STM32: SPI: LCD — Вы всё делаете не так [восклицательный знак]

- Программирование микроконтроллеров

Надеюсь сообщество простит меня за такой заголовок, просто в последнее время все чаще и чаще сталкиваюсь с программами в которых к микроконтроллерам **STM32** подключают различные дисплеи с интерфейсом **SPI** и очень часто передачу данных при этом делают не правильно.

Как следствие — либо код не работает совсем и тогда в него внедряют различные задержки, или пишут код таким образом что он гарантированно будет работать медленно (по сравнению с возможной скоростью). А кто то, не разобравшись просто копирует чужой «с костылями» код, и потом такие «произведения» ходят по интернету из примера в пример…

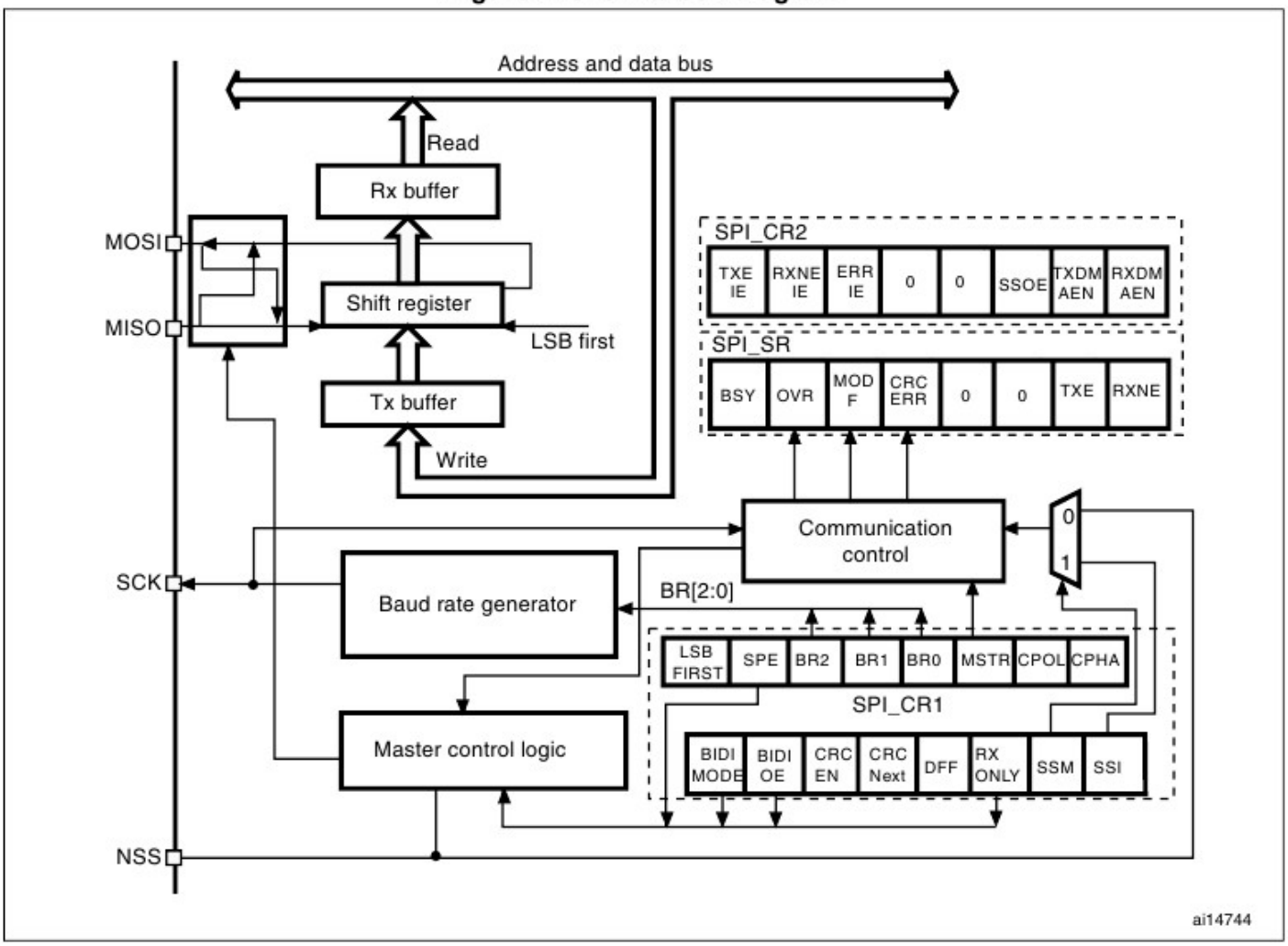
Блок SPI описанный в данной статье точно есть у контроллеров семейств: **STM32F1**, **STM32F2**, **STM32F4**. По другим смотрите **Reference Manual**.

Откуда растут такие проблемы и каким образом они решаются под катом.

Для начала я расскажу как работает интерфейс **SPI** при передаче данных в режиме **MASTER**.

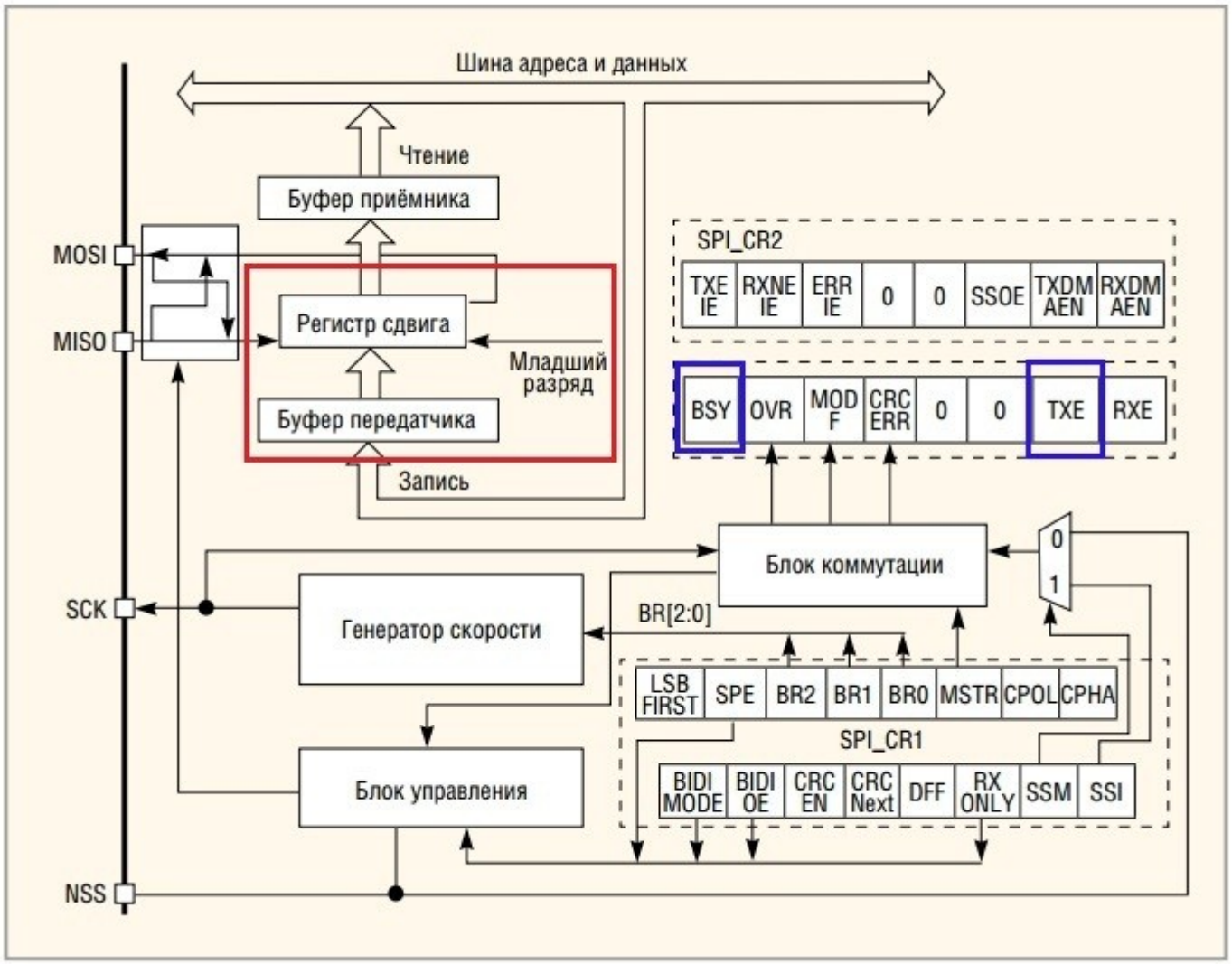
В Reference manual на стр. 868 есть наглядная схема устройства интерфейса:

Figure 246. SPI block diagram



если вы уже пробовали передавать данные при помощи SPI, то эту диаграмму вы уже знаете наизусть, ведь так?

наверняка для многих эта схема вообще уже выглядит исключительно понятной, кстати для тех кто ее еще не настолько изучил — я нашел ее русский вариант (документ источник доступен по клику на изображение):



ну что тут может быть проще?

при отправке данных мы записываем их в "**Буфер передатчика**", из которых они попадают в сдвиговый регистр и по очереди «выдаются» на линию **MOSI** (на схеме выделено красным цветом). При передаче генерируются флаги **BSY** и **TXE** по которым можно узнать состояние передачи.

Как у AVR ?

НЕТ !!

на схеме нарисовано абсолютно точно: при отправке используются два регистра:

— регистр "**Буфера передатчика**" (**SPI_DR**),

— **регистр сдвига** (shift register)

и это два регистра, КАЖДЫЙ из которых может иметь свое значение!!!

Для КАЖДОГО из этих регистров предусмотрен свой флаг — который показывает их заполненность.

- Для регистра **SPI_DR** — это флаг **TXE**
- Для **регистра сдвига** — это флаг **BSY**

Для того чтобы понять как вся эта связка двух регистров и двух флагов работает (в **Reference manual** я не нашел прямого ответа на этот вопрос, хотя если внимательно читать — то там это описывается в логике работы) — разберемся как происходит передача:

1. для отправки значения (8-ми либо 16-ти битного) мы записываем его в **SPI_DR**, одновременно происходит установка флага **TXE = 0** — что показывает что **SPI_DR** содержит значение для отправки
2. поскольку это первое отправляемое значение (до операции флаг **BSY = 0**), то значение записывается одновременно и в **SPI_DR** и в регистр сдвига (**Shift_Reg**), с которого первый бит (в зависимости от настроек **MSB/LSB**) выставляется по линии **MOSI**
3. В следующем **SCK** такте, после записи значения в **SPI_DR** и в сдвиговый регистр (**Shift_Reg**), устанавливается флаг **TXE = 1** — что означает что в регистр **SPI_DR** можно записать следующее значение для отправки. Обращаю внимание, прежнее значение содержится в **Shift_Reg** и еще выгружается на линию **MOSI**! (см. схему ниже! в момент установки флага **TXE = 1** происходит отправка лишь второго бита первоначального значения (из 8 или 16 бит значения)

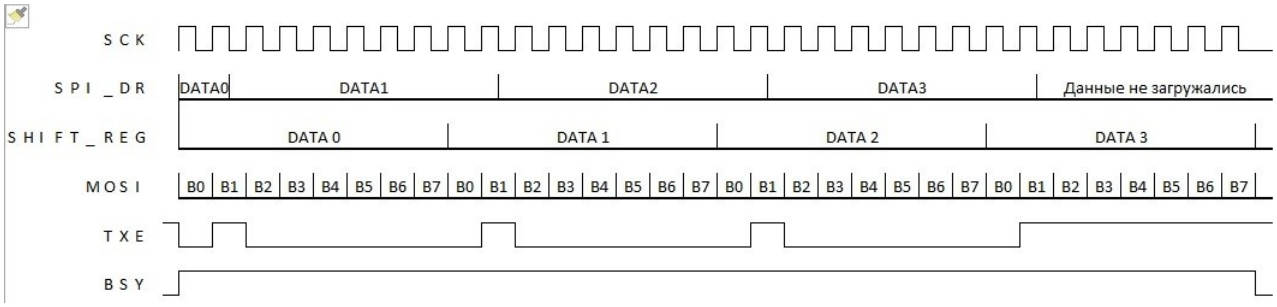
4. поскольку в **SPI_DR** больше данные не записываем — то с флагом **TXE =1** ничего и не происходит, интерфейс ждет загрузки следующего байта..
5. данные из **Shift_Reg** выгружаются по такту **SCK** на линию **MOSI**. При передаче последнего бита данных, проверяется есть ли новое значение в **SPI_DR** для отправки (в этом случае флаг **TXE = 0**), если нет (это флаг **TXE = 1**), то устанавливается флаг **BSY = 0** и передача прекращается.

Схематично передача одного байта по **SPI** будет выглядеть так:

Согласитесь это не сложно!!!

Зачем так сделано? — для того чтобы обеспечить непрерывность передаваемых данных!

В тех же **AVR**, при помощи **SPI**, невозможно передать несколько байт данных без перерыва, всегда между передаваемыми значениями будет пауза в 1 такт **SPI**. А вот в **STM32** возможна по настоящему непрерывная передача, которая будет выглядеть вот так:



Как видно из схемы — для обеспечения непрерывности передачи достаточно всего лишь ожидать установления флага **TXE = 1** и записывать в **SPI_DR** следующее значение для передачи.

Теперь о подключении дисплеев к STM32 по SPI.

1. 90% приведенных в интернете решений (не правильных решений) предлагают делать отправку данных на дисплей одним из следующих вариантов кода:

- Первый вариант, проверка флага TXE после загрузки данных в SPI_DR:

```
void SPISend(uint16_t data)
{
  SPI_I2S_SendData(SPI1, data);                                    // отправили данные
  while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET){}; // ждём, пока данные не отправятся
}
```

- Второй, на первый взгляд более правильный, вариант, проверка флага BSY после отправки данных в SPI_DR:

```
void SPISend(uint16_t data)
{
  SPI_I2S_SendData(SPI1, data);                                    //Передаем байт data через SPI1
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET){};   //Передатчик занят?
}
```

Оба варианта не правильны ! Кстати на просторах интернета еще предлагают третий вариант — проверять флаг **RXE** после отправки данных в **SPI_DR**, который обычно используется при получении каждого байта (слова) данных — это оставляю без комментариев…

Посмотрите внимательно на схемы которые я приводил выше! Флаг **TXE** нужно проверять перед отправкой данных в **SPI_DR**… Дальше будет работать конвейер самого **SPI** (в **MOSI** уйдет первый байт из**Shift_Reg**, потом **Shift_Reg** прогрузится значением из **SPI_DR**, и опять произойдет отправка в **MOSI**)

То есть код отправки данных на дисплей должен выглядеть следующим образом (пример для 16-ти битных посылок):

```
void SPISend(uint16_t data)
{
  while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET){};   // ждём, освобождения SPI_DR
  SPI_I2S_SendData(SPI1, data);                                        // отправим байт данных в очередь на отправку
}
```

Только в этом случае можно получить самую большую скорость обмена по SPI!

Почему же самый очевидный способ не используют?

Дело в том что многие дисплеи имеют наряду со стандартными **SPI** выводами (SCK, MOSI, MISO, CS) и такой вывод как **DC** (D/C, A0, CMD и так далее)

Вывод **DC** показывает что же передается в дисплей, обычно при **DC=0** дисплей воспринимает переданное как команду, а при **DC=1** — как данные.

Соответственно код отправки команды и данных после нее обычно пишут таким образом

```
// процедура отправки, правильный вариант который все равно не будет работать
void SPISend(uint16_t data)
{
  while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET){};   // ждём, освобождения SPI_DR
  SPI_I2S_SendData(SPI1, data);                                        // отправим байт данных в очередь на отправку
}


// сама процедура отправки команды и потом данных
void SPISendCommand(uint16_t data, data1, data2)
{
  SetCS0;                     // выбор дисплея для операции
  //[1]
  SetDC0;                     // установили режим передачи команд
  //[2]
  SPISend(commandCode);       // отправим команду
  //[3]
  SetDC1;                     // установили режим передачи данных
  SPISend(data);              // отправим данные
  SPISend(data1);             // отправим данные
  SPISend(data2);             // отправим данные
  //[4]
  SetCS1;                     // отмена выбора дисплея
}
```

Примечание к коду:

SetCSx — выбирает/отменяет выбор дисплея (здесь не привожу)

SetDCx — установка режима передачи команд/данных для дисплея

И этот код не работает !!!

Почему?

В точке [1] мы указываем дисплею что собираемся передавать команды, затем передаем код команды [2], но согласно нашей процедуры отправки и схем работы **SPI** которые я приводил выше — мы вернемся из подпрограммы отправки байта данных к шагу [3] к моменту отправки всего 2-3 бита команды (!!) — причем чем медленнее интерфейс **SPI** (ниже частота **SCK**) — тем меньше бит мы успеем передать!

И в этот момент мы указываем дисплею, что дальше идут данные [3] — у ЛЮБОГО дисплея смена состояния пина **DC** во время передачи команды/данных вызывает сбой!!!

Потом отправляем три байта данных (хотя команда уже не прошла), и в итоге отменяя выбор дисплея [4] мы окончательно «сносим голову» дисплею!!! ведь у нас согласно схемы работы **SPI** при отправке нескольких байт — скорее всего при выполнении команды **SetCS1** будет передаваться только data1 (он будет в регистре сдвига), а data2 будет еще ждать своей очереди в **SPI_DR**

Как большинство выходит из этой ситуации? — используют процедуру отправки с проверкой флага **BSY** после записи в **SPI_DR** (второй вариант решений который я приводил выше)… или вообще используют искусственные задержки!!! (например, командами delay !)

Другая крайность, это использование флага **BSY** везде! код работать будет, но вот о максимальной скорости передачи данных придется забыть, потери составят около 10-20% практически независимо от частоты **SCK** (!!), так как код будет постоянно ожидать установку **BSY=0** и только потом будет готовиться к следующей передаче (готовить следующий байт), и если это приемлемо и правильно при отправке команды (как правило один байт), то при отправке например буфера экрана, например для **PCD8544**(**Nokia 5110**) — будет работать заметно медленнее!!!

```
// отправка данных\команд на дисплей
void lcd8544_senddata(unsigned char data)
{
  SPI_I2S_SendData(SPI2, data);
  while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET){};      //Передатчик занят?
}

// очистка дисплея
void lcd8544_refresh(void)
{
```

```
  LCD_DC0;                                                  // режим передачи команд

  lcd8544_senddata(0x40);                                   // установка курсора в позицию Y=0; X=0
  lcd8544_senddata(0x80);

  LCD_DC1;                                                  // режим передачи данных

  unsigned char y, x;

  for (y = 0; y < 6; y++)
    for (x = 0; x < 84; x++)
      lcd8544_senddata(lcd8544_buff[y * 84 + x]);           // отправка буфера
}
```

хотя, как вы уже наверное догадались — выход лежит на поверхности — при необходимости смены вида передаваемых данных (по линии **DC**), или отмены выбора дисплея (линией **CS**) предварительно нужно проверять флаг **BSY** для того чтобы убедиться что физическая передача данных/команды завершилась. В остальных случаях нужно использовать проверку флага **TXE** ПЕРЕД загрузкой значения в **SPI_DR**:

```
// передача данных на дисплей
void SPI2_SendByte(uint8_t sendData)
{
  while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) == RESET){};
  SPI_I2S_SendData(SPI2, sendData);
}

// определение области для вывода
void ili9341c_SetWindow(uint16_t ystart, uint16_t xstart, uint16_t yend, uint16_t xend)
{
  //. . .
  GPIO_ResetBits(GPIOB, GPIO_Pin_12);                       // DC=0;

  SPI2_SendByte(LCD_PAGE_ADDR);

  while (SPI_I2S_GetFlagStatus(SPI2, SPI_FLAG_BSY) != RESET){};       //ждем пока команда уйдет на дисплей (а не перейдет в shift reg)

  GPIO_SetBits(GPIOB, GPIO_Pin_12);                         // DC=1;

  SPI2_SendByte(xstart >> 8);                               // данные
  SPI2_SendByte(xstart & 0xFF);                             // передаются
  SPI2_SendByte(xend >> 8);                                 // в непрерывном
  SPI2_SendByte(xend & 0xFF);                               // режиме без пауз !

  while (SPI_I2S_GetFlagStatus(SPI2, SPI_FLAG_BSY) != RESET){};       // ждем пока данные передадутся до конца

//. . .
}
```

этот код будет работать и максимально быстро и самое главное правильно!!!

некоторые статьи, где реализованы неправильные алгоритмы отправки, либо тема «правильной» отправки так и не раскрыта.

шел 2021-ый год, и как же я вспоминал эту статью, самим же собой написанную когда отлавливал глюки при подключении LCD на ST7735…

просто фото без комментариев :-)

(Фото на сайте похерено)

вот так вот бывает В любом правиле бывают исключения

p.s. Некоторое время назад я сам разбирался с этим интерфейсом, и был, в отличие от многих, удивлен его продуманностью и функциональностью, надеюсь теперь и для Вас интерфейс SPI у STM32 это не черный ящик с непонятно когда используемыми флагами, а четкий, понятный и продуманный автомат для максимально быстрой отправки/получения данных!

если что не так — пишите в комментарии к статье, в личку, или на емайл gorbukov @ тот_кто_знает_все. ru