# Оглавление

**Предисловие от издательства** ..... 11
**Об авторе** ..... 12
**О рецензентах** ..... 13
**Предисловие** ..... 14
Для кого предназначена книга ..... 14
О чем эта книга ..... 15
Максимальная отдача от книги ..... 15
Загрузите файлы с примерами кода ..... 16
Код в действии ..... 16
Загрузите цветные изображения ..... 16
Используемые соглашения ..... 16
Обратная связь ..... 17
Отзывы ..... 18
**Глава 1. Начало работы с TinyGo** ..... 19
Технические требования ..... 19
Знакомимся с языком TinyGo ..... 20
Как работает TinyGo ..... 20
Сравнение TinyGo с Go ..... 21
Поддерживаемые языковые функции ..... 22
Поддерживаемые стандартные пакеты ..... 22
Операции volatile ..... 23
Встроенный ассемблер ..... 23
Распределение памяти ..... 23
Сборка мусора ..... 23
Установка TinyGo ..... 24
Установка в Linux ..... 24
Установка в Windows ..... 25
Установка на macOS ..... 26
Установка в Docker ..... 27
Настройка интеграции IDE с TinyGo ..... 27
Интеграция в VS Code ..... 28
Общая интеграция с IDE ..... 31
Настройка Goland ..... 32
Интеграция любого редактора ..... 33
Arduino UNO ..... 34
Знакомство с техническими характеристиками ..... 34
Изучение распиновки ..... 35
Проверяем работу программы Hello world в устройстве ..... 36
Подготовка ..... 36
Подготовка проекта ..... 36
Программирование микроконтроллера ..... 36

Прошивка программы ..... 38
Использование игровой площадки TinyGo ..... 39
Резюме ..... 39
Вопросы ..... 39

**Глава 2. Построение системы управления светофорами ..... 40**

Технические требования ..... 40
Освещение внешним светодиодом ..... 41
Использование макетных плат ..... 41
Знакомство со структурой светодиодов ..... 42
Использование портов GPIO ..... 42
Сборка электрической схемы ..... 43
Написание кода ..... 44
Управление светодиодом с помощью кнопки ..... 45
Построение электрической схемы ..... 45
Программирование логики ..... 47
Функция main ..... 47
Подтягивающий резистор ..... 49
Создание светофора ..... 49
Построение электрической схемы ..... 49
Создание структуры папок ..... 51
Написание логики ..... 51
Создание светофора со световыми индикаторами для пешеходов ..... 53
Сборка схемы ..... 53
Написание логики ..... 55
Реализация основной логики ..... 59
Резюме ..... 61
Вопросы ..... 61
Дополнительное чтение ..... 61

**Глава 3. Создание кодового замка с использованием клавиатуры ..... 62**

Технические требования ..... 63
Запись в последовательный порт ..... 63
Мониторинг последовательного порта ..... 64
Отслеживание ввода с клавиатуры ..... 67
Создание электрической схемы ..... 67
Понимание работы клавиатуры 4×4 ..... 69
Написание драйвера ..... 70
Переменные Driver ..... 70
Configure ..... 71
GetIndices ..... 72
GetKey ..... 74
main ..... 74
Поиск драйверов для TinyGo ..... 76
Помощь в поиске и создания драйверов для TinyGo ..... 76

Управление сервомотором ..... 77
Изучение сервомоторов SG90 ..... 77
Построение схемы ..... 77
Написание логики сервоуправления ..... 78
Создание кодового замка с помощью клавиатуры ..... 83
Построение схемы ..... 84
Написание логики ..... 85
Резюме ..... 89
Вопросы ..... 90

**Глава 4. Создание системы полива растений ..... 91**

Технические требования ..... 91
Считывание данных датчика влажности почвы ..... 92
Сборка схемы ..... 92
Нахождение пороговых значений ..... 93
Понимание АЦП в TinyGo ..... 97
Написание библиотеки для датчика ..... 98
Тестирование библиотеки ..... 102
Считывание данных датчика уровня воды ..... 104
Написание библиотеки датчиков уровня воды ..... 105
Тестирование библиотеки ..... 107
Управление зуммером ..... 109
Написание библиотеки для зуммеров ..... 110
Управление насосом ..... 112
Работа с реле ..... 112
Написание библиотеки для насоса ..... 114
Полив ваших растений ..... 116
Резюме ..... 119
Вопросы ..... 119
Рекомендации ..... 119

**Глава 5. Создание таймера для бесконтактного мытья рук ..... 120**

Технические требования ..... 120
Разбираем функционал Arduino Nano 33 IoT ..... 121
Установка Bossa ..... 123
Учимся измерять расстояния ..... 124
Разбираемся в датчике HC-SR04 ..... 124
Сборка схемы ..... 126
Написание библиотеки ..... 127
Модульное тестирование в TinyGo ..... 131
Написание примера программы для библиотеки ..... 135
Использование четырехзначных семисегментных дисплеев ..... 136
Использование MAX7219 ..... 137
Написание библиотеки для управления MAX7219 ..... 140
Написание библиотеки для управления дисплеем hs42561k ..... 144
Собирая все это вместе ..... 150

Резюме ..... 154
Вопросы ..... 155

**Глава 6. Построение дисплеев для связи с использованием интерфейса I2C и SPI ..... 156**

Технические требования ..... 156
Изучение драйверов TinyGo ..... 157
Отображение текста на ЖК-дисплее HD44780 16×2 ..... 158
Построение схемы ..... 160
Знакомимся с I2C ..... 161
Написание кода ..... 162
Отображение пользовательского ввода на дисплее ..... 164
Создание интерфейса командной строки ..... 167
Понимание SPI ..... 172
Отображение простой игры ..... 173
Построение схемы ..... 174
Использование дисплея ST7735 ..... 175
Разработка игры ..... 180
Резюме ..... 189
Вопросы ..... 189

**Глава 7. Мониторинг погоды на панели управления Wasm TinyGo ..... 190**

Технические требования ..... 190
Создание метеостанции ..... 191
Сборка схемы ..... 191
Программирование метеостанции ..... 193
Расчет предупреждений о погоде ..... 196
Отправка сообщений MQTT брокеру ..... 200
Реализация пакета Wi-Fi ..... 200
Реализация вариации универсальности клиента MQTT ..... 204
Изучение MQTT ..... 205
Внедрение метеостанции ..... 210
Представляем Wasm ..... 219
Отображение данных датчиков и предупреждений о погоде на странице Wasm ..... 220
Обслуживание заявки ..... 220
Внедрение приложения «Погода» ..... 221
Резюме ..... 232
Вопросы ..... 233

**Глава 8. Автоматизация и мониторинг вашего дома с помощью панели управления TinyGo Wasm ..... 234**

Технические требования ..... 235
Создание панели управления домашней автоматизацией ..... 235
Создание универсального компонента MQTT ..... 235

Настройка кода создания экземпляра Wasm....238
Создание HTML-шаблона....239
Реализация логики представления входа в систему....240
Реализация компонента панели мониторинга....245
Реализация основной логики....249
Обслуживание приложения....251
Создание клиента домашней автоматизации....253
Настройка схемы....253
Реализация логики....254
Запрос данных с микроконтроллера....257
Проверка других идей реализации....263
Резюме....264
Вопросы....264
**Приложение «Go»ing Ahead....265**
Блокирование горутины....265
Чтение с канала....265
Инструкция select....265
Задержка по времени – блокирующий вызов....266
Поиск распределений кучи (объема памяти)....267
**Замечания....269**
**Послесловие....271**
**Предметный указатель....272**

# Предисловие от издательства

## Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте **www.dmkpress.com**, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

## Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

## Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства ДМК Пресс и Packt Publishing очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты **dmkpress@gmail.com**.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

# Об авторе

**Тобиас Тил** – технический руководитель и разработчик в немецком стартапе FinTech fino, а также ведущий инженер-программист в стартапе RegTech и ClariLab. Является архитектором программного обеспечения и экспертом в языках Go, TinyGo и параллельно в C# и Java. Прошел сертификацию по iSAQB.

Тил является активным участником сообщества на StackOverflow и входит в число 10 % самых ведущих его представителей по C# и Unity3D и 20 % – по .NET, Go и Visual Studio.

В свободное от работы время его можно найти разрабатывающим игры, в основном на Game Jame, таких как Ludum Dare Jam. Задача данных соревнований – это разработать игру с нуля в течении 72 ч. Будучи активным спикером на технических дискуссиях и участником многочисленных хакатонов, Тил любит делиться своими знаниями в области разработки программного обеспечения с другими энтузиастами.

*Я хочу поблагодарить свою команду в ClariLab за то, что она предоставила мне так много выходных дней, чтобы закончить эту книгу. Также выражаю особую благодарность Йоханнесу Колате (Johannes Kolata) за столь ценный вклад в подготовку книги.*

# О рецензентах

**Энрико фон Отте** (Enrico von Otte) начал изучать программирование в 11-летнем возрасте на старом добром Commodore C64. Позже он кодировал на Amiga 2000, а в середине 90-х перешел на персональный компьютер. Он работает в области профессиональной разработки программного обеспечения с 2005 года. После длительного опыта разработки аппаратного обеспечения, близкого к программному обеспечению на C и C++, он перешел в мир C# в 2008 году.

До 2015 года Энрико разрабатывал GIS-системы и системы управления документами. Теперь он профессиональный архитектор программного обеспечения с сильной привязанностью к self-made-коду и девизом «Сборка не разработка».

**Йоханнес Колата** (Johannes Kolata) начал программировать 8 лет назад, работая над системами управления документами. С 2018 года Колата работает инженером по программному обеспечению в немецкой компании FinTech fino, которая наиболее известна созданием первого цифрового коммутатора банковских счетов. Попутно он стал экспертом в разработке на C#, Java, Golang и C++, а также получил представление о множестве других языков программирования. Помимо работы в индустрии FinTech, он страстный энтузиаст открытого исходного кода и 3D-печати и с нуля создает системы домашней автоматизации, которые характеризуются взаимодействием CAN (Controller Area Network), затрагивающим API (Application Programming Interface) и панель инструментов Angular. Когда Колата не работает над проектами, он участвует в Game Jame и хакатонах.

# Предисловие

Если языки JavaScript или C# могут применяться для программирования микроконтроллеров, то Go может сделать это еще лучше. В то время как стандартный Go производит огромные бинарные коды, TinyGo производит двоичные коды, которые подходят для самых маленьких устройств. Почему же вы должны выбрать Go для микроконтроллера и программирование Wasm (сокращение от WebAssembly)? Мои любимые причины в том, что язык Go легко выучить, легко читать и легко на нем писать. Кроме того, Go поставляется с мощной стандартной библиотекой, которая легко привязывается и имеет широкие возможности параллелизма.

Если вы любите язык программирования Go, то эта книга для вас. После прочтения у вас будут все инструменты и знания, необходимые для создания всех проектов, связанных с микроконтроллерами, о которых вы когда-либо мечтали. Кроме того, в качестве дополнительного преимущества вы сможете создавать информационные панели и приложения для управления домом с помощью Wasm для ваших проектов домашней автоматизации. Всего этого можно достичь с помощью TinyGo.

Если вы никогда раньше не работали с микроконтроллерами, вот несколько причин, по которым это следует попробовать:

- если вы уже являетесь программистом, здорово видеть, как ваш код влияет на работу реальных устройств. Это действительно прекрасное чувство – завершить проект и, наконец, увидеть, как вращается двигатель, мигает светодиод, раздается звуковой сигнал и т. д.;
- вы будете постоянно узнавать что-то новое и глубже понимать, как работают компьютеры в целом, поскольку познакомитесь с различными типами шин компьютерных систем, протоколов, аппаратных интерфейсов и многим другим;
- возможности практически безграничны, когда вы играете с микроконтроллерами. Вы не привязаны к тому, что доступно на рынке, так как можете просто построить все самостоятельно;
- можно научиться писать небольшие эффективные программы, чтобы сообщить микроконтроллеру, чего вы от него хотите. В целом это также поможет вам улучшить навыки разработчика;
- вы можете вносить свой вклад в интересные проекты и вступать в контакт с отличными сообществами единомышленников.

## Для кого предназначена книга

Если вы разработчик Go, который хочет программировать маломощные устройства и оборудование, такие как Arduino UNO и Arduino Nano

IoT 33, или который хочет расширить свои знания об использовании Go с WebAssembly для программирования на данном языке в браузере, тогда эта книга для вас. Программисты-любители языка Go, которые заинтересованы в том, чтобы узнать больше о TinyGo, работая над проектами DIY, также найдут это практическое руководство полезным.

## О чем эта книга

*Глава 1*, «Начало работы с TinyGo», показывает, как настроить TinyGo и скомпилировать свою первую программу!

*В главе 2*, «Построение системы управления светофорами», вы создадите систему управления светофорами, включая световые индикаторы для пешеходов и кнопку управления; кроме этого, вы изучите, как использовать функции Gorutine (горутины) в TinyGo.

*В главе 3*, «Создание кодового замка с использованием клавиатуры», рассматривается использование клавиатуры 4×4 и сервомотора для создания замка, который открывается при вводе правильного пароля.

*Глава 4*, «Создание системы полива растений», объясняет, как использовать различные типы датчиков, чтобы построить автоматическую систему полива растений. Вам больше не придется поливать растения вручную!

*Глава 5*, «Создание таймера для безконтактного мытья рук», посвящена использованию четырехзначного семисегментного дисплея и ультразвукового датчика расстояния для распознавания движения недалеко расположенного объекта, чтобы запустить таймер, который будет регулировать длительность процесса мытья рук.

*Глава 6*, «Подключение дисплеев для связи с использованием интерфейсов I2C и SPI», объясняет концепции **межинтегрированной схемы** (I2C) и **последовательного периферийного интерфейса** (SPI) на примере использования дисплеев, которые взаимодействуют с помощью шин I2C и SPI. К концу главы вы узнаете, как использовать различные типы дисплеев в TinyGo.

*Глава 7*, «Мониторинг погоды на панели управления Wasm TinyGo», посвящена созданию и обслуживанию приложения Wasm, которое отображает данные датчиков, отправленные с Arduino Nano 33 IoT по Wi-Fi.

*Глава 8*, «Автоматизация и мониторинг вашего дома с помощью панели управления TinyGo Wasm», объясняет, как управлять и контролировать устройства в вашем доме с помощью панели управления Wasm.

## Максимальная отдача от книги

Все примеры кода были протестированы с Go 1.16.2 в Ubuntu, но они также будут работать с будущими выпусками Go и в других операционных системах. Код на Visual Studio использовался в качестве редактора на протяжении всей книги, но можно использовать любой другой редактор.

| Программное и аппаратное обеспечение, описанное в книге | Требования к операционной системе |
|---|---|
| Go 1.15.x или новее | Windows, macOS или Linux |
| Visual Studio Code | Windows, macOS или Linux |

*Если вы используете цифровую версию этой книги, мы советуем вам писать код самостоятельно или получить доступ к коду через репозиторий GitHub (ссылка доступна в следующем разделе). Это поможет избежать любых потенциальных ошибок, связанных с копированием и вставкой кода.*

Я бы хотел увидеть проекты, которые вы создадите после прочтения этой книги в социальных сетях. Не стесняйтесь отмечать меня в Twitter, используя следующий тег: **@Nooby_Games**.

## Загрузите файлы с примерами кода

Вы можете скачать примеры файлов кода для этой книги с GitHub по адресу **https://github.com/PacktPublishing/Creative-DIY-Microcontroller-Projects-with-TinyGo-and-WebAssembly**. В случае обновления кода он будет обновлен в существующем репозитории GitHub.

У нас также есть другие пакеты с кодами из нашего богатого каталога книг и видео, доступных по адресу **https://github.com/PacktPublishing/**. Посмотрите их!

## Код в действии

Видеоролики с рабочими кодами для этой книги можно посмотреть по адресу **https://bit.ly/3cYZOh4**.

## Загрузите цветные изображения

Мы также предоставляем PDF-файл с цветными изображениями скриншотов/диаграмм, используемых в этой книге. Вы можете скачать их здесь: **https://static.packt-cdn.com/downloads/9781800560208_ColorImages.pdf**.

## Используемые соглашения

В этой книге используется ряд текстовых соглашений. Код в тексте указывает кодовые слова в тексте, имена таблиц базы данных, имена папок, имена файлов, расширения файлов, пути, фиктивные URL-адреса, вводимые пользователем, и дескрипторы Twitter. Вот пример: «Когда мы получаем начало команды, мы добавляем все последующие символы в *commandBuffer*».

Блок кода задается следующим образом:

```
data, err := uart.ReadByte()
if err != nil {
```

```
    println(err.Error())
}
```

Когда мы хотим привлечь ваше внимание к определенной части блока кода, соответствующие строки или элементы выделены жирным шрифтом:

```
func main() {
    blocker := make(chan bool, 1)
    <-blocker
    println("это никогда не печатается")
}
```

Любой ввод или вывод командной строки записывается следующим образом:

```
tinygo flash -target=arduino-nano33 Chapter06/tinygame/main.go
```

**Жирный шрифт** указывает на новый термин, важное слово или слова, которые вы видите на экране. Например, слова в меню или диалоговых окнах отображаются в тексте следующим образом. Пример: «Значение довольно стабильно на уровне **37888**».

> **Советы или важные примечания**
>
> Выглядят так.

## Обратная связь

Отзывы наших читателей всегда приветствуются.

**Общая обратная связь:** если у вас есть вопросы по какому-либо аспекту этой книги, укажите название книги в теме своего сообщения и напишите нам по адресу **customercare@packtpub.com**.

**Ошибки:** хотя мы позаботились о том, чтобы обеспечить точность нашего контента, ошибки все же случаются. При обнаружении ошибки в этой книге мы были бы признательны, если бы вы сообщили нам об этом. Пожалуйста, посетите **www.packtpub.com/support/errata**, выберите свою книгу, нажмите на ссылку «форма отправки с ошибками» и введите данные.

**Пиратство:** если вы столкнетесь с любыми незаконными копиями наших работ в любой форме в интернете, мы будем признательны, если вы сообщите нам адрес местонахождения или название веб-сайта. Пожалуйста, свяжитесь с нами по адресу **copyright@packt.com** со ссылкой на материал.

**Если вы заинтересованы в том, чтобы стать автором:** при наличии темы, в которой вы разбираетесь, и, если вы заинтересованы в написании или участии в книге, пожалуйста, посетите **https://authors.packtpub.com/**.

## Отзывы

Пожалуйста, оставьте отзыв. После того как вы прочитали и использовали эту книгу, почему бы не оставить отзыв на сайте, на котором вы ее купили? Потенциальные читатели смогут увидеть и использовать ваше непредвзятое мнение для принятия решений о покупке. Мы в Packt сможем понять, что вы думаете о наших продуктах, а наши авторы смогут увидеть ваши отзывы о своей книге. Спасибо!

Для получения дополнительной информации о Packt, пожалуйста, посетите **https://www.packtpub.com/**.

# Глава 1

# Начало работы с TinyGo

На мой взгляд, язык Go легко выучить, легко читать и легко, писать. Язык не перегружен причудливыми функциями, а скорее ориентирован на лаконичность. Встроенный параллелизм, быстрое время компиляции, высокая производительность выполнения и богатые стандартные библиотеки создают отличное сочетание для этого потрясающего языка. Вот почему я хочу отправиться с вами в путешествие от очень простых высокоуровневых программ Go до глубин микроконтроллеров, использующих всю мощь TinyGo.

В этой главе мы собираемся настроить TinyGo и узнать, как заставить наш готовый код работать в VS Code и в других редакторах. После того как это будет сделано, мы рассмотрим плату Arduino UNO и его технические характеристики. Затем необходимо сравнить TinyGo с Go и поговорить о том, что делает TinyGo особенным по сравнению с другими языками для микроконтроллеров. В конце этой главы мы напишем, скомпилируем, развернем и запустим нашу первую программу на языке TinyGo для реального микроконтроллера. Рассмотрев все эти темы, вы узнаете, как писать, создавать и запускать программы на микроконтроллерах.

В этой главе рассмотрим следующие основные темы:

- знакомимство с языком TinyGo;
- настройку TinyGo;
- настройку интеграции IDE с TinyGo;
- Arduino UNO;
- тестирование устройств программой Hello World.

## Технические требования

Для того чтобы продолжить, вам необходимо иметь следующее:

- Go должен быть установлен;
- GOPATH должен быть настроен;
- Git должен быть установлен;
- Arduino Uno – предпочтительно версия Rev3, но можно использовать другие Arduino-платы.

Вы можете найти все примеры кода из этой главы в следующем репозитории GitHub: **https://github.com/PacktPublishing/Creative-DIYMicrocontroller-Projects-with-TinyGo-and-WebAssembly/tree/master/Chapter01**. Видео с рабочим кодом для этой главы можно найти здесь: **https://bit.ly/3mLFCCJ**.

# Знакомимся с языком TinyGo

TinyGo – это отдельно написанный компилятор со своей собственной реализацией среды выполнения. Он предназначен для программирования микроконтроллеров, веб-сборки (WASM) и инструментов CLI. TinyGo активно использует инфраструктуру LLVM для оптимизации и компиляции бинарного кода, понятного микроконтроллеру.

Первый релиз TinyGo (v0.1) был опубликован 1 февраля 2019 года на GitHub. С тех пор проект быстро внедрил множество функций и никогда не прекращал добавлять поддержку большего количества микроконтроллеров, датчиков, дисплеев и других устройств.

2 февраля 2020 года TinyGo объявил, что теперь он официально является проектом, спонсируемым Google. Это был большой шаг в развитии проекта.

## Как работает TinyGo

Компилятор TinyGo использует набор шагов, отличный от других языков, для преобразования исходного кода Go в машинный код. Однако мы не будем вдаваться в подробности, но давайте взглянем на усеченную структуру компилятора.

1. Мы пишем исходный код на Go.
2. Этот исходный код переводится в Go SSA (**Static Single Assignment**).
3. SSA Go преобразуется в LLVM IR (Low Level Virtual Machine) с помощью пакета компилятора TinyGo.
4. Код инициализации в LLVRM IR интерпретируется пакетами взаимодействия TinyGo. Этот шаг оптимизирует глобальные значения, константы и многое другое.
5. Затем результат оптимизируется с помощью некоторых проходов оптимизации LLVM (например, оптимизации *string* в *[] byte*).
6. Затем результат снова оптимизируется модификатором LLVM.
7. Далее, некоторые исправления выполняются пакетом компилятора.
8. И в качестве последнего шага LLVM создает машинный код.

Если сейчас это звучит сложно, не волнуйтесь – нам не нужно заботиться об этом процессе. TinyGo делает все это за нас. Теперь давайте посмотрим, что делает TinyGo особенным по сравнению с Go.

## Сравнение TinyGo с Go

TinyGo может скомпилировать некоторые, но не все программы Go. Давайте рассмотрим пример, который может быть скомпилирован обеими языковыми оболочками. Для примера напишем небольшую программу Hello World в Go – создадим ее и посмотрим размер программы.

1. Это самая минимальная программа Hello World, которую я могу придумать на данный момент:

```
package main
func main() {
print("Hello World\n")
}
```

   Для печати строки не требуется внешний пакет, такой как `fmt`.

2. Я буду использовать Go 1.15.2 в операционной системе Ubuntu 20.01. Чтобы проверить текущую установленную версию Go, используйте команду *go version*:

```
$ go version
go version go1.15.2 linux/amd64
```

3. Мы создаем файл с программой с помощью команды *go build*:

```
$ go build ./ch1/hello-world/
```

4. Теперь проверяем размер с помощью команды *ls –l*:

```
$ ls -l
-rwxrwxr-x 1 tobias tobias 1231780 Okt 4 19:31 hello-world
```

Итак, программа имеет 1 231 780 байт, что составляет 1,23178 Мб. Это довольно много для программы, состоящей всего из четырех строк кода.

> **Примечание**
>
> Команда *ls* доступна не во всех операционных системах. Если вы хотите самостоятельно проверить размеры, необходимо использовать инструменты, доступные в вашей операционной системе. Размер бинарного файла может отличаться при его анализе, так как команда разработчиков языка Go продолжает оптимизировать компилятор. Кроме того, размер бинарного файла может отличаться при сборке для других операционных систем.

Теперь давайте проверим, каков размер той же программы, но на этот раз скомпилированной с использованием TinyGo. Поскольку TinyGo не

поддерживает создание двоичного кода для Windows, я позабочусь о компиляции, поэтому вы можете просто сравнить размеры, изложенные в книге.

1. Я использовал следующую команду для создания бинарного файла:

```
$ tinygo build -o hello-world-tiny ch1/hello-world/main.go
```

   Команда *tinygo build* имеет синтаксис, аналогичный команде *Go build*.

2. Затем я проверил размер с помощью команды *ls –l*, как мы делали раньше:

```
$ ls -l
-rwxrwxr-x 1 tobias tobias 1231780 Okt 4 19:31 hello-world
-rwxrwxr-x 1 tobias tobias 21152 Okt 4 19:39 hello-world-tiny
```

Мы видим, что версия TinyGo нашей программы Hello World составляет лишь малую часть того размера, который был выдан компилятором Go. Версия TinyGo составляет всего 21 152 байта, что примерно 0,021152 Мб. Размер программа на TinyGo в 58 раз меньше программы, написанной на языке Go. Это огромная разница. Если вы все еще хотите проверить это самостоятельно, то можете сделать это после настройки TinyGo.

Мы узнали с вами, что TinyGo может компилировать некоторые, но не все программы Go. Кроме того, также узнали, что программы, скомпилированные с помощью TinyGo, очень малы. В следующих разделах мы узнаем, почему TinyGo не может скомпилировать все программы Go и какие функции TinyGo предлагает, а Go не предлагает.

## Поддерживаемые языковые функции

TinyGo поддерживает часть функций языка Go, но не все поддерживается прямо сейчас. Горутины (goroutines) и каналы работают на большинстве микроконтроллеров. Рефлексия поддерживается для большинства типов. Хотя срезы (slice) и поддерживаются, но при работе с картами (хеш-таблицами) могут возникнуть некоторые проблемы. Поддерживаются только определенные типы данных: строки, целые числа, указатели, структуры и массивы. Таким образом, в целом значительная часть языка Go поддерживается в TinyGo.

## Поддерживаемые стандартные пакеты

Большая часть стандартной библиотеки также поддерживается в TinyGo. Однако на момент написания статьи большинство *net*- и *crypto*-пакетов все еще не компилируется. Это означает, что если вы их будете импортировать, то, скорее всего, получите ошибки компиляции. Вы можете просмотреть список поддерживаемых в настоящее время стандартных пакетов здесь: **https://tinygo.org/lang-support/stdlib/**.

> **Примечание**
>
> Каждая функция в пакете, указанном в таблице поддержки, в действительности может не использоваться в TinyGo. Некоторые функции все еще могут вызывать ошибки компиляции.

## Операции volatile

Операции volatile могут использоваться для чтения и записи из регистров, отображенных в памяти. Значения внутри этих регистров могут меняться между несколькими считываниями без ведома компилятора. Компилятор не знает о последствиях этих операций, поэтому они называются изменчивыми (volatile).

У Go нет оператора volatile, поэтому TinyGo предоставляет пакет volatile. В большинстве случаев нам не понадобятся операции volatile, так как они абстрагируются машинным пакетом.

## Встроенный ассемблер

Язык ассемблера (ASM) – это язык, специально разработанный для определенной архитектуры процессора. Он необходим потому, что сборка зависит от набора инструкций машинного кода. Пакеты TinyGo, предназначенные для конкретных устройств, предоставляют пакеты такой сборки. Это позволяет нам использовать встроенный ассемблерный код в наших программах Go, что невозможно в стандартном языковом пакете Go.

## Распределение памяти

*Heap* (куча) – это часть памяти, в которой во время выполнения происходят динамические распределения и освобождения. Поэтому, когда наше приложение хочет зарезервировать часть памяти, оно взаимодействует с heap, чтобы зарезервировать память. Затем эта часть памяти будет помечена как используемая. Поскольку это пространство довольно ограничено на микроконтроллерах, а сбор мусора является дорогостоящим и медленным, TinyGo пытается оптимизировать распределение памяти. В результате часто объекты могут быть распределены статически, а не динамически.

## Сборка мусора

Сборка мусора – это процесс освобождения памяти. Поэтому, когда вашему приложению больше не нужна часть памяти, которую оно запрашивало ранее, эта память помечается как неиспользуемая (свободная).

Для этой цели TinyGo реализовал свой собственный вариант сбора мусора. TinyGo использует сборку мусора с консервативной меткой/разверткой, где консервативность означает, что сборщик мусора (GC) не знает, что является указателем, а что – нет. Процесс GC разделен на две части.

- **Отметка**: на этапе маркировки gc помечает объекты как доступные.
- **Очистка**: на этапе очистки gc освобождает память, помечая области, где отсутствуют объекты, как свободные. Эти освобожденные области памяти затем могут быть повторно использованы для выделения под новые объекты.

Теперь мы знаем, что такое TinyGo и какие существуют различия между TinyGo и Go. Также узнали, что такое куча, GC и пакет volatile. Следующий логический шаг –продолжить изучать и настраивать TinyGo, что мы и сделаем в следующем разделе.

# Установка TinyGo

Самый простой способ установить TinyGo и все его зависимости – это выполнить быстрый запуск руководства для Linux, macOS, Windows и Docker по следующей ссылке: **https://tinygo.org/getting-started/**.

Поскольку эти руководства охватывают важные части, я расскажу только о части быстрого запуска для архитектур на базе x64 и только для операционных систем на базе Debian, таких как Ubuntu для Linux.

Первое, что нужно сделать, прежде чем мы начнем настройку, – это проверить последнюю версию TinyGo. Для этого перейдите на **https://github.com/tinygo-org/tinygo/releases** и узнайте последнюю версию выпуска. Теперь запишите эту информацию где-нибудь или запомните, так как мы будем использовать ее позже.

## Установка в Linux

Следующие шаги охватывают установку TinyGo для разновидности операционной системы Linux, которая основана на Debian.

1. Мы используем следующую команду, чтобы загрузить пакет *deb* с GitHub и установить его с помощью *dpkg*:

```
wget https://github.com/tinygo-org/tinygo/releases/
download/v0.15.0/tinygo_0.15.0_amd64.deb
sudo dpkg -i tinygo_0.15.0_amd64.deb
```

   Вы можете изменить версию, указанную в пути и имени файла, на самую новую версию выпуска, которую нашли ранее.

2. Теперь мы должны добавить TinyGo в GOPATH. Используйте следующую команду:

```
export PATH=$PATH:/usr/local/tinygo/bin
```

   Вы также можете расширить GOPATH, отредактировав свой файл *.profile* или *.netrc*.

3. Следующим шагом является проверка установки. Используйте команду *tinygo version*, чтобы убедиться, что TinyGo успешно установлен:

```
$ tinygo version
tinygo version 0.15.0 linux/amd64 (using go version
go1.15.2 and LLVM version 10.0.1)
```

4. Зависимости AVR: поскольку мы собираемся работать с Arduino UNO в первых главах, нам необходимо установить некоторые дополнительные зависимости. Устанавливаем их с помощью следующих команд:

```
sudo apt-get install gcc-avr
sudo apt-get install avr-libc
sudo apt-get install avrdude
```

После установки этих зависимостей мы можем компилировать код на платы, разработанные на базе AVR, такие как Arduino UNO.

Если вы используете Fedora, Arch Linux или другие дистрибутивы, пожалуйста, следуйте руководству по установке: **https://tinygo.org/getting-started/linux/**.

## Установка в Windows

В этом разделе мы узнаем, как установить TinyGo в Windows. Затем увидим, как устанавливать зависимости, которые необходимы для прошивки Arduino UNO.

> **Очень важное замечание**
>
> Вы не можете создавать программы под Windows с помощью TinyGo, но все еще можете компилировать и прошивать программы для реализации целей микроконтроллера и веб-сборки.
>
> Возможно, вы захотите напрямую установить и использовать TinyGo внутри Windows с помощью **Subsystem for Linux** (WSL). WSL – это хороший инструмент, который я рекомендую пользователям Windows.

Если необходимо установить TinyGo в Windows без использования WSL, то я рекомендую использовать Scoop – установщик командной строки для Windows. Убедитесь перед этим, что у вас установлены PowerShell 5 (или более поздняя версия) и .NET Framework 4.5 (или более поздняя версия). Для этого, пожалуйста, выполните следующие действия.

1. Включите PowerShell для вашей текущей учетной записи пользователя, используя следующую команду:

```
Set-ExecutionPolicy RemoteSigned -scope CurrentUser
```

2. Теперь выполните следующую команду для загрузки Scoop:

```
iwr -useb get.scoop.sh | iex
```

3. Вы можете установить TinyGo, используя следующую команду:

```
scoop install tinygo
```

4. Теперь, чтобы убедиться, что установка прошла успешно, используйте команду:

```
tinygo version
```

   Вывод должен выглядеть следующим образом:

```
tinygo version 0.15.0 windows/amd64 (using go version
go1.15.3 and LLVM version 10.0.1)
```

   Фактическая версия TinyGo и Go может отличаться.

5. Зависимости AVR: для того чтобы иметь возможность компилировать и запускать программы для Arduino UNO, нам необходимо установить 8-разрядную цепочку инструментов AVR. Вы можете найти информацию по загрузке здесь: **https://www.microchip.com/mplab/avr-support/avr-and-arm-toolchains-c-compilers**.

   Расширьте свой *%PATH%* и убедитесь, что папка *bin* включена.

6. Затем загрузите и установите GNU Make для Windows. Вы можете найти GNU Make здесь: **http://gnuwin32.sourceforge.net/packages/make.htm**.

7. В качестве последнего шага необходимо загрузить и установить avrdude. Исполняемый файл avrdude также должен быть внутри вашего *%PATH%*. Можете скачать AVR Dude здесь: **http://download.savannah.gnu.org/releases/avrdude/**. Файл, который вы ищете, называется *avrdude-6.3-mingw32.zip*.

Если у вас возникли какие-либо проблемы с настройкой avr или вы не знаете, как настроить переменные среды, можете ознакомиться со следующим руководством: **https://fab.cba.mit.edu/classes/863.16/doc/projects/ftsmin/windows_avr.HTML**.

> **Установка WSL**
>
> Также возможно установить TinyGo непосредственно в **Windows Subsystem for Linux** (WSL). Просто следуйте указаниям по разделу Linux, чтобы сделать это.

## Установка на macOS

Установка на macOS проста. Давайте быстро взглянем на шаги.

1. Мы собираемся использовать Homebrew для установки *tinygo*. Используйте следующие две команды:

```
brew tap tinygo-org/tools
brew install tinygo
```

2. Затем запустите команду *tinygo version*, чтобы проверить установку:

```
$ tinygo version
tinygo version 0.15.0 darwin/amd64 (using go version
go1.15 and LLVM version 10.0.1)
```

3. Выполните следующие команды, чтобы установить дополнительные требования, необходимые для компиляции программ для микроконтроллеров на базе AVR, таких как Arduino UNO, которые мы собираемся использовать в первых нескольких главах книги:

```
brew tap osx-cross/avr
brew install avr-gcc
brew install avrdude
```

### Установка в Docker

Для компиляции наших программ можно напрямую использовать образ Docker. Однако прошить программы с его помощью невозможно. Просто загрузите образ, используя следующее:

```
docker pull tinygo/tinygo:0.15.0
```

> **Примечание**
>
> Фактическая версия TinyGo может отличаться. Используйте новейшую версию TinyGo из проверки, которую мы сделали, когда начали раздел.

Вот пример вызова для создания программы:

```
docker run -v $GOPATH:/go -e "GOPATH=/go" tinygo/tinygo:0.15.0
tinygo build -o /go/src/github.com/myuser/myrepo/wasm.wasm
-target wasm --no-debug /go/src/github.com/myuser/myrepo/wasmmain.go
```

Теперь мы успешно настроили TinyGo и установили все дополнения, согласно требованиям для компиляции и прошивки программ под микроконтроллер Arduino UNO. Кроме того, все, что нужно для веб-сборки, теперь также настроено. Следующим шагом является настройка интеграции IDE, прежде чем мы начнем писать нашу первую программу для микроконтроллера.

## Настройка интеграции IDE с TinyGo

Наличие правильно настроенной интегрированной среды разработки (IDE) – это действительно благо, поскольку мы извлекаем выгоду из ее функций для правильного написания кода, функциональной компоновки и т. д. Таким образом, не нужно изучать исходный код или документацию для каждой функции, которую мы хотим вызвать.

В этом разделе мы рассмотрим процесс интеграции TinyGo в VS Code, Goland и другие редакторы, что позволит выбрать любой редактор, который мы предпочитаем использовать.

## Интеграция в VS Code

VS Code предлагает систему расширений, позволяющую легко интегрировать набор инструментов Go и TinyGo в среду разработки. Мы собираемся установить расширение Go, которое предлагает поддержку языка программирования Go. После этого необходимо установить расширение TinyGo, обеспечивающее поддержку TinyGo.

### *Расширение Go*

Вы установите расширение Go с помощью вкладки **Extensions**, выполнив следующие действия.

1. Откройте **Extensions**, щелкнув значок расширений или нажав **Ctrl**+**Shift**+**X**.
2. Найдите **Go**.
3. Выберите первую запись в списке, которая называется Go и принадлежит команде разработчиков Go от Google.
4. Нажмите на кнопку **Install**, как показано на следующем снимке экрана.


**Рис. 1.1.** Установка из окна Extensions

5. После первой установки расширения вам может быть предложено установить дополнительные зависимости. Сделайте это, нажав на кнопку **Install**. Не получив приглашения, вы также можете установить все зависимости, нажав **Ctrl**+**Shift**+**P** и введя следующую команду:

```
Go: install
```

6. Выберите **Go: Install/Update Tools** и нажмите **Enter**.

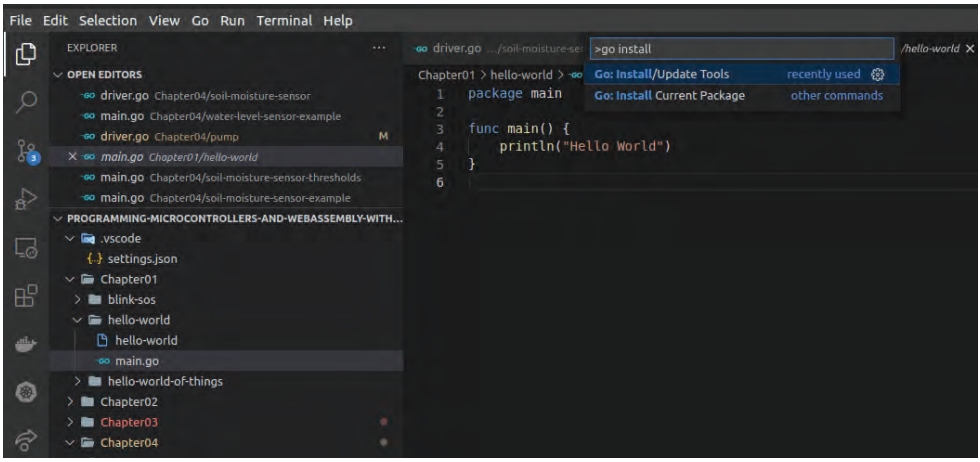


**Рис. 1.2. Go: Install/Update Tools**, команда для выполнения

7. Теперь выберите все зависимости, установив флажок слева, и нажмите кнопку **OK**.

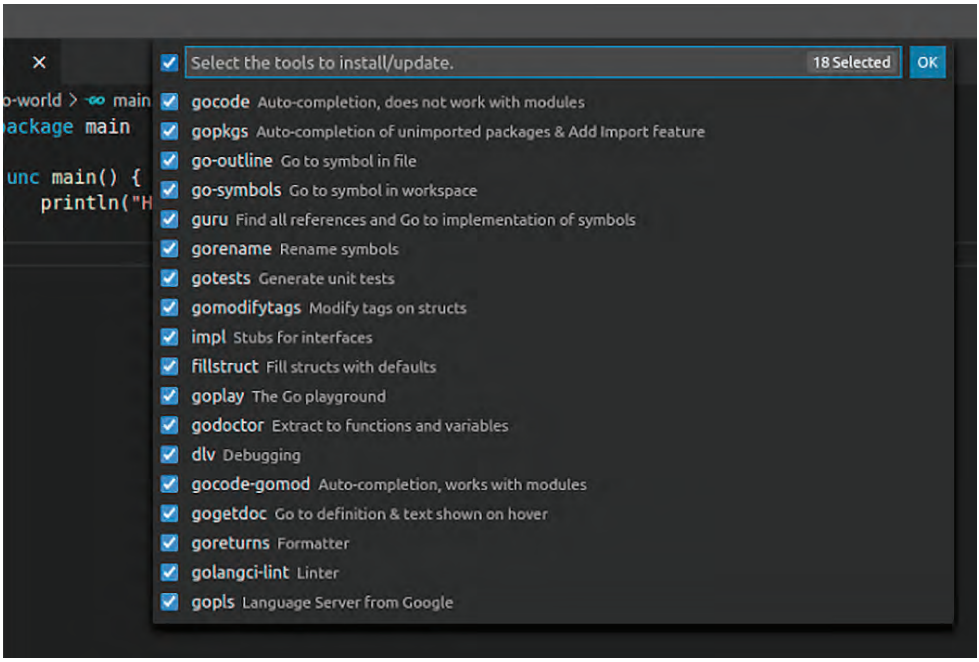


**Рис. 1.3.** Выбор всех зависимостей

8. VS Code теперь установит все зависимости, и по завершении он должен напечатать следующее сообщение:

```
All tools successfully installed. You are ready to Go :).
```

Далее мы рассмотрим интеграцию TinyGo в VS Code.

### *Расширение TinyGo*

Интеграция TinyGo в VS Code проста, так как существует расширение TinyGo, которое нам просто нужно установить. Давайте быстро пройдемся по шагам.

1. Откройте окно **Extensions**, щелкнув значок расширений или нажав **Ctrl+Shift+X**.
2. Найдите TinyGo.
3. Выберите первую запись в списке, которая называется *TinyGo* и принадлежит команде разработчиков TinyGo.
4. Нажмите на кнопку **Install**, как показано на следующем снимке экрана.

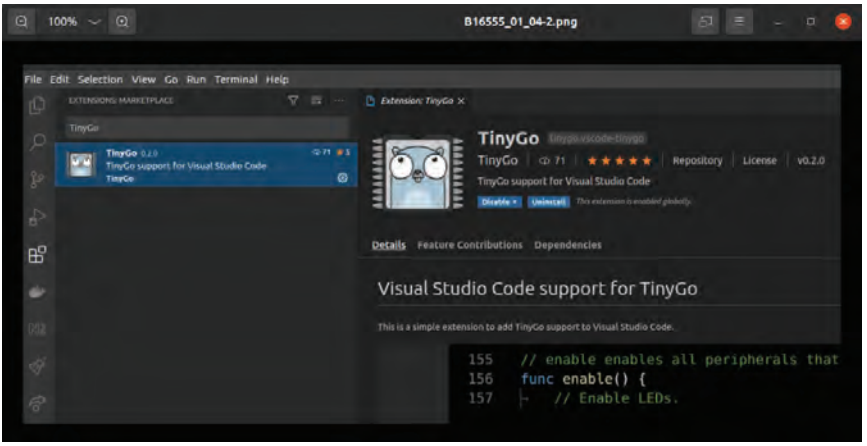


**Рис. 1.4.** Нахождение TinyGo в Extensions

5. Мы еще не закончили с установкой расширения. Нужно использовать другую команду для настройки назначения, для которой и хотим установить данное расширение. Нажмите **Ctrl+Shift+P**, введите *TinyGo target* и нажмите **Enter**.
6. Теперь найдите `arduino` и нажмите **Enter**, как мы видим на следующем скриншоте.

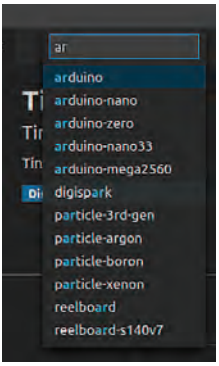


**Рис. 1.5.** Всплывающее окно выбора цели

7. VS Code откроет всплывающее окно, сообщающая вам, что ему необходимо перезагрузиться. Сделайте это, нажав на кнопку **Reload**.

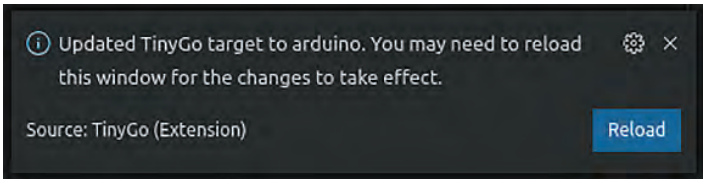


**Рис. 1.6.** Всплывающее окно с просьбой перезагрузить окно

Хорошо, теперь мы установили расширение и выбрали его назначение. Но что оно делает? Единственная функция этого расширения – установить переменную *go.toolsEnvVars* в файле *vs code settings.json* вашего текущего проекта. Это может выглядеть как в следующем примере:

```
{
  "go.toolsEnvVars": {
    "GOROOT": "/home/user/.cache/tinygo/goroot-go1.14-f930d5b
        5f36579e8cbf1f-syscall",
    "GOFLAGS": "-tags=cortexm,baremetal,linux,arm,nrf51822,
        nrf51,nrf,microbit,tinygo,gc.conservative,scheduler.tasks"
  }
}
```

Иногда может появляться всплывающее окно, похожее на то, что показано на следующем снимке экрана. Не нажимайте на инструменты обновления; просто закройте его.

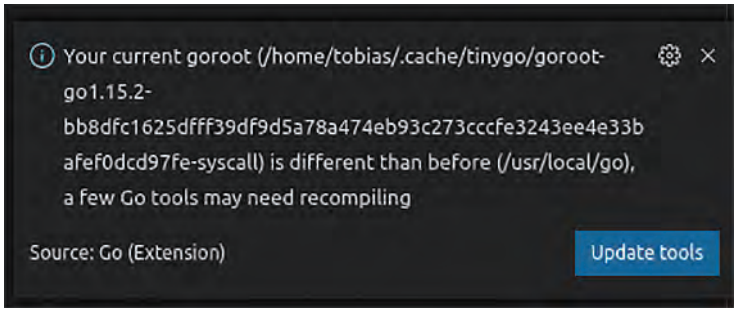


**Рис. 1.7.** Всплывающее окно с просьбой обновить инструменты

Если вы используете VS Code – поздравляю, вы закончили настройку и готовы к работе! В следующих разделах будет объяснено, как настроить интеграцию IDE в других редакторах.

## Общая интеграция с IDE

Вы можете задаться вопросом, как интеграция IDE работает с TinyGo. Ну, нам просто нужно настроить стандартный инструментарий Go, особенно языковой сервер *gopls*.

TinyGo имеет собственную реализацию стандартных библиотек, а также предоставляет дополнительные библиотеки, такие как пакет *machine*. Языковой сервер *gopls* должен знать, где искать эти пакеты. Поэтому нам нужно установить **GOROOT** для этого проекта.

TinyGo активно использует флаги компилятора. Эти флаги используются во время компиляции для определения, какие файлы должны быть включены в сборку, как мы видим на следующем снимке экрана.

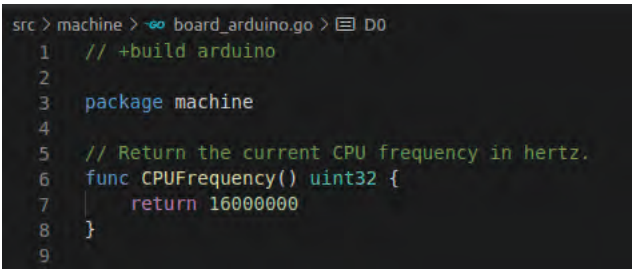


**Рис. 1.8.** Файл *board_arduino.go* из исходного кода TinyGo с флагом сборки

Таким образом, в основном мы интегрируем TinyGo в среду разработки, локально устанавливая эти переменные среды.

Нам не нужно угадывать правильные значения для *GOROOT* и *GOFLAGS*. TinyGo предоставляет команду для этого назначения. Допустим, мы хотим установить правильные флаги для Arduino. Для этого воспользуемся следующей командой:

```
tinygo info arduino
```

Это выведет следующий результат:

```
LLVM triple:            avr-unknown-unknown
GOOS:                   linux
GOARCH:                 arm
build tags:             avr baremetal linux arm atmega328p atmega
avr5 arduino tinygo gc.conservative scheduler.none
garbage collector:      conservative
scheduler:              none
cached GOROOT:          /home/tobias/.cache/tinygo/goroot-go1.15.2-
bb8dfc1625dfff39df9d5a78a474eb93c273cccfe3243ee4e33bafef0dcd97fe-syscall
```

Важными частями выходных данных являются теги сборки и кешированные данные. Поскольку теперь мы знаем, где найти необходимую информацию, то можем сделать следующий шаг и настроить любую среду разработки, которую хотим использовать.

## Настройка Goland

Поскольку мы теперь знаем, что должны установить *GOROOT* и настроить теги, то теперь можем перейти к настройке интеграции с Goland.

Установите GOROOT в кешированный *GOROOT* с помощью команды *tinygo info*, как показано на следующем снимке экрана.

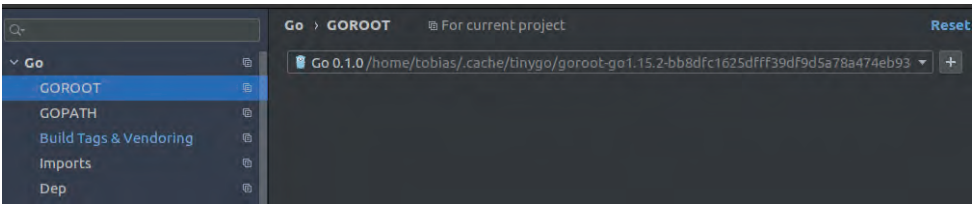


**Рис. 1.9.** Настройка GOROOT в Goland

Следующий шаг – установить теги сборки. Вы можете найти их в разделе **Build Tags & Vendoring**. Добавьте теги в поле **Custom tags**.

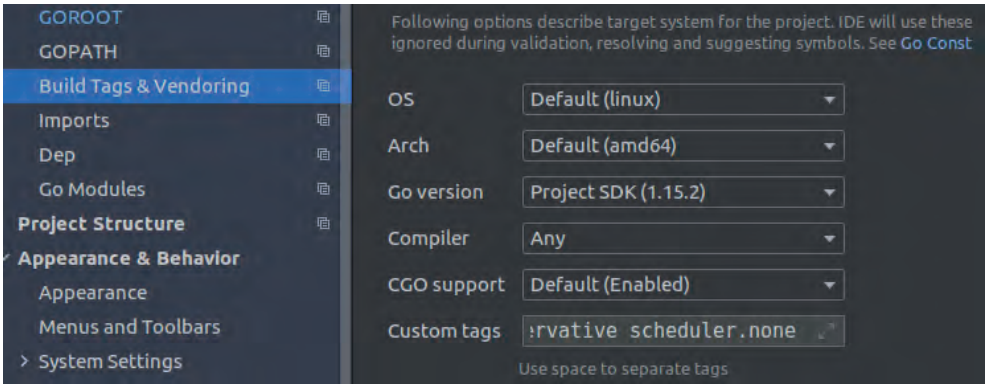


**Рис. 1.10.** Настройка пользовательских тегов в Goland

> **Примечание**
>
> Вы должны вручную изменять пользовательские теги каждый раз, когда хотите программировать для другого микроконтроллера.

## Интеграция любого редактора

Если у вас установлен стандартный Go tolling, то вы можете использовать любой другой редактор, такой как Vim или Nano, чтобы получить поддержку IDE. Поскольку в других редакторах могут отсутствовать файлы конфигурации, мы можем обойти этот факт, передав им переменные значения среды в момент их запуска.

В следующем примере мы запускаем экземпляр VS Code, сначала задав переменные среды, а затем запустив VS Code:

```
export GOFLAGS=-tags=avr,baremetal,linux,arm,atmega328p,atmega,
avr5,arduino,tinygo,gc.conservative,scheduler.none; code
```

Вы можете изменить вызов кода для любой другой программы, такой как *vim* или *nano*. В системах Windows вызов может выглядеть немного по-другому.

Поскольку теперь мы знаем, как настроить любую интегрированную среду разработки для использования TinyGo, то перейдем к изучению платы Arduino UNO.

# Arduino UNO

Arduino UNO – одна из самых популярных плат в мире. Она питается от 8-разрядного микроконтроллера ATmega328P, и на момент написания этой книги существует множество производных от оригинальных плат Arduino UNO. Давайте получше познакомимся с этим в следующих подразделах.

## Знакомство с техническими характеристиками

Как вы можете видеть в следующей таблице, ATmega328P имеет только 16 МГц и 32 Кб флеш-памяти. Стандартный Go производит программу Hello World объемом около 1,2 Мб, которая даже не поместилась бы на этом микроконтроллере. Итак, мы работаем здесь с очень ограниченным по возможностям оборудованием, но вы увидите, что этого достаточно для создания потрясающих проектов.

Краткий обзор технических характеристик Arduino UNO приведен в табл. 1.1.

**Таблица 1.1.** Технические характеристики

| | |
|---|---|
| Рабочее напряжение | 5 В |
| Пределы входного напряжения | 6–20 В |
| Рекомендуемое входное напряжение | 7–12 В |
| Цифровые контакты ввода–вывода | 14 (6 из них с ШИМ-выходом) |
| Контакты аналогового входа | 6 |
| Постоянный ток на контакты ввода–вывода | 20 мА |
| Постоянный ток для вывода 3,3 В | 50 мА |
| Флеш-память | 32 Кб |
| Статистическое ОЗУ (SRAM) | 2 Кб |
| Частота | 16 МГц |
| Встроенные светодиод, связанный с контактом ввода–вывода | 13 |

> **Примечание**
>
> В качестве верхнего предела считайте постоянный ток на вывод ввода–вывода 20 мА. Не стоит превышать этот предел, чтобы предотвратить повреждение вашего микроконтроллера.

Давайте посмотрим на распиновку более подробно.

## Изучение распиновки

Распиновка – это в основном карта контактов. Данная распиновка будет использована в описании контактов для всех проектов, созданных с помощью Arduino UNO. Данное описание нам понадобится, чтобы правильно собрать электрическую схему из наших компонентов.

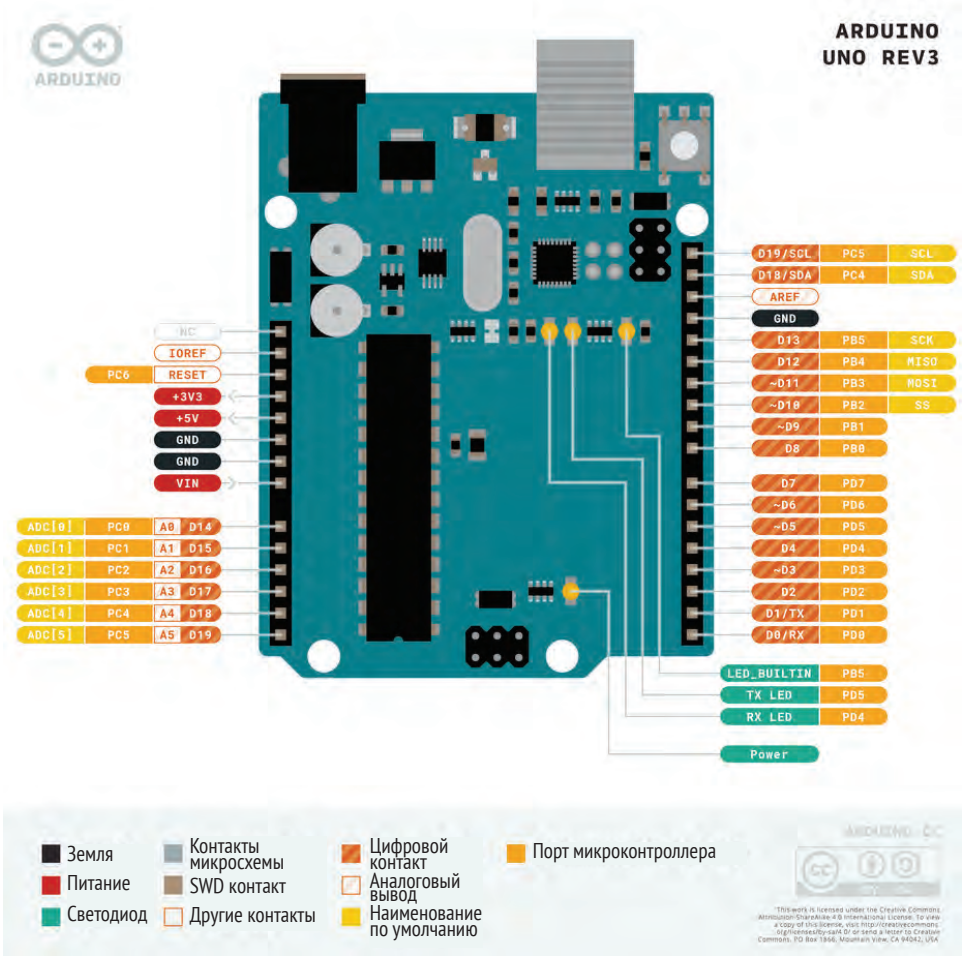


**Рис. 1.11.** Распиновка Arduino UNO REV3

Поскольку теперь мы узнали некоторые основные сведения об Arduino UNO, давайте продолжим и напишем нашу первую программу.

# Проверяем работу программы Hello world в устройстве

Программа Hello World – это типичный способ начать путешествие на новом языке программирования. Программа Hello World на микроконтроллере выглядит немного иначе по сравнению с обычной. Мы собираемся написать программу Hello World, чтобы встроенный светодиод мигал. Давайте начнем!

## Подготовка

Чтобы начать работу с нашей программой, нам нужно следующее:

- Arduino UNO;
- один USB-кабель для подключения его к компьютеру.

## Подготовка проекта

Внимательно выполните следующие действия.

1. Создайте новую папку с именем *ch1* в корне вашего проекта.
2. В ней нужно создать папку с именем *hello-world-of-things*, и внутри нее – новый файл *main.go*.
3. Теперь ваша структура проекта должна выглядеть следующим образом.

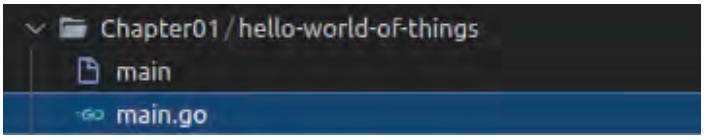


**Рис. 1.12.** Структура проекта

Поскольку мы уже подготовили проект, то можем приступить к написанию нашей первой программы.

## Программирование микроконтроллера

Нужно заставить светодиод на плате Arduino мигать с определенным интервалом. Для нас это самый простой из возможных стартовых проектов. Пример, который мы используем, вдохновлен примером Blinky из исходного кода TinyGo, который также используется в качестве демонстрации Hello World of Things. Давайте тщательно изучим каждый шаг.

1. Объявите основной пакет:

```
package main
```

2. Импортируйте пакеты *machine* и *time*:

```
import (
    "machine"
    "time"
)
```

3. Добавьте функцию *main*:

```
func main() {
```

4. Инициализируйте переменную с именем *led* с помощью значения *machine.LED*:

```
    led := machine.LED
```

5. Настройте пин со светодиодом в качестве выходного контакта:

```
    led.Configure(machine.PinConfig{Mode: machine.
    PinOutput})
```

6. Объявите бесконечный цикл:

```
    for {
```

7. Установите *led* на значение *Low*, чтобы на светодиоде не подавалось напряжение:

```
    led.Low()
```

8. Установите режим ожидания на 300 мс, т. е. время, в течение которого светодиод выключен:

```
    time.Sleep(time.Millisecond * 300)
```

9. Установите *led* на значение *High*, чтобы на светодиод подавалось напряжение, тогда он начнет светиться:

```
    led.High()
```

10. Установите режим ожидания на 300 мс, что соответствует времени, в течение которого горит индикатор:

```
    time.Sleep(time.Millisecond * 300)
```

11. Закройте цикл *for*:

```
}
```

12. Закрывающие скобки для основной функции:

```
}
```

Пакет *machine* предоставляет константы для сопоставления выводов на плате Arduino и еще несколько функций, которые напрямую связаны с используемым микроконтроллером.

Мы должны подождать определенное время между подачей напряжения на светодиод и его повторным выключением, чтобы наблюдать мигание.

Настройка пина в качестве выходного означает, что мы сообщаем микроконтроллеру, что будем отправлять сигналы только с помощью этого пина. Можно также настроить пин в качестве входного, что позволяет нам считывать состояние с пина.

## Прошивка программы

Прошивка программы – это простая команда, если вы используете Linux, macOS или Windows WSL.

Просто подключите Arduino UNO к любому USB-порту и выполните следующую команду:

```
tinygo flash --target=arduino ch1/hello-world-of-things/main.go
```

Для команды *tinygo flash* требуются, по крайней мере, следующие параметры:

- *--target*, который назначает микроконтроллеру задачу на вспышку;
- путь к файлу *main.go*.

Ваш вывод должен выглядеть так:

```
avrdude: AVR device initialized and ready to accept instructions
Reading | ################################################## |
100% 0.00s
avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: NOTE: "flash" memory has been specified, an erase
cycle will be performed
        To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "/tmp/tinygo208327574/main.hex"
avrdude: writing flash (558 bytes):
Writing | ################################################## |
100% 0.10s
avrdude: 558 bytes of flash written
avrdude: verifying flash memory against /tmp/tinygo208327574/
main.hex:
avrdude: load data flash data from input file /tmp/tinygo208327574/
main.hex:
avrdude: input file /tmp/tinygo208327574/main.hex contains 558 bytes
avrdude: reading on-chip flash data:
Reading | ################################################## |
100% 0.08s
avrdude: verifying ...
avrdude: 558 bytes of flash verified
avrdude done. Thank you.
```

Как вы можете видеть, в моем примере код, загруженный на Arduino UNO, использует только 558 байт памяти.

Поздравляем, вы успешно написали, создали и запустили свою первую программу на Arduino UNO с помощью TinyGo.

### Использование игровой площадки TinyGo

У вас сейчас нет Arduino UNO? Можете протестировать код с помощью игровой площадки TinyGo. Игровая площадка TinyGo использует веб-сборку для эмуляции поведения небольшого числа плат, таких как Arduino Nano IoT 33 и Arduino UNO. Она также может компилировать программы для Arduino Nano IoT 33. Но, пожалуйста, имейте в виду, что результат работы программы на игровой площадке TinyGo может отличаться от реального оборудования.

Можете найти игровую площадку TinyGo по адресу **https://play.tinygo.org/**.

## Резюме

Вы узнали, что такое TinyGo, чем он отличается от стандартного Go. Также получили базовые знания о самой Arduino UNO, о том, как настроить TinyGo, а также интегрированную среду разработки под работу с TinyGo и, наконец, написали и перенесли нашу первую программу с кодом на реальное оборудование, заставив мигать светодиод. Разве это не интересное начало?

В следующей главе мы собираемся создать систему управления светофорами.

## Вопросы

1. Какую команду можно использовать для определения значений переменных среды, необходимых для интеграции с IDE?
2. Какую команду можно использовать для переноса программы на микроконтроллер?
3. Почему мы должны устанавливать временную задержку на определенное количество времени, подавая напряжение или снимая напряжение со светодиода?
4. Как бы вы позволили светодиоду мигать S-O-S в азбуке Морзе?