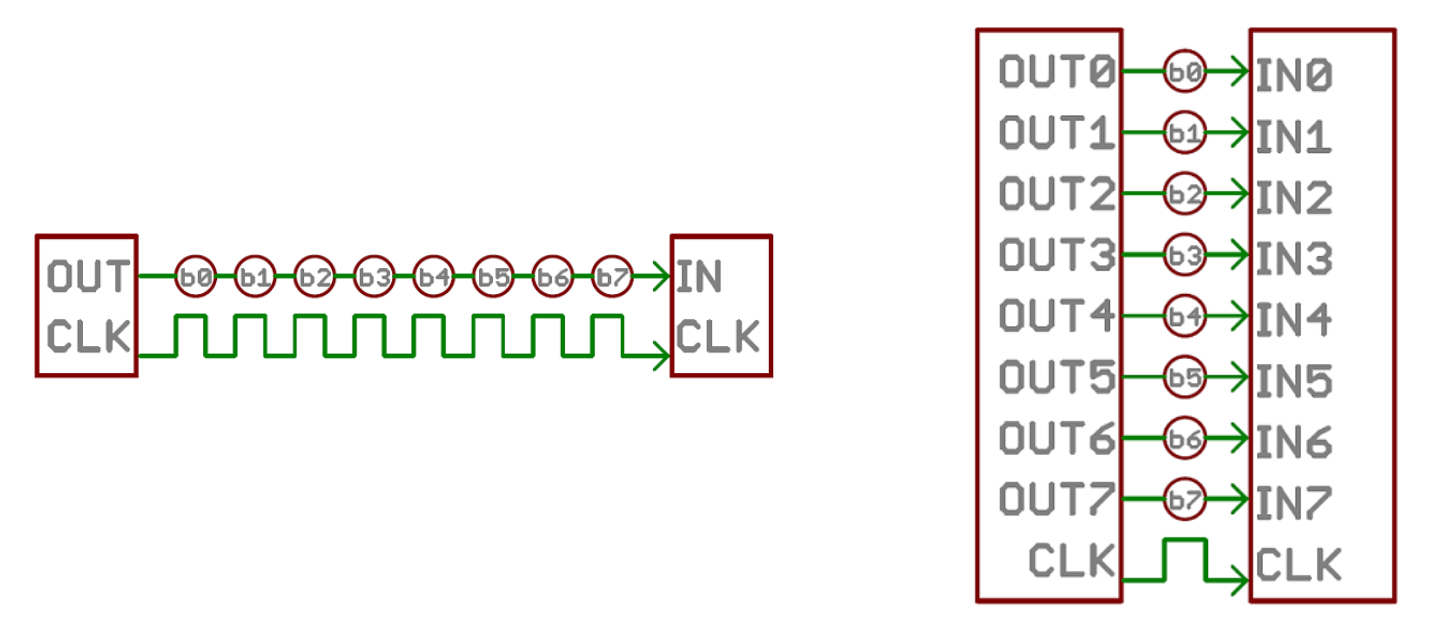
# Последовательная связь

Последовательная передача данных заключается в передаче одного бита за раз, а не многих битов (как при параллельной передаче данных). Параллельная схема требует как минимум N линий при передаче N битов, в то время как последовательная схема может обойтись только 1 линией. Во многих случаях для арбитража операций передачи используются дополнительные линии (например, тактовая линия). На следующей схеме показаны последовательная и параллельная версии 8-битной схемы передачи данных, в которой используется тактовая линия.

  
**Сравнение последовательной (слева) и параллельной (справа) передачи данных 8 бит**(взято из [SparkFun Electronics](https://learn.sparkfun.com/tutorials/serial-communication), предоставлено по [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/))

GPIO хорош для чтения нескольких входных переключателей или кнопок и управления несколькими выходами, такими как светодиоды или двигатели, но он не справляется, когда нам нужно передавать сообщения туда и обратно между микроконтроллером и другими устройствами. Это может звучать не очень интуитивно, если мы рассмотрим преимущество параллельной связи, передавая все N битов одновременно с помощью N линий GPIO. Однако наличие N линий сделает обмен данными только в N раз быстрее, чем последовательный вариант. Недостатки параллельной связи обычно перевешивают это преимущество. Вот несколько недостатков для обмена 32-битными словами:

* Для связи двух микросхем на одной плате требуется 32 линии на печатной плате, что заняло бы значительную площадь.
* Для связи двух микросхем на отдельных платах требуется кабель или лента с 32 проводами. Более толстые кабели тяжелее и с ними сложнее обращаться.
* 32 электрические линии в кабеле или печатной плате могут иметь 32 проблемы с электромагнитными помехами.
* Невозможно использовать инфракрасную связь (как у телевизионного пульта) из 32 параллельных линий, так как 32 мигающих огонька будут мешать друг другу.
* Использование 32 волоконно-оптических кабелей было бы очень дорого.
* Использование 32 радиоканалов также было бы неудобным.

Серьезность этих проблем снижается или устраняется при последовательном обмене данными. Фактически, несколько интерфейсов последовательной связи превосходят свои параллельные аналоги. Один из известных примеров - интерфейсы дисковых накопителей, где Serial ATA намного быстрее Parallel ATA. Самый быстрый интерфейс Parallel ATA способен передавать данные со скоростью 133 МБ в секунду, тогда как самый быстрый Serial ATA - 600 МБ в секунду.

  
  
**7-контактный кабель Serial ATA и 40-контактный ленточный кабель Parallel ATA**   
(левое изображение взято из [Wikimedia Commons](https://commons.wikimedia.org/w/index.php?curid=135185), предоставлено по лицензии [CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/), правое изображение взято из [Wikimedia Commons](https://commons.wikimedia.org/w/index.php?curid=11450019), By Evan Amos, Public Domain)

# Параллельная связь - это не плохо

Цель предыдущего обсуждения - показать обстоятельства, при которых последовательная связь предпочтительнее параллельной.

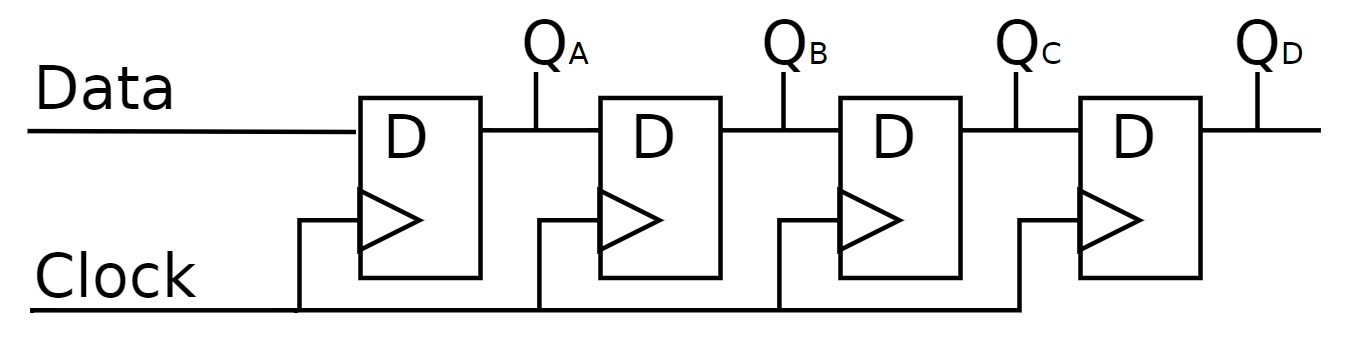
Однако параллельная связь находит применение во многих областях вычислительной техники. Одно из самых сильных применений - в архитектуре компьютера, где у нас есть схема шины для соединения центрального процессора, памяти и устройств ввода-вывода. Это модель памяти, используемая в языке C. Фактически, когда мы используем адрес памяти для доступа к данным, этот адрес в конечном итоге преобразуется в физический адрес, который передается по параллельной шине адреса.

Адресованные данные будут доступны в параллельной шине данных. Хотя существуют последовательные шины, такие как I2C и USB, внутренняя шина микроконтроллера всегда была параллельной, и не похоже, что эта практика изменится в ближайшем будущем.

# Сдвиговые регистры

Последовательная передача данных осуществляется путем взятия слова определенной ширины (например, 8, 32 или 64 бита) и последовательной передачи всех битов по одному каналу. Именно эту операцию и выполняет сдвиговый регистр. Вы можете быть знакомы со сдвигом битов с точки зрения программиста. Он заключается в сдвиге всех битов вправо или влево, обычно на одну позицию за раз. Для последовательной связи нас интересует бит, который "выпадает" из регистра после сдвига.

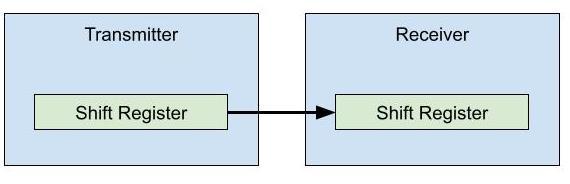
Регистр сдвига - это последовательность D флип-флопов, которые выполняют операцию сдвига между битами, хранящимися в каждом флип-флопе. Здесь мы имеем 4-битный регистр сдвига:

  
**4-битный регистр сдвига**   
(взято из [Wikimedia Commons](https://commons.wikimedia.org/w/index.php?curid=8543185), автор MichaelFrey, предоставлено по лицензии [CC BY-SA 2.0](https://creativecommons.org/licenses/by-sa/2.0/))

На приведенной выше схеме каждый раз, когда на линии *Clock* возникает нарастающий фронт импульса, все биты сдвигаются вправо. Регистр может иметь механизм для параллельного чтения и записи своих данных (все 4 бита одновременно).

Линия *Data* служит в качестве последовательного входа, а линия **QD** - в качестве последовательного выхода.

Все последовательные коммуникационные интерфейсы используют сдвиговый регистр в той или иной части своего оборудования, поскольку им необходимо отправлять некоторые данные по одному биту за раз. На следующей схеме показано, что общего у всех схем последовательной передачи данных.

  
**Простая сеть последовательной связи**

Здесь у нас есть передатчик, который посылает слово из N бит (например, 8 или 32 бита) приемнику, используя их соответствующие сдвиговые регистры. Через N тактов слово, которое первоначально было в передатчике, окажется в приемнике. На схеме показаны передатчик и приемник, но обратите внимание, что сдвиговый регистр в так называемом приемнике может отправить свой крайний правый бит обратно в так называемый передатчик, что делает эту операцию не передачей, а *подменой*.

В некоторых схемах элементы этой сети называются не передатчиком и приемником, а *ведущим* и *ведомым*, где ведущий управляет обменом данными, а ведомые принимают данные, отправленные ведущим, или отвечают данными, запрошенными ведущим.

Одно из основных различий между множеством последовательных коммуникационных интерфейсов заключается в том, откуда поступает тактовый сигнал для каждого сдвигового регистра:

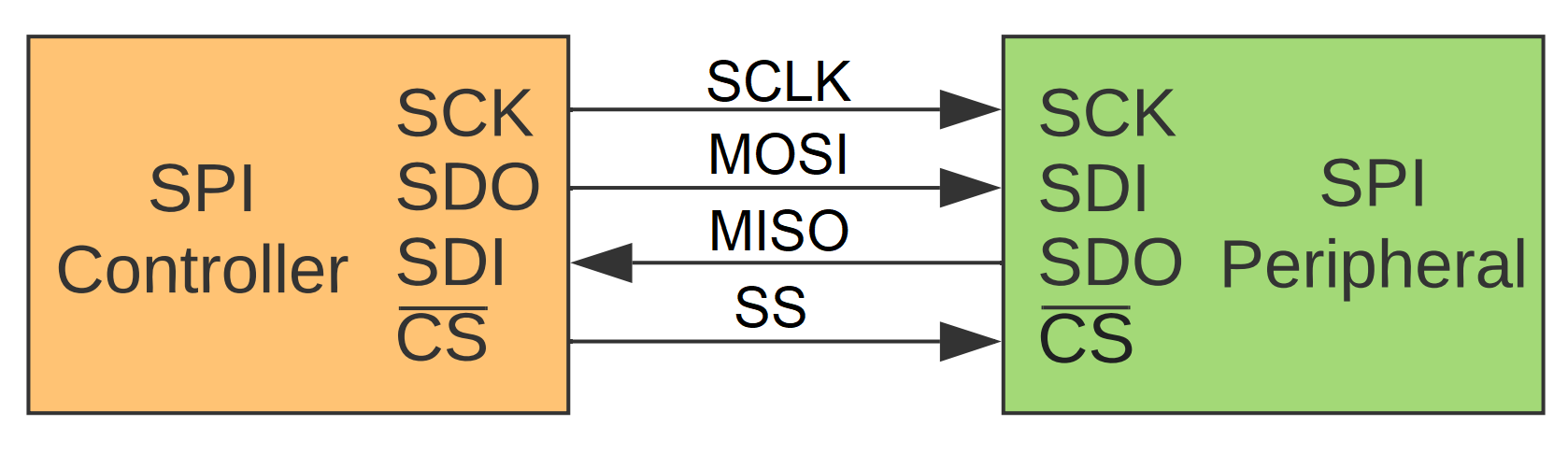
* В SPI он поступает от ведущего устройства, а ведомые устройства получают этот сигнал от ведущего устройства по отдельной линии.
* При асинхронной последовательной связи передатчик и приемник имеют собственные тактовые сигналы, работающие на одинаковой частоте, чтобы обеспечить соответствие скорости передачи данных.
* В I2C тактовый сигнал также посылается ведущим устройством, но ведомые устройства также могут переводить тактовую линию в низкий уровень, чтобы сообщить, что они еще не готовы к ответу.

# Последовательный периферийный интерфейс (SPI)

Простейшей формой последовательной связи было бы соединение двух сдвиговых регистров, содержащихся в двух отдельных устройствах. Именно так реализован **последовательный периферийный интерфейс**. Он состоит из регистра сдвига и некоторых выводов ввода/вывода для каждого устройства. Он широко известен как SPI (произносится как "СПАЙ").

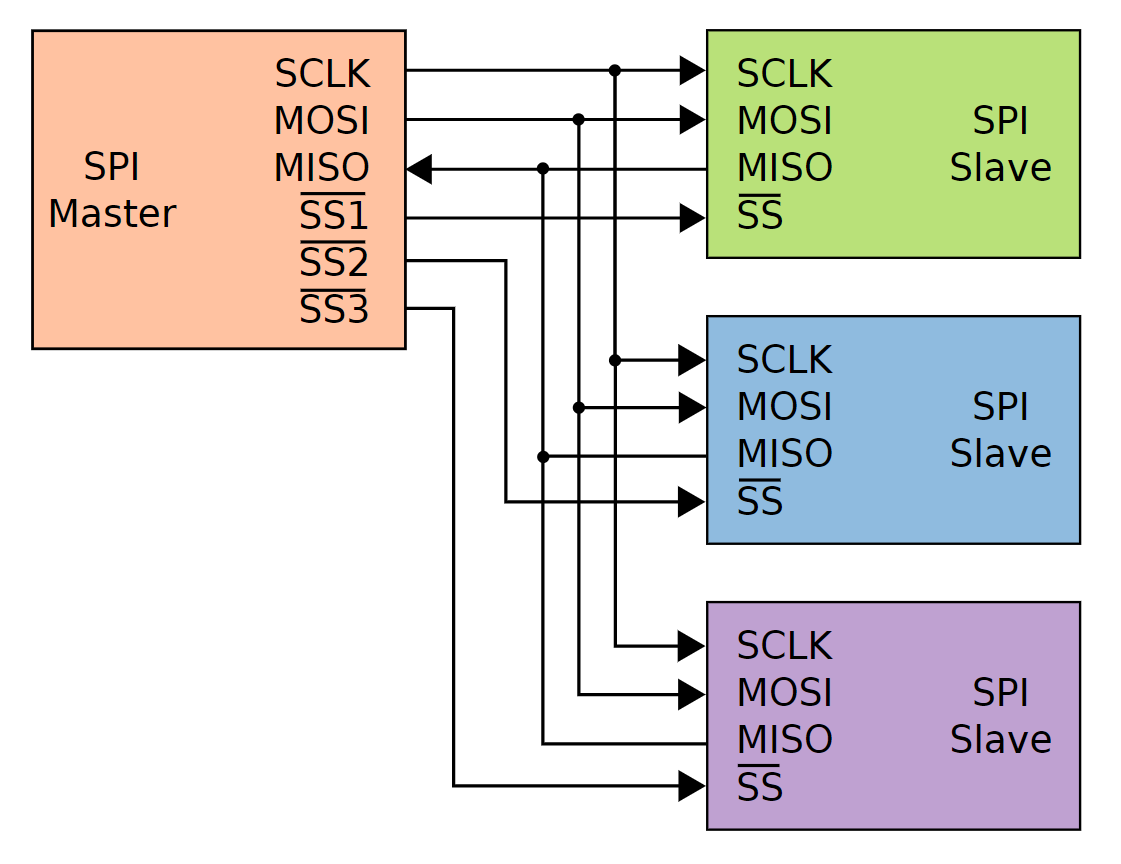
Допустим, мы хотим соединить контроллер с периферийным устройством. Контроллер содержит сдвиговый регистр, и его выход подключен к последовательному входу периферийного устройства. Периферийное устройство на другом конце имеет соответствующий сдвиговый регистр, способный таким же образом отправлять данные обратно на контроллер. Микроконтроллер обычно является ведущим устройством этой сети, поэтому он выдает свой собственный тактовый сигнал для сдвига и некоторые линии выбора для адресации конкретного ведомого устройства. Таким образом, помимо линии заземления, SPI использует 3 сигнальные линии: MISO (Master In Slave Out), MOSI (Master Out Slave In) и линия Clock, которая управляется ведущим устройством. Если используется, ведущий также управляет линиями выбора ведомого устройства (SS).

На следующей схеме показано простое SPI-соединение между контроллером и периферийным устройством.

  
**Простая сеть SPI**(взято из [SparkFun Electronics](https://cdn.sparkfun.com/assets/home_page_posts/3/3/1/7/SPI_Basic_Interface.pdf), предоставлено под [CC BY 2.0](https://creativecommons.org/licenses/by/2.0/))

Создание сети SPI с более чем одним ведомым устройством требует наличия GPIO для выбора ведомого устройства, с которым должен общаться ведущий. Это обычно считается недостатком SPI.

Здесь мы имеем сеть SPI с одним ведущим и тремя ведомыми устройствами:

  
**Сеть SPI с одним ведущим и тремя ведомыми устройствами**   
(взято из [Wikimedia Commons](https://commons.wikimedia.org/w/index.php?curid=1476503), Cburnett, предоставлено по лицензии [CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/))

Чтобы узнать больше о SPI, вы можете прочитать [руководство от SparkFun.](https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi)

# Переопределение названий сигналов SPI

SPI различает элемент, управляющий коммуникацией, и периферийные устройства, которые к нему подключены. С момента своего появления в 1979 году эти элементы были известны как Master и Slaves, но недавнее движение предложило перейти к более описательным названиям, которые не делают ненужных ссылок на рабство. В связи с этим новое предложение заключается в использовании терминов **Controller** вместо master и **Peripheral** вместо slave.

В SparkFun есть [подробное объяснение этого переопределения имен сигналов](https://www.sparkfun.com/spi_signal_names). Это предложение было сделано недавно, а изменения требуют времени, поэтому в большинстве документации по SPI, которую вы найдете, вероятно, будут использоваться термины MISO и MOSI. Например, в руководстве по FE310 до сих пор используются эти термины.

# Асинхронная последовательная связь (UART)

UART означает универсальный асинхронный приемник-передатчик, который является классическим последовательным интерфейсом связи, доступным в старых компьютерах. Вы можете подключиться к последовательному порту вашего компьютера через интерфейс COM-порта с помощью приложения-эмулятора терминала, такого как PuTTY или Tera Term.

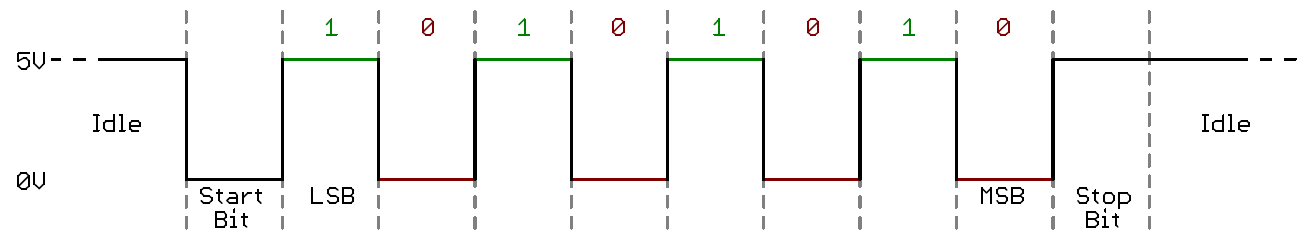
Этот тип последовательной связи известен как *асинхронный,* поскольку не требует наличия тактовой линии между передатчиком и приемником. Это очень удобная функция для сигналов, которые должны использовать один единственный канал (например, инфракрасный пульт дистанционного управления телевизором).

#### Но как синхронизируются передатчик и приемник?

Они следуют протоколу. Вот он для 8-битной передачи:

* Связь осуществляется в одну сторону по линии передачи данных, поэтому есть передатчик и приемник.
* Линия данных начинается в состоянии ожидания, которое равно 1, или высокому уровню.
* Когда передатчик хочет начать передачу, он подает на линию данных низкий уровень в течение определенного времени. Этот 0 известен как **стартовый бит**.
* Эта длительность известна как время 1 бита, которое является обратной величиной согласованной частоты для сдвига битов. Эта частота соответствует скорости передачи данных. В данном типе связи эта скорость передачи данных также известна как скорость в бодах. Двумя очень распространенными скоростями передачи данных являются 9600 бит в секунду и 115 200 бит в секунду, для которых требуется тактовый сигнал передатчика 9,6 кГц и 115,2 кГц соответственно.
* Далее 8 битов данных по очереди передаются по линии, каждый из которых занимает 1 бит. Обычно наименее значимый бит (LSB) идет первым.
* После передачи всех 8 битов данных передатчик подает высокий уровень на линию данных в течение 1 бита. Это называется стоп-битом.

Следующая временная диаграмма показывает передачу байта 0x55 с использованием логики TTL (5 В для высокого уровня и 0 В для низкого):

  
**Временная диаграмма асинхронной 8-битной последовательной передачи**   
(взято из [SparkFun Electronics](https://learn.sparkfun.com/tutorials/serial-communication), предоставлено по [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/))

В протоколе есть 4 основных параметра:

1. Скорость передачи данных. Список вариантов очень длинный: 9600 бод, 19200 бод, 115 200 бод и так далее.
2. Количество битов для передачи. Обычно это 8 бит, но может быть от 5 до 9 бит.
3. Тип четности: Четная четность, нечетная четность или отсутствие четности.
4. Длительность стоп-бита: 1, 1,5 или 2 длительности бита.

Эти параметры фиксированы в системе, поэтому они не должны меняться во время выполнения.

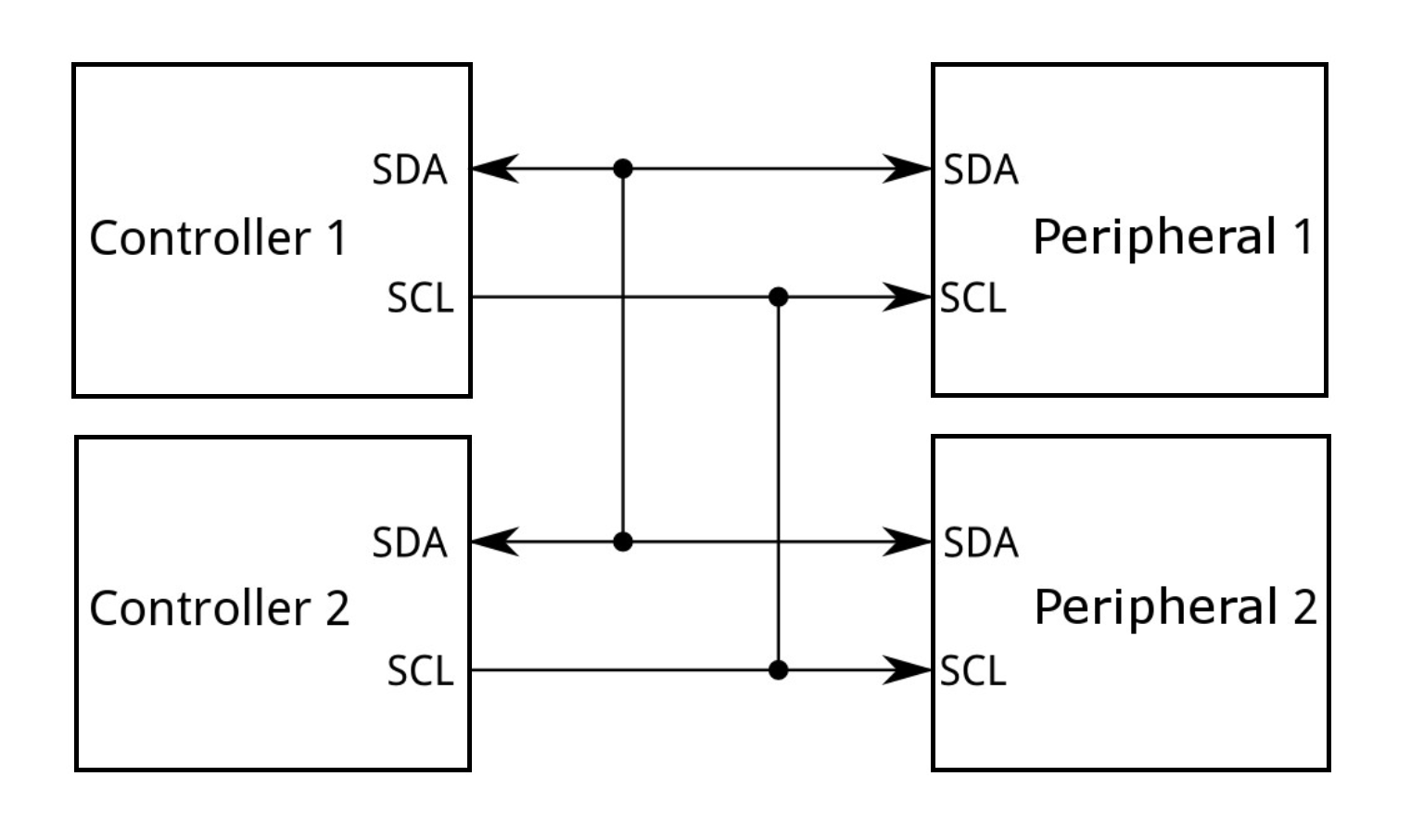
Чтобы узнать больше об асинхронной последовательной связи, вы можете прочитать [учебное пособие от SparkFun](https://learn.sparkfun.com/tutorials/serial-communication).

# Интерфейс межсетевого взаимодействия

Также известен как I2C, I2 C или "I в квадрате C". Это более сложный интерфейс, похожий на SPI, но он использует только 2 линии для всех устройств, подключенных к шине. Эти 2 линии - SDA (линия последовательных данных) и SCL (линия последовательного тактового генератора). Любое устройство может управлять этими двумя линиями, и каждое устройство имеет свой адрес в шине, так что микроконтроллер может взаимодействовать со многими внешними устройствами в одной шине. Именно этот интерфейс используется в системе Qwiic Connect.

Любое устройство в сети может управлять этими двумя линиями. Это электрически безопасно, поскольку ими управляют выходы с открытым затвором. Это означает, что устройства могут подтягивать эти сигналы только вниз, но не вверх. Для обеспечения высокого уровня по умолчанию требуется пара внешних подтягивающих резисторов.

I2C поддерживает множество ведущих/контроллеров и множество ведомых/периферийных устройств, как мы видим на этой диаграмме:

  
**Сеть I2C с двумя контроллерами и двумя периферийными устройствами**   
(взято из [SparkFun Electronics](https://learn.sparkfun.com/tutorials/i2c), предоставлено по [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Адрес каждого устройства может быть жестко задан, может конфигурироваться аппаратно (с помощью перемычек или переключателей), или может конфигурироваться программно.

И снова этот интерфейс включает в себя протокол с более высоким уровнем абстракции, где ведущий управляет тактовой линией, передавая сообщения, которые получают все ведомые, но адресованные одному из них. Адресованное ведомое устройство должно ответить ведущему согласованным количеством передач. Эти передачи также транслируются, и все элементы сети слушают их.

С точки зрения микроконтроллера, типичным способом работы с периферийным устройством I2C является использование следующих операций:

* Отправка данных для записи во внутренние регистры устройства (по адресу). Это известно как операция записи.
* Запрос данных во внутренних регистрах устройства (также по адресу). Это операция чтения.

Операции чтения и записи представляют собой сообщения I2C, состоящие из кадра адреса и одного или нескольких кадров данных. Кадр адреса - это I2C-адрес периферийного устройства, с которым хочет установить связь ведущий, а первый кадр данных обычно представляет собой адрес регистра периферийного устройства для чтения/записи. Для операций записи, как правило, второй кадр данных - это байт для записи.

Сообщения I2C имеют сигнальное поведение условий запуска и остановки (аналогично битам запуска и остановки UART), что позволяет передавать многобайтовые сообщения, состоящие из нескольких сообщений с одним условием запуска в первом сообщении и условием остановки в последнем.

#### Быстрый пример

Предположим, у нас есть ультразвуковой датчик приближения со следующими характеристиками:

* I2C фиксированный адрес: **0x24**.
* Все данные имеют ширину 8 бит.
* Регистр конфигурации единиц измерения по адресу устройства **0x80**. Запишите в этот регистр **0x01** для сантиметров или **0x00** для дюймов.
* Регистр измерения расстояния находится по адресу устройства **0x84**. В этом регистре хранится расстояние, обнаруженное датчиком в любой момент времени, в настроенных единицах измерения.

Чтобы настроить этот датчик на сообщение о расстоянии в сантиметрах, ведущий должен отправить операцию записи, адресованную устройству **0x24**, со значением данных **0x01.**

Чтобы прочитать расстояние, ведущее устройство должно отправить операцию чтения, адресованную устройству **0x24**, с адресом **0x84**, и принять данные в ответном сообщении за запрошенное расстояние.

Как обычно, мы приводим [учебник по I2C от SparkFun](https://learn.sparkfun.com/tutorials/i2c), если вы хотите узнать больше.

# Сравнение SPI, UART и I2C

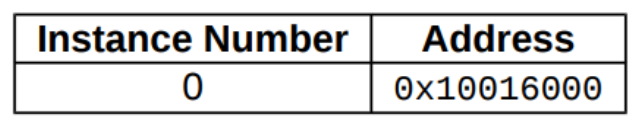
Каждая из этих технологий имеет свои области применения, в которых она господствует, и каждая из них ужасна в некоторых областях. Итак, что же лучше? Как и почти во всем остальном, это зависит от ситуации. Здесь мы приводим некоторые моменты, которые следует учитывать, пытаясь решить, какой последовательный интерфейс использовать для ваших приложений:

* SPI и I2C хороши для высокоскоростной связи, потому что они используют линию синхронизации. Это практически ограничивает скорость передачи данных максимальной скоростью, на которой могут работать цифровые схемы и электрические элементы.
* SPI может работать на очень высоких скоростях, определенно превышающих 1 Мбит/с.
* I2C не такой быстрый, как SPI, но все же довольно быстрый: Стандартная скорость I2C составляет 100 кбит/с.
* UART часто реализуют физические уровни для достижения больших расстояний около 100 м, в то время как I2C и SPI предназначены для связи устройств внутри платы или между двумя платами с помощью коротких кабелей.
* SPI проще всего реализовать внутри чипа, поэтому логично, что он может стоить дешевле в производстве, занимать меньшую площадь в чипе, потреблять меньше энергии и так далее.
* UART требует только одну линию для односторонней передачи данных. Это делает его идеальным для инфракрасной связи с использованием только одного светового сигнала.
* По этой причине UART является самым простым в настройке с точки зрения пользователя.
* Однако UART - это интерфейс "точка-точка", в котором нет ведущих и ведомых, только две конечные точки.
* I2C является наиболее гибким для создания сети устройств, поддерживая более одного ведущего устройства и практически неограниченное количество ведомых.
* I2C является полудуплексным (вы можете отправить или получить только одно сообщение в определенный момент времени), в то время как SPI и UART являются полнодуплексными (вы можете отправлять и получать сообщения одновременно).

# Последовательная связь в микроконтроллере FE310

Микроконтроллер FE310 оснащен 1 устройством I2C, 2 устройствами UART, 2 устройствами SPI. На данном этапе вы можете обратиться к руководству FE310, чтобы проверить описание этих устройств. В этом случае вы найдете достаточно информации для создания собственных приложений, использующих интерфейс UART или SPI.

Теперь для интерфейса I2C есть своя загвоздка: Его глава занимает менее одной страницы. Фактически, все, что мы получаем, это базовый адрес единственного устройства I2C в таблице 98 и примечание, указывающее, что модуль I2C основан на открытой конструкции, с указанием его URL.

  
**Базовый адрес устройства I2C в микроконтроллере FE310**   
(Изображение из руководства пользователя FE310-G002, воспроизведено с разрешения компании SiFive, Inc.)

Это немного усложняет ситуацию, поскольку мы собираемся использовать устройства Qwiic, которые используют интерфейс I2C, поэтому у нас есть 2 варианта:

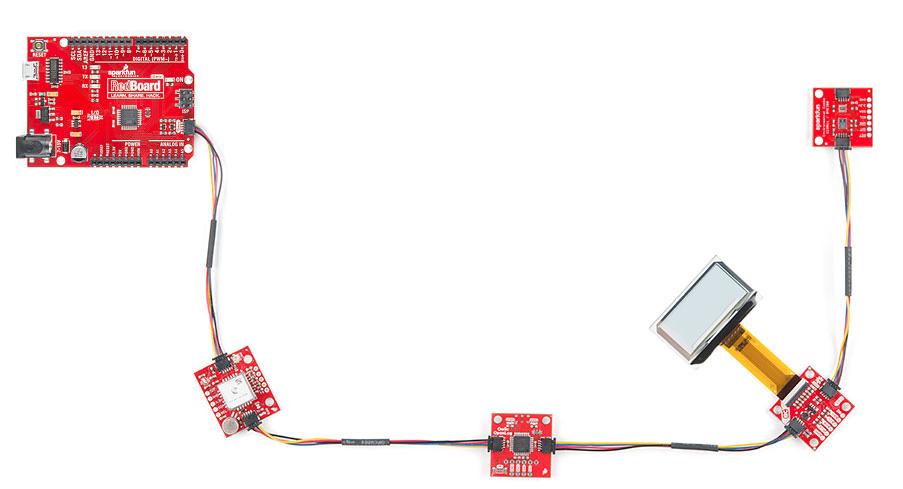
1. Перейдите на страницу проекта открытой конструкции и изучите ее использование там. Однако этот вариант не позволит узнать адреса регистров.
2. Просто используйте библиотеку Freedom Metal Library, которая предоставляет высокоуровневые функции, необходимые для работы с I2C.

На этот раз мы выберем вариант 2. Теперь, если вы хотите углубиться в оригинальный дизайн этого модуля, вы можете посетить [страницу проекта на opencores.org](https://opencores.org/projects/i2c).

# Система Qwiic Connect

Компания SparkFun разработала систему Qwiic Connect, чтобы облегчить пайку при создании сетей I2C между своими микроконтроллерными платами и модулями.

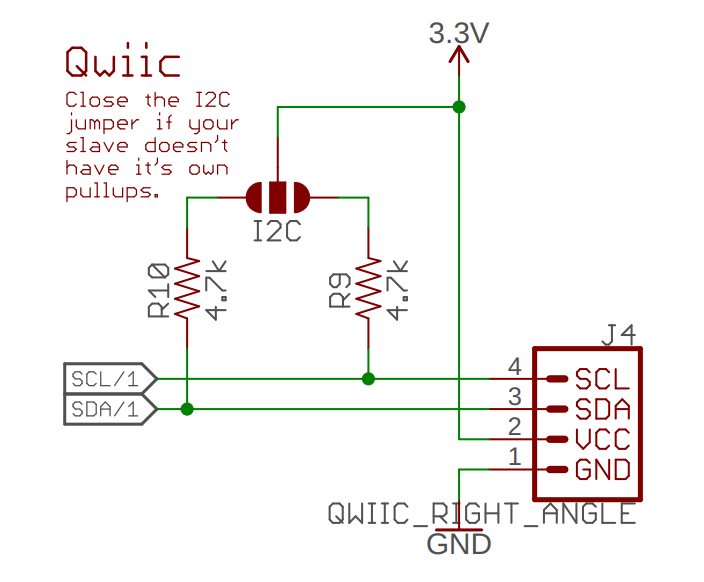
Система Qwiic Connect System не добавляет никакой функциональности к стандарту I2C. Это просто удобный стандарт разъемов, позволяющий легко подключить микроконтроллер к модулям I2C. По этой причине большинство плат Qwiic имеют два разъема, что позволяет создать сеть с гирляндной цепью, как на следующем рисунке.

  
**Сеть Qwiic с одним контроллером и четырьмя устройствами**   
(взято с сайта [SparkFun Electronics](https://www.sparkfun.com/qwiic), предоставлено по [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/))

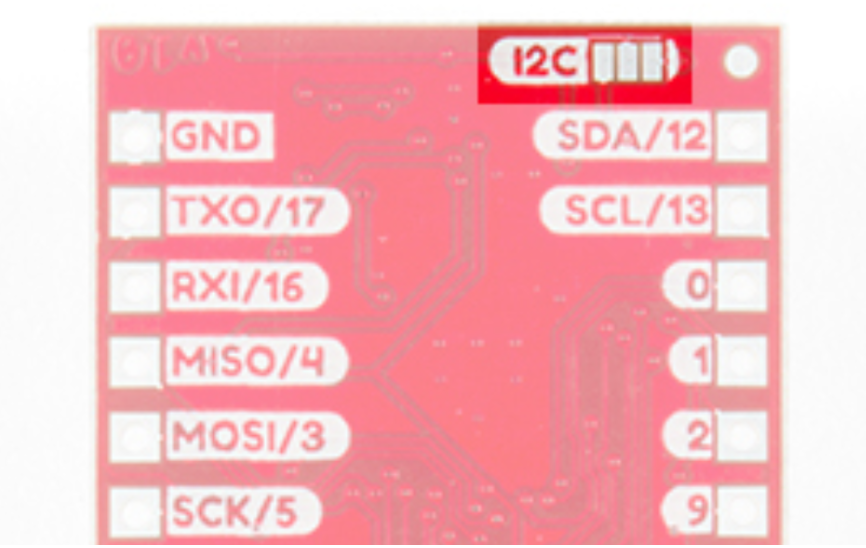
Чтобы узнать больше о системе Qwiic Connect System, посетите сайт [SparkFun](https://www.sparkfun.com/qwiic).

# Разъем Qwiic в Red-V Thing Plus

На схеме Red-V Thing Plus показан разъем Qwiic:

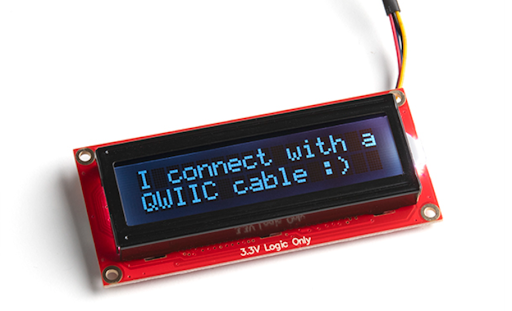
  
**Частичная схема Red-V Thing Plus с изображением разъема Qwiic**(Изображение предоставлено компанией SparkFun Electronics)

Два подтягивающих резистора доступны через 2 припаянные перемычки на печатной плате. Вы можете припаять эти перемычки, если устройства I2C, которые вы собираетесь использовать, не включают подтягивающие резисторы. Однако, если вы используете устройства Qwiic, вам не нужно ничего делать с этими перемычками, так как периферийные платы Qwiic включают подтягивающие резисторы.

  
**Частичный вид снизу Red-V Thing Plus, показывающий подтягивающие перемычки I2C**   
(взято с сайта [SparkFun Electronics](https://learn.sparkfun.com/tutorials/red-v-thing-plus-hookup-guide), предоставлено по [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/))

# Устройства Qwiic: ЖК-экран

Первое устройство, с которым мы будем работать, - это жидкокристаллический экран 16x2 символа. Вы можете получить доступ к его документации на странице продукта.

  
**ЖК-экран SparkFun 16x2 Qwiic**(взято из [SparkFun Electronics](https://www.sparkfun.com/products/16397), предоставлено под [CC BY 2.0](https://creativecommons.org/licenses/by/2.0/))

Этот модуль ЖК-экрана оснащен микроконтроллером, который обеспечивает интерфейс I2C для печати символов, отправляемых на экран с помощью простой операции записи 1 байта. Встроенный микроконтроллер также заботится об управлении экраном с помощью функций, которые можно ожидать от текстового редактора, таких как перемещение курсора вперед после печати символа, перемещение курсора из последней позиции верхней строки в первую позицию нижней строки после печати символа, а также оперативный символ backspace.

Модуль также принимает команды для конфигурации, например, изменение скорости передачи данных, и операции, например, перемещение курсора в определенную позицию. Полное описание этих команд можно найти на странице продукта. Вот команды, которые мы будем использовать:

* + - **Очистить экран**   
      Отправьте 0x7C для входа в режим настроек, затем 0x2D для очистки экрана и перемещения курсора в исходное положение. Эти значения являются символами ASCII для вертикального штриха ('|') и тире ('-') соответственно.
    - **Перемещение курсора в определенную позицию**   
      Отправьте командный символ 0xFE (без привязанного знака, ASCII 254), за которым следует (128 + смещение строки + позиция). Смещение строки равно 0 для строки 1 и 64 для строки 2. Позиции нумеруются от 0 до 15.

# Несколько слов о устройствах Qwiic и скорости

По какой-то необъяснимой причине многие устройства Qwiic от SparkFun работают со скоростью передачи данных 9 600 бод. Это совсем не стандартное значение.

9,600 бод - это стандартная скорость передачи данных для UART, и, конечно, по умолчанию, если вам нужно угадать. Однако весь смысл I2C заключается в том, чтобы воспользоваться преимуществами наличия тактовой линии и работать на той высокой скорости, которую она позволяет. Фактически, в I2C определено несколько скоростных режимов: Стандартный режим, пара быстрых режимов и пара высокоскоростных режимов. Стандартный режим - самый медленный, он работает со скоростью 100 кбит/с.

В последующих приложениях вы увидите, что модуль I2C настроен на работу со скоростью 9 600 бод. Это будет сделано только для соответствия используемым нами модулям Qwiic.

ЖК-экран не отвечает ведущему устройству на стандартной скорости, если только вы не измените скорость передачи данных. Однако, переходя от устройств Qwiic, вы наверняка столкнетесь с устройствами I2C, которые просто не указывают скорость передачи данных, поскольку предполагается, что скорость будет стандартной. Ограничение скорости устройства I2C часто считается излишним, поскольку ведомые устройства I2C могут легко поддерживать стандартный скоростной режим.

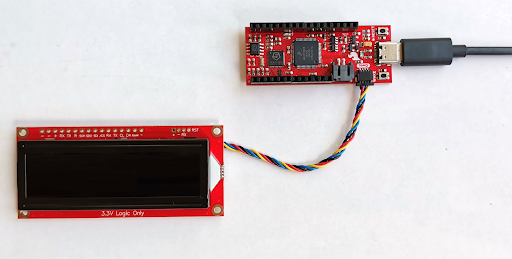
# Применение ЖК-экрана

Далее мы создадим приложение hello world с ЖК-экраном.

Оборудование для этого демонстрационного приложения очень простое, поскольку в нем используется система Qwiic Connect System. Просто подключите один конец кабеля Qwiic к разъему Qwiic устройства Red-V Thing Plus, а другой конец - к модулю ЖК-экрана.

Подключение разъемов Qwiic может быть сложным, если вы не делали этого раньше, поэтому уделите некоторое время наблюдению за разъемами и постарайтесь обращаться с ними осторожно.

В этот раз мы не будем рассматривать схему, потому что все, что вам нужно - это Red-V Thing Plus, ЖК-экран и кабель Qwiic. Здесь представлена фотография аппаратного обеспечения приложения:

  
**Аппаратные соединения для демонстрационного приложения ЖК-экрана**

# Функции I2C в библиотеке Freedom Metal

Если вы просмотрите библиотеку и перейдете в **./freedom-metal/src/**, вы найдете функции, которые мы будем использовать для настройки и использования модуля I2C в файлах **i2c.h** и **i2c.c.**

#### Получение экземпляра I2C

Как обычно, мы получим единственный экземпляр устройства I2C. То есть экземпляр 0:

**struct metal\_i2c \*i2c;  
 i2c = metal\_i2c\_get\_device(0);**

#### Инициализация устройства I2C

Далее мы инициализируем это устройство I2C, указав устройство i2c, скорость передачи данных и режим работы (ведущий или ведомый).

**metal\_i2c\_init(i2c, I2C\_BAUDRATE, I2C\_MASTER);**

В качестве напоминания, для скорости передачи данных мы будем использовать 9 600, но не привыкайте к этому. Для обычных устройств I2C этот аргумент гораздо более вероятен - 100,000, чем 9,600.

#### Операции записи

Для отправки символов и команд на экран мы будем использовать следующую функцию для выполнения операций записи. В качестве аргументов нам нужно указать устройство I2C, адрес I2C периферийного устройства, количество символов для отправки, буфер, в котором находятся эти символы, и нужно ли завершить передачу (послать условие остановки). Следующий фрагмент кода посылает символ 'H' на ЖК-экран:

**unsigned char buf[1];  
 buf[0] = 'H';  
 metal\_i2c\_write(i2c, LCD\_ADDR, 1, buf, 1);**

# Бонусный модуль: Часы реального времени

Чтобы сделать нашу демонстрацию ЖК-дисплея немного более захватывающей, чем просто приветствие мира, мы получим некоторые числовые данные и выведем их на ЖК-экран. Эти данные могут поступать от датчика или из любой части нашей программы.

Чтобы упростить задачу, мы будем использовать модуль Real-Time Clock для получения значения его свободно работающего счетчика. RTC уникален тем, что на него не влияет сигнал кнопки сброса после того, как он начал работать. По этой причине наше приложение просто включит RTC, напечатает сообщение hello-world, а затем выведет относительное время из младшего байта счетчика RTC. Затем приложение застрянет в бесконечном цикле. Идея заключается в том, чтобы печатать другое значение времени каждый раз, когда мы нажимаем кнопку сброса на плате.

Чтобы запустить счетчик RTC, нам нужно просто установить бит 12 в регистре конфигурации RTC, расположенном по адресу **0x10000040**. Значением, которое мы будем использовать, будет младший байт счетчика, находящийся по адресу **0x10000048**. Следующие макросы помогут нам включить счетчик RTC и считать его младший байт, соответственно:

**#define  Red\_V\_RTC\_start()       \*((uint32\_t \*) 0x10000040) |= 1<<12  
     #define  Red\_V\_RTC\_rtccountlo   (\*((uint32\_t \*) 0x10000048))**

Как обычно, если вы хотите узнать больше о модуле RTC, ознакомьтесь с руководством FE310.

# Функции утилиты ЖК-дисплея

Здесь у нас есть 3 функции полезности LCD, которые мы будем использовать:

#### Очистка экрана

Помните, чтобы очистить экран, вы должны отправить символ вертикальной полосы, за которым следует символ тире

**void LCD\_clear(struct metal\_i2c \*i2c){  
 unsigned char buf[3] = "|-";  
 metal\_i2c\_write(i2c, LCD\_ADDR, 2, buf, 1);  
}**

#### Перемещение курсора в положение X-Y

Чтобы установить позицию курсора, нужно послать командный символ 254, за которым следует 128 + x + y\*64. Здесь у нас есть функция, которая делает именно это:

**void LCD\_gotoxy(struct metal\_i2c \*i2c, uint32\_t x, uint32\_t y){  
 unsigned char buf[2];  
 x &= 0x0f; // Ограничение x от 0 до 15  
 y &= 0x01; // Ограничить y от 0 до 1**

**buf[0] = 254;  
 buf[1] = 128 + x + y\*64;  
 metal\_i2c\_write(i2c, LCD\_ADDR, 2, buf, 1);  
}**

#### Печать строки

Здесь представлена классическая функция print string, которая получает в качестве аргумента строку с нулевым окончанием. Обратите внимание, что она использует функцию записи I2C для отправки каждого символа на экран, как и было задумано.

**void LCD\_print\_str(struct metal\_i2c \*i2c, unsigned char \*str){  
 metal\_i2c\_write(i2c, LCD\_ADDR, strlen(str), str, 1);  
}**

Как вы, возможно, знаете, для использования функции **strlen()** нам необходимо **#include<string.h>.**

#### Печать младшего байта счетчика RTC

Поскольку мы планируем вывести некоторое числовое измерение времени из младшего байта счетчика RTC, мы будем использовать широко известную функцию **sprintf()** из **stdio.h**. Вот функциональная часть приложения:

**LCD\_clear(i2c);  
LCD\_print\_str(i2c, "Hey there!");**

**now = Red\_V\_RTC\_rtccountlo & 0xff;  
sprintf(buff, "Time: %d\0",now);**

**LCD\_gotoxy(i2c,3,1);  
LCD\_print\_str(i2c,buff);**

# Код применения ЖК-экрана

Теперь, когда мы увидели большинство частей кода, пришло время собрать его воедино. Вот код для приложения ЖК-экрана и RTC. Если вы следите за развитием событий, создайте новый проект на основе проекта примера hello, как обычно, и скопируйте следующий код, чтобы заменить содержимое файла **hello.c**.

Попробуйте приложение, чтобы убедиться, что оно работает правильно.

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
Red-V Thing Plus LCD\_Demo,   
автор Эдуардо Корпеньо**

**Используя ЖК-экран Qwiic 16x2 символов, выведите сообщение hello-world   
и младший байт счетчика часов реального времени.**

**Дата: 20 июня 2022 года   
Разработано с использованием Freedom Studio v4.18.0.2021-04-1 на Windows 10   
ЛИЦЕНЗИЯ: Этот код выпущен под лицензией MIT (  
http://opensource.org/licenses/MIT).**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include <metal/i2c.h>   
#include <stdint.h>   
#include <stdio.h>   
#include <string.h>**

**// ЖК-экран подключен к шине I2C0   
#define LCD\_ADDR 0x72   
#define I2C\_BAUDRATE 9600   
#define I2C\_MASTER 1**

**// Макрофункции   
#define Red\_V\_RTC\_start() \*((uint32\_t \*) 0x10000040) |= 1<<12  
#define Red\_V\_RTC\_rtccountlo (\*((unint32\_t \*) 0x10000048))**

**// Функции ЖК-дисплея   
void LCD\_clear(struct metal\_i2c \*i2c){   
unsigned char buf[3] = "|-";   
metal\_i2c\_write(i2c, LCD\_ADDR, 2, buf, 1);   
}**

**void LCD\_gotoxy(struct metal\_i2c \*i2c, uint32\_t x, uint32\_t y){   
unsigned char buf[2];   
x &= 0x0f; // Ограничение x от 0 до 15   
y &= 0x01; // Ограничение y от 0 до 1**

**buf[0] = 254; buf[1] = 128 + x + y\*64;   
metal\_i2c\_write(i2c, LCD\_ADDR, 2, buf, 1);   
}**

**void LCD\_print\_str(struct metal\_i2c \*i2c, unsigned char \*str){   
metal\_i2c\_write(i2c, LCD\_ADDR, strlen(str), str, 1);   
}**

**int main(void){ struct metal\_i2c \*i2c;   
uint32\_t now;   
unsigned char buff[16];**

**Red\_V\_RTC\_start()**

**i2c = metal\_i2c\_get\_device(0);   
metal\_i2c\_init(i2c, I2C\_BAUDRATE, I2C\_MASTER);**

**LCD\_clear(i2c); LCD\_print\_str(i2c, "Hey there!");**

**now = Red\_V\_RTC\_rtccountlo & 0xff;**

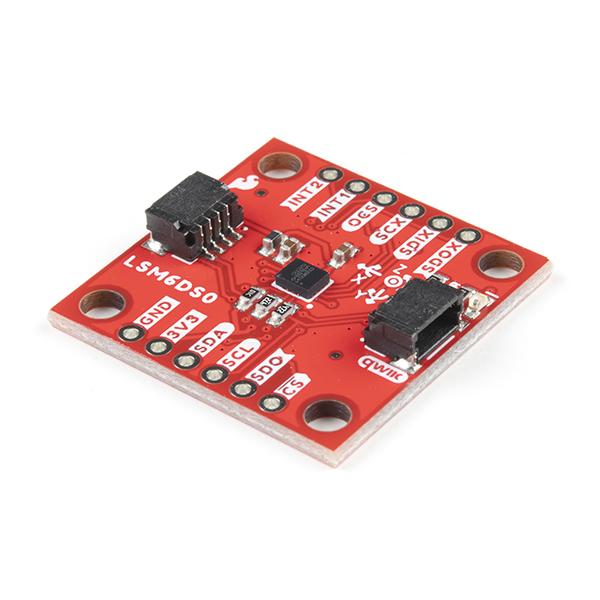
**sprintf(buff, "Time: %d\0",now);**

**LCD\_gotoxy(i2c,3,1); LCD\_print\_str(i2c,buff);**

**while (1); // Сброс для повторной печати   
}**

# Устройство Qwiic: MEMS-акселерометр

Второе устройство Qwiic, которое мы будем использовать, - это разводная плата MEMS 3D акселерометра и гироскопа. Вы можете прочитать все о ней на [странице продукта](https://www.sparkfun.com/products/18020).

  
**Разводная плата SparkFun LSM6DSO Qwiic**(взято из [SparkFun Electronics](https://www.sparkfun.com/products/18020), предоставлено по [CC BY 2.0](https://creativecommons.org/licenses/by/2.0/))

Этот Qwiic-модуль представляет собой разводную плату для МЭМС интегральной схемы акселерометра и гироскопа LSM6DSO производства STMicroelectronics. Плата не имеет большого количества вспомогательного оборудования вокруг микросхемы, поскольку она уже реализует интерфейс I2C, поэтому она идеально подходит для модуля Qwiic.

MEMS означает микроэлектромеханическая система. Это новейшая технология, которая заключается во встраивании миниатюрных механических систем внутрь интегральных схем, например, системы масса-пружина для измерения ускорения.

Этот чип способен измерять ускорение и угловую скорость по 3 осям. Как и большинство устройств I2C, этот чип конфигурируется посредством операций записи регистров в шину I2C. В руководстве (доступно на странице продукта) указаны все эти регистры.

Чувствительность акселерометра по умолчанию обеспечивает диапазон измерений ±2G на ось. При работе с 8-битными измерениями показание 64 означает 1G (земное притяжение), поэтому если держать акселерометр так, чтобы одна из его осей была направлена вниз, то показания в этом направлении будут равны 64. Это может быть +64 или -64, в зависимости от того, куда направлена ось - вверх или вниз.

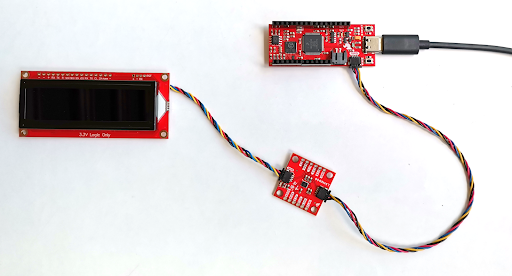
Для демонстрационного приложения, показывающего ускорения по осям X и Y, нас интересуют 3 регистра:

* + - **Регистр управления акселерометром 1**  
      Мы можем включить акселерометр с помощью этого регистра, расположенного по адресу **0x10.** Четыре старших бита этого регистра включают акселерометр и настраивают частоту работы. Мы выберем использование его на частоте 52 Гц, записав **0x30** в этот регистр.
    - **Выходной регистр датчика линейного ускорения по оси X**  
      Это старший байт 16-битного результата. Мы будем использовать старший байт для 8-битного чтения, расположенный по адресу **0x29.**
    - **Выходной регистр датчика линейного ускорения по оси Y**   
      Аналогично описанному выше, но для оси Y. Он расположен по адресу **0x2B.**

# Аппаратное обеспечение приложения акселерометра

Это приложение будет использовать как ЖК-экран, так и акселерометр для демонстрации гирляндной сети I2C.

Для этого приложения мы также не увидим принципиальной схемы. Вместо этого мы имеем фотографию аппаратного обеспечения приложения:

  
**Аппаратные соединения для демонстрационного приложения акселерометра**

# Код приложения акселерометра

Акселерометр - это очень простое дополнение к нашему предыдущему приложению, поэтому здесь мы приводим код приложения для акселерометра. Если вы следите за развитием событий, создайте новый проект на основе проекта-примера hello, как обычно, и скопируйте следующий код, чтобы заменить содержимое файла **hello.c**.

Попробуйте приложение, чтобы убедиться, что оно работает правильно.

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
Red-V Thing Plus Accelerometer\_Demo,   
автор Eduardo Corpeño**

**Используя ЖК-экран Qwiic 16x2 символа и акселерометр LSM6DSO,   
постоянно печатайте ускорение по осям X и Y.**

**Дата: 20 июня 2022 года   
Разработано с использованием Freedom Studio v4.18.0.2021-04-1 на Windows 10   
ЛИЦЕНЗИЯ: Этот код выпущен под лицензией MIT (  
http://opensource.org/licenses/MIT).**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include <metal/i2c.h>   
#include <stdint.h>   
#include <stdio.h>   
#include <string.h>**

**// ЖК-экран подключен к шине I2C0   
#define LCD\_ADDR 0x72   
#define ACCEL\_ADDR 0x6B   
#define I2C\_BAUDRATE 9600   
#define I2C\_MASTER 1**

**// Функции ЖК-дисплея   
void LCD\_clear(struct metal\_i2c \*i2c){   
unsigned char buf[3] = "|-";   
metal\_i2c\_write(i2c, LCD\_ADDR, 2, buf, 1);   
}**

**void LCD\_gotoxy(struct metal\_i2c \*i2c, uint32\_t x, uint32\_t y){   
unsigned char buf[2];   
x = 0x0f; // Ограничение x от 0 до 15   
y = 0x01; // Ограничение y от 0 до 1**

**buf[0] = 254; buf[1] = 128 + x + y\*64;   
metal\_i2c\_write(i2c, LCD\_ADDR, 2, buf, 1);   
}**

**void LCD\_print\_str(struct metal\_i2c \*i2c, unsigned char \*str){   
metal\_i2c\_write(i2c, LCD\_ADDR, strlen(str), str, 1);   
}**

**// Быстрая и грязная функция задержки для экспериментального использования.  
void delay(int t){   
volatile uint32\_t x, y;   
x=t;   
while(x--){   
y=x;   
while(y--);   
}  
 }**

**int main(void){ struct metal\_i2c \*i2c;  
 unsigned char buff[16];   
unsigned char reg[2], accel[1];**

**i2c = metal\_i2c\_get\_device(0);   
metal\_i2c\_init(i2c, I2C\_BAUDRATE, I2C\_MASTER);**

**reg[0]=0x10; // Адрес 1 регистра управления акселерометром reg[1]=0x30; // Включение акселерометра на частоте 52 Гц   
metal\_i2c\_write(i2c, ACCEL\_ADDR, 2, reg, 1);**

**LCD\_clear(i2c);**

**while (1){ reg[0]=0x29; // Читаем ускорение X   
metal\_i2c\_write(i2c, ACCEL\_ADDR, 1, reg, 1);   
metal\_i2c\_read(i2c, ACCEL\_ADDR, 1, accel, 1);**

**sprintf(buff, "X: %d \0", (signed char) (accel[0])); LCD\_gotoxy(i2c,3,0); LCD\_print\_str(i2c,buff);**

**reg[0]=0x2B; // Считывание ускорения Y   
metal\_i2c\_write(i2c, ACCEL\_ADDR, 1, reg, 1);   
metal\_i2c\_read(i2c, ACCEL\_ADDR, 1, accel, 1);**

**sprintf(buff, "Y: %d \0", (signed char) (accel[0])); LCD\_gotoxy(i2c,3,1); LCD\_print\_str(i2c,buff);**

**delay(1000);   
}  
 }**

# Итоги главы

Отлично! Теперь вы знаете, как использовать устройства I2C и модули Qwiic.

Вот краткий обзор того, что мы узнали в этой главе:

* Мы рассмотрели, что такое последовательная связь и чем она лучше GPIO.
* Мы обсудили 3 последовательных интерфейса связи, обычно включаемых в микроконтроллеры.
* Мы изучили библиотеку Freedom Metal Library, чтобы узнать, как использовать устройство I2C.
* Мы представили систему Qwiic Connect System.
* Мы создали приложение для управления ЖК-экраном.
* Мы создали приложение акселерометра.

Не стесняйтесь возвращаться к этому материалу, если в будущем вам понадобится подтянуть знания.