Глава 2. Микроконтроллер

# Представление

## Обзор главы (Видеоролик)

[Эдуардо Карреньо] Теперь пришло время поговорить о микроконтроллерах и их библиотеке. В этой главе мы поговорим о написании кода. Мы начнем с различия между блоками микроконтроллеров и системой на кристалле (СнК). Потом мы поговорим о микроконтроллерах RISC-V. А после, мы обратимся к микроконтроллеру FE310 созданному SiFive, который является одним из Red-V Thing Plus. Затем мы познакомимся с библиотекой Freedom Metal которая является фирменной библиотекой SiFive's для их устройств. Мы создадим простой проект в Blinky. И в заключении мы изучим библиотеки для Freedom Studio.

## Учебные задачи

К концу этой главы, Вы будете способны:

* Находить любую информацию, которая вам необходима, в технической документации к плате или микроконтроллеру.
* Описывать разницу между микроконтроллерами и СнК.
* Иметь представление о ключевых функциях и модулях микроконтроллера FE310.
* Описывать возможности RV32IMAC ISA.
* Работать с библиотекой Freedom Metal в Freedom Studio.

# Микроконтроллер

## Важная документация

В остальной части курса Вы будете ознакомлены с определенными разделами следующих документов. Просьба загрузить их и сделать доступными для ознакомления:

***Руководство по эксплуатации микроконтроллера FE310 G-002*** К этому документу мы будем обращаться больше всего. Он содержит в себе все подробности о внутренней работе микросхемы микроконтроллера.

***Спецификация микроконтроллера FE310 G-002*** Это классическая спецификация чипа. Она содержит информацию об корпусе, выводах микросхемы и электронные характеристики. Временами мы будем ссылаться на нее.

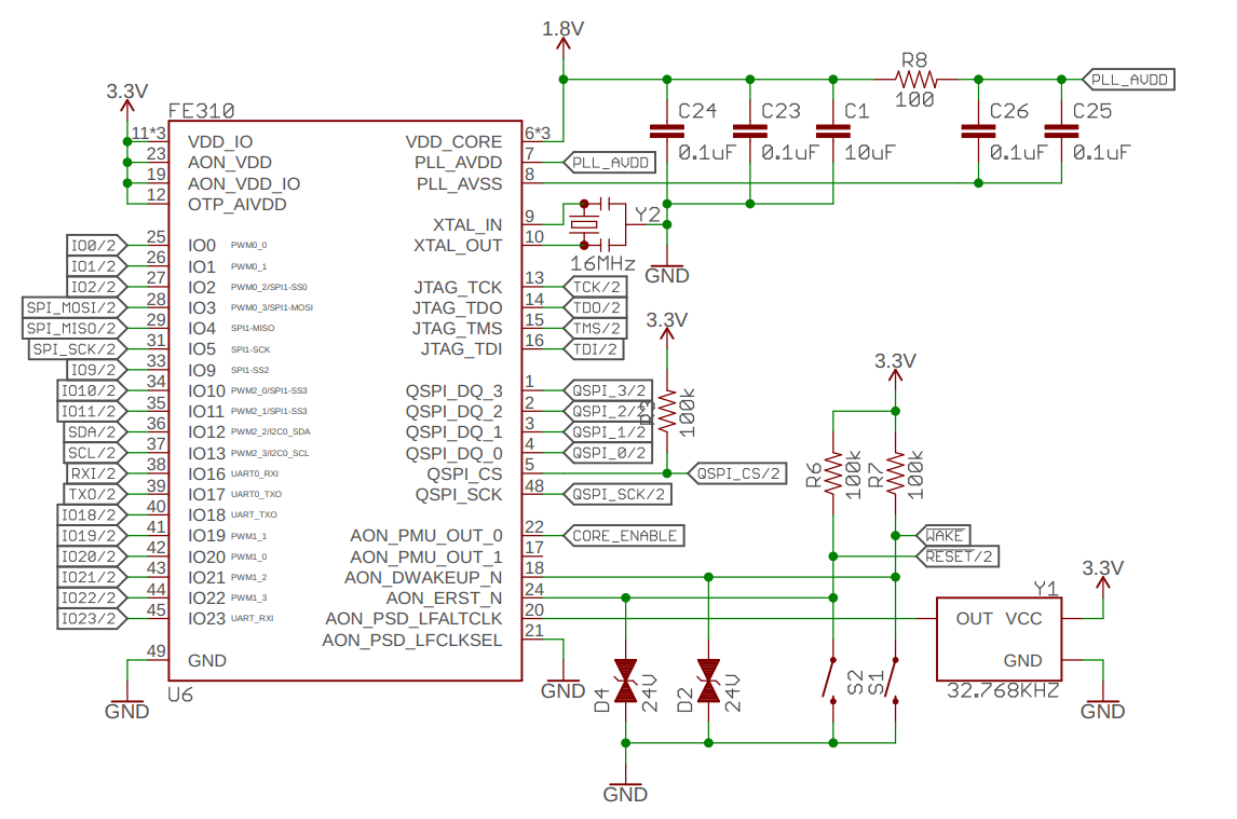
***Принципиальная схема Red-V Thing Plus***Она будет полезна нам, когда нам будет необходимо решить, какие контакты мы будем использовать в конкретном случае, так как мы будем способны узнать какие контакты в микросхеме подключены к каким контактным площадкам или разъемам.

Возможно, вам захочется быстро просмотреть эти документы, но не тратьте на них слишком много времени. На протяжении всего курса вы будете получать ссылки на соответствующие разделы. И не волнуйтесь, если у вас не очень хороший опыт работы с оборудованием. Мы будем действовать медленно. Если вы почувствуете, что что-то нуждается в уточнении, пожалуйста, свяжитесь с нами на форуме курса.

Наконец, корпорации SiFive и SparkFun Electronics любезно разрешили нам использовать их иллюстрации в дальнейшем материале. Иллюстрации взяты из упомянутых 3 документов и из руководств компании SparkFun.

## Red-V Thing Plus

Начнем с рассказа о плате Red-V Thing Plus. Для начала взглянем на принципиальную схему:

  
**Частичная принципиальная схема Red-V Thing Plus, SparkFun Electronics.**  
**Обратите внимание на чип FE310**  
(Изображение предоставлено SparkFun Electronics)

Чип микроконтроллера на самом деле является СнК Freedom E310.

Теперь обсудим некоторые вещи о СнК.

СнК означает ***Система на кристалле***. Это тип интегральных схем, которые различаются по сложности, но во всех них есть некоторая система внутри чипа.   
Эта система лежит в диапазоне от 8-битного ЦП с набором операционных модулей, соединенных простой шиной, до высокопроизводительных ядер с кэшем, оперативной памятью, и даже с графическим ускорителем, все на одном ядре.

## Микроконтроллеры(MCUs)

Первый упомянутый тип СнК (с бюджетными процессорами), часто называемые микроконтроллерами(MCUs). Обычно они имеют скромный центральный процессор с внутренней шиной для реализации схемы с отображением в память для доступа к ряду оперативных модулей ввода-вывода, таких как ввод/вывод общего назначения (GPIO), аналого-цифровые преобразователи (АЦП), цифроаналоговые преобразователи (ЦАП), таймеры (для ШИМ), последовательный порты (USART, I2C, SPI, USB), если упоминать лишь некоторые из них.

Схема отображения памяти для GPIO означает, что модули ввода-вывода (АЦП, SPI и т.д.) имеют регистры состояния и управления, которые имеют адрес памяти в адресном пространстве микроконтроллера. Технически, они подключены к архитектуре внутренней шины. Взаимодействие с устройствами ввода-вывода осуществляется через их регистры состояния и управления, которые действуют как почтовые ящики для обмена сообщениями между центральным процессором и устройствами ввода-вывода. Например, чтобы начать аналого-цифровое преобразование, вы можете записать логическую единицу в определенный бит в регистр состояния и управления АЦП, который имеет свой собственный адрес памяти. Аналогично, чтобы считать последний байт, который был получен последовательным портом, нужно считать регистр входных данных для этого последовательного порта, который также имеет фиксированный адрес памяти. Обычно в этой архитектуре шины также имеется несколько устройств памяти: флэш-память для хранения программ и констант и память SRAM для переменных.

Цель состоит в том, чтобы создать простые приложения для цифрового управления. Эти приложения могут быть простыми (без операционной системы), или они могут использовать скромную операционную систему реального времени (RTOS), которая обычно представляет собой просто многопоточный планировщик с драйверами ввода-вывода.

Существует множество хорошо известных семейств MCU, и вы можете узнать некоторые из следующих:

Семейство ATmega от Atmel. Их можно узнать по Arduino UNO.

Семейство s08 от NXP Semiconductors. Популярны в бытовой технике.

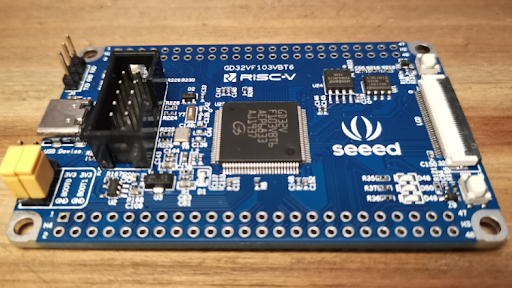
Семейство Kinetis от NXP Semiconductors. В Red-V Thing Plus есть один из них для реализации интерфейса USB J-Link.

Семейство STM32 от STMicroelectronics. Оснащены процессорами ARM-Cortex M.

Семейство MSP430 от Texas Instruments. Платы TI LaunchPad используют именно этот микроконтроллер.

Все семейства ARM Cortex M0+, M3 и M4, создаются всеми вышеперечисленными.

Отдельного упоминания заслуживает семейство GD32V от GigaDevice. Этот микроконтроллер представлен на одной из первых микроконтроллерных плат RISC-V, плате разработки GD32 RISC-V от SeeedStudio. Как и Red-V Thing Plus, эта плата имеет очень мало поддерживающих аппаратных элементов вокруг микроконтроллера, так что вы можете работать непосредственно с чипом.

  
**Плата разработки GD32 от SeedStudio**

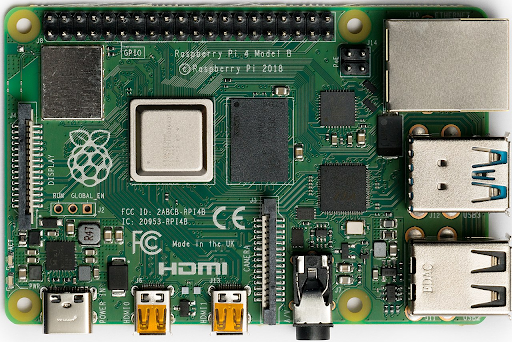
## Эта плата пользуется большой популярностью, потому что она построена на базе микроконтроллера GD32VF103, прямого конкурента микроконтроллеру STM32F103 (который оснащен процессором ARM Cortex M-3). Конкуренция, мягко говоря, интересная, поскольку GigaDevice решила использовать ту же номенклатуру, что и STMicro, что означает, что этот чип имеет те же компоненты, что и STMicro. Одно большое отличие заключается в том, что микроконтроллер GD32VF103 от GigaDevice использует ядро RISC-V, точнее, RV32IMAC. Это тот же процессор, который мы будем использовать в Red-V Thing Plus. Если у вас есть опыт работы с микроконтроллерами семейства STM32, эта плата может стать плавным переходом от ARM к RISC-V.

## СнК

На другом конце спектра у нас есть системы высокого класса с кэш-памятью, основной памятью и графическими процессорами, все в одном чипе. Их часто называют СнК, и среди них тоже есть звезды:

* Broadcom BCM 2711, который вы можете найти в Raspberry Pi 4.
* Texas Instruments OMAP 4430, который является ядром известного гаджета Google Glass.
* Семейство SnapDragon от Qualcomm. Очень популярны в устройствах Android.
* Семейство Exynos от Samsung. Неудивительно, что эти СнК лежат в основе смартфонов и планшетов Samsung.

Цель СнК - обеспечить возможность создания пользовательских приложений, работающих под управлением операционной системы (например, Android, Windows или Linux), с использованием как можно меньшего количества чипов. Очевидно, пространство в мобильном устройстве имеет решающее значение, поэтому удобнее иметь все в одном чипе, чем кучу чипов, разбросанных по печатной плате. “Все” означает то, что вы могли бы найти на материнской плате ПК. Вот почему на платах СнК, таких как Raspberry Pi, так мало микросхем:

  
**Raspberry Pi 4B**(Взято с Викисклада, предоставлено в соответствии с CC BY-SA 4.0)

# Основные характеристики СнК FE310

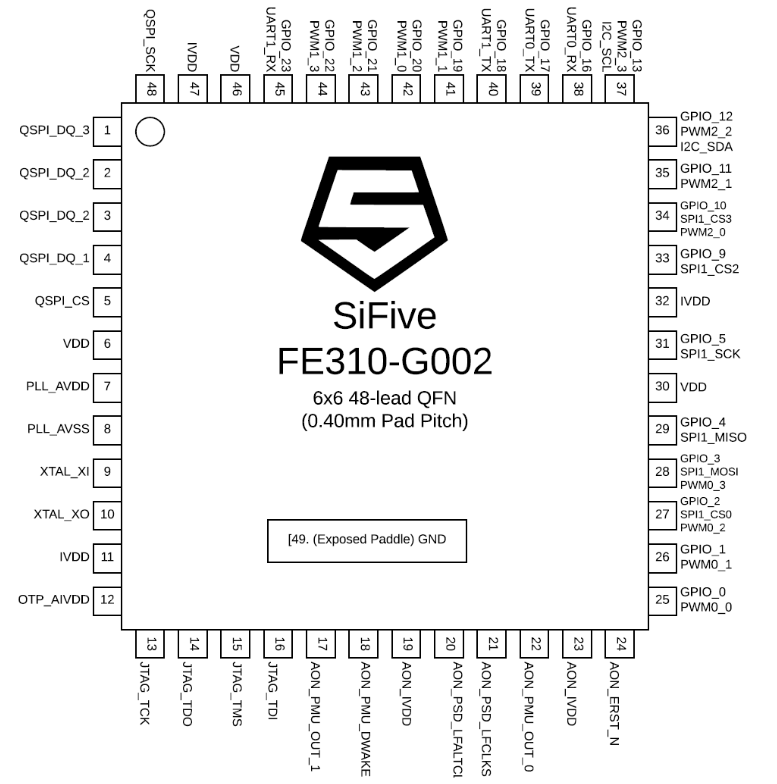
В Red-V Thing Plus используется СнК Freedom E310-G002. Вот некоторые из его особенностей:

* Комплекс ядер SiFive E31 с частотой до 320 МГц
* Гибкие параметры синхронизации, которые включают внутреннюю ФАПЧ, кольцевой осциллятор со свободным ходом и внешний кристалл с частотой 16 МГц
* Процессор RISC-V RV32IMAC
* Программная память OTP объемом 8 Кбайт
* Кэш команд объемом 16 КБ
* 16 КБ данных SRAM
* 3 независимых ШИМ-контроллера
* Интерфейсы JTAG, SPI I2C и UART
* Флэш-интерфейс QSPI
* 8 КБ ПЗУ с маской
* Требуются источники питания напряжением 1,8 В и 3,3 В

Как вы можете видеть, это больше похоже на MCU, чем на СнК, как описано выше. Однако в этом чипе реализован ряд улучшений производительности, что ставит его на более высокий уровень в категории микроконтроллеров. Эти усовершенствования включают кэш команд, конвейер выполнения, предсказатель ветвлений, интерфейс QSPI для внешней флэш-памяти и тесно интегрированную память данных.

Эти улучшения не получили широкого распространения в популярных микроконтроллерах, и в этом курсе мы, безусловно, используя этот мощный чип в качестве традиционного микроконтроллера, не раскроем его потенциал полностью.

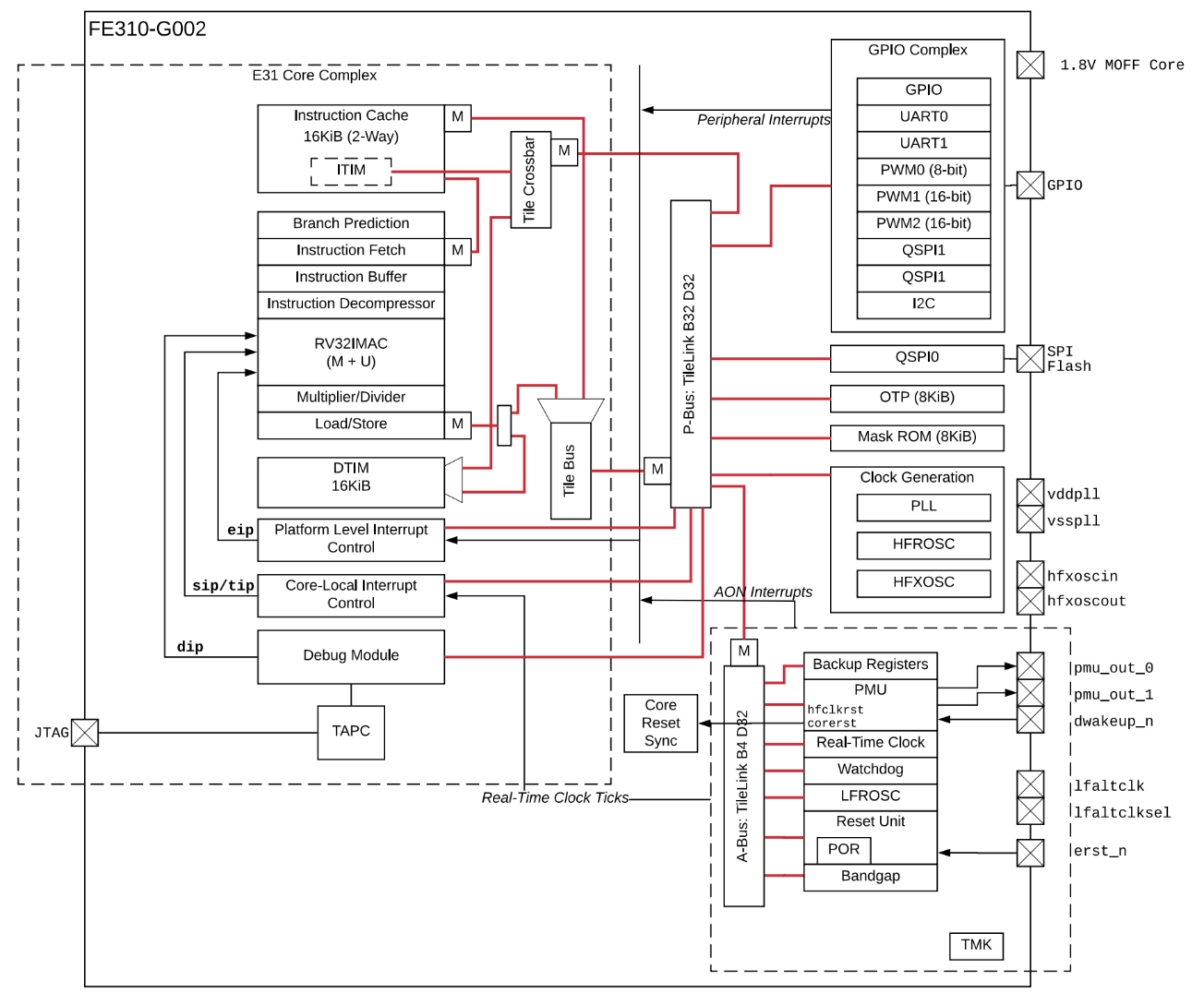
Чип имеет относительно небольшое количество контактов, и он доступен только в корпусе QFN48:

  
**Комплектация FE 310-G002 и распиновка**   
(Изображение из спецификации FE 310-G002, воспроизведенное с разрешения SiFive)

## Чтобы узнать больше о функциях микроконтроллера, не стесняйтесь обращаться к техническому описанию FE301-G002.

## Принципиальная схема

На рисунке представлена принципиальная схема FE 310. Бегло ознакомьтесь с его компонентами и организацией, но не тратьте слишком много времени на то, чтобы разобраться в каждой детали или аббревиатуре. Далее мы рассмотрим несколько из этих модулей.

  
**Принципиальная схема верхнего уровня FE310**(Изображение из руководства пользователя FE 310-G002, воспроизведенное с разрешения SiFive, Inc.)

Пунктирный прямоугольник слева обозначает ядро E31, которое представляет собой усовершенствованный дизайн процессора SiFive. Наиболее важной частью ядра является центральный процессор, который представляет собой процессор RISC-V RV32IMAC. Подробнее об этом вы узнаете позже.

Верхняя правая группа блоков - это комплекс GPIO. GPIO расшифровывается как ввод-вывод общего назначения и относится к цифровым выводам ввода-вывода в микросхеме, которые доступны пользователю. Один из этих блоков называется GPIO, поскольку он относится к возможностям цифрового ввода/вывода этих выводов. Однако эти контакты могут быть перенаправлены для управления отдельными операционными модулями, доступными программисту в режиме отображения памяти (SiFive называет это “Аппаратно-управляемые функции ввода-вывода”).

Наконец, в правом нижнем углу пунктирным прямоугольником обозначены модули в домене Always-On, который включает счетчик реального времени, сторожевой таймер, резервные регистры, низкочастотную синхронизацию, а также схемы сброса и управления питанием. Как вы можете понять, домен Always-On включает в себя модули, которые должны быть способны реагировать всегда, даже когда чип находится в спящем режиме.

Чтобы узнать больше о блоках в этой схеме, не стесняйтесь обращаться к руководству FE301-G002.

## Комплекс GPIO

Следующие модули являются блоками, содержащимися в комплексе GPIO:

*GPIO*

Это контроллер GPIO, который позволяет вам устанавливать рабочие параметры выводов ввода-вывода, например, устанавливать вывод в качестве входа или выхода, разрешать прерывания для определенных выводов, включать внутренние подтягивающие резисторы для входных выводов или обеспечивать высокую мощность возбуждения по току для выходных выводов.

*UART0 и UART1*

Это традиционные асинхронные последовательные порты. UART расшифровывается как универсальный асинхронный приемник-передатчик, и это классический интерфейс последовательной связи, доступный в устаревших компьютерах. Вы можете подключиться к последовательному порту вашего компьютера через интерфейс COM-порта с помощью приложения-эмулятора терминала, такого как PuTTY или Tera Term.

*PWM 0, PWM1, и PWM2*

ШИМ расшифровывается как широтно-импульсная модуляция, популярный метод управления мощностью электрической нагрузки путем изменения ***рабочего цикла*** сигнала, который управляет этой нагрузкой. В микроконтроллерах, ШИМ - это классическое применение модулей ***таймера***, которые имеют приложения, связанные со временем ввода и вывода, такие как ***захват входных данных*** и ***сравнение выходных данных***. Многие модули таймера в микроконтроллерах имеют специальную функцию ШИМ-генератора, которая позволяет программисту легко генерировать периодические цифровые сигналы. SiFive пропустил функции захвата входных данных и сравнения выходных данных и сразу перешел к реализации ШИМ-генератора. Это имеет смысл: ШИМ на сегодняшний день является наиболее широко используемым приложением модулей таймера. Это то, что позволяет вам так легко изменять яркость экрана вашего смартфона.

*SPI1 and SPI2*

(На диаграмме неправильно указаны SPI0 и SPI1, но это должны были быть SPI1 и SPI2.) Это два контроллера последовательного периферийного интерфейса, доступных пользователю. SPI - это простейший последовательный коммуникационный интерфейс, который состоит из сдвигового регистра для передачи и сдвигового регистра для приема. Устройство на другом конце имеет два соответствующих регистра сдвига. Микроконтроллер обычно является ведущим в этой сети, поэтому он выводит тактовый сигнал для переключения и несколько линий выбора, чтобы выбрать, к какому ведомому устройству обращаться. Таким образом, в дополнение к линии GND, SPI использует 3 сигнальные линии: MISO (Вход ведомого), MOSI (Выход ведомого) и линию синхронизации, которая управляется ведущим устройством. Ведущий также управляет подчиненными линиями выбора.

*I2C*

Это интерфейс между интегральными схемами, также известный как I2C или I2CB. Это более удобный интерфейс, похожий на SPI, но он использует только 2 линии для всех устройств, подключенных по шине. Эти 2 линии - SDA (линия передачи данных) и SCL (линия синхронизации). Любое устройство может управлять этими двумя линиями, и каждое устройство имеет адрес в шине, поэтому микроконтроллер может взаимодействовать со многими внешними устройствами на одной шине. Это интерфейс, используемый системой Qwiic Connect.

*QSPI0*

Это контроллер внешней флэш-памяти. Эта память предназначена для хранения ваших программ в тех случаях, когда внутренней энергонезависимой памяти недостаточно. SparkFun решил использовать внешнюю флэш-память, чтобы вы могли создавать большие программы с помощью Red-V Thing Plus. Поскольку внешняя флэш-память используется как часть операции выборки ядром процессора для выполнения команд, программист не имеет никакого контроля над этим модулем, кроме выбора, использовать его или нет. Это делается с помощью контактов выбора режима в микросхеме (MSEL).

## Ядро RISC-V

Архитектура набора команд, реализованная нашим микроконтроллером - это RV32IMAC. Давайте обсудим, что это значит.

Являясь пятым поколением исследовательского проекта, начатого в 1980 году, RISC-V - это опытная архитектура, разработанная для достижения успеха там, где другие терпели неудачу в прошлом, учась на своих ошибках. По этой причине RISC-V был разработан как модульный ISA вместо традиционных инкрементных ISA.Это означает, что реализация RISC-V состоит из обязательного базового ISA и ряда расширений ISA, так что пользовательские процессоры могут быть адаптированы к потребностям приложения.

Соглашение об именовании пользовательских ISA состоит из букв RV (для RISC-V), за которыми следует разрядность и идентификатор модели.

Имея это в виду, RV32IMAC означает:

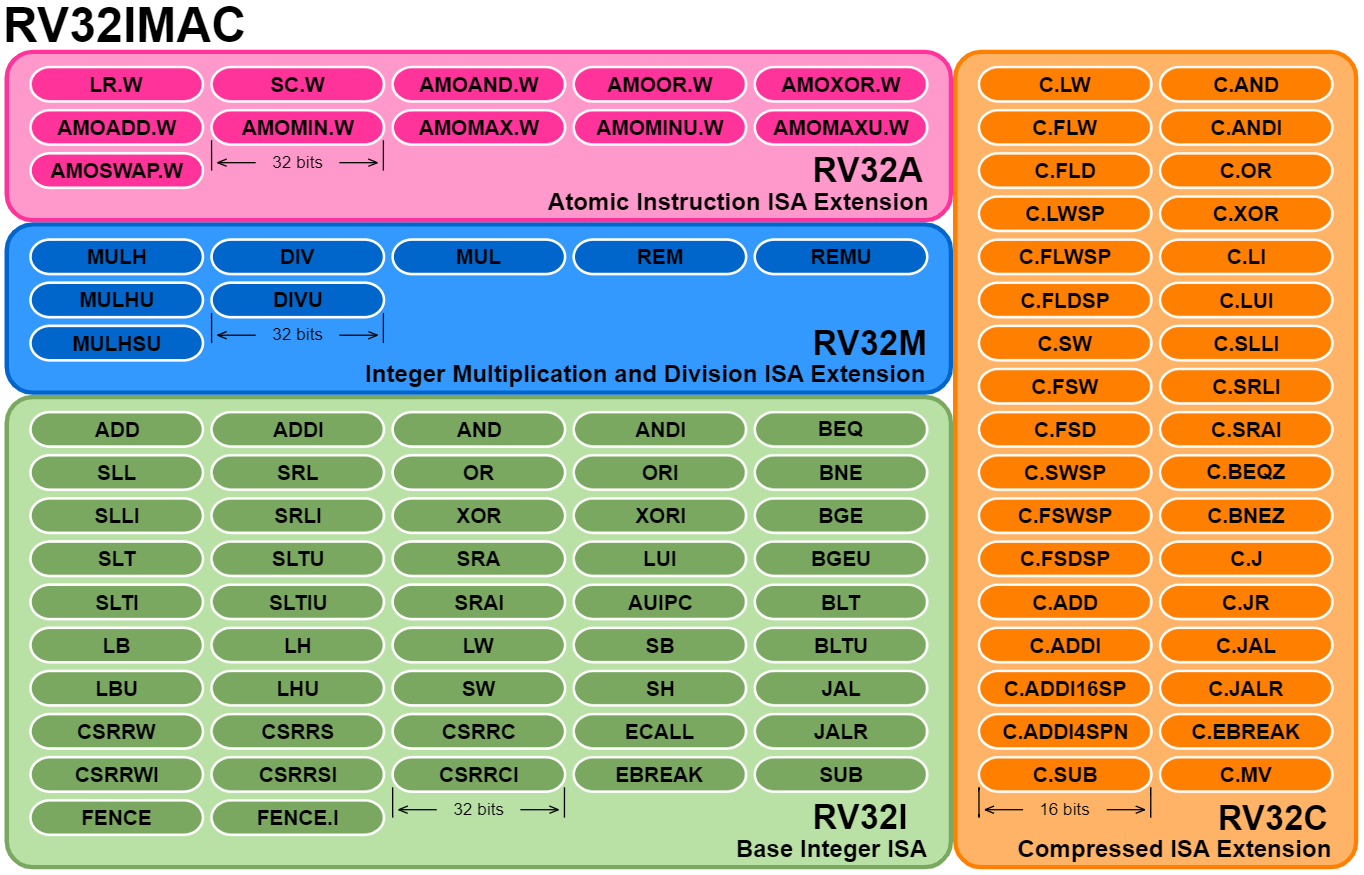
**RV32I**: 32-разрядный процессор с базовым ISA. Сюда входят абсолютно необходимые базовые операции.

**M**: Расширение для умножения и деления целых чисел.

**A**: Расширение атомарных команд.

**C**: Расширение команд сжатия. Это расширение является разумным дополнением к RISC-V ISA, поскольку оно обеспечивает альтернативную 16-битную кодировку для специального подмножества существующих команд.

Базовое ISA состоит всего из 47 команд, и каждое расширение добавляет несколько инструкций. Ниже представлено графическое представление набора команд RV32IMAC:

  
**Набор команд для RV32IMAC демонстрирует модульную (а не инкрементную) природу RISC-V.**  
**Обязательный базовый ISA сочетается с набором расширений.**

Компиляторы информируются о расширениях, включенных в целевой процессор, чтобы они генерировали наилучший возможный код. Если код содержит команды из расширений, отсутствующие в физическом процессоре, аппаратное обеспечение перехватывает и выполняет программные функции из стандартной библиотеки.

Итак, в следующий раз, когда вы увидите код ассемблера в окне дизассемблера в Freedom Studio, постарайтесь обратить внимание, что он содержит инструкции по сборке RISC-V, которые могут быть перечислены, а могут и не быть перечислены на этой диаграмме. Возможно, вы даже захотите проверить выполнение этих команд, чтобы лучше понять, что происходит.

Чтобы узнать больше о RISC-V ISA, прочтите книгу Дэвида Паттерсона и Эндрю Уотермана под названием "RISC-V Reader". Английская версия стоит очень недорого, а переводы на испанский, португальский, китайский и корейский языки можно скачать бесплатно.

## Библиотека Freedom Metal

Все детали, которые мы до сих пор рассматривали в этом разделе, относятся к аппаратному обеспечению и низкоуровневым элементам, которыми вы будете управлять и взаимодействовать с помощью библиотеки встроенного ПО. Думайте об этой библиотеке как об API.

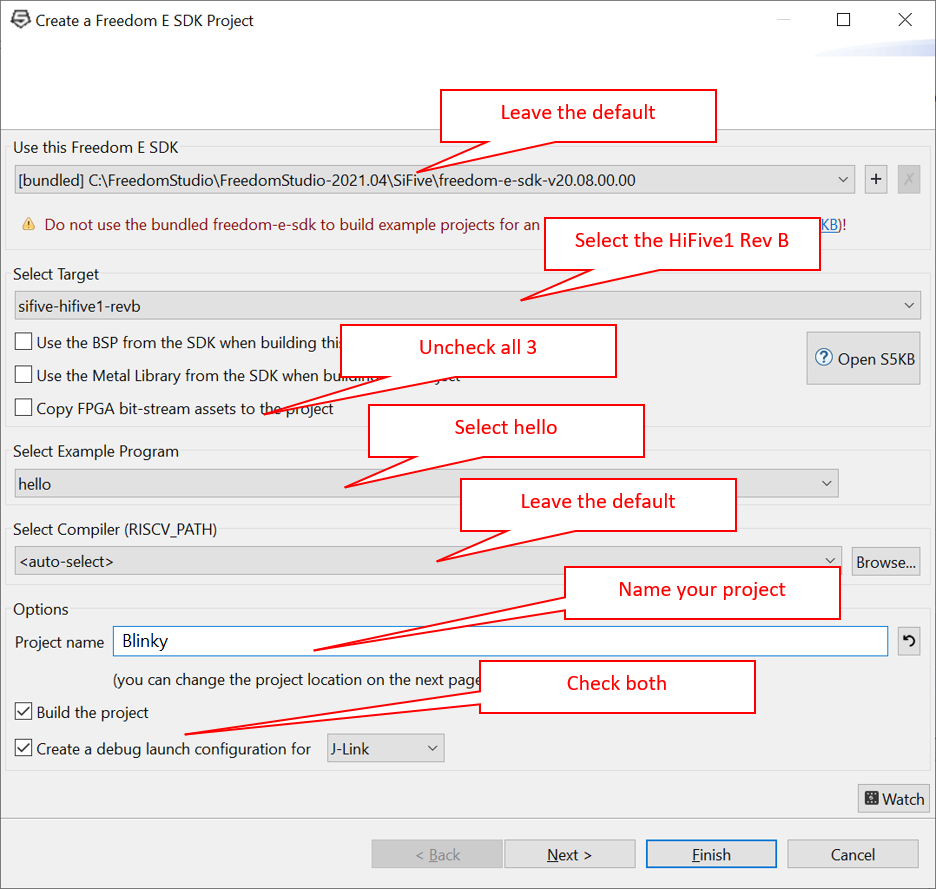
Так же, как и API, совершенно обычным делом является доступ к нескольким библиотекам встроенного ПО, разработанным разными организациями. Например, для Arduino UNO доступно несколько библиотек, включая ту, которая поставляется с Arduino IDE, драйвера, написанные Atmel (поставщиком микроконтроллеров), драйверы, написанные поставщиками устройств, такими как производитель ЖК-модуля, и даже библиотеки, написанные любителями.

Библиотека, которую мы будем использовать в этом курсе, легко доступна в Freedom Studio и написана SiFive. Она называется библиотека ***Freedom Metal*** и включена в пакет поддержки платы (или BSP) для платы HiFive1 Rev B.

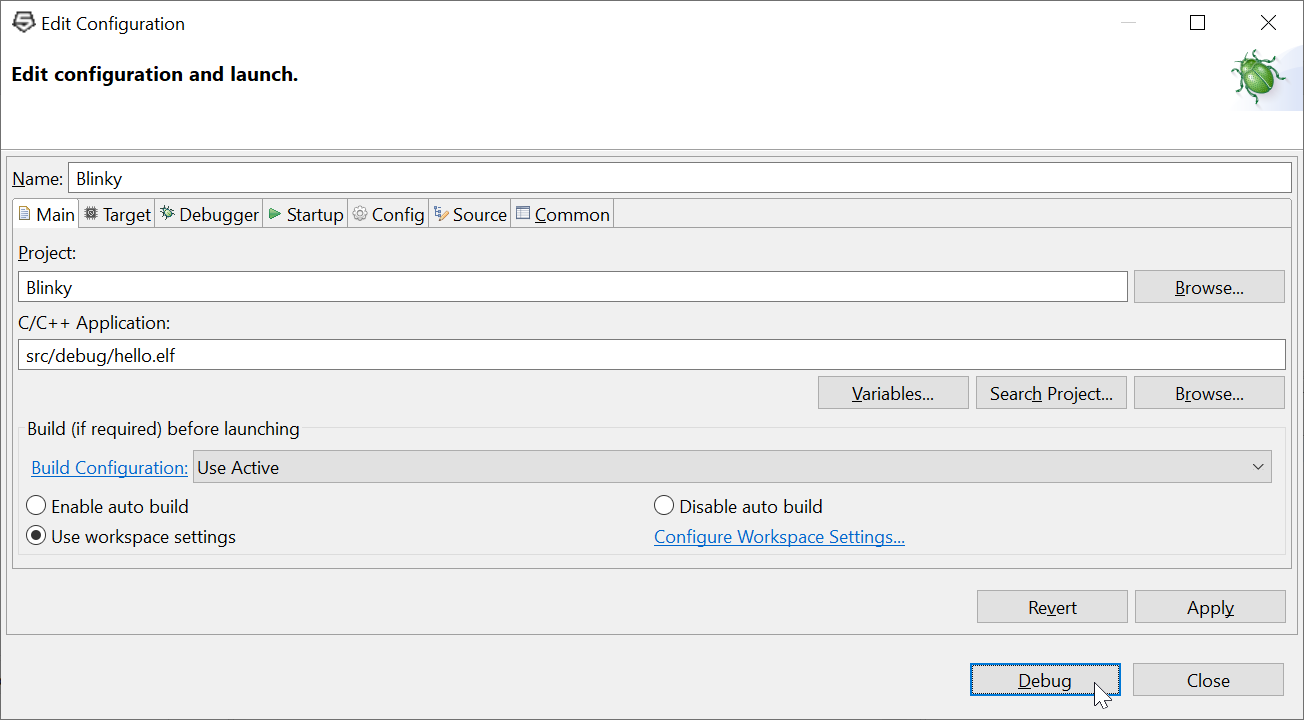
Давайте представим эту библиотеку практическим, экспериментальным способом. Мы создадим проект, использующий исходные файлы из библиотеки Freedom Metal, включенной в BSP.

## Создание проекта в Blinky

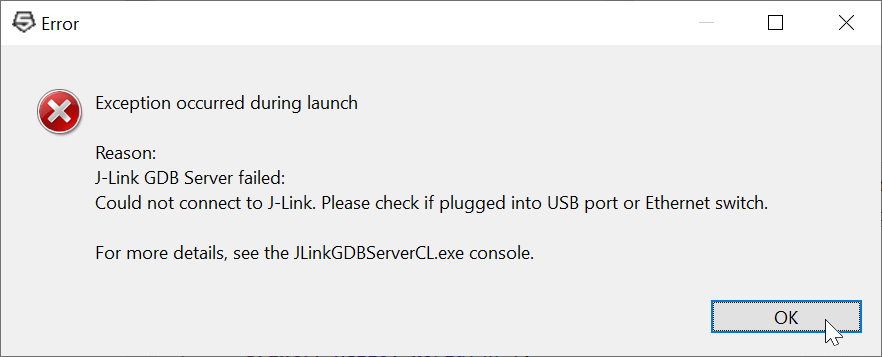
Мы начнем с создания проекта так, как вы узнали в предыдущем разделе. То есть создайте проект Freedom E SKD для платы HiFive1 Rev-B, используя программу hello Example, но назовите его Blinky, как показано ниже. Нажмите на кнопку Готово, когда будете готовы.

  
**Мастер создания проекта Freedom Studio, показывающий рекомендуемые настройки для проекта Blinky**

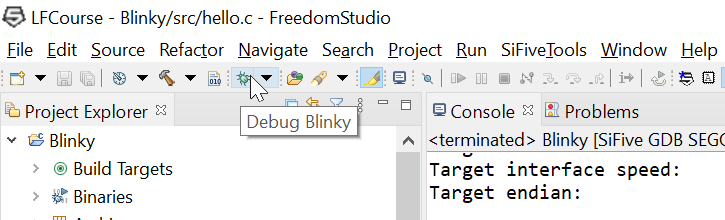
Когда Freedom Studio завершит создание вашего проекта (что может занять некоторое время), примите ее предложение по сборке и запуску конфигурации JLink, чтобы упростить отладку. Нажмите на кнопку Отладки.

  
**Диалоговое окно настройки отладки**

Не волнуйтесь, если вы увидите следующее окно с сообщением. Это просто означает, что у вас плата не подключена к USB-порту. Не стесняйтесь отклонить это сообщение.

  
**Сообщение об ошибке из-за того, что плата не подключена к USB-порту**

После этого вы должны увидеть, что ваш вновь созданный проект открыт в среде IDE. Если вы наведете указатель мыши на кнопку отладки, вы увидите текст всплывающей подсказки, который гласит: Debug Blinky (или любое другое название, которое вы дали своему проекту). Это показывает, что конфигурация отладки J-Link доступна простым нажатием этой кнопки. Мы сделаем это вскоре, когда подключим плату к USB-порту. А пока мы добавим немного кода.

  
**Debug Blinky tooltip**

## Код Blinky

Программа будет мигать. То есть пользовательский светодиод на Red-V Thing Plus будет мигать с постоянной частотой 1 Гц, включаясь на полсекунды и выключаясь на полсекунды.

Следующий фрагмент кода был получен из руководства SparkFun по разработке Red-V и изменен в двух незначительных деталях:

1. **Временная шкала, используемая функцией задержки**

Команда SparkFun, которая тестировала это приложение, использовала плату, отличную от Red-V Thing Plus. В этом коде используется Redboard Red-V, также представленный в этом руководстве. Единицы времени, которые они использовали для достижения задержки в 1 секунду, составляли 2 000 000 единиц. Однако в конечном итоге код разделил эти единицы на 10, чтобы достичь задержки в 1 секунду. Наш случай был другим: функция delay предлагается для получения параметра задержки в микросекундах, и это именно то, что сработало в Red-V Thing Plus. Для задержки в 0,5 секунды в этом параметре было использовано значение 500 000.

1. **Некоторые комментарии были изменены, а некоторые удалены**

Комментарии и код, касающиеся адаптации шкалы времени, были удалены. Ссылки на контакты ввода-вывода платы также были изменены, поскольку первоначально они относились к плате Red-V Red Board, а не к Red-V Thing Plus.

Скопируйте этот код и вставьте его, чтобы заменить содержимое hello.c в вашем проекте Blinky. Кроме того, продолжайте бегло просматривать код, не останавливаясь, чтобы разобраться с каждой функцией. Просто попробуйте разобраться в этом. Мы перейдем к функциям позже.

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

**Red-V Thing Plus Blinky,**

**Minor modifications by Eduardo Corpeño, June-9-2022**

**This code was originally written by: Ho Yun "Bobby" Chan and "Tron Monroe"**

**@ SparkFun Electronics**

**DATE:  11/21/2019**

**DEVELOPMENT ENVIRONMENT SPECIFICS:**

**Firmware developed using Freedom Studio v4.18.0.2021-04-1**

**on Windows 10**

**========== RESOURCES ==========**

**Freedom E SDK**

**========== DESCRIPTION ==========**

**Using the built-in LED. To test with different pin,**

**simply modify the reference pin and connect a standard LED**

**and 100Ohm resistor between the respective pin and GND.**

**LICENSE: This code is released under the MIT License   
(http://opensource.org/licenses/MIT)  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include <stdio.h>      //** **включить библиотеку Serial**

**#include <time.h>       //включить библиотеку таймера**

**#include <metal/gpio.h> //включить библиотеку GPIO, https://sifive.github.io/freedom-metal-docs/apiref/gpio.html**

**// Пользовательская функция задержки записи, поскольку у нас нет такой функции, как у Arduino**

**void delay(int number\_of\_microseconds){**

**clock\_t start\_time = clock();  // Сохранение времени начала**

**// Выполнение цикла до тех пор, пока не будет достигнуто требуемое время**

**while (clock() < start\_time + number\_of\_microseconds);**

**}**

**int main (void) {**

**printf("RED-V Example: Blink\n");**

**struct metal\_gpio \*led0; //** **Создайте образец GPIO**

**// Примечание: Последовательность вызовов этих функций имеет значение!**

**// Получите дескриптор устройства GPIO, т.е. определите вывод IC здесь, где GPIO IC = 5, вывод silkscreen = SCK/5**

**// это индекс устройства GPIO, ссылка на который начинается с 0, обязательно проверьте схему**

**led0 = metal\_gpio\_get\_device(0);**

**// Быстрая проверка, чтобы убедиться, правильно ли мы настроили metal\_gpio, была основана на примере кода "sifive-welcome.c"**

**if (led0 == NULL) {**

**printf("LED is null.\n");**

**return 1;**

**}**

**// Контакты устанавливаются при инициализации, поэтому мы должны отключить их, когда используем в качестве ввода / вывода**

**metal\_gpio\_disable\_input(led0, 5);**

**// Установить gpio в качестве выходных данных**

**metal\_gpio\_enable\_output(led0, 5);**

**// Контакты выполняют несколько функций, убедитесь, что мы отсоединяем все подключенное**

**metal\_gpio\_disable\_pinmux(led0, 5);**

**// Включите вывод**

**metal\_gpio\_set\_pin(led0, 5, 1);**

**while(1){**

**metal\_gpio\_set\_pin(led0, 5, 0);  // Выключите вывод**

**delay(500000);  // 500мс в микросекундах**

**metal\_gpio\_set\_pin(led0, 5, 1);  // Включите вывод**

**delay(500000);**

**}**

**return 0; // недостижимый код**

**}**

# Создание приложения Blinky

Теперь скомпилируйте код и запустите его на своей плате. Помните, что у вас есть два варианта для этого:

Вариант А: Отладьте свою программу

Подключите плату к USB-порту, нажмите кнопку **отладки** и нажмите кнопку **возобновления**, крайнюю здесь слева:

Code execution buttons including Resume  
**Кнопки выполнения кода**

Не забудьте завершить и очистить процесс отладки, прежде чем вернуться к редактированию или попытке повторного запуска вашего кода.

Если ваша конфигурация отладки недоступна, выберите ее, нажав на стрелку раскрывающегося списка кнопки **Отладки**, а затем выполните **отладку конфигураций**…

Вариант Б: Загрузите программу на свою плату

Подключите плату к USB-порту и нажмите кнопку сборки (молоток). Как только приложение будет создано, извлеките hex-файл и скопируйте его на внешний диск вашей платы. Наконец, нажмите кнопку сброса на вашей плате. Если все пройдет хорошо, вы увидите, что светодиод мигает 1 раз в секунду (это частота 1 Гц).

В противном случае, если вы получите какие-либо ошибки компилятора или компоновщика, сделайте все возможное, чтобы обнаружить любые синтаксические или логические ошибки с вашей стороны, исправьте код и повторите попытку.

## Blinky Демо (Видео)

[Eduardo Corpeño] Итак, давайте посмотрим, как это работает. Здесь у нас есть проект в Freedom Studio, и я нажму на кнопку отладки. Обратите внимание на всплывающую подсказку, которая гласит: Debug Blinky. А теперь я начну выполнение, нажав кнопку Возобновить. И обратите внимание, как мигает синий светодиод с частотой, по-видимому, в один герц. Вот оно, оно работает.

## Доступ к объявлениям функций в Freedom Studio

Отлично! У нас есть работающая мигалка. Но откуда взялись эти функции? Ну, очевидно из включенных файлов stdio.h, time.h и metal/gpio.h, конечно. Но где находятся эти файлы?

Если вы перейдете в окно исходного кода и щелкните правой кнопкой мыши на любом имени, макросе или функции, у вас есть возможность получить доступ к исходному файлу, в котором он объявлен или определен.

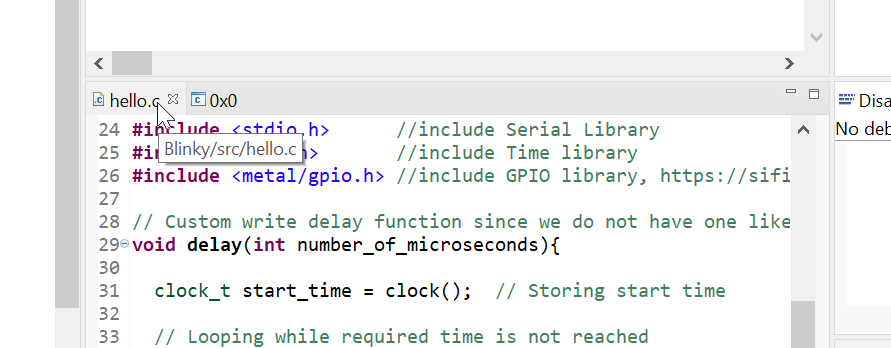
Помните:

***Объявления функций*** являются их прототипами, которые содержат информацию о типе их возвращаемого значения и параметрах.

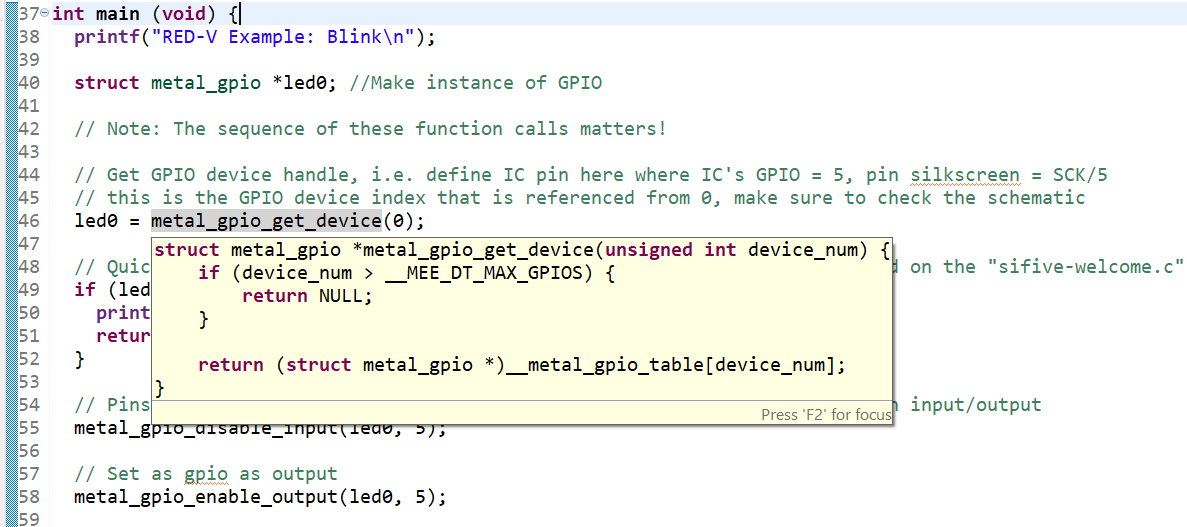
***Определения функций*** - это их реализации с точными строками кода на языке Си, которые описывают их работу.

Некоторые IDE проводят различие между объявлениями и определениями, в то время как другие этого не делают. В Freedom Studio можно увидеть только определения функций с помощью опции контекстного меню **Open Declaration**. Пожалуйста, примите во внимание тот факт, что термины "**определение**" и **"объявление"** используются взаимозаменяемо.

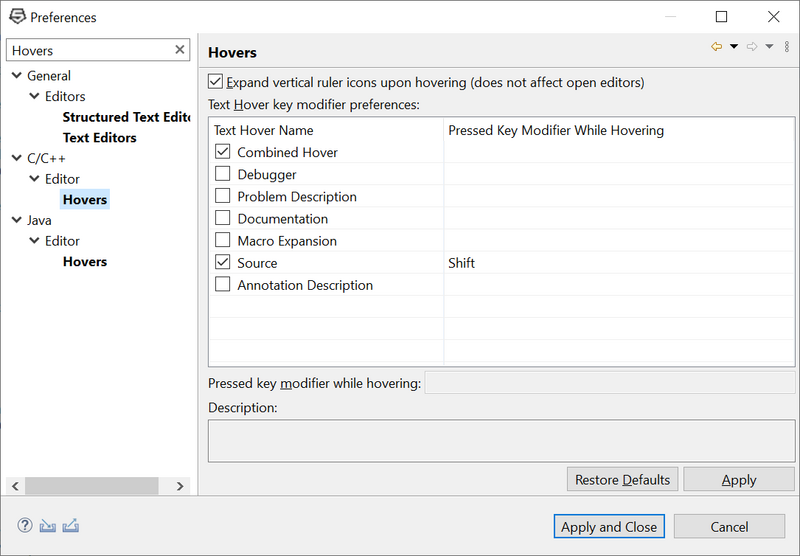
Вы можете развернуть любой раздел, дважды щелкнув по любой из его вкладок. Продолжайте и дважды щелкните на hello.c, чтобы развернуть исходный раздел. Чтобы восстановить его обратно, дважды щелкните на любой вкладке еще раз.

  
**Разверните исходный код на языке Си, дважды щелкнув по имени файла**

Теперь прокрутите вниз около строки 46 и наведите указатель мыши (не щелкайте) на любой символ в имени функции metal\_gpio\_get\_device(). Вы увидите приятную функцию eclipse под названием ***hover***, которая представляет собой большую всплывающую подсказку, показывающую предварительный просмотр определения функции. На самом деле, если вы наведете указатель мыши на всплывающую подсказку, вы получите полосы прокрутки для просмотра всего кода определения функции.

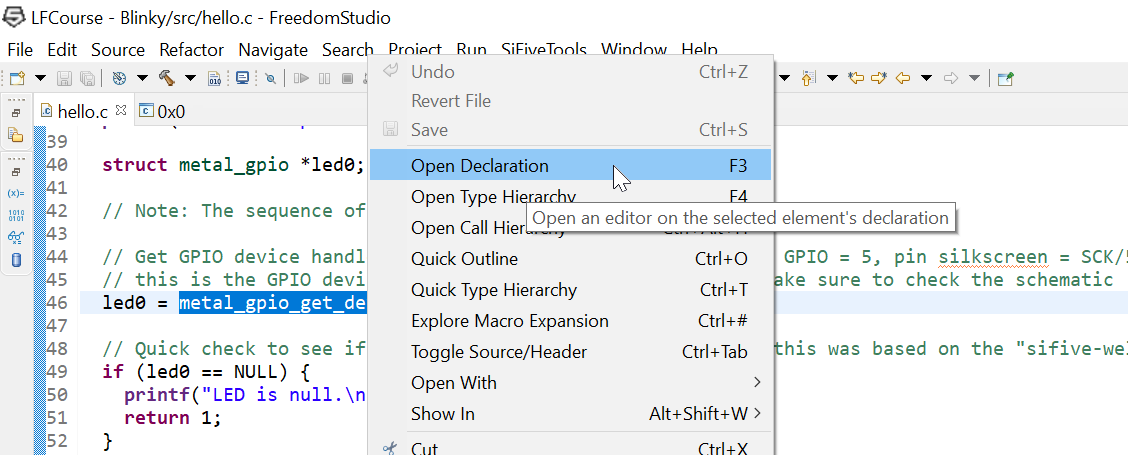
  
**Всплывающая подсказка по определению функции, также известная как hover**

Некоторые программисты находят эти предварительные просмотры всплывающих подсказок раздражающими или навязчивыми. Если вы хотите отключить их, вы можете сделать это, перейдя в "Окно" → "Настройки" и набрав "hovers" в строке поиска (таким образом вы можете найти любые настройки). Среди результатов слева выберите тот, который находится в разделе C/C++ → Редактор, и снимите флажок Комбинированный режим наведения курсора.

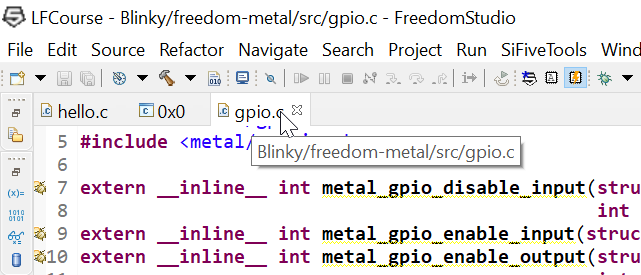
  
**Диалоговое окно настроек, показывающее, как отключить зависание**

# Некоторые объявления функций

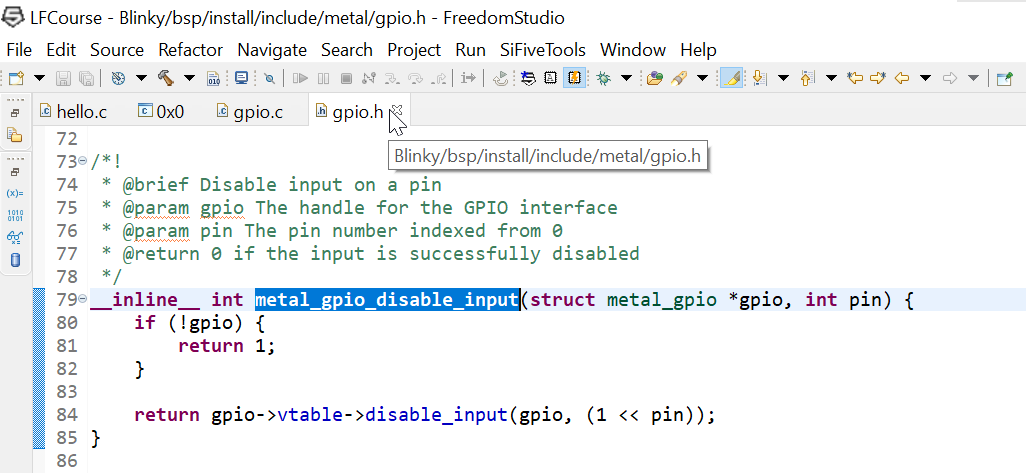
Теперь мы вернемся к коду, щелкнем правой кнопкой мыши на любом символе в имени функции metal\_gpio\_get\_device() рядом со строкой 46 и выберем Открыть объявление в контекстном меню.

  
**Как открыть объявление из контекстного меню**

Вы увидите, что эта функция определена в gpio.c, в папке ./freedom-metal/src/. Наведите курсор на имя файла на вкладке, и вы увидите путь к нему во всплывающей подсказке.

  
**Всплывающая подсказка с путем к исходному файлу**

Теперь вернитесь к hello.c и откройте определение другой функции. На этот раз давайте откроем объявление для metal\_gpio\_disable\_input(), и вы заметите кое-что, что может показаться странным. Пожалуйста, сохраняйте спокойствие и посмотрите на файл, в котором определена эта функция:

  
**Исполняемый код для функции в gpio.h, заголовочном файле**

Да, этот заголовочный файл содержит исполняемый код, но не пугайтесь: все в порядке. Обратите внимание, что все реализованные функции в этом файле являются встроенными функциями.

Ключевое слово \_\_inline\_\_ информирует компилятор о намерении программиста встроить функцию. Встраивание функции имеет множество преимуществ, таких как более быстрое время выполнения, поскольку не требуется ни передавать параметры, ни возвращать значения. В качестве недостатка это также означает больший объем кода, поскольку код функции будет копироваться каждый раз, когда функция используется.

Встраивание определенных функций является обычной практикой во встраиваемых системах, и в этом заголовочном файле мы видим несколько примеров таких функций. Встроенные функции являются одними из немногих исключений, когда допустимо наличие исполняемого кода в заголовочном файле. Все в порядке.

Все остальные функции, используемые в функции main() нашего кода Blinky, находятся в gpio.h:

**metal\_gpio\_disable\_input()**

**metal\_gpio\_enable\_output()**

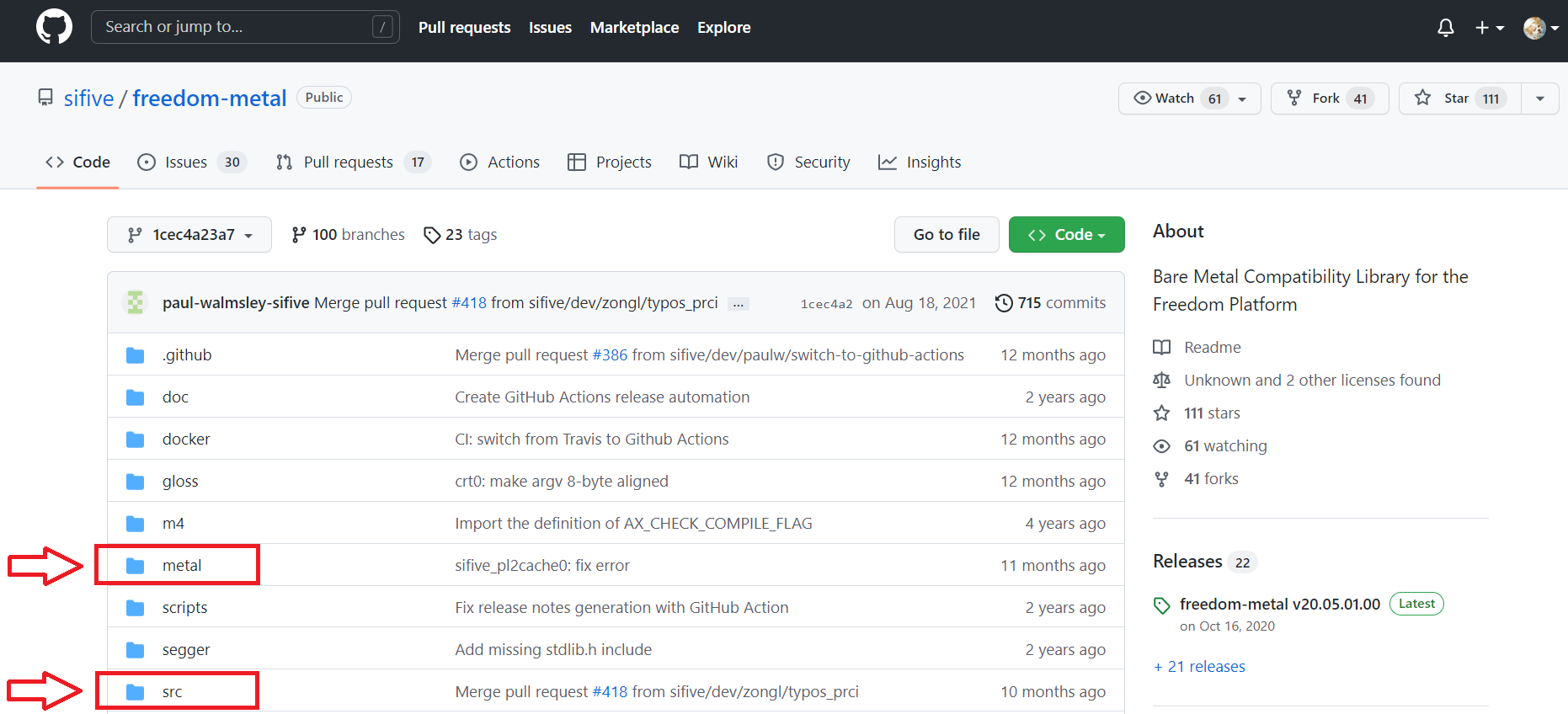
**metal\_gpio\_disable\_pinmux()**

**metal\_gpio\_set\_pin()**

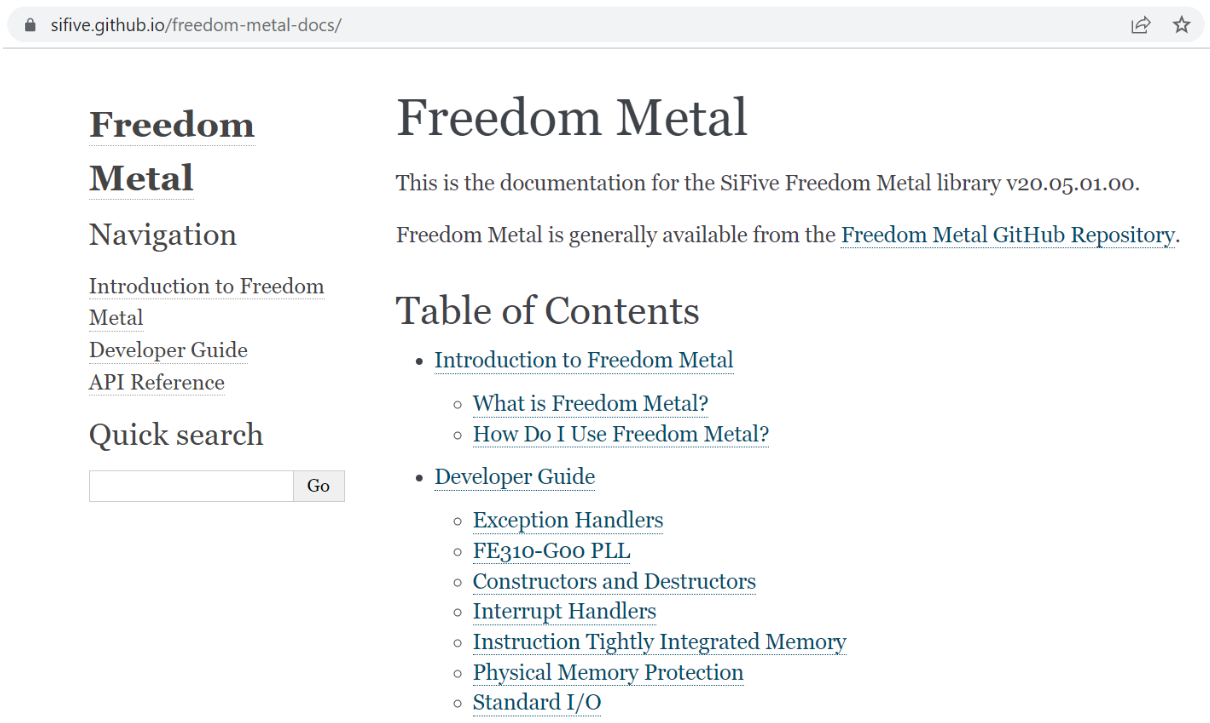
## Обратите внимание, что этот файл находится в папке ./bsp/install/include/metal/.

## Библиотека Freedom Metal Library на GitHub

Вы можете найти все эти файлы в Freedom E SDK. Первым элементом на странице программного обеспечения SiFive является ссылка на репозиторий sifive/freedom-e-sdk на GitHub. Библиотеки, которые мы используем, находятся в папке ./freedom-metal. Затем исходные файлы находятся в папке ./src, а заголовочные файлы - в папке ./metal:

  
**Репозиторий библиотеки Freedom Metal на GitHub**

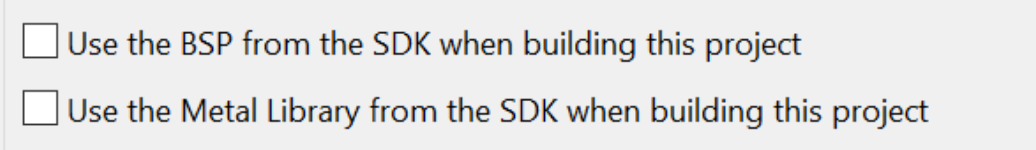
Внизу этой страницы вы можете найти документацию по библиотеке Freedom Metal:

  
**Документация библиотеки Freedom Metal в формате HTML на GitHub**

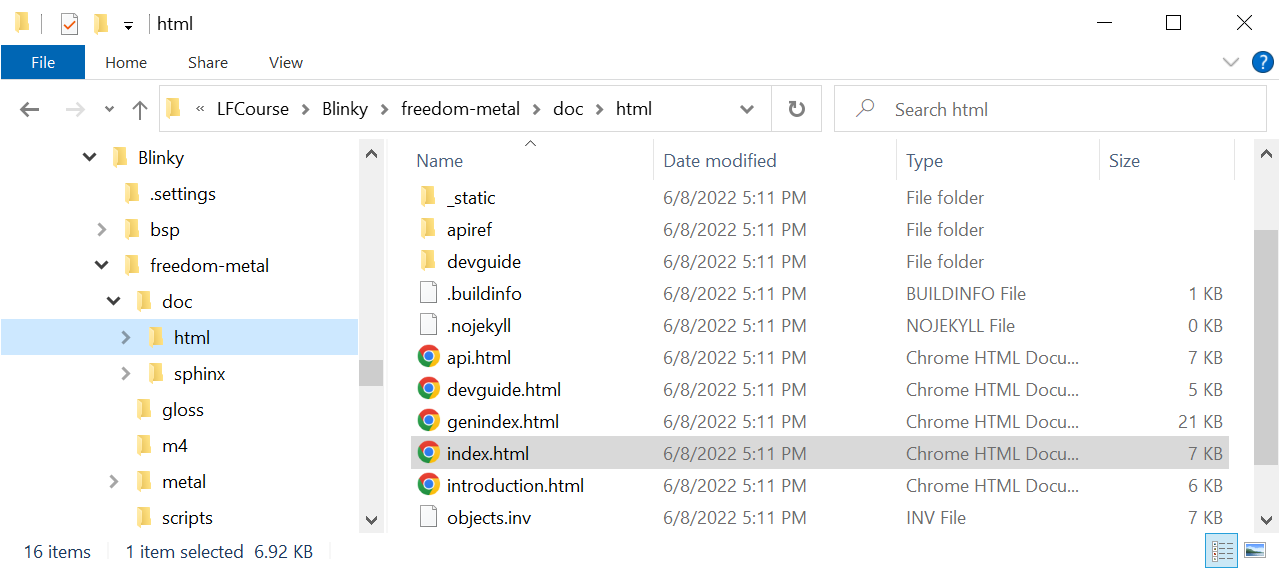
## Библиотека Freedom Metal на вашем компьютере

Библиотека Freedom Metal в папке вашего проекта

Напомним, что мы не устанавливали эти два флажка при создании обоих проектов Freedom E SDK:

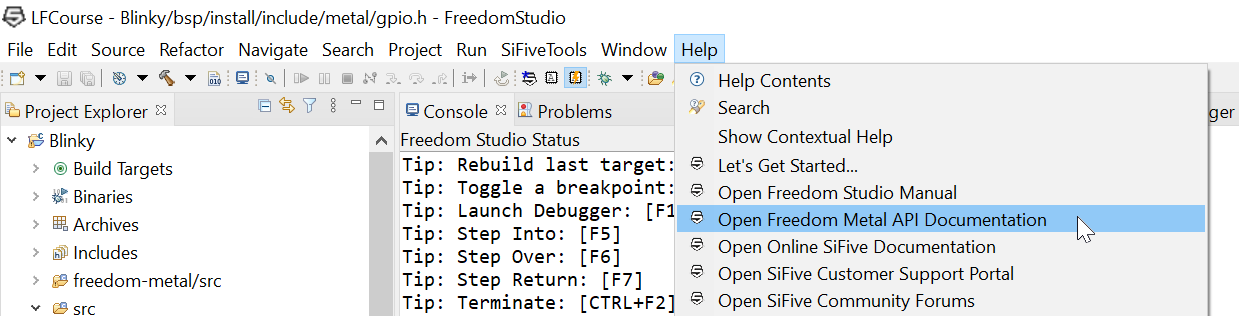
  
**Два неустановленных флажка в мастере создания проекта**

При наведении курсора на эти флажки отображаются всплывающие подсказки, объясняющие, что, если их не устанавливать, создается локальная копия библиотек в каталоге вашего проекта. Это означает, что у вас есть копия исходных файлов и документации в папках вашего проекта. Здесь у нас есть документация в папке проекта Blinky, которая начинается с файла index.html:

  
**Расположение HTML-документации в папке проекта**

Библиотека Freedom Metal в Freedom Studio

Библиотека Freedom Metal интегрирована в Freedom Studio, поскольку она поддерживает все микропроцессоры и микроконтроллеры SiFive. Благодаря этому вы можете легко найти его документацию в меню "Справка".

  
**К HTML-документации можно легко получить доступ из меню справки**

Как вы можете видеть, меню "Справка" содержит множество других полезных документов и ссылок. Найдите минутку, чтобы изучить те, которые кажутся вам интересными.

## Итоги главы

Отлично! Вы справились с этой главой.

Приобретенные вами навыки сразу же пригодятся, когда вы начнете писать свои собственные приложения.

* Теперь вы знаете, где искать любую необходимую вам информацию в документации.
* Вы знаете некоторые основные сведения о микроконтроллере FE310.
* Вы также узнали, как найти определение любой функции, используя специальные возможности Freedom Studio.
* Мы рассмотрели 3 способа доступа к документации библиотеки Freedom Metal.

Не стесняйтесь возвращаться к этому материалу, если вам понадобится освежить его в памяти в будущем.

[Представление 1](#_Toc151298740)

[Обзор главы (Видеоролик) 1](#_Toc151298741)

[Учебные задачи 1](#_Toc151298742)

[Микроконтроллер 1](#_Toc151298743)

[Важная документация 1](#_Toc151298744)

[Red-V Thing Plus 2](#_Toc151298745)

[Микроконтроллеры(MCUs) 3](#_Toc151298746)

[Эта плата пользуется большой популярностью, потому что она построена на базе микроконтроллера GD32VF103, прямого конкурента микроконтроллеру STM32F103 (который оснащен процессором ARM Cortex M-3). Конкуренция, мягко говоря, интересная, поскольку GigaDevice решила использовать ту же номенклатуру, что и STMicro, что означает, что этот чип имеет те же компоненты, что и STMicro. Одно большое отличие заключается в том, что микроконтроллер GD32VF103 от GigaDevice использует ядро RISC-V, точнее, RV32IMAC. Это тот же процессор, который мы будем использовать в Red-V Thing Plus. Если у вас есть опыт работы с микроконтроллерами семейства STM32, эта плата может стать плавным переходом от ARM к RISC-V. 5](#_Toc151298747)

[СнК 5](#_Toc151298748)

[Основные характеристики СнК FE310 6](#_Toc151298749)

[Чтобы узнать больше о функциях микроконтроллера, не стесняйтесь обращаться к техническому описанию FE301-G002. 8](#_Toc151298750)

[Принципиальная схема 8](#_Toc151298751)

[Комплекс GPIO 10](#_Toc151298752)

[Ядро RISC-V 12](#_Toc151298753)

[Библиотека Freedom Metal 13](#_Toc151298754)

[Создание проекта в Blinky 14](#_Toc151298755)

[Код Blinky 16](#_Toc151298756)

[Создание приложения Blinky 19](#_Toc151298757)

[Blinky Демо (Видео) 20](#_Toc151298758)

[Доступ к объявлениям функций в Freedom Studio 20](#_Toc151298759)

[Некоторые объявления функций 22](#_Toc151298760)

[Обратите внимание, что этот файл находится в папке ./bsp/install/include/metal/. 24](#_Toc151298761)

[Библиотека Freedom Metal Library на GitHub 24](#_Toc151298762)

[Библиотека Freedom Metal на вашем компьютере 25](#_Toc151298763)

[Итоги главы 27](#_Toc151298764)