# **Описание используемых типов**

## **UART\_type**

|  |  |
| --- | --- |
| **Имя поля** | **Описание** |
| DR | Регистр данных |
| RXSTAT | Регистр состояния приемника |
| UCR\_LOAD | Регистр управления загрузка |
| UCR\_SET | Регистр управления установка |
| UCR\_CLEAR | Регистр управления сброс |
| UBitRate | Регистр управления скоростью обмена |
| UFLAG | Регистр флагов |
| UINTM | Регистр запросов прерываний (маскированный) |
| UINT | Регистр запросов прерываний (немаскированный) |

## **UART\_Init\_type**

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Описание** | **Значения** |
| BitRate | скорость работы интерфейса в битах | Скорость работы интерфейса. Ограничения:  максимум - XTI/4  (при OverSampling = *UART\_OverSampling\_normal*)  максимум - XTI/16  (при OverSampling = *UART\_OverSampling\_high*) |
| WorkMode | режим работы (передатчик, приемник, трансивер) | *UART\_Mode\_Tx* – UART инициализирован только на передачу  *UART\_Mode\_Rx* – UART инициализирован только на прием  *UART\_Mode\_TxRx* – UART инициализирован и на прием и на передачу |
| OverSampling | Количество синхросигналов на один бит информации | *UART\_OverSampling\_normal* – 4 такта синхросигнала для приема/передачи одного бита данных  *UART\_OverSampling\_high* – 16 тактов синхросигнала для приема/передачи одного бита данных. Этот режим обеспечивает надежный прием в условиях помех. |
| WordLength | Длина передаваемых данных | *UART\_WordLen\_5bit* – длина посылки 5 бит  *UART\_WordLen\_6bit* – длина посылки 6 бит  *UART\_WordLen\_7bit* – длина посылки 7 бит  *UART\_WordLen\_8bit* – длина посылки 8 бит  Позволяют запрограммировать количество реальных информационных бит в посылке. Таким образом, посылка может быть длиной от 7 бит (старт, 5 бит данных, стоп) до 12 бит (старт, 8 бит данных, бит четности и два стоповых). |
| StopBits | Количество стоп-битов | *UART\_Stop\_1bit* – Один стоп-бит  *UART\_Stop\_2bit* – Два стоп-бита  Это поле используется только передатчиком. Приемник не проверяет количество принятых стоп-битов больше одного. Увеличение количества стоп-битов до двух иногда полезно в случаях, когда существует расхождение в скорости обмена передающей и принимающей сторон. |
| ParityMode | Управление режимом проверки четности | *UART\_Parity\_even* – дополнение посылки до четного  *UART\_Parity\_odd –* дополнение посылки до четного  *UART\_Parity\_off –* проверка четности не выполняется  Позволяет задать тип проверки: четное или нечетное количество единичных бит в посылке. Если этот поле равно even, дополнительный бит четности дополняет количество единичных бит в посылке до четного количества. К примеру, если передается значение 0х15, бит четности равен 1.  Если передается значение 0х55, то бит четности равен 0. |
| FIFOSize | Размер FIFO приемника и передатчика | *UART\_FIFOSize\_Byte* – размер FIFO равняет байту (8 бит)  *UART\_FIFOSize\_8Byte*– размер FIFO равняет двойному слову (64 бита) |
| TXDMode | Управление режимом работы линии TXD | *UART\_TXD\_low* – на выходе линии данных всегда 0  *UART\_TXD\_high* – на выходе линии данных всегда 1  *UART\_TXD\_direct* – передача данных  *UART\_TXD\_compliment* – передача инверсных данных |
| DMACtrlErr | Управление запросом к контроллеру DMA во время обнаружения ошибки приема | Отслеживание ошибки по DMA:  *UART\_DMACtrl\_dis* – запрещено  *UART\_DMACtrl\_en* – разрешено  Работа в состоянии “ UART\_DMACtrl\_en ”:  В случае если при приеме данных возникнет ошибка, произойдет блокировка запроса к контроллеру DMA, что исключит прием ошибочных данных |

# **Описание используемых функций**

## **HAL\_UART\_Init**

**Резюме**

uint32\_t **HAL\_UART\_Init**( UART\_type\* UARTx, UART\_Init\_type\* InitStruct )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

UART\_Init\_Type\* InitStruct – указатель на структуру инициализации

**Описание**

Функция **HAL\_UART\_Init** заполняет регистры UART0/UART1 в соответствии с заданными параметрами структуры типа *UART\_Init\_type.* Обращаю внимание на то, что в файле hal\_uart.h два указателя на структуры типа *UART\_type* уже объявлены в полях регистров UART0/UART1, и называются LX\_UART0 и LX\_UART1. В случае обнаружении ошибки при инициализации, функция возвращает код ошибки.

|  |  |
| --- | --- |
| **Код ошибки** | **Описание** |
| 0 | Ошибок нет. Инициализация успешна. |
| 1 | Неверный UART\_type\*. Используйте уже объявленные указатели на структуры - UART0 или UART1. |
| 2 | BitRate не задан (равен 0). |
| 3 | BitRate превышает максимальную скорость UART. |
| 4 | BitRate превышает максимальную скорость UART для данного OverSampling. |

**Пример**

// Инициализация UART0

**#include <hal\_1967VN044.h>**

**void** **main**(**void**)

{

**int** errFlag = 0;

UART\_Init\_type UARTInit;

UARTInit.BitRate = 10000;

UARTInit.WorkMode = *UART\_Mode\_TxRx*;

UARTInit.OverSampling = *UART\_OverSampling\_High*;

UARTInit.WordLength = *UART\_WordLen\_8bit*;

UARTInit.StopBits = *UART\_Stop\_1bit*;

UARTInit.ParityMode = *UART\_Parity\_Even*;

UARTInit.FIFOSize = *UART\_FIFOSize\_DWord*;

UARTInit.TXDMode = *UART\_TXD\_Direct*;

UARTInit.DMACtrlErr = *UART\_DMACtrl\_Dis*;

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_x* | *GPIO\_PIN\_x*, *GPIO\_PinMode\_Alt*); // GPIO\_PIN\_x, x – зависит от выбранного UART

errFlag = HAL\_UART\_Init (LX\_UART0, &UARTInit);

}

## **HAL\_UART\_DefaultInitStruct**

**Резюме**

**void** **HAL\_UART\_DefaultInitStruct**( UART\_Init\_type\* InitStruct )

UART\_Init\_Type\* InitStruct – указатель на структуру инициализации

**Описание**

Функция **HAL\_UART\_Init** заполняет поля структуры типа *UART\_Init\_type* значениями по умолчанию. Если значениями по умолчанию необходимо заполнить лишь некоторые поля, то после вызова данный функции можно переинициализировать нужные поля вручную и затем вызвать функцию **HAL\_UART\_Init** (UART\_type\* UARTx, UART\_Init\_type\* InitStruct), для включения UART интерфейса.

|  |  |
| --- | --- |
| **Имя поля** | **Значение** |
| BitRate | 9600 |
| WorkMode | UART\_Mode\_TxRx |
| OverSampling | UART\_OverSampling\_normal |
| WordLength | UART\_WordLen\_8bit |
| StopBits | UART\_Stop\_1bit |
| ParityMode | UART\_Parity\_off |
| FIFOSize | UART\_FIFOSize\_8byte |
| TXDMode | UART\_TXD\_direct |
| DMACtrlErr | UART\_DMACtrl\_dis |

**Пример**

**#include <hal\_1967VN044.h>**

**void** **main**(**void**)

{

**int** errFlag = 0;

UART\_Init\_type UARTInit;

UART\_DefaultInit (&UARTInit);

errFlag = HAL\_UART\_Init (LX\_UART0, &UARTInit);

}

## **HAL\_UART\_DeInit**

**Резюме**

**void** **HAL\_UART\_DeInit**( UART\_type\* UARTx )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

**Описание**

Функция **HAL\_UART\_DeInit** записывает ноль в три поля структуры UARTx – UBitRate, UINT, UCR.

**Пример**

**#include <hal\_1967VN044.h>**

**void** **main**(**void**)

{

**int** errFlag = 0;

HAL\_UART\_DeInit (LX\_UART0);

}

## **HAL\_UART\_ITConfig**

**Резюме**

**void HAL\_UART\_ITConfig**( UART\_type\* UARTx, uint32\_t IT, UART\_IT\_type IT\_State )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

uint32\_t IT – прерывания

UART\_IT\_type IT\_State – состояние прерываний

**Описание**

Функция включает или выключает требуемые запросы прерываний от UART.

Параметр IT может состоять из:

UART\_IT\_TX – Запрос прерывания от передатчика

UART\_IT\_RX – Запрос прерывания от приемника

UART\_IT\_RXERR – Ошибка приема во время работы с контроллером DMA

UART\_IT\_MS – Запрос прерывания от модема

UART\_IT\_UD – Запрос прерывания от приемника в состоянии «выключен»

UART\_IT\_UTXE – Запрос прерывания от буфера передатчика при отсутствии в нем данных

UART\_IT\_URXT – Запрос прерывания от приемника при обнаружении ситуации “time-out”

Параметр IT\_State может быть *UART\_IT\_Dis* или *UART\_IT\_En.*

**Пример**

**#include <hal\_1967VN044.h>**

**void** **main**(**void**)

{

HAL\_UART\_ITConfig(LX\_UART0, UART\_IT\_TX | UART\_IT\_RX, *UART\_IT\_En*);

}

## **HAL\_UART\_ITEnable**

**Резюме**

**void HAL\_UART\_ITEnable**( UART\_type\* )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

**Описание**

Функция включает прерывание от модуля UART и задает встроенную функцию обработчик прерывания.

**Пример**

**#include <hal\_1967VN044.h>**

**void** **main**(**void**)

{

HAL\_UART\_ITConfig(LX\_UART0, UART\_IT\_TX | UART\_IT\_RX, *UART\_IT\_En*);

HAL\_UART\_ITEnable(LX\_UART0);

}

## **HAL\_UART\_ITDisable**

**Резюме**

**void HAL\_UART\_ITDisable**( UART\_type\* )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

**Описание**

Функция выключает прерывание от UART.

**Пример**

**#include <hal\_1967VN044.h>**

**void** **main**(**void**)

{

HAL\_UART\_ITConfig(LX\_UART0, UART\_IT\_TX | UART\_IT\_RX, *UART\_IT\_Dis*);

HAL\_UART\_ITDisable(LX\_UART0);

}

## **HAL\_UART\_Send**

**Резюме**

uint32\_t **HAL\_UART\_Send**( UART\_type\* UARTx, uint8\_t\* BuffAddr, uint32\_t amount )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

uint8\_t\* BuffAddr – указатель на буфер с данными для передачи

uint32\_t amount – количество передаваемых данных

**Описание**

Функция **HAL\_UART\_Send** выполняют передачу данных по UARTx. Возвращает код ошибки. Обращаю внимание на то, что в функции не происходит отслеживание конца передачи (то есть бит UBUSY регистра UFLAG не анализируется).

|  |  |
| --- | --- |
| **Код ошибки** | **Описание** |
| 0 | Ошибок нет. Передача выполнена успешно. |
| 1 | Неверный UART\_type\*. Используйте уже объявленные указатели на структуры - UART0 или UART1. |

**Пример**

**#include <hal\_1967VN044.h>**

**#define N 1024**

**char** data\_tx[N];

**void** **main**(**void**)

{

**int** errFlag = 0;

UART\_Init\_type UARTInit;

HAL\_UART\_DefaultInitStruct (&UARTInit);

errFlag = HAL\_UART\_Init (LX\_UART0, &UARTInit);

errFlag = HAL\_UART\_Send (LX\_UART0, &data\_tx, N);

}

## **HAL\_UART\_DmaSend**

**Резюме**

**void HAL\_UART\_DmaSend**( UART\_type\* UARTx, uint32\_t Channel, **void**\* BuffAddr, uint32\_t Lenght, **void** \*DmaIsrHandler )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

uint32\_t Channel – используемый канал DMA (4-7)

uint8\_t\* BuffAddr – указатель на буфер с данными для передачи

uint32\_t Lenght – количество передаваемых данных

**void** \*DmaIsrHandler – функция обработчик прерывания, если не нужна – передаем 0

**Описание**

Функция инициализирует передачу данных через UART с загрузкой передаваемых данных в буфер UART по DMA.

**Ограничения**

Функция работает корректно только для char = 32, либо для других 32 разрядных типов.

**Пример**

**#include <hal\_1967VN044.h>**

**#define N 1024**

**int** data\_tx[N];

**void** **main**(**void**)

{

**int** errFlag = 0, errDMA;

UART\_Init\_type UARTInit;

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_0* | *GPIO\_PIN\_1*, *GPIO\_PinMode\_Alt*);

errFlag = HAL\_UART\_Init (LX\_UART0, &UARTInit);

//Send and Receive data

HAL\_UART\_DmaSend(LX\_UART0, 4, data\_tx, N, 0);

errDMA = HAL\_DMA\_WaitForChannel(4);

**if** (errDMA) error();

}

## **HAL\_UART\_DmaReceive**

**Резюме**

**void HAL\_UART\_DmaReceive**( UART\_type\* UARTx, uint32\_t Channel, **void**\* BuffAddr, uint32\_t Lenght, **void** \*DmaIsrHandler )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

uint32\_t Channel – используемый канал DMA (8-11)

uint8\_t\* BuffAddr – указатель на приемный буфер

uint32\_t Lenght – количество принимаемых данных

**void** \*DmaIsrHandler – функция обработчик прерывания, если не нужна – передаем 0

**Описание**

Функция инициализирует прием данных через UART с загрузкой принимаемых данных из буфера UART по DMA.

**Ограничения**

Функция работает корректно только для char = 32, либо для других 32 разрядных типов.

**Пример**

**#include <hal\_1967VN044.h>**

**#define N 1024**

**int** data\_rx[N], data\_tx[N];

**void** **main**(**void**)

{

**int** errFlag = 0, errDMA;

UART\_Init\_type UARTInit;

//UART Enable

HAL\_UART\_DefaultInitStruct(&UARTInit);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_0* | *GPIO\_PIN\_1*, *GPIO\_PinMode\_Alt*);

errFlag = HAL\_UART\_Init (LX\_UART0, &UARTInit);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_2* | *GPIO\_PIN\_3*, *GPIO\_PinMode\_Alt*);

errFlag = HAL\_UART\_Init (LX\_UART1, &UARTInit);

//Send and Receive data

HAL\_UART\_DmaReceive(LX\_UART1, 8, data\_rx, N, 0);

HAL\_UART\_DmaSend(LX\_UART0, 4, data\_tx, N, 0);

errDMA = HAL\_DMA\_WaitForChannel(4);

**if** (errDMA) error();

errDMA = HAL\_DMA\_WaitForChannel(8);

**if** (errDMA) error();

//while (k!=N);

**for** (i=0;i<N;i++){

**if** (data\_tx[i] != data\_rx[i]) errCount++;

}

HAL\_UART\_DeInit(LX\_UART0);

HAL\_UART\_DeInit(LX\_UART1);

}

## **HAL\_UART\_ITSend**

**Резюме**

**void HAL\_UART\_ITSend**( UART\_type\* UARTx, **void**\* BuffAddr, uint32\_t Length )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

uint8\_t\* BuffAddr – указатель на буфер с данными для передачи

uint32\_t Lenght – количество передаваемых данных

**Описание**

Функция инициализирует передачу данных через UART с использованием прерываний. Состояние передачи находится в структуре UART\_TxRxStatus\_type:

**typedef** **struct**

{

uint8\_t TxReady[ *UART\_NUMS* ]; // Все данные переданы

uint8\_t RxReady[ *UART\_NUMS* ]; // Приняты новые данные

uint32\_t RxLenght[ *UART\_NUMS* ]; // Количество принятых данных

} UART\_TxRxStatus\_type;

**Пример**

**#include** <hal\_1967VN044.h>

uint8\_t ucTestText[ ] = { 'H', 'e', 'l', 'l', 'o', ' ', 'f', 'r', 'o', 'm', ' ', '1', '9', '6', '7', 'V', 'N', '0', '3', '4', 0x0D, 0x0A };

**void** **main**(**void**)

{

UART\_Init\_type UARTInit;

HAL\_UART\_DefaultInitStruct( &UARTInit );

HAL\_GPIO\_Init( LX\_GPIO\_PA, GPIO\_PIN\_2 | GPIO\_PIN\_3, GPIO\_PinMode\_Alt );

HAL\_UART\_Init( LX\_UART1, &UARTInit );

HAL\_UART\_ITEnable( LX\_UART1 );

HAL\_Interrupt\_GlobalEnable();

**while**( 1 )

{

HAL\_UART\_ITSend( LX\_UART1, ucTestText, 22 );

**while**( !UartTxRxStatus.TxReady[ *UART1* ] );

}

}

## **HAL\_UART\_ITReceive**

**Резюме**

**void HAL\_UART\_ITReceive**( UART\_type\* UARTx, **void**\* BuffAddr, uint32\_t MaxLength )

UART\_Type\* UARTx – указатель на структуру интерфейса UART

uint8\_t\* BuffAddr – указатель на приемный буфер

uint32\_t MaxLenght – максимальное количество принимаемых данных

**Описание**

Функция инициализирует прием данных через UART с использованием прерываний. Состояние передачи находится в структуре UART\_TxRxStatus\_type:

**typedef** **struct**

{

uint8\_t TxReady[ *UART\_NUMS* ]; // Все данные переданы

uint8\_t RxReady[ *UART\_NUMS* ]; // Приняты новые данные

uint32\_t RxLenght[ *UART\_NUMS* ]; // Количество принятых данных

} UART\_TxRxStatus\_type;

**Пример**

**#include** <hal\_1967VN044.h>

uint8\_t ucRxData[ 128 ];

**void** **main**(**void**)

{

UART\_Init\_type UARTInit;

HAL\_UART\_DefaultInitStruct( &UARTInit );

HAL\_GPIO\_Init( LX\_GPIO\_PA, GPIO\_PIN\_2 | GPIO\_PIN\_3, GPIO\_PinMode\_Alt );

HAL\_UART\_Init( LX\_UART1, &UARTInit );

HAL\_UART\_ITEnable( LX\_UART1 );

HAL\_Interrupt\_GlobalEnable();

**while**( 1 )

{

HAL\_UART\_ITReceive( LX\_UART1, ucTestText, 128 );

**while**( !UartTxRxStatus.RxReady[ *UART1* ] );

}

}