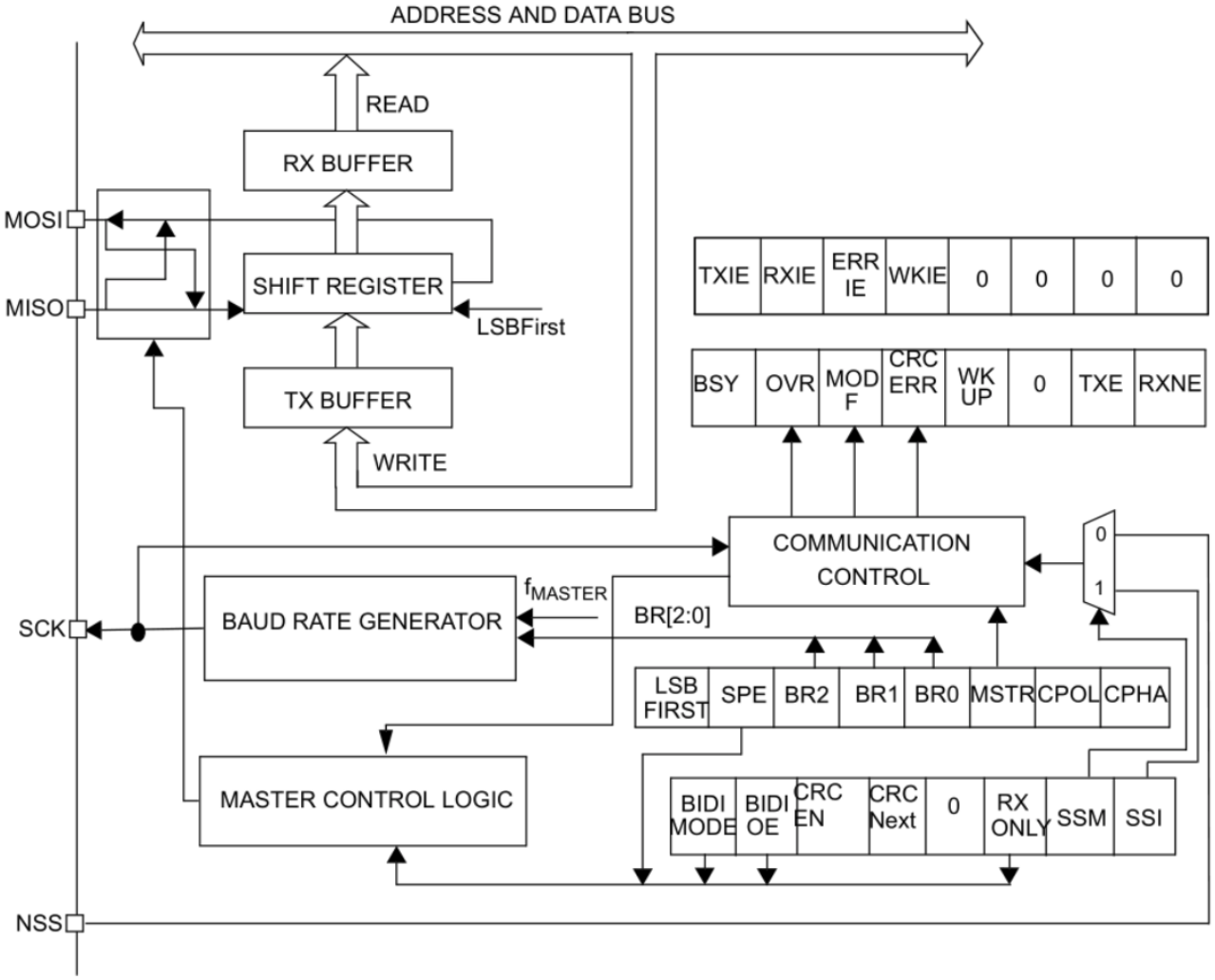
STM8S SPI настройка и использование в SPI.

Основная задача интерфейса SPI это обеспечение связью внутри схемы или на не большое растояние между контроллерами, датчиками и устройствами типа TM1638. О принципе работы, недостатках и преимуществах данного интерфейса я описывать не буду так как много информации интернете.

Для осуществления передачи информации между двумя устройствами или более, одно из них должно быть главным MASTER а все остальные подчиненные SLAVE, задача MASTER полный контроль передачи информации(подчиненные не могут сами начинать передовать данные только с разрешения MASTER). На MASTER лежат обязанности организации передачи данных, выбора устройства(если больше одного) и его тактирования, SLAVE устройство всегда ожидает когда его выберут через NSS вход и начнут тактировать для получения или отправки данных от них.

Теперь давайте ознакомимся с функциональной схемой из datasheet.



Из схемы мы видим что у SPI имеются четыре пина, коротко о них.

MOSI-если настроен модуль как MASTER то пин будет выходом для данных, как SLAVE будет входом.

MISO-тут наоборот MASTER вход для данных, SLAVE выход.

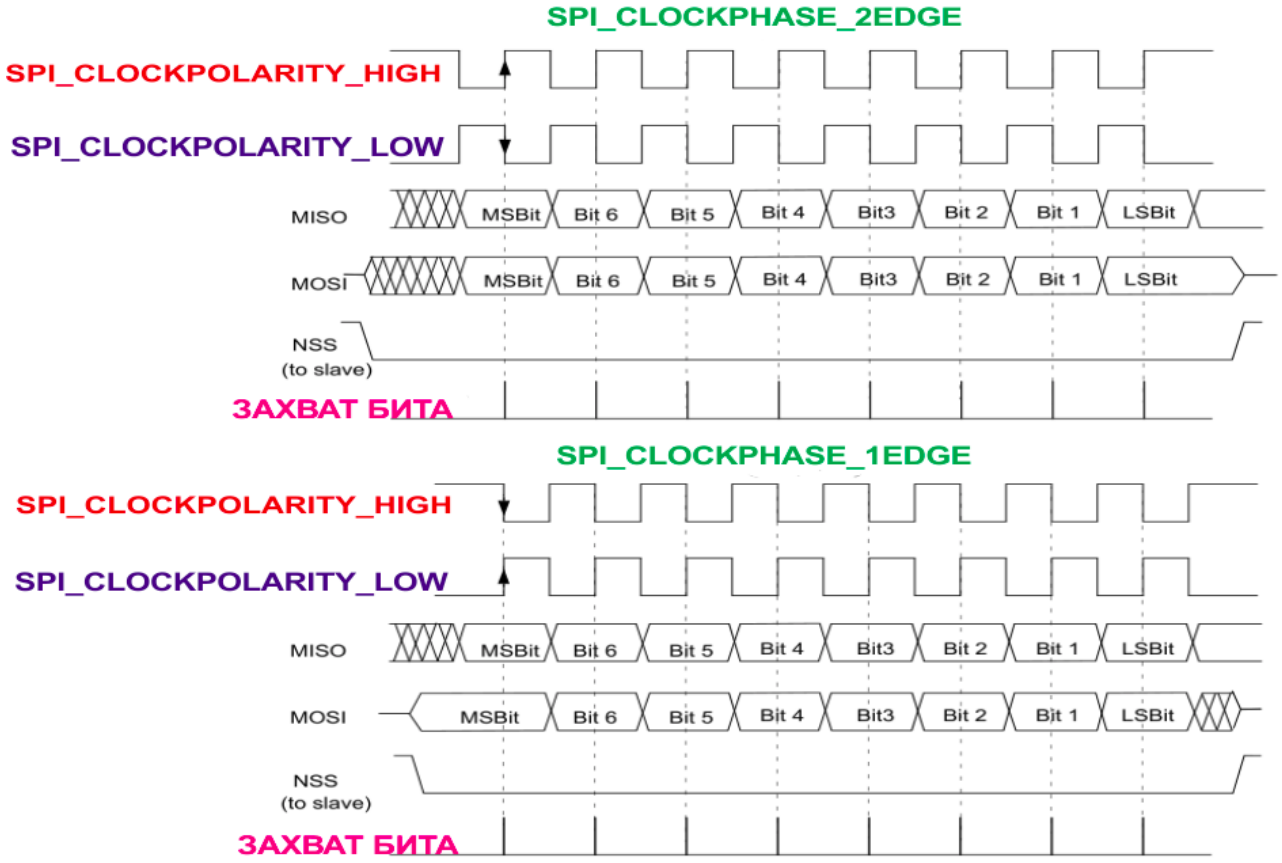
SCK-у MASTER(выход) тактирует SLAVE(вход) устройства.

NSS-через это пин MASTER разрешает работу SLAVE устройства, обычно когда SLAVE

несколько.

Описание функций SPL.

Начнем с основной функции.
SPI_Init(FirstBit, BaudRatePrescaler, Mode, ClockPolarity, ClockPhase, Data_Direction, Slave_Management, CRCPolynomial)-установка параметров SPI.
Где:
FirstBit- как будем отправлять байт данных:
SPI_FIRSTBIT_MSB-отправку начинаем со старшего бита заканчиваем младшим.
SPI_FIRSTBIT_LSB-отправку начинаем с младшего бита заканчиваем старшим.
BaudRatePrescaler-на какой частоте MASTER будет тактировать SLAVE устройства, частота тактирования контроллера делится на следующие варианты:
SPI_BAUDRATEPRESCALER_2
SPI_BAUDRATEPRESCALER_4
SPI_BAUDRATEPRESCALER_8
SPI_BAUDRATEPRESCALER_16
SPI_BAUDRATEPRESCALER_32
SPI_BAUDRATEPRESCALER_64
SPI_BAUDRATEPRESCALER_128
SPI_BAUDRATEPRESCALER_256
Mode-как будет работать модуль:
SPI_MODE_MASTER-режим MASTER(главный).
SPI_MODE_SLAVE-режим SLAVE(подчиненный).
ClockPolarity-высоким или низким уровнем будет тактовый сигнал:
SPI_CLOCKPOLARITY_LOW-в ожидании на выходе у MASTER будет низкий уровень, тактовый сигнал высокий.
SPI_CLOCKPOLARITY_HIGH-тут наоборот,в ожидании на выходе у MASTER будет высокий уровень, тактовый сигнал низкий.
ClockPhase-когда будет производится выборка бита данных, зависит от ClockPolarity:
SPI_CLOCKPHASE_1EDGE-выборка происходит по первому краю тактового сигнала.
SPI_CLOCKPHASE_2EDGE-выборка уже происходит по второму краю тактового сигнала.



Data_Direction-как будут подключены устройства и будет осуществляться обмен данными:
SPI_DATADIRECTION_2LINES_FULLDUPLEX-устройство одновременно может отсылать и принимать данные, принцип даю данные и одновременно забираю.
SPI_DATADIRECTION_2LINES_RXONLY-только принимает данные.
SPI_DATADIRECTION_1LINE_RX-настроено на прием но можно сменить на передачу.
SPI_DATADIRECTION_1LINE_TX-настроено на передачу но можно сменить на прием.
Slave_Management-для устройство работающее в режиме SLAVE как будет управляться NSS(выбор нужного SLAVE устройства или синхронизация передачи данных).
SPI_NSS_SOFT-программное управление, после инициализации разрешает использовать.
SPI_NSS_HARD-внешнее управление через вывод NSS, если MASTER выставит низкий уровень значит разрешает работу, высокий запрещает.
CRCPolynomial-данное число(полином) используется для вычисления контрольной суммы если используете CRC calculation, по умолчанию в нем установлено значение 7, если не используете контроль значение не важно, на передачу и прием данных не влияет.

SPI_DeInit( )-функция сбрасывает все настройки, выключает SPI и его прерывания.

SPI_Cmd(NewState)-функция включает если NewState ENABLE и выключает DISABLE модуль SPI.

SPI_ITConfig(SPI_IT, NewState)-функция включает если NewState ENABLE и выключает DISABLE нужные прерывания от SPI.
SPI_IT-необходимые прерывания:
SPI_IT_TXE-прерывание будет срабатывать когда будет свободен Tx буфер, даже если будет отправляться байт данных из сдвигового регистра.
SPI_IT_RXNE-прерывание срабатывает если Rx буфер не свободен то есть приняли байт данных.
SPI_IT_ERR-прерывание сработает если будут ошибки: CRC error неправильная контрольная сумма, Overrun condition пришел новый байт данных но старый не успели

прочитать(переполнение Rx буфера), Master mode fault (MODF) в режиме MASTER если используете вывод NSS для контроля SLAVE и кто то кроме него установит низкий уровень сработает это прерывание.
SPI_IT_WKUP-сработает когда контроллер выйдет из спящего состояния.

SPI_SendData(Data)-отправляет байт данных.

SPI_ReceiveData()-читает из Rx буфера байт принятых данных.

SPI_NSSInternalSoftwareCmd(NewState)-функция переключает управление NSS если NewState ENABLE с внешнего управления как вывод на внутреннее программное и NewState DISABLE наоборот управление через вывод.

SPI_TransmitCRC( )-используется для отправки контрольной суммы в случае когда работает CRC calculation.

SPI_CalculateCRCCmd(NewState)-функция включает если NewState ENABLE и выключает DISABLE подсчет контрольной суммы(CRC calculation).

SPI_GetCRC(SPI_CRC)-возвращает контрольную сумму если SPI_CRC SPI_CRC_TX отправленных данных и SPI_CRC_RX принятых.

SPI_GetCRCPolynomial( )-возвращает значение полином регистра.

SPI_ResetCRC( )-очищает регистр контрольной суммы отправленных и принятых данных.

SPI_BiDirectionalLineConfig(SPI_Direction)-используется для изменения направления передачи данных если MOSI используется для приема и передачи данных. SPI_Direction следующие варианты:
SPI_DIRECTION_RX-модуль будет настроен на прием данных.
SPI_DIRECTION_TX-модуль будет настроен на передачу данных.

SPI_GetFlagStatus(SPI_FLAG)-возвращает состояние флагов.
SPI_FLAG_OVR-переполнение Rx буфера.
SPI_FLAG_MODF-ошибка на NSS выводе в режиме MASTER.
SPI_FLAG_CRCERR-не правильная контрольная сумма.
SPI_FLAG_WKUP-выход из сна контроллера.
SPI_FLAG_TXE-Tx буфер пустой.
SPI_FLAG_RXNE-Rx буфер полон.
SPI_FLAG_BSY-осуществляется прием или передача данных.

SPI_ClearFlag(SPI_FLAG)-функция сбрасывает выше упомянутые флаги кроме SPI_FLAG_BSY, SPI_FLAG_TXE и SPI_FLAG_RXNE эти очищаются контроллером.

SPI_GetITStatus(SPI_IT)-проверяем включено или выключено нужное прерывание.

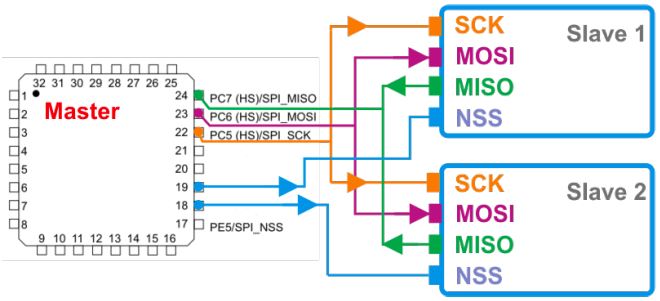
SPI_ClearITPendingBit(SPI_IT)-функция очищает флаги прерываний кроме SPI_IT_TXE сбрасывается когда Tx буфер полон и SPI_IT_RXNE очищается после прочтения принятого байта данных из Rx буфера, варианты SPI_IT:
SPI_IT_WKUP-выход из сна контроллера.
SPI_IT_OVR-переполнение Rx буфера.
SPI_IT_MODF-ошибка на NSS выводе в режиме MASTER.
SPI_IT_CRCERR-не правильная контрольная сумма.

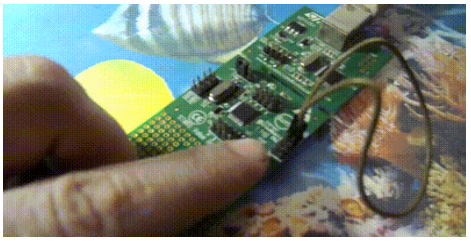
Вывод NSS-это вывод используется SLAVE устройствами что бы знать когда MASTER захочет с ним работать и помогает синхронизировать передачу данных. Со стороны MASTER этот вывод не используется, для управления подчиненными берется любой подходящий порт например как у меня на рисунке ниже PORTC пины 1,2(настраиваются как выходы), если на них будут высокий уровень значит запрещено

работу обоим устройствам, низкий уровень разрешит работу нужному устройству.
Теперь немного о тонкостях если будете использовать контролер как MASTER в режиме SPI_NSS_HARD то вывод NSS должен быть подтянут к питанию иначе SPI будет сброшен и переключен в подчиненного. Пример использования этого вывода ниже в SPI_DATADIRECTION_1LINE_RX и SPI_DATADIRECTION_1LINE_TX режимах.

Практика использования.

Подключение и передача данных в режиме SPI_DATADIRECTION_2LINES_FULLDUPLEX.





Ниже пример кода в котором при нажатии кнопки(использую отладочную плату stm8s value line discovery) SPI модуль будет отправлять байт самому себе, для этого просто соедините проводом выводы MOSI и MISO. При отправке программа зажжет штатный светодиод после удачного приема, проверки байта и небольшой паузы светодиод должен погаснуть, если захотите проверить просто измените отправляемый байт тогда светодиод будет только загораться.

```
#include "stm8s.h"
void pause(uint32_t p);
int main( void ){
//Настройка кнопки и светодиода.
GPIO_Init( GPIOB, GPIO_PIN_7, GPIO_MODE_IN_FL_NO_IT);
GPIO_Init( GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_HIZ_SLOW);
//Настраиваем выводы для SPI.
GPIO_Init( GPIOC, GPIO_PIN_5, GPIO_MODE_OUT_PP_LOW_FAST);
GPIO_Init( GPIOC, GPIO_PIN_6, GPIO_MODE_OUT_PP_LOW_FAST);
GPIO_Init( GPIOC, GPIO_PIN_7, GPIO_MODE_IN_FL_NO_IT);
//Настройка самого SPI как MASTER.
SPI_Init( SPI_FIRSTBIT_LSB, SPI_BAUDRATEPRESCALER_2, SPI_MODE_MASTER,
SPI_CLOCKPOLARITY_LOW, SPI_CLOCKPHASE_1EDGE, SPI_DATADIRECTION_2LINES_FULLDUPLEX,
SPI_NSS_SOFT, 7);
//Включаем SPI.
SPI_Cmd( ENABLE);
while(1){
//Проверяем нажатие кнопки, если нажата зажигаем светодиод и отправляем байт 111.
if(!GPIO_ReadInputPin( GPIOB, GPIO_PIN_7)){ GPIO_WriteLow( GPIOD, GPIO_PIN_0);
SPI_SendData(111);}
//Проверяем пришли данные.
if( SPI_GetFlagStatus( SPI_FLAG_RXNE)){
//Если пришли проверяем байт данных, если 111, ждем паузу и гасим светодиод.
if( SPI_ReceiveData()==111){ pause( 100000); GPIO_WriteHigh( GPIOD, GPIO_PIN_0);}
}
}
}
void pause( uint32_t p){
```

```
for( uint32_t i=0; i<p; i++){ }
}
#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line)
{
while (1){ }
}
#endif
```

Если вы будете использовать контроллер как SLAVE нужно будет изменить настройку выводов SPI вот так

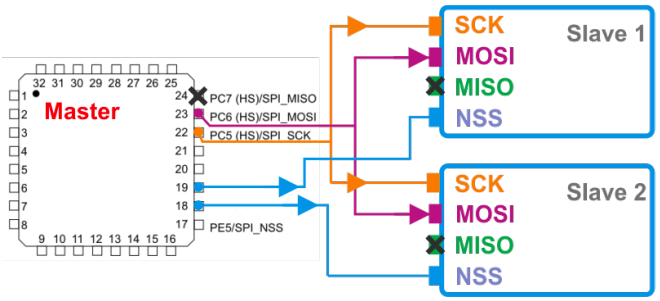
```
GPIO_Init( GPIOC, GPIO_PIN_5, GPIO_MODE_IN_FL_NO_IT);
GPIO_Init( GPIOC, GPIO_PIN_6, GPIO_MODE_IN_FL_NO_IT);
GPIO_Init( GPIOC, GPIO_PIN_7, GPIO_MODE_OUT_PP_LOW_FAST);
```

и основную настройку SPI

```
SPI_Init( SPI_FIRSTBIT_LSB, SPI_BAUDRATEPRESCALER_2, SPI_MODE_SLAVE,
SPI_CLOCKPOLARITY_LOW, SPI_CLOCKPHASE_1EDGE, SPI_DATADIRECTION_2LINES_FULLDUPLEX,
SPI_NSS_SOFT, 7);
```

Не забываем про NSS вывод если будете использовать внешнее управление то замените SPI_NSS_SOFT на SPI_NSS_HARD.

Передача данных в режиме SPI_DATADIRECTION_2LINES_RXONLY.



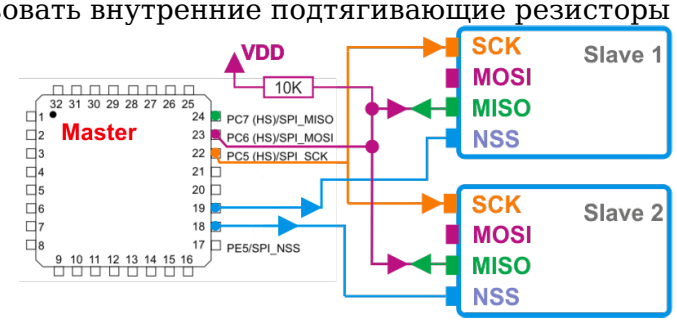
Рационально использовать этот режим только для SLAVE устройств которым нужно будет только принимать информацию, поэтому в подключении через MISO нет надобности. Все настройки аналогично предыдущему режиму как для SLAVE за исключение следующих.

```
SPI_Init( SPI_FIRSTBIT_LSB, SPI_BAUDRATEPRESCALER_2, SPI_MODE_SLAVE,
SPI_CLOCKPOLARITY_LOW, SPI_CLOCKPHASE_1EDGE, SPI_DATADIRECTION_2LINES_RXONLY,
SPI_NSS_SOFT, 7);
```

Подключение и передача данных в режиме SPI_DATADIRECTION_1LINE_RX и SPI_DATADIRECTION_1LINE_TX.

Эти два режима полностью одинаковы за исключением направленности передачи данных SPI_DATADIRECTION_1LINE_RX будет только принимать данные и SPI_DATADIRECTION_1LINE_TX отправлять, также по необходимости можно сменить направление без перенастройки SPI с помощью функцией SPI_BiDirectionalLineConfig(SPI_Direction). Если будете использовать данное подключение со сменной направлений данных тогда подтяните линию MOSI к питанию через резистор 10Ком,

настраивать сам пин не ненужно должны быть настройки по умолчанию, для SCK и NSS можно использовать внутренние подтягивающие резисторы контроллера.



Ниже пример для двух устройств, которые будут обмениваться данными, MASTER после нажатия кнопки(использую отладочную плату stm8s value line discovery) отправит байт 101, перенастроиться на прием, SLAVE будет ожидать прихода байта 101 после перенастроиться на отправку и отошлет байт 128, если пройдет все удачно светодиод должен поменять свое состояние(погаснуть или загореться). Если будет необходимость использовать прерывание то раскомментируйте нужные строки и удалите не нужные.

```
#include "stm8s.h"
void pause(uint32_t p);
uint8_t a;
int main( void ){
//Настройки выводов для кнопки и светодиода.
GPIO_Init( GPIOB, GPIO_PIN_7, GPIO_MODE_IN_FL_NO_IT);
GPIO_Init( GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_HIZ_SLOW);
//Настройки выводов для работы SPI.
GPIO_Init( GPIOC, GPIO_PIN_5, GPIO_MODE_OUT_PP_LOW_FAST);
//Этот вывод будет управлять NSS SLAVE.
GPIO_Init( GPIOC, GPIO_PIN_1, GPIO_MODE_OUT_PP_HIGH_FAST);
//Настраиваем SPI как MASTER на отправку данных.
SPI_Init( SPI_FIRSTBIT_LSB, SPI_BAUDRATEPRESCALER_64, SPI_MODE_MASTER,
SPI_CLOCKPOLARITY_LOW, SPI_CLOCKPHASE_1EDGE, SPI_DATADIRECTION_1LINE_TX,
SPI_NSS_SOFT, 7);
//Для использования прерывания раскомментируйте ниже две строки.
//SPI_ITConfig( SPI_IT_RXNE, ENABLE);
//enableInterrupts();
SPI_Cmd( ENABLE);
while(1){
//Проверяем нажатие кнопки, если нажата отправляем байт и ждем ответа.
if(!GPIO_ReadInputPin( GPIOB, GPIO_PIN_7)){
//Разрешаем работу SLAVE.
GPIO_WriteLow( GPIOC, GPIO_PIN_1);
//Отправляем 101.
SPI_SendData(101);
//Ожидаем отправки.
while(!SPI_GetFlagStatus( SPI_FLAG_TXE));
while(SPI_GetFlagStatus( SPI_FLAG_BSY));
//Запрещаем работу SLAVE.
GPIO_WriteHigh( GPIOC, GPIO_PIN_1);
//Ожидаем что бы SLAVE успел подготовить ответ.
pause(100);
//Разрешаем работу SLAVE.
GPIO_WriteLow( GPIOC, GPIO_PIN_1);
//Меняем направление на прием.
SPI_BiDirectionalLineConfig( SPI_DIRECTION_RX);
//Удалите все строки до ++++++до сюда++++++, если используете прерывание.
//Ждем когда придут данные.
while(!SPI_GetFlagStatus( SPI_FLAG_RXNE));
if( SPI_GetFlagStatus( SPI_FLAG_RXNE)){
a=SPI_ReceiveData();
if( a==128){ GPIO_WriteReverse( GPIOD, GPIO_PIN_0);}
GPIO_WriteHigh( GPIOC, GPIO_PIN_1);
SPI_BiDirectionalLineConfig( SPI_DIRECTION_TX);
}
//++++++до сюда++++++
pause(30000);
```

```
}
}}
//Обработчик прерывания, если не нужен удалите.
INTERRUPT_HANDLER(SPI_IRQHandler, 10){
a=SPI_ReceiveData();
if( a==128){ GPIO_WriteReverse( GPIOD, GPIO_PIN_0);}
GPIO_WriteHigh( GPIOC, GPIO_PIN_1);
SPI_BiDirectionalLineConfig( SPI_DIRECTION_TX);
}
void pause( uint32_t p){
for( uint32_t i=0; i<p; i++){ }
}
#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line)
{
while (1){ }
}
#endif
```

Пример для SLAVE настроенный для приема данных, программа будет отвечать предыдущему примеру.

```
#include "stm8s.h"
void pause(uint32_t p);
uint8_t a;
int main( void ){
//Настройки выводов для работы SPI.
GPIO_Init( GPIOC, GPIO_PIN_5, GPIO_MODE_IN_FL_NO_IT);
//Настраиваем NSS(SPI_NSS).
GPIO_Init( GPIOE, GPIO_PIN_5, GPIO_MODE_IN_FL_NO_IT);
//Настраиваем SPI как SLAVE для приема данных.
SPI_Init( SPI_FIRSTBIT_LSB, SPI_BAUDRATEPRESCALER_64, SPI_MODE_SLAVE,
SPI_CLOCKPOLARITY_LOW, SPI_CLOCKPHASE_1EDGE, SPI_DATADIRECTION_1LINE_RX,
SPI_NSS_HARD, 7);
//Для использования прерывания раскомментируйте ниже две строки.
//SPI_ITConfig( SPI_IT_RXNE, ENABLE);
//enableInterrupts();
SPI_Cmd( ENABLE);
while(1){
//Удалите все строки до ++++++до сюда++++++, если используете прерывание.
if(SPI_GetFlagStatus( SPI_FLAG_RXNE)){
a=SPI_ReceiveData();
if( a==101){
SPI_BiDirectionalLineConfig( SPI_DIRECTION_TX);
SPI_SendData(128);
while(!SPI_GetFlagStatus( SPI_FLAG_TXE));
while(SPI_GetFlagStatus( SPI_FLAG_BSY));
SPI_BiDirectionalLineConfig( SPI_DIRECTION_RX);
}
}
//++++++до сюда++++++
}}
//Обработчик прерывания, если не нужен удалите.
INTERRUPT_HANDLER(SPI_IRQHandler, 10){
if(SPI_GetFlagStatus( SPI_FLAG_RXNE)){
a=SPI_ReceiveData();
if( a==101){
SPI_BiDirectionalLineConfig( SPI_DIRECTION_TX);
SPI_SendData(128);
while(!SPI_GetFlagStatus( SPI_FLAG_TXE));
while(SPI_GetFlagStatus( SPI_FLAG_BSY));
SPI_BiDirectionalLineConfig( SPI_DIRECTION_RX);
}
}
}
void pause( uint32_t p){
for( uint32_t i=0; i<p; i++){ }
}
#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line)
```

```
{
while (1){ }
}
#endif
```

Комментариев нет **Только зарегистрированные пользователи могут оставлять комментарии!**

Если очень хочешь, то поддержи сайт!

Поддержать