);
    / * Статус ошибки возврата */
    status = HAL_BUSY;
  }
  return status;
}
/**
  * @brief Измените адрес Memory0 или Memory1 на лету.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param адрес новый адрес
  * @param memom
  * Следующие значения:
  * Memory0 /
  * Память1
  * @note Адрес Memory0 может быть изменен только при использовании текущей передачи
  * Memory1 и адрес памяти1 могут быть изменены только при текущем
  * Передача используйте память0.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMAEx_ChangeMemory(DMA_HandleTypeDef *hdma, uint32_t Address,
HAL_DMA_MemoryTypeDef memory)
{
  if(memory == MEMORY0)
  {
    / * Измените адрес памяти0 */
    hdma->Instance->M0AR = Address;
```

```
  }
  else
  {
    / * Измените адрес памяти1 */
    hdma->Instance->M1AR = Address;
  }
  return HAL_OK;
}
/**
  * @}
  */
/**
  * @}
  */
/** @addtogroup dmaex_private_functions
  * @{
  */
/**
  * @brief Установите параметр передачи DMA.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param srcaddress Адрес буфера памяти источника
  * @param dstaddress Адрес буфера памяти назначения
  * @param DataLength продолжительность передачи данных из источника в назначение
  * @retval hal status
  */
static void DMA_MultiBufferSetConfig(DMA_HandleTypeDef *hdma, uint32_t SrcAddress,
uint32_t DstAddress, uint32_t DataLength)
{
  / * Настройка длины данных потока DMA */
  hdma->Instance->NDTR = DataLength;
  / * Периферийно для памяти */
  if((hdma->Init.Direction) == DMA_MEMORY_TO_PERIPH)
  {
    / * Настроить адрес назначения потока DMA */
    hdma->Instance->PAR = DstAddress;
    / * Настройка адреса источника потока DMA */
    hdma->Instance->M0AR = SrcAddress;
  }
  / * Память к периферическому */
```

```
  else
  {
    / * Настройка адреса источника потока DMA */
    hdma->Instance->PAR = SrcAddress;
    / * Настроить адрес назначения потока DMA */
    hdma->Instance->M0AR = DstAddress;
  }
}
/**
  * @}
  */
#endif / * Hal_dma_module_enabled */
/**
  * @}
  */
/**
  * @}
  */
```

Приложение Б

Описание файла STM32F4XX_HAL_DMA.C

```
 /**

******************************************************************************
**
 * @file STM32F4XX_HAL_DMA.C
 * @Author MCD -команда приложений
 * @brief DMA HAL Driver Module.
 *
 * Этот файл предоставляет функции прошивки для управления следующим
 * Функциональные возможности периферийного устройства прямого доступа к памяти (DMA):
 * + Функции инициализации и де-инициализации
 * + Функции эксплуатации IO
 * + Функции периферического состояния и ошибок
 @verbatim

==============================================================================
================
 ##### Как использовать этот драйвер #####

==============================================================================
================
 [..]
 (#) Включите и настраивайте периферийное устройство для подключения к потоку DMA
 (За исключением внутренних воспоминаний о SRAM/Flash: нет инициализации
 необходимо), пожалуйста, обратитесь к справочному руководству для связи между
периферийными устройствами
 и DMA запросы.
 (#) Для данного потока программируйте необходимую конфигурацию через следующие
параметры:
 Направление передачи, форматы данных источника и назначения,
 Круглый, нормальный или режим управления периферическим потоком, уровень приоритета
потока,
 Режим приращения источника и назначения, режим FIFO и его порог (при необходимости),
 Режим взрыва для источника и/или пункта назначения (при необходимости) с
использованием функции hal_dma_init ().
```

-@- До HAL_DMA_INIT () часы должны быть включены для DMA через следующие макросы:
__Hal_rcc_dma1_clk_enable () или __hal_rcc_dma2_clk_enable ().

*** Режим опроса в io операция ***

==========

[..]

(+) Используйте HAL_DMA_START () для запуска передачи DMA после конфигурации источника

адрес и адрес назначения и продолжительность передачи данных.

(+) Используйте HAL_DMA_POLLFORTRANSFER () для опроса на конец текущей передачи, в этом

Случай фиксированный тайм -аут может быть настроен пользователем в зависимости от его приложения.

(+) Используйте функцию hal_dma_abort (), чтобы прервать текущую передачу.

*** Режим прерывания в io операция ***

==========

[..]

(+) Настройте приоритет прерывания DMA с использованием hal_nvic_setpriority ()

(+) Включить обработчик DMA IRQ с помощью hal_nvic_enableirq ()

(+) Используйте HAL_DMA_START_IT (), чтобы запустить передачу DMA после конфигурации

Адрес источника и адрес назначения и продолжительность передачи данных.В этом

Случай прерывания DMA настроен

(+) Используйте hal_dma_irqhandler (), называемое в подпрограмме DMA_IRQHandler ()

(+) В конце передачи данных функция HAL_DMA_IRQHandler () выполняется и пользователь может

Добавить свою собственную функцию путем настройки указателя функции xfercpltcallback и

Xfererrorcallback (то есть член структуры ручки DMA).

[..]

(#) Используйте функцию hal_dma_getstate (), чтобы вернуть состояние DMA и HAL_DMA_GETERROR () В случае ошибки

обнаружение.

(#) Используйте функцию hal_dma_abort_it (), чтобы прервать текущую передачу

-@-В режиме передачи памяти в память, круговой режим не допускается.

-@- FIFO используется в основном для сокращения использования шины и для разрешения упаковки/распаковки данных: это

возможно установить разные размеры данных для периферического и памяти (т.е.

вы можете установить

Размер данных половины слов для периферийного устройства для доступа к его регистрации данных и установке размера данных Word Data

Чтобы память, чтобы получить время доступа.Каждые два половины слов будут упакованы и написаны в

один доступ к слову в памяти).

-@- Когда FIFO отключен, не разрешается настраивать разные размеры данных для источника

и назначение.В этом случае периферический размер данных будет применен к обоим источнику

и назначение.

*** Список макросов драйверов DMA HAL ***

==========

[..]

Под списком большинства используемых макросов в драйвере DMA HAL.

(+) __Hal_dma_enable: включить указанный поток DMA.

(+) __Hal_dma_disable: отключить указанный поток DMA.

(+) __Hal_dma_get_it_source: Проверьте, произошло ли указанное прерывание потока DMA или нет.

[..]

(@) Вы можете обратиться к файлу заголовка драйвера DMA HAL для получения более полезных макросов

@endverbatim

```
  */
/* Включает ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
/** @addtogroup stm32f4xx_hal_driver
  * @{
  */
/** @defgroup dma dma
  * @Brief DMA HAL Driver Module
  * @{
  */
#ifdef HAL_DMA_MODULE_ENABLED
/* Частные типы
------------------------------------------------------------------------------*/
typedef struct
{
  __IO uint32_t ISR;   / *! <Регистр статуса прерывания DMA */
  __IO uint32_t Reserved0;
  __IO uint32_t IFCR;  / *! <DMA прерывание флаг прерывания очистить регистр */
} DMA_Base_Registers;
/* Приватные переменные ------------------------------------------------------*/
/* Частные константы ---------------------------------------------------------*/
/** @addtogroup dma_private_constants
 * @{
 */
 #define HAL_TIMEOUT_DMA_ABORT    5U  / * 5 мс */
/**
  * @}
  */
/* Приватные макросы ---------------------------------------------------------*//
/* Приватные функции
------------------------------------------------------------------------------*/
/** @addtogroup dma_private_functions
  * @{
  */
static void DMA_SetConfig(DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t
DstAddress, uint32_t DataLength);
static uint32_t DMA_CalcBaseAndBitshift(DMA_HandleTypeDef *hdma);
static HAL_StatusTypeDef DMA_CheckFifoParam(DMA_HandleTypeDef *hdma);
/**
  * @}
```

```
  */
/* Экспортируемые функции ---------------------------------------------------*/
/** @addtogroup dma_exported_functions
  * @{
  */
/** @addtogroup dma_exported_functions_group1
  *
@verbatim

===============================================================================
=================
             ##### Функции инициализации и де-инициализации #####

===============================================================================
=================
    [..]
    В этом разделе представлены функции, позволяющие инициализировать источник потока
DMA
    и адреса назначения, увеличение и размеры данных, направление передачи,
    Выбор круговой/нормальной режима, выбор режима памяти к памяти и значение
приоритета потока.
    [..]
    Функция hal_dma_init () следует за процедурами конфигурации DMA, как описано в
    справочное руководство.
@endverbatim
  * @{
  */
/**
  * @Brief инициализируйте DMA в соответствии с указанным
  * Параметры в DMA_INITTYPEDEFE и создайте связанную ручку.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMA_Init(DMA_HandleTypeDef *hdma)
{
  uint32_t tmp = 0U;
  uint32_t tickstart = HAL_GetTick();
  DMA_Base_Registers *regs;
  / * Проверьте периферическое состояние DMA */
```

```
  if(hdma == NULL)
  {
    return HAL_ERROR;
  }
  / * Проверьте параметры */
  assert_param(IS_DMA_STREAM_ALL_INSTANCE(hdma->Instance));
  assert_param(IS_DMA_CHANNEL(hdma->Init.Channel));
  assert_param(IS_DMA_DIRECTION(hdma->Init.Direction));
  assert_param(IS_DMA_PERIPHERAL_INC_STATE(hdma->Init.PeriphInc));
  assert_param(IS_DMA_MEMORY_INC_STATE(hdma->Init.MemInc));
  assert_param(IS_DMA_PERIPHERAL_DATA_SIZE(hdma->Init.PeriphDataAlignment));
  assert_param(IS_DMA_MEMORY_DATA_SIZE(hdma->Init.MemDataAlignment));
  assert_param(IS_DMA_MODE(hdma->Init.Mode));
  assert_param(IS_DMA_PRIORITY(hdma->Init.Priority));
  assert_param(IS_DMA_FIFO_MODE_STATE(hdma->Init.FIFOMode));
  /* Проверьте только всплеск памяти, только периферический взрыв и пороговые параметры
FIFO
     Когда режим FIFO включен */
  if(hdma->Init.FIFOMode != DMA_FIFOMODE_DISABLE)
  {
    assert_param(IS_DMA_FIFO_THRESHOLD(hdma->Init.FIFOThreshold));
    assert_param(IS_DMA_MEMORY_BURST(hdma->Init.MemBurst));
    assert_param(IS_DMA_PERIPHERAL_BURST(hdma->Init.PeriphBurst));
  }
  / * Изменить периферическое состояние DMA */
  hdma->State = HAL_DMA_STATE_BUSY;
  / * Выделите ресурс блокировки */
  __HAL_UNLOCK(hdma);
  / * Отключить периферическое */
  __HAL_DMA_DISABLE(hdma);
  / * Проверьте, фактически отключен поток DMA *//
  while((hdma->Instance->CR & DMA_SxCR_EN) != RESET)
  {
    / * Проверьте время ожидания */
    if((HAL_GetTick() - tickstart ) > HAL_TIMEOUT_DMA_ABORT)
    {
      / * Обновление кода ошибки */
      hdma->ErrorCode = HAL_DMA_ERROR_TIMEOUT;
      / * Измените состояние DMA */
      hdma->State = HAL_DMA_STATE_TIMEOUT;
```

```
      return HAL_TIMEOUT;
    }
  }
  / * Получить значение регистра CR */
  tmp = hdma->Instance->CR;
  / * Clear Chsel, Mburst, Pbbrst, PL, MSIZE, PSIZE, MINC, PINC, CIRC, DIR, CT и DBM
BITS */
  tmp &= ((uint32_t)~(DMA_SxCR_CHSEL | DMA_SxCR_MBURST | DMA_SxCR_PBURST | \
                      DMA_SxCR_PL    | DMA_SxCR_MSIZE  | DMA_SxCR_PSIZE  | \
                      DMA_SxCR_MINC  | DMA_SxCR_PINC   | DMA_SxCR_CIRC   | \
                      DMA_SxCR_DIR   | DMA_SxCR_CT     | DMA_SxCR_DBM));
  / * Подготовьте конфигурацию потока DMA */
  tmp |=  hdma->Init.Channel             | hdma->Init.Direction        |
          hdma->Init.PeriphInc           | hdma->Init.MemInc           |
          hdma->Init.PeriphDataAlignment | hdma->Init.MemDataAlignment |
          hdma->Init.Mode                | hdma->Init.Priority;
  / * Взрыв памяти и периферический взрыв не используется, когда FIFO отключено */
  if(hdma->Init.FIFOMode == DMA_FIFOMODE_ENABLE)
  {
    / * Получите всплеск памяти и периферический взрыв */
    tmp |=  hdma->Init.MemBurst | hdma->Init.PeriphBurst;
  }
  / * Записать в DMA Stream CR регистр */
  hdma->Instance->CR = tmp;
  / * Получить значение регистра FCR */
  tmp = hdma->Instance->FCR;
  / * Очистить прямой режим и пороговые биты FIFO */
  tmp &= (uint32_t)~(DMA_SxFCR_DMDIS | DMA_SxFCR_FTH);
  / * Подготовьте конфигурацию DMA Stream FIFO */
  tmp |= hdma->Init.FIFOMode;
  / * Порог FIFO не используется при отключении режима FIFO */
  if(hdma->Init.FIFOMode == DMA_FIFOMODE_ENABLE)
  {
    / * Получите порог FIFO */
    tmp |= hdma->Init.FIFOThreshold;
    / * Проверьте совместимость между пороговым уровнем FIFO и размером разрыва памяти
*/
    / * Для INGR4, INGR8, INGR16 BURSTS */
    if (hdma->Init.MemBurst != DMA_MBURST_SINGLE)
    {
```

```
      if (DMA_CheckFifoParam(hdma) != HAL_OK)
      {
        / * Обновление кода ошибки */
        hdma->ErrorCode = HAL_DMA_ERROR_PARAM;
        / * Измените состояние DMA */
        hdma->State = HAL_DMA_STATE_READY;
        return HAL_ERROR;
      }
    }
  }
  / * Напишите в DMA Stream fcr */
  hdma->Instance->FCR = tmp;
  /* Инициализировать параметры StreambaseadDress и Streamindex, которые будут
использоваться для расчета
     Паровой адрес DMA, необходимый HAL_DMA_IRQHandler () и HAL_DMA_POLLFORTRANSFER ()
*/
  regs = (DMA_Base_Registers *)DMA_CalcBaseAndBitshift(hdma);
  / * Очистить все флаги прерывания */
  regs->IFCR = 0x3FU << hdma->StreamIndex;
  / * Инициализировать код ошибки */
  hdma->ErrorCode = HAL_DMA_ERROR_NONE;
  / * Инициализировать состояние DMA */
  hdma->State = HAL_DMA_STATE_READY;
  return HAL_OK;
}
/**
  * @brief deinitialize периферийные устройства DMA
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMA_DeInit(DMA_HandleTypeDef *hdma)
{
  DMA_Base_Registers *regs;
  / * Проверьте периферическое состояние DMA */
  if(hdma == NULL)
  {
    return HAL_ERROR;
  }
  / * Проверьте периферическое состояние DMA */
```

```
  if(hdma->State == HAL_DMA_STATE_BUSY)
  {
    / * Статус ошибки возврата */
    return HAL_BUSY;
  }
  / * Проверьте параметры */
  assert_param(IS_DMA_STREAM_ALL_INSTANCE(hdma->Instance));
  / * Отключить выбранный DMA Streamx */
  __HAL_DMA_DISABLE(hdma);
  / * Сбросить DMA Streamx Control Register */
  hdma->Instance->CR   = 0U;
  / * Сбросить DMA Streamx Количество данных для передачи регистра */
  hdma->Instance->NDTR = 0U;
  / * Сбросить регистр периферийных адресов DMA -потока */
  hdma->Instance->PAR  = 0U;
  / * Сбросить DMA Streamx Memory 0 Регистр адреса */
  hdma->Instance->M0AR = 0U;
  / * Сбросить DMA Streamx Memory 1 Регистр адреса */
  hdma->Instance->M1AR = 0U;
  / * Сбросить DMA Streamx FIFO Control Register */
  hdma->Instance->FCR  = 0x00000021U;
  / * Получите базовый адрес DMA *//
  regs = (DMA_Base_Registers *)DMA_CalcBaseAndBitshift(hdma);
  / * Очистите все обратные вызовы */
  hdma->XferCpltCallback = NULL;
  hdma->XferHalfCpltCallback = NULL;
  hdma->XferM1CpltCallback = NULL;
  hdma->XferM1HalfCpltCallback = NULL;
  hdma->XferErrorCallback = NULL;
  hdma->XferAbortCallback = NULL;
  / * Очистить все флаги прерывания при правильном смещении в регистре */
  regs->IFCR = 0x3FU << hdma->StreamIndex;
  / * Сбросить код ошибки */
  hdma->ErrorCode = HAL_DMA_ERROR_NONE;
  / * Сбросить состояние DMA */
  hdma->State = HAL_DMA_STATE_RESET;
  / * Выпуск блокировки */
  __HAL_UNLOCK(hdma);
  return HAL_OK;
}
```

```
/**
  * @}
  */
/** @addtogroup dma_exported_functions_group2
  *
@verbatim

===============================================================================
=================
                      ##### функции работы #####

===============================================================================
=================
    [..] В этом разделе представлены функции, позволяющие:
      (+) Настройте источник, адрес назначения и длину данных и запуск DMA Transfer
      (+) Настройте источник, адрес назначения и длину данных и
          Начать передачу DMA с прерыванием
      (+) Прервать передачу DMA
      (+) Опрос для завершения передачи
      (+) Обработка запроса прерывания DMA
@endverbatim
  * @{
  */
/**
  * @brief начинает передачу DMA.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param srcaddress Адрес буфера памяти источника
  * @param dstaddress Адрес буфера памяти назначения
  * @param DataLength продолжительность передачи данных из источника в назначение
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMA_Start(DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t
DstAddress, uint32_t DataLength)
{
  HAL_StatusTypeDef status = HAL_OK;
  / * Проверьте параметры */
  assert_param(IS_DMA_BUFFER_SIZE(DataLength));
  / * Процесс заблокирован */
  __HAL_LOCK(hdma);
```

```
  if(HAL_DMA_STATE_READY == hdma->State)
  {
    / * Изменить периферическое состояние DMA */
    hdma->State = HAL_DMA_STATE_BUSY;
    / * Инициализировать код ошибки */
    hdma->ErrorCode = HAL_DMA_ERROR_NONE;
    / * Настройте источник, адрес назначения и длину данных */
    DMA_SetConfig(hdma, SrcAddress, DstAddress, DataLength);
    / * Включить периферическое */
    __HAL_DMA_ENABLE(hdma);
  }
  else
  {
    / * Процесс разблокирован */
    __HAL_UNLOCK(hdma);
    / * Статус ошибки возврата */
    status = HAL_BUSY;
  }
  return status;
}
/**
  * @brief запустить передачу DMA с включенным прерыванием.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param srcaddress Адрес буфера памяти источника
  * @param dstaddress Адрес буфера памяти назначения
  * @param DataLength продолжительность передачи данных из источника в назначение
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMA_Start_IT(DMA_HandleTypeDef *hdma, uint32_t SrcAddress,
uint32_t DstAddress, uint32_t DataLength)
{
  HAL_StatusTypeDef status = HAL_OK;
  / * Рассчитайте базовый и номер потока DMA */
  DMA_Base_Registers *regs = (DMA_Base_Registers *)hdma->StreamBaseAddress;
  / * Проверьте параметры */
  assert_param(IS_DMA_BUFFER_SIZE(DataLength));
  / * Процесс заблокирован */
  __HAL_LOCK(hdma);
  if(HAL_DMA_STATE_READY == hdma->State)
```

```
  {
    / * Изменить периферическое состояние DMA */
    hdma->State = HAL_DMA_STATE_BUSY;
    / * Инициализировать код ошибки */
    hdma->ErrorCode = HAL_DMA_ERROR_NONE;
    / * Настройте источник, адрес назначения и длину данных */
    DMA_SetConfig(hdma, SrcAddress, DstAddress, DataLength);
    / * Очистить все флаги прерывания при правильном смещении в регистре */
    regs->IFCR = 0x3FU << hdma->StreamIndex;
    /* Включить общие прерывания*/
    hdma->Instance->CR  |= DMA_IT_TC | DMA_IT_TE | DMA_IT_DME;
    if(hdma->XferHalfCpltCallback != NULL)
    {
      hdma->Instance->CR  |= DMA_IT_HT;
    }
    / * Включить периферическое */
    __HAL_DMA_ENABLE(hdma);
  }
  else
  {
    / * Процесс разблокирован */
    __HAL_UNLOCK(hdma);
    / * Статус ошибки возврата */
    status = HAL_BUSY;
  }
  return status;
}
/**
  * @brief прерывает передачу DMA.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  *
  * @note после отключения потока DMA, проверка для ожидания, пока поток DMA
  * Эффективно отключен добавлен.Если поток отключен
  * Хотя передача данных продолжается, текущие данные будут переданы
  * и поток будет эффективно отключен только после передачи
  * Эти единственные данные завершены.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMA_Abort(DMA_HandleTypeDef *hdma)
```

```
{
  / * Рассчитайте базовый и номер потока DMA */
  DMA_Base_Registers *regs = (DMA_Base_Registers *)hdma->StreamBaseAddress;
  uint32_t tickstart = HAL_GetTick();
  if(hdma->State != HAL_DMA_STATE_BUSY)
  {
    hdma->ErrorCode = HAL_DMA_ERROR_NO_XFER;
    / * Процесс разблокирован */
    __HAL_UNLOCK(hdma);
    return HAL_ERROR;
  }
  else
  {
    / * Отключить все прерывания передачи */
    hdma->Instance->CR  &= ~(DMA_IT_TC | DMA_IT_TE | DMA_IT_DME);
    hdma->Instance->FCR &= ~(DMA_IT_FE);
    if((hdma->XferHalfCpltCallback != NULL) || (hdma->XferM1HalfCpltCallback != NULL))
    {
      hdma->Instance->CR  &= ~(DMA_IT_HT);
    }
    / * Отключить поток */
    __HAL_DMA_DISABLE(hdma);
    / * Проверьте, фактически отключен поток DMA *//
    while((hdma->Instance->CR & DMA_SxCR_EN) != RESET)
    {
      / * Проверьте время ожидания */
      if((HAL_GetTick() - tickstart ) > HAL_TIMEOUT_DMA_ABORT)
      {
        / * Обновление кода ошибки */
        hdma->ErrorCode = HAL_DMA_ERROR_TIMEOUT;
        / * Измените состояние DMA */
        hdma->State = HAL_DMA_STATE_TIMEOUT;
        / * Процесс разблокирован */
        __HAL_UNLOCK(hdma);
        return HAL_TIMEOUT;
      }
    }
    / * Очистить все флаги прерывания при правильном смещении в регистре */
    regs->IFCR = 0x3FU << hdma->StreamIndex;
    /* Измените состояние DMA*/
```

```
    hdma->State = HAL_DMA_STATE_READY;
    / * Процесс разблокирован */
    __HAL_UNLOCK(hdma);
  }
  return HAL_OK;
}
/**
  * @Brief прерывает передачу DMA в режиме прерывания.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMA_Abort_IT(DMA_HandleTypeDef *hdma)
{
  if(hdma->State != HAL_DMA_STATE_BUSY)
  {
    hdma->ErrorCode = HAL_DMA_ERROR_NO_XFER;
    return HAL_ERROR;
  }
  else
  {
    / * Установите состояние прерванного *//
    hdma->State = HAL_DMA_STATE_ABORT;
    / * Отключить поток */
    __HAL_DMA_DISABLE(hdma);
  }
  return HAL_OK;
}
/**
  * @brief Offuls для перевода завершена.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param expleelvel Указывает, что уровень DMA завершен.
  * @note Режим опроса хранится в этой версии для Legacy.Вместо этого рекомендуется
использовать ИТ -модель.
  * Эта модель может быть использована для отладки.
  * @Note API API HAL_DMA_POLLFORTRANSFER не может использоваться в режиме круглой и
двойной буферизации (автоматический круглый режим).
  * @param Timeout Timeout.
  * @retval hal status
```

```
  */
HAL_StatusTypeDef HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma,
HAL_DMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout)
{
  HAL_StatusTypeDef status = HAL_OK;
  uint32_t mask_cpltlevel;
  uint32_t tickstart = HAL_GetTick();
  uint32_t tmpisr;
  / * Рассчитайте базовый и номер потока DMA */
  DMA_Base_Registers *regs;
  if(HAL_DMA_STATE_BUSY != hdma->State)
  {
    / * НЕТ ПЕРЕВОДА.
    hdma->ErrorCode = HAL_DMA_ERROR_NO_XFER;
    __HAL_UNLOCK(hdma);
    return HAL_ERROR;
  }
  / * Режим опроса не поддерживается в круглом режиме и двойном режиме буферизации */
  if ((hdma->Instance->CR & DMA_SxCR_CIRC) != RESET)
  {
    hdma->ErrorCode = HAL_DMA_ERROR_NOT_SUPPORTED;
    return HAL_ERROR;
  }
  / * Получите полный флаг переноса уровня */
  if(CompleteLevel == HAL_DMA_FULL_TRANSFER)
  {
    / * Передача полный флаг */
    mask_cpltlevel = DMA_FLAG_TCIF0_4 << hdma->StreamIndex;
  }
  else
  {
    / * Половина передачи полного флага */
    mask_cpltlevel = DMA_FLAG_HTIF0_4 << hdma->StreamIndex;
  }
  regs = (DMA_Base_Registers *)hdma->StreamBaseAddress;
  tmpisr = regs->ISR;
  while(((tmpisr & mask_cpltlevel) == RESET) && ((hdma->ErrorCode & HAL_DMA_ERROR_TE)
== RESET))
  {
    /* Проверьте время ожидания (не применимо в круговом режиме)*/
```

```
  if(Timeout != HAL_MAX_DELAY)
  {
    if((Timeout == 0U)||((HAL_GetTick() - tickstart ) > Timeout))
    {
      / * Обновление кода ошибки */
      hdma->ErrorCode = HAL_DMA_ERROR_TIMEOUT;
      / * Измените состояние DMA */
      hdma->State = HAL_DMA_STATE_READY;
      / * Процесс разблокирован */
      __HAL_UNLOCK(hdma);
      return HAL_TIMEOUT;
    }
  }
  / * Получить значение регистра ISR */
  tmpisr = regs->ISR;
  if((tmpisr & (DMA_FLAG_TEIF0_4 << hdma->StreamIndex)) != RESET)
  {
    / * Обновление кода ошибки */
    hdma->ErrorCode |= HAL_DMA_ERROR_TE;
    / * Очистить флаг ошибки передачи */
    regs->IFCR = DMA_FLAG_TEIF0_4 << hdma->StreamIndex;
  }
  if((tmpisr & (DMA_FLAG_FEIF0_4 << hdma->StreamIndex)) != RESET)
  {
    / * Обновление кода ошибки */
    hdma->ErrorCode |= HAL_DMA_ERROR_FE;
    / * Очистить флаг ошибки FIFO */
    regs->IFCR = DMA_FLAG_FEIF0_4 << hdma->StreamIndex;
  }
  if((tmpisr & (DMA_FLAG_DMEIF0_4 << hdma->StreamIndex)) != RESET)
  {
    / * Обновление кода ошибки */
    hdma->ErrorCode |= HAL_DMA_ERROR_DME;
    / * Очистить флаг ошибки прямого режима */
    regs->IFCR = DMA_FLAG_DMEIF0_4 << hdma->StreamIndex;
  }
}
if(hdma->ErrorCode != HAL_DMA_ERROR_NONE)
{
  if((hdma->ErrorCode & HAL_DMA_ERROR_TE) != RESET)
```

```
    {
      HAL_DMA_Abort(hdma);
      / * Очистить половину передачи и переносить полные флаги */
      regs->IFCR = (DMA_FLAG_HTIF0_4 | DMA_FLAG_TCIF0_4) << hdma->StreamIndex;
      / * Измените состояние DMA */
      hdma->State= HAL_DMA_STATE_READY;
      / * Процесс разблокирован */
      __HAL_UNLOCK(hdma);
      return HAL_ERROR;
   }
  }
  / * Получите полный флаг переноса уровня */
  if(CompleteLevel == HAL_DMA_FULL_TRANSFER)
  {
    / * Очистить половину передачи и переносить полные флаги */
    regs->IFCR = (DMA_FLAG_HTIF0_4 | DMA_FLAG_TCIF0_4) << hdma->StreamIndex;
    hdma->State = HAL_DMA_STATE_READY;
    / * Процесс разблокирован */
    __HAL_UNLOCK(hdma);
  }
  else
  {
    / * Очистить половину передачи и переносить полные флаги */
    regs->IFCR = (DMA_FLAG_HTIF0_4) << hdma->StreamIndex;
  }
  return status;
}
/**
  * @Brief обрабатывает запрос прерывания DMA.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @retval нет
  */
void HAL_DMA_IRQHandler(DMA_HandleTypeDef *hdma)
{
  uint32_t tmpisr;
  __IO uint32_t count = 0U;
  uint32_t timeout = SystemCoreClock / 9600U;
  / * Рассчитайте базовый и номер потока DMA */
  DMA_Base_Registers *regs = (DMA_Base_Registers *)hdma->StreamBaseAddress;
```

```
tmpisr = regs->ISR;
/*Управление ошибкой трансфертов ******************************
if ((tmpisr & (DMA_FLAG_TEIF0_4 << hdma->StreamIndex)) != RESET)
{
  if(__HAL_DMA_GET_IT_SOURCE(hdma, DMA_IT_TE) != RESET)
  {
    / * Отключить прерывание ошибки передачи */
    hdma->Instance->CR  &= ~(DMA_IT_TE);
    / * Очистить флаг ошибки передачи */
    regs->IFCR = DMA_FLAG_TEIF0_4 << hdma->StreamIndex;
    / * Обновление кода ошибки */
    hdma->ErrorCode |= HAL_DMA_ERROR_TE;
  }
}
/*FIFO ошибка управления прерыванием ***********************************/
if ((tmpisr & (DMA_FLAG_FEIF0_4 << hdma->StreamIndex)) != RESET)
{
  if(__HAL_DMA_GET_IT_SOURCE(hdma, DMA_IT_FE) != RESET)
  {
    / * Очистить флаг ошибки FIFO */
    regs->IFCR = DMA_FLAG_FEIF0_4 << hdma->StreamIndex;
    / * Обновление кода ошибки */
    hdma->ErrorCode |= HAL_DMA_ERROR_FE;
  }
}
/*Управление ошибкой прямого режима ****************************/
if ((tmpisr & (DMA_FLAG_DMEIF0_4 << hdma->StreamIndex)) != RESET)
{
  if(__HAL_DMA_GET_IT_SOURCE(hdma, DMA_IT_DME) != RESET)
  {
    / * Очистить флаг ошибки прямого режима */
    regs->IFCR = DMA_FLAG_DMEIF0_4 << hdma->StreamIndex;
    / * Обновление кода ошибки */
    hdma->ErrorCode |= HAL_DMA_ERROR_DME;
  }
}
/*Наполовину передача полное управление прерыванием *************************/
if ((tmpisr & (DMA_FLAG_HTIF0_4 << hdma->StreamIndex)) != RESET)
{
  if(__HAL_DMA_GET_IT_SOURCE(hdma, DMA_IT_HT) != RESET)
```

```
    {
      / * Очистить половину переноса полного флага */
      regs->IFCR = DMA_FLAG_HTIF0_4 << hdma->StreamIndex;
      / * Режим Multi_buffering включен *//
      if(((hdma->Instance->CR) & (uint32_t)(DMA_SxCR_DBM)) != RESET)
      {
        / * Используется текущий буфер памяти - память 0 */
        if((hdma->Instance->CR & DMA_SxCR_CT) == RESET)
        {
          if(hdma->XferHalfCpltCallback != NULL)
          {
            / * Половина передачи обратного вызова */
            hdma->XferHalfCpltCallback(hdma);
          }
        }
        / * Используется текущий буфер памяти - память 1 */
        else
        {
          if(hdma->XferM1HalfCpltCallback != NULL)
          {
            / * Половина передачи обратного вызова */
            hdma->XferM1HalfCpltCallback(hdma);
          }
        }
      }
      else
      {
        / * Отключить наполовину переносное прерывание, если режим DMA не является
круглым */
        if((hdma->Instance->CR & DMA_SxCR_CIRC) == RESET)
        {
          / * Отключить наполовину переносное прерывание */
          hdma->Instance->CR  &= ~(DMA_IT_HT);
        }
        if(hdma->XferHalfCpltCallback != NULL)
        {
          / * Половина передачи обратного вызова */
          hdma->XferHalfCpltCallback(hdma);
        }
      }
```

```
    }
  }
  /*Перевод полного управления прерыванием ****************************/
  if ((tmpisr & (DMA_FLAG_TCIF0_4 << hdma->StreamIndex)) != RESET)
  {
    if(__HAL_DMA_GET_IT_SOURCE(hdma, DMA_IT_TC) != RESET)
    {
      / * Очистить перевод полный флаг */
      regs->IFCR = DMA_FLAG_TCIF0_4 << hdma->StreamIndex;
      if(HAL_DMA_STATE_ABORT == hdma->State)
      {
        / * Отключить все прерывания передачи */
        hdma->Instance->CR  &= ~(DMA_IT_TC | DMA_IT_TE | DMA_IT_DME);
        hdma->Instance->FCR &= ~(DMA_IT_FE);
        if((hdma->XferHalfCpltCallback != NULL) || (hdma->XferM1HalfCpltCallback !=
NULL))
        {
          hdma->Instance->CR  &= ~(DMA_IT_HT);
        }
        / * Очистить все флаги прерывания при правильном смещении в регистре */
        regs->IFCR = 0x3FU << hdma->StreamIndex;
        / * Измените состояние DMA */
        hdma->State = HAL_DMA_STATE_READY;
        / * Процесс разблокирован */
        __HAL_UNLOCK(hdma);
        if(hdma->XferAbortCallback != NULL)
        {
          hdma->XferAbortCallback(hdma);
        }
        return;
      }
      if(((hdma->Instance->CR) & (uint32_t)(DMA_SxCR_DBM)) != RESET)
      {
        / * Используется текущий буфер памяти - память 0 */
        if((hdma->Instance->CR & DMA_SxCR_CT) == RESET)
        {
          if(hdma->XferM1CpltCallback != NULL)
          {
            / * Перенос полный обратный вызов для Memory1 */
            hdma->XferM1CpltCallback(hdma);
```

```
        }
      }
      / * Используется текущий буфер памяти - память 1 */
      else
      {
        if(hdma->XferCpltCallback != NULL)
        {
          / * Перенос полный обратный вызов для Memory0 */
          hdma->XferCpltCallback(hdma);
        }
      }
    }
    / * Отключить перевод полного прерывания, если режим DMA не является круглым */
    else
    {
      if((hdma->Instance->CR & DMA_SxCR_CIRC) == RESET)
      {
        / * Отключить перевод полного прерывания */
        hdma->Instance->CR  &= ~(DMA_IT_TC);
        / * Измените состояние DMA */
        hdma->State = HAL_DMA_STATE_READY;
        / * Процесс разблокирован */
        __HAL_UNLOCK(hdma);
      }
      if(hdma->XferCpltCallback != NULL)
      {
        / * Перенос полный обратный вызов */
        hdma->XferCpltCallback(hdma);
      }
    }
  }
}
/ * Управление случаем ошибки */
if(hdma->ErrorCode != HAL_DMA_ERROR_NONE)
{
  if((hdma->ErrorCode & HAL_DMA_ERROR_TE) != RESET)
  {
    hdma->State = HAL_DMA_STATE_ABORT;
    / * Отключить поток */
    __HAL_DMA_DISABLE(hdma);
```

```
      do
      {
        if (++count > timeout)
        {
          break;
        }
      }
      while((hdma->Instance->CR & DMA_SxCR_EN) != RESET);
      / * Измените состояние DMA */
      hdma->State = HAL_DMA_STATE_READY;
      / * Процесс разблокирован */
      __HAL_UNLOCK(hdma);
    }
    if(hdma->XferErrorCallback != NULL)
    {
      / * Трансферный ошибка обратный вызов */
      hdma->XferErrorCallback(hdma);
    }
  }
}
/**
  * @Brief Регистрация обратных вызовов
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param callbackid идентификатор обратного вызова пользователя
  * Структура DMA_HANDLEYPEDEF в качестве параметра.
  * @param pcallback pointer на частную функцию обратного вызова, которая имеет
указатель на
  * Структура DMA_HANDLEYPEDEF в качестве параметра.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMA_RegisterCallback(DMA_HandleTypeDef *hdma,
HAL_DMA_CallbackIDTypeDef CallbackID, void (* pCallback)(DMA_HandleTypeDef *_hdma))
{
  HAL_StatusTypeDef status = HAL_OK;
  / * Процесс заблокирован */
  __HAL_LOCK(hdma);
  if(HAL_DMA_STATE_READY == hdma->State)
  {
    switch (CallbackID)
```

```
    {
    case  HAL_DMA_XFER_CPLT_CB_ID:
      hdma->XferCpltCallback = pCallback;
      break;
    case  HAL_DMA_XFER_HALFCPLT_CB_ID:
      hdma->XferHalfCpltCallback = pCallback;
      break;
    case  HAL_DMA_XFER_M1CPLT_CB_ID:
      hdma->XferM1CpltCallback = pCallback;
      break;
    case  HAL_DMA_XFER_M1HALFCPLT_CB_ID:
      hdma->XferM1HalfCpltCallback = pCallback;
      break;
    case  HAL_DMA_XFER_ERROR_CB_ID:
      hdma->XferErrorCallback = pCallback;
      break;
    case  HAL_DMA_XFER_ABORT_CB_ID:
      hdma->XferAbortCallback = pCallback;
      break;
    default:
      / * Статус ошибки возврата */
      status =  HAL_ERROR;
      break;
    }
  }
  else
  {
    / * Статус ошибки возврата */
    status =  HAL_ERROR;
  }
  / * Выпуск блокировки */
  __HAL_UNLOCK(hdma);
  return status;
}
/**
  * @brief unregister обратные вызовы
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param callbackid идентификатор обратного вызова пользователя
  * a hal_dma_callbackidtypedef enum в качестве параметра.
```

```
  * @retval hal status
  */
HAL_StatusTypeDef HAL_DMA_UnRegisterCallback(DMA_HandleTypeDef *hdma,
HAL_DMA_CallbackIDTypeDef CallbackID)
{
  HAL_StatusTypeDef status = HAL_OK;
  / * Процесс заблокирован */
  __HAL_LOCK(hdma);
  if(HAL_DMA_STATE_READY == hdma->State)
  {
    switch (CallbackID)
    {
    case  HAL_DMA_XFER_CPLT_CB_ID:
      hdma->XferCpltCallback = NULL;
      break;
    case  HAL_DMA_XFER_HALFCPLT_CB_ID:
      hdma->XferHalfCpltCallback = NULL;
      break;
    case  HAL_DMA_XFER_M1CPLT_CB_ID:
      hdma->XferM1CpltCallback = NULL;
      break;
    case  HAL_DMA_XFER_M1HALFCPLT_CB_ID:
      hdma->XferM1HalfCpltCallback = NULL;
      break;
    case  HAL_DMA_XFER_ERROR_CB_ID:
      hdma->XferErrorCallback = NULL;
      break;
    case  HAL_DMA_XFER_ABORT_CB_ID:
      hdma->XferAbortCallback = NULL;
      break;
    case   HAL_DMA_XFER_ALL_CB_ID:
      hdma->XferCpltCallback = NULL;
      hdma->XferHalfCpltCallback = NULL;
      hdma->XferM1CpltCallback = NULL;
      hdma->XferM1HalfCpltCallback = NULL;
      hdma->XferErrorCallback = NULL;
      hdma->XferAbortCallback = NULL;
      break;
    default:
      status = HAL_ERROR;
```

```
      break;
    }
  }
  else
  {
    status = HAL_ERROR;
  }
  / * Выпуск блокировки */
  __HAL_UNLOCK(hdma);
  return status;
}
/**
  * @}
  */
/** @addtogroup dma_exported_functions_group3
  *
@verbatim

==============================================================================
=================
                    ##### Функции состояния и ошибок #####

==============================================================================
=================
    [..]
    Этот подраздел предоставляет функции, позволяющие
      (+) Проверьте состояние DMA
      (+) Получить код ошибки
@endverbatim
  * @{
  */
/**
  * @brief возвращает состояние DMA.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @retval Hal State
  */
HAL_DMA_StateTypeDef HAL_DMA_GetState(DMA_HandleTypeDef *hdma)
{
  return hdma->State;
```

```
}
/**
  * @brief вернуть код ошибки DMA
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @retval код ошибки DMA
  */
uint32_t HAL_DMA_GetError(DMA_HandleTypeDef *hdma)
{
  return hdma->ErrorCode;
}
/**
  * @}
  */
/**
  * @}
  */
/** @addtogroup dma_private_functions
  * @{
  */
/**
  * @brief устанавливает параметр передачи DMA.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @param srcaddress Адрес буфера памяти источника
  * @param dstaddress Адрес буфера памяти назначения
  * @param DataLength продолжительность передачи данных из источника в назначение
  * @retval hal status
  */
static void DMA_SetConfig(DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t
DstAddress, uint32_t DataLength)
{
  / * Очистить бит DBM */
  hdma->Instance->CR &= (uint32_t)(~DMA_SxCR_DBM);
  / * Настройка длины данных потока DMA */
  hdma->Instance->NDTR = DataLength;
  / * Память к периферическому */
  if((hdma->Init.Direction) == DMA_MEMORY_TO_PERIPH)
  {
    / * Настроить адрес назначения потока DMA */
```

```
    hdma->Instance->PAR = DstAddress;
    / * Настройка адреса источника потока DMA */
    hdma->Instance->M0AR = SrcAddress;
  }
  / * Периферийно для памяти */
  else
  {
    / * Настройка адреса источника потока DMA */
    hdma->Instance->PAR = SrcAddress;
    / * Настроить адрес назначения потока DMA */
    hdma->Instance->M0AR = DstAddress;
  }
}
/**
  * @brief возвращает базовый адрес потока DMA в зависимости от номера потока
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @retval Stream базовый адрес базового адреса
  */
static uint32_t DMA_CalcBaseAndBitshift(DMA_HandleTypeDef *hdma)
{
  uint32_t stream_number = (((uint32_t)hdma->Instance & 0xFFU) - 16U) / 24U;
  / * Поиск таблицы для необходимого бит -смещения флагов в регистрах статуса */
  static const uint8_t flagBitshiftOffset[8U] = {0U, 6U, 16U, 22U, 0U, 6U, 16U, 22U};
  hdma->StreamIndex = flagBitshiftOffset[stream_number];
  if (stream_number > 3U)
  {
    / * Вернуть указатель на HISR и hifcr */
    hdma->StreamBaseAddress = (((uint32_t)hdma->Instance & (uint32_t)(~0x3FFU)) + 4U);
  }
  else
  {
    / * Вернуть указатель на список и жизнь */
    hdma->StreamBaseAddress = ((uint32_t)hdma->Instance & (uint32_t)(~0x3FFU));
  }
  return hdma->StreamBaseAddress;
}
/**
  * @brief Проверка совместимость между пороговым уровнем FIFO и размером разрыва
памяти
```

```
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного потока DMA.
  * @retval hal status
  */
static HAL_StatusTypeDef DMA_CheckFifoParam(DMA_HandleTypeDef *hdma)
{
  HAL_StatusTypeDef status = HAL_OK;
  uint32_t tmp = hdma->Init.FIFOThreshold;
  / * Размер данных памяти равен байту */
  if(hdma->Init.MemDataAlignment == DMA_MDATAALIGN_BYTE)
  {
    switch (tmp)
    {
    case DMA_FIFO_THRESHOLD_1QUARTERFULL:
    case DMA_FIFO_THRESHOLD_3QUARTERSFULL:
      if ((hdma->Init.MemBurst & DMA_SxCR_MBURST_1) == DMA_SxCR_MBURST_1)
      {
        status = HAL_ERROR;
      }
      break;
    case DMA_FIFO_THRESHOLD_HALFFULL:
      if (hdma->Init.MemBurst == DMA_MBURST_INC16)
      {
        status = HAL_ERROR;
      }
      break;
    case DMA_FIFO_THRESHOLD_FULL:
      break;
    default:
      break;
    }
  }
  / * Размер данных памяти, равный полуослужам */
  else if (hdma->Init.MemDataAlignment == DMA_MDATAALIGN_HALFWORD)
  {
    switch (tmp)
    {
    case DMA_FIFO_THRESHOLD_1QUARTERFULL:
    case DMA_FIFO_THRESHOLD_3QUARTERSFULL:
      status = HAL_ERROR;
```

```
      break;
    case DMA_FIFO_THRESHOLD_HALFFULL:
      if ((hdma->Init.MemBurst & DMA_SxCR_MBURST_1) == DMA_SxCR_MBURST_1)
      {
        status = HAL_ERROR;
      }
      break;
    case DMA_FIFO_THRESHOLD_FULL:
      if (hdma->Init.MemBurst == DMA_MBURST_INC16)
      {
        status = HAL_ERROR;
      }
      break;
    default:
      break;
    }
  }
  / * Размер данных памяти равен слову */
  else
  {
    switch (tmp)
    {
    case DMA_FIFO_THRESHOLD_1QUARTERFULL:
    case DMA_FIFO_THRESHOLD_HALFFULL:
    case DMA_FIFO_THRESHOLD_3QUARTERSFULL:
      status = HAL_ERROR;
      break;
    case DMA_FIFO_THRESHOLD_FULL:
      if ((hdma->Init.MemBurst & DMA_SxCR_MBURST_1) == DMA_SxCR_MBURST_1)
      {
        status = HAL_ERROR;
      }
      break;
    default:
      break;
    }
  }
  return status;
}
/**
```

```
  * @}
  */
#endif / * Hal_dma_module_enabled */
/**
  * @}
  */
/**
  * @}
  */
```

Приложение В

Описание файла stm32f4xx_it.c

```
/ * Код пользователя начинается за заголовок */
/**
******************************************************************************
***
  * @file STM32F4XX_IT.C
  * @brief прерывать процедуры службы.

******************************************************************************
***
  * @внимание
  *
  * Copyright (C) 2024 Stmicroelectronics.
  * Все права защищены.
  *
  * Это программное обеспечение лицензировано в соответствии с условиями, которые можно
найти в файле лицензии
  * В корневом каталоге этого программного компонента.
  * Если нет лицензионного файла с этим программным обеспечением, оно предоставляется
как есть.
  *
******************************************************************************
***
  */
/ * Заголовок кода пользователя */
/* Включает ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_it.h"
/* Private включает в себя
----------------------------------------------------------------------------*/
/ * Код пользователя начинается в включении */
/ * Конец кода пользователя включает */
/* Private Typedef -----------------------------------------------------------*/
/ * Код пользователя начинается TD */
/ * Код пользователя END TD */
```

```
/* Private Define
-------------------------------------------------------------------------*//
/ * Код пользователя начинается PD */
/ * Код пользователя END PD */
/* Частный макрос ---------------------------------------------------------*/
/ * Код пользователя начинается PM */
/ * Код пользователя END PM */
/* Приватные переменные ---------------------------------------------------------*/
/ * Код пользователя начинается PV */
/ * Код пользователя END PV */
/* Прототипы частной функции ----------------------------------------------*/
/ * Код пользователя начинается PFP */
/ * Код пользователя END PFP */
/* Частный код пользователя
------------------------------------------------------------*/
/ * Код пользователя начинается 0 */
/ * Код пользователя конец 0 */
/* Внешние переменные ---------------------------------------------------------*/
extern DMA_HandleTypeDef hdma_usart1_rx;
extern DMA_HandleTypeDef hdma_usart1_tx;
/ * Код пользователя начинается EV */
/ * Код пользователя END EV */
/*****************************************************************************/
/ * Cortex-M4 прерывание процессора и обработчики исключений */
/*****************************************************************************/
/**
  * @brief Эта функция обрабатывает не маскируемое прерывание.
  */
void NMI_Handler(void)
{
  / * Код пользователя начинается unmakableint_irqn 0 */
  / * Код пользователя END NONASKABLEINT_IRQN 0 */
  / * Код пользователя начинается unmakableint_irqn 1 */
  while (1)
  {
  }
  / * Код пользователя END NONASKABLEINT_IRQN 1 */
}
/**
  * @brief Эта функция обрабатывает сильное нарушение разлома.
```

```
  */
void HardFault_Handler(void)
{
  / * Пользовательский код начинается hardfault_irqn 0 */
  / * Код пользователя End Hardfault_irqn 0 */
  while (1)
  {
    / * Код пользователя начинается w1_hardfault_irqn 0 */
    / * Код пользователя END W1_HARDFAULT_IRQN 0 */
  }
}
/**
  * @brief Эта функция обрабатывает ошибку управления памятью.
  */
void MemManage_Handler(void)
{
  / * Пользовательский код начинает memorymanagement_irqn 0 */
  / * Код пользователя END MEMEMANGAMENT_IRQN 0 */
  while (1)
  {
    / * Код пользователя начинается w1_memorymanagement_irqn 0 */
    / * Код пользователя End W1_MemoryManagement_IRQN 0 */
  }
}
/**
  * @brief Эта функция обрабатывает ошибку перед избранной, ошибкой доступа к памяти.
  */
void BusFault_Handler(void)
{
  / * Пользовательский код begin busfault_irqn 0 */
  / * Код пользователя END BUSFAULT_IRQN 0 */
  while (1)
  {
    / * Код пользователя начинается w1_busfault_irqn 0 */
    / * Код пользователя END W1_BUSFAULT_IRQN 0 */
  }
}
/**
  * @brief Эта функция обрабатывает неопределенную инструкцию или незаконное состояние.
  */
```

```
void UsageFault_Handler(void)
{
  / * Код пользователя начинает usagefault_irqn 0 */
  / * Код пользователя END usagefault_irqn 0 */
  while (1)
  {
    / * Код пользователя начинается W1_USAGEFAULT_IRQN 0 */
    / * Код пользователя END W1_USAGEFAULT_IRQN 0 */
  }
}
/**
  * @brief Эта функция обрабатывает системный сервис Service Service через инструкцию
SWI.
  */
void SVC_Handler(void)
{
  / * Код пользователя начинается svcall_irqn 0 */
  / * Код пользователя END SVCALL_IRQN 0 */
  / * Код пользователя начинается svcall_irqn 1 */
  / * Код пользователя END SVCALL_IRQN 1 */
}
/**
  * @brief Эта функция обрабатывает монитор отладки.
  */
void DebugMon_Handler(void)
{
  / * Пользовательский код начинает debugmonitor_irqn 0 */
  / * Код пользователя END DEBUGMONITOR_IRQN 0 */
  / * Код пользователя начинает Debugmonitor_irqn 1 */
  / * Код пользователя END DEBUGMONITOR_IRQN 1 */
}
/**
  * @brief Эта функция обрабатывает запрашиваемый запрос на системную службу.
  */
void PendSV_Handler(void)
{
  / * Код пользователя начинается pendsv_irqn 0 */
  / * Код пользователя End pendsv_irqn 0 */
  / * Код пользователя начинается pendsv_irqn 1 */
  / * Код пользователя End pendsv_irqn 1 */
```

```
}
/**
  * @brief Эта функция обрабатывает системный таймер.
  */
void SysTick_Handler(void)
{
  / * Код пользователя начинается SYSTICK_IRQN 0 */
  / * Код пользователя END SYSTICK_IRQN 0 */
  HAL_IncTick();
  / * Код пользователя начинается SYSTICK_IRQN 1 */
  / * Код пользователя END SYSTICK_IRQN 1 */
}
/******************************************************************************/
/ * STM32F4XX Периферийные обработчики прерываний */
/* Добавьте здесь обработчики прерываний для используемых периферийных устройств.*/
/ * Для доступных имен обработчиков периферического прерывания, */
/* Пожалуйста, обратитесь к файлу запуска (startup_stm32f4xx.s).*/
/******************************************************************************/
/**
  * @brief Эта функция обрабатывает DMA2 Stream2 Global прерывание.
  */
void DMA2_Stream2_IRQHandler(void)
{
  / * Код пользователя начинается dma2_stream2_irqn 0 */
  / * Код пользователя END DMA2_STREAM2_IRQN 0 */
  HAL_DMA_IRQHandler(&hdma_usart1_rx);
  / * Код пользователя начинается dma2_stream2_irqn 1 */
  / * Код пользователя END DMA2_STREAM2_IRQN 1 */
}
/**
  * @brief Эта функция обрабатывает DMA2 Stream7 Global прерывание.
  */
void DMA2_Stream7_IRQHandler(void)
{
  / * Код пользователя начинается dma2_stream7_irqn 0 */
  / * Код пользователя END DMA2_STREAM7_IRQN 0 */
  HAL_DMA_IRQHandler(&hdma_usart1_tx);
  / * Код пользователя начинается dma2_stream7_irqn 1 */
  / * Код пользователя END DMA2_STREAM7_IRQN 1 */
}
```

```
/ * Код пользователя начинается 1 */
/ * Код пользователя Конец 1 */
```

Приложение Г

Описание файла stm32f4xx_hal_msp.c

```
/ * Код пользователя начинается за заголовок */
/**
******************************************************************************
***
  * @file STM32F4XX_HAL_MSP.C
  * @brief Этот файл предоставляет код для инициализации MSP
  * и коды де-инициализации.
******************************************************************************
***
  * @внимание
  *
  * Copyright (C) 2024 Stmicroelectronics.
  * Все права защищены.
  *
  * Это программное обеспечение лицензировано в соответствии с условиями, которые можно
найти в файле лицензии
  * В корневом каталоге этого программного компонента.
  * Если нет лицензионного файла с этим программным обеспечением, оно предоставляется
как есть.
  *
******************************************************************************
***
  */
/ * Заголовок кода пользователя */
/* Включает ------------------------------------------------------------------*/
#include "main.h"
/ * Код пользователя начинается в включении */
/ * Конец кода пользователя включает */
extern DMA_HandleTypeDef hdma_usart1_rx;
extern DMA_HandleTypeDef hdma_usart1_tx;
/* Private Typedef -----------------------------------------------------------*/
/ * Код пользователя начинается TD */
/ * Код пользователя END TD */
/* Private Define
------------------------------------------------------------------------*//
```

```
/ * Код пользователя начинается определить */
/ * Код пользователя END DEFINE */
/* Частный макрос ---------------------------------------------------------*/
/ * Код пользователя начинать макрос *//
/ * Код пользователя End Macro */
/* Приватные переменные ---------------------------------------------------------*/
/ * Код пользователя начинается PV */
/ * Код пользователя END PV */
/* Прототипы частной функции ---------------------------------------------------*/
/ * Код пользователя начинается PFP */
/ * Код пользователя END PFP */
/* Внешние функции ---------------------------------------------------------*/
/ * Код пользователя начинает внешние функции */
/ * Код пользователя и внешние функции */
/ * Код пользователя начинается 0 */
/ * Код пользователя конец 0 */
/**
  * Инициализирует глобальный MSP.
  */
void HAL_MspInit(void)
{
  / * Код пользователя начинается mspinit 0 */
  / * Код пользователя END MSPINIT 0 */
  __HAL_RCC_SYSCFG_CLK_ENABLE();
  __HAL_RCC_PWR_CLK_ENABLE();
  /* Системное прерывание init*/
  / * Код пользователя начинается mspinit 1 */
  / * Код пользователя END MSPINIT 1 */
}
/**
* @brief Uart MSP Инициализация
* Эта функция настраивает аппаратные ресурсы, используемые в этом примере
* @param huart: указатель ручки Uart
* @retval нет
*/
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART1)
  {
```

```
/ * Код пользователя начинается USART1_MSPINIT 0 */
/ * Код пользователя END USART1_MSPINIT 0 */
  / * Включение периферических часов */
  __HAL_RCC_USART1_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  /** USART1 Конфигурация GPIO
  PA9 -------> USART1_TX
  PA10 ------> USART1_RX
  */
  GPIO_InitStruct.Pin = GPIO_PIN_9|GPIO_PIN_10;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  / * USART1 DMA init */
  / * Usart1_rx init */
  hdma_usart1_rx.Instance = DMA2_Stream2;
  hdma_usart1_rx.Init.Channel = DMA_CHANNEL_4;
  hdma_usart1_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;
  hdma_usart1_rx.Init.PeriphInc = DMA_PINC_DISABLE;
  hdma_usart1_rx.Init.MemInc = DMA_MINC_ENABLE;
  hdma_usart1_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
  hdma_usart1_rx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
  hdma_usart1_rx.Init.Mode = DMA_NORMAL;
  hdma_usart1_rx.Init.Priority = DMA_PRIORITY_LOW;
  hdma_usart1_rx.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
  if (HAL_DMA_Init(&hdma_usart1_rx) != HAL_OK)
  {
    Error_Handler();
  }
  __HAL_LINKDMA(huart,hdmarx,hdma_usart1_rx);
  / * Usart1_tx init */
  hdma_usart1_tx.Instance = DMA2_Stream7;
  hdma_usart1_tx.Init.Channel = DMA_CHANNEL_4;
  hdma_usart1_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
  hdma_usart1_tx.Init.PeriphInc = DMA_PINC_DISABLE;
  hdma_usart1_tx.Init.MemInc = DMA_MINC_ENABLE;
  hdma_usart1_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
  hdma_usart1_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
```

```
    hdma_usart1_tx.Init.Mode = DMA_NORMAL;
    hdma_usart1_tx.Init.Priority = DMA_PRIORITY_LOW;
    hdma_usart1_tx.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
    if (HAL_DMA_Init(&hdma_usart1_tx) != HAL_OK)
    {
      Error_Handler();
    }
    __HAL_LINKDMA(huart,hdmatx,hdma_usart1_tx);
  / * Код пользователя начинается USART1_MSPINIT 1 */
  / * Код пользователя END USART1_MSPINIT 1 */
  }
}
/**
* @brief uart msp деиализация
* Эта функция замораживает аппаратные ресурсы, используемые в этом примере
* @param huart: указатель ручки Uart
* @retval нет
*/
void HAL_UART_MspDeInit(UART_HandleTypeDef* huart)
{
  if(huart->Instance==USART1)
  {
  / * Код пользователя начинается usart1_mspdeinit 0 */
  / * Код пользователя END USART1_MSPDEINIT 0 */
    / * Периферические часы отключены */
    __HAL_RCC_USART1_CLK_DISABLE();
    /** USART1 Конфигурация GPIO
    PA9 -------> USART1_TX
    PA10 ------> USART1_RX
    */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_9|GPIO_PIN_10);
    / * Usart1 dma deinit */
    HAL_DMA_DeInit(huart->hdmarx);
    HAL_DMA_DeInit(huart->hdmatx);
  / * Код пользователя начинается usart1_mspdeinit 1 */
  / * Код пользователя END USART1_MSPDEINIT 1 */
  }
}
/ * Код пользователя начинается 1 */
/ * Код пользователя Конец 1 */
```

Приложение Д

Описание файла stm32f4xx_hal_uart.c

```
/**
  ******************************************************************************
  * @file    stm32f4xx_hal_uart.c
  * @author  MCD Application Team
  * @brief   UART HAL module driver.
  *          This file provides firmware functions to manage the following
  *          functionalities of the Universal Asynchronous Receiver Transmitter
Peripheral (UART).
  *           + Initialization and de-initialization functions
  *           + IO operation functions
  *           + Peripheral Control functions
  *           + Peripheral State and Errors functions
  *
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2016 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  @verbatim
  ==============================================================================
                      ##### How to use this driver #####
  ==============================================================================
  [..]
    The UART HAL driver can be used as follows:
    (#) Declare a UART_HandleTypeDef handle structure (eg. UART_HandleTypeDef huart).
    (#) Initialize the UART low level resources by implementing the HAL_UART_MspInit()
API:
        (##) Enable the USARTx interface clock.
        (##) UART pins configuration:
```

```
            (+++) Enable the clock for the UART GPIOs.
            (+++) Configure the UART TX/RX pins as alternate function pull-up.
        (##) NVIC configuration if you need to use interrupt process
(HAL_UART_Transmit_IT()
             and HAL_UART_Receive_IT() APIs):
            (+++) Configure the USARTx interrupt priority.
            (+++) Enable the NVIC USART IRQ handle.
        (##) DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA()
             and HAL_UART_Receive_DMA() APIs):
            (+++) Declare a DMA handle structure for the Tx/Rx stream.
            (+++) Enable the DMAx interface clock.
            (+++) Configure the declared DMA handle structure with the required
                  Tx/Rx parameters.
            (+++) Configure the DMA Tx/Rx stream.
            (+++) Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
            (+++) Configure the priority and enable the NVIC for the transfer complete
                  interrupt on the DMA Tx/Rx stream.
            (+++) Configure the USARTx interrupt priority and enable the NVIC USART IRQ
handle
                  (used for last byte sending completion detection in DMA non circular
mode)
    (#) Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware
        flow control and Mode(Receiver/Transmitter) in the huart Init structure.
    (#) For the UART asynchronous mode, initialize the UART registers by calling
        the HAL_UART_Init() API.
    (#) For the UART Half duplex mode, initialize the UART registers by calling
        the HAL_HalfDuplex_Init() API.
    (#) For the LIN mode, initialize the UART registers by calling the HAL_LIN_Init()
API.
    (#) For the Multi-Processor mode, initialize the UART registers by calling
        the HAL_MultiProcessor_Init() API.
     [..]
       (@) The specific UART interrupts (Transmission complete interrupt,
            RXNE interrupt and Error Interrupts) will be managed using the macros
            __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit
            and receive process.
     [..]
       (@) These APIs (HAL_UART_Init() and HAL_HalfDuplex_Init()) configure also the
            low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized
            HAL_UART_MspInit() API.
```

```
    ##### Callback registration #####
    ==================================
    [..]
    The compilation define USE_HAL_UART_REGISTER_CALLBACKS when set to 1
    allows the user to configure dynamically the driver callbacks.
    [..]
    Use Function HAL_UART_RegisterCallback() to register a user callback.
    Function HAL_UART_RegisterCallback() allows to register following callbacks:
    (+) TxHalfCpltCallback        : Tx Half Complete Callback.
    (+) TxCpltCallback            : Tx Complete Callback.
    (+) RxHalfCpltCallback        : Rx Half Complete Callback.
    (+) RxCpltCallback            : Rx Complete Callback.
    (+) ErrorCallback             : Error Callback.
    (+) AbortCpltCallback         : Abort Complete Callback.
    (+) AbortTransmitCpltCallback : Abort Transmit Complete Callback.
    (+) AbortReceiveCpltCallback  : Abort Receive Complete Callback.
    (+) MspInitCallback           : UART MspInit.
    (+) MspDeInitCallback         : UART MspDeInit.
    This function takes as parameters the HAL peripheral handle, the Callback ID
    and a pointer to the user callback function.
    [..]
    Use function HAL_UART_UnRegisterCallback() to reset a callback to the default
    weak (surcharged) function.
    HAL_UART_UnRegisterCallback() takes as parameters the HAL peripheral handle,
    and the Callback ID.
    This function allows to reset following callbacks:
    (+) TxHalfCpltCallback        : Tx Half Complete Callback.
    (+) TxCpltCallback            : Tx Complete Callback.
    (+) RxHalfCpltCallback        : Rx Half Complete Callback.
    (+) RxCpltCallback            : Rx Complete Callback.
    (+) ErrorCallback             : Error Callback.
    (+) AbortCpltCallback         : Abort Complete Callback.
    (+) AbortTransmitCpltCallback : Abort Transmit Complete Callback.
    (+) AbortReceiveCpltCallback  : Abort Receive Complete Callback.
    (+) MspInitCallback           : UART MspInit.
    (+) MspDeInitCallback         : UART MspDeInit.
    [..]
    For specific callback RxEventCallback, use dedicated registration/reset functions:
    respectively HAL_UART_RegisterRxEventCallback() ,
HAL_UART_UnRegisterRxEventCallback().
```

```
    [..]
    By default, after the HAL_UART_Init() and when the state is HAL_UART_STATE_RESET
    all callbacks are set to the corresponding weak (surcharged) functions:
    examples HAL_UART_TxCpltCallback(), HAL_UART_RxHalfCpltCallback().
    Exception done for MspInit and MspDeInit functions that are respectively
    reset to the legacy weak (surcharged) functions in the HAL_UART_Init()
    and HAL_UART_DeInit() only when these callbacks are null (not registered
beforehand).
    If not, MspInit or MspDeInit are not null, the HAL_UART_Init() and
HAL_UART_DeInit()
    keep and use the user MspInit/MspDeInit callbacks (registered beforehand).
    [..]
    Callbacks can be registered/unregistered in HAL_UART_STATE_READY state only.
    Exception done MspInit/MspDeInit that can be registered/unregistered
    in HAL_UART_STATE_READY or HAL_UART_STATE_RESET state, thus registered (user)
    MspInit/DeInit callbacks can be used during the Init/DeInit.
    In that case first register the MspInit/MspDeInit user callbacks
    using HAL_UART_RegisterCallback() before calling HAL_UART_DeInit()
    or HAL_UART_Init() function.
    [..]
    When The compilation define USE_HAL_UART_REGISTER_CALLBACKS is set to 0 or
    not defined, the callback registration feature is not available
    and weak (surcharged) callbacks are used.
     [..]
        Three operation modes are available within this driver :
     *** Polling mode IO operation ***
     =================================
     [..]
       (+) Send an amount of data in blocking mode using HAL_UART_Transmit()
       (+) Receive an amount of data in blocking mode using HAL_UART_Receive()
     *** Interrupt mode IO operation ***
     ===================================
     [..]
       (+) Send an amount of data in non blocking mode using HAL_UART_Transmit_IT()
       (+) At transmission end of transfer HAL_UART_TxCpltCallback is executed and user
can
            add his own code by customization of function pointer
HAL_UART_TxCpltCallback
       (+) Receive an amount of data in non blocking mode using HAL_UART_Receive_IT()
```

```
       (+) At reception end of transfer HAL_UART_RxCpltCallback is executed and user
can
            add his own code by customization of function pointer
HAL_UART_RxCpltCallback
       (+) In case of transfer Error, HAL_UART_ErrorCallback() function is executed and
user can
            add his own code by customization of function pointer
HAL_UART_ErrorCallback
     *** DMA mode IO operation ***
     ==============================
     [..]
       (+) Send an amount of data in non blocking mode (DMA) using
HAL_UART_Transmit_DMA()
       (+) At transmission end of half transfer HAL_UART_TxHalfCpltCallback is executed
and user can
            add his own code by customization of function pointer
HAL_UART_TxHalfCpltCallback
       (+) At transmission end of transfer HAL_UART_TxCpltCallback is executed and user
can
            add his own code by customization of function pointer
HAL_UART_TxCpltCallback
       (+) Receive an amount of data in non blocking mode (DMA) using
HAL_UART_Receive_DMA()
       (+) At reception end of half transfer HAL_UART_RxHalfCpltCallback is executed
and user can
            add his own code by customization of function pointer
HAL_UART_RxHalfCpltCallback
       (+) At reception end of transfer HAL_UART_RxCpltCallback is executed and user
can
            add his own code by customization of function pointer
HAL_UART_RxCpltCallback
       (+) In case of transfer Error, HAL_UART_ErrorCallback() function is executed and
user can
            add his own code by customization of function pointer
HAL_UART_ErrorCallback
       (+) Pause the DMA Transfer using HAL_UART_DMAPause()
       (+) Resume the DMA Transfer using HAL_UART_DMAResume()
       (+) Stop the DMA Transfer using HAL_UART_DMAStop()
```

```
    [..] This subsection also provides a set of additional functions providing enhanced reception
    services to user. (For example, these functions allow application to handle use cases
    where number of data to be received is unknown).
    (#) Compared to standard reception services which only consider number of received
        data elements as reception completion criteria, these functions also consider additional events
        as triggers for updating reception status to caller :
       (+) Detection of inactivity period (RX line has not been active for a given period).
          (++) RX inactivity detected by IDLE event, i.e. RX line has been in idle state (normally high state)
               for 1 frame time, after last received byte.
    (#) There are two mode of transfer:
       (+) Blocking mode: The reception is performed in polling mode, until either expected number of data is received,
           or till IDLE event occurs. Reception is handled only during function execution.
           When function exits, no data reception could occur. HAL status and number of actually received data elements,
           are returned by function after finishing transfer.
       (+) Non-Blocking mode: The reception is performed using Interrupts or DMA.
           These API's return the HAL status.
           The end of the data processing will be indicated through the
           dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
           The HAL_UARTEx_RxEventCallback() user callback will be executed during Receive process
           The HAL_UART_ErrorCallback()user callback will be executed when a reception error is detected.
    (#) Blocking mode API:
        (+) HAL_UARTEx_ReceiveToIdle()
    (#) Non-Blocking mode API with Interrupt:
        (+) HAL_UARTEx_ReceiveToIdle_IT()
    (#) Non-Blocking mode API with DMA:
        (+) HAL_UARTEx_ReceiveToIdle_DMA()

     *** UART HAL driver macros list ***
     ============================================
```

```
     [..]
       Below the list of most used macros in UART HAL driver.
      (+) __HAL_UART_ENABLE: Enable the UART peripheral
      (+) __HAL_UART_DISABLE: Disable the UART peripheral
      (+) __HAL_UART_GET_FLAG : Check whether the specified UART flag is set or not
      (+) __HAL_UART_CLEAR_FLAG : Clear the specified UART pending flag
      (+) __HAL_UART_ENABLE_IT: Enable the specified UART interrupt
      (+) __HAL_UART_DISABLE_IT: Disable the specified UART interrupt
      (+) __HAL_UART_GET_IT_SOURCE: Check whether the specified UART interrupt has
occurred or not
     [..]
       (@) You can refer to the UART HAL driver header file for more useful macros
  @endverbatim
     [..]
       (@) Additional remark: If the parity is enabled, then the MSB bit of the data
written
           in the data register is transmitted but is changed by the parity bit.
           Depending on the frame length defined by the M bit (8-bits or 9-bits),
           the possible UART frame formats are as listed in the following table:
    +-------------------------------------------------------------+
    |   M bit |  PCE bit  |            UART frame                 |
    |---------------------|---------------------------------------|
    |    0    |    0      |    | SB | 8 bit data | STB |          |
    |---------|-----------|---------------------------------------|
    |    0    |    1      |    | SB | 7 bit data | PB | STB |     |
    |---------|-----------|---------------------------------------|
    |    1    |    0      |    | SB | 9 bit data | STB |          |
    |---------|-----------|---------------------------------------|
    |    1    |    1      |    | SB | 8 bit data | PB | STB |     |
    +-------------------------------------------------------------+
  ******************************************************************************
  */
/* Включает ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
/** @addtogroup stm32f4xx_hal_driver
  * @{
  */
/** @defgroup uart uart
  * @brief Hal Uart Driver
  * @{
```

```
  */
#ifdef HAL_UART_MODULE_ENABLED
/* Private Typedef -----------------------------------------------------------*/
/* Private Define
------------------------------------------------------------------------------*//
/** @addtogroup uart_private_constants
  * @{
  */
/**
  * @}
  */
/* Частный макрос ------------------------------------------------------------*/
/* Приватные переменные ------------------------------------------------------*/
/* Прототипы частной функции -------------------------------------------------*/
/** @addtogroup uart_private_functions uart частные функции
  * @{
  */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
void UART_InitCallbacksToDefault(UART_HandleTypeDef *huart);
#endif / * Use_hal_uart_register_callbacks */
static void UART_EndTxTransfer(UART_HandleTypeDef *huart);
static void UART_EndRxTransfer(UART_HandleTypeDef *huart);
static void UART_DMATransmitCplt(DMA_HandleTypeDef *hdma);
static void UART_DMAReceiveCplt(DMA_HandleTypeDef *hdma);
static void UART_DMATxHalfCplt(DMA_HandleTypeDef *hdma);
static void UART_DMARxHalfCplt(DMA_HandleTypeDef *hdma);
static void UART_DMAError(DMA_HandleTypeDef *hdma);
static void UART_DMAAbortOnError(DMA_HandleTypeDef *hdma);
static void UART_DMATxAbortCallback(DMA_HandleTypeDef *hdma);
static void UART_DMARxAbortCallback(DMA_HandleTypeDef *hdma);
static void UART_DMATxOnlyAbortCallback(DMA_HandleTypeDef *hdma);
static void UART_DMARxOnlyAbortCallback(DMA_HandleTypeDef *hdma);
static HAL_StatusTypeDef UART_Transmit_IT(UART_HandleTypeDef *huart);
static HAL_StatusTypeDef UART_EndTransmit_IT(UART_HandleTypeDef *huart);
static HAL_StatusTypeDef UART_Receive_IT(UART_HandleTypeDef *huart);
static HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout(UART_HandleTypeDef *huart,
uint32_t Flag, FlagStatus Status,
                                                     uint32_t Tickstart, uint32_t
Timeout);
static void UART_SetConfig(UART_HandleTypeDef *huart);
```

```
/**
  * @}
  */
/* Экспортируемые функции ---------------------------------------------------*/
/** @defgroup uart_exported_functions uart экспортируемые функции
  * @{
  */
/** @defgroup uart_exported_functions_group1 инициализация и де-инициализация функций
  * @brief инициализация и конфигурация
  *
@verbatim

==============================================================================
=================
            ##### Функции инициализации и конфигурации #####

==============================================================================
=================
    [..]
    Этот подраздел предоставляет набор функций, позволяющих инициализировать USARTX или
Uarty
    в асинхронном режиме.
      (+) Для асинхронного режима можно настроить только эти параметры:
        (++) Скорость передачи
        (++) длина слова
        (++) остановить бит
        (++) паритет: если паритет включен, то бить MSB написанных
             В регистре данных передается, но изменяется битом паритета.
             В зависимости от длины кадра, определенной битом M (8-бит или 9-бит),
             Пожалуйста, обратитесь к справочному руководству для возможных форматов
кадров UART.
        (++) Управление аппаратным потоком
        (++) режимы приемника/передатчика
        (++) по методу отбора проб
    [..]
    Hal_uart_init (), hal_halfduplex_init (), hal_lin_init () и hal_multiprocessor_init
() APIS
    Следуйте соответственно асинхронная конфигурация UART, полусуплекс, LIN и
многопроцессор
    Процедуры (подробности процедур доступны в справочном руководстве
```

```
    (RM0430 для MCUS STM32F4X3XX и RM0402 для MCUS STM32F412XX
     RM0383 для STM32F411XC/E MCUS и RM0401 для MCUS STM32F410XX
     RM0090 для STM32F4X5XX/STM32F4X7XX/STM32F429XX/STM32F439XX MCUS
     RM0390 для MCUS STM32F446XX и RM0386 для STM32F469XX/STM32F479XX MCUS)).
@endverbatim
  * @{
  */
/**
  * @Brief инициализирует режим UART в соответствии с указанными параметрами в
  * UART_INITTYPEDEF и создайте соответствующую ручку.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart)
{
  / * Проверьте распределение ручки UART */
  if (huart == NULL)
  {
    return HAL_ERROR;
  }
  / * Проверьте параметры */
  if (huart->Init.HwFlowCtl != UART_HWCONTROL_NONE)
  {
    /* Управление аппаратным потоком доступно только для USART1, USART2, USART3 и
USART6.
       За исключением устройств STM32F446XX, которые доступны для USART1, USART2,
USART3, USART6, UART4 и UART5.
    */
    assert_param(IS_UART_HWFLOW_INSTANCE(huart->Instance));
    assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));
  }
  else
  {
    assert_param(IS_UART_INSTANCE(huart->Instance));
  }
  assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
  assert_param(IS_UART_OVERSAMPLING(huart->Init.OverSampling));
  if (huart->gState == HAL_UART_STATE_RESET)
  {
```

```
    / * Выделите ресурс блокировки и инициализируйте его */
    huart->Lock = HAL_UNLOCKED;
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    UART_InitCallbacksToDefault(huart);
    if (huart->MspInitCallback == NULL)
    {
      huart->MspInitCallback = HAL_UART_MspInit;
    }
    / * Инициировать оборудование низкого уровня */
    huart->MspInitCallback(huart);
#else
    / * Init Оборудование низкого уровня: gpio, часы */
    HAL_UART_MspInit(huart);
#endif / * (Use_hal_uart_register_callbacks) */
  }
  huart->gState = HAL_UART_STATE_BUSY;
  / * Отключить периферическое */
  __HAL_UART_DISABLE(huart);
  / * Установите параметры связи UART */
  UART_SetConfig(huart);
  /* В асинхронном режиме должны быть очищены следующие биты:
     - биты льна и Clken в регистрации USART_CR2,
     - Сцена, HDSEL и Iren Bits в регистрации USART_CR3.*/
  CLEAR_BIT(huart->Instance->CR2, (USART_CR2_LINEN | USART_CR2_CLKEN));
  CLEAR_BIT(huart->Instance->CR3, (USART_CR3_SCEN | USART_CR3_HDSEL | USART_CR3_IREN));
  / * Включить периферическое */
  __HAL_UART_ENABLE(huart);
  / * Инициализировать состояние UART */
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  huart->gState = HAL_UART_STATE_READY;
  huart->RxState = HAL_UART_STATE_READY;
  return HAL_OK;
}
/**
  * @Brief инициализирует режим полумуплекса в соответствии с указанным
  * Параметры в UART_INITTYPEDEF и создайте соответствующую ручку.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
```

```
HAL_StatusTypeDef HAL_HalfDuplex_Init(UART_HandleTypeDef *huart)
{
  / * Проверьте распределение ручки UART */
  if (huart == NULL)
  {
    return HAL_ERROR;
  }
  / * Проверьте параметры */
  assert_param(IS_UART_HALFDUPLEX_INSTANCE(huart->Instance));
  assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
  assert_param(IS_UART_OVERSAMPLING(huart->Init.OverSampling));
  if (huart->gState == HAL_UART_STATE_RESET)
  {
    / * Выделите ресурс блокировки и инициализируйте его */
    huart->Lock = HAL_UNLOCKED;
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    UART_InitCallbacksToDefault(huart);
    if (huart->MspInitCallback == NULL)
    {
      huart->MspInitCallback = HAL_UART_MspInit;
    }
    / * Инициировать оборудование низкого уровня */
    huart->MspInitCallback(huart);
#else
    / * Init Оборудование низкого уровня: gpio, часы */
    HAL_UART_MspInit(huart);
#endif / * (Use_hal_uart_register_callbacks) */
  }
  huart->gState = HAL_UART_STATE_BUSY;
  / * Отключить периферическое */
  __HAL_UART_DISABLE(huart);
  / * Установите параметры связи UART */
  UART_SetConfig(huart);
  /* В полудуплексном режиме следует очистить следующие биты:
     - биты льна и Clken в регистрации USART_CR2,
     - Сцена и Ирен в регистрации USART_CR3.*/
  CLEAR_BIT(huart->Instance->CR2, (USART_CR2_LINEN | USART_CR2_CLKEN));
  CLEAR_BIT(huart->Instance->CR3, (USART_CR3_IREN | USART_CR3_SCEN));
  / * Включить режим полумуплекса, установив бит HDSEL в регистре CR3 */
  SET_BIT(huart->Instance->CR3, USART_CR3_HDSEL);
```

```
  / * Включить периферическое */
  __HAL_UART_ENABLE(huart);
  /* Инициализировать состояние UART*/
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  huart->gState = HAL_UART_STATE_READY;
  huart->RxState = HAL_UART_STATE_READY;
  return HAL_OK;
}
/**
  * @Brief инициализирует режим LIN в соответствии с указанным
  * Параметры в UART_INITTYPEDEF и создайте соответствующую ручку.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @param breakdetectlength определяет длину обнаружения разрыва LIN.
  * Этот параметр может быть одним из следующих значений:
  * @arg uart_linebreakdetectlength_10b: 10-битное обнаружение разрыва
  * @arg uart_linebreakdetectlength_11b: 11-битное обнаружение разрыва
  * @retval hal status
  */
HAL_StatusTypeDef HAL_LIN_Init(UART_HandleTypeDef *huart, uint32_t BreakDetectLength)
{
  / * Проверьте распределение ручки UART */
  if (huart == NULL)
  {
    return HAL_ERROR;
  }
  / * Проверьте экземпляр Lin Uart */
  assert_param(IS_UART_LIN_INSTANCE(huart->Instance));
  / * Проверьте параметр длины обнаружения разрыва */
  assert_param(IS_UART_LIN_BREAK_DETECT_LENGTH(BreakDetectLength));
  assert_param(IS_UART_LIN_WORD_LENGTH(huart->Init.WordLength));
  assert_param(IS_UART_LIN_OVERSAMPLING(huart->Init.OverSampling));
  if (huart->gState == HAL_UART_STATE_RESET)
  {
    / * Выделите ресурс блокировки и инициализируйте его */
    huart->Lock = HAL_UNLOCKED;
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    UART_InitCallbacksToDefault(huart);
    if (huart->MspInitCallback == NULL)
    {
```

```
      huart->MspInitCallback = HAL_UART_MspInit;
    }
    / * Инициировать оборудование низкого уровня */
    huart->MspInitCallback(huart);
#else
    / * Init Оборудование низкого уровня: gpio, часы */
    HAL_UART_MspInit(huart);
#endif / * (Use_hal_uart_register_callbacks) */
  }
  huart->gState = HAL_UART_STATE_BUSY;
  / * Отключить периферическое */
  __HAL_UART_DISABLE(huart);
  / * Установите параметры связи UART */
  UART_SetConfig(huart);
  /* В режиме Лин, необходимо очистить следующие биты:
     - биты Clken в регистре usart_cr2,
     - Сцена, HDSEL и Iren Bits в регистрации USART_CR3.*/
  CLEAR_BIT(huart->Instance->CR2, (USART_CR2_CLKEN));
  CLEAR_BIT(huart->Instance->CR3, (USART_CR3_HDSEL | USART_CR3_IREN | USART_CR3_SCEN));
  / * Включите режим LIN, установив бит белья в регистре CR2 */
  SET_BIT(huart->Instance->CR2, USART_CR2_LINEN);
  /* Установите длину обнаружения разрыва USART.*/
  CLEAR_BIT(huart->Instance->CR2, USART_CR2_LBDL);
  SET_BIT(huart->Instance->CR2, BreakDetectLength);
  / * Включить периферическое */
  __HAL_UART_ENABLE(huart);
  /* Инициализировать состояние UART*/
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  huart->gState = HAL_UART_STATE_READY;
  huart->RxState = HAL_UART_STATE_READY;
  return HAL_OK;
}
/**
  * @Brief инициализирует многопроцессорный режим в соответствии с указанным
  * Параметры в UART_INITTYPEDEF и создайте соответствующую ручку.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @param адрес адрес USART
  * @param Wakeupmethod указывает метод пробуждения USART.
  * Этот параметр может быть одним из следующих значений:
```

```
  * @arg uart_wakeupmethod_idleline: пробуждение от обнаружения линии на холостом ходу
  * @arg uart_wakeupmethod_addressmark: Wake-up по адресу адреса
  * @retval hal status
  */
HAL_StatusTypeDef HAL_MultiProcessor_Init(UART_HandleTypeDef *huart, uint8_t Address,
uint32_t WakeUpMethod)
{
  / * Проверьте распределение ручки UART */
  if (huart == NULL)
  {
    return HAL_ERROR;
  }
  / * Проверьте параметры */
  assert_param(IS_UART_INSTANCE(huart->Instance));
  / * Проверьте параметры метода адреса и пробуждения */
  assert_param(IS_UART_WAKEUPMETHOD(WakeUpMethod));
  assert_param(IS_UART_ADDRESS(Address));
  assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
  assert_param(IS_UART_OVERSAMPLING(huart->Init.OverSampling));
  if (huart->gState == HAL_UART_STATE_RESET)
  {
    / * Выделите ресурс блокировки и инициализируйте его */
    huart->Lock = HAL_UNLOCKED;
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    UART_InitCallbacksToDefault(huart);
    if (huart->MspInitCallback == NULL)
    {
      huart->MspInitCallback = HAL_UART_MspInit;
    }
    / * Инициировать оборудование низкого уровня */
    huart->MspInitCallback(huart);
#else
    / * Init Оборудование низкого уровня: gpio, часы */
    HAL_UART_MspInit(huart);
#endif / * (Use_hal_uart_register_callbacks) */
  }
  huart->gState = HAL_UART_STATE_BUSY;
  / * Отключить периферическое */
  __HAL_UART_DISABLE(huart);
  / * Установите параметры связи UART */
```

```
  UART_SetConfig(huart);
  /* В многопроцессорном режиме необходимо очистить следующие биты:
     - биты льна и Clken в регистрации USART_CR2,
     - Сцена, HDSEL и Iren Bits в регистрации USART_CR3 */
  CLEAR_BIT(huart->Instance->CR2, (USART_CR2_LINEN | USART_CR2_CLKEN));
  CLEAR_BIT(huart->Instance->CR3, (USART_CR3_SCEN | USART_CR3_HDSEL | USART_CR3_IREN));
  / * Установите адресный узел USART */
  CLEAR_BIT(huart->Instance->CR2, USART_CR2_ADD);
  SET_BIT(huart->Instance->CR2, Address);
  / * Установите метод Wake Up, установив бит Wake в регистре CR1 */
  CLEAR_BIT(huart->Instance->CR1, USART_CR1_WAKE);
  SET_BIT(huart->Instance->CR1, WakeUpMethod);
  / * Включить периферическое */
  __HAL_UART_ENABLE(huart);
  / * Инициализировать состояние UART */
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  huart->gState = HAL_UART_STATE_READY;
  huart->RxState = HAL_UART_STATE_READY;
  return HAL_OK;
}
/**
  * @brief деинитирует периферийное устройство UART.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_DeInit(UART_HandleTypeDef *huart)
{
  / * Проверьте распределение ручки UART */
  if (huart == NULL)
  {
    return HAL_ERROR;
  }
  / * Проверьте параметры */
  assert_param(IS_UART_INSTANCE(huart->Instance));
  huart->gState = HAL_UART_STATE_BUSY;
  / * Отключить периферическое */
  __HAL_UART_DISABLE(huart);
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
  if (huart->MspDeInitCallback == NULL)
```

```
  {
    huart->MspDeInitCallback = HAL_UART_MspDeInit;
  }
  / * Дейнит оборудование низкого уровня */
  huart->MspDeInitCallback(huart);
#else
  / * Дейнит оборудование низкого уровня */
  HAL_UART_MspDeInit(huart);
#endif / * (Use_hal_uart_register_callbacks) */
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  huart->gState = HAL_UART_STATE_RESET;
  huart->RxState = HAL_UART_STATE_RESET;
  huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
  / * Разблокировать процесс */
  __HAL_UNLOCK(huart);
  return HAL_OK;
}
/**
  * @brief uart msp init.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval нет
  */
__weak void HAL_UART_MspInit(UART_HandleTypeDef *huart)
{
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_MSPINIT может быть реализован в пользовательском файле
   */
}
/**
  * @brief uart msp deinit.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval нет
  */
__weak void HAL_UART_MspDeInit(UART_HandleTypeDef *huart)
{
  / * Предупреждение о компиляции неиспользованных аргументов
```

```
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_MSPDEINIT может быть реализован в пользовательском файле
   */
}
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
/**
  * @brief Зарегистрировать обратный вызов пользователя UART
  * Использоваться вместо слабого предопределенного обратного вызова
  * @param huart uart
  * @param callbackid идентификатор обратного вызова, который будет зарегистрирован
  * Этот параметр может быть одним из следующих значений:
  * @arg @ref hal_uart_tx_halfcomplete_cb_id tx Половина полного идентификатора
обратного вызова обратного вызова
  * @arg @ref hal_uart_tx_complete_cb_id tx Полный обратный идентификатор обратного
вызова
  * @arg @ref hal_uart_rx_halfcomplete_cb_id rx Половина полного идентификатора
обратного вызова обратного вызова
  * @arg @ref hal_uart_rx_complete_cb_id rx Полный обратный идентификатор обратного
вызова
  * @arg @ref hal_uart_error_cb_id ошибка идентификатор обратного вызова обратного
вызова
  * @Arg @Ref hal_uart_abort_complete_cb_id Abort Complete Callwack ID
  * @Arg @Ref hal_uart_abort_transmit_complete_cb_id.
  * @Arg @Ref hal_uart_abort_receive_complete_cb_id Abort Получить полное обратное
идентификатор обратного вызова
  * @arg @ref hal_uart_mspinit_cb_id mspinit идентификатор обратного вызова
  * @arg @ref hal_uart_mspdeinit_cb_id mspdeinit идентификатор обратного вызова
  * @param pcallback pointer на функцию обратного вызова
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_RegisterCallback(UART_HandleTypeDef *huart,
HAL_UART_CallbackIDTypeDef CallbackID,
                                            pUART_CallbackTypeDef pCallback)
{
  HAL_StatusTypeDef status = HAL_OK;
  if (pCallback == NULL)
  {
    / * Обновите код ошибки */
    huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
```

```
  return HAL_ERROR;
}
/ * Процесс заблокирован */
__HAL_LOCK(huart);
if (huart->gState == HAL_UART_STATE_READY)
{
  switch (CallbackID)
  {
    case HAL_UART_TX_HALFCOMPLETE_CB_ID :
      huart->TxHalfCpltCallback = pCallback;
      break;
    case HAL_UART_TX_COMPLETE_CB_ID :
      huart->TxCpltCallback = pCallback;
      break;
    case HAL_UART_RX_HALFCOMPLETE_CB_ID :
      huart->RxHalfCpltCallback = pCallback;
      break;
    case HAL_UART_RX_COMPLETE_CB_ID :
      huart->RxCpltCallback = pCallback;
      break;
    case HAL_UART_ERROR_CB_ID :
      huart->ErrorCallback = pCallback;
      break;
    case HAL_UART_ABORT_COMPLETE_CB_ID :
      huart->AbortCpltCallback = pCallback;
      break;
    case HAL_UART_ABORT_TRANSMIT_COMPLETE_CB_ID :
      huart->AbortTransmitCpltCallback = pCallback;
      break;
    case HAL_UART_ABORT_RECEIVE_COMPLETE_CB_ID :
      huart->AbortReceiveCpltCallback = pCallback;
      break;
    case HAL_UART_MSPINIT_CB_ID :
      huart->MspInitCallback = pCallback;
      break;
    case HAL_UART_MSPDEINIT_CB_ID :
      huart->MspDeInitCallback = pCallback;
      break;
    default :
      / * Обновите код ошибки */
```

```
        huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
        / * Статус ошибки возврата */
        status =  HAL_ERROR;
        break;
    }
  }
  else if (huart->gState == HAL_UART_STATE_RESET)
  {
    switch (CallbackID)
    {
      case HAL_UART_MSPINIT_CB_ID :
        huart->MspInitCallback = pCallback;
        break;
      case HAL_UART_MSPDEINIT_CB_ID :
        huart->MspDeInitCallback = pCallback;
        break;
      default :
        / * Обновите код ошибки */
        huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
        / * Статус ошибки возврата */
        status =  HAL_ERROR;
        break;
    }
  }
  else
  {
    / * Обновите код ошибки */
    huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
    / * Статус ошибки возврата */
    status =  HAL_ERROR;
  }
  / * Выпуск блокировки */
  __HAL_UNLOCK(huart);
  return status;
}
/**
  * @brief не регистрирует обратный вызов UART
  * Uart Callaback перенаправлен на слабый предопределенный обратный вызов
  * @param huart uart
  * @param обратный идентификатор обратного вызова, чтобы быть незарегистрированным
```

```
  * Этот параметр может быть одним из следующих значений:
  * @arg @ref hal_uart_tx_halfcomplete_cb_id tx Половина полного идентификатора
обратного вызова обратного вызова
  * @arg @ref hal_uart_tx_complete_cb_id tx Полный обратный идентификатор обратного
вызова
  * @arg @ref hal_uart_rx_halfcomplete_cb_id rx Половина полного идентификатора
обратного вызова обратного вызова
  * @arg @ref hal_uart_rx_complete_cb_id rx Полный обратный идентификатор обратного
вызова
  * @arg @ref hal_uart_error_cb_id ошибка идентификатор обратного вызова обратного
вызова
  * @Arg @Ref hal_uart_abort_complete_cb_id Abort Complete Callwack ID
  * @Arg @Ref hal_uart_abort_transmit_complete_cb_id.
  * @Arg @Ref hal_uart_abort_receive_complete_cb_id Abort Получить полное обратное
идентификатор обратного вызова
  * @arg @ref hal_uart_mspinit_cb_id mspinit идентификатор обратного вызова
  * @arg @ref hal_uart_mspdeinit_cb_id mspdeinit идентификатор обратного вызова
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_UnRegisterCallback(UART_HandleTypeDef *huart,
HAL_UART_CallbackIDTypeDef CallbackID)
{
  HAL_StatusTypeDef status = HAL_OK;
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  if (HAL_UART_STATE_READY == huart->gState)
  {
    switch (CallbackID)
    {
      case HAL_UART_TX_HALFCOMPLETE_CB_ID :
        huart->TxHalfCpltCallback = HAL_UART_TxHalfCpltCallback;               / *
Legacy Sleed txhalfcpltcallback */
        break;
      case HAL_UART_TX_COMPLETE_CB_ID :
        huart->TxCpltCallback = HAL_UART_TxCpltCallback;                       / *
Legacy слабый txcpltcallback */
        break;
      case HAL_UART_RX_HALFCOMPLETE_CB_ID :
        huart->RxHalfCpltCallback = HAL_UART_RxHalfCpltCallback;               / *
Legacy слабый rxhalfcpltcallback */
```

```
        break;
      case HAL_UART_RX_COMPLETE_CB_ID :
        huart->RxCpltCallback = HAL_UART_RxCpltCallback;                        / *
Legacy слабый rxcpltcallback */
        break;
      case HAL_UART_ERROR_CB_ID :
        huart->ErrorCallback = HAL_UART_ErrorCallback;                          / *
Legacy Shad ErryCallback */
        break;
      case HAL_UART_ABORT_COMPLETE_CB_ID :
        huart->AbortCpltCallback = HAL_UART_AbortCpltCallback;                  / *
Legacy Shad Abortcpltcallback */
        break;
      case HAL_UART_ABORT_TRANSMIT_COMPLETE_CB_ID :
        huart->AbortTransmitCpltCallback = HAL_UART_AbortTransmitCpltCallback; / *
Legacy Shad AbortTransmitcpltcallback */
        break;
      case HAL_UART_ABORT_RECEIVE_COMPLETE_CB_ID :
        huart->AbortReceiveCpltCallback = HAL_UART_AbortReceiveCpltCallback;   / *
Legacy Shad Abortreceivecpltcallback */
        break;
      case HAL_UART_MSPINIT_CB_ID :
        huart->MspInitCallback = HAL_UART_MspInit;                              / *
Legacy слабый mspinitcallback */
        break;
      case HAL_UART_MSPDEINIT_CB_ID :
        huart->MspDeInitCallback = HAL_UART_MspDeInit;                          / *
Legacy слабый mspdeinitcallback */
        break;
      default :
        / * Обновите код ошибки */
        huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
        / * Статус ошибки возврата */
        status =  HAL_ERROR;
        break;
    }
  }
  else if (HAL_UART_STATE_RESET == huart->gState)
  {
    switch (CallbackID)
```

```
    {
      case HAL_UART_MSPINIT_CB_ID :
        huart->MspInitCallback = HAL_UART_MspInit;
        break;
      case HAL_UART_MSPDEINIT_CB_ID :
        huart->MspDeInitCallback = HAL_UART_MspDeInit;
        break;
      default :
        / * Обновите код ошибки */
        huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
        / * Статус ошибки возврата */
        status =  HAL_ERROR;
        break;
    }
  }
  else
  {
    / * Обновите код ошибки */
    huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
    / * Статус ошибки возврата */
    status =  HAL_ERROR;
  }
  / * Выпуск блокировки */
  __HAL_UNLOCK(huart);
  return status;
}
/**
  * @brief Зарегистрировать пользователь UART RX Callback Event
  * Использоваться вместо слабого предопределенного обратного вызова
  * @param huart uart
  * @param pcallback pointer на функцию обратного вызова RX Event
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_RegisterRxEventCallback(UART_HandleTypeDef *huart,
pUART_RxEventCallbackTypeDef pCallback)
{
  HAL_StatusTypeDef status = HAL_OK;
  if (pCallback == NULL)
  {
    huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
```

```
    return HAL_ERROR;
  }
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  if (huart->gState == HAL_UART_STATE_READY)
  {
    huart->RxEventCallback = pCallback;
  }
  else
  {
    huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
    status =  HAL_ERROR;
  }
  / * Выпуск блокировки */
  __HAL_UNLOCK(huart);
  return status;
}
/**
  * @brief unregister the uart rx callback
  * UART RX Callback перенаправлен на слабый HAL_UARTEX_RXEVENTCALLBACK ()
Предварительно определенный обратный вызов
  * @param huart uart
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_UnRegisterRxEventCallback(UART_HandleTypeDef *huart)
{
  HAL_StatusTypeDef status = HAL_OK;
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  if (huart->gState == HAL_UART_STATE_READY)
  {
    huart->RxEventCallback = HAL_UARTEx_RxEventCallback; / * Legacy Shad Uart Rx
Callback *//
  }
  else
  {
    huart->ErrorCode |= HAL_UART_ERROR_INVALID_CALLBACK;
    status =  HAL_ERROR;
  }
  / * Выпуск блокировки */
```

```
  __HAL_UNLOCK(huart);
  return status;
}
#endif / * Use_hal_uart_register_callbacks */
/**
  * @}
  */
/** @defgroup uart_exported_functions_group2 функции работы
  * @Brief UART передавать и получать функции
  *
@verbatim

==============================================================================
=================
                     ##### функции работы #####

==============================================================================
=================
    Этот подраздел предоставляет набор функций, позволяющих управлять асинхронным UART
    и половина дуплексных данных.
    (#) Есть два режима передачи:
       (+) Режим блокировки: связь выполняется в режиме опроса.
           Статус HAL всей обработки данных возвращается одной и той же функцией
           После окончания переноса.
       (+) Режим без блокировки: связь выполняется с использованием прерываний
           или DMA, эти API возвращают статус HAL.
           Конец обработки данных будет указан через
           Выделенный UART IRQ при использовании режима прерывания или DMA IRQ, когда
           Использование режима DMA.
           HAL_UART_TXCPLTCALLBACK (), HAL_UART_RXCPLTCALLBACK () Образцы пользователя
           будет выполняться соответственно в конце процесса передачи или приема
           Обратный вызов пользователя hal_uart_errorcallback () будет выполнен при
обнаружении ошибки связи.
    (#) API режима блокировки:
        (+) Hal_uart_transmit ()
        (+) Hal_uart_receive ()
    (#) API не блокирующих режима с прерыванием:
        (+) Hal_uart_transmit_it ()
        (+) Hal_uart_receive_it ()
        (+) Hal_uart_irqhandler ()
```

(#) API не блокирующих режима с DMA:

(+) Hal_uart_transmit_dma ()

(+) Hal_uart_receive_dma ()

(+) Hal_uart_dmapause ()

(+) Hal_uart_dmaresume ()

(+) Hal_uart_dmastop ()

(#) В режиме нереблокируется комплект полных обратных вызовов:

(+) Hal_uart_txhalfcpltcallback ()

(+) Hal_uart_txcpltcallback ()

(+) Hal_uart_rxhalfcpltcallback ()

(+) Hal_uart_rxcpltcallback ()

(+) Hal_uart_errorcallback ()

(#) Незащитные передачи режима могут быть прерваны с помощью Abort API:

(+) Hal_uart_abort ()

(+) Hal_uart_aborttransmit ()

(+) Hal_uart_abortreceive ()

(+) Hal_uart_abort_it ()

(+) Hal_uart_aborttransmit_it ()

(+) Hal_uart_abortreceive_it ()

(#) Для получения абортов на основе прерываний (HAL_UART_ABORTXXX_IT) предоставляется набор полных обратных вызовов:

(+) Hal_uart_abortcpltcallback ()

(+) Hal_uart_aborttransmitcpltcallback ()

(+) Hal_uart_abortreceivecpltcallback ()

(#) Обратный вызов приема событий RX (уведомление о событии RX) доступен для режимов Non_blocking Enhanced Reception:

(+) Hal_uartex_rxeventcallback ()

(#) В неблокирующих режим переноса возможные ошибки разделены на 2 категории.

Ошибки обрабатываются следующим образом:

(+) Ошибка считается восстанавливаемой и не блокирующей: передача может идти до конца, но серьезность ошибки - это

Чтобы оценить пользователь: это касается ошибки кадра, ошибки паритета или ошибок шума в приеме режима прерывания.

Полученный символ затем получают и сохраняются в буфере RX, код ошибки устанавливается для того, чтобы пользователь позволил пользователю идентифицировать тип ошибки,

и HAL_UART_ERRORCALLBACK () выполняется обратный вызов пользователя.Передача сохраняется на стороне UART.

Если пользователь хочет прервать его, пользователь должен вызвать услуги.

```
       (+) Ошибка рассматривается как блокировка: передача не может быть завершена
должным образом и прерывается.
           Это касается ошибок переполнения при приеме режима прерывания и всех ошибок
в режиме DMA.
           Код ошибки устанавливается, чтобы позволить пользователю идентифицировать
тип ошибки, и выполняется обратный вызов пользователя HAL_UART_ERRORCALLBACK ().
    -@- В полусвятном общении запрещено запускать передачу
        и процесс получения параллельно, государство UART HAL_UART_STATE_BUSY_TX_RX не
может быть полезно.
@endverbatim
  * @{
  */
/**
  * @brief отправляет объем данных в режиме блокировки.
  * @note, когда паритет UART не включен (PCE = 0), а длина слова настроена на 9 бит
(M1-M0 = 01),
  * Отправленные данные обрабатываются как набор U16.В этом случае размер должен
указывать число
  * u16 предоставлено через PDATA.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @param PDATA Указатель на буфер данных (элементы данных U8 или U16).
  * @param Размер размера элементов данных (U8 или U16), которые будут отправлены
  * @param Timeout Timeout
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, const uint8_t *pData,
uint16_t Size, uint32_t Timeout)
{
  const uint8_t  *pdata8bits;
  const uint16_t *pdata16bits;
  uint32_t tickstart = 0U;
  / * Убедитесь, что процесс TX еще не продолжается */
  if (huart->gState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
      return  HAL_ERROR;
    }
    / * Процесс заблокирован */
```

```
    __HAL_LOCK(huart);
    huart->ErrorCode = HAL_UART_ERROR_NONE;
    huart->gState = HAL_UART_STATE_BUSY_TX;
    / * Init tickstart для управления таймаутом */
    tickstart = HAL_GetTick();
    huart->TxXferSize = Size;
    huart->TxXferCount = Size;
    / * В случае 9BITS/NO PARITY TRANSFER, PDATA необходимо обрабатывать как указатель
UINT16_T */
    if ((huart->Init.WordLength == UART_WORDLENGTH_9B) && (huart->Init.Parity ==
UART_PARITY_NONE))
    {
      pdata8bits  = NULL;
      pdata16bits = (const uint16_t *) pData;
    }
    else
    {
      pdata8bits  = pData;
      pdata16bits = NULL;
    }
    / * Процесс разблокирован */
    __HAL_UNLOCK(huart);
    while (huart->TxXferCount > 0U)
    {
      if (UART_WaitOnFlagUntilTimeout(huart, UART_FLAG_TXE, RESET, tickstart,
Timeout) != HAL_OK)
      {
        return HAL_TIMEOUT;
      }
      if (pdata8bits == NULL)
      {
        huart->Instance->DR = (uint16_t)(*pdata16bits & 0x01FFU);
        pdata16bits++;
      }
      else
      {
        huart->Instance->DR = (uint8_t)(*pdata8bits & 0xFFU);
        pdata8bits++;
      }
      huart->TxXferCount--;
```

```
    }
    if (UART_WaitOnFlagUntilTimeout(huart, UART_FLAG_TC, RESET, tickstart, Timeout) !=
HAL_OK)
    {
      return HAL_TIMEOUT;
    }
    / * В конце процесса TX, восстановите Huart-> gstate, чтобы подготовить */
    huart->gState = HAL_UART_STATE_READY;
    return HAL_OK;
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @brief получает объем данных в режиме блокировки.
  * @note, когда паритет UART не включен (PCE = 0), а длина слова настроена на 9 бит
(M1-M0 = 01),
  * Полученные данные обрабатываются как набор U16.В этом случае размер должен
указывать число
  * u16 доступно через PDATA.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @param PDATA Указатель на буфер данных (элементы данных U8 или U16).
  * @param Размер размера элементов данных (U8 или U16), которые будут получены.
  * @param Timeout Timeout
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t
Size, uint32_t Timeout)
{
  uint8_t  *pdata8bits;
  uint16_t *pdata16bits;
  uint32_t tickstart = 0U;
  / * Убедитесь, что процесс RX еще не продолжается */
  if (huart->RxState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
```

```
      return  HAL_ERROR;
    }
    / * Процесс заблокирован */
    __HAL_LOCK(huart);
    huart->ErrorCode = HAL_UART_ERROR_NONE;
    huart->RxState = HAL_UART_STATE_BUSY_RX;
    huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
    / * Init tickstart для управления таймаутом */
    tickstart = HAL_GetTick();
    huart->RxXferSize = Size;
    huart->RxXferCount = Size;
    / * В случае 9BITS/NO PARITY TRANSFER, PRXDATA необходимо обрабатывать как
указатель UINT16_T */
    if ((huart->Init.WordLength == UART_WORDLENGTH_9B) && (huart->Init.Parity ==
UART_PARITY_NONE))
    {
      pdata8bits  = NULL;
      pdata16bits = (uint16_t *) pData;
    }
    else
    {
      pdata8bits  = pData;
      pdata16bits = NULL;
    }
    / * Процесс разблокирован */
    __HAL_UNLOCK(huart);
    / * Проверьте полученные данные остаться */
    while (huart->RxXferCount > 0U)
    {
      if (UART_WaitOnFlagUntilTimeout(huart, UART_FLAG_RXNE, RESET, tickstart, Timeout)
!= HAL_OK)
      {
        return HAL_TIMEOUT;
      }
      if (pdata8bits == NULL)
      {
        *pdata16bits = (uint16_t)(huart->Instance->DR & 0x01FF);
        pdata16bits++;
      }
      else
```

```
      {
        if ((huart->Init.WordLength == UART_WORDLENGTH_9B) || ((huart->Init.WordLength
== UART_WORDLENGTH_8B) && (huart->Init.Parity == UART_PARITY_NONE)))
        {
          *pdata8bits = (uint8_t)(huart->Instance->DR & (uint8_t)0x00FF);
        }
        else
        {
          *pdata8bits = (uint8_t)(huart->Instance->DR & (uint8_t)0x007F);
        }
        pdata8bits++;
      }
      huart->RxXferCount--;
    }
    / * В конце процесса RX, восстановите huart-> rxstate, чтобы подготовить */
    huart->RxState = HAL_UART_STATE_READY;
    return HAL_OK;
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @brief отправляет объем данных в режиме не блокировки.
  * @note, когда паритет UART не включен (PCE = 0), а длина слова настроена на 9 бит
(M1-M0 = 01),
  * Отправленные данные обрабатываются как набор U16.В этом случае размер должен
указывать число
  * u16 предоставлено через PDATA.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @param PDATA Указатель на буфер данных (элементы данных U8 или U16).
  * @param Размер размера элементов данных (U8 или U16), которые будут отправлены
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, const uint8_t *pData,
uint16_t Size)
{
  / * Убедитесь, что процесс TX еще не продолжается */
```

```
  if (huart->gState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
      return HAL_ERROR;
    }
    / * Процесс заблокирован */
    __HAL_LOCK(huart);
    huart->pTxBuffPtr = pData;
    huart->TxXferSize = Size;
    huart->TxXferCount = Size;
    huart->ErrorCode = HAL_UART_ERROR_NONE;
    huart->gState = HAL_UART_STATE_BUSY_TX;
    / * Процесс разблокирован */
    __HAL_UNLOCK(huart);
    / * Включить регистр данных передачи UART пустое прерывание */
    __HAL_UART_ENABLE_IT(huart, UART_IT_TXE);
    return HAL_OK;
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @brief получает объем данных в режиме не блокировки.
  * @note, когда паритет UART не включен (PCE = 0), а длина слова настроена на 9 бит
(M1-M0 = 01),
  * Полученные данные обрабатываются как набор U16.В этом случае размер должен
указывать число
  * u16 доступно через PDATA.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @param PDATA Указатель на буфер данных (элементы данных U8 или U16).
  * @param Размер размера элементов данных (U8 или U16), которые будут получены.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData,
uint16_t Size)
{
```

```
  / * Убедитесь, что процесс RX еще не продолжается */
  if (huart->RxState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
      return HAL_ERROR;
    }
    / * Процесс заблокирован */
    __HAL_LOCK(huart);
    / * Установите тип приема на стандартный прием */
    huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
    return (UART_Start_Receive_IT(huart, pData, Size));
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @brief отправляет объем данных в режиме DMA.
  * @note, когда паритет UART не включен (PCE = 0), а длина слова настроена на 9 бит
(M1-M0 = 01),
  * Отправленные данные обрабатываются как набор U16.В этом случае размер должен
указывать число
  * u16 предоставлено через PDATA.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @param PDATA Указатель на буфер данных (элементы данных U8 или U16).
  * @param Размер размера элементов данных (U8 или U16), которые будут отправлены
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart, const uint8_t
*pData, uint16_t Size)
{
  const uint32_t *tmp;
  / * Убедитесь, что процесс TX еще не продолжается */
  if (huart->gState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
```

```
      return HAL_ERROR;
    }
    / * Процесс заблокирован */
    __HAL_LOCK(huart);
    huart->pTxBuffPtr = pData;
    huart->TxXferSize = Size;
    huart->TxXferCount = Size;
    huart->ErrorCode = HAL_UART_ERROR_NONE;
    huart->gState = HAL_UART_STATE_BUSY_TX;
    / * Установите uart DMA Transfer Complete Callback */
    huart->hdmatx->XferCpltCallback = UART_DMATransmitCplt;
    / * Установите UART DMA Half Transfer Complete Callback */
    huart->hdmatx->XferHalfCpltCallback = UART_DMATxHalfCplt;
    / * Установите обратный вызов ошибки DMA */
    huart->hdmatx->XferErrorCallback = UART_DMAError;
    / * Установите обратный вызов DMA Abort */
    huart->hdmatx->XferAbortCallback = NULL;
    / * Включить uart передавать поток DMA */
    tmp = (const uint32_t *)&pData;
    HAL_DMA_Start_IT(huart->hdmatx, *(const uint32_t *)tmp, (uint32_t)&huart->Instance-
>DR, Size);
    / * Очистить флаг TC в реестре SR, написав 0 на него */
    __HAL_UART_CLEAR_FLAG(huart, UART_FLAG_TC);
    / * Процесс разблокирован */
    __HAL_UNLOCK(huart);
    /* Включить передачу DMA для запроса на передачу путем установки бита DMAT
       В регистре UART CR3 */
    ATOMIC_SET_BIT(huart->Instance->CR3, USART_CR3_DMAT);
    return HAL_OK;
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @Brief получает объем данных в режиме DMA.
  * @note, когда паритет UART не включен (PCE = 0), а длина слова настроена на 9 бит
(M1-M0 = 01),
```

```
  * Полученные данные обрабатываются как набор U16.В этом случае размер должен
указывать число
  * u16 доступно через PDATA.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @param PDATA Указатель на буфер данных (элементы данных U8 или U16).
  * @param Размер размера элементов данных (U8 или U16), которые будут получены.
  * @note Когда паритет UART включен (PCE = 1) полученные данные содержит бит паритета.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData,
uint16_t Size)
{
  / * Убедитесь, что процесс RX еще не продолжается */
  if (huart->RxState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
      return HAL_ERROR;
    }
    / * Процесс заблокирован */
    __HAL_LOCK(huart);
    / * Установите тип приема на стандартный прием */
    huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
    return (UART_Start_Receive_DMA(huart, pData, Size));
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @brief паузирует передачу DMA.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_DMAPause(UART_HandleTypeDef *huart)
{
  uint32_t dmarequest = 0x00U;
```

```
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  dmarequest = HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAT);
  if ((huart->gState == HAL_UART_STATE_BUSY_TX) && dmarequest)
  {
    / * Отключить запрос UART DMA TX */
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAT);
  }
  dmarequest = HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR);
  if ((huart->RxState == HAL_UART_STATE_BUSY_RX) && dmarequest)
  {
    / * Отключить rxne, pe и err (ошибка кадра, ошибка шума, ошибка переосмысления)
прерывает */
    ATOMIC_CLEAR_BIT(huart->Instance->CR1, USART_CR1_PEIE);
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
    / * Отключить запрос UART DMA RX */
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAR);
  }
  / * Процесс разблокирован */
  __HAL_UNLOCK(huart);
  return HAL_OK;
}
/**
  * @brief возобновляет передачу DMA.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_DMAResume(UART_HandleTypeDef *huart)
{
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  if (huart->gState == HAL_UART_STATE_BUSY_TX)
  {
    / * Включить запрос UART DMA TX */
    ATOMIC_SET_BIT(huart->Instance->CR3, USART_CR3_DMAT);
  }
  if (huart->RxState == HAL_UART_STATE_BUSY_RX)
  {
    /* Очистить флаг из перегрузки перед возобновлением переноса RX*/
```

```
    __HAL_UART_CLEAR_OREFLAG(huart);
    / * Повторно включено PE и ERR (ошибка кадра, ошибка шума, ошибка переосмысления)
прерывает */
    if (huart->Init.Parity != UART_PARITY_NONE)
    {
      ATOMIC_SET_BIT(huart->Instance->CR1, USART_CR1_PEIE);
    }
    ATOMIC_SET_BIT(huart->Instance->CR3, USART_CR3_EIE);
    / * Включить запрос UART DMA RX */
    ATOMIC_SET_BIT(huart->Instance->CR3, USART_CR3_DMAR);
  }
  / * Процесс разблокирован */
  __HAL_UNLOCK(huart);
  return HAL_OK;
}
/**
  * @brief останавливает передачу DMA.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_DMAStop(UART_HandleTypeDef *huart)
{
  uint32_t dmarequest = 0x00U;
  /* Замок не реализован в этом API, чтобы разрешить пользовательское приложение
     Чтобы вызвать API Hal Uart в соответствии с обратными вызовами
hal_uart_txcpltcallback () / hal_uart_rxcpltcallback ():
     При вызове HAL_DMA_ABORT () API DMA TX/RX TRANSFER COMPLEATE ARTRUPT создается
     и соответствующий вызов выполняется hal_uart_txcpltcallback () /
hal_uart_rxcpltcallback ()
     */
  / * ОСТАНОВИТЬ UART DMA TX Запрос, если он продолжается */
  dmarequest = HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAT);
  if ((huart->gState == HAL_UART_STATE_BUSY_TX) && dmarequest)
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAT);
    / * Прервать поток UART DMA TX */
    if (huart->hdmatx != NULL)
    {
      HAL_DMA_Abort(huart->hdmatx);
```

```
    }
    UART_EndTxTransfer(huart);
  }
  / * ОСТАНОВИТЬ UART DMA RX Запрос, если он продолжается */
  dmarequest = HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR);
  if ((huart->RxState == HAL_UART_STATE_BUSY_RX) && dmarequest)
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAR);
    / * Прервать поток UART DMA RX */
    if (huart->hdmarx != NULL)
    {
      HAL_DMA_Abort(huart->hdmarx);
    }
    UART_EndRxTransfer(huart);
  }
  return HAL_OK;
}
/**
  * @brief получает объем данных в режиме блокировки до тех пор, пока не будет получено ожидаемое количество данных, либо произойдет событие холостого хода.
  * @note hal_ok возвращается, если заполнен прием (ожидаемое количество данных было получено)
  * Или, если прием остановлен после события простоя (меньше, чем ожидаемое количество данных было получено)
  * В этом случае выходной параметр rxlen указывает количество данных, доступных в буфере регистрации.
  * @note, когда паритет UART не включен (PCE = 0), а длина слова настроена на 9 бит (M = 01),
  * Полученные данные обрабатываются как набор UINT16_T.В этом случае размер должен указывать число
  * of uint16_t доступен через PDATA.
  * @param huart uart handle.
  * @param pdata pointer на буфер данных (uint8_t или uint16_t элементы данных).
  * @param Размер размера элементов данных (uint8_t или uint16_t).
  * @param Rxlen Количество полученных элементов данных (может быть ниже, чем размер, в случае, если прием заканчивается на мероприятии на холостом ходу)
  * @Param Timeout Timeout Duration, выраженная в MS (охватывает всю последовательность приема).
  * @retval hal status
  */
```

```
HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle(UART_HandleTypeDef *huart, uint8_t *pData,
uint16_t Size, uint16_t *RxLen,
                                           uint32_t Timeout)
{
  uint8_t  *pdata8bits;
  uint16_t *pdata16bits;
  uint32_t tickstart;
  / * Убедитесь, что процесс RX еще не продолжается */
  if (huart->RxState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
      return  HAL_ERROR;
    }
    __HAL_LOCK(huart);
    huart->ErrorCode = HAL_UART_ERROR_NONE;
    huart->RxState = HAL_UART_STATE_BUSY_RX;
    huart->ReceptionType = HAL_UART_RECEPTION_TOIDLE;
    / * Init tickstart для управления таймаутом */
    tickstart = HAL_GetTick();
    huart->RxXferSize  = Size;
    huart->RxXferCount = Size;
    / * В случае 9BITS/NO PARITY TRANSFER, PRXDATA необходимо обрабатывать как
указатель UINT16_T */
    if ((huart->Init.WordLength == UART_WORDLENGTH_9B) && (huart->Init.Parity ==
UART_PARITY_NONE))
    {
      pdata8bits  = NULL;
      pdata16bits = (uint16_t *) pData;
    }
    else
    {
      pdata8bits  = pData;
      pdata16bits = NULL;
    }
    __HAL_UNLOCK(huart);
    / * Инициализировать выходной номер полученных элементов */
    *RxLen = 0U;
    / * Пока данные должны быть получены */
    while (huart->RxXferCount > 0U)
```

```
    {
      / * Проверьте, установлен ли флаг холостого хода */
      if (__HAL_UART_GET_FLAG(huart, UART_FLAG_IDLE))
      {
        / * Clear Idle Flag в ISR */
        __HAL_UART_CLEAR_IDLEFLAG(huart);
        / * Если установить, но не полученные данные, очистите флаг без выхода из петли
*/
        / * Если установлено, и данные уже получены, это означает, что событие IDLE
действителен: End Acreption */
        if (*RxLen > 0U)
        {
          huart->RxState = HAL_UART_STATE_READY;
          return HAL_OK;
        }
      }
      / * Проверьте, установлен ли флаг RXNE */
      if (__HAL_UART_GET_FLAG(huart, UART_FLAG_RXNE))
      {
        if (pdata8bits == NULL)
        {
          *pdata16bits = (uint16_t)(huart->Instance->DR & (uint16_t)0x01FF);
          pdata16bits++;
        }
        else
        {
          if ((huart->Init.WordLength == UART_WORDLENGTH_9B) || ((huart-
>Init.WordLength == UART_WORDLENGTH_8B) && (huart->Init.Parity == UART_PARITY_NONE)))
          {
            *pdata8bits = (uint8_t)(huart->Instance->DR & (uint8_t)0x00FF);
          }
          else
          {
            *pdata8bits = (uint8_t)(huart->Instance->DR & (uint8_t)0x007F);
          }
          pdata8bits++;
        }
        / * Увеличение количества полученных элементов */
        *RxLen += 1U;
        huart->RxXferCount--;
```

```
      }
      / * Проверьте время ожидания */
      if (Timeout != HAL_MAX_DELAY)
      {
        if (((HAL_GetTick() - tickstart) > Timeout) || (Timeout == 0U))
        {
          huart->RxState = HAL_UART_STATE_READY;
          return HAL_TIMEOUT;
        }
      }
    }
    / * Установите номер полученных элементов в выходном параметре: rxlen */
    *RxLen = huart->RxXferSize - huart->RxXferCount;
    / * В конце процесса RX, восстановите huart-> rxstate, чтобы подготовить */
    huart->RxState = HAL_UART_STATE_READY;
    return HAL_OK;
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @brief получает объем данных в режиме прерывания до тех пор, пока не будет получено
либо ожидаемое количество данных, либо произойдет событие холостого хода.
  * @note прием инициируется этим вызовом функции.Дальнейший прогресс приема
достигается спасибо
  * До прерывания UART, поднятые RXNE и IDLE Events.Обратный вызов вызывается в конце
приема, указывающего
  * Количество полученных элементов данных.
  * @note, когда паритет UART не включен (PCE = 0), а длина слова настроена на 9 бит (M
= 01),
  * Полученные данные обрабатываются как набор UINT16_T.В этом случае размер должен
указывать число
  * of uint16_t доступен через PDATA.
  * @param huart uart handle.
  * @param pdata pointer на буфер данных (uint8_t или uint16_t элементы данных).
  * @param Размер размера элементов данных (uint8_t или uint16_t).
  * @retval hal status
  */
```

```
HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_IT(UART_HandleTypeDef *huart, uint8_t
*pData, uint16_t Size)
{
  HAL_StatusTypeDef status;
  / * Убедитесь, что процесс RX еще не продолжается */
  if (huart->RxState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
      return HAL_ERROR;
    }
    __HAL_LOCK(huart);
    /* Установите тип приема на прием, пока не простаивает событие*/
    huart->ReceptionType = HAL_UART_RECEPTION_TOIDLE;
    status =  UART_Start_Receive_IT(huart, pData, Size);
    / * Процесс RX процесс был успешно начат */
    if (status == HAL_OK)
    {
      if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
      {
        __HAL_UART_CLEAR_IDLEFLAG(huart);
        ATOMIC_SET_BIT(huart->Instance->CR1, USART_CR1_IDLEIE);
      }
      else
      {
        /* В случае ошибок, уже ожидающих, когда начинается прием,
           Прерывания, возможно, уже были подняты и приводят к приемом аборта.
           (Например, ошибка перевернута).
           В таком случае прием был сброшен в HAL_UART_RECEPTION_STANDARD.*/
        status = HAL_ERROR;
      }
    }
    return status;
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
```

```
  * @brief получает объем данных в режиме DMA до тех пор, пока не будет получено либо ожидаемое количество данных, либо произойдет праздничное событие.
  * @note прием инициируется этим вызовом функции.Дальнейший прогресс приема достигается спасибо
  * в службы DMA, передача автоматически полученных элементов данных в буфере приема пользователей и
  * Позвоните в зарегистрированные обратные вызовы на половине/конец приема.События на холостом ходу также используются для рассмотрения
  * Фаза приема как закончилась.Во всех случаях выполнение обратного вызова будет указывать количество полученных элементов данных.
  * @note Когда паритет UART включен (PCE = 1), полученные данные содержат
  * Бит паритета (положение MSB).
  * @note, когда паритет UART не включен (PCE = 0), а длина слова настроена на 9 бит (M = 01),
  * Полученные данные обрабатываются как набор UINT16_T.В этом случае размер должен указывать число
  * of uint16_t доступен через PDATA.
  * @param huart uart handle.
  * @param pdata pointer на буфер данных (uint8_t или uint16_t элементы данных).
  * @param Размер размера элементов данных (uint8_t или uint16_t).
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)
{
  HAL_StatusTypeDef status;
  / * Убедитесь, что процесс RX еще не продолжается */
  if (huart->RxState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
      return HAL_ERROR;
    }
    __HAL_LOCK(huart);
    /* Установите тип приема на прием, пока не простаивает событие*/
    huart->ReceptionType = HAL_UART_RECEPTION_TOIDLE;
    status =  UART_Start_Receive_DMA(huart, pData, Size);
    / * Процесс RX процесс был успешно начат */
    if (status == HAL_OK)
    {
```

```
      if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
      {
        __HAL_UART_CLEAR_IDLEFLAG(huart);
        ATOMIC_SET_BIT(huart->Instance->CR1, USART_CR1_IDLEIE);
      }
      else
      {
        /* В случае ошибок, уже ожидающих, когда начинается прием,
           Прерывания, возможно, уже были подняты и приводят к приемом аборта.
           (Например, ошибка перевернута).
           В таком случае прием был сброшен в HAL_UART_RECEPTION_STANDARD.*/
        status = HAL_ERROR;
      }
    }
    return status;
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @brief прерывает постоянные переводы (режим блокировки).
  * @param huart uart handle.
  * @note Эта процедура может быть использована для прерывания любой постоянной
передачи, начатой в режиме прерывания или DMA.
  * Эта процедура выполняет следующие операции:
  * - Отключить прерывания UART (TX и RX)
  * - Отключить передачу DMA в периферийном регистре (если включено)
  * - прервать передачу DMA, позвонив в HAL_DMA_ABORT (в случае передачи в режиме DMA)
  * - Установите состояние ручки, чтобы подготовить
  * @note Эта процедура выполняется в режиме блокировки: при выходе на функцию прерван
считается завершенным.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_Abort(UART_HandleTypeDef *huart)
{
  / * Отключить txeie, tcie, rxne, pe и err (ошибка кадра, ошибка шума, ошибка
переиздания) прерывает */
```

```
  ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_RXNEIE | USART_CR1_PEIE |
USART_CR1_TXEIE | USART_CR1_TCIE));
  ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
  / * Если прием до холостого хода не было постоянным, отключить Idleie Interrupt */
  if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_IDLEIE));
  }
  / * Отключить запрос UART DMA TX, если включен */
  if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAT))
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAT);
    / * Прервать поток UART DMA TX: используйте блокировку API DMA ABORT (без обратного
вызова) */
    if (huart->hdmatx != NULL)
    {
      /* Установите обратный вызов UART DMA ABORT в NULL.
         Нет обратного выполнения вызовов в конце процедуры прервания DMA */
      huart->hdmatx->XferAbortCallback = NULL;
      if (HAL_DMA_Abort(huart->hdmatx) != HAL_OK)
      {
        if (HAL_DMA_GetError(huart->hdmatx) == HAL_DMA_ERROR_TIMEOUT)
        {
          / * Установить код ошибки в DMA */
          huart->ErrorCode = HAL_UART_ERROR_DMA;
          return HAL_TIMEOUT;
        }
      }
    }
  }
  / * Отключить запрос UART DMA RX, если включен */
  if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR))
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAR);
    / * Прерывать поток UART DMA RX: используйте блокировку API DMA ABORT (без
обратного вызова) */
    if (huart->hdmarx != NULL)
    {
      /* Установите обратный вызов UART DMA ABORT в NULL.
         Нет обратного выполнения вызовов в конце процедуры прервания DMA */
```

```
      huart->hdmarx->XferAbortCallback = NULL;
      if (HAL_DMA_Abort(huart->hdmarx) != HAL_OK)
      {
        if (HAL_DMA_GetError(huart->hdmarx) == HAL_DMA_ERROR_TIMEOUT)
        {
          / * Установить код ошибки в DMA */
          huart->ErrorCode = HAL_UART_ERROR_DMA;
          return HAL_TIMEOUT;
        }
      }
    }
  }
  / * Сбросить счетчики передачи TX и RX */
  huart->TxXferCount = 0x00U;
  huart->RxXferCount = 0x00U;
  / * Сбросить код ошибки */
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  / * Восстановить Huart-> rxstate и Huart-> gstate, чтобы готовить */
  huart->RxState = HAL_UART_STATE_READY;
  huart->gState = HAL_UART_STATE_READY;
  huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
  return HAL_OK;
}
/**
  * @Brief прерывает продолжающуюся передачу передачи (режим блокировки).
  * @param huart uart handle.
  * @note Эта процедура может быть использована для прерывания любого продолжающегося
передачи TX, начатой в режиме прерывания или DMA.
  * Эта процедура выполняет следующие операции:
  * - Отключить прерывания UART (TX)
  * - Отключить передачу DMA в периферийном регистре (если включено)
  * - прервать передачу DMA, позвонив в HAL_DMA_ABORT (в случае передачи в режиме DMA)
  * - Установите состояние ручки, чтобы подготовить
  * @note Эта процедура выполняется в режиме блокировки: при выходе на функцию прерван
считается завершенным.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_AbortTransmit(UART_HandleTypeDef *huart)
{
  / * Отключить прерывания txeie и tcie */
```

```
  ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_TXEIE | USART_CR1_TCIE));
  / * Отключить запрос UART DMA TX, если включен */
  if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAT))
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAT);
    / * Прервать поток UART DMA TX: используйте блокировку API DMA ABORT (без обратного
вызова) */
    if (huart->hdmatx != NULL)
    {
      /* Установите обратный вызов UART DMA ABORT в NULL.
         Нет обратного выполнения вызовов в конце процедуры прервания DMA */
      huart->hdmatx->XferAbortCallback = NULL;
      if (HAL_DMA_Abort(huart->hdmatx) != HAL_OK)
      {
        if (HAL_DMA_GetError(huart->hdmatx) == HAL_DMA_ERROR_TIMEOUT)
        {
          / * Установить код ошибки в DMA */
          huart->ErrorCode = HAL_UART_ERROR_DMA;
          return HAL_TIMEOUT;
        }
      }
    }
  }
  / * Сбросить счетчик передачи TX */
  huart->TxXferCount = 0x00U;
  / * Восстановить huart-> gstate, чтобы подготовиться */
  huart->gState = HAL_UART_STATE_READY;
  return HAL_OK;
}
/**
  * @brief претерпевает постоянную передачу (режим блокировки).
  * @param huart uart handle.
  * @note Эта процедура может быть использована для прерывания любого продолжающегося
переноса RX, начатой в режиме прерывания или DMA.
  * Эта процедура выполняет следующие операции:
  * - Отключить прерывания UART (RX)
  * - Отключить передачу DMA в периферийном регистре (если включено)
  * - прервать передачу DMA, позвонив в HAL_DMA_ABORT (в случае передачи в режиме DMA)
  * - Установите состояние ручки, чтобы подготовить
```

```
  * @note Эта процедура выполняется в режиме блокировки: при выходе на функцию прерван
считается завершенным.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_AbortReceive(UART_HandleTypeDef *huart)
{
  / * Отключить rxne, pe и err (ошибка кадра, ошибка шума, ошибка переосмысления)
прерывает */
  ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_RXNEIE | USART_CR1_PEIE));
  ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
  / * Если прием до холостого хода не было постоянным, отключить Idleie Interrupt */
  if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_IDLEIE));
  }
  / * Отключить запрос UART DMA RX, если включен */
  if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR))
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAR);
    / * Прерывать поток UART DMA RX: используйте блокировку API DMA ABORT (без
обратного вызова) */
    if (huart->hdmarx != NULL)
    {
      /* Установите обратный вызов UART DMA ABORT в NULL.
         Нет обратного выполнения вызовов в конце процедуры прервания DMA */
      huart->hdmarx->XferAbortCallback = NULL;
      if (HAL_DMA_Abort(huart->hdmarx) != HAL_OK)
      {
        if (HAL_DMA_GetError(huart->hdmarx) == HAL_DMA_ERROR_TIMEOUT)
        {
          / * Установить код ошибки в DMA */
          huart->ErrorCode = HAL_UART_ERROR_DMA;
          return HAL_TIMEOUT;
        }
      }
    }
  }
  / * Сбросить счетчик передачи rx */
  huart->RxXferCount = 0x00U;
  / * Восстановите Huart-> rxstate, чтобы подготовить */
```

```
  huart->RxState = HAL_UART_STATE_READY;
  huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
  return HAL_OK;
}
/**
  * @Brief прерывает текущие переводы (режим прерывания).
  * @param huart uart handle.
  * @note Эта процедура может быть использована для прерывания любой постоянной
передачи, начатой в режиме прерывания или DMA.
  * Эта процедура выполняет следующие операции:
  * - Отключить прерывания UART (TX и RX)
  * - Отключить передачу DMA в периферийном регистре (если включено)
  * - прервать передачу DMA, позвонив в HAL_DMA_ABORT_IT (в случае передачи в режиме
DMA)
  * - Установите состояние ручки, чтобы подготовить
  * - При завершении с прерванием, позвоните пользователю прервать полный обратный
вызов
  * @note Эта процедура выполняется в режиме прерывания, что означает, что процедура
прерывания может быть
  * считается завершенным только при выполнении полного обратного вызова пользователя
(не при выходе из функции).
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_Abort_IT(UART_HandleTypeDef *huart)
{
  uint32_t AbortCplt = 0x01U;
  / * Отключить txeie, tcie, rxne, pe и err (ошибка кадра, ошибка шума, ошибка
переиздания) прерывает */
  ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_RXNEIE | USART_CR1_PEIE |
USART_CR1_TXEIE | USART_CR1_TCIE));
  ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
  / * Если прием до холостого хода не было постоянным, отключить Idleie Interrupt */
  if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_IDLEIE));
  }
  /* Если ручки DMA TX и/или DMA RX связаны с ручкой UART, должны быть инициализированы
полные обратные вызовы DMA.
     Перед каким -либо вызовом к функциям DMA прервать */
  / * DMA TX Ручка действительна */
```

```
  if (huart->hdmatx != NULL)
  {
    /* Установите DMA Abort Complete Callback, если uart dma tx запрос, если включен.
       В противном случае установите его на null */
    if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAT))
    {
      huart->hdmatx->XferAbortCallback = UART_DMATxAbortCallback;
    }
    else
    {
      huart->hdmatx->XferAbortCallback = NULL;
    }
  }
  / * DMA RX Ручка действительна */
  if (huart->hdmarx != NULL)
  {
    /* Установите DMA Abort Complete Callback, если uart dma rx запрос, если включен.
       В противном случае установите его на null */
    if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR))
    {
      huart->hdmarx->XferAbortCallback = UART_DMARxAbortCallback;
    }
    else
    {
      huart->hdmarx->XferAbortCallback = NULL;
    }
  }
  / * Отключить запрос UART DMA TX, если включен */
  if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAT))
  {
    / * Отключить DMA TX на уровне UART */
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAT);
    / * Прерывание потока UART DMA TX: используйте не блокирующую API DMA ABORT
(обратный вызов) */
    if (huart->hdmatx != NULL)
    {
      /* UART TX DMA ABORT обратный вызов уже инициализирован:
         Приведет к вызову hal_uart_abortcpltcallback () в конце процедуры Abort DMA */
      / * Прервать dma tx */
      if (HAL_DMA_Abort_IT(huart->hdmatx) != HAL_OK)
```

```
      {
        huart->hdmatx->XferAbortCallback = NULL;
      }
      else
      {
        AbortCplt = 0x00U;
      }
    }
  }
  / * Отключить запрос UART DMA RX, если включен */
  if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR))
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAR);
    / * Прерывание потока UART DMA RX: используйте не блокирующую API DMA ABORT
(обратный вызов) */
    if (huart->hdmarx != NULL)
    {
      /* UART RX DMA ABORT обратный вызов уже инициализирован:
         Приведет к вызову hal_uart_abortcpltcallback () в конце процедуры Abort DMA */
      / * Прервать dma rx */
      if (HAL_DMA_Abort_IT(huart->hdmarx) != HAL_OK)
      {
        huart->hdmarx->XferAbortCallback = NULL;
        AbortCplt = 0x01U;
      }
      else
      {
        AbortCplt = 0x00U;
      }
    }
  }
  / * Если не требуется полное выполнение Callback Callback.
  if (AbortCplt == 0x01U)
  {
    / * Сбросить счетчики передачи TX и RX */
    huart->TxXferCount = 0x00U;
    huart->RxXferCount = 0x00U;
    / * Сбросить код ошибки */
    huart->ErrorCode = HAL_UART_ERROR_NONE;
    / * Восстановить Huart-> Gstate и Huart-> rxstate, чтобы подготовить */
```

```
    huart->gState  = HAL_UART_STATE_READY;
    huart->RxState = HAL_UART_STATE_READY;
    huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
    / * Поскольку DMA не нужно прервать, вызовите непосредственно пользователь,
прервать полный обратный вызов */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    / * Вызовите зарегистрированный Abort Complete Callback */
    huart->AbortCpltCallback(huart);
#else
    / * Назовите Legacy Shad Abort Complete Callback */
    HAL_UART_AbortCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
  }
  return HAL_OK;
}
/**
  * @Brief прерывать продолжающуюся передачу передачи (режим прерывания).
  * @param huart uart handle.
  * @note Эта процедура может быть использована для прерывания любого продолжающегося
передачи TX, начатой в режиме прерывания или DMA.
  * Эта процедура выполняет следующие операции:
  * - Отключить прерывания UART (TX)
  * - Отключить передачу DMA в периферийном регистре (если включено)
  * - прервать передачу DMA, позвонив в HAL_DMA_ABORT_IT (в случае передачи в режиме
DMA)
  * - Установите состояние ручки, чтобы подготовить
  * - При завершении с прерванием, позвоните пользователю прервать полный обратный
вызов
  * @note Эта процедура выполняется в режиме прерывания, что означает, что процедура
прерывания может быть
  * считается завершенным только при выполнении полного обратного вызова пользователя
(не при выходе из функции).
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_AbortTransmit_IT(UART_HandleTypeDef *huart)
{
  / * Отключить прерывания txeie и tcie */
  ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_TXEIE | USART_CR1_TCIE));
  / * Отключить запрос UART DMA TX, если включен */
  if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAT))
```

```
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAT);
    / * Прервать поток UART DMA TX: используйте блокировку API DMA ABORT (без обратного
вызова) */
    if (huart->hdmatx != NULL)
    {
      /* Установите обратный вызов UART DMA ABORT:
         Приведет к вызову hal_uart_abortcpltcallback () в конце процедуры Abort DMA */
      huart->hdmatx->XferAbortCallback = UART_DMATxOnlyAbortCallback;
      / * Прервать dma tx */
      if (HAL_DMA_Abort_IT(huart->hdmatx) != HAL_OK)
      {
        / * Вызов непосредственно huart-> hdmatx-> xferabortcallback функция в случае
ошибки */
        huart->hdmatx->XferAbortCallback(huart->hdmatx);
      }
    }
    else
    {
      / * Сбросить счетчик передачи TX */
      huart->TxXferCount = 0x00U;
      / * Восстановить huart-> gstate, чтобы подготовиться */
      huart->gState = HAL_UART_STATE_READY;
      / * Поскольку DMA не нужно прервать, вызовите непосредственно пользователь,
прервать полный обратный вызов */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
      / * Вызовите зарегистрированное прерванное передача полного обратного вызова */
      huart->AbortTransmitCpltCallback(huart);
#else
      / * Вызовите Legacy слабый прерван, передавайте полный обратный вызов */
      HAL_UART_AbortTransmitCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
    }
  }
  else
  {
    / * Сбросить счетчик передачи TX */
    huart->TxXferCount = 0x00U;
    / * Восстановить huart-> gstate, чтобы подготовиться */
    huart->gState = HAL_UART_STATE_READY;
```

```
    / * Поскольку DMA не нужно прервать, вызовите непосредственно пользователь,
прервать полный обратный вызов */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    / * Вызовите зарегистрированное прерванное передача полного обратного вызова */
    huart->AbortTransmitCpltCallback(huart);
#else
    / * Вызовите Legacy слабый прерван, передавайте полный обратный вызов */
    HAL_UART_AbortTransmitCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
  }
  return HAL_OK;
}
/**
  * @brief претерпевает постоянную передачу получения (режим прерывания).
  * @param huart uart handle.
  * @note Эта процедура может быть использована для прерывания любого продолжающегося
переноса RX, начатой в режиме прерывания или DMA.
  * Эта процедура выполняет следующие операции:
  * - Отключить прерывания UART (RX)
  * - Отключить передачу DMA в периферийном регистре (если включено)
  * - прервать передачу DMA, позвонив в HAL_DMA_ABORT_IT (в случае передачи в режиме
DMA)
  * - Установите состояние ручки, чтобы подготовить
  * - При завершении с прерванием, позвоните пользователю прервать полный обратный
вызов
  * @note Эта процедура выполняется в режиме прерывания, что означает, что процедура
прерывания может быть
  * считается завершенным только при выполнении полного обратного вызова пользователя
(не при выходе из функции).
  * @retval hal status
  */
HAL_StatusTypeDef HAL_UART_AbortReceive_IT(UART_HandleTypeDef *huart)
{
  / * Отключить rxne, pe и err (ошибка кадра, ошибка шума, ошибка переосмысления)
прерывает */
  ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_RXNEIE | USART_CR1_PEIE));
  ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
  / * Если прием до холостого хода не было постоянным, отключить Idleie Interrupt */
  if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
  {
```

```
    ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_IDLEIE));
  }
  / * Отключить запрос UART DMA RX, если включен */
  if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR))
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAR);
    / * Прерывать поток UART DMA RX: используйте блокировку API DMA ABORT (без
обратного вызова) */
    if (huart->hdmarx != NULL)
    {
      /* Установите обратный вызов UART DMA ABORT:
         Приведет к вызову hal_uart_abortcpltcallback () в конце процедуры Abort DMA */
      huart->hdmarx->XferAbortCallback = UART_DMARxOnlyAbortCallback;
      / * Прервать dma rx */
      if (HAL_DMA_Abort_IT(huart->hdmarx) != HAL_OK)
      {
        / * Вызов непосредственно huart-> hdmarx-> xferabortcallback функция в случае
ошибки */
        huart->hdmarx->XferAbortCallback(huart->hdmarx);
      }
    }
    else
    {
      / * Сбросить счетчик передачи rx */
      huart->RxXferCount = 0x00U;
      / * Восстановите Huart-> rxstate, чтобы подготовить */
      huart->RxState = HAL_UART_STATE_READY;
      huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
      / * Поскольку DMA не нужно прервать, вызовите непосредственно пользователь,
прервать полный обратный вызов */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
      / * Вызовите зарегистрированное Abort получить полный обратный вызов */
      huart->AbortReceiveCpltCallback(huart);
#else
      / * Вызовите Legacy Shad Abort получить полный обратный вызов */
      HAL_UART_AbortReceiveCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
    }
  }
  else
```

```
  {
    / * Сбросить счетчик передачи rx */
    huart->RxXferCount = 0x00U;
    / * Восстановите Huart-> rxstate, чтобы подготовить */
    huart->RxState = HAL_UART_STATE_READY;
    huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
    / * Поскольку DMA не нужно прервать, вызовите непосредственно пользователь,
прервать полный обратный вызов */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    / * Вызовите зарегистрированное Abort получить полный обратный вызов */
    huart->AbortReceiveCpltCallback(huart);
#else
    / * Вызовите Legacy Shad Abort получить полный обратный вызов */
    HAL_UART_AbortReceiveCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
  }
  return HAL_OK;
}
/**
  * @brief Эта функция обрабатывает запрос прерывания UART.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval нет
  */
void HAL_UART_IRQHandler(UART_HandleTypeDef *huart)
{
  uint32_t isrflags   = READ_REG(huart->Instance->SR);
  uint32_t cr1its     = READ_REG(huart->Instance->CR1);
  uint32_t cr3its     = READ_REG(huart->Instance->CR3);
  uint32_t errorflags = 0x00U;
  uint32_t dmarequest = 0x00U;
  / * Если ошибка не происходит */
  errorflags = (isrflags & (uint32_t)(USART_SR_PE | USART_SR_FE | USART_SR_ORE |
USART_SR_NE));
  if (errorflags == RESET)
  {
    /* UART в режиме приемника -----------------------------------------------------*/
    if (((isrflags & USART_SR_RXNE) != RESET) && ((cr1its & USART_CR1_RXNEIE) !=
RESET))
    {
```

```
      UART_Receive_IT(huart);
      return;
    }
  }
  / * Если возникают некоторые ошибки */
  if ((errorflags != RESET) && (((cr3its & USART_CR3_EIE) != RESET)
                                || ((cr1its & (USART_CR1_RXNEIE | USART_CR1_PEIE)) !=
RESET)))
  {
    /* Произошло прерывание ошибки паритета.
    if (((isrflags & USART_SR_PE) != RESET) && ((cr1its & USART_CR1_PEIE) != RESET))
    {
      huart->ErrorCode |= HAL_UART_ERROR_PE;
    }
    /* Произошло прерывание ошибок шума UART -----------------------------------*/
    if (((isrflags & USART_SR_NE) != RESET) && ((cr3its & USART_CR3_EIE) != RESET))
    {
      huart->ErrorCode |= HAL_UART_ERROR_NE;
    }
    /* Произошло прерывание ошибки кадра UART ----------------------------------*/
    if (((isrflags & USART_SR_FE) != RESET) && ((cr3its & USART_CR3_EIE) != RESET))
    {
      huart->ErrorCode |= HAL_UART_ERROR_FE;
    }
    /* Произошло прерывание UART -----------------------------------------*///
    if (((isrflags & USART_SR_ORE) != RESET) && (((cr1its & USART_CR1_RXNEIE) != RESET)
                                                 || ((cr3its & USART_CR3_EIE) !=
RESET)))
    {
      huart->ErrorCode |= HAL_UART_ERROR_ORE;
    }
    /* Вызовите функцию обратного вызова uart, если необходимо
--------------------------*//
    if (huart->ErrorCode != HAL_UART_ERROR_NONE)
    {
      /* UART в режиме приемника -----------------------------------------------*/
      if (((isrflags & USART_SR_RXNE) != RESET) && ((cr1its & USART_CR1_RXNEIE) !=
RESET))
      {
        UART_Receive_IT(huart);
```

```
      }
      /* Если возникает ошибка переполнения, или если какая -либо ошибка возникает в
приеме режима DMA,
         рассматривать ошибку как блокирование */
      dmarequest = HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR);
      if (((huart->ErrorCode & HAL_UART_ERROR_ORE) != RESET) || dmarequest)
      {
        /* Ошибка блокировки: передача прерывается
           Установите состояние UART, чтобы получить возможность снова начать процесс,
           Отключить прерывания RX и отключить запрос RX DMA, если он продолжается */
        UART_EndRxTransfer(huart);
        / * Отключить запрос UART DMA RX, если включен */
        if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR))
        {
          ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAR);
          / * Прервать поток UART DMA RX */
          if (huart->hdmarx != NULL)
          {
            /* Установите обратный вызов UART DMA ABORT:
               Приведет к вызову hal_uart_errorcallback () в конце процедуры Abort DMA
*/
            huart->hdmarx->XferAbortCallback = UART_DMAAbortOnError;
            if (HAL_DMA_Abort_IT(huart->hdmarx) != HAL_OK)
            {
              / * Вызов непосредственно xferabortCallback Функция в случае ошибки */
              huart->hdmarx->XferAbortCallback(huart->hdmarx);
            }
          }
          else
          {
            / * Вызов пользователя ошибки обратный вызов */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
            /*Вызовы зарегистрированный обратный вызов ошибки*/
            huart->ErrorCallback(huart);
#else
            /*Вызовите Legacy слабый обратный вызов ошибки*/
            HAL_UART_ErrorCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
          }
        }
```

```
        else
        {
          / * Вызов пользователя ошибки обратный вызов */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
          /*Вызовы зарегистрированный обратный вызов ошибки*/
          huart->ErrorCallback(huart);
#else
          /*Вызовите Legacy слабый обратный вызов ошибки*/
          HAL_UART_ErrorCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
        }
      }
      else
      {
        /* Ошибка без блокировки: передача может продолжаться.
           Ошибка уведомляется пользователю через обратный вызов ошибки пользователя */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Вызовы зарегистрированный обратный вызов ошибки*/
        huart->ErrorCallback(huart);
#else
        /*Вызовите Legacy слабый обратный вызов ошибки*/
        HAL_UART_ErrorCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
        huart->ErrorCode = HAL_UART_ERROR_NONE;
      }
    }
    return;
  } / * Конец, если возникает какая -то ошибка */
  /* Проверьте текущий режим приема:
     Если будет выбран прием до холостого хода: */
  if ((huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
      && ((isrflags & USART_SR_IDLE) != 0U)
      && ((cr1its & USART_SR_IDLE) != 0U))
  {
    __HAL_UART_CLEAR_IDLEFLAG(huart);
    / * Проверьте, включен ли режим DMA в UART */
    if (HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR))
    {
      / * Режим DMA включен */
```

```
      /* Проверьте полученную длину: если все ожидаемые данные получены, ничего не
делайте,
         (DMA CPLT будет вызван).
         В противном случае, если по крайней мере один данные уже получены, событие на
холостом ходу должно быть уведомлено пользователю */
      uint16_t nb_remaining_rx_data = (uint16_t) __HAL_DMA_GET_COUNTER(huart->hdmarx);
      if ((nb_remaining_rx_data > 0U)
          && (nb_remaining_rx_data < huart->RxXferSize))
      {
        / * Прием не завершен */
        huart->RxXferCount = nb_remaining_rx_data;
        /* В нормальном режиме, End DMA XFER и HAL UART RX Процесс*/
        if (huart->hdmarx->Init.Mode != DMA_CIRCULAR)
        {
          / * Отключить PE и ERR (ошибка кадра, ошибка шума, ошибка переосмысления)
прерывает */
          ATOMIC_CLEAR_BIT(huart->Instance->CR1, USART_CR1_PEIE);
          ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
          /* Отключить передачу DMA для запроса получателя, сбросив бит DMAR
             В регистре UART CR3 */
          ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAR);
          / * В конце процесса RX, восстановите huart-> rxstate, чтобы подготовить */
          huart->RxState = HAL_UART_STATE_READY;
          huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
          ATOMIC_CLEAR_BIT(huart->Instance->CR1, USART_CR1_IDLEIE);
          / * Полученные байты, поэтому не нужно, чтобы прервание не было немедленным
*/
          (void)HAL_DMA_Abort(huart->hdmarx);
        }
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Звоните зарегистрированный обратный вызов событий RX*/
        huart->RxEventCallback(huart, (huart->RxXferSize - huart->RxXferCount));
#else
        /*Вызовите Legacy Shad RX Callback*//
        HAL_UARTEx_RxEventCallback(huart, (huart->RxXferSize - huart->RxXferCount));
#endif / * Use_hal_uart_register_callbacks */
      }
      return;
    }
    else
```

```
    {
      / * Режим DMA не включен */
      /* Проверьте полученную длину: если все ожидаемые данные получены, ничего не
делайте.
         В противном случае, если по крайней мере один данные уже получены, событие на
холостом ходу должно быть уведомлено пользователю */
      uint16_t nb_rx_data = huart->RxXferSize - huart->RxXferCount;
      if ((huart->RxXferCount > 0U)
          && (nb_rx_data > 0U))
      {
        / * Отключить прерывание ошибок паритета UART и прерывания RXNE */
        ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_RXNEIE | USART_CR1_PEIE));
        / * Отключить прерывание ошибки UART: (ошибка кадра, ошибка шума, ошибка
переосмысления) */
        ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
        / * Rx процесс завершен, восстановить huart-> rxstate до готовности */
        huart->RxState = HAL_UART_STATE_READY;
        huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
        ATOMIC_CLEAR_BIT(huart->Instance->CR1, USART_CR1_IDLEIE);
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Вызовите зарегистрированный Rx Complete Callback*/
        huart->RxEventCallback(huart, nb_rx_data);
#else
        /*Вызовите Legacy Shad RX Callback*//
        HAL_UARTEx_RxEventCallback(huart, nb_rx_data);
#endif / * Use_hal_uart_register_callbacks */
      }
      return;
    }
  }
  /* UART в режиме передатчика ------------------------------------------------------*/
  if (((isrflags & USART_SR_TXE) != RESET) && ((cr1its & USART_CR1_TXEIE) != RESET))
  {
    UART_Transmit_IT(huart);
    return;
  }
  /* UART в режиме передатчика Конец ----------------------------------------------
*/
  if (((isrflags & USART_SR_TC) != RESET) && ((cr1its & USART_CR1_TCIE) != RESET))
  {
```

```
    UART_EndTransmit_IT(huart);
    return;
  }
}
/**
  * @Brief TX Transfer завершены обратные вызовы.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval нет
  */
__weak void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_TXCPLTCALLBACK может быть реализован в пользовательском файле
   */
}
/**
  * @Brief TX Half Transfer завершенные обратные вызовы.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval нет
  */
__weak void HAL_UART_TxHalfCpltCallback(UART_HandleTypeDef *huart)
{
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_TXHALFCPLTCALLBACK может быть реализован в пользовательском файле
   */
}
/**
  * @Brief RX Transfer завершенные обратные вызовы.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval нет
  */
__weak void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
```

```
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_RXCPLTCALLBACK может быть реализован в пользовательском файле
   */
}
/**
  * @Brief RX Half Transfer Завершенные обратные вызовы.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval нет
  */
__weak void HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef *huart)
{
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_RXHALFCPLTCALLBACK может быть реализован в пользовательском файле
   */
}
/**
  * @brief Uart Ошибка обратных вызовов.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval нет
  */
__weak void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_ERRORCALLBACK может быть реализован в пользовательском файле
   */
}
/**
  * @brief uart прервать полный обратный вызов.
  * @param huart uart handle.
  * @retval нет
  */
__weak void HAL_UART_AbortCpltCallback(UART_HandleTypeDef *huart)
```

```
{
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_ABORTCPLTCALLBACK может быть реализован в пользовательском файле.
   */
}
/**
  * @brief uart прервать полный обратный вызов.
  * @param huart uart handle.
  * @retval нет
  */
__weak void HAL_UART_AbortTransmitCpltCallback(UART_HandleTypeDef *huart)
{
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_ABORTTRANSTICPLTCALLBACK может быть реализован в пользовательском
файле.
   */
}
/**
  * @brief uart Abort Получите полный обратный вызов.
  * @param huart uart handle.
  * @retval нет
  */
__weak void HAL_UART_AbortReceiveCpltCallback(UART_HandleTypeDef *huart)
{
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UART_ABORTECEIVECPLTCALLBACK может быть реализован в пользовательском
файле.
   */
}
/**
  * @brief -прием Callback (уведомление о событии RX вызвано после использования
передовой службы регистрации).
  * @param huart uart
```

```
  * @param Размер Номер данных, доступных в буфере регистрации приложений (указывает на позицию в
  * Прием буфер, пока не доступны данные)
  * @retval нет
  */
__weak void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size)
{
  / * Предупреждение о компиляции неиспользованных аргументов
  UNUSED(huart);
  UNUSED(Size);
  /* Примечание: эта функция не должна быть изменена, когда требуется обратный вызов,
           HAL_UARTEX_RXEVENTCALLBACK может быть реализован в пользовательском файле.
   */
}
/**
  * @}
  */
/** @defgroup uart_exported_functions_group3 Функции периферического управления
  * @Brief UART управляющие функции
  *
@verbatim

==============================================================================
                 ##### Функции периферического управления #####

==============================================================================
  [..]
    Этот подраздел предоставляет набор функций, позволяющих управлять UART:
    (+) HAL_LIN_SENDBREAK () API может быть полезен для передачи символа разрыва.
    (+) Hal_multiprocessor_EntermuteMode () API может быть полезен для входа в режим UART в режиме Mute.
    (+) Hal_multiprocessor_exitmutemode () API может быть полезен для выхода из режима Mute Uart по программному обеспечению.
    (+) Hal_halfduplex_enableTransmitter () API, чтобы включить передатчик UART и отключить приемник UART в режиме половины дуплекса
    (+) Hal_halfduplex_enablereceiver () API, чтобы включить приемник UART и отключить передатчик UART в половине дуплексного режима
@endverbatim
```

```
  * @{
  */
/**
  * @brief передает персонажей разрыва.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_LIN_SendBreak(UART_HandleTypeDef *huart)
{
  / * Проверьте параметры */
  assert_param(IS_UART_INSTANCE(huart->Instance));
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  huart->gState = HAL_UART_STATE_BUSY;
  / * Отправить символы перерыва */
  ATOMIC_SET_BIT(huart->Instance->CR1, USART_CR1_SBK);
  huart->gState = HAL_UART_STATE_READY;
  / * Процесс разблокирован */
  __HAL_UNLOCK(huart);
  return HAL_OK;
}
/**
  * @brief входит в UART в режиме Mute.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_MultiProcessor_EnterMuteMode(UART_HandleTypeDef *huart)
{
  / * Проверьте параметры */
  assert_param(IS_UART_INSTANCE(huart->Instance));
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  huart->gState = HAL_UART_STATE_BUSY;
  / * Включить режим MUTE USART, установив бит RWU в регистре CR1 */
  ATOMIC_SET_BIT(huart->Instance->CR1, USART_CR1_RWU);
  huart->gState = HAL_UART_STATE_READY;
  / * Процесс разблокирован */
  __HAL_UNLOCK(huart);
```

```
  return HAL_OK;
}
/**
  * @brief выходит из режима Mute Uart: Программное обеспечение Wake Up.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_MultiProcessor_ExitMuteMode(UART_HandleTypeDef *huart)
{
  / * Проверьте параметры */
  assert_param(IS_UART_INSTANCE(huart->Instance));
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  huart->gState = HAL_UART_STATE_BUSY;
  / * Отключить режим MUTE USART, очистив бит RWU в регистре CR1 */
  ATOMIC_CLEAR_BIT(huart->Instance->CR1, USART_CR1_RWU);
  huart->gState = HAL_UART_STATE_READY;
  / * Процесс разблокирован */
  __HAL_UNLOCK(huart);
  return HAL_OK;
}
/**
  * @Brief позволяет передатчику UART и отключает приемник UART.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter(UART_HandleTypeDef *huart)
{
  uint32_t tmpreg = 0x00U;
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  huart->gState = HAL_UART_STATE_BUSY;
  /*---------------------------- USART CR1 Конфигурация -----------------------*/
  tmpreg = huart->Instance->CR1;
  / * Clear te и re bits */
  tmpreg &= (uint32_t)~((uint32_t)(USART_CR1_TE | USART_CR1_RE));
  / * Включить интерфейс передачи USART, установив бит TE в регистре USART CR1 */
  tmpreg |= (uint32_t)USART_CR1_TE;
```

```
  / * Напишите в USART CR1 */
  WRITE_REG(huart->Instance->CR1, (uint32_t)tmpreg);
  huart->gState = HAL_UART_STATE_READY;
  / * Процесс разблокирован */
  __HAL_UNLOCK(huart);
  return HAL_OK;
}
/**
  * @brief позволяет приемнику UART и отключает передатчик UART.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver(UART_HandleTypeDef *huart)
{
  uint32_t tmpreg = 0x00U;
  / * Процесс заблокирован */
  __HAL_LOCK(huart);
  huart->gState = HAL_UART_STATE_BUSY;
  /*---------------------------- USART CR1 Конфигурация -----------------------*/
  tmpreg = huart->Instance->CR1;
  / * Clear te и re bits */
  tmpreg &= (uint32_t)~((uint32_t)(USART_CR1_TE | USART_CR1_RE));
  / * Включить интерфейс приема USART, установив бит в регистре USART CR1 */
  tmpreg |= (uint32_t)USART_CR1_RE;
  / * Напишите в USART CR1 */
  WRITE_REG(huart->Instance->CR1, (uint32_t)tmpreg);
  huart->gState = HAL_UART_STATE_READY;
  / * Процесс разблокирован */
  __HAL_UNLOCK(huart);
  return HAL_OK;
}
/**
  * @}
  */
/** @defgroup uart_exported_functions_group4 Периферийное состояние и функции ошибок
  * @Brief UART CATE и функции ошибок
  *
@verbatim
```

```
==============================================================================
                 ##### Функции периферического состояния и ошибок #####
==============================================================================
 [..]
   Этот подраздел предоставляет набор функций, позволяющих вернуть состояние
   Процесс связи UART, обратные периферические ошибки произошли во время связи
   процесс
   (+) Hal_uart_getState () API может быть полезен для проверки времени выполнения
состояния периферийного устройства UART.
   (+) Hal_uart_geterror () Проверьте ошибки времени выполнения, которые могут
произойти во время связи.
@endverbatim
  * @{
  */
/**
  * @brief возвращает состояние UART.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval Hal State
  */
HAL_UART_StateTypeDef HAL_UART_GetState(UART_HandleTypeDef *huart)
{
  uint32_t temp1 = 0x00U, temp2 = 0x00U;
  temp1 = huart->gState;
  temp2 = huart->RxState;
  return (HAL_UART_StateTypeDef)(temp1 | temp2);
}
/**
  * @brief вернуть код ошибки UART
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного UART.
  * @retval uart код ошибки
  */
uint32_t HAL_UART_GetError(UART_HandleTypeDef *huart)
{
  return huart->ErrorCode;
```

```
}
/**
  * @}
  */
/**
  * @}
  */
/** @defgroup uart_private_functions uart частные функции
  * @{
  */
/**
  * @Brief инициализируйте обратные вызовы к значениям по умолчанию.
  * @param huart uart handle.
  * @retval нет
  */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
void UART_InitCallbacksToDefault(UART_HandleTypeDef *huart)
{
  / * Настройки обратного вызова UART */
  huart->TxHalfCpltCallback        = HAL_UART_TxHalfCpltCallback;        / * Legacy
Sleed txhalfcpltcallback */
  huart->TxCpltCallback            = HAL_UART_TxCpltCallback;            / * Legacy
слабый txcpltcallback */
  huart->RxHalfCpltCallback        = HAL_UART_RxHalfCpltCallback;        / * Legacy
слабый rxhalfcpltcallback */
  huart->RxCpltCallback            = HAL_UART_RxCpltCallback;            / * Legacy
слабый rxcpltcallback */
  huart->ErrorCallback             = HAL_UART_ErrorCallback;             / * Legacy
Shad ErryCallback */
  huart->AbortCpltCallback         = HAL_UART_AbortCpltCallback;         / * Legacy
Shad Abortcpltcallback */
  huart->AbortTransmitCpltCallback = HAL_UART_AbortTransmitCpltCallback; / * Legacy
Shad AbortTransmitcpltcallback */
  huart->AbortReceiveCpltCallback  = HAL_UART_AbortReceiveCpltCallback;  / * Legacy
Shad Abortreceivecpltcallback */
  huart->RxEventCallback           = HAL_UARTEx_RxEventCallback;         / * Legacy
слабый rxeventcallback */
}
#endif / * Use_hal_uart_register_callbacks */
/**
```

```
  * @brief dma uart Процесс передачи Полный обратный вызов.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
  */
static void UART_DMATransmitCplt(DMA_HandleTypeDef *hdma)
{
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
  /* Нормальный режим DMA*/
  if ((hdma->Instance->CR & DMA_SxCR_CIRC) == 0U)
  {
    huart->TxXferCount = 0x00U;
    /* Отключить передачу DMA для запроса на передачу путем настройки бита DMAT
       В регистре UART CR3 */
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAT);
    / * Включить UART передать полное прерывание */
    ATOMIC_SET_BIT(huart->Instance->CR1, USART_CR1_TCIE);
  }
  / * Круглый режим DMA */
  else
  {
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    /*Звоните зарегистрировано TX Complete Callback*/
    huart->TxCpltCallback(huart);
#else
    /*Вызовите Legacy Sleed TX Complete Callback*/
    HAL_UART_TxCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
  }
}
/**
  * @brief dma uart Процесс передачи Половина полного обратного вызова
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
  */
static void UART_DMATxHalfCplt(DMA_HandleTypeDef *hdma)
{
```

```
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
  /*Звоните зарегистрировано TX Complete Callback*/
  huart->TxHalfCpltCallback(huart);
#else
  /*Вызовите Legacy Sleed TX Complete Callback*/
  HAL_UART_TxHalfCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
}
/**
  * @brief dma uart
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
  */
static void UART_DMAReceiveCplt(DMA_HandleTypeDef *hdma)
{
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
  /* Нормальный режим DMA*/
  if ((hdma->Instance->CR & DMA_SxCR_CIRC) == 0U)
  {
    huart->RxXferCount = 0U;
    / * Отключить rxne, pe и err (ошибка кадра, ошибка шума, ошибка переосмысления)
прерывает */
    ATOMIC_CLEAR_BIT(huart->Instance->CR1, USART_CR1_PEIE);
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
    /* Отключить передачу DMA для запроса получателя, установив бит DMAR
       В регистре UART CR3 */
    ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_DMAR);
    / * В конце процесса RX, восстановите huart-> rxstate, чтобы подготовить */
    huart->RxState = HAL_UART_STATE_READY;
    / * Если прием до холостого хода не был выбран, отключить простоя прерывание */
    if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
    {
      ATOMIC_CLEAR_BIT(huart->Instance->CR1, USART_CR1_IDLEIE);
    }
  }
  /* Проверьте текущий режим приема:
```

```
     Если будет выбран прием до холостого хода: используйте обратный вызов события RX
*/
  if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
  {
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    /*Звоните зарегистрированный обратный вызов событий RX*/
    huart->RxEventCallback(huart, huart->RxXferSize);
#else
    /*Вызовите Legacy Shad RX Callback*//
    HAL_UARTEx_RxEventCallback(huart, huart->RxXferSize);
#endif / * Use_hal_uart_register_callbacks */
  }
  else
  {
    / * В других случаях: используйте RX полный обратный вызов */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    /*Вызовите зарегистрированный Rx Complete Callback*/
    huart->RxCpltCallback(huart);
#else
    /*Вызовите Legacy слабый RX полный обратный вызов*/
    HAL_UART_RxCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
  }
}
/**
  * @brief dma uart
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
  */
static void UART_DMARxHalfCplt(DMA_HandleTypeDef *hdma)
{
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
  /* Проверьте текущий режим приема:
     Если будет выбран прием до холостого хода: используйте обратный вызов события RX
*/
  if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
  {
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
```

```
    /*Звоните зарегистрированный обратный вызов событий RX*/
    huart->RxEventCallback(huart, huart->RxXferSize / 2U);
#else
    /*Вызовите Legacy Shad RX Callback*//
    HAL_UARTEx_RxEventCallback(huart, huart->RxXferSize / 2U);
#endif / * Use_hal_uart_register_callbacks */
  }
  else
  {
    / * В других случаях: используйте RX половину полного обратного вызова */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
    /*Вызовите зарегистрированный RX Половина полного обратного вызова*/
    huart->RxHalfCpltCallback(huart);
#else
    /*Вызовите Legacy слабый RX половина полного обратного вызова*/
    HAL_UART_RxHalfCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
  }
}
/**
  * @brief dma uart communication ошибка обратного вызова.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
  */
static void UART_DMAError(DMA_HandleTypeDef *hdma)
{
  uint32_t dmarequest = 0x00U;
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
  / * ОСТАНОВИТЬ UART DMA TX Запрос, если он продолжается */
  dmarequest = HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAT);
  if ((huart->gState == HAL_UART_STATE_BUSY_TX) && dmarequest)
  {
    huart->TxXferCount = 0x00U;
    UART_EndTxTransfer(huart);
  }
  / * ОСТАНОВИТЬ UART DMA RX Запрос, если он продолжается */
  dmarequest = HAL_IS_BIT_SET(huart->Instance->CR3, USART_CR3_DMAR);
  if ((huart->RxState == HAL_UART_STATE_BUSY_RX) && dmarequest)
```

```
  {
    huart->RxXferCount = 0x00U;
    UART_EndRxTransfer(huart);
  }
  huart->ErrorCode |= HAL_UART_ERROR_DMA;
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
  /*Вызовы зарегистрированный обратный вызов ошибки*/
  huart->ErrorCallback(huart);
#else
  /*Вызовите Legacy слабый обратный вызов ошибки*/
  HAL_UART_ErrorCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
}
/**
  * @brief Эта функция обрабатывает тайм -аут общения UART.Он ждет
  * пока флаг больше не будет в указанном статусе.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @param flag указывает флаг UART, чтобы проверить.
  * @param Статус фактический статус флага (установлен или сброс).
  * @param tickstart tite Start Value
  * @param Timeout Timeout
  * @retval hal status
  */
static HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout(UART_HandleTypeDef *huart,
uint32_t Flag, FlagStatus Status,
                                                     uint32_t Tickstart, uint32_t
Timeout)
{
  / * Подождите, пока флаг не установлен */
  while ((__HAL_UART_GET_FLAG(huart, Flag) ? SET : RESET) == Status)
  {
    / * Проверьте время ожидания */
    if (Timeout != HAL_MAX_DELAY)
    {
      if ((Timeout == 0U) || ((HAL_GetTick() - Tickstart) > Timeout))
      {
        / * Отключить txe, rxne, pe и err (ошибка кадра, ошибка шума, ошибка
переполнения). Прерывания для процесса прерывания */
```

```
        ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_RXNEIE | USART_CR1_PEIE |
USART_CR1_TXEIE));
        ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
        huart->gState  = HAL_UART_STATE_READY;
        huart->RxState = HAL_UART_STATE_READY;
        / * Процесс разблокирован */
        __HAL_UNLOCK(huart);
        return HAL_TIMEOUT;
      }
    }
  }
  return HAL_OK;
}
/**
  * @brief Start Recape в режиме прерывания в режиме прерывания.
  * @Note Эта функция может быть вызвана всеми API Hal UART, обеспечивающим прием в
режиме прерывания.
  * @note При вызове этой функции, достоверность параметров считается уже проверенным,
  * т.е. состояние RX, адрес буфера, ...
  * Uart -ручка предполагается как заблокированная.
  * @param huart uart handle.
  * @param PDATA Указатель на буфер данных (элементы данных U8 или U16).
  * @param Размер размера элементов данных (U8 или U16), которые будут получены.
  * @retval hal status
  */
HAL_StatusTypeDef UART_Start_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData,
uint16_t Size)
{
  huart->pRxBuffPtr = pData;
  huart->RxXferSize = Size;
  huart->RxXferCount = Size;
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  huart->RxState = HAL_UART_STATE_BUSY_RX;
  / * Процесс разблокирован */
  __HAL_UNLOCK(huart);
  if (huart->Init.Parity != UART_PARITY_NONE)
  {
    / * Включить прерывание ошибок паритета UART */
    __HAL_UART_ENABLE_IT(huart, UART_IT_PE);
  }
```

```
  / * Включить прерывание ошибки UART: (ошибка кадра, ошибка шума, ошибка
переосмысления) */
  __HAL_UART_ENABLE_IT(huart, UART_IT_ERR);
  / * Включить регистр данных UART, а не пустое прерывание */
  __HAL_UART_ENABLE_IT(huart, UART_IT_RXNE);
  return HAL_OK;
}
/**
  * @brief Start Ratement в режиме DMA.
  * @Note Эта функция может быть вызвана всеми API Hal UART, обеспечивающим прием в
режиме DMA.
  * @note При вызове этой функции, достоверность параметров считается уже проверенным,
  * т.е. состояние RX, адрес буфера, ...
  * Uart -ручка предполагается как заблокированная.
  * @param huart uart handle.
  * @param PDATA Указатель на буфер данных (элементы данных U8 или U16).
  * @param Размер размера элементов данных (U8 или U16), которые будут получены.
  * @retval hal status
  */
HAL_StatusTypeDef UART_Start_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData,
uint16_t Size)
{
  uint32_t *tmp;
  huart->pRxBuffPtr = pData;
  huart->RxXferSize = Size;
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  huart->RxState = HAL_UART_STATE_BUSY_RX;
  / * Установите uart DMA Transfer Complete Callback */
  huart->hdmarx->XferCpltCallback = UART_DMAReceiveCplt;
  / * Установите UART DMA Half Transfer Complete Callback */
  huart->hdmarx->XferHalfCpltCallback = UART_DMARxHalfCplt;
  / * Установите обратный вызов ошибки DMA */
  huart->hdmarx->XferErrorCallback = UART_DMAError;
  / * Установите обратный вызов DMA Abort */
  huart->hdmarx->XferAbortCallback = NULL;
  / * Включить поток DMA */
  tmp = (uint32_t *)&pData;
  HAL_DMA_Start_IT(huart->hdmarx, (uint32_t)&huart->Instance->DR, *(uint32_t *)tmp,
Size);
```

```
  / * Очистите флаг из переполнения непосредственно перед тем, как разрешить запрос DMA
RX: может быть обязательным для второй передачи */
  __HAL_UART_CLEAR_OREFLAG(huart);
  / * Процесс разблокирован */
  __HAL_UNLOCK(huart);
  if (huart->Init.Parity != UART_PARITY_NONE)
  {
    / * Включить прерывание ошибок паритета UART */
    ATOMIC_SET_BIT(huart->Instance->CR1, USART_CR1_PEIE);
  }
  / * Включить прерывание ошибки UART: (ошибка кадра, ошибка шума, ошибка
переосмысления) */
  ATOMIC_SET_BIT(huart->Instance->CR3, USART_CR3_EIE);
  /* Включить передачу DMA для запроса получателя, установив бит DMAR
  В регистре UART CR3 */
  ATOMIC_SET_BIT(huart->Instance->CR3, USART_CR3_DMAR);
  return HAL_OK;
}
/**
  * @brief заканчивается продолжающейся передачей TX на периферийном устройстве UART
(после обнаружения ошибок или завершения передачи).
  * @param huart uart handle.
  * @retval нет
  */
static void UART_EndTxTransfer(UART_HandleTypeDef *huart)
{
  / * Отключить прерывания txeie и tcie */
  ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_TXEIE | USART_CR1_TCIE));
  / * В конце процесса TX, восстановите Huart-> gstate, чтобы подготовить */
  huart->gState = HAL_UART_STATE_READY;
}
/**
  * @Brief End Comting Rx перенос на периферийном устройстве UART (после обнаружения
ошибок или завершения приема).
  * @param huart uart handle.
  * @retval нет
  */
static void UART_EndRxTransfer(UART_HandleTypeDef *huart)
{
```

```
  / * Отключить rxne, pe и err (ошибка кадра, ошибка шума, ошибка переосмысления)
прерывает */
  ATOMIC_CLEAR_BIT(huart->Instance->CR1, (USART_CR1_RXNEIE | USART_CR1_PEIE));
  ATOMIC_CLEAR_BIT(huart->Instance->CR3, USART_CR3_EIE);
  / * В случае приема в ожидании события холостого хода, отключите также источник
прерывания в холостом ходу *//
  if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
  {
    ATOMIC_CLEAR_BIT(huart->Instance->CR1, USART_CR1_IDLEIE);
  }
  / * В конце процесса RX, восстановите huart-> rxstate, чтобы подготовить */
  huart->RxState = HAL_UART_STATE_READY;
  huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
}
/**
  * @brief DMA UART Communication Abort Callback, когда инициируется HAL Services по
ошибке
  * (Должно быть вызвано в конце процедуры прервания DMA после возникновения ошибки).
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
  */
static void UART_DMAAbortOnError(DMA_HandleTypeDef *hdma)
{
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
  huart->RxXferCount = 0x00U;
  huart->TxXferCount = 0x00U;
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
  /*Вызовы зарегистрированный обратный вызов ошибки*/
  huart->ErrorCallback(huart);
#else
  /*Вызовите Legacy слабый обратный вызов ошибки*/
  HAL_UART_ErrorCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
}
/**
  * @brief DMA UART TX Communication Abort Callback, когда инициируется пользователем
  * (Следует вызвать в конце процедуры прервания DMA TX после запроса пользователя
прервать).
```

```
  * @note Когда этот обратный вызов выполняется, пользователь прерван, завершенный
вызов обратно, только если нет
  * Прервать все еще продолжается для ручки RX DMA.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
  */
static void UART_DMATxAbortCallback(DMA_HandleTypeDef *hdma)
{
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
  huart->hdmatx->XferAbortCallback = NULL;
  / * Проверьте, остается ли процесс отборки *//
  if (huart->hdmarx != NULL)
  {
    if (huart->hdmarx->XferAbortCallback != NULL)
    {
      return;
    }
  }
  / * Процесс прерваний не продолжается: все каналы DMA прерываются, позвоните
пользователю, прервать полный обратный вызов */
  huart->TxXferCount = 0x00U;
  huart->RxXferCount = 0x00U;
  / * Сбросить код ошибки */
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  / * Восстановить Huart-> Gstate и Huart-> rxstate, чтобы подготовить */
  huart->gState  = HAL_UART_STATE_READY;
  huart->RxState = HAL_UART_STATE_READY;
  huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
  / * Пользователь позвонит, прервись полным обратным вызовом */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
  / * Вызовите зарегистрированный Abort Complete Callback */
  huart->AbortCpltCallback(huart);
#else
  / * Назовите Legacy Shad Abort Complete Callback */
  HAL_UART_AbortCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
}
/**
```

```
  * @brief dma uart rx Communication Abort Callback, когда инициируется пользователем
  * (Следует вызвать в конце процедуры DMA RX Abort после запроса об изменении
пользователя).
  * @note Когда этот обратный вызов выполняется, пользователь прерван, завершенный
вызов обратно, только если нет
  * Прервать все еще продолжается для TX DMA Handle.
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
  */
static void UART_DMARxAbortCallback(DMA_HandleTypeDef *hdma)
{
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
  huart->hdmarx->XferAbortCallback = NULL;
  / * Проверьте, остается ли процесс отборки *//
  if (huart->hdmatx != NULL)
  {
    if (huart->hdmatx->XferAbortCallback != NULL)
    {
      return;
    }
  }
  / * Процесс прерваний не продолжается: все каналы DMA прерываются, позвоните
пользователю, прервать полный обратный вызов */
  huart->TxXferCount = 0x00U;
  huart->RxXferCount = 0x00U;
  / * Сбросить код ошибки */
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  / * Восстановить Huart-> Gstate и Huart-> rxstate, чтобы подготовить */
  huart->gState  = HAL_UART_STATE_READY;
  huart->RxState = HAL_UART_STATE_READY;
  huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
  / * Пользователь позвонит, прервись полным обратным вызовом */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
  / * Вызовите зарегистрированный Abort Complete Callback */
  huart->AbortCpltCallback(huart);
#else
  / * Назовите Legacy Shad Abort Complete Callback */
  HAL_UART_AbortCpltCallback(huart);
```

```
#endif / * Use_hal_uart_register_callbacks */
}
/**
  * @brief dma uart tx communication прервать обратный вызов, когда инициировался
пользователем по вызову в
  * Hal_uart_aborttransmit_it api (прервать только TX Transfer)
  * (Этот обратный вызов выполняется в конце процедуры прервания DMA TX после запроса
об изменении пользователя,
  * и приводит к тому, что пользователь TX Abort Complete Callback выполните).
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
  */
static void UART_DMATxOnlyAbortCallback(DMA_HandleTypeDef *hdma)
{
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
  huart->TxXferCount = 0x00U;
  / * Восстановить huart-> gstate, чтобы подготовиться */
  huart->gState = HAL_UART_STATE_READY;
  / * Пользователь позвонит, прервись полным обратным вызовом */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
  / * Вызовите зарегистрированное прерванное передача полного обратного вызова */
  huart->AbortTransmitCpltCallback(huart);
#else
  / * Вызовите Legacy слабый прерван, передавайте полный обратный вызов */
  HAL_UART_AbortTransmitCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
}
/**
  * @brief dma uart rx communication прервать обратный вызов, когда он инициирован
пользователем по вызову в
  * Hal_uart_abortreceive_it api (прервать только rx перенос)
  * (Этот обратный вызов выполняется в конце процедуры Abort DMA RX после запроса об
изменении пользователя,
  * и приводит к тому, что пользователь rx прерван в полном выполнении обратного
вызова).
  * @param hdma pointer на структуру dma_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля DMA.
  * @retval нет
```

```
  */
static void UART_DMARxOnlyAbortCallback(DMA_HandleTypeDef *hdma)
{
  UART_HandleTypeDef *huart = (UART_HandleTypeDef *)((DMA_HandleTypeDef *)hdma)-
>Parent;
  huart->RxXferCount = 0x00U;
  / * Восстановите Huart-> rxstate, чтобы подготовить */
  huart->RxState = HAL_UART_STATE_READY;
  huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
  / * Пользователь позвонит, прервись полным обратным вызовом */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
  / * Вызовите зарегистрированное Abort получить полный обратный вызов */
  huart->AbortReceiveCpltCallback(huart);
#else
  / * Вызовите Legacy Shad Abort получить полный обратный вызов */
  HAL_UART_AbortReceiveCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
}
/**
  * @brief отправляет объем данных в режиме не блокировки.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
static HAL_StatusTypeDef UART_Transmit_IT(UART_HandleTypeDef *huart)
{
  const uint16_t *tmp;
  / * Убедитесь, что процесс TX продолжается */
  if (huart->gState == HAL_UART_STATE_BUSY_TX)
  {
    if ((huart->Init.WordLength == UART_WORDLENGTH_9B) && (huart->Init.Parity ==
UART_PARITY_NONE))
    {
      tmp = (const uint16_t *) huart->pTxBuffPtr;
      huart->Instance->DR = (uint16_t)(*tmp & (uint16_t)0x01FF);
      huart->pTxBuffPtr += 2U;
    }
    else
    {
      huart->Instance->DR = (uint8_t)(*huart->pTxBuffPtr++ & (uint8_t)0x00FF);
```

```
    }
    if (--huart->TxXferCount == 0U)
    {
      / * Отключить регистр данных передачи UART пустое прерывание */
      __HAL_UART_DISABLE_IT(huart, UART_IT_TXE);
      / * Включить UART передать полное прерывание */
      __HAL_UART_ENABLE_IT(huart, UART_IT_TC);
    }
    return HAL_OK;
  }
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @brief завершает передачу в режиме без блокировки.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval hal status
  */
static HAL_StatusTypeDef UART_EndTransmit_IT(UART_HandleTypeDef *huart)
{
  / * Отключить UART передавать полное прерывание */
  __HAL_UART_DISABLE_IT(huart, UART_IT_TC);
  / * TX-процесс завершен, восстановить Huart-> gstate, чтобы подготовить */
  huart->gState = HAL_UART_STATE_READY;
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
  /*Звоните зарегистрировано TX Complete Callback*/
  huart->TxCpltCallback(huart);
#else
  /*Вызовите Legacy Sleed TX Complete Callback*/
  HAL_UART_TxCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
  return HAL_OK;
}
/**
  * @brief получает объем данных в режиме не блокировки
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
```

```
  * @retval hal status
  */
static HAL_StatusTypeDef UART_Receive_IT(UART_HandleTypeDef *huart)
{
  uint8_t  *pdata8bits;
  uint16_t *pdata16bits;
  / * Убедитесь, что процесс RX продолжается */
  if (huart->RxState == HAL_UART_STATE_BUSY_RX)
  {
    if ((huart->Init.WordLength == UART_WORDLENGTH_9B) && (huart->Init.Parity ==
UART_PARITY_NONE))
    {
      pdata8bits  = NULL;
      pdata16bits = (uint16_t *) huart->pRxBuffPtr;
      *pdata16bits = (uint16_t)(huart->Instance->DR & (uint16_t)0x01FF);
      huart->pRxBuffPtr += 2U;
    }
    else
    {
      pdata8bits = (uint8_t *) huart->pRxBuffPtr;
      pdata16bits  = NULL;
      if ((huart->Init.WordLength == UART_WORDLENGTH_9B) || ((huart->Init.WordLength ==
UART_WORDLENGTH_8B) && (huart->Init.Parity == UART_PARITY_NONE)))
      {
        *pdata8bits = (uint8_t)(huart->Instance->DR & (uint8_t)0x00FF);
      }
      else
      {
        *pdata8bits = (uint8_t)(huart->Instance->DR & (uint8_t)0x007F);
      }
      huart->pRxBuffPtr += 1U;
    }
    if (--huart->RxXferCount == 0U)
    {
      / * Отключить регистр данных UART, а не пустое прерывание */
      __HAL_UART_DISABLE_IT(huart, UART_IT_RXNE);
      / * Отключить прерывание ошибки паритета UART */
      __HAL_UART_DISABLE_IT(huart, UART_IT_PE);
      / * Отключить прерывание ошибки UART: (ошибка кадра, ошибка шума, ошибка
переосмысления) */
```

```
      __HAL_UART_DISABLE_IT(huart, UART_IT_ERR);
      / * Rx процесс завершен, восстановить huart-> rxstate до готовности */
      huart->RxState = HAL_UART_STATE_READY;
      /* Проверьте текущий режим приема:
         Если будет выбран прием до холостого хода: */
      if (huart->ReceptionType == HAL_UART_RECEPTION_TOIDLE)
      {
        / * Установите тип приема в стандартный */
        huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
        / * Отключить простоя прерывание */
        ATOMIC_CLEAR_BIT(huart->Instance->CR1, USART_CR1_IDLEIE);
        / * Проверьте, установлен ли флаг холостого хода */
        if (__HAL_UART_GET_FLAG(huart, UART_FLAG_IDLE))
        {
          / * Clear Idle Flag в ISR */
          __HAL_UART_CLEAR_IDLEFLAG(huart);
        }
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Звоните зарегистрированный обратный вызов событий RX*/
        huart->RxEventCallback(huart, huart->RxXferSize);
#else
        /*Вызовите Legacy Shad RX Callback*//
        HAL_UARTEx_RxEventCallback(huart, huart->RxXferSize);
#endif / * Use_hal_uart_register_callbacks */
      }
      else
      {
        / * Стандартный прием API под названием */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Вызовите зарегистрированный Rx Complete Callback*/
        huart->RxCpltCallback(huart);
#else
        /*Вызовите Legacy слабый RX полный обратный вызов*/
        HAL_UART_RxCpltCallback(huart);
#endif / * Use_hal_uart_register_callbacks */
      }
      return HAL_OK;
    }
    return HAL_OK;
  }
```

```
  else
  {
    return HAL_BUSY;
  }
}
/**
  * @Brief настраивает периферийное устройство UART.
  * @param huart указан на структуру uart_handletypedef, которая содержит
  * Информация о конфигурации для указанного модуля UART.
  * @retval нет
  */
static void UART_SetConfig(UART_HandleTypeDef *huart)
{
  uint32_t tmpreg;
  uint32_t pclk;
  / * Проверьте параметры */
  assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
  assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
  assert_param(IS_UART_PARITY(huart->Init.Parity));
  assert_param(IS_UART_MODE(huart->Init.Mode));
  /*---------------------------- USART COR2 Конфигурация ----------------------*/
  /* Настроить биты uart Stop: установить остановку [13:12] биты
     Согласно Huart-> init.stopbits значение */
  MODIFY_REG(huart->Instance->CR2, USART_CR2_STOP, huart->Init.StopBits);
  /*---------------------------- USART CR1 Конфигурация ----------------------*/
  /* Настройка длины слова, паритет и режима UART:
     Установите M-биты в соответствии с huart-> init.wordlength
     Установить биты PCE и PS в соответствии с Huart-> init.parity значение
     Установите TE и повторные биты в соответствии с значением Huart-> init.mode
     Установите более 8 бит в соответствии с Huart-> init.oversampling value */
  tmpreg = (uint32_t)huart->Init.WordLength | huart->Init.Parity | huart->Init.Mode |
huart->Init.OverSampling;
  MODIFY_REG(huart->Instance->CR1,
             (uint32_t)(USART_CR1_M | USART_CR1_PCE | USART_CR1_PS | USART_CR1_TE |
USART_CR1_RE | USART_CR1_OVER8),
             tmpreg);
  /*---------------------------- USART CR3 Конфигурация ----------------------*/
  / * Настройте UART HFC: установить биты CTSE и RTSE в соответствии с huart->
init.hwflowctl value */
```

```
  MODIFY_REG(huart->Instance->CR3, (USART_CR3_RTSE | USART_CR3_CTSE), huart-
>Init.HwFlowCtl);

#if defined(USART6) && defined(UART9) && defined(UART10)
    if ((huart->Instance == USART1) || (huart->Instance == USART6) || (huart->Instance
== UART9) || (huart->Instance == UART10))
    {
      pclk = HAL_RCC_GetPCLK2Freq();
    }
#elif defined(USART6)
    if ((huart->Instance == USART1) || (huart->Instance == USART6))
    {
      pclk = HAL_RCC_GetPCLK2Freq();
    }
#else
    if (huart->Instance == USART1)
    {
      pclk = HAL_RCC_GetPCLK2Freq();
    }
#endif / * Usart6 */
    else
    {
      pclk = HAL_RCC_GetPCLK1Freq();
    }
  /*-------------------------- USART BRR Конфигурация --------------------*/
  if (huart->Init.OverSampling == UART_OVERSAMPLING_8)
  {
    huart->Instance->BRR = UART_BRR_SAMPLING8(pclk, huart->Init.BaudRate);
  }
  else
  {
    huart->Instance->BRR = UART_BRR_SAMPLING16(pclk, huart->Init.BaudRate);
  }
}
/**
  * @}
  */
#endif / * Hal_uart_module_enabled */
/**
  * @}
```

```
  */
/**
  * @}
  */
```