## Получение данных с гироскопа

Гироскоп – это отдельный модуль на плате, который не входит в состав процессора, поэтому мы не можем напрямую обращаться к его регистрам, как мы это делали ранее. Связь между гироскопом и процессором реализована при помощи интерфейса SPI. Именно посредством этого интерфейса мы и будем получать показания с гироскопов.

## Принцип работы интерфейса SPI

Интерфейс SPI (Serial Peripheral Interface – последовательный периферийный интерфейс) используется для передачи информации между микросхемами. В нашем случае обмен будет производится между процессором и датчиками (а точнее, их контроллерами). Логика работы интерфейса устроена следующим образом. Есть, как минимум, два устройства, причем одно из них должно быть ведущим (Master) а второе ведомым (Slave). Ведомых устройст может быть несколько, ведущее устройство должно быть одно. Между ведущим и ведомым устройством есть три основных канала, которые называются SCK (system clock, иногда SCLK), MOSI(Master Out Slave In), MISO (Master In Slave Out). Так же существует канал CS (chip select, иногда SS – slave select) котрый нужен для определения с каким именно ведомым устройством работает ведущее устройство. Принципиальные схемы организации интерфейса SPI изображены на рисунках №1, 2

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/e/ed/SPI_single_slave.svg/250px-SPI_single_slave.svg.png | https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/SPI_three_slaves.svg/800px-SPI_three_slaves.svg.png |
| Рисунок №1 – SPI c одним ведомым устройством | Рисунок №2 – SPI c несколькими ведомыми устройствами |

Всё управление обменом со стороны микросхемы Master. Обмен происходит по следующему сценарию. В канале CS Master выставляет сигнал, который сообщает устройству, с которым будет вестить собмен, что инициирована операция обмена. Вслед за этим в канале SCLK появляется тактирующий сигнал. По возрастающим фронтам тактирующего сигнала передаются биты от Master к Slave (по линии MOSI). По противоположным фронтам передаются биты от Slave к Master (по линии MISO). Отметим также, что Slave будет передавать биты только в ответ на переданные ему биты. То есть, чтобы считать данные от Slave, всё равно нужно ему что-то да отправить. Обычно, подобные данные называют dummy. Данные для передачи находятся в специальных регистрах. Далее разберём настройку и ргинизацию запииси/чтения по SPI. Это нам будет необходимо, потому что гироскопы на плате подключены именно через этот интерфейс.

## Настройка SPI

Прежде чем пытаться считать какие-нибудь данные с гироскопа нужно настроить SPI. Первым делом необходимо найти через какие контакты реализованы основные линии SPI. Для этого обратимся к документации на плату. В конце документа есть несколько схем, нам поднадобится та, что изображена на рисунке №3, а конкретно та её часть, которая демонстритует блок, связанный с датчиками (MEMS) – рисунок №4.

|  |
| --- |
|  |
| Рисунок №3 – общая схема |

|  |
| --- |
|  |
| Рисунок №4 – схема гироскопа L3GD20 |

Итак, на схеме мы видим шесть контактов, идущих от гироскопа к процессору. К интерфейсу SPI относятся: SPI1\_SCK, SPI1\_MOSI, SPI1\_MISO и CS\_I2C/SPI. Остальные две линии нужны для сигналов прерываний (interruption). Итак, наши линии ведут к ножкам GPIO:

|  |  |
| --- | --- |
| SCK | PA5 |
| MOSI | PA7 |
| MISO | PA6 |
| CS | PE3 |

Нужно не забыть включить соответствующие банки контактов (А и Е). После включения нужно настроить эти контакты. Для обмена по SPI (да и не только для него) предусмотрены специальные режимы работы. При первой настройке ножек мы обращали внимание, что кроме режимов «вход» и «выход» есть еще режим альтернативной функции. Описание режимов альтернативных функций есть в документации на девайс. Там есть несколько специальных таблиц (для каждого банка), которые показывают какие варианты альтернативных функций существуют для каждой ножки. Приведём пример из такой таблицы (рисунок №5).

|  |
| --- |
|  |
| Рисунок №5 – таблица альтернативных функций |

Находим в ней, что для ножки PA5 альтернативная функция SPI1\_SCK обозначается как AF5, для PA6 и PA7 необходимые нам альтернативные функции тоже обозначаются как AF5. Нам понадобится это чуть дальше. В таблице для банка Е находим ножку PE3 и оказывается, что для неё нет одельной альтернативной функции, работающей как CS. Поэтому придётся настраивать её на выход и записывать в регистр данных какое значение мы хотим видеть на ножке в конкретный момент. При работе с альтернативными функциями не достаточно просто установить в регистре GPIO\_MODER режим альтернативной функции. У каждой ножки может быть несколько альтернативных функций, поэтому существует отдельный регистр (а точнее даже два), которые отвечают за выбор конкретной альтернативной функции. Переходим в документацию на контроллер и находим там регистр GPIO\_AFRL (alternate function register). Этот регистр отвечает за альтернативные функции первых восьми ножек банка. Он показан на рисунке №6.

|  |
| --- |
|  |
| Рисунок №6 – Регистр GPIO\_AFRL |

А на рисунке №7 показаны значения, которые может принимать каждое поле этого регистра.

|  |
| --- |
|  |
| Рисунок №7 – Значения полей регистра GPIO\_AFRL |

Нам нужно настроить ножки PA5-PA7 таким образом, чтобы их альтернативная функция была AF5. То есть в поля с битами 20-23, 24-27, 28-31 нужно записать число 01012.

Кроме настройки этих линий нужно ещё задать некоторые настройки SPI. Для этого нам понадобится регистр SPI1\_CR1 и SPI1\_CR2. Из основных задач здесь: настроить скорость, задать состояние Master и включить SPI. Рекомендуется настроить SPI с помощью следующих строчек кода, во избежание осложнений, связанных с последовательностью установки бит и прочим:

SPI1->CR2 = 0x1700;//Настроили Control Register 2

SPI1->CR1 = 0x357;//Настроили Control Register 1

Рекомендуется всё же поинтересоваться, каким образом данные строки настраивают SPI.

После настройки нужно приветси линию CS в выключенное состояние (нет обмена между устройствами). В нашем случае, нужно подать на эту линию единицу. Иными словами, нужно записать единицу в третий бит регистра GPIOE\_ODR (ножка PE3 предварительно должна быть настроена на выход).

## Чтение и запись по SPI

Итак, линии настроены и мы управляем устройством Master. Для обмена по SPI мы должны первым делом установить линию CS в нужное состояние (в данном случае на линии должен быть нуль). После этого нужно убедиться, что буфер для передачи пуст, и, в случае, если это не так, подождать, когда он станет пустым. Состояние будера передачи определяется первым битом регистра SPI1\_SR. Когда этот буфер пуст мы готовы обмениваться данными. Чтобы отправить данные нужно записать их в регистр SPI1\_DR (data register). Будем отправлять один байт. Чтобы не было ошибок, связанных с размером регистра, явно приведём его к нужному указателю:

\*((unsigned char\*)&SPI1->DR) = byte; //Записываем значение byte, которое хотим передать

Далее, если всё настроено правильно, начнётся обмен с гироскопом. Нам нужно подождать, пока он не закончится. Узнать об этом можно по биту 0 регистра SPI1\_SR. После окончания обмена нужно вычитать из регистра SPI1\_DR полученое от гироскопа значение, даже если оно нам и не нужно.

Выделим этот функционал в отдельную функцию, которая будет отправлять по SPI заданный байт. Выглядеть она будет примерно следующим образом:

unsigned char sendOneByteToGyroscope(unsigned char byte)

{

while(!(SPI1->SR & 2)) //Ждём пока transmit buffer not empty

{

}

\*((unsigned char\*)&SPI1->DR) = byte;

while(!(SPI1->SR & 1)) //Ждём пока Receive buffer empty

{

}

return SPI1->DR;

}

Теперь мы готовы читать регистры гироскопа. О том, какие регистры есть у гироскопа можно узнать из документации на гироскоп. Начнём с первого из них, который называется WHO\_AM\_I. Этот регистр содержит некоторый постоянный идентификатор устройства. В нашем случае он должен быть равен 0xd4. Регистр этот расположен по адресу 0x0F.

Итак, как происходит обмен по SPI с точки зрения программирования? Откроем документацию на гироскоп, раздел SPI Bus Interface. В этом разделе находим схему (рисунок №8):

|  |
| --- |
|  |
| Рисунок №8 – Протокол обмена с гироскопом |

Эта схема даёт ответ на то, как считать и записать данные гироскопа. Здесь SPC – то же что и SCK, а SDI и SDO это линии MOSI и MISO. Сначала задаём на линии CS нуль, который будем держать до конца обмена. На линии SCK появляется тактирующий сигнал. Больше всего нас интересует линия SDI. Здесь показано, что первым битом отправляется бит, обозначающий режим «чтения/записи». Чтобы считать данные нужно отправить – 1, чтобы считать – 0. Далее отправляется «multiple/single». Вернёмся к нему чуть позже. Следующие шесть бит – это адрес регистра гироскопа с которым будет проводится операция (чтение или запись). Далее на схеме изображены некоторые биты даных идущих как от Master (DI) так и от Slave (DO). На самом деле, Slave посылает какие-то биты и во время отправки первых восьми битов (адреса, и режима работы), но они никакой информации не несут, так что на схеме не изображены (однако их всё равно придётся считывать). Итак, если мы хотим считать данные регистра, нам нужно установить бит чтения равным 1 и отправить адрес, по которому мы хотим производить чтение. После этого нужно отправить столько битайт, сколько мы хотим считать. Вот здесь нам и нужен бит «multiple/single». Он говорит устройству Slave, как мы хотим считать регистры. Если установить в этот бит 1, то Slave будет отправлять значение регистров начиная с того адреса, который мы задали и до того момента, пока мы отправляем ему какие-либо байты. Он автоматически будет переключаться на следующий регистр. Это удобно когда требуется считать несколько регистров, расположенных вместе. Если мы хотим считать один регистр, мы устанавливаем бит «multiple/single» равным нулю и всё равно отправляем один dummy-байт (каждый регистр гироскопа содержит один байт информации). С записью регистров всё то же самое. После окончания сеанса обмена необходимо установить на линию CS единицу. Прочитаем регистр WHO\_AM\_I используя уже готовую функцию отправки байта в гироскоп (sendOneByteToGyroscope). Для этого напишем функцию, которая будет заниматься более общими вещами: считывать значение одного регистра. Итак, внутри функции мы задаём режим «чтение» и «single», а затем дописываем адрес, который передан в функцию (чтобы с помощью этой функции можно было читать любые регистры). Отправляем первый байт, ждём окончания записи и отправляем dummy-байт, чтобы в ответ на него получить значение регистра и вернуть его пользователю этой функции. В итоге функция будет выглядеть так:

unsigned char readGyroscopeRegister(int addres)

{

GPIOE->ODR &= ~(1 << 3);//Опустили чип селектор

unsigned char firstMsg = (addres | (1 << 7)); //Сформировали первый байт

unsigned char ret;

sendOneByteToGyroscope(firstMsg);//Задали режим чтения и адрес с которого нужно читать

ret = sendOneByteToGyroscope(0x00); //Отправили dummy-байт, считали значение регистра

GPIOE->ODR |= (1 << 3);//Подняли чип селектор

return ret;

}

Использование этой функции для чтения регистра WHO\_AM\_I будет таким:

unsigned char whoAmIValue = readGyroscopeRegister(0x0F); //Считывание WHO\_AM\_I

Функцию для чтения нескольких регистров подряд предлагается написать самостоятельно. Вместо общей функции чтения нескольких регистров можно создать более частную функцию для считывания показаний гироскопов. Показания гироскопов расположены в регистрах OUT\_X\_L – OUT\_Z\_H. То есть под каждый гироскоп отведено по два регистра, все регистры расположены последовательно, так что читать их лучше все вместе. Очевидно, что внутри функции следует не просто считать значения этих регистров, но и правильно их соединить и получить некоторое целое число(32-х битное), которое характеризует угловую скорость (показания гироскопах в кодах). Так как мы не можем вернуть из функции сразу три значения, то есть два варианта, как предоставить пользователю функции возможность получить их. Либо передать как аргументы функции указатели на область памяти в которой пользователь ожидает увидеть результат выполнения функции, либо ввести структуру, которая будет содержать в себе три поля подходящего типа и возвращать переменную-экземпляр такой структуры.

Запись регистра будет происходить таким же образом (необходимо реализовать отдельную функцию для записи), только в первом бите будет 0, а вместо dummy-байта будет отправляться желаемое значение регистра.