# Описание используемых типов

## I2C\_Conf\_type

|  |  |
| --- | --- |
| **Имя поля** | **Описание** |
| eMode | Режим работы:  *I2C\_MASTER* - ведущий,  *I2C\_SLAVE* - ведомый |
| ulClockSpeed | Частота клока SCL |
| eAddrMode | Режим адресации:  *I2C\_ADDR\_7BIT* – 7 бит  *I2C\_ADDR\_10BIT* – 10 бит |
| usAddr | Адрес устройства в режиме ведомого |
| eAddrZero | Управление доступом по нулевому адресу устройства:  *I2C\_ADDR\_0\_DISABLE* – доступ запрещен  *I2C\_ADDR\_0\_ENABLE* – доступ разрешен |

## I2C\_Struct\_type

|  |  |
| --- | --- |
| **Имя поля** | **Описание** |
| eStatus | Текущее состояние интерфейса:  *I2C\_OK* – нет ошибок  *I2C\_BUSY* – интерфейс занят  *I2C\_NO\_ANSWER* – ведомое устройство не ответило  *I2C\_SLAVE\_STOP* – интерфейс закончил работу в режиме ведомого  *I2C\_BUS\_FAIL* – ошибка шины |
| eControl | Конфигурация операции в режиме мастера:  *I2C\_MSTR\_WR* – чтение  *I2C\_MSTR\_RD* – запись  *I2C\_MSTR\_RD\_ADDR* – чтение из заданного адреса памяти |
| ucRxCnt | Количество принятых данных ведомым |
| ucSlaveInBuff | Входной буфер ведомого |
| ucSlaveOutBuff | Выходной буфер ведомого |
| ucMasterBuff | Буфер мастера |
| ucMasterDataLen | Количество данных для передачи/приема |
| ucSlaveAddr | Адрес ведомого, в который производится запись в режиме мастера |
| ucDataAddr | Адрес данных, которые собираемся читать |

# Описание используемых функций

## HAL\_I2C\_Init

**Резюме**

**void** **HAL\_I2C\_Init**( I2C\_Conf\_type \*pxConf )

I2C\_Conf\_type \*pxConf – указатель на структуру с конфигурационными настройками

**Описание**

Функция инициализации контроллера i2c параметрами из структуры, на которую указывает указатель pxConf.

## HAL\_I2C\_SetupGpio

**Резюме**

**void** **HAL\_I2C\_SetupGpio**( **void** )

**Описание**

Функция используется для инициализации пинов PC11 и PC12 для работы с интерфейсом i2c.

## HAL\_I2C\_Enable

**Резюме**

**void** **HAL\_I2C\_Enable**( **void** )

**Описание**

Функция используется для включения интерфейса i2c и разрешения генерации им прерываний.

## HAL\_I2C\_Write

**Резюме**

**void** **HAL\_I2C\_Write**( uint8\_t ucDevAddr, uint8\_t \*pucBuff, uint8\_t ucLenght )

uint8\_t ucDevAddr – адрес ведомого устройства

uint8\_t \*pucBuff – указатель на буфер с данными

uint8\_t ucLenght – количество данных для записи

**Описание**

Функция используется для записи в ведомое устройства массива байт.

## HAL\_I2C\_Read

**Резюме**

**void** **HAL\_I2C\_Read**( uint8\_t ucDevAddr, uint8\_t ucLenght )

uint8\_t ucDevAddr – адрес ведомого устройства

uint8\_t ucLenght – количество данных для чтения

**Описание**

Функция используется для чтения из ведомого устройства массива байт. Прочитанные данные будут находится в буфере ucMasterBuff в структуре I2C\_Struct.

## HAL\_I2C\_ReadMemAddr

**Резюме**

**void** **HAL\_I2C\_ReadMemAddr**( uint8\_t ucDevAddr, uint8\_t ucDataAddr, uint8\_t ucLenght )

uint8\_t ucDevAddr – адрес ведомого устройства

uint8\_t ucDataAddr – адрес в памяти ведомого, начиная с которого будет производится чтение

uint8\_t ucLenght – количество данных для чтения

**Описание**

Функция используется для чтения из ведомого устройства массива байт, расположенных по заданному адресу. Прочитанные данные будут находится в буфере ucMasterBuff в структуре I2C\_Struct. Данная функция используетя для чтения данных из микросхем EEPROM.

# Пример работы

## Режим ведущего

**#include** "hal\_1967VN044.h"

**#define** CORECLK\_KHZ 200000

uint8\_t TxBuff[ 5 ]; // Буфер с данными

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

{

I2C\_Conf\_type xI2cConf;

LX\_CMU->CFG8.word = 0xFFFFFFFF; // Отключение тактирования всех блоков

HAL\_PLL\_CoreSetup( CORECLK\_KHZ );

LX\_CMU->CFG8.b.I2C\_DIS = 0; // Включение тактирования i2c

xI2cConf.eMode = *I2C\_MASTER*;

xI2cConf.ulClockSpeed = 400000; // Скорость обмена 400 кГц

HAL\_I2C\_PinConf();

HAL\_I2C\_Init( &xI2cConf );

HAL\_I2C\_Enable();

TxBuff[ 0 ] = 1;

TxBuff[ 1 ] = 2;

TxBuff[ 2 ] = 3;

TxBuff[ 3 ] = 4;

TxBuff[ 4 ] = 5;

HAL\_I2C\_Write( 0x21, TxBuff, 5 ); // Запись данных в ведомое устройство

HAL\_I2C\_Read( 0x21, 5 ); // Чтение данных из ведомого устройства

**if**( I2C\_Struct.eStatus == *I2C\_OK* ) // Если успешно прочитали данные

{

TxBuff[ 0 ] = I2C\_Struct.ucMasterBuff[ 0 ];

TxBuff[ 1 ] = I2C\_Struct.ucMasterBuff[ 1 ];

}

HAL\_I2C\_ReadMemAddr( 0x21, 0x10, 3 ); // Чтение 3 байт по адресу 0x10

**if**( I2C\_Struct.eStatus == *I2C\_OK* ) // Если успешно прочитали данные

{

TxBuff[ 0 ] = I2C\_Struct.ucMasterBuff[ 0 ];

TxBuff[ 1 ] = I2C\_Struct.ucMasterBuff[ 1 ];

TxBuff[ 2 ] = I2C\_Struct.ucMasterBuff[ 2 ];

}

}

В данном примере происходит инициализация интерфейса i2c в режиме ведущего. Размер буфера ведущего задается в файле hal\_i2c.h:

**#define** I2C\_MASTER\_BUFF\_SIZE 32

Для корректного задания скорости работы интерфейса необходимо указать частоту внешнего тактирования в файле hal\_pll.h:

**#define** XTI\_KHZ 25000 // Частота внешнего тактирования, кГц

## Режим ведомого

**#include** "hal\_1967VN044.h"

**#define** CORECLK\_KHZ 200000

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

{

I2C\_Conf\_type xI2cConf;

LX\_CMU->CFG8.word = 0xFFFFFFFF; // Отключение тактирования всех блоков

HAL\_PLL\_CoreSetup( CORECLK\_KHZ );

LX\_CMU->CFG8.b.I2C\_DIS = 0; // Включение тактирования i2c

I2C\_Struct.ucSlaveOutBuff[ 0 ] = 'V';

I2C\_Struct.ucSlaveOutBuff[ 1 ] = 'N';

I2C\_Struct.ucSlaveOutBuff[ 2 ] = '0';

I2C\_Struct.ucSlaveOutBuff[ 3 ] = '3';

I2C\_Struct.ucSlaveOutBuff[ 4 ] = '4';

xI2cConf.eMode = I2C\_SLAVE;

xI2cConf.eAddrMode = I2C\_ADDR\_7BIT;

xI2cConf.usAddr = 0x0A;

xI2cConf.eAddrZero = I2C\_ADDR\_0\_DISABLE;

HAL\_I2C\_PinConf();

HAL\_I2C\_Init( &xI2cConf );

HAL\_I2C\_Enable();

**while**( I2C\_Struct.eStatus != *I2C\_SLAVE\_STOP* ) // Ждем конца обмена

**if**( I2C\_Struct.ucRxCnt != 0 ) // Мастер что-то записал

{

// Делаем необходимые действия, например, загружаем новые данные в буфер

}

}

В данном примере происходит инициализация интерфейса i2c в режиме ведомого.

Размер буфера ведомого задается в файле hal\_i2c.h:

**#define** I2C\_SLAVE\_BUFF\_SIZE 32

Данные которые мы хотим отправить ведущему нужно записать в буфер I2C\_Struct.ucSlaveOutBuff. Данные которые мы получим от ведущего будут записаны в буфер I2C\_Struct.ucSlaveInBuff, а их количество в I2C\_Struct.ucRxCnt. После завершения чтения или записи данных ведущим статус I2C\_Struct.eStatus станет равным *I2C\_SLAVE\_STOP.*

## Ошибки

На текущий момент в реализации интерфейса есть ошибка в переключении используемых пинов. Для того, чтобы интерфейс работал стабильно, требуется на выводе SCL поставить RC фильтр. Номинал резистора 100 Ом, а конденсатора 30 -56 пФ. Такие номиналы позволяют работать до скорости обмена в 400 кГц.