# Широкоимпульсная модуляция

[Широкоимпульсная модуляция 1](#_Toc158977754)

[Обзор главы 2](#_Toc158977755)

[Цели обучения 2](#_Toc158977756)

[ШИМ 2](#_Toc158977757)

[Мысленный эксперимент 2](#_Toc158977758)

[Прямоугольные волны 4](#_Toc158977759)

[Широтно-импульсная модуляция 5](#_Toc158977760)

[Широтно-импульсные модуляторы в микроконтроллерах 6](#_Toc158977761)

[Широтно-импульсный модулятор в FE310 8](#_Toc158977762)

[Упрощенная версия 8](#_Toc158977763)

[Блок-схема ШИМ FE310 9](#_Toc158977764)

[Регистры ШИМ в микроконтроллере FE310 11](#_Toc158977765)

[Регистр конфигурации ШИМ 12](#_Toc158977766)

[Пины ШИМ в Red-V Thing Plus 14](#_Toc158977767)

[Функции ШИМ в библиотеке Freedom Metal 16](#_Toc158977768)

[Функции ШИМ НЕ в библиотеке Freedom Metal 18](#_Toc158977769)

[Адреса регистров ШИМ в библиотеке Freedom Metal Library 18](#_Toc158977770)

[Управление яркостью светодиодов 20](#_Toc158977771)

[Дребезг кнопки 23](#_Toc158977772)

[Избавление от дребезга кнопок 23](#_Toc158977773)

[Демонстрационный код приложения для управления яркостью светодиодов 25](#_Toc158977774)

[Шаговые сервоприводы 29](#_Toc158977775)

[Приложение для контроля сервопривода 32](#_Toc158977776)

[Предупреждения о работе с сервоприводами 34](#_Toc158977777)

[Код приложения для демонстрации сервопривода 35](#_Toc158977778)

[Итоги главы 40](#_Toc158977779)

## Обзор главы

Теперь поговорим о широтно-импульсной модуляции. В этой главе мы поговорим о генерации сигналов. Сначала мы представим, что такое широтно-импульсная модуляция. Затем мы познакомимся с тем, как работает ШИМ в микроконтроллере FE310. И снова мы увидим аппаратное обеспечение из руководства и программное обеспечение в библиотеке Freedom Metal. Далее мы узнаем, как избежать дребезга кнопки. Дребезг - неизбежная проблема с кнопками, и мы узнаем, как ее решить. И на этом этапе мы получим возможность создать приложение для управления яркостью светодиодов. Затем мы сможем внести в это приложение несколько корректировок, чтобы превратить его в приложение для управления шаговыми сервоприводами.

Итак, давайте повеселимся!

## Цели обучения

К концу этой главы, вы должны:

* Понимать принцип работы ШИМ
* Уметь описывать работу регистров конфигурации ШИМ в микроконтроллере FE310
* Управлять яркостью светодиода с помощью ШИМ
* Иметь четкое представление о том, как работают шаговые сервоприводы и как их использовать.
* Уметь подключать внешнее оборудование к Red-V Thing Plus для создания простых ШИМ-приложений.

# ШИМ

## Мысленный эксперимент

Широтно-импульсная модуляция, или сокращенно ШИМ, — это модное название очень простой техники передачи энергии. Она заключается в управлении количеством мощности, передаваемой на нагрузку с течением времени. Во многих случаях эта передаваемая мощность является средней по времени.

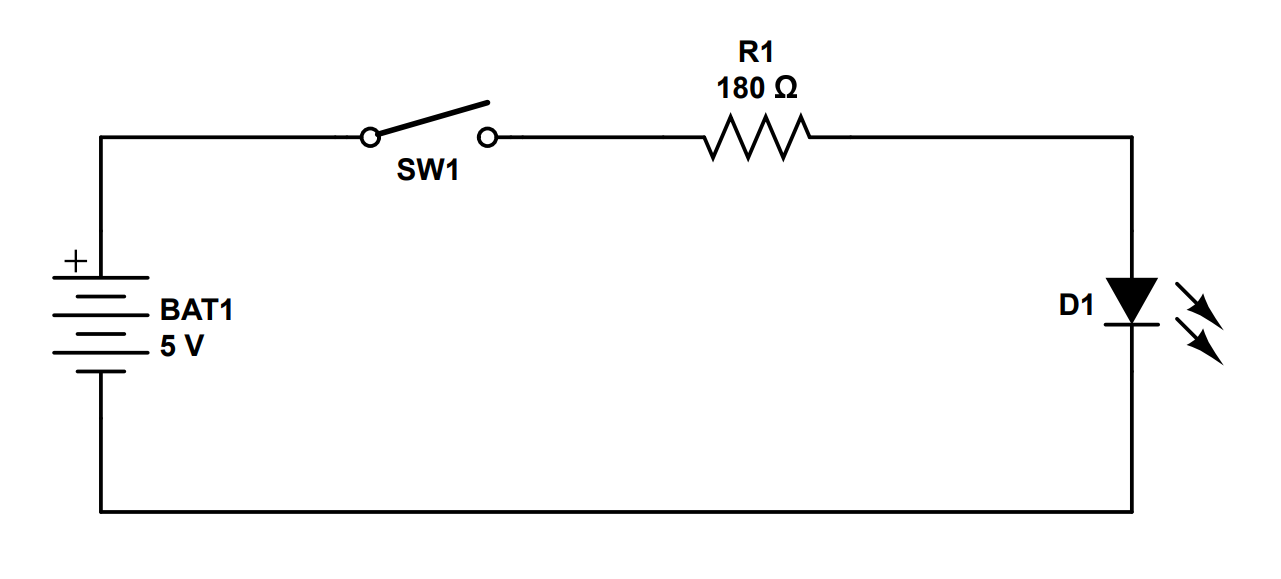
Начнем с мысленного эксперимента: рассмотрим два соседних дома на пляже, которые не заселены, и в каждом из них пока горит только одна лампочка.

Владелец дома А установил таймер, который включает 100-ваттную лампочку на 12 часов, а затем выключает ее на 12 часов каждый день.

Владелец дома Б просто установил 50-ваттную лампочку и держит ее включенной все время.

К концу месяца счет за электроэнергию будет одинаковым для обоих домов, потому что общее потребление энергии будет одинаковым.

Теперь ускорьте эксперимент в меньшем масштабе: представьте на минуту, что вы можете очень быстро включать и выключать выключатель. Насколько быстро? Ну, скажем, вы можете включить светодиод на 500 микросекунд и выключить еще на 500 микросекунд снова и снова, тысячу раз в секунду. Если бы вы могли это сделать, вы бы включали светодиод половину времени и выключали половину времени.

  
**Мысленный эксперимент для многократного включения и выключения светодиода**

Оказалось, что светодиод будет светить в половину своей полной яркости. Это потому, что каждый раз, когда светодиод включен, он будет излучать свет с полной яркостью, а каждый раз, когда он выключен, он не будет излучать свет. Таким образом, в среднем он излучает свет с половиной своей полной яркости.

К счастью, люди не могут заметить, что светодиод мигает так быстро, поэтому мы склонны просто воспринимать среднюю яркость. Этот эффект известен как постоянство зрения, или сокращенно POV.

Конечно, **человек** не может управлять переключателем с такой скоростью, но схема может: **ц**ифровая схема, такая как микроконтроллер, способна периодически выдавать 5 В в течение 500 микросекунд и 0 В в течение 500 микросекунд. Подача этого сигнала на вышеуказанную пару резистор-светодиод (вместо батарейки и выключателя) позволит добиться того же, что и в нашем эксперименте с выключателем.

## Прямоугольные волны

Сигнал, который приведет светодиод к уменьшению яркости в два раза, относится к семейству сигналов, известных как прямоугольные волны, которые имеют несколько параметров. Вот параметры, которые нас интересуют:

* Амплитуда

Вертикальный размер сигнала (измеряется в вольтах для сигнала напряжения или в амперах для сигнала тока). Он может быть выражен в виде высокого и низкого значений, но в данном типе прямоугольной волны низкое значение равно 0. В нашем примере сигнал будет иметь высокое значение 5 В и низкое значение 0 В.

* Период

Горизонтальный размер повторяющейся части волны (измеряется в секундах, миллисекундах или микросекундах). Для нашего примера сигнал имеет период 1 мс.

* Частота

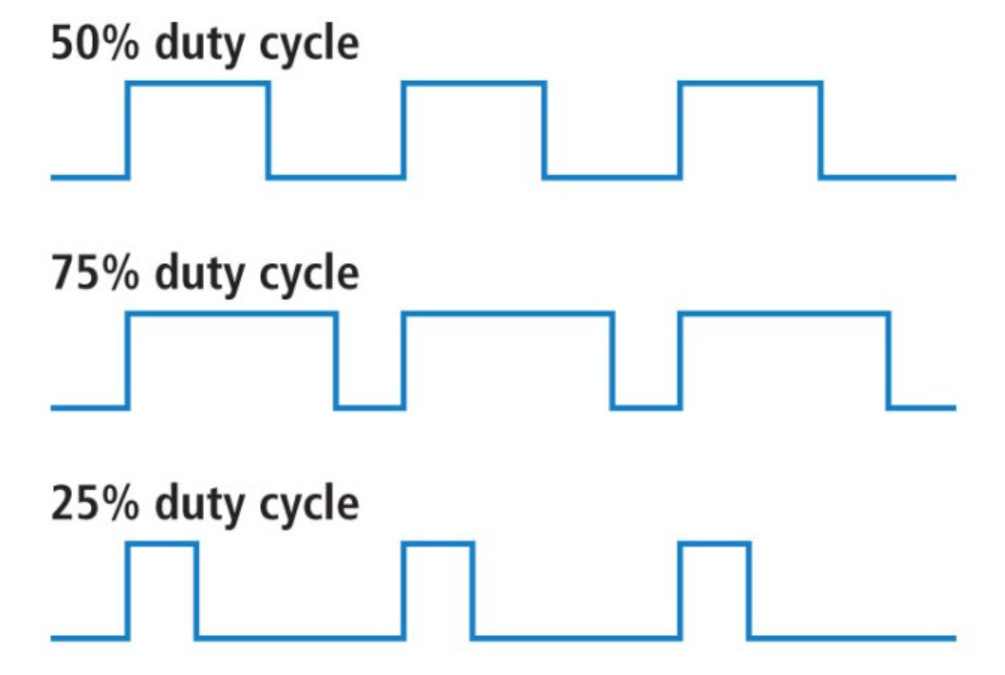
Частота повторения сигнала (измеряется в герцах). Она является обратной (взаимной) величиной периода. В нашем примере сигнал имеет частоту 1 кГц.

* Ширина импульса

Это длительность высокой части волны (иногда это длительность низкой части, в этом случае делается различие). Этот параметр также является временем, поэтому он измеряется в секундах. В нашем примере сигнал имеет длительность импульса 500us (500 микросекунд).

* Рабочий цикл

Отношение ширины импульса к периоду. Это процент, который выражает, сколько процентов от общего периода прямоугольный сигнал остается в состоянии высокого уровня. В нашем примере сигнал имеет рабочий цикл 500us/1,000us = 50%.

Здесь мы имеем 3 сигнала с одинаковым периодом, но разными рабочими циклами:

Три периодических сигнала, показывающие рабочие циклы 50%, 75% и 25%, соответственно

(Взято с сайта SparkFun Electronics, предоставлено по лицензии CC BY-SA 4.0)

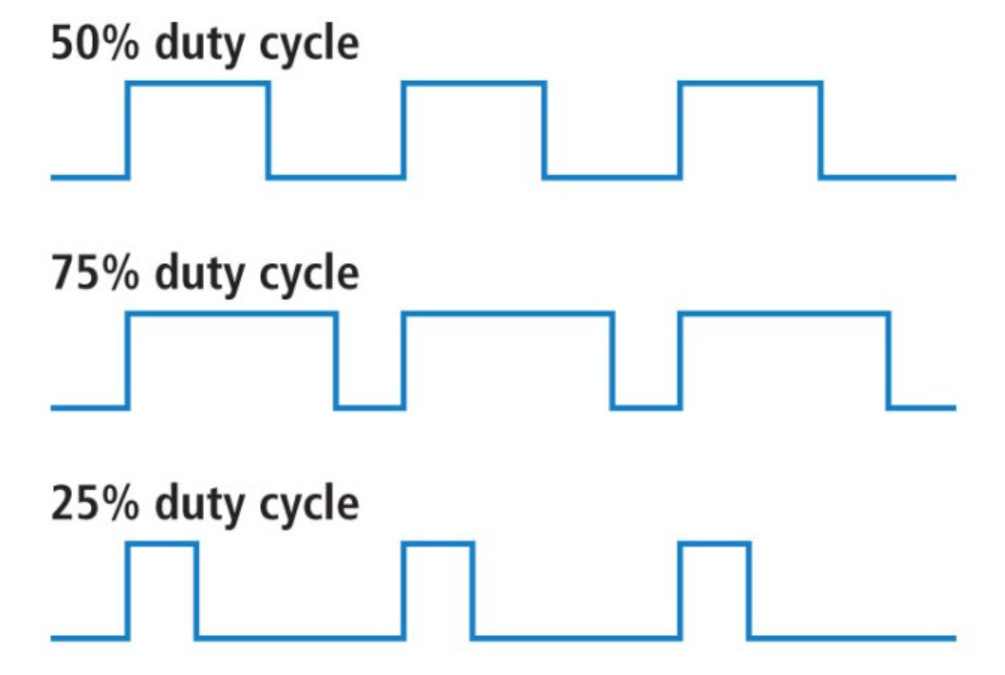
## Широтно-импульсная модуляция

Как уже говорилось, широтно-импульсная модуляция - это модное название чего-то очень простого.

В технике, модуляция - это процесс изменения параметров сигнала для передачи сообщения. Модифицированный сигнал называется несущим, а сообщение - модулирующим сигналом. Возможно, вы знакомы с амплитудной модуляцией и частотной модуляцией - диапазонами AM и FM в радиоприемнике.

Широтно-импульсная модуляция - это процесс изменения ширины импульса прямоугольной волны для передачи определенного процента мощности на нагрузку. В этом случае несущий сигнал - это прямоугольная волна, изменяющаяся в диапазоне от 0 до 1 (например, от 0 до 5 В или от 0 до 3,3 В), модулирующий сигнал - это процент мощности, которую мы хотим передать, а параметр несущего сигнала, который мы изменяем, - это ширина импульса. Поскольку период постоянен, ширина импульса должна иметь длительность между 0 и периодом.

Давайте снова рассмотрим пример ШИМ-сигналов:

  
**Три периодических сигнала с рабочими циклами 50%, 75% и 25% соответственно**

(Взято с сайта SparkFun Electronics, предоставлено по лицензии CC BY-SA 4.0)

Это три случая одного и того же ШИМ-сигнала. Обратите внимание, что период одинаков: все три сигнала достигают высокого уровня в одно и то же время, то есть их длительность (или период) одинаковы.

Если мы подадим эти сигналы на нагрузку (светодиод, двигатель или нагреватель), нагрузка получит дозу общей возможной мощности, эквивалентную рабочему циклу.

Светодиод будет излучать 50%, 75% или 25% от своей полной яркости.

Двигатель будет работать на 50%, 75% или 25% от своей полной скорости.

Нагреватель будет вырабатывать 50%, 75% или 25% своего полного тепла.

Вот и весь ШИМ.

## Широтно-импульсные модуляторы в микроконтроллерах

Практически все микроконтроллеры имеют модуль ШИМ. Чаще всего микроконтроллеры поддерживают ШИМ как функцию в модуле таймера. Модули таймера обычно реализуют операции ввода и вывода. Их схема проста:

Таймеры имеют регистр, который работает как счетчик.

Этот счетчик увеличивает свое значение в каждом цикле входного тактового сигнала.

Этот тактовый сигнал может поступать из нескольких источников, например, от системных часов, и обычно имеется делитель для уменьшения частоты, если он вам нужен.

Этот свободно работающий счетчик с постоянной входной частотой обычно поступает от надежного осциллятора, выступающего в качестве эталона времени.

Таймеры имеют специальный набор регистров, называемых регистрами значений, для выполнения операций ввода и вывода. На каждый входной/выходной вывод приходится один регистр значений.

Здесь представлены 3 наиболее популярные операции ввода/вывода, реализуемые модулями таймеров:

1. **Захват ввода**  
   Эта функция похожа на секундомер. Регистр значений может автоматически сохранять время, когда на входном контакте произошло событие. Например, нам может понадобиться точно знать, когда входной пин перешел из низкого уровня в высокий. В этот точный момент значение счетчика автоматически копируется в регистр значений этого вывода. Схема таймера выполняет эту операцию за нас.
2. **Сравнение выводов**  
   Это похоже на будильник. Выходной вывод может автоматически менять свое состояние, когда счетчик таймера достигает значения, которое мы сохранили в регистре значений. Например, нам может понадобиться выводить 1 в течение определенного времени, поэтому мы можем установить этот вывод в 1, а затем записать значение таймера плюс длительность желаемого импульса в регистр значений (как при установке будильника на 5 часов от текущего времени). Когда счетчик достигнет регистра значения, вывод автоматически изменит свое состояние на 0. Схема таймера позаботится об этой операции за нас.
3. **ШИМ**  
   Схема сравнения выходов вместе с опцией Toggle-on-Overflow (переключение при переполнении) делают ШИМ возможным и простым в реализации. Мы просто включаем функцию Toggle-on-Overflow и записываем желаемую ширину импульса в регистр значений для выбранного вывода. ШИМ-сигнал будет генерироваться на этом выводе, и его ширина импульса будет меняться каждый раз, когда мы изменяем содержимое регистра значений.

## Широтно-импульсный модулятор в FE310

FE310 нетипичен тем, что реализует ШИМ в специальном типе устройства, называемом широтно-импульсным модулятором. У него их 3 (PWM0, PWM1, PWM2), и в нем нет таймеров общего назначения с функциями захвата входа и сравнения выхода. Каждое устройство PWM способно управлять максимум 4мя каналами (0, 1, 2, 3) через назначенные пины GPIO.

Широтно-импульсный модулятор FE310 имеет несколько режимов работы, некоторые из которых накладывают определенные ограничения на генерируемые сигналы. По этой причине в данной главе мы рассмотрим только самый гибкий режим работы, который также является наиболее распространенным случаем использования. Этот режим позволит нам выводить ШИМ-сигналы через каналы 1, 2 и 3, и активируется он установкой конфигурационного бита **pwmzerocmp**.

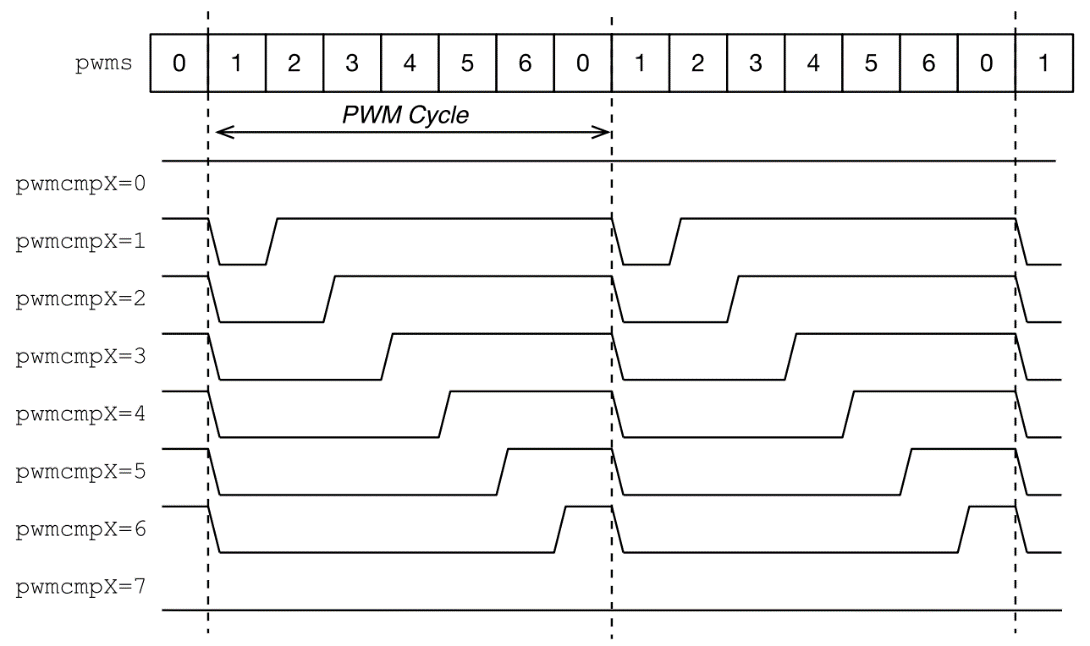
Если вы хотите узнать о других режимах работы (ШИМ с выравниванием по центру, вывод 4-го ШИМ-сигнала через канал 0, периодические прерывания и многое другое), пожалуйста, подождите, пока вы не пройдете эту главу, а затем обратитесь к руководству.

## Упрощенная версия

Из-за большого количества регистров и терминов конфигурация и использование ШИМ-модуля могут быть чрезмерно сложными и запутанными. Поэтому мы рассмотрим интересующий нас конкретный случай использования с простым объяснением.

Допустим, наш счетчик настроен на счет от 0 до 6. Счетчик виден в регистре **pwms**.

Следующая временная диаграмма показывает поведение всех 8 возможных ШИМ-сигналов, генерируемых в канале X (1, 2 или 3). Регистры значений называются *регистрами сравнения*, поэтому для канала X регистр сравнения будет **pwmcmpX**.

**  
Временная диаграмма всех возможных случаев сигнала с периодом 7 единиц счета**

(Изображение из руководства пользователя FE310-G002, воспроизведено с разрешения компании SiFive, Inc.)

Обратите внимание на следующие детали временной диаграммы:

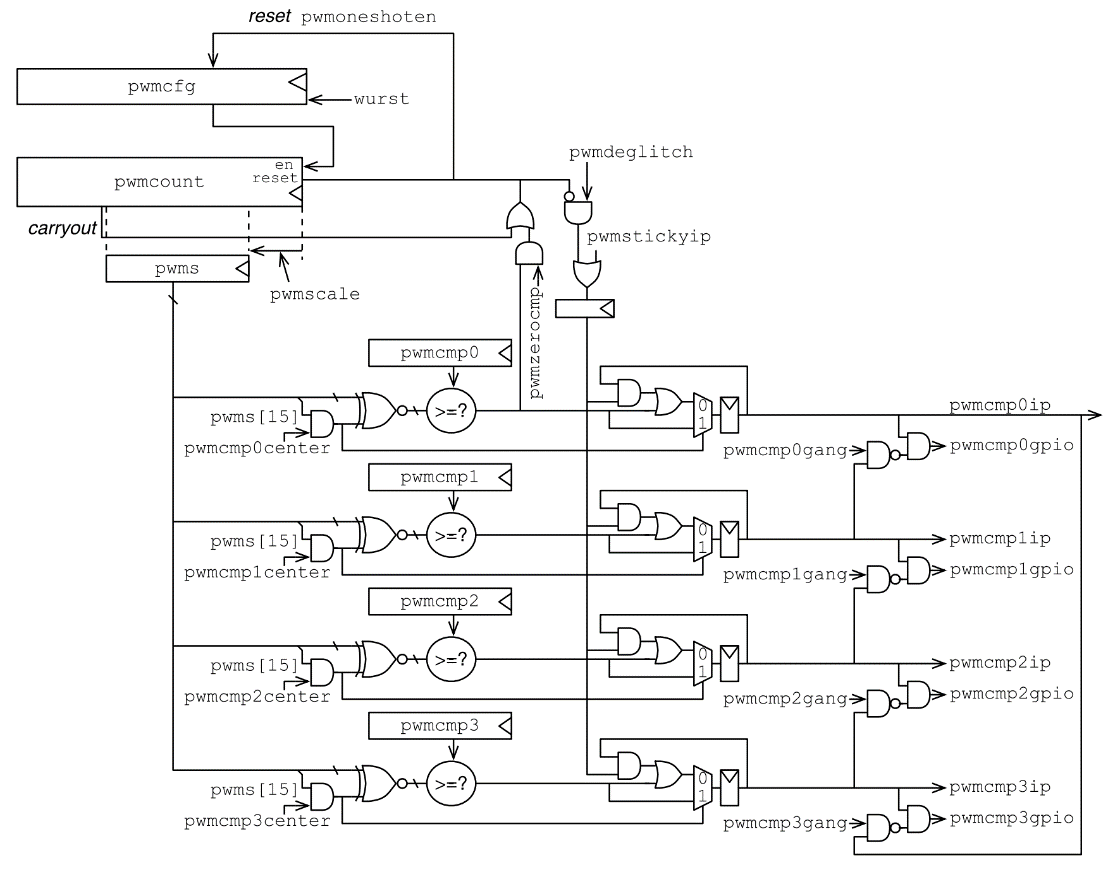
* Выход ШИМ увеличивается, когда счетчик (pwms) достигает значения сравнения.
* Выход ШИМ переходит в низкий уровень, когда счетчик обнуляется.
* Это означает, что регистр **pwmcmpX** управляет шириной импульса низкого уровня, а не высокого.
* Не беспокойтесь об этом. Чтобы заставить **pwmcmpX** управлять широтой импульсов высокого уровня, мы можем инвертировать выход пина GPIO. Помните регистр **out\_xor?**
* Это упрощенный пример с периодом в 7 единиц счета. Хотя это функционально, в ваших практических приложениях, скорее всего, будет использоваться что-то вроде 100, 1 000 или 20 000 единиц счета..

## Блок-схема ШИМ FE310

Здесь мы рассмотрим упрощенную блок-схему ШИМ-контроллера FE310. Важно знать эту схему, поскольку она поможет вам понять, почему нам нужно взаимодействовать с каждым из задействованных регистров.

**Пожалуйста, обратите внимание:** пусть вас не пугает аппаратное обеспечение. Оно не очень сложное, и вы в любом случае будете использовать высокоуровневые программные инструменты для управления им. Просто сядьте поудобнее и терпеливо наблюдайте. В итоге вы получите доступ к высокоуровневой библиотеке для настройки и использования всех модулей.

Для каждого ШИМ-устройства (в FE310 их 3) имеется следующая схема, которая управляет до 4 выходных выводов:

  
**Схема для каждого ШИМ-устройства в микроконтроллере FE310**

(Изображение из руководства пользователя FE310-G002, воспроизведено с разрешения компании SiFive, Inc.)

Теперь мы рассмотрим некоторые детали этой схемы.

Длинные прямоугольники с треугольником справа — это регистры конфигурации ШИМ-устройства. Все они привязаны к памяти (имеют адрес и битовое смещение), и вы можете записывать в них данные программно. Вот что они делают:

* **pwmcfg: Регистр конфигурации ШИМ**  
  Этот регистр содержит биты конфигурации для управления поведением ШИМ-устройства. Все именованные сигналы, подключенные к входам на схеме (кроме **wurst**), являются битами из этого регистра.
* **pwmcount: Регистр подсчета ШИМ**  
  Это счетчик. Он имеет (**cmpwidth** + 15) бит, причем **cmpwidth** – это размер регистров для сравнения (8 бит для **PWM0**, и 16 бит для **PWM1** и **PWM2**). Остальные старшие биты до 31 всегда читаются как 0.

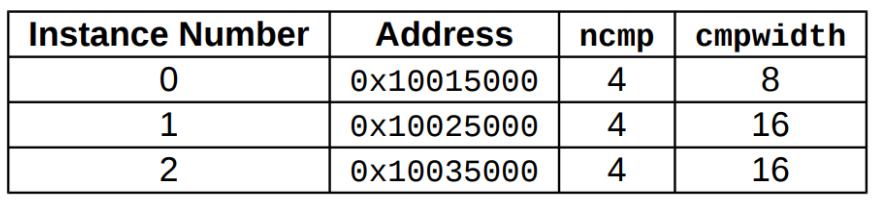
Для PWM0 **pwmcount** — это 23-битный счетчик, поскольку регистры сравнения имеют ширину 8 бит.

Для PWM1 и PWM2 **pwmcount** является 31-битным счетчиком, поскольку регистры сравнения имеют ширину 16 бит.

* **pwms: Масштабированный регистр подсчета ШИМ**  
  Это умная и элегантная реализация пределителя от SiFive. Вместо деления входящей тактовой частоты, конфигурируемая часть счетчика наблюдается и отправляется в регистры сравнения. Технически, это не обязательно должен быть регистр, просто селектор битов **cmpwidth** в **pwmcount**. Смещение битов для этого выбора находится в поле **pwmscale** в регистре **pwmconfig**. Посмотрите на схему, она хорошо показывает эту логику.
* **pwmcmp{1,2,3}: Регистры сравнения ШИМ 1, 2, 3**   
  Это регистры значений. Как показано на схеме, когда счет становится больше или равен значению в одном из этих регистров, его компаратор выдает 1. Когда счет меньше значения в одном из этих регистров, его компаратор выдает 0. Каждый из этих компараторов может влиять на один выходной вывод.
* **pwmcmp0: Регистр сравнения ШИМ 0**  
  Это специальный регистр среди 4 регистров сравнения, поскольку он используется для хранения периода ШИМ-сигналов, которые могут генерироваться на выходных каналах 1, 2 и 3. Выходной вывод, связанный с каналом 0, не будет генерировать ШИМ-сигнал. Вместо этого, когда масштабированный счетчик достигает значения в **pwmcmp0**, счетчик обнуляется. Это означает, что **pwmcmp0** эффективно управляет периодом всех трех выходных сигналов.

## Регистры ШИМ в микроконтроллере FE310

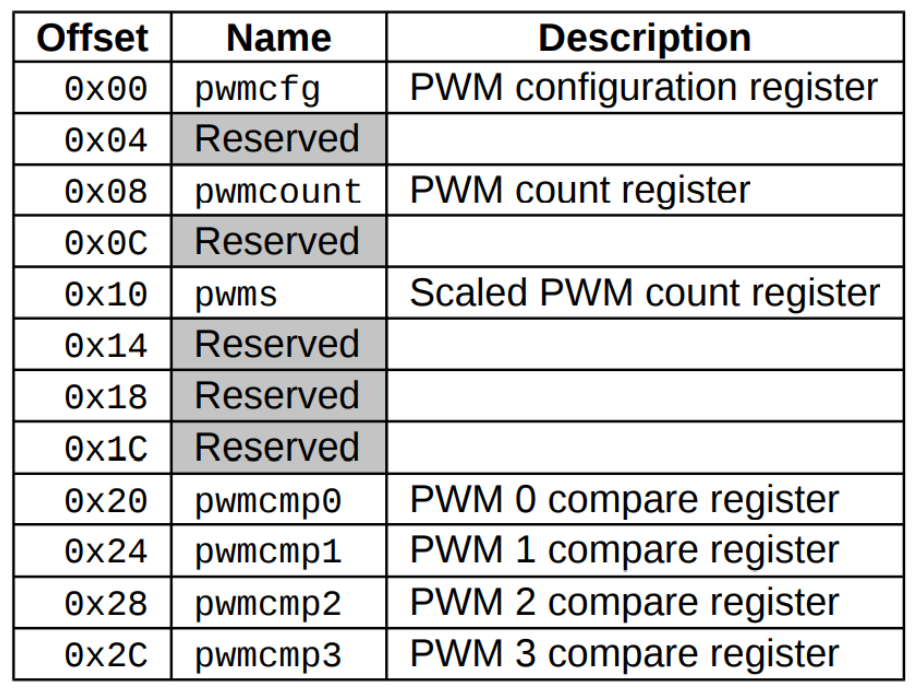
FE310 имеет 3 устройства ШИМ, пользователю доступно до 12 выводов (9 в выбранном нами режиме работы). Как и в случае с GPIO, все регистры конфигурации упакованы в блоки, которые начинаются с соответствующего базового адреса регистра каждого устройства, как показано в таблице 88 руководства пользователя:

  
**Экземпляры устройств ШИМ в микроконтроллере FE310**   
(Изображение из руководства пользователя FE310-G002, воспроизведено с разрешения компании SiFive, Inc.)

**ncmp** - количество каналов, поддерживаемых ШИМ-устройством.

**cmpwidth** - ширина регистров сравнения.

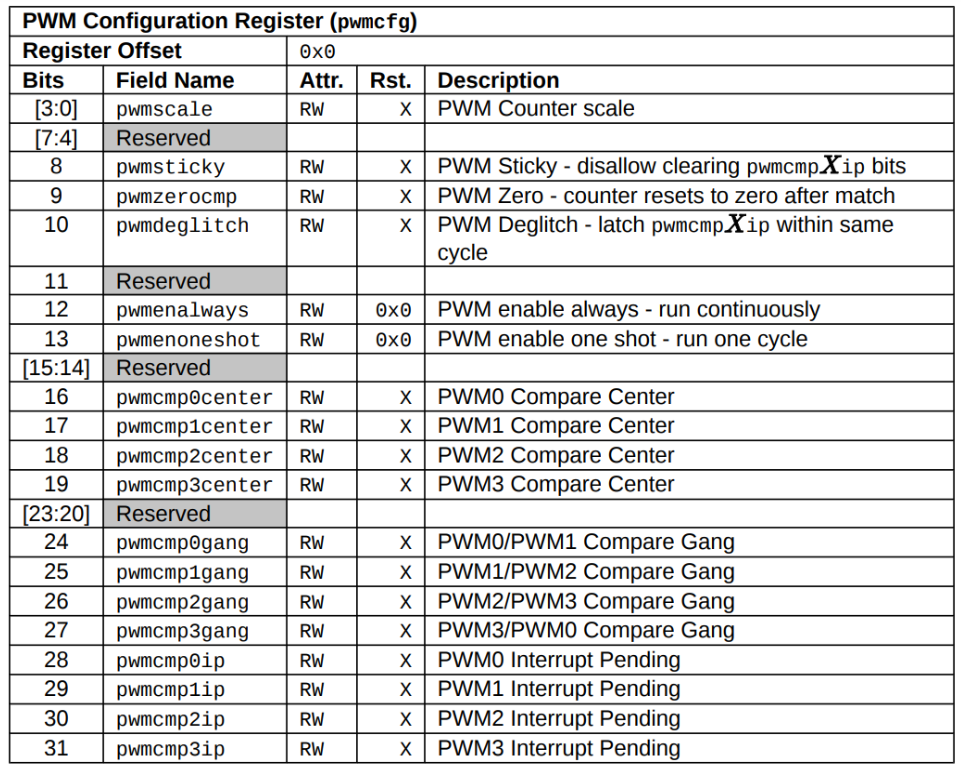
Каждый 32-битный конфигурационный регистр расположен по определенному смещению. Эти смещения указаны в таблице 89 руководства пользователя:

  
**Смещение и описание регистров конфигурации ШИМ**   
(Изображение из руководства пользователя FE310-G002, воспроизведено с разрешения компании SiFive, Inc.)

Помните, что мы можем легко вычислить адрес каждого из конфигурационных регистров ШИМ-устройства, добавив базовый адрес к смещению регистра. Например, **pwmcmp1** PWM1 находится по адресу **0x10025024**, а **pwms** - по адресу **0x10025010**

## Регистр конфигурации ШИМ

**pwmcfg** имеет множество битовых полей конфигурации, которые влияют на работу ШИМ-устройства. Они подробно описаны в таблице 91 руководства пользователя:

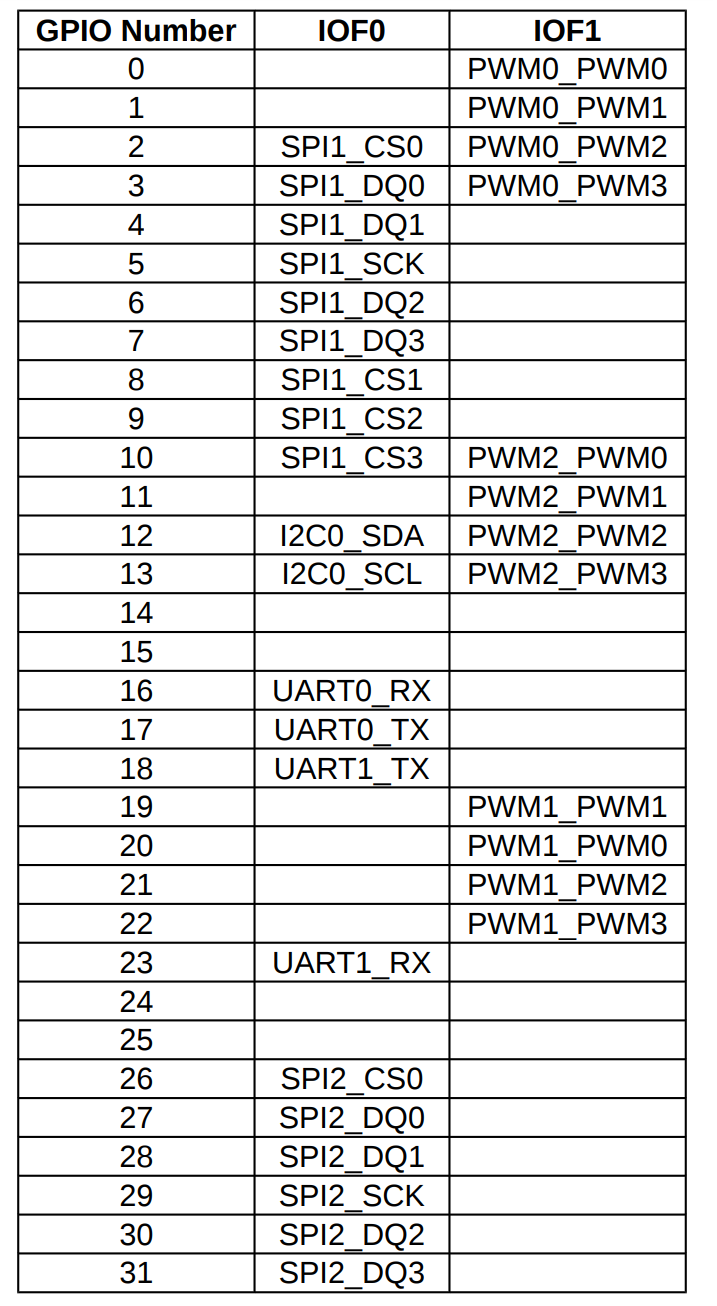
  
**Битовые поля регистра конфигурации ШИМ**   
(Изображение из руководства пользователя FE310-G002, воспроизведено с разрешения компании SiFive, Inc.)

И снова мы сосредоточимся только на тех полях, которые влияют на желаемый режим работы:

* **Биты [3:0], pwmscale**   
  Это 4-битное поле управляет смещением счетчика ШИМ перед подачей его на компараторы ШИМ. Значение в **pwmscale** — это битовая позиция в регистре **pwmcount** начала **cmpwidth**-битового поля pwms. Значение 0 в **pwmscale** означает отсутствие масштабирования, и **pwms** будет равно младшим битам **cmpwidth** регистра **pwmcount**. Максимальное значение 15 в **pwmscale** соответствует делению входящей тактовой частоты на 215 (32 768), поэтому для входной тактовой частоты шины 16 МГц **pwms** будет увеличиваться с частотой 488,3 Гц. Аналогично, значение 1 в **pwmscale** означает деление входной частоты на 2, значение 2 - деление на 4, значение 3 - деление на 8, и так далее, и так далее.
* **Бит 9, pwmzerocmp**  
  Установка этого бита приводит к обнулению счетчика через один цикл после достижения им значения в **pwmcmp0**. Таким образом, мы хотим установить этот бит.
* **Бит 10, pwmdeglitch**  
  Установка этого бита предотвращает переход ШИМ-сигнала с высокого уровня на низкий в цикле ШИМ. Это позволяет нам свободно изменять регистры сравнения, не беспокоясь о глюках в ширине импульсов. Таким образом, мы хотим установить этот бит.
* **Бит 12, pwmenalways**  
  Этот бит включает счетчик безусловно. Другими словами, когда этот бит установлен, у нас есть свободно работающий счетчик. Таким образом, мы хотим установить этот бит.

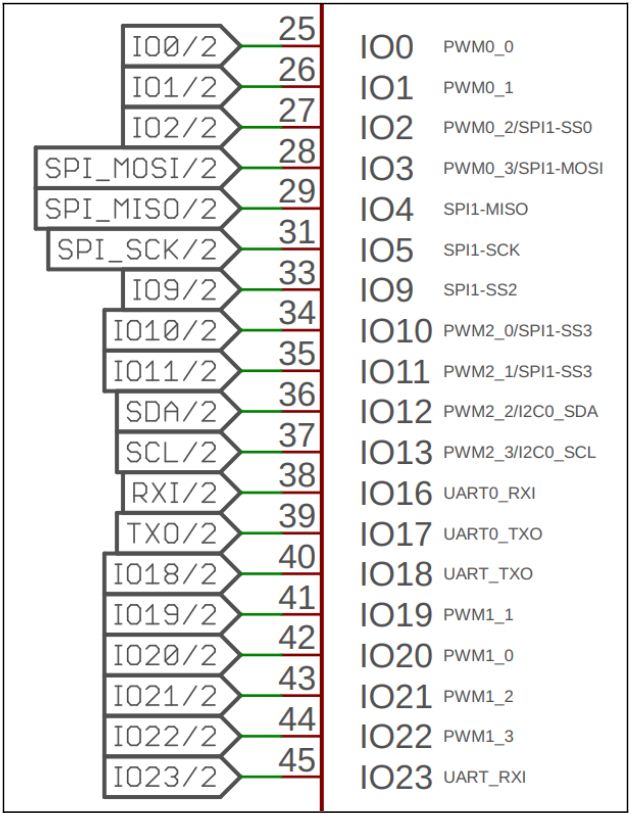
## Пины ШИМ в Red-V Thing Plus

Теперь, если мы хотим передать наши сигналы ШИМ во внешний мир, нам лучше знать, какие пины использовать. Давайте посмотрим на отображение GPIO IOF в таблице 53 руководства FE301:

  
**Сопоставление IOF0 и IOF1 для каждого вывода GPIO**   
(Изображение из руководства пользователя FE310-G002, воспроизведено с разрешения компании SiFive, Inc.)

Обратите внимание, что только 5 выводов имеют две альтернативные функции IOF. Кажется, что SiFive могла бы обойтись только одной функцией IOF для этого микроконтроллера. Однако большинство микроконтроллеров принадлежат к семейству, члены которого имеют больше функций ввода/вывода, чем другие, поэтому мы, возможно, рассматриваем младшего брата этого семейства.

Теперь вернитесь к схеме Red-V Thing Plus и обратите внимание, что все 12 выходов ШИМ подключены к контактам в заголовках ввода/вывода:

  
**Частичная схема Red-V Thing Plus, производитель SparkFun Electronics**   
(Изображение любезно предоставлено SparkFun Electronics)

Помните, что для режима работы, который мы собираемся использовать, у нас есть только 9 доступных выводов ШИМ: 3 на каждое ШИМ-устройство. Выводы канала 0 недоступны для ШИМ, поскольку регистры **pwmcmp0** используются для управления периодом сигналов ШИМ. Однако эти 3 вывода по-прежнему доступны для других функций.

## Функции ШИМ в библиотеке Freedom Metal

Теперь, когда мы знакомы с библиотекой Freedom Metal Library, давайте рассмотрим несколько функций, которые мы будем использовать.

Допустим, мы хотим управлять яркостью светодиода, подключенного к выходу канала 1 в PWM1 (сокращенно PWM1\_1).

Для генерации ШИМ-сигнала частотой 1 кГц с рабочим циклом 70% в PWM1\_1 мы можем просто использовать этот код:

**struct metal\_pwm \*pwm\_1;             // Экземпляр ШИМ  
pwm\_1 = metal\_pwm\_get\_device(1);     // Получение экземпляра ШИМ-устройства  
metal\_pwm\_enable(pwm\_1);             // Включить ШИМ1**

**metal\_pwm\_set\_freq(pwm\_1, 1, 1000);  // Установить частоту ШИМ1\_1 на 1 кГц  
metal\_pwm\_set\_duty(pwm\_1, 1, 70, METAL\_PWM\_PHASE\_CORRECT\_DISABLE); // Установить рабочий цикл для ШИМ1\_1   
metal\_pwm\_trigger(pwm\_1, 1, METAL\_PWM\_CONTINUOUS);                // Запуск в непрерывном режиме**

Код может быть уже понятен, но обратите внимание на следующие детали:

* Эта функция выбирает конфигурацию IOF для всех 4 пинов GPIO, связанных с устройством ШИМ. Это не всегда то, что нам нужно. На самом деле, в данном примере мы хотим использовать только PWM1\_1, который использует **GPIO0**, вывод 19. Поэтому, возможно, мы захотим изменить значение **io\_en** после использования этой функции. Об этом позаботится функция **metal\_gpio\_disable\_pinmux()**. Пины GPIO, связанные с PWM1, - это 19, 20, 21 и 22, поэтому мы можем вызвать эту функцию для тех пинов, которые мы хотим использовать в качестве GPIO.
* **metal\_pwm\_set\_freq()**  
  Эта функция получает в качестве третьего аргумента частоту, которую вы хотите получить от этого ШИМ-сигнала. Это хорошая функция, которая избавляет вас от необходимости вычислять период в единицах предварительно масштабированного счетчика.
* **metal\_pwm\_set\_duty()**  
  Эта функция получает в качестве третьего аргумента нужный вам рабочий цикл, поэтому он должен быть числом от 0 до 100. Более того, это беззнаковое число. В четвертом аргументе мы имеем макрос **METAL\_PWM\_PHASE\_CORRECT\_DISABLE**, что нормально. Альтернативой этому является генерация выровненного по центру ШИМ-сигнала, но это невозможно в выбранном нами режиме
* **metal\_pwm\_trigger()**  
  Это запускает счетчик в непрерывном режиме

## Функции ШИМ НЕ в библиотеке Freedom Metal

Вспомните, что код, который мы только что рассмотрели, будет генерировать ШИМ-сигнал с активным низким уровнем. Другими словами, рабочий цикл выходного сигнала будет составлять 30% вместо 70%. Это может быть нормально, если вы управляете нагрузкой с активным низким уровнем, но, если вам нужно инвертировать логику, вы можете просто установить бит XOR для используемого пина GPIO. В случае с PWM1\_1 это будет пин 19 **GPIO0**.

Мы также можем использовать высокий уровень выходного сигнала, чтобы иметь возможность управлять уровнем яркости внешнего светодиода. Как и в случае с включением подтягивающих резисторов, в библиотеке Freedom Metal Library нет функций для включения или отключения высокой выходной мощности или XOR выхода для инвертирования его значения. Поэтому мы можем использовать макрофункции **Red\_V\_enable\_XOR()** и **Red\_V\_enable\_DS(), приведенные** ниже.

**#define  Red\_V\_enable\_DS(x)     \*((uint32\_t \*) 0x10012014) |= (1<<(x))  
#define Red\_V\_enable\_XOR(x)     \*((uint32\_t \*) 0x10012040) |= (1<<(x))**

**// Мы будем использовать PWM1\_1, который использует GPIO0\_19.  
Red\_V\_enable\_DS(19);  // Высокий уровень сигнала на выходе  
Red\_V\_enable\_XOR(19); // Инвертируем выход для активно-высокого ШИМ**

## Адреса регистров ШИМ в библиотеке Freedom Metal Library

Как и в случае с GPIO, макроопределения базового адреса и смещения регистра можно найти в файле**./bsp/install/include/metal/machine/platform.h**:

**/\* Из pwm@10015000 \*/**  
**#define METAL\_SIFIVE\_PWM0\_10015000\_BASE\_ADDRESS 268521472UL**  
**#define METAL\_SIFIVE\_PWM0\_0\_BASE\_ADDRESS 268521472UL**  
**#define METAL\_SIFIVE\_PWM0\_10015000\_SIZE 4096UL**  
**#define METAL\_SIFIVE\_PWM0\_0\_SIZE 4096UL**

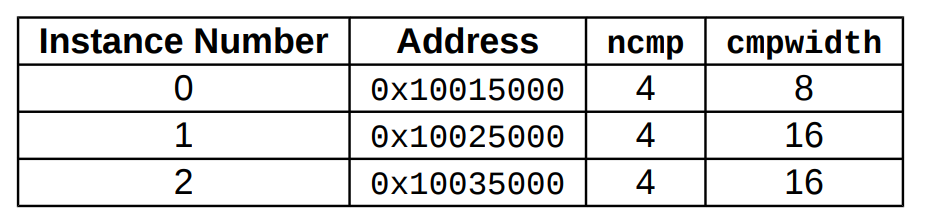
**/\* Из pwm@10025000 \*/**  
**#define METAL\_SIFIVE\_PWM0\_10025000\_BASE\_ADDRESS 268587008UL**  
**#define METAL\_SIFIVE\_PWM0\_1\_BASE\_ADDRESS 268587008UL**  
**#define METAL\_SIFIVE\_PWM0\_10025000\_SIZE 4096UL**  
**#define METAL\_SIFIVE\_PWM0\_1\_SIZE 4096UL**

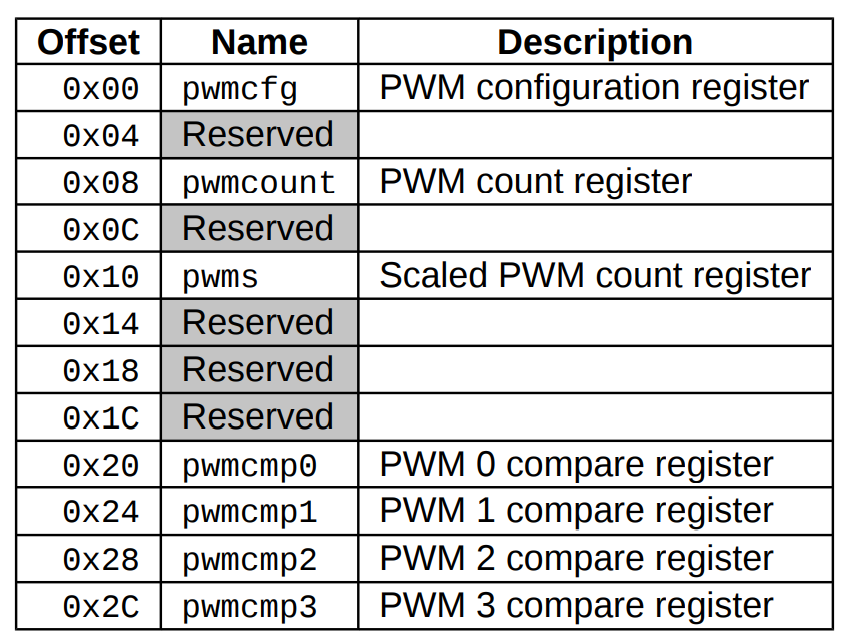
**/\* Из pwm@10035000 \*/**  
**#define METAL\_SIFIVE\_PWM0\_10035000\_BASE\_ADDRESS 268652544UL**  
**#define METAL\_SIFIVE\_PWM0\_2\_BASE\_ADDRESS 268652544UL**  
**#define METAL\_SIFIVE\_PWM0\_10035000\_SIZE 4096UL**  
**#define METAL\_SIFIVE\_PWM0\_2\_SIZE 4096UL**

**#define METAL\_SIFIVE\_PWM0**  
**#define METAL\_SIFIVE\_PWM0\_PWMCFG 0UL**  
**#define METAL\_SIFIVE\_PWM0\_PWMCOUNT 8UL**  
**#define METAL\_SIFIVE\_PWM0\_PWMS 16UL**  
**#define METAL\_SIFIVE\_PWM0\_PWMCMP0 32UL**  
**#define METAL\_SIFIVE\_PWM0\_PWMCMP1 36UL**  
**#define METAL\_SIFIVE\_PWM0\_PWMCMP2 40UL**  
**#define METAL\_SIFIVE\_PWM0\_PWMCMP3 44UL**

**Забавный факт: PWM0\_0**, **PWM0\_1** и **PWM0\_2 — это** имена, которые библиотека присвоила PWM0, PWM1 и PWM2 соответственно.

Если вы обратитесь к таблицам 88 и 89 в руководстве FE310, то увидите, что это базовые адреса ШИМ и смещения регистров. Вот они, для вашего удобства:



  
**Базовые адреса ШИМ и смещения регистров конфигурации**   
(Изображение из руководства пользователя FE310-G002, воспроизведено с разрешения компании SiFive, Inc.)

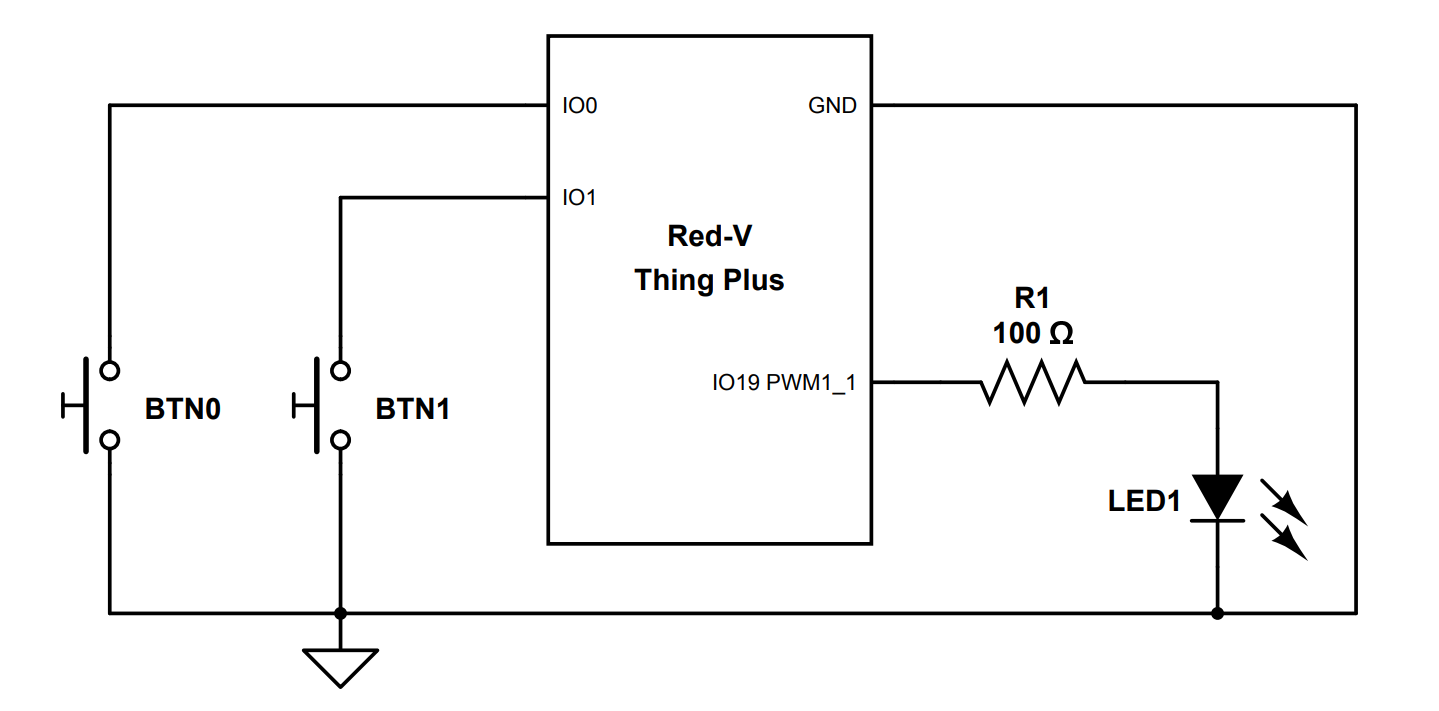
## Управление яркостью светодиодов

Пришло время для другого проекта.

В этой демонстрации будут использоваться те же входные переключатели, что и в демонстрации GPIO, для управления яркостью светодиода с помощью ШИМ. Теперь мы не сможем управлять яркостью встроенного светодиода, потому что он подключен к GPIO0\_5, а этот вывод не поддерживает ШИМ в качестве IOF. Однако мы будем использовать бортовой светодиод для чего-то другого.

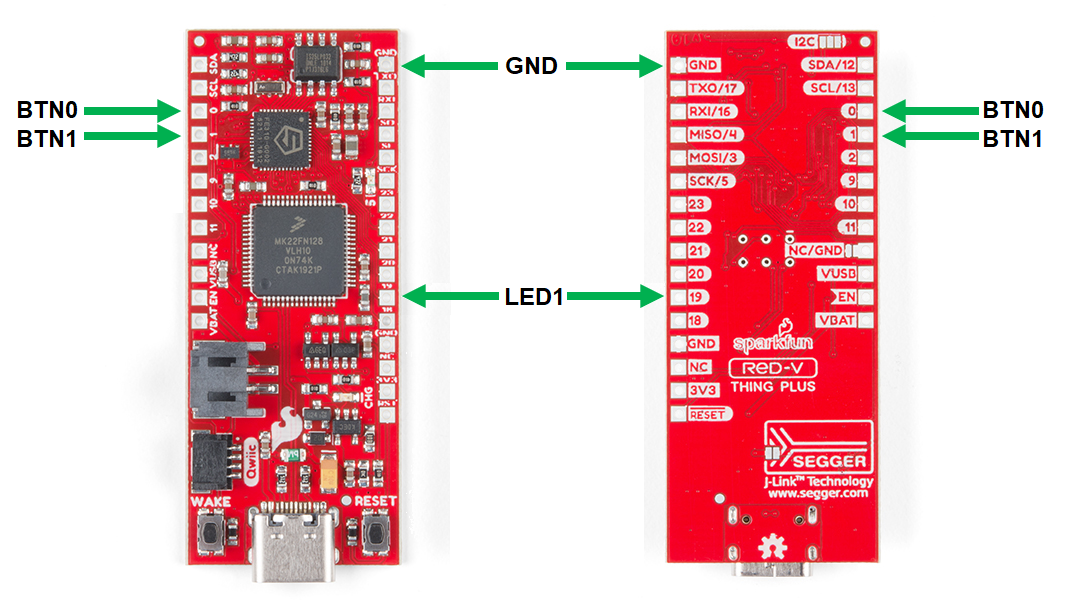
Из-за этого ограничения мы будем использовать ШИМ1\_1 для управления внешним светодиодом с активным высоким уровнем. Помните, что этот выход ШИМ является IOF GPIO0\_19.

Вот принципиальная схема:

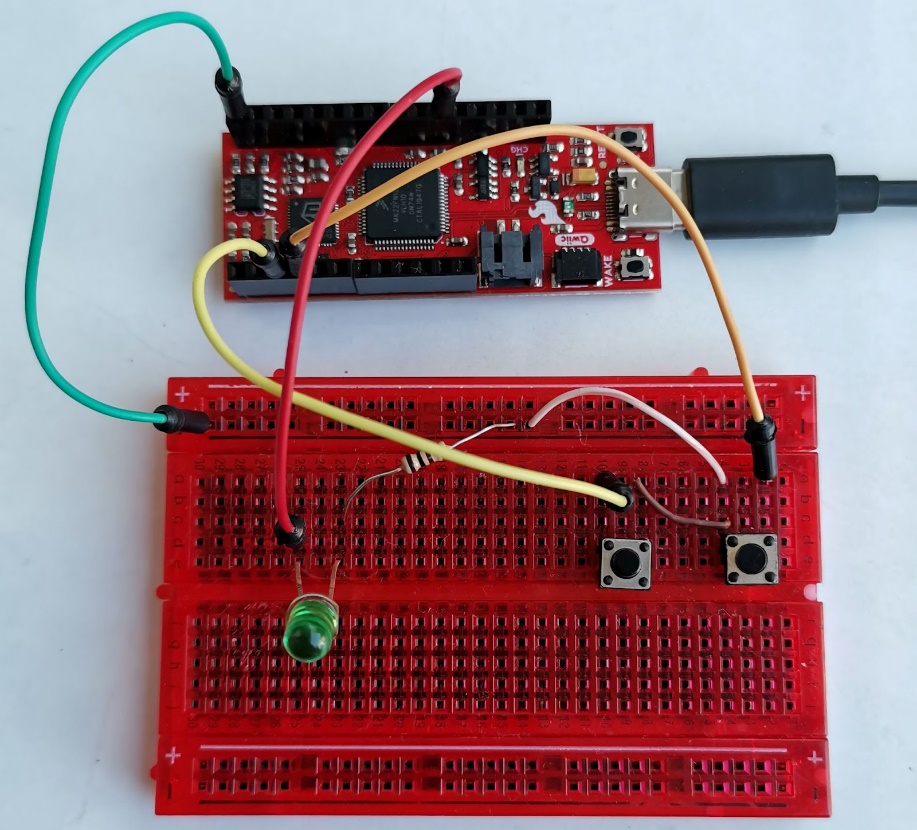
  
**Принципиальная схема демонстрационного приложения ШИМ**

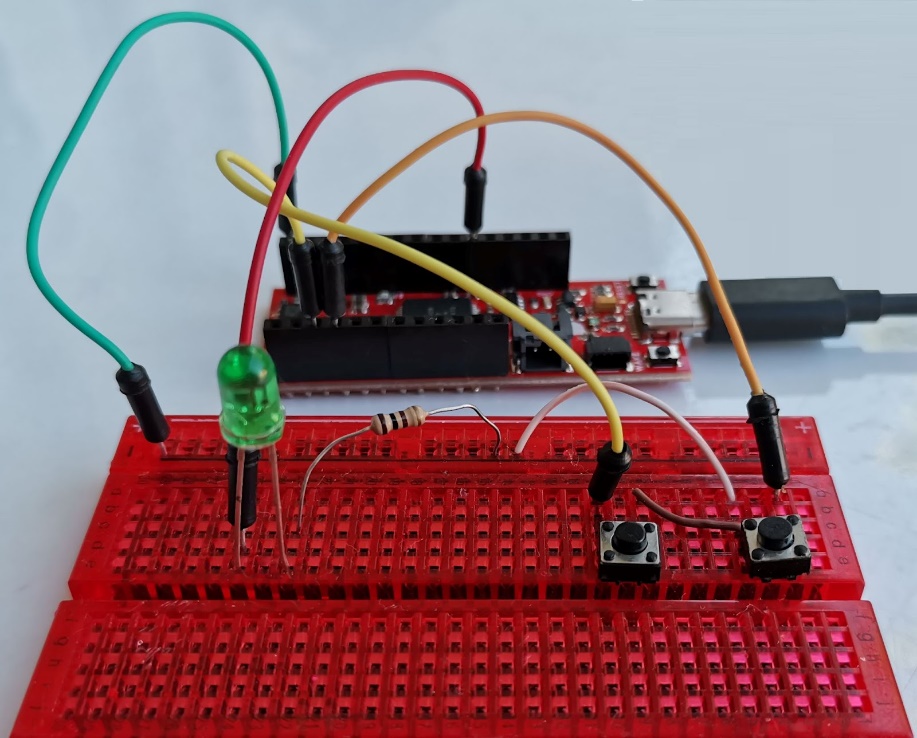
Кнопка 0 уменьшит яркость на ступень, а кнопка 1 увеличит ее на ступень. Внешний светодиод имеет 11 уровней яркости, увеличивающихся с шагом в 10%, поэтому яркость будет увеличиваться от 0% до 100%.

Это контакты ввода/вывода, которые мы будем использовать для подключения кнопок и внешнего светодиода:

  
**Виды сверху и снизу Red-V Thing Plus, показывающие контакты, которые будут использоваться в демонстрационном приложении ШИМ**(взято из [SparkFun Electronics](https://www.sparkfun.com/products/15799), предоставлено по лицензии [CC BY 2.0](https://creativecommons.org/licenses/by/2.0/))

Здесь представлены две фотографии фактических аппаратных соединений:



  
**Аппаратные соединения для демонстрационного приложения LED PWM**

## Дребезг кнопки

Механические переключатели имеют неизбежный недостаток, известный как ***дребезг***.

Название выдает его: когда вы нажимаете на кнопку, вы заставляете две металлические части соприкасаться, и это действие кажется мгновенным, но это не так. После первоначального удара кусочки металла много раз подпрыгивают, могут тереться друг о друга, а через некоторое время их контакт стабилизируется. То же самое происходит, когда вы отпускаете кнопку.

Проблема в том, что цифровая схема будет воспринимать одно нажатие кнопки как несколько (скажем, 4) нажатий. Представьте, что вы нажимаете кнопку **увеличения громкости на** Bluetooth-динамике один раз, но громкость увеличивается на 5 ступеней.

Как это может произойти, если отскок происходит так быстро? Ну, для нас, людей, это происходит быстро. Паразитные импульсы непостоянного переключателя происходят в течение порядка миллисекунд, но цифровая система может воспринимать изменения в течение порядка наносекунд. Это на 6 порядков быстрее! Другими словами, цифровая схема может реагировать на один миллион импульсов, когда переключатель испытывает *дребезг*. Мгновение для нас — это вечность для компьютера.

Наше ШИМ-приложение страдает от дребезга дважды, потому что у него 2 кнопки, и поскольку оно имеет то же входное оборудование, что и наше приложение GPIO, у этого приложения тоже есть дребезг. Однако обнаружить дребезг в нашем приложении GPIO невозможно, потому что оно использует одну кнопку для включения светодиода, а другую - для его выключения. Вы не можете включить или выключить светодиод "сильнее". В нашем приложении PWM нажатие кнопки "**Яркость вниз",** когда яркость светодиода составляет 50%, может привести к полному выключению светодиода. Почему? Из-за неповоротливой кнопки.

## Избавление от дребезга кнопок

Существует множество подходов к развязке выключателя. Здесь представлены два наиболее популярных:

* **RC - фильтры**   
  Это так же просто, как добавить конденсатор параллельно переключателю. RC-цепь задерживает изменения напряжения, физически уменьшая эффект дребезга на электрическом уровне. Технически, этот фильтр низких частот действует аналогично амортизатору в подвеске автомобиля.
* **Программный метод**  
  Простой способ сделать систему устойчивой к дребезгу - реализовать задержку сразу после обнаружения нажатия (или отпускания) кнопки. Обратите внимание, что это не исправляет дребезг переключателя. Вместо этого, это делает приложение устойчивым к дребезгу.

В любом случае, длительность задержки имеет решающее значение: если она будет слишком короткой, отскок все равно повлияет на вашу схему, а если она будет слишком длинной, ваша схема может стать менее отзывчивой на быстрое нажатие клавиш

Мы реализуем программное избавление от дребезга в нашем ШИМ-приложении. Встроенный светодиод будет мигать в течение периода дребезга при нажатии кнопки, как показано ниже:

**while(1){  
  if(Red\_V\_read\_pin(0) == 0){      // Считываем входной пин 0  
    Red\_V\_set\_pin(5);              // Включаем светодиод  
    /\* Полезный код здесь \*/    // Реагируем на нажатие кнопки  
    delay(300);                    // Избегаем дребезг  
    Red\_V\_clear\_pin(5);            // Выключаем светодиод  
    while(Red\_V\_read\_pin(0) == 0); // Ждем, пока кнопка будет отпущена  
    delay(300);                    // Избегаем дребезг  
  }  
  else if(Red\_V\_read\_pin(1) == 0){ // Читаем входной пин 1  
    Red\_V\_set\_pin(5);              // Включаем светодиод  
    /\* Полезный код здесь \*/    // Реагируем на нажатие кнопки  
    delay(300);                    // Избегаем дребезг  
    Red\_V\_clear\_pin(5);            // Выключаем светодиод  
    while(Red\_V\_read\_pin(1) == 0); // Ждем, пока кнопка будет отпущена  
    delay(300);                    // Избегаем дребезг  
  }  
}**

Итак, функция задержки - это быстрый и грязный цикл потери времени, который работает с произвольными единицами времени. Задержка в 300 единиц, использованная выше, была выбрана экспериментально для достижения задержки около 50 миллисекунд

**void delay(int t){  
      volatile uint32\_t x, y;  
      x = t;  
      while(x--){  
            y = x;  
            while(y--);  
      }  
}**

Если вы чувствуете, что с этой функцией задержки что-то не так, то вы угадали. Потеря времени не будет оптимизирована из-за квалификатора **volatile**, поэтому она действительно будет терять время пропорционально аргументу. Однако эта техника сильно зависит от настроек компилятора, поэтому она может привести к разным задержкам для конфигураций debug и release, не говоря уже об использовании других цепочек компиляторов.

Существует множество более подходящих способов сделать функцию задержки с помощью микроконтроллера. Например, мы можем использовать таймер, такой как модуль ШИМ, или часы реального времени. На самом деле, мы видели довольно хорошую функцию задержки в проекте Blinky. Если вы хотите провести эксперимент, замените эту быструю и грязную функцию задержки на функцию из проекта Blinky, используя задержку в 50 миллисекунд (это аргумент в 50 000), и не забудьте **#include <time.h**>.

Чтобы узнать больше о развязывающих переключателях, вы можете прочитать эту [статью от Digi-Key Electronics.](https://www.digikey.com/en/articles/how-to-implement-hardware-debounce-for-switches-and-relays)

## Демонстрационный код приложения для управления яркостью светодиодов

Теперь, когда мы увидели многие части кода, пришло время собрать его воедино. Вот код приложения для управления яркостью светодиодов. Если вы следите за развитием событий, создайте новый проект на основе проекта примера hello, как обычно, и скопируйте следующий код, заменив им содержимое файла **hello.c**.

Бесконечный цикл просто проверяет значение входных контактов (деблокированных) и соответственно увеличивает или уменьшает яркость светодиода.

Попробуйте приложение, чтобы убедиться, что оно работает правильно.

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
    Red-V Thing Plus PWM\_Demo,  
    by Eduardo Corpeño**

**Controlling the brightness of the built-in LED with two external buttons using PWM.**

**Date: June 20, 2022  
    Developed using Freedom Studio v4.18.0.2021-04-1 on Windows 10  
    LICENSE: This code is released under the CC BY-SA 4.0 License  
    (https://creativecommons.org/licenses/by-sa/4.0/) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include <metal/pwm.h>  
#include <metal/gpio.h>  
#include <stdint.h>**

**// Наши собственные макросы, определенные для эффективности или потому, что   
// Freedom Metal Library не обеспечила их функциональность.**

**#define Red\_V\_enable\_output(x) \*((uint32\_t \*) 0x10012008) |= (1<<(x))  
#define Red\_V\_set\_pin(x)       \*((uint32\_t \*) 0x1001200C) |= (1<<(x))  
#define Red\_V\_clear\_pin(x)     \*((uint32\_t \*) 0x1001200C) &= ~(1<<(x))  
#define Red\_V\_read\_pin(x)     (\*((uint32\_t \*) 0x10012000) & (1<<(x)))  
#define Red\_V\_enable\_pullup(x) \*((uint32\_t \*) 0x10012010) |= (1<<(x))  
#define Red\_V\_enable\_DS(x)     \*((uint32\_t \*) 0x10012014) |= (1<<(x))  
#define Red\_V\_enable\_XOR(x)    \*((uint32\_t \*) 0x10012040) |= (1<<(x))**

**// 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) {  
  unsigned int dc = 50;        // Рабочий цикл по умолчанию 50%  
  struct metal\_gpio \*gpio\_0;   // Экземпляр GPIO  
  struct metal\_pwm \*pwm\_1;     // Экземпляр ШИМ**

**// Конфигурация GPIO для встроенного светодиода и кнопок**

**gpio\_0 = metal\_gpio\_get\_device(0);     // Получение экземпляра устройства GPIO  
  // Светодиод  
  metal\_gpio\_disable\_input(gpio\_0, 5);   // Отключить ввод 5  
  Red\_V\_enable\_output(5);                // Включить вывод 5  
  metal\_gpio\_disable\_pinmux(gpio\_0, 5);  // Отключить альтернативные функции для пина 5  
  // Нажатие кнопок  
  metal\_gpio\_enable\_input(gpio\_0, 0);    // Включить ввод 0  
  metal\_gpio\_enable\_input(gpio\_0, 1);    // Включить ввод 1  
  metal\_gpio\_disable\_output(gpio\_0, 0);  // Выключить вывод 0  
  metal\_gpio\_disable\_output(gpio\_0, 1);  // Выключить вывод 1  
  metal\_gpio\_disable\_pinmux(gpio\_0, 0);  // Отключить альтернативные функции для пина 0**

**metal\_gpio\_disable\_pinmux(gpio\_0, 1);  // Отключить альтернативные функции для пина 1  
  Red\_V\_enable\_pullup(0);                // Включить подтягивающий резистор для пина 0  
  Red\_V\_enable\_pullup(1);                // Включить подтягивающий резистор для пина 1**

**// Конфигурация ШИМ. Мы будем использовать PWM1\_1, который использует GPIO0\_19.  
  Red\_V\_enable\_DS(19);                   // Высокий уровень сигнала на входе  
  Red\_V\_enable\_XOR(19);                  //** **Инвертируем выход для активно-высокого ШИМ**

**pwm\_1 = metal\_pwm\_get\_device(1);       // Получение экземпляра ШИМ-устройства  
  while (pwm\_1 == NULL);                 // Остановка. Что-то пошло не так.  
  metal\_pwm\_enable(pwm\_1);               // Включить ШИМ1  
  metal\_gpio\_disable\_pinmux(gpio\_0, 20); // Отключить альтернативные функции для вывода 20  
  metal\_gpio\_disable\_pinmux(gpio\_0, 21); // Отключить альтернативные функции для вывода 21  
  metal\_gpio\_disable\_pinmux(gpio\_0, 22); // Отключить альтернативные функции для вывода 22  
  metal\_pwm\_set\_freq(pwm\_1, 1, 1000);    // Установка частоты ШИМ1\_1 на 1 кГц для светодиода**

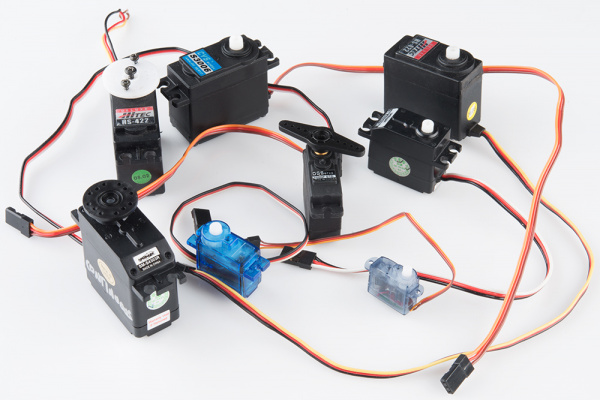
**metal\_pwm\_set\_duty(pwm\_1, 1, dc, METAL\_PWM\_PHASE\_CORRECT\_DISABLE); // Установить рабочий цикл для PWM1\_1  
  metal\_pwm\_trigger(pwm\_1, 1, METAL\_PWM\_CONTINUOUS);                 // Запуск в непрерывном режиме**

**while(1){  
          if(Red\_V\_read\_pin(0) == 0){    // Считываем входной пин 0  
            Red\_V\_set\_pin(5);            // Включаем светодиод  
            dc = (dc>0)? dc - 10 : dc;   // Шаг 10% для яркости  
            metal\_pwm\_set\_duty(pwm\_1, 1, dc, METAL\_PWM\_PHASE\_CORRECT\_DISABLE);  
            delay(300);                   // Избегаем дребезг  
            Red\_V\_clear\_pin(5);           // Выключаем светодиод  
            while(Red\_V\_read\_pin(0) == 0);// Ждем, пока кнопка будет отпущена  
            delay(300);                   // Избегаем дребезг  
          }  
          else if(Red\_V\_read\_pin(1) == 0){ // Считываем входной пин 1  
            Red\_V\_set\_pin(5);              // Включаем светодиод  
            dc = (dc<100)? dc + 10 : dc;  // Шаг 10% для яркости  
            metal\_pwm\_set\_duty(pwm\_1, 1, dc, METAL\_PWM\_PHASE\_CORRECT\_DISABLE);  
            delay(300);                    // Избегаем дребезг  
            Red\_V\_clear\_pin(5);            // Выключаем светодиод  
            while(Red\_V\_read\_pin(1) == 0); // Ждем, пока кнопка будет отпущена  
            delay(300);                    // Избегаем дребезг  
          }  
  }  
  return 0; // Недостижимый код возврата  
}**

## Шаговые сервоприводы

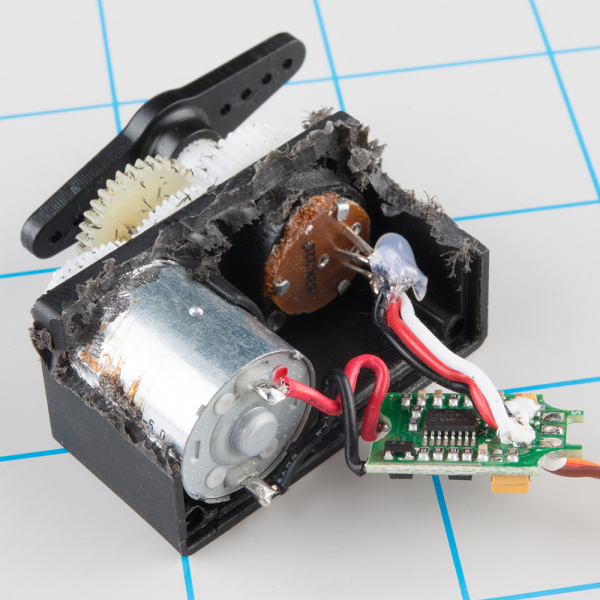
Всеми любимый электромеханический привод — это шаговый сервопривод, также известный как радиоуправляемый сервопривод или просто RC-серво. Не такой скучный, как двигатель постоянного тока, шаговый сервопривод имеет множество применений благодаря вращательному и (как бы) линейному механическому эффекту, который он может производить в окружающей среде.

Существует множество типов сервомеханизмов. Нас интересуют аналоговые, ограниченного вращения и сервоприводы с угловым управлением. Более того, мы будем использовать наиболее часто применяемые параметры для радиоуправляемых авиамоделей. На следующем рисунке представлен ассортимент таких сервоприводов.

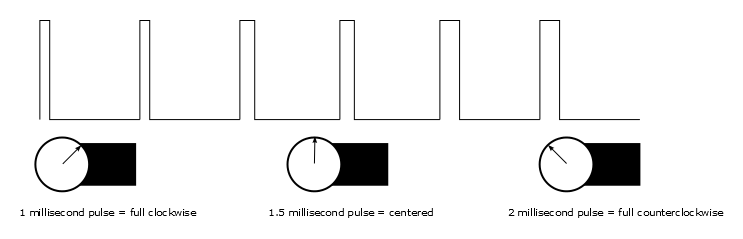
  
**Ассортимент шаговых сервоприводов**(Взято из [SparkFun Electronics](https://learn.sparkfun.com/tutorials/hobby-servo-tutorial), предоставлено по [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/))

Шаговые сервоприводы - это не простые приводы, как соленоиды или шаговые двигатели, и не электромеханические машины, как реле. Сервоприводы - это полноценные встраиваемые устройства. Они содержат плату микроконтроллера, драйвер силового двигателя, двигатель постоянного тока, редуктор, выходной вал и потенциометр. Редуктор прикреплен к двигателю, снижая его скорость, чтобы выходной вал вращался в механически ограниченном диапазоне вращения. Этот диапазон может составлять от 90 до 270 градусов. Этот вал обычно прикреплен к потенциометру, который действует как прибор обратной связи, посылающий напряжение на микроконтроллер. Микроконтроллер реализует систему управления с замкнутым контуром, перемещая двигатель, чтобы выходной вал достиг желаемого угла.

Попробуйте определить эти части на следующем рисунке:

  
**Разобранный шаговый сервопривод**  
(Взято с сайта [SparkFun Electronics](https://learn.sparkfun.com/tutorials/hobby-servo-tutorial), предоставлено под [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/))

Желаемый угол задается пользователем через линию ввода. Мы сообщаем сервоприводу желаемый угол с помощью прямоугольного сигнала частотой 50 Гц (период 20 мс). Ширина импульса этого сигнала кодирует желаемый угол. Обычно длительность импульса 1 мс кодирует один конец вращения, а 2 мс - другой конец. Середина обычно кодируется как 1,5 мс.

  
**Ширина импульсов для обоих концов и центра вращения шагового сервопривода**   
(Получено из [SparkFun Electronics](https://learn.sparkfun.com/tutorials/hobby-servo-tutorial), предоставлено под [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/))

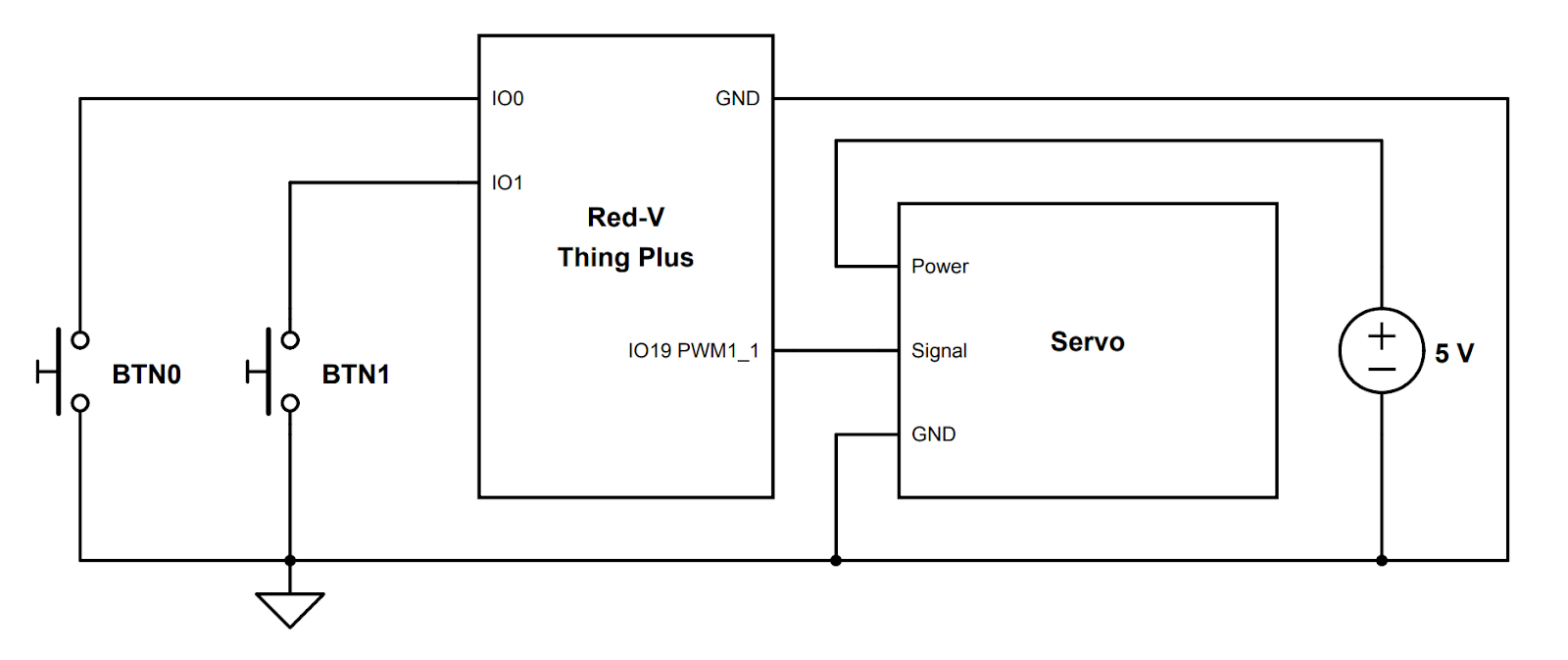
Чтобы узнать больше о сервоприводах, вы можете ознакомиться с этим [учебником от SparkFun Electronics.](https://learn.sparkfun.com/tutorials/hobby-servo-tutorial)

## Приложение для контроля сервопривода

Теперь мы будем управлять шаговым сервоприводом!

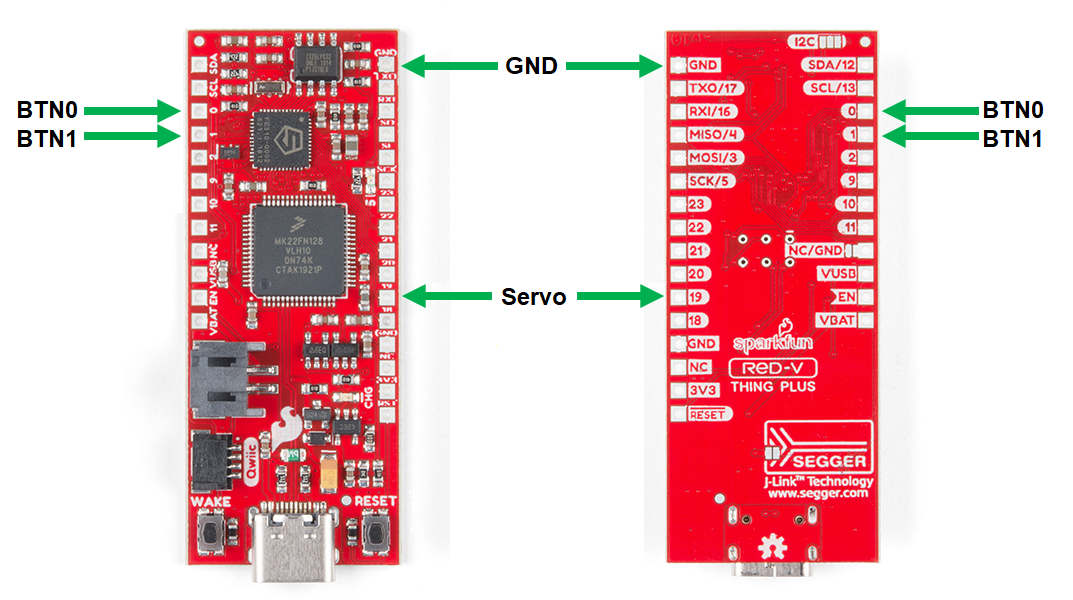
В этой демонстрации будет использоваться почти то же оборудование, что и в приложении яркости светодиодов, но вместо светодиода мы будем использовать сервопривод.

Вот принципиальная схема:

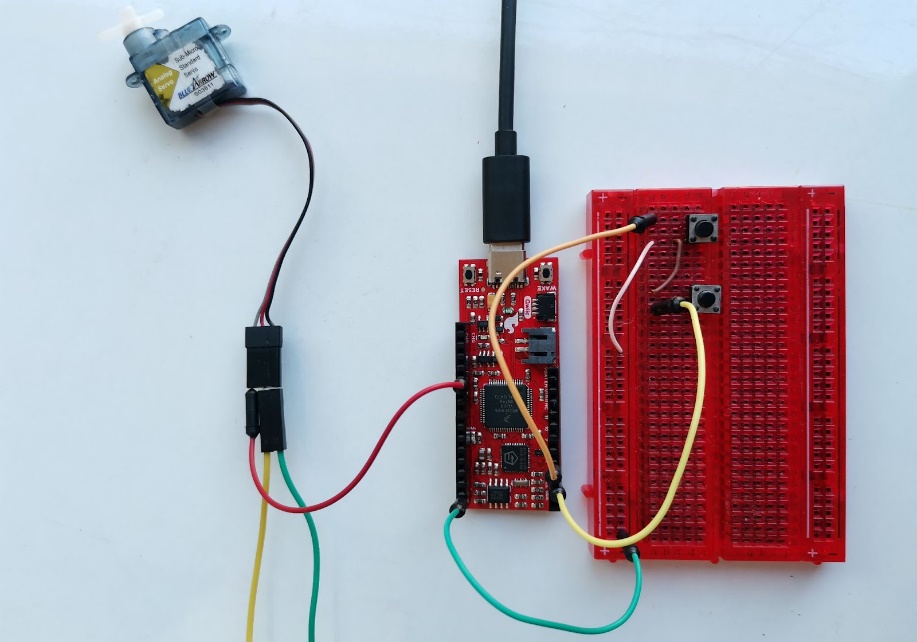
  
**Принципиальная схема для управления шаговым сервоприводом**

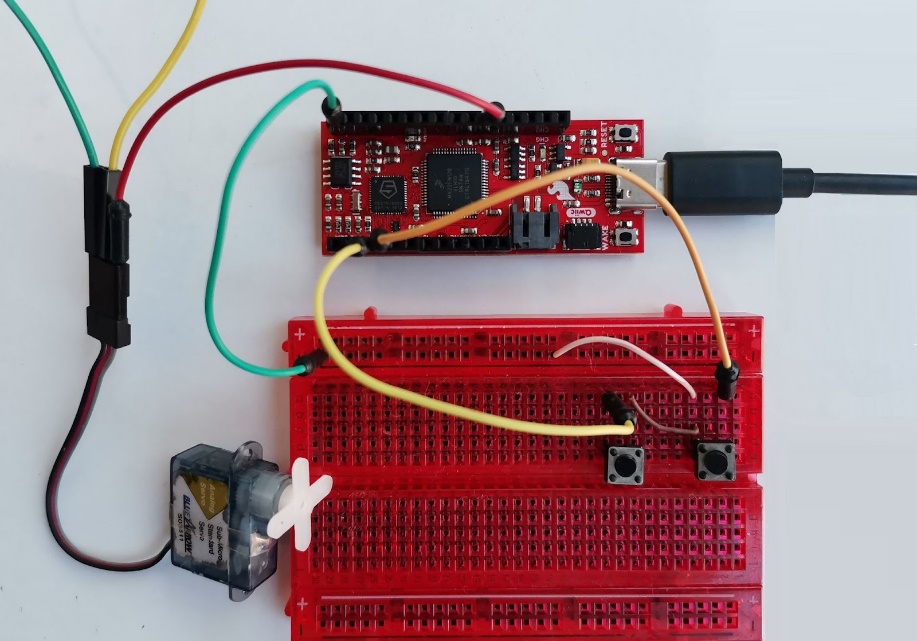
Кнопка 0 повернет вал сервопривода против часовой стрелки на шаг, а кнопка 1 - по часовой стрелке на шаг.

Это те же самые контакты ввода/вывода, которые мы использовали для подключения кнопок и внешнего светодиода. Просто подключите линию светодиода к линии входного сигнала сервопривода.

  
**Виды сверху и снизу Red-V Thing Plus, показывающие контакты, которые мы будем использовать в демонстрационном приложении для сервопривода**(взято из [SparkFun Electronics](https://www.sparkfun.com/products/15799), предоставлено по лицензии [CC BY 2.0](https://creativecommons.org/licenses/by/2.0/))

Здесь представлены две фотографии фактических аппаратных соединений. Пожалуйста, изучите документацию на ваш собственный сервопривод и **НЕ** следуйте схеме подключения сервопривода, которую вы видите на следующих фотографиях, поскольку разные сервоприводы имеют разную схему подключения.



  
**Аппаратные соединения для демонстрационного приложения сервопривода**

## Предупреждения о работе с сервоприводами

Среди всех устройств, используемых в данном курсе, сервопривод, безусловно, является тем, который потребляет больше всего энергии. Именно поэтому вы должны обратить особое внимание на следующие рекомендации:

Сервопривод имеет 3 провода: Питание, Сигнал и Заземление. Не существует стандарта на порядок или цветовую маркировку этих проводов, поэтому обязательно ознакомьтесь с документацией на сервопривод, чтобы сделать правильные подключения.

1. **НЕ** следуйте схеме подключения на фотографиях сервопривода, который вы только что видели. Эта схема подходит для конкретного сервопривода, который был использован в этом приложении.
2. **НЕ** пробуйте несколько соединений, пока не найдете подходящее. Если вы подключите линии питания в обратном порядке, вы можете повредить сервопривод, источник питания или и то, и другое.
3. Убедитесь, что линия заземления Red-V Thing Plus и линия заземления сервопривода соединены вместе.
4. Обязательно обеспечьте соответствующий источник питания для вашего сервопривода. Прочтите примечание ниже.
5. Трижды проверьте подключения и убедитесь, что все в порядке.
6. Мы выбрали уменьшенный сервопривод, чтобы снизить потребление энергии, и вы тоже должны это сделать. Чем больше сервопривод, тем больше энергии он будет требовать, и тем больший блок питания вам понадобится.

Вспомните рекомендации по питанию сервопривода в списке деталей:

Напряжение питания сервопривода должно соответствовать напряжению сервопривода, обычно 5 В. Вы можете подойти к этому творчески:

* Подойдет выход 5 В Arduino UNO (если он у вас есть). Именно этот источник питания используется в демонстрационном примере.
* [Кабель Hydra](https://www.sparkfun.com/products/11579) также является одним из вариантов. Вы можете получать питание от USB-порта (не рекомендуется) или [настенного USB-адаптера](https://www.sparkfun.com/products/16893), который, вероятно, у вас уже есть (например, зарядное устройство для смартфона).
* Обратите внимание, что использование для этого выхода 3V3 устройства Red-V Thing Plus может повредить плату микроконтроллера.

## Код приложения для демонстрации сервопривода

В нашем приложении есть все необходимое для управления сервоприводом. Оно нуждается лишь в нескольких настройках:

* Частота должна изменяться от 1 кГц до 50 Гц.
* Ширина импульса должна изменяться не в диапазоне от 0% до 100%, а в диапазоне от 1 мс до 2 мс. Теперь наше приложение управляет ***рабочим циклом***, а не шириной импульса. Таким образом, при постоянном периоде 20 мс требуемый диапазон ширины импульса для сервопривода соответствует диапазону рабочего цикла от 5% до 10%. Мы можем сделать это с шагом в 1%. На самом деле, мы и должны так делать, потому что параметр рабочего цикла является целым числом.
* Пусковой рабочий цикл должен измениться с 50% до 7%.

Учитывая это, давайте взглянем на код.

Учитывая, что приложение LED требует нескольких доработок, здесь представлена модифицированная версия с использованием директив для условного включения кода (#ifdef). Макрос SERVO определяет, будет ли приложение управлять светодиодом или шаговым сервоприводом. 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**. Не стесняйтесь попробовать приложение, чтобы убедиться, что оно работает правильно.

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
     Red-V Thing Plus PWM\_Demo,  
     by Eduardo Corpeño**

**Controlling the brightness of the built-in LED with two external buttons using PWM.  
     Alternatively, a hobby servo can be controlled with those buttons.  
     The servo must receive a 50Hz signal, with pulse widths between 1ms and 2ms.  
     Date: June 20, 2022  
     Developed using Freedom Studio v4.18.0.2021-04-1 on Windows 10  
     LICENSE: This code is released under the CC BY-SA 4.0 License  
     (https://creativecommons.org/licenses/by-sa/4.0/)**

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

**#include <metal/pwm.h>  
#include <metal/gpio.h>  
#include <stdint.h>**

**// Откомментируйте следующее определение для применения сервопривода.  
// Оставьте его закомментированным для приложения яркости светодиодов.  
#define SERVO**

**// Наши собственные макросы, определенные для эффективности или потому, что   
// Freedom Metal Library не обеспечила их функциональность.**

**#define Red\_V\_enable\_output(x)  \*((uint32\_t \*) 0x10012008) |= (1<<(x))  
#define Red\_V\_set\_pin(x)        \*((uint32\_t \*) 0x1001200C) |= (1<<(x))  
#define Red\_V\_clear\_pin(x)      \*((uint32\_t \*) 0x1001200C) &= ~(1<<(x))  
#define Red\_V\_read\_pin(x)       (\*((uint32\_t \*) 0x10012000) & (1<<(x)))  
#define Red\_V\_enable\_pullup(x)  \*((uint32\_t \*) 0x10012010) |= (1<<(x))  
#define Red\_V\_enable\_DS(x)      \*((uint32\_t \*) 0x10012014) |= (1<<(x))  
#define Red\_V\_enable\_XOR(x)     \*((uint32\_t \*) 0x10012040) |= (1<<(x))**

**// 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) {**

**#ifdef SERVO  
  unsigned int dc = 7;          // Примерно центральное положение сервоаривода  
#else  
  unsigned int dc = 50;         // Рабочий цикл по умолчанию 50%  
#endif**

**struct metal\_gpio \*gpio\_0;    // Экземпляр GPIO  
  struct metal\_pwm \*pwm\_1;      // Экземпляр ШИМ**

**// Конфигурация GPIO для встроенного светодиода и кнопок  
  gpio\_0 = metal\_gpio\_get\_device(0);     // Получение экземпляра устройства GPIO  
  // LED  
  metal\_gpio\_disable\_input(gpio\_0, 5);   // Отключить ввод 5  
  Red\_V\_enable\_output(5);                // Отключить вывод 5  
  metal\_gpio\_disable\_pinmux(gpio\_0, 5);  // Отключить альтернативные функции для пина 5  
  // Нажатие кнопок  
  metal\_gpio\_enable\_input(gpio\_0, 0);    // Включить ввод 0  
  metal\_gpio\_enable\_input(gpio\_0, 1);    // Включить ввод 1  
  metal\_gpio\_disable\_output(gpio\_0, 0);  // Выключить вывод 0  
  metal\_gpio\_disable\_output(gpio\_0, 1);  // Выключить вывод 1  
  metal\_gpio\_disable\_pinmux(gpio\_0, 0);  // Отключить альтернативные функции для пина 0  
  metal\_gpio\_disable\_pinmux(gpio\_0, 1);  // Отключить альтернативные функции для пина 1  
  Red\_V\_enable\_pullup(0);                // Включить подтягивающий резистор для пина 0  
  Red\_V\_enable\_pullup(1);                // Включить подтягивающий резистор для пина 1**

**//** **Конфигурация ШИМ. Мы будем использовать PWM1\_1, который использует GPIO0\_19.  
#ifndef SERVO  
  Red\_V\_enable\_DS(19);  // Высокий уровень сигнала на входе  
#endif  
  Red\_V\_enable\_XOR(19); // Инвертируем выход для активно-высокого ШИМ**

**pwm\_1 = metal\_pwm\_get\_device(1);       // Получение экземпляра ШИМ-устройства  
  while (pwm\_1 == NULL);                 // Остановка. Что-то пошло не так.  
  metal\_pwm\_enable(pwm\_1);               // Включить ШИМ1  
 metal\_gpio\_disable\_pinmux(gpio\_0, 20); // Отключить альтернативные функции для вывода 20  
 metal\_gpio\_disable\_pinmux(gpio\_0, 21); // Отключить альтернативные функции для вывода 21  
 metal\_gpio\_disable\_pinmux(gpio\_0, 22); // Отключить альтернативные функции для вывода 22    
 // Установка частоты ШИМ на 1кГц или 50Гц  
#ifdef SERVO  
  metal\_pwm\_set\_freq(pwm\_1, 1, 50);      // Установка частоты ШИМ1\_1 на 50Гц для сервопривода  
#else  
  metal\_pwm\_set\_freq(pwm\_1, 1, 1000);    // Установка частоты ШИМ1\_1 на 1 кГц для светодиода  
#endif**

**metal\_pwm\_set\_duty(pwm\_1, 1, dc, METAL\_PWM\_PHASE\_CORRECT\_DISABLE); // Установить рабочий цикл для PWM1\_1  
  metal\_pwm\_trigger(pwm\_1, 1, METAL\_PWM\_CONTINUOUS);                 // Запуск в непрерывном режиме mode**

**while(1){  
          if(Red\_V\_read\_pin(0) == 0){    // Считываем входной пин 0  
            Red\_V\_set\_pin(5);            // Включаем светодиод  
  #ifdef SERVO  
            dc = (dc>5)? dc - 1 : dc;    // Шаг 1% для серво  
  #else  
            dc = (dc>0)? dc - 10 : dc;   // Шаг 10% для яркости  
  #endif  
            metal\_pwm\_set\_duty(pwm\_1, 1, dc,**

**METAL\_PWM\_PHASE\_CORRECT\_DISABLE);  
            delay(300);                   // Избегаем дребезг  
            Red\_V\_clear\_pin(5);           // Выключаем светодиод  
            while(Red\_V\_read\_pin(0) == 0);// Ждем, пока кнопка будет отпущена  
            delay(300);                   // Избегаем дребезг  
          }  
          else if(Red\_V\_read\_pin(1) == 0){// Считываем входной пин 1  
            Red\_V\_set\_pin(5);             // Включаем светодиод  
  #ifdef SERVO  
            dc = (dc<10)? dc + 1 : dc;    // Шаг 1% для серво  
  #else  
            dc = (dc<100)? dc + 10 : dc;  // Шаг 10% для яркости  
  #endif  
            metal\_pwm\_set\_duty(pwm\_1, 1, dc, METAL\_PWM\_PHASE\_CORRECT\_DISABLE);  
            delay(300);                   // Избегаем дребезг  
            Red\_V\_clear\_pin(5);           // Выключаем светодиод  
            while(Red\_V\_read\_pin(1) == 0);// Ждем, пока кнопка будет отпущена  
            delay(300);                   // Избегаем дребезг  
          }  
   }  
   return 0; // Недостижимый код возврата  
}**

## Итоги главы

Поздравляю! Вы наверняка узнали много нового только в этой главе. Давайте вспомним, что мы сделали в этой главе:

* Мы рассмотрели, что такое широтно-импульсная модуляция и как ее можно использовать для подачи энергии на нагрузку.
* Мы обсудили, как микроконтроллеры обычно генерируют сигналы ШИМ. Мы изучили руководство FE310, чтобы узнать об аппаратном обеспечении ШИМ и регистрах конфигурации.
* Мы изучили руководство FE310, чтобы узнать об аппаратном обеспечении ШИМ и регистрах конфигурации.
* Мы изучили дребезг кнопок и несколько методов борьбы с ним.
* Мы создали приложение для управления яркостью светодиодов.
* Мы узнали, как подключить шаговый сервопривод к вашей плате.
* Мы создали приложение для управления шаговым сервоприводом.

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