# Описание используемых типов

## SPI\_Init\_type

|  |  |  |
| --- | --- | --- |
| **Поле** | **Значения** | **Описание** |
| WordSize | от 4 до 32 | Количество бит в одном слове |
| CLK\_Polarity | *SPI\_CLK\_Polarity\_High,*  *SPI\_CLK\_Polarity\_Low* | Полярность тактового сигнала |
| CLK\_Phase | *SPI\_CLK\_Phase\_Negedge,*  *SPI\_CLK\_Phase\_Posedge* | Фаза тактового сигнала: спадающий (Negedge) или возрастающий (Posedge) фронт. |
| FirstBit | *SPI\_FirstBit\_MSB,*  *SPI\_FirstBit\_LSB* | Очередность выдачи бит в слове: сначала старший (MSB) или сначала младший (LSB) бит. |
| CSNum | от 0 до 5 | Номер пина SPI\_CSx для выбора устройства на линии |
| CLK\_Prescaler | от 0 до 4095 | Делитель тактового сигнала SPI |
| Mode | *SPI\_Mode\_Master,*  *SPI\_Mode\_Slave* | Режим работы: ведущий (Master) или ведомый (Slave) |
| CS\_Hold | *SPI\_CS\_Unhold,*  *SPI\_CS\_Hold* | Удержания сигнала CS в активном состоянии в перерывах между приемо-передачей |
| CS\_Hold\_Delay | От 0 до15 | Длительность неактивного уровня CS между словами, при CS\_Unhold в тактах SCK |
| CS\_Active | *SPI\_CS\_Active\_Low,*  *SPI\_CS\_Active\_High* | Активное состояние сигнала CS |
| LoopBack | *SPI\_LoopBack\_Off,*  *SPI\_LoopBack\_On* | Тестовый режим, замыкающий выходные данные на входные |

## SPI\_TransferStatus\_type

|  |  |  |
| --- | --- | --- |
| **Поле** | **Значения** | **Описание** |
| Spi0\_TxDmaReady | 0…1 | Завершена передача по SPI0 |
| Spi1\_TxDmaReady | 0…1 | Завершена передача по SPI1 |
| Spi2\_TxDmaReady | 0…1 | Завершена передача по SPI2 |
| Spi0\_RxDmaReady | 0…1 | Завершен прием по SPI0 |
| Spi1\_RxDmaReady | 0…1 | Завершен прием по SPI1 |
| Spi2\_RxDmaReady | 0…1 | Завершен прием по SPI2 |

# Описание используемых функций

## HAL\_SPI\_Init

**Резюме**

**void** **HAL\_SPI\_Init** ( SPI\_type \*SPI, SPI\_Init\_type \*SPI )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

SPI\_Init\_type \*SPI – указатель на структуру с параметрами инициализации

**Описание**

Функция производит первичную инициализацию интерфейса SPI. В функции производится проверка параметров входной структуры настроек SPI\_Init\_type \*SPI и инициализация регистров SPI. Переключение пинов под SPI не проводится. Так же функция может использоваться для переконфигурирования SPI в процессе работы.

**Пример**

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

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

{

SPI\_Init\_type SPI;

SPI.CLK\_Phase = *SPI\_CLK\_Phase\_Negedge*; // активность по спадающему фронту

SPI.CLK\_Polarity = *SPI\_CLK\_Polarity\_High*; // высокий уровень CLK

SPI.CLK\_Prescaler = 2; // предделитель частоты на (2 + 1)

SPI.CSNum = 0; // обмен по сигналу СS0

SPI.CS\_Active = *SPI\_CS\_Active\_Low*; // активный уровень CS - низкий

SPI.FirstBit = *SPI\_FirstBit\_MSB*; // первым выдает старший байт слова

SPI.Mode = *SPI\_Mode\_Master*; // режим работы - ведущий

SPI.CS\_Hold = *SPI\_CS\_Unhold*; // сигнал CS не удерживается в активном состоянии

SPI.WordSize = 8; // размер выдаваемого слова - 8 бит

SPI.LoopBack = *SPI\_LoopBack\_Off*; // тестовый режим возврата данных выключен

HAL\_SPI\_Init (LX\_SPI0, &SPI);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_4* | *GPIO\_PIN\_5* | *GPIO\_PIN\_7*, *GPIO\_PinMode\_Alt*);

## }HAL\_SPI\_DeInit

**Резюме**

**void** **HAL\_SPI\_DeInit**( SPI\_type \*SPI )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

**Описание**

Функция переключает сбрасывает регистры SPI в начальные значения. Выводы, задействованные под SPI, должны деинициализироваться отдельно.

**Пример**

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

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

{

HAL\_GPIO\_DeInit (LX\_GPIO\_PA, *GPIO\_PIN\_4* | *GPIO\_PIN\_5* | *GPIO\_PIN\_7*);

HAL\_SPI\_DeInit (LX\_SPI0);

}

## HAL\_SPI\_DefaultInitStruct

**Резюме**

**void** **HAL\_SPI\_DefaultInitStruct**( SPI\_Init\_type \*SPI )

SPI\_Init\_type \*SPI – указатель на структуру с параметрами инициализации

**Описание**

Функция заполняет структуру инициализации SPI, переданную по указателю, значениями по умолчанию. Присваиваемые значения отображены в таблице:

|  |  |
| --- | --- |
| **Поле структуры** | **Значение** |
| WordSize | 8 |
| CLK\_Polarity | *SPI\_CLK\_Polarity\_High* |
| CLK\_Phase | *SPI\_CLK\_Phase\_Negedge* |
| FirstBit | *SPI\_FirstBit\_MSB* |
| CSNum | 0 |
| CLK\_Prescaler | 2 |
| Mode | *SPI\_Mode\_Master* |
| CS\_Hold | *SPI\_CS\_Unhold* |
| CS\_Hold\_Delay | *0* |
| CS\_Active | *SPI\_CS\_Active\_Low* |
| LoopBack | *SPI\_LoopBack\_Off* |

Описание присваиваемых значений приведены в описании используемых типов.

**Пример**

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

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

{

SPI\_Init\_type SPI;

HAL\_SPI\_DefaultInitStruct (&SPI);

SPI.CSNum = 3;

HAL\_SPI\_Init (LX\_SPI0, &SPI);

}

## HAL\_SPI\_Control

**Резюме**

**void** **HAL\_SPI\_Control**( SPI\_type \*SPI, SPI\_State\_type State )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

SPI\_State\_type State – состояние интерфейса

**Описание**

Входной аргумент SPI\_State\_type State может принимать одно из двух значений: *SPI\_State\_Disable* или *SPI\_State\_Enable*. Если он принимает значение *SPI\_State\_Disable*, то интерфейс SPI выключается и обмен данными не производится. Если он принимает значение *SPI\_State\_Enable*, то интерфейс SPI включается и готов к обмену данными.

Данная функция автоматически вызывается при инициализации SPI с помощью **HAL\_SPI\_Init** c аргументом *SPI\_State\_Enable*.

**Пример**

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

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

{

SPI\_Init\_type SPI;

HAL\_SPI\_DefaultInitStruct (&SPI);

HAL\_SPI\_Init (LX\_SPI0, &SPI);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_4* | *GPIO\_PIN\_5* | *GPIO\_PIN\_7*, *GPIO\_PinMode\_Alt*);

HAL\_SPI\_Control (LX\_SPI0, *SPI\_State\_Disable*);

}

## HAL\_SPI\_SetExchangeMode

**Резюме**

**void** **HAL\_SPI\_SetExchangeMode**( SPI\_type \*SPI, SPI\_ExchangeMode\_type Mode )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

SPI\_ExchangeMode\_type Mode – режим обмена

**Описание**

Функция настройки режима обмена по интерфейсу SPI. Режим выбирается с помощью входного аргумента SPI\_ExchangeMode\_type Mode и может быть следующих типов:

*SPI\_ExchangeMode\_All* - одновременный прием и передача,

*SPI\_ExchangeMode\_TxOnly* - только передача данных,

*SPI\_ExchangeMode\_RxOnly* - только прием.

Функция может использоваться как альтернатива установки и сброса бит TXO и RXO регистра SPCR1.

**Пример**

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

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

{

SPI\_Init\_type SPI;

uint32\_t Buf[8] = {1, 2, 3, 4, 5, 6, 7, 8};

HAL\_SPI\_DeInit (LX\_SPI0);

HAL\_SPI\_DefaultInitStruct (&SPI);

HAL\_SPI\_Init (LX\_SPI0, &SPI);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_4* | *GPIO\_PIN\_5* | *GPIO\_PIN\_7*, *GPIO\_PinMode\_Alt*);

HAL\_SPI\_SetExchangeMode (LX\_SPI0, *SPI\_ExchangeMode\_TxOnly*);

HAL\_SPI\_WriteToTxFIFO (LX\_SPI0, Buf, 8);

HAL\_SPI\_SetExchangeMode (LX\_SPI0, *SPI\_ExchangeMode\_All*);

}

## HAL\_SPI\_GetExchangeMode

**Резюме**

SPI\_ExchangeMode\_type **HAL\_SPI\_GetExchangeMode** ( SPI\_type \*SPI )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

**Описание**

Функция возвращает текущий режим обмена по интерфейсу SPI:

*SPI\_ExchangeMode\_All* - одновременный прием и передача,

*SPI\_ExchangeMode\_TxOnly* - только передача данных,

*SPI\_ExchangeMode\_RxOnly* - только прием.

Функция может быть использована как альтернатива чтению бит TXO и RXO регистра SPCR1.

**Пример**

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

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

{

SPI\_Init\_type SPI;

uint32\_t Buf[8] = {1, 2, 3, 4, 5, 6, 7, 8};

HAL\_SPI\_DeInit (LX\_SPI0);

HAL\_SPI\_DefaultInitStruct (&SPI);

HAL\_SPI\_Init (LX\_SPI0, &SPI);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_4* | *GPIO\_PIN\_5* | *GPIO\_PIN\_7*, *GPIO\_PinMode\_Alt*);

**if** (HAL\_SPI\_GetExchangeMode(LX\_SPI0) == *SPI\_ExchangeMode\_RxOnly*)

HAL\_SPI\_SetExchangeMode (LX\_SPI0, *SPI\_ExchangeMode\_All*);

}

## HAL\_SPI\_WriteToTxFIFO

**Резюме**

uint32\_t **HAL\_SPI\_WriteToTxFIFO**( SPI\_type \*SPI, uint32\_t \*src, uint32\_t  
Size )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

uint32\_t \*src – указатель на буфер с данными

uint32\_t Size – количество данных

**Описание**

Функция копирует данные из массива с начальным адресом src в FIFO передатчика SPI. Копирование происходит до тех пор, пока либо FIFO не заполнится полностью, либо количество копирований не достигнет Size.

Функция возвращает общее количество скопированных слов. Диапазон значений от 0 (FIFO было полностью заполнено) до Size (скопированы все данные). Переданный указатель на массив \*src инкрементируется до значения, соответствующего следующему потенциально выдаваемому байту.

**Пример**

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

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

{

SPI\_Init\_type SPI;

uint32\_t Buf[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

uint32\_t \*pBuf = Buf;

uint32\_t S;

HAL\_SPI\_DefaultInitStruct (&SPI);

HAL\_SPI\_Init (LX\_SPI0, &SPI);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_4* | *GPIO\_PIN\_5* | *GPIO\_PIN\_7*, *GPIO\_PinMode\_Alt*);

S = HAL\_SPI\_WriteToTxFIFO (LX\_SPI0, Buf, 6); // S = 6

**while** (HAL\_SPI\_GetFlag(LX\_SPI0, *SPI\_Flag\_Busy*));

S = HAL\_SPI\_WriteToTxFIFO (LX\_SPI0, Buf, 10); // S = 8 (max FIFO size)

**while** (HAL\_SPI\_GetFlag(LX\_SPI0, *SPI\_Flag\_Busy*));

S = 10;

**while** (S != 0)

{

S -= HAL\_SPI\_WriteToTxFIFO (LX\_SPI0, pBuf, S);

} // S = 0

}

## HAL\_SPI\_ReadFromRxFIFO

**Резюме**

uint32\_t **HAL\_SPI\_ReadFromRxFIFO**( SPI\_type \*SPI, uint32\_t \*dst, uint32\_t Size )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

uint32\_t \*dst – указатель на приемный буфер

uint32\_t Size – количество данных

**Описание**

Функция копирует данные из FIFO приемника SPI в массив с начальным адресом dst. Копирование происходит до тех пор, пока либо FIFO не опустеет полностью, либо количество копирований не достигнет Size.

Функция возвращает количество считанных из FIFO слов. Диапазон значений от 0 (в FIFO не было данных) до Size (доступная для записи память закончилась).

**Пример**

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

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

{

SPI\_Init\_type SPI;

uint32\_t Buf[10];

uint32\_t \*pBuf = Buf;

uint32\_t S;

HAL\_SPI\_DefaultInitStruct (&SPI);

HAL\_SPI\_Init (LX\_SPI0, &SPI);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_4* | *GPIO\_PIN\_5* | *GPIO\_PIN\_7*, *GPIO\_PinMode\_Alt*);

S = HAL\_SPI\_ReadFromRxFIFO (LX\_SPI0, Buf, 6); // S = 0...6

S = HAL\_SPI\_ReadFromRxFIFO (LX\_SPI0, Buf, 10); // S = 0...8

S = 10;

**while** (S != 0)

{

S -= HAL\_SPI\_ReadFromRxFIFO (LX\_SPI0, pBuf, S);

} // S = 10

**return** 0;

}

## HAL\_SPI\_SendOnly

**Резюме**

**void** **HAL\_SPI\_SendOnly**( SPI\_type \*SPI, uint32\_t \*src, uint32\_t Size )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

uint32\_t \*src – указатель на буфер с данными

uint32\_t Size – количество данных

**Описание**

Функция может быть использована в случае необходимости только отправки данных по SPI без одновременного приема. При отправке данных с помощью данной функции отключается прием данных, что блокирует перезапись слов в FIFO приемника.

**Пример**

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

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

{

SPI\_Init\_type SPI;

uint32\_t Buf[8] = {1, 2, 3, 4, 5, 6, 7, 8};

SPI\_State\_type State;

HAL\_SPI\_DefaultInitStruct (&SPI);

HAL\_SPI\_Init (LX\_SPI0, &SPI);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_4* | *GPIO\_PIN\_5* | *GPIO\_PIN\_7*, *GPIO\_PinMode\_Alt*);

// Обычная отправка данных:

HAL\_SPI\_WriteToTxFIFO (LX\_SPI0, Buf, 8);

**while** (HAL\_SPI\_GetFlag(LX\_SPI0, *SPI\_Flag\_Busy*));

State = HAL\_SPI\_GetFlag (LX\_SPI0, *SPI\_Flag\_RxFifoFull*); // State = SPI\_State\_Enable

HAL\_SPI\_ReadFromRxFIFO (LX\_SPI0, Buf, 8);

// Только отправка данных:

HAL\_SPI\_SendOnly (LX\_SPI0, Buf, 8);

**while** (HAL\_SPI\_GetFlag(LX\_SPI0, *SPI\_Flag\_Busy*));

State = HAL\_SPI\_GetFlag (LX\_SPI0, *SPI\_Flag\_RxFifoFull*); // State = SPI\_State\_Disable

}

## HAL\_SPI\_ReceiveOnly

**Резюме**

**void** **HAL\_SPI\_ReceiveOnly**( SPI\_type \*SPI, uint32\_t \*dst, uint32\_t Size, uint8\_t DO\_State )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

uint32\_t \*dst – указатель на приемный буфер

uint32\_t Size – количество данных

uint8\_t DO\_State – сигнал выдаваемый на линию TX

**Описание**

Функция может быть использована при необходимости только приема данных по интерфейсу SPI без одновременной выдачи данных. При приеме данных с помощью данной функции отключается передача данных, что блокирует чтение слов из FIFO передатчика. Вместо данных из FIFO на линию выставляется сигнал, заданный в DO\_State (0 или 1).

**Пример**

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

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

{

SPI\_Init\_type SPI;

uint32\_t Buf[12];

HAL\_SPI\_DefaultInitStruct (&SPI);

HAL\_SPI\_Init (LX\_SPI0, &SPI);

HAL\_GPIO\_Init (LX\_GPIO\_PA, *GPIO\_PIN\_4* | *GPIO\_PIN\_5* | *GPIO\_PIN\_7*, *GPIO\_PinMode\_Alt*);

HAL\_SPI\_ReceiveOnly (LX\_SPI0, Buf, 12, 0);

}

## HAL\_SPI\_SendAndReceive

**Резюме**

**void** **HAL\_SPI\_SendAndReceive**( SPI\_type \*SPI, uint32\_t \*src, uint32\_t \*dst, uint32\_t Size )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

uint32\_t \*src – указатель на буфер для передачи

uint32\_t \*dst – указатель на приемный буфер

uint32\_t Size – количество данных

**Описание**

Функция может быть использована при необходимости одновременной отправки и приеме данных длинны, превышающих размер FIFO. Функция будет выполнятся до тех пор, пока количество отданных и полученных слов не достигнет Size. В процессе копирования указатели на массивы src и dst инкрементируются.

В функции учтена проблема получения данных из FIFO приемника при одновременном обмене данными на линии.

**Пример**

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

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

{

SPI\_Init\_type SPI;

uint32\_t BufTx[12] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

uint32\_t BufRx[12] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

HAL\_SPI\_DefaultInitStruct (&SPI);

SPI.LoopBack = *SPI\_LoopBack\_On*;

HAL\_SPI\_Init (LX\_SPI0, &SPI);

//HAL\_GPIO\_Init (LX\_GPIO\_PA, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_7, GPIO\_PinMode\_Alt);

HAL\_SPI\_SendAndReceive (LX\_SPI0, BufTx, BufRx, 12);

// BufRx = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

}

## HAL\_SPI\_GetFlag

**Резюме**

SPI\_State\_type **HAL\_SPI\_GetFlag**( SPI\_type \*SPI, **const** SPI\_Flag\_type Flag )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

**const** SPI\_Flag\_type Flag – флаг интерфейса SPI

**Описание**

Функция возвращает значение флага интерфейса SPI из регистра SPSR. Флаг выбирается с помощью входного аргумента SPI\_Flag\_type Flag:

*SPI\_Flag\_TxFifoEmpty* - заполнение FIFO передатчика («пустой»)

*SPI\_Flag\_TxFifoNotFull* - заполнение FIFO передатчика («пустой»)

*SPI\_Flag\_RxFifoNotEmpty* - FIFO приемника не пусто

*SPI\_Flag\_Busy* - происходит приемо-передача

*SPI\_Flag\_TxFifoHalfEmpty* - FIFO передатчика заполнен на половину или менее

*SPI\_Flag\_RxFifoHalfFull* - FIFO приемника заполнен на половину или более

*SPI\_Flag\_RxFifoOverflow* - переполнение в FIFO приемника

*SPI\_Flag\_TxFifoReadEmpty* - произведено чтение из пустого FIFO передатчика

*SPI\_Flag\_RxFifoFull* - FIFO приемника заполнен полностью

Функция возвращает одно из следующих значений:

*SPI\_State\_Disable* = 0 - ложь,

*SPI\_State\_Enable* = 1 - истина.

Функция может быть использована для получения актуальных значений флагов регистра SPSR как альтернатива непосредственного чтения из данного регистра.

**Пример**

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

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

{

SPI\_Init\_type SPI;

SPI\_State\_type State;

uint32\_t Buf[8] = {1, 2, 3, 4, 5, 6, 7, 8};

HAL\_SPI\_DefaultInitStruct (&SPI);

SPI.LoopBack = *SPI\_LoopBack\_On*;

HAL\_SPI\_Init(LX\_SPI0, &SPI);

//HAL\_GPIO\_Init(LX\_GPIO\_PA, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_7, GPIO\_PinMode\_Alt);

State = HAL\_SPI\_GetFlag(LX\_SPI0, *SPI\_Flag\_RxFifoNotEmpty*); // State = SPI\_State\_Disable

HAL\_SPI\_WriteToTxFIFO(LX\_SPI0, Buf, 8);

State = HAL\_SPI\_GetFlag(LX\_SPI0, *SPI\_Flag\_RxFifoNotEmpty*); // State = SPI\_State\_Enable

}

## HAL\_SPI\_ClearFlag

**Резюме**

**void** **HAL\_SPI\_ClearFlag**( SPI\_type \*SPI, **const** SPI\_Flag\_type Flag )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

**const** SPI\_Flag\_type Flag – флаг интерфейса SPI

**Описание**

Функция для сброса флага в регистре SPSR. Сбрасываемый флаг задается входным аргументом SPI\_Flag\_type Flag. Допустимые значения только *SPI\_Flag\_RxFifoOverflow* или *SPI\_Flag\_TxFifoReadEmpty*. Остальные значения эффекта не имеют.

Функция может быть использована как альтернатива непосредственной записи «1» для сброса флагов в регистре SPSR.

**Пример**

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

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

{

SPI\_Init\_type SPI;

SPI\_State\_type State;

uint32\_t Buf[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

uint32\_t \*pBuf = Buf;

uint32\_t S = 10;

HAL\_SPI\_DefaultInitStruct (&SPI);

SPI.LoopBack = *SPI\_LoopBack\_On*;

HAL\_SPI\_Init (LX\_SPI0, &SPI);

//HAL\_GPIO\_Init(LX\_GPIO\_PA, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_7, GPIO\_PinMode\_Alt);

State = HAL\_SPI\_GetFlag (LX\_SPI0, *SPI\_Flag\_RxFifoOverflow*); // State = SPI\_State\_Disable

**while** (S != 0) S -= HAL\_SPI\_WriteToTxFIFO (LX\_SPI0, pBuf, S);

State = HAL\_SPI\_GetFlag (LX\_SPI0, *SPI\_Flag\_RxFifoOverflow*); // State = SPI\_State\_Enable

HAL\_SPI\_ClearFlag (LX\_SPI0, *SPI\_Flag\_RxFifoOverflow*);

State = HAL\_SPI\_GetFlag (LX\_SPI0, *SPI\_Flag\_RxFifoOverflow*); // State = SPI\_State\_Disable

}

## HAL\_SPI\_ITConfig

**Резюме**

**void HAL\_SPI\_ITConfig**( SPI\_type \*SPI, **const** SPI\_IT\_type IT, **const** SPI\_State\_type State )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

**const** SPI\_IT\_type IT – прерывание интерфейса SPI

**const** SPI\_State\_type State – включение/отключение прерывания

**Описание**

Функция должна использоваться для включения или отключения события, приводящего к прерыванию от интерфейса SPI. Доступны следующие типы событий SPI\_IT\_type:

*SPI\_IT\_TxFifoNotFull* - FIFO передатчика заполнено не полностью

*SPI\_IT\_TxFifoHalfEmpty* - FIFO передатчика пусто на половину или меньше

*SPI\_IT\_TxReadEmptyFifo* - была попытка чтения из пустого FIFO передатчика

*SPI\_IT\_RxFifoNotEmpty* - FIFO приемника не пустое

*SPI\_IT\_RxFifoHalfFull* - FIFO приемника заполнено на половину или более

*SPI\_IT\_RxFifoOverflow* - было переполнение FIFO приемника

*SPI\_IT\_RxCntEnd* - завершено чтение по линии SPI без приема данных

Для включения или выключения прерывания по выбранному событию необходимо в аргументе State указать *SPI\_State\_Enable* или *SPI\_State\_Disable* соответственно.

**Пример**

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

**void** **SPI\_Handler** ();

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

{

SPI\_Init\_type SPI;

SPI\_State\_type State;

uint32\_t Buf[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

uint32\_t \*pBuf = Buf;

uint32\_t S = 10;

HAL\_SPI\_DefaultInitStruct (&SPI);

SPI.LoopBack = *SPI\_LoopBack\_On*;

HAL\_SPI\_Init (LX\_SPI0, &SPI);

//HAL\_GPIO\_Init(LX\_GPIO\_PA, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_7, GPIO\_PinMode\_Alt);

HAL\_Interrupt\_IMASKClear();

HAL\_Interrupt\_ILATClear();

HAL\_Interrupt\_GlobalEnable();

HAL\_Interrupt\_Enable (*intSPI0*, SPI\_Handler);

HAL\_SPI\_ITConfig (LX\_SPI0, *SPI\_IT\_RxFifoOverflow*, *SPI\_State\_Enable*);

**while** (S != 0) S -= HAL\_SPI\_WriteToTxFIFO (LX\_SPI0, pBuf, S);

**return** 0;

}

**#pragma** interrupt

**void** **SPI\_Handler** () {

**if** (HAL\_SPI\_ITCheck (LX\_SPI0, *SPI\_IT\_RxFifoOverflow*)) {

// тут можно сделать что-то полезное

HAL\_SPI\_ClearFlag (LX\_SPI0, *SPI\_Flag\_RxFifoOverflow*);

}

}

## HAL\_SPI\_ITCheck

**Резюме**

SPI\_State\_type **HAL\_SPI\_ITCheck**( SPI\_Type \*SPI, **const** SPI\_IT\_type IT )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

**const** SPI\_IT\_type IT – прерывание интерфейса SPI

**Описание**

Функция проверки наступления события, вызвавшего прерывание от интерфейса SPI. В качестве аргумента передается тип события. Все типы описаны в Описании функции **HAL\_SPI\_ITConfig**.

Возвращаемое значение:

*SPI\_State\_Enable* - событие наступило

*SPI\_State\_Disable* - событие не наступило

Функция предназначена для использования в обработчике прерывания от интерфейса SPI для определения события, вызвавшего прерывание. В данной функции учитываются как статусные флаги, так и флаги, разрешающие генерировать прерывание событиям.

**Пример**

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

**void** **SPI0\_Handler** ();

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

{

SPI\_Init\_type SPI;

SPI\_State\_type State;

uint32\_t Buf[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

uint32\_t \*pBuf = Buf;

uint32\_t S = 10;

HAL\_SPI\_DefaultInitStruct (&SPI);

SPI.LoopBack = *SPI\_LoopBack\_On*;

HAL\_SPI\_Init (LX\_SPI0, &SPI);

//HAL\_GPIO\_Init(LX\_GPIO\_PA, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_7, GPIO\_PinMode\_Alt);

HAL\_Interrupt\_IMASKClear();

HAL\_Interrupt\_ILATClear();

HAL\_Interrupt\_GlobalEnable();

HAL\_Interrupt\_Enable (*intSPI0*, SPI0\_Handler);

HAL\_SPI\_ITConfig (LX\_SPI0, *SPI\_IT\_RxFifoOverflow*, *SPI\_State\_Enable*);

**while** (S != 0) S -= HAL\_SPI\_WriteToTxFIFO (LX\_SPI0, pBuf, S);

**return** 0;

}

**#pragma** interrupt

**void** **SPI0\_Handler** ()

{

**if** (HAL\_SPI\_ITCheck (LX\_SPI0, *SPI\_IT\_RxCntEnd*)) {

// Тут можно сделать что-то полезное

HAL\_SPI\_ITConfig (LX\_SPI0, *SPI\_IT\_RxCntEnd*, *SPI\_State\_Disable*);

}

**if** (HAL\_SPI\_ITCheck (LX\_SPI0, *SPI\_IT\_RxFifoHalfFull*)) {

// Тут можно сделать что-то полезное

HAL\_SPI\_ITConfig (LX\_SPI0, *SPI\_IT\_RxFifoHalfFull*, *SPI\_State\_Disable*);

}

**if** (HAL\_SPI\_ITCheck (LX\_SPI0, *SPI\_IT\_RxFifoNotEmpty*)) {

// Тут можно сделать что-то полезное

HAL\_SPI\_ITConfig (LX\_SPI0, *SPI\_IT\_RxFifoNotEmpty*, *SPI\_State\_Disable*);

}

**if** (HAL\_SPI\_ITCheck (LX\_SPI0, *SPI\_IT\_RxFifoOverflow*)) {

// Тут можно сделать что-то полезное

HAL\_SPI\_ClearFlag (LX\_SPI0, *SPI\_Flag\_RxFifoOverflow*);

}

**if** (HAL\_SPI\_ITCheck (LX\_SPI0, *SPI\_IT\_TxFifoHalfEmpty*)) {

// Тут можно сделать что-то полезное

HAL\_SPI\_ITConfig (LX\_SPI0, *SPI\_IT\_TxFifoHalfEmpty*, *SPI\_State\_Disable*);

}

**if** (HAL\_SPI\_ITCheck (LX\_SPI0, *SPI\_IT\_TxFifoNotFull*)) {

// Тут можно сделать что-то полезное

HAL\_SPI\_ITConfig (LX\_SPI0, *SPI\_IT\_TxFifoNotFull*, *SPI\_State\_Disable*);

}

**if** (HAL\_SPI\_ITCheck (LX\_SPI0, *SPI\_IT\_TxReadEmptyFifo*)) {

// Тут можно сделать что-то полезное

HAL\_SPI\_ClearFlag (LX\_SPI0, *SPI\_Flag\_TxFifoReadEmpty*);

}

}

## HAL\_SPI\_DMA\_SendOnly

**Резюме**

**void** **HAL\_SPI\_DMA\_SendOnly**( SPI\_type \*SPI, uint32\_t ulChannel, uint16\_t \*pusBuff, uint16\_t usSize )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

uint32\_t ulChannel – номер канала DMA (4 - 7)

uint32\_t \*pusBuff – указатель на буфер для передачи

uint16\_t usSize – количество данных для передачи

**Описание**

Функция запускает выдачу данных на линию SPI с помощью прямого доступа к памяти.

**Пример**

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

uint32\_t **\_\_attribute**((aligned(4))) BufTx[50];

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

{

SPI\_Init\_type SPI;

uint32\_t i;

HAL\_SPI\_DeInit(LX\_SPI0);

HAL\_SPI\_DefaultInitStruct(&SPI);

HAL\_SPI\_Init (LX\_SPI0, &SPI);

//HAL\_GPIO\_Init (LX\_GPIO\_PA, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_7, GPIO\_PinMode\_Alt);

**for** (i = 0; i < 50; i++) {

BufTx[i] = i+1;

}

HAL\_Interrupt\_GlobalEnable();

HAL\_SPI\_DMA\_SendOnly(LX\_SPI0, 4, BufTx, 50);

**while**(!SPI\_TransferStatus.Spi0\_TxDmaReady);

}

## HAL\_SPI\_DMA\_ReceiveOnly

**Резюме**

**void** **HAL\_SPI\_DMA\_ReceiveOnly**( SPI\_type \*SPI, uint32\_t ulChannel, uint16\_t \*pusBuff, uint16\_t usSize )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

uint32\_t ulChannel – номер канала DMA (8 - 11)

uint32\_t \*pusBuff – указатель на буфер для передачи

uint16\_t usSize – количество данных для передачи

**Описание**

Функция запускает прием данных по SPI с помощью прямого доступа к памяти.

**Пример**

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

uint32\_t **\_\_attribute**((aligned(4))) BufRx[50];

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

{

SPI\_Init\_type SPI;

uint32\_t i;

HAL\_SPI\_DeInit (LX\_SPI0);

HAL\_SPI\_DefaultInitStruct (&SPI);

HAL\_SPI\_Init (LX\_SPI0, &SPI);

//HAL\_GPIO\_Init (LX\_GPIO\_PA, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_7, GPIO\_PinMode\_Alt);

HAL\_Interrupt\_GlobalEnable();

HAL\_SPI\_DMA\_ReceiveOnly(LX\_SPI0, 8, BufRx, 50);

**while**(!SPI\_TransferStatus.Spi0\_RxDmaReady);

}

## HAL\_SPI\_DMA\_SendAndReceive

**Резюме**

**void** **HAL\_SPI\_DMA\_SendAndReceive**( SPI\_type \*SPI, uint32\_t ulChannelRx, uint32\_t \*pusBuffRx, uint32\_t ulChannelTx, uint32\_t \*pusBuffTx, uint16\_t   
usSize )

SPI\_type \*SPI – указатель на структуру интерфейса SPI

uint32\_t ulChannelRx – номер канала DMA (4 - 7)

uint32\_t \*pusBuffRx – указатель на приемный буфер

uint32\_t ulChannelTx – номер канала DMA (8 - 11)

uint32\_t \*pusBuffTx – указатель на буфер для передачи

uint16\_t usSize – количество данных для передачи

**Описание**

Функция запускает отправку с одновременным приемом данных по SPI с помощью прямого доступа к памяти.

**Пример**

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

uint32\_t **\_\_attribute**((aligned(4))) BufRx[50];

uint32\_t **\_\_attribute**((aligned(4))) BufTx[50];

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

{

SPI\_Init\_type SPI;

uint32\_t i;

**for** (i = 0; i < 50; i++) {

BufTx[i] = i+1;

}

HAL\_SPI\_DeInit(LX\_SPI0);

HAL\_SPI\_DefaultInitStruct(&SPI);

SPI.LoopBack=*SPI\_LoopBack\_On*;

HAL\_SPI\_Init (LX\_SPI0, &SPI);

//HAL\_GPIO\_Init (LX\_GPIO\_PA, GPIO\_PIN\_4 | GPIO\_PIN\_5 | GPIO\_PIN\_7, GPIO\_PinMode\_Alt);

HAL\_Interrupt\_GlobalEnable();

HAL\_SPI\_DMA\_SendAndReceive(LX\_SPI0, 8, BufRx, 4, BufTx,50);

**while**(!SPI\_TransferStatus.Spi0\_RxDmaReady);

}

# Пример работы

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

uint32\_t **\_\_attribute**((aligned(4))) usTxDataMs[ 15 ] = { 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F, 0x70, 0x71, 0x72 };

uint32\_t **\_\_attribute**((aligned(4))) usTxDataSl[ 15 ] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };

uint32\_t **\_\_attribute**((aligned(4))) usRxDataMs[ 15 ];

uint32\_t **\_\_attribute**((aligned(4))) usRxDataSl[ 15 ];

uint32\_t i;

**#define** SPI2 1

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

{

SPI\_Init\_type xSpiConf;

LX\_CMU->CFG8.word = 0xFFFFFFFF;

HAL\_PLL\_CoreSetup( 150000 );

LX\_CMU->CFG8.b.SPI0\_DIS = 0;

LX\_CMU->CFG8.b.SPI1\_DIS = 0;

LX\_CMU->CFG8.b.SPI2\_DIS = 0;

LX\_CMU->CFG1.b.SPI1\_EN = 1;

LX\_CMU->CFG1.b.SPI2\_EN = 1;

/\* Настройка ведомого \*/

xSpiConf.Mode = *SPI\_Mode\_Slave*;

xSpiConf.WordSize = 32;

xSpiConf.CSNum = 1;

xSpiConf.CS\_Hold = *SPI\_CS\_Unhold*;

xSpiConf.CS\_Hold\_Delay = 0;

xSpiConf.LoopBack = *SPI\_LoopBack\_Off*;

xSpiConf.CLK\_Prescaler = 10;

xSpiConf.FirstBit = *SPI\_FirstBit\_LSB*;

xSpiConf.CLK\_Phase = *SPI\_CLK\_Phase\_Posedge*;

xSpiConf.CLK\_Polarity = *SPI\_CLK\_Polarity\_Low*;

xSpiConf.CS\_Active = *SPI\_CS\_Active\_Low*;

HAL\_SPI\_Init( LX\_SPI0, &xSpiConf );

HAL\_GPIO\_Init( LX\_GPIO\_PA, *GPIO\_PIN\_6*, *GPIO\_PinMode\_Alt* ); //> SPI0 MISO

HAL\_GPIO\_Init( LX\_GPIO\_PA, *GPIO\_PIN\_4*, *GPIO\_PinMode\_Alt* );

HAL\_GPIO\_Init( LX\_GPIO\_PA, *GPIO\_PIN\_8*, *GPIO\_PinMode\_Alt* );

HAL\_GPIO\_Init( LX\_GPIO\_PA, *GPIO\_PIN\_5*, *GPIO\_PinMode\_Alt* );

/\* Настройка ведущего \*/

xSpiConf.Mode = *SPI\_Mode\_Master*;

xSpiConf.WordSize = 32;

xSpiConf.CSNum = 0;

xSpiConf.CS\_Hold = *SPI\_CS\_Unhold*;

xSpiConf.CS\_Hold\_Delay = 0;

xSpiConf.LoopBack = *SPI\_LoopBack\_Off*;

xSpiConf.CLK\_Prescaler = 10;

xSpiConf.FirstBit = *SPI\_FirstBit\_LSB*;

xSpiConf.CLK\_Phase = *SPI\_CLK\_Phase\_Posedge*;

xSpiConf.CLK\_Polarity = *SPI\_CLK\_Polarity\_Low*;

xSpiConf.CS\_Active = *SPI\_CS\_Active\_Low*;

**#if** SPI2

HAL\_SPI\_Init( LX\_SPI2, &xSpiConf );

HAL\_GPIO\_Init( LX\_GPIO\_PB, *GPIO\_PIN\_0*, *GPIO\_PinMode\_Alt* ); //> SPI1 CLK

HAL\_GPIO\_Init( LX\_GPIO\_PB, *GPIO\_PIN\_1*, *GPIO\_PinMode\_Alt* ); //> SPI1 MOSI

HAL\_GPIO\_Init( LX\_GPIO\_PB, *GPIO\_PIN\_2*, *GPIO\_PinMode\_Alt* ); //> SPI1 CS

HAL\_GPIO\_Init( LX\_GPIO\_PB, *GPIO\_PIN\_3*, *GPIO\_PinMode\_Alt* );

**#else**

HAL\_SPI\_Init( LX\_SPI1, &xSpiConf );

HAL\_GPIO\_Init( LX\_GPIO\_PA, GPIO\_PIN\_13, GPIO\_PinMode\_Alt ); //> SPI1 CLK

HAL\_GPIO\_Init( LX\_GPIO\_PA, GPIO\_PIN\_14, GPIO\_PinMode\_Alt ); //> SPI1 MOSI

HAL\_GPIO\_Init( LX\_GPIO\_PA, GPIO\_PIN\_16, GPIO\_PinMode\_Alt ); //> SPI1 CS

HAL\_GPIO\_Init( LX\_GPIO\_PA, GPIO\_PIN\_15, GPIO\_PinMode\_Alt );

**#endif**

HAL\_Interrupt\_GlobalEnable();

HAL\_SPI\_DMA\_SendAndReceive( LX\_SPI0, 8, usRxDataSl, 4, usTxDataSl, 15 );

HAL\_SPI\_DMA\_SendAndReceive( LX\_SPI2, 9, usRxDataMs, 5, usTxDataMs, 15 );

**while**(!SPI\_TransferStatus.Spi0\_RxDmaReady);

// Обычная отправка данных:

**while**(1)

{

}

}

}

В данном примере SPI0 настраивается в режиме ведомого, а SPI2 в режиме ведущего. Обмен данными происходит с помощью DMA. При работе SPI0 в режиме ведомого в качестве входа CS необходимо выбирать CSNum = 1. Для SPI1 и SPI2 CSNum всегда равен нулю. В качестве ведущего в примере можно выбрать использовать SPI1 или SPI2.