Взгляд на SPI и ее реализацию в STM32 с точки зрения новичка в SPI и STM32. Примеры рассматриваю для протокола датчика давления LPS22HB с последовательным чтением серии регистров.
В STM32 для режима master есть, по сути, два режима работы: 4-х проводной и 3-х проводной, они же full duplex и half duplex. Режим только приема и только передачи — это частный случай half duplex.
По сигналам и именованиям:

- SCK — синхросигнал, выдается мастером.
- MOSI — Master Output, Slave Input — передача мастера.
- MISO — Master Input, Slave Output — прием мастера.
- NSS — Negative Slave Select — выбор чипа, активен в нуле.

Для начала разберем full duplex: это одновременная передача и прием. Всегда одновременная, даже, если идет только прием, все равно нужно что-то передавать. Судя по тому, что я читал на профильных форумах, это не редко ставит в тупик тех, кто в первый раз с этим сталкивается. Предположим, я опрашиваю датчик и в протоколе у меня 1 байт команды, после чего идет чтение или запись 1 и более байт данных. При записи казалось бы все просто: пишем команды и данные в SPIx_DR. Но нет, данные всегда еще и принимаются, их нужно считывать, если в будущем хотим принимать данные через этот же SPI. Вот диаграмма из мануала.

При чтении мы должны писать даже во время приема. Что именно писать — не регламентируется. Точнее, это может регламентироваться производителем устройства, а может и нет. Самое важное в full duplex: синхросигнал выдается только если есть, что передавать. Опустел буфер передачи — так же останавливается и прием.

Небольшое отступление: ST предлагает использовать свои библиотеки: HAL (Hardware Abstract Layer) и LL (Low Level). По моему, их наименование не совсем отвечает их функциям. Абстракцией являются обе. Только HAL больше скрывает реализацию алгоритмов работы периферии, а LL — меньше. Если использовать визард STM32CubeMX, он автоматически сгенерирует код конфигурации периферии МК. При этом HAL сразу предоставляет включенное и сконфигуренное устройство, а LL только сконфигуренное, включить, калибровать и прочее придется самому. Я сперва начал использовать HAL, но не люблю, когда от меня скрывают истинную работу, при непонятках это только сбивает с толку. По этому перешел на LL, а местами использую более низкоуровневый подход, ничем не уменьшающий уровень абстракции в сравнении с LL.

Реализацию чтения и записи я подсмотрел в HAL.
Код: (C)
```
// Чтение датчика LPS22: передаю номер регистра, если бит 7 установлен — это чтение.
// spi_retain() — активация NSS
// spi_release() — дезактивация NSS

// Для синхронизации используем флаг состояния SPI: RXNE (Receiver's buffer Not Empty).

uint8_t lps22_read(uint8_t reg) {
   spi_retain();
   LL_SPI_TransmitData8(SPI1, reg | 0x80);
   LL_SPI_TransmitData8(SPI1, 0xff);
   while (!LL_SPI_IsActiveFlag_RXNE(SPI1));
   LL_SPI_ReceiveData8(SPI1);
   while (!LL_SPI_IsActiveFlag_RXNE(SPI1));
   uint8_t data = LL_SPI_ReceiveData8(SPI1);
   spi_release();
   return data;
}

void lps22_write(uint8_t reg, uint8_t data) {
   spi_retain();
   LL_SPI_TransmitData8(SPI1, reg & 0x7f);
   LL_SPI_TransmitData8(SPI1, data);
   while (!LL_SPI_IsActiveFlag_RXNE(SPI1));
   LL_SPI_ReceiveData8(SPI1);
   while (!LL_SPI_IsActiveFlag_RXNE(SPI1));
   LL_SPI_ReceiveData8(SPI1);
   spi_release();
}
```

Буфер передачи и буфер приема по 4 байта. По этому можно сразу запихнуть пару байт в передачу и ждать окончания. Здесь приемо-передача будет пакетом, синхросигнал без разрыва.

Теперь посмотрим на half duplex. Я считаю избыточным 4-х проводной интерфейс для такого простого датчика. Попробуем 3-х проводку. В чем отличия от full duplex? Прием происходит по той же линии MOSI, что и передача. Передача ничем не отличается, но прием в буфер при этом не производится — не нужно читать пустышки. При переключении на прием MOSI переходит в режим входа, а синхросигнал генерируется непрерывно, пока мы не переключимся на передачу. Переключиться надо после окончания приема.
Вариант кода для half duplex.
Код: (C)
```
uint8_t lps22_read(uint8_t reg) {
   spi_retain();
```

```
    LL_SPI_TransmitData8(SPI1, reg | 0x80);
    while (LL_SPI_IsActiveFlag_BSY(SPI1));
    LL_SPI_SetTransferDirection(SPI1, LL_SPI_HALF_DUPLEX_RX);
    while (!LL_SPI_IsActiveFlag_RXNE(SPI1));
    uint8_t data = LL_SPI_ReceiveData8(SPI1);
    LL_SPI_SetTransferDirection(SPI1, LL_SPI_HALF_DUPLEX_TX);
    spi_release();
    return data;
}

void lps22_write(uint8_t reg, uint8_t data) {
    spi_retain();
    LL_SPI_TransmitData8(SPI1, reg & 0x7f);
    LL_SPI_TransmitData8(SPI1, data);
    while (LL_SPI_IsActiveFlag_BSY(SPI1));
    spi_release();
}
```

Сразу видно, что при передаче слежение за RXNE уже не имеет смысла. И слежение за TXE (Transmiter's buffer Empty) не имеет смысла, потому как он отражает состояние буфера, а не сдвигового регистра. По этому уже следим за флагом BSY. Приемо-передача уже не пакетом, после команды следует пауза, но никакой проблемы, кроме эстетической, оно не создает. А вот невозможность вовремя остановить прием, вот это проблема! Активация RXNE показывает не только конец приема очередного байта, но и начало следующего. Тут зависит от режима работы датчика на чтение: будет он отдавать повторно тот же свой регистр, который мы запросили, или инкрементирует внутренний указатель и отдаст следующий. Незапланированное чтение может создать посторонний эффект и вообще нарушить логику устройства. А так как мне нужен режим инкрементирования, то проблема...

Тут диаграмма подписана как full duplex, но для half duplex все тоже самое. Что характерно, документация ST старательно избегает half duplex в примерах кода и диаграммах.
Вот код для чтения последовательности регистров из датчика.
Код: (C)

```
void lps22_read_seq(uint8_t *buffer, uint8_t start_reg, uint8_t len) {
    spi_retain();
    LL_SPI_TransmitData8(SPI1, start_reg | 0x80);
    while (LL_SPI_IsActiveFlag_BSY(SPI1));
    LL_SPI_SetTransferDirection(SPI1, LL_SPI_HALF_DUPLEX_RX);

    while (len--) {
        while (!LL_SPI_IsActiveFlag_RXNE(SPI1));
        *buffer++ = LL_SPI_ReceiveData8(SPI1);
    }

    LL_SPI_SetTransferDirection(SPI1, LL_SPI_HALF_DUPLEX_TX);
    spi_release();
}
```

Ищу и наискиваю: документ AN5543 "Enhanced methods to handle SPI communication on STM32 devices". Выясняется, что у STM32 SPI есть версия реализации и от нее многое зависит. У меня чипы F0xx и L4xx — значится версия 1.3. Опция "Programmable transaction counters" доступна только для жирных чипов H7 и MP1 и для неясной категории "Most of STM32 devices launched in 2021 or later". Скорее всего тут имеются в виду свежие Gxxx чипы, но не уверен. Нам же контролировать длину пакета они предлагают ограничением длины передачи DMA. Но пардоньте, это передачу мы можем ограничить, а прием то как? Довести буфер до переполнения? В общем, непродуманно тут все. Кстати, попробовал обрывать передачу посредством NSS, работает нестабильно и неочевидно, короче, без гарантии результата. Также поиграл с задержкой + переключение на передачу, иногда срабатывает, но тоже очень ненадежно получается. Короче, мне оно не нравится и не подходит.

Вариант третий: МК в режиме full duplex, датчик — half duplex. Что по этому поводу думает интернет? Предлагают поставить на MOSI резистор и соединить через него с MISO. Номинал нигде не обоснован: где 1 кОм, где 3.3 кОм, где 10 кОм. У ардуинщиков нашел даже вариант, когда резисторы установлены на MOSI и MISO, а противоположные выводы соединены и подключаются к устройству — отличный пример, как делать не надо. Есть еще вариант 1 кОм + подтяжка MISO к VCC через 47 кОм, почему-то на схеме выход MOSI показан как открытый сток (open drain).

Что характерно, документация ST не запрещает такую конфигурацию.

А вот STM32CubeMX не позволяет выбрать режим, отличный от Pull-Push. Т.е. в коде мы можем написать что угодно, а кодогенератор визарда не дает. Если мы сгенерим код один раз и поправим, все отлично. Если воспользуемся визардом еще раз, он перезапишет конфигурацию. Переконфигурировать GPIO после конфигуратора? Костыльненько...
Кстати, а зачем там на схеме был резистор подтяжки? У нас в чипе встроенная подтяжка, ее можно включить на MISO! В общем, нет общего универсального решения.
Я пошел своим путем, тоже костыльным, но хотя бы мой костыль подпирает свой код, а не валяется посреди дороги. Для перехода на прием включаю режим аналогового вывода и возвращаю режим альтернативной функции после. Т.е. передатчик SPI на время не подключен к выводу.
Код: (C)

```
static uint8_t lps22_read(uint8_t reg) {
    spi_retain();
    LL_SPI_TransmitData8(SPI1, reg | 0x80);
    while (LL_SPI_IsActiveFlag_BSY(SPI1));
    LL_GPIO_SetPinMode(SPI_MOSI_GPIO_Port, SPI_MOSI_Pin, LL_GPIO_MODE_ANALOG);
    LL_SPI_TransmitData8(SPI1, 0xff);
    while (LL_SPI_IsActiveFlag_BSY(SPI1));
    LL_SPI_ReceiveData8(SPI1);
    while (!READ_BIT(SPI1->SR, SPI_SR_RXNE));
    uint8_t data = LL_SPI_ReceiveData8(SPI1);
    spi_release();
    LL_GPIO_SetPinMode(SPI_MOSI_GPIO_Port, SPI_MOSI_Pin, LL_GPIO_MODE_ALTERNATE);
    return data;
}

static void lps22_read_seq(uint8_t *buffer, uint8_t start_reg, uint8_t len) {
    spi_retain();
    LL_SPI_TransmitData8(SPI1, start_reg | 0x80);
    while (LL_SPI_IsActiveFlag_BSY(SPI1));
    LL_GPIO_SetPinMode(SPI_MOSI_GPIO_Port, SPI_MOSI_Pin, LL_GPIO_MODE_ANALOG);
    LL_SPI_TransmitData8(SPI1, 0xff);
    while (!LL_SPI_IsActiveFlag_RXNE(SPI1));
    LL_SPI_ReceiveData8(SPI1);

    while (len--) {
        if (len) {
            LL_SPI_TransmitData8(SPI1, 0xff);
        }

        while (!LL_SPI_IsActiveFlag_RXNE(SPI1));
        *buffer++ = LL_SPI_ReceiveData8(SPI1);
    }

    spi_release();
    LL_GPIO_SetPinMode(SPI_MOSI_GPIO_Port, SPI_MOSI_Pin, LL_GPIO_MODE_ALTERNATE);
}
```

Кстати, на счет подтяжки: я очень не уверен в ее пользе. К примеру, в спецификации датчика сказано, что у него на входах триггеры Шмидта с порогами 0.2 и 0.8 от VCC для 0 и 1 соответственно. Вот как выглядит подтяжка на осциллографе со стороны датчика. Стрелкой указан момент перевода MOSI в высокоимпедансное состояние, подтяжка дает примерно 0.5 В. Хорошо, что это передача датчика, а не прием.

Значит, во время передачи МК подтяжка не влияет, а при приеме задирает уровень нуля. Если сопротивление линии будет высоким, на МК может быть нестабильный прием. Для входов МК заявлены пороги 0.3 и 0.7 VCC. Без подтяжки выглядит отлично, не вижу смысла ее использовать.