С. Яманур, Ш. Яманур

Raspberry Pi Pico в любительских проектах

Под науч. ред. Ю. В. Ревича

ДМК Пресс
издательство

Сай Яманур
Шрихари Яманур

Raspberry Pi Pico в любительских проектах

Raspberry Pi Pico DIY Workshop

Sai Yamanoor
Srihari Yamanoor

BIRMINGHAM – MUMBAI

Raspberry Pi Pico в любительских проектах

Сай Яманур
Шрихари Яманур

Москва, 2023

УДК 004.738, 004.62
ББК 32.973
С14

С14 Сай Яманур, Шрихари Яманур
Raspberry Pi Pico в любительских проектах / пер. с англ. Ю. В. Ревича. – М.: ДМК Пресс, 2022. – 282 с.: ил.

ISBN 978-5-93700-165-8

Эта книга поможет вам начать работу с миниатюрным и быстрым микроконтроллером Pico и использовать его возможности для разработки инновационных продуктов. Вы узнаете о периферийных устройствах микроконтроллера, а также о разновидностях плат Pico, разработанных и изготовленных различными производителями; затем освоите практические навыки на ряде проектов – начиная с мигающего светодиода и заканчивая большим семисегментным дисплеем.

Издание предназначено любителям DIY, а также будет полезно программистам, студентам, инженерам, студентам и школьникам, которые хотят разрабатывать встроенные системы для домашней автоматизации, «интернета вещей», робототехники и др.

ISBN 978-1-80181-481-2 (англ.)
ISBN 978-5-93700-165-8 (рус.)

Всем умельцам, которые начинают несколько проектов одновременно и мечтают однажды закончить их.

– Сай Ямануp (Sai Yamanoor)

Замечательным кошкам, завладевшим моим сердцем: Сквики, Бобу, Саксону, Джи-Ви, Пушистому Туксу, Ангелу, Серому Коту, а теперь еще и Альфи, Фиш-Боуну и Сакси!

– Шрихари Ямануp (Srihari Yamanoor)

Оглавление

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 авторах

Сай Яманур – старший инженер по приложениям интернета вещей (IoT) в компании по производству промышленных газов в Буффало, штат Нью-Йорк. Имеет более чем 10-летний опыт работы в качестве эксперта по встраиваемым системам, работая как над разработкой, так и над внедрением аппаратного и программного обеспечения. Соавтор двух книг об использовании Raspberry Pi для выполнения проектов «сделай сам». Представил персональную панель мониторинга здоровья на общенациональной выставке Maker Fair. В настоящее время Сай работает над проектами, направленными на улучшение качества жизни (QoL) людей с хроническими заболеваниями.

> *«Я хочу поблагодарить своих родителей и моего брата и соавтора Шри за всю помощь и поддержку; наших технических рецензентов Салмана Фариса и Джонатана Виттса за их проницательные комментарии и за тщательный анализ нашей работы; Рахула Наира за предоставленную нам возможность работать с издательством Packt. Я также хотел бы поблагодарить Роми Диас и Вайдехи Савант за их терпение и поддержку нашей работы».*

Шрихари Яманур – инженер-механик с опытом работы в области проектирования медицинских устройств, CAD/CAM, механотроники и надежной аппаратуры. В сотрудничестве со своим братом разрабатывает аппаратные продукты с открытым исходным кодом, направленные на повышение популярности в любительских кругах. Имеет множество сертификатов в области обеспечения качества, САПР и ВЭД. Помимо дизайна, производства и качества электронных изделий, его текущие интересы включают изменение поведения и активное самосовершенствование в борьбе с диабетом, инновационные парадигмы и методологии в области здоровья, а также влияние искусственного интеллекта на здравоохранение. Он является соавтором двух книг по приложениям Raspberry Pi и пишет блоги на различные темы.

> *«Я хочу поблагодарить своих родителей, наставников, друзей, кошек, а также моего брата и соавтора Саи за всю помощь и поддержку. Я хотел бы выразить благодарность своим наставникам Анну Тамуру и доктору Судхи Гаутама, а также моему другу Сатьяканту Тьяграджe за то, что он поддержал меня в трудные времена, и команде издательства Packt за их поддержку этой книги и других наших усилий на протяжении многих лет».*

О рецензентах

Салман Фарис – энтузиаст разработки и быстрого создания прототипов цифровых продуктов из Индии. Имеет степень бакалавра в области компьютерных наук и диплом цифровой разработки Академии Fab. В настоящее время работает инженером технической поддержки в английской компании Nebra и является ключевым участником сообщества разработчиков MakerGram, где занимается разработкой электронных и аппаратных продуктов.

Салман также является частью экспертной группы Edge Impulse, сообществ Qubitro, RAK и Seeed Studio, а также послом и основным участником крупнейшего в Индии собрания разработчиков Maker Faire (Хайдарабад) и соорганизатором фестиваля Maker Fest в Керале.

> *«Сначала я хотел бы поблагодарить Аллаха за Его всемогущее руководство в любых решениях, которые я принимаю. Я также хотел бы поблагодарить издательство Packt Publishing за возможность ознакомиться с этой замечательной книгой, особенно Шагуна и Эшвина, которые руководили рецензированием и помогали мне советом и поддержкой на протяжении всего процесса. Спасибо моим родителям, братьям и сестрам, родственникам, друзьям, наставникам и команде Nebra».*

Джон Виттс работает в области информационных технологий в сфере образования уже более 17 лет. Имеет ученую степень по изобразительному искусству, а также дизайну и разработке электронного обучения. В своей нынешней роли директора по цифровой стратегии Джон руководит всеми технологическими решениями в своей школе, а также преподает информатику учащимся в возрасте 11–16 лет. Джон также проводит мероприятия Hull Raspberry Jam в своем родном городе: бесплатные семинары по программированию для молодых людей, использующих компьютер Raspberry Pi. Джон рецензировал ряд изданий для Packt и написал свою собственную книгу «Wearable-Tech Projects with the Raspberry Pi Zero», опубликованную Packt. В свободное время Джон с удовольствием занимается объединением компьютерных технологий с искусством (generative art), используя библиотеки JawaScript и элементы физических вычислений на Raspberry Pi.

> *«Я хотел бы поблагодарить мою жену Салли и наших трех дочерей, Мейбл, Эмбер и Аду, за всю их поддержку, позволившую мне работать над этой книгой, а также авторов и всю команду издательства Packt за то, что они позволили мне принять участие в процессе создания этой замечательной публикации».*

Предисловие

Когда в январе 2021 года компанией Raspberry Pi Foundation был сделан анонс Raspberry Pi Pico, мы были поражены новыми возможностями, которые в плате за 4 доллара США открылись для любителей, специалистов широкого профиля, гражданских[1] и профессиональных ученых, преподавателей и студентов по всему миру. Доступный в различных формах, мощный, но недорогой микроконтроллер действительно может работать сам по себе и с другими инструментами, помогая людям разрабатывать очень мощные и элегантные решения. Мы ожидаем, что, подобно предыдущим поколениям продуктов от Raspberry Pi Foundation, Raspberry Pi Pico совершит еще одну революцию в области технологий, образования, развлечений и других массовых начинаний.

Основываясь на нашем опыте создания публикаций об **одноплатных компьютерах** (Single-Board Computers, SBC) Raspberry Pi, мы написали эту книгу, чтобы познакомить читателя с новыми и старыми проектами, для удовлетворения различных потребностей целевой аудитории: студентов, преподавателей, инженеров, ученых, художников и технических энтузиастов, которые хотят разрабатывать встроенные системы, предназначенные для экономичной автоматизации, устройств IoT[2], робототехники, медицинских устройств и художественных проектов.

Мы постарались сохранить разнообразие в проектах, а также представили различные датчики, способы программирования, описания интерфейсов и другие подробности, достаточные для того, чтобы как новички, так и продвинутые читатели могли создавать и реализовывать свои задумки на основе Raspberry Pi Pico.

Для кого предназначена эта книга

Как уже говорилось, мы разработали материалы к проектам, рассчитанным на широкий круг читателей. Возможно, вы опытный любитель или профессионал, заинтересованный в понимании того, как Pico может помочь вам в ваших проектах. У вас может быть маленький или большой опыт работы с электроникой, одноплатными компьютерами, микроконтроллерами или программированием. Вы можете обладать всеми необходимыми навыками и быть в поиске новых проектов, чтобы развлечь себя или преподавать своим ученикам. Эта книга рассчитана на людей с самым разнообразным опытом и направлением работы.

Тем не менее некоторый базовый опыт в программировании, электронике и смежных областях будет очень полезен при ознакомлении с материалами и проектами в книге. Если вы хотите начать свой опыт программирования на

---

[1] Гражданская наука (citizen science) – направление исследований с привлечением добровольцев из числа непрофессионалов. Термин «гражданский ученый» (citizen scientist) распространился на Западе в последнее десятилетие. Как правило, этим термином называют волонтеров, участвующих на добровольных началах в каком-либо профессиональном проекте, а не просто ученых-любителей. – *Прим. перев.*

[2] Internet of Things, «интернет вещей». – *Прим. перев.*

Python, то можете обратиться к другой нашей публикации *Python Programming with Raspberry Pi* («Программирование на Python с помощью Raspberry Pi»), также выпущенной издательством Packt[3].

О чем рассказывает эта книга

В **главе 1** «*Начало работы с Raspberry Pi Pico*» излагаются основы Raspberry Pi Pico, его разновидности, аксессуары и способы программирования. Мы также покажем, как создать классический пример «Hello World» и заставить мигать светодиод.

В **главе 2** «*Последовательные интерфейсы и их приложения*» мы исследуем, как использовать последовательные интерфейсы Raspberry Pi Pico для взаимодействия с датчиками, дисплеями и другим оборудованием. Мы также продемонстрируем, как заставить работать модуль Wi-Fi и подключить Raspberry Pi Pico к интернету.

В **главе 3** «*Проекты домашней автоматизации*» рассматриваются простые проекты домашней автоматизации, которые можно выполнить за выходные, а также приложения с последовательными интерфейсами. Мы также представляем Arduino RP2040 Connect и то, как его можно использовать вместо Pico.

Глава 4 «*Весело проводите время в саду!*» позволяет нам глубже разобраться в реализации проектов с помощью Pico. Мы подключаем датчик почвы к живому растению, измеряем температуру и влажность почвы, загружаем данные на аналитическую платформу IoT и визуализируем собранные данные.

Глава 5 «*Строим метеостанцию*» – это особое удовольствие для любителей погоды и гражданских ученых. Мы построим метеостанцию с различными датчиками и вариантами интерфейса с Raspberry Pi Pico.

Глава 6 «*Проектируем настенный семисегментный дисплей*» посвящена созданию средств отображения. Мы обсуждаем управление дисплеем через последовательный порт или из локальной сети.

В **главе 7** «*Разрабатываем устройство слежения за качеством воздуха*», продолжая предыдущую главу, мы демонстрируем другое применение средств наглядного отображения, на этот раз используя два разных подхода: один с использованием существующих источников данных, а другой – с использованием датчика двуокиси углерода для определения качества воздуха.

В **главе 8** «*Беспроводная связь*» мы выходим за рамки Wi-Fi и исследуем иные способы сбора и передачи данных по беспроводной сети, применяя LoRa, Sigfox и Bluetooth. Это позволит вам свободно разрабатывать беспроводные приложения с помощью Pico.

[3] На русском языке можно рекомендовать онлайн-курс «Программирование на Python на Raspberry Pi» (**https://myraspberry.ru/programmirovanie-python-na-raspberry-pi.html**). Для более подробного ознакомления с языком Python рекомендуется книга «Основы программирования на языке Python» (М.: ДМК Пресс, 2017), для общего введения в одноплатный компьютер Raspberry Pi – книга «Raspberry Pi. Руководство по настройке и применению» (М.: ДМК Пресс, 2014). – *Прим. перев.*

Глава 9 «*Строим робота!*» предназначена для энтузиастов робототехники. Мы демонстрируем, как можно управлять роботом с помощью Pico. В этой главе мы также представим MicroPython для тех, кто планирует использовать легкий код.

Глава 10 «*Знакомство с приложениями TinyML*» – это руководство для тех из вас, кто хочет разрабатывать приложения искусственного интеллекта (ИИ) с помощью Pico. Мы познакомим вас с TinyML – фреймворком[4], специально ориентированным на облегченные приложения ИИ. Покажем вам пример распознавания ключевых слов, который поможет подготовить почву для дальнейшего изучения этого направления.

Глава 11 «*Создаем готовый продукт*» демонстрирует, как еще больше ускорить процесс и создать готовый продукт. Мы демонстрируем метод создания несущей печатной платы для Pico, а также разъясняем, как использовать сотовый модуль для подключения.

В **главе 12** «*Дополнительные возможности работы с Pico*» мы завершаем книгу советами и рекомендациями, которые помогут продвинуть ваши проекты с помощью Pico дальше. Мы обсуждаем, как можно обновить прошивку Pico, как Arduino IDE можно использовать для программирования Pico, как провести измерение потребления Pico и программирование контактов ввода/вывода (PIO).

Мы надеемся, что продемонстрированные в книге проекты подготовят вас к будущим приключениям с Raspberry Pi Pico.

Как извлечь максимум пользы из этой книги

Программное/аппаратное обеспечение, описанное в книге	Требования к операционной системе
CircuitPython	Windows, macOS или Linux
MicroPython	Windows, macOS или Linux
Arduino IDE (C/C++)	Windows, macOS или Linux

Проекты, обсуждаемые в этой книге, требуют большого количества разнообразного аппаратного обеспечения. Чтобы сохранить единство повествования, мы использовали Raspberry Pi Pico во всех главах. Вам также понадобится беспроводной модуль ESP32 для подключения к сети. В каждой главе приведен список рекомендуемого оборудования, и мы по возможности перечислили альтернативные варианты. Мы оставляем за вами право заменять компоненты по своему усмотрению.

[4] Фреймворк (framework) – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта. – *Прим. перев.*

> Если вы используете цифровую версию этой книги, мы советуем вам вводить код самостоятельно или получить доступ к коду из репозитория книги на GitHub (ссылка доступна в следующем разделе). Это поможет вам избежать любых потенциальных ошибок, связанных с копированием и вставкой кода.
>
> Если вы обнаружите какие-либо конкретные проблемы, связанные с оборудованием или с примерами кода, размещенными в репозитории, пожалуйста, не стесняйтесь поднимать проблему на GitHub.

Загрузка файлов с примерами кода

Вы можете загрузить файлы с примерами кода для этой книги с GitHub по адресу **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIY-Workshop**. Если код будет обновлен, он будет обновлен в репозитории GitHub.

У нас также есть другие пакеты кода из нашего богатого каталога книг и видео, доступного по адресу **https://github.com/PacktPublishing/**. Загляните туда!

Код в действии

Видеоролики с кодом в процессе работы для этой книги можно посмотреть по адресу **https://bit.ly/3OZJb5Z**.

Загрузите цветные изображения

Мы также предоставляем PDF-файл, содержащий цветные изображения скриншотов и рисунков, используемых в этой книге. Вы можете скачать его здесь: **https://static.packt-cdn.com/downloads/9781801814812_ColorImages.pdf**.

Текстовые соглашения

В этой книге используется ряд текстовых соглашений.

`Код в тексте`: указывает кодовые слова и имена переменных в тексте, имена таблиц базы данных, названия программных библиотек. Пример: «Модуль `board` содержит определения контактов и периферийных устройств, специфичных для платы».

Имена файлов, *имена папок, расширения файлов, пути файловой системы*. Пример: «Файл *rp2040-datasheet.pdf* содержит документацию микроконтроллера RP2040».

Доменные имена и адреса в интернете, если они не представляют собой законченного URL, представляются полужирным курсивом: ***bit.ly***.

Блок кода задается следующим образом:

```
from machine import Pin
import utime
led = Pin(25, Pin.OUT)
while True:
```

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

```
while True:
led.toggle()
utime.sleep(1)
```

Любой ввод или вывод из командной строки записывается следующим образом:

```
>>> print("Hello World")
```

Жирный шрифт: обозначает новый термин или важное слово. Названия в меню или диалоговых окнах также выделены жирным шрифтом. Пример: «Импульс можно создать на вкладке **Create impulse** слева».

Советы или важные примечания

Обозначаются вот так.

Связаться с нами

Обратная связь от наших читателей всегда приветствуется.

Несмотря на то что мы приложили все усилия для обеспечения точности нашего контента, ошибки случаются. Если вы нашли ошибку в этой книге, мы были бы признательны, если бы вы сообщили нам об этом. Если у вас есть вопросы по какому-либо аспекту этой книги, напишите нам по адресу **dmkpress@gmail.com** и укажите название книги в теме вашего сообщения.

Поделитесь своим впечатлением

Когда вы прочитаете «*Raspberry Pi Pico в любительских проектах*», мы будем рады узнать ваше мнение о книге! Пожалуйста, разыщите книгу в каталоге сайта издательства «ДМК Пресс» (**https://dmkpress.com**) и поделитесь своими впечатлениями.

Ваш отзыв важен для нас!

Часть I

Введение в Pico

Цель этой части – представить Raspberry Pi Pico, его варианты и периферийные устройства, доступные на Raspberry Pi Pico. Часть I начинается с проекта мигающего светодиода и описания последовательных интерфейсов, доступных на микроконтроллере RP2040. Затем мы будем продвигаться вперед, работая над простыми проектами домашней автоматизации и садоводства.

Эта часть содержит следующие главы:

- глава 1 *«Начало работы с Raspberry Pi Pico»*;
- глава 2 *«Последовательные интерфейсы и их приложения»*;
- глава 3 *«Проекты домашней автоматизации»*;
- глава 4 *«Весело проводите время в саду!»*.

Глава 1

Начало работы с Raspberry Pi Pico

В этой главе мы хотели бы углубиться в краткое введение в Raspberry Pi Pico и микроконтроллер RP2040. Мы рассмотрим функции Raspberry Pi Pico, периферийные устройства RP2040, дополнительное оборудование для Pico и платы на основе RP2040, разработанные другими производителями. Мы также обсудим варианты языка программирования, доступные для Pico, и дополним главу обсуждением простых примеров «Hello World!», где мы печатаем что-то на экране и мигаем светодиодом.

К концу этой первой главы вы освоитесь с Pico, будете готовы приступить к программированию микроконтроллера RP2040 и начать планировать реализацию проектов из последующих глав этой книги, а также продумывать, как вы можете создавать свои собственные проекты с помощью Raspberry Pi Pico!

Мы собираемся осветить следующие основные темы:

- представление Raspberry Pi Pico и RP2040;
- обсуждение вариантов платы Pico;
- пайка разъемов на плате Pico;
- реализация примера «Hello World!»;
- реализация примера мигания светодиода;
- определение полезного дополнительного оборудования для Pico.

Технические требования

Аппаратное и программное обеспечение, необходимое для этой вводной главы, будет использоваться на протяжении всей книги. В следующих главах мы также представим дополнительные или специфические для главы требования к оборудованию.

Требования к оборудованию для этой главы:

- ноутбук или персональный компьютер с портом **универсальной последовательной шины (USB)**;
- *дополнительно:* паяльное оборудование, включая электропаяльник, припой, защитные очки и разное оборудование;

- *дополнительно*: беспаечная макетная плата и комплект соединительных проводов.

Видеоролики с кодом в действии для этой главы можно посмотреть по адресу **https://bit.ly/3MRdYjx**.

Представление Raspberry Pi Pico и RP2040

Raspberry Pi Pico – это новый одноплатный микроконтроллер, представленный компанией Raspberry Pi Foundation. Плата микроконтроллера Pico, несмотря на невысокую цену, обладает неплохими характеристиками. Pico построен на RP2040, двухъядерном микроконтроллере с ядром Cortex-M0+. Плата поставляется в общей сложности с 40 контактами, по 20 контактов с каждой стороны, как показано на рис. 1.1. Pico поставляется с 2 МБ встроенной флеш-памяти и светодиодом на выводе GP25. GP (или полностью **GPIO**) означает **General Purpose Input/Output** – контакт ввода/вывода общего назначения.

Рис. 1.1. Raspberry Pi Pico

Спецификация для Raspberry Pi Pico доступна по адресу: **https://bit.ly/3cwv1lc**. В проектах, изложенных в этой книге, мы будем использовать различные периферийные устройства, доступные на Pico. Удобно распечатать разводку выводов, предоставленную Raspberry Pi Foundation (источник: **https://bit.ly/3wa0nwq**). Схема разводки может помочь с выбором выводов при планировании проекта. На рис. 1.2 показана схема разводки, представленная Adafruit Industries.

Плата Pico может использоваться в различных приложениях, связанных с роботами, дистанционным мониторингом, гражданской наукой и т. д. В этой книге мы познакомим вас с различными примерами применения, включающими в том числе и периферийные устройства микроконтроллера RP2040.

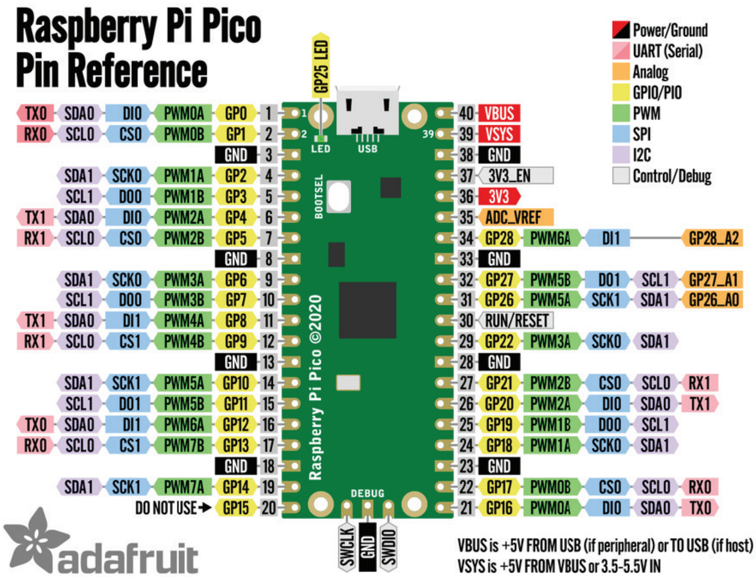


Рисунок 1.2. Разводка выводов Raspberry Pi Pico
(источник: Adafruit Industries)

Микроконтроллер RP2040

RP2040 представляет собой двухъядерный микроконтроллер ARM Cortex-M0+ с 264 килобайтами (КБ) статической оперативной памяти (SRAM), но не имеет встроенной флеш-памяти. RP2040 включает в себя целый ряд периферийных устройств: двухпроводной последовательный интерфейс (I2C), последовательный периферийный интерфейс (SPI), универсальный асинхронный интерфейс приемника-передатчика (UART) и контакты с программируемым вводом/выводом (PIO). Выводы PIO микроконтроллера RP2040 позволяют вам создавать свой собственный интерфейс, такой как дополнительный UART или видеоинтерфейс. В главе 12 «*Дополнительные возможности работы с Pico*» мы обсудим использование PIO для подключения периферийных устройств.

Вот список ресурсов для более подробного изучения RP2040:

- технические характеристики RP2040 доступны по следующей ссылке:
  https://bit.ly/3rw41x5;
- техническое описание Raspberry Pi Pico доступно по следующей ссылке:
  https://bit.ly/3cwv1lc;
- видео от Raspberry Pi foundation о PIO в микроконтроллере RP2040 можно найти по следующей ссылке:
  https://bit.ly/39ni6Xg;
- документацию по RP2040 и Raspberry Pi Pico от Raspberry Pi Foundation можно найти по следующей ссылке:
  https://bit.ly/3flFLv9.

Мы рекомендуем вам загрузить технические описания Pico и RP2040. Они пригодятся в качестве справочного материала при разработке, и в определенных местах этой книги мы будем отсылать вас к этим документам для получения дополнительной информации.

О приобретении компонентов в России

Приобрести Raspberry Pi Pico и его варианты, а также комплектующие, необходимые для выполнения проектов из книги, в России можно в магазинах сети «Чип и Дип», в российских интернет-магазинах, торгующих электронными компонентами (например, **iarduino.ru**; **electronshik.ru**; **amperka.ru** и др.), а также на AliExpress. В этой главе далее и некоторых других разделах книги встречаются ссылки на производителей и интернет-магазины, доступные в США и Европе, но не работающие с Россией (например, **adafruit.com** или **digikey.com**). В большинстве случаев подобная ссылка сопровождается сноской с указанием российского источника напрямую. Если она отсутствует, то указанные там товары или их аналоги следует самостоятельно поискать в отечественных интернет-магазинах или через специальных посредников по приобретению.

Обзор вариантов платы Pico

С момента запуска Raspberry Pi Pico было выпущено несколько вариантов платы на основе RP2040 от различных компаний, выпускающих свободное аппаратное обеспечение. Опишем более подробно некоторые из них.

- **SparkFun Thing Plus – RP2040**: это плата с открытым исходным кодом от SparkFun (**https://bit.ly/2NS5vUn**). Thing Plus выпускается в форм-факторе **Feather** от Adafruit[5]. Уникальность этой платы заключается в том, что она поставляется с держателем карты microSD и индивидуально адресуемым RGB-светодиодом. На рис. 1.3 представлена плата RP2040 Thing Plus с нижней стороны.

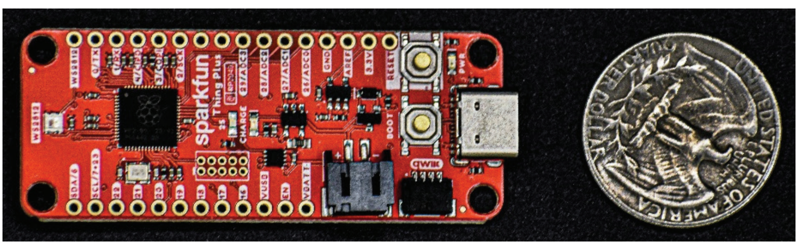


Рисунок 1.3. SparkFun Thing Plus (RP2040)

5 Форм-фактор Feather – популярная типовая конфигурация платы размерами 60×23 мм (2,38×0,9 дюйма), разработанная компанией Adafruit. В форм-факторе Feather выпускается множество любительских электронных устройств по всему миру (среди самых популярных, например, ESP 8266 и ESP 32). Подробнее об этом стандарте см. **https://learn.adafruit.com/adafruit-feather/feather-specification**. – *Прим. перев.*

- Процессор **SparkFun MicroMod RP2040**. Это еще один вариант от SparkFun (**https://bit.ly/3clp0hG**). Он поставляется с 16 МБ встроенной флеш-памяти в форм-факторе MicroMod, использующем стандарт M.2[6]. На рис. 1.4 вы можете увидеть верхнюю сторону платы. Обратите внимание на выемку в форме полумесяца, которая используется для крепления модуля к несущей плате с помощью винта M2.5.

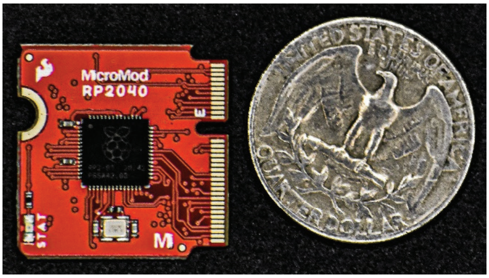


Рисунок 1.4. Процессор MicroMod RP2040

SparkFun также производит различные несущие платы для экосистемы MicroMod. Например, несущая плата (**https://bit.ly/3cnlrHF**), показанная на следующем рисунке, была разработана для управления дисплеем с интерфейсом **HDMI** (**high-definition multimedia interface**, *мультимедийный интерфейс высокой четкости*) с использованием RP2040.

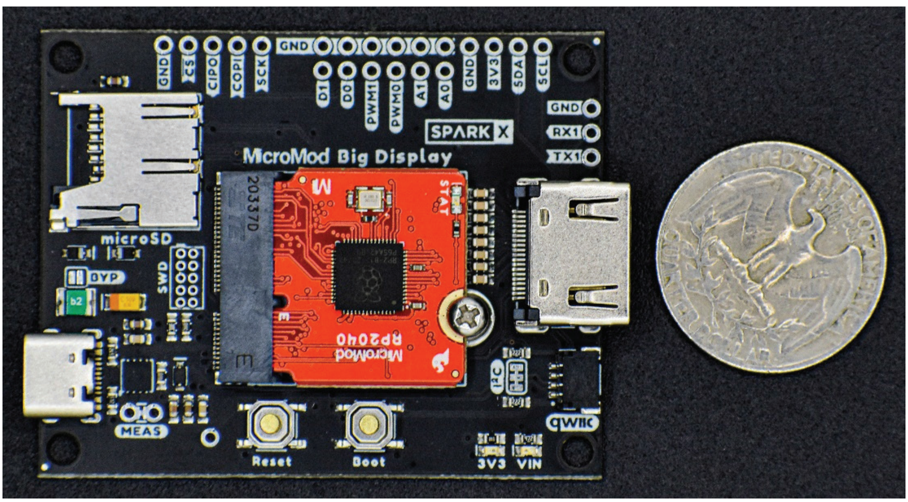


Рисунок 1.5. Большая дисплейная плата MicroMod для процессора RP2040

- **SparkFun Pro Micro – RP2040**. Плата Pro Micro – RP2040 (**https://bit.ly/3cnhVgH**) – это вариант, который принадлежит к относительно небольшой экосистеме семейства плат Pro Micro. На плате размещается 16 МБ

[6] Формат M.2 – спецификация компьютерных карт расширения и их разъёмов, расширяющая возможности PCI Express в сторону большей компактности. Плата стандарта M.2 имеет ножевой разъем, аналогичный PCI Express, но с уменьшенным шагом 0,5 мм (см. рис. 1.4). – *Прим. перев.*

флеш-памяти, индивидуально адресуемые светодиоды RGB и универсальные зубчатые площадки выводов, которые позволяют либо установить разъемы, либо припаивать модуль непосредственно к другой печатной плате. Универсальные контакты Pro Micro показаны на рис. 1.6.

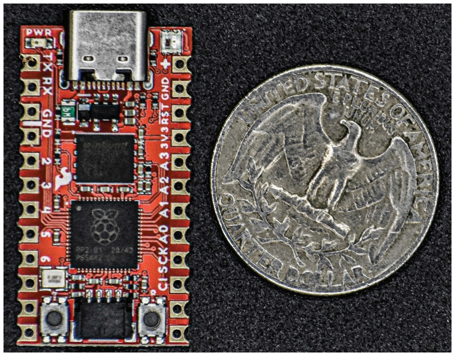


Рисунок 1.6. SparkFun Pro Micro – RP2040

- **Pimoroni Tiny 2040**. Эта плата от Pimoroni (**https://bit.ly/3d9f7Tf**) размером примерно с четверть дюйма имеет 8 МБ флеш-памяти и RGB-светодиод. Универсальные зубчатые площадки выводов позволяют припаять его либо к разъему, либо непосредственно к пользовательской печатной плате. Мы должны отметить, что вам понадобится вырез в вашей плате, чтобы припаять таким образом модуль. Это связано с тем, что микроконтроллер в плате Tiny 2040 находится на нижней стороне, как показано на рис. 1.7. Мы далее проиллюстрируем установку этой платы на пользовательскую печатную плату.

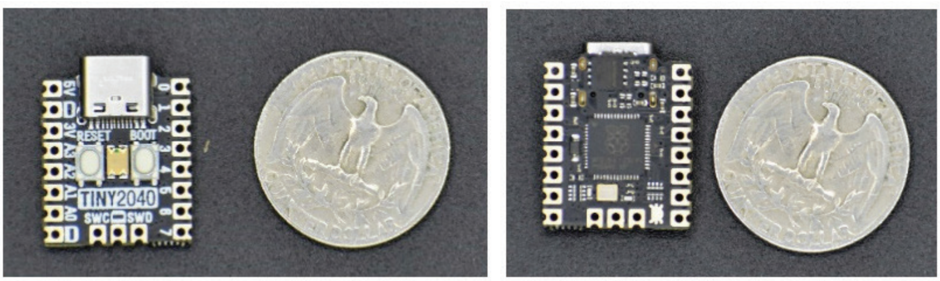


Рисунок 1.7. Pimoroni Tiny 2040

- **Adafruit Feather RP2040**. Как следует из названия, эта плата от Adafruit (**https://bit.ly/3cm3tW0**) представляет собой печатную плату с микроконтроллером RP2040, несущую 8 МБ флеш-памяти. Как и в случае с SparkFun Thing Plus, она отлично сочетается с разъемами Qwiic/STEMMA[7]. Плата показана на рис. 1.8.

[7] Qwiic и STEMMA – стандарты разъемов для подключения внешних модулей от компаний SparkFun (Qwiic) и Adafruit (STEMMA, см. также далее). Другие известные в нашей стране подобные стандарты – Grove (Seeed Studio), отечественные Trema (**iarduino.ru**) и Troyka (**amperka.ru**). – *Прим. перев.*

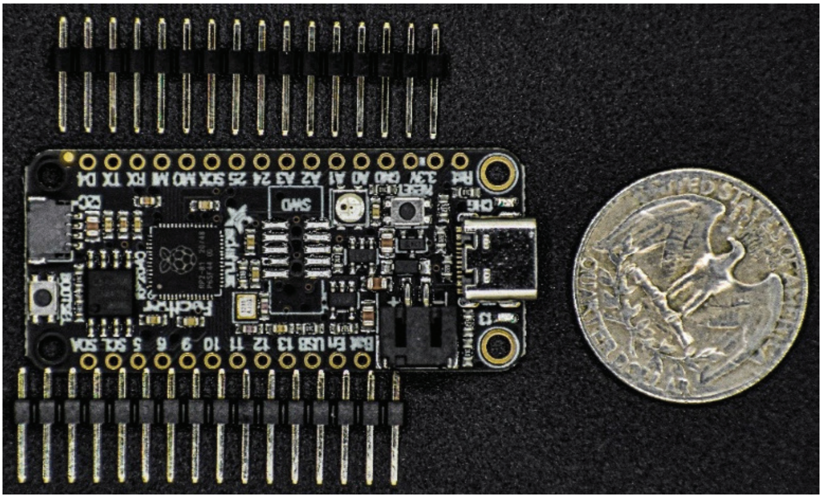


Рисунок 1.8. Adafruit Feather RP2040

- **Adafruit ItsyBitsy RP2040**. Эта плата от Adafruit (**https://bit.ly/3sqdB5R**) является дополнением к их линейке продуктов Itsy Bitsy. С точки зрения разводки выводов она идентична другим продуктам Itsy Bitsy от Adafruit. Эта плата поставляется с 8 МБ встроенной флеш-памяти. Вариант Itsy Bitsy, показанный на следующем рисунке, удобен для макетирования, что позволяет встроить плату в ваш проект.

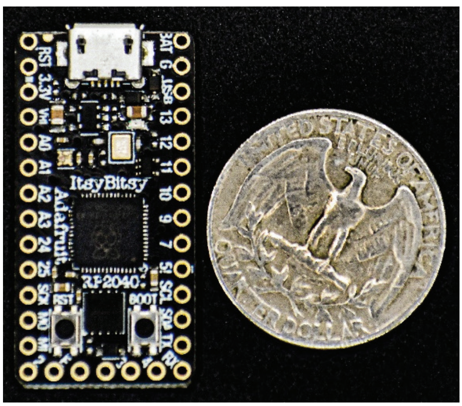


Рисунок 1.9. Adafruit ItsyBitsy RP2040

- **Adafruit QT Py RP2040**. Эта плата (**https://bit.ly/3lU2O1q**) является дополнением к семейству продуктов **QT Py** от Adafruit. Она также поставляется с 8 МБ встроенной флеш-памяти. Зубчатые контакты модуля, показанные на рисунке ниже, позволяют спроектировать несущую плату, с помощью которой плата QT Py RP2040 может быть встроена в ваш дизайн. Поскольку микроконтроллер RP2040 расположен на нижней стороне, вам необходимо убедиться, что в вашей конструкции есть вырез для размещения QT Py.

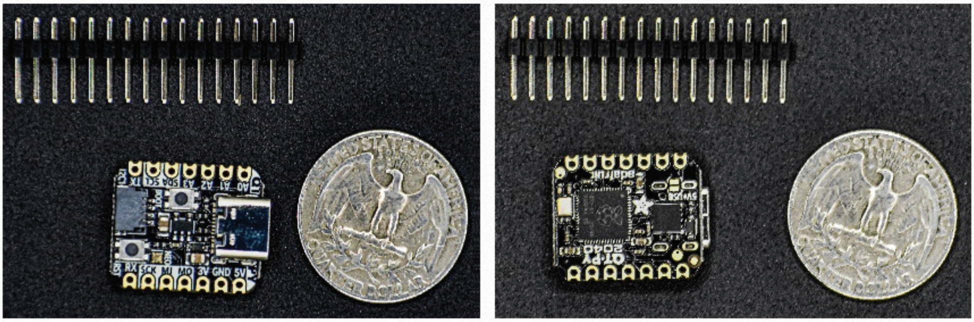


Рисунок 1.10. Adafruit QT Py RP2040

Варианты, которые мы обсуждали здесь, не исчерпывают все разнообразие плат на основе RP2040, но мы хотели представить некоторый выбор для начала работы с микроконтроллером. Например, если вы знакомы с форм-фактором Feather, то можете начать с платы Thing Plus от SparkFun или платы Feather от Adafruit, которые рассматриваются в этом разделе. Вы можете использовать любую плату по вашему выбору, но режим использования и интерфейс могут отличаться в зависимости от варианта. Мы постараемся подчеркнуть различия, где это возможно.

Пайка соединительного разъема Pico

В этом разделе мы рассмотрим настройку Pico для предстоящих проектов. Это включает в себя пайку штыревого разъема для внешних соединений и, возможно, изготовление корпуса с помощью 3D-принтера.

Пайка разъема

Pico выпускается с 40 отверстиями для разъема в два ряда по 20 контактов. Нам нужно припаять разъем, чтобы получить доступ к периферийным устройствам микроконтроллера RP2040 для нашего проекта.

> **Важное замечание**
>
> Пайка разъемов требует предварительной подготовки и присмотра взрослых. Не пытайтесь выполнять пайку без предварительной подготовки. Здесь вы можете просмотреть учебное пособие по пайке:
>
> **https://electricsexpert.ru/kak-pravilno-payat-payalnikom/**.

Вы можете приобрести разъемы из того же источника, что и Pico[8]. Шаги, которые необходимо выполнить для пайки, включают следующее:

[8] Требуется два штыревых разъема на 20 прямых контактов с шагом 2,54 мм (они показаны на рис. 1.8 и 1.10). Официальное название разъема – PLS-20. Если разъемы не входят в комплект поставки платы Raspberry Pi Pico, то в России их можно приобрести из источников, указанных в примечании на стр. 23. – *Прим. перев.*

1. Проще всего припаять разъемы с помощью макетной платы. Разместите их на макетной плате и уложите поверх нее Pico, как показано на рис. 1.11.

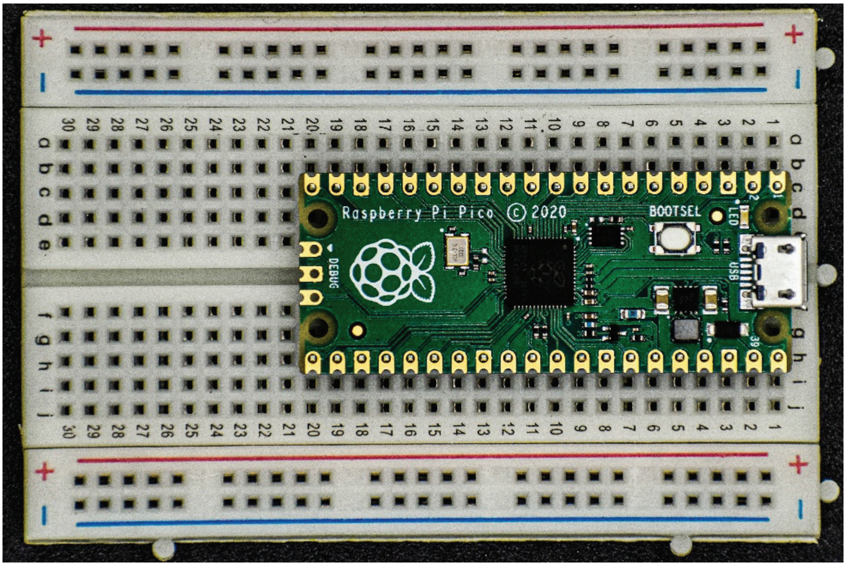


Рисунок 1.11. Pico с разъемами на макетной плате

2. Если вы будете паять отдельные контакты слишком медленно, то это может привести к повреждению макетной платы из-за избыточного тепла. На следующем рисунке показан Pico с запаянными контактами:

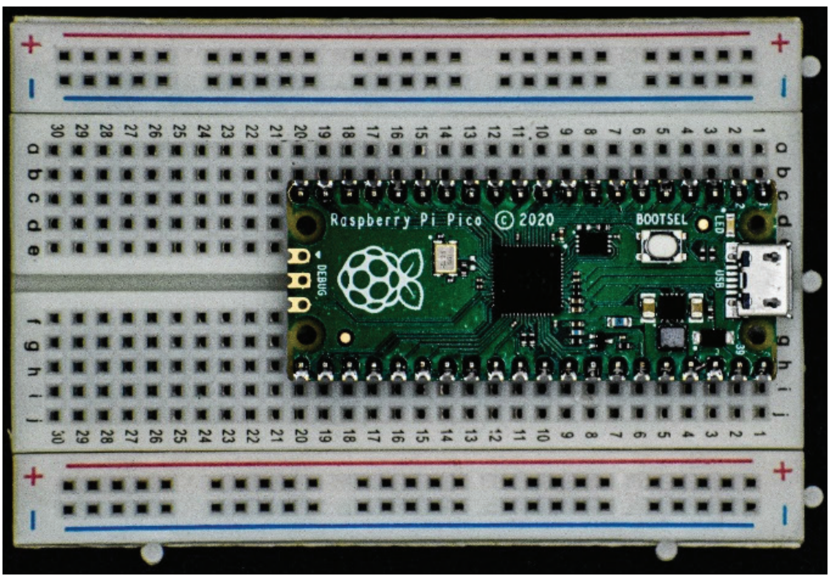


Рисунок 1.12. Pico на макетной плате

Теперь, когда мы припаяли разъемы, мы готовы попробовать. В следующем разделе мы рассмотрим необязательный шаг добавления кнопки сброса для Pico.

Реализация примера «Hello World!»

На любом языке программирования первое упражнение по кодированию состоит в том, чтобы вывести на экран «Hello World». Мы обсудим этот пример с использованием MicroPython и CircuitPython.

Кнопка сброса для Pico (дополнительно)

Pico не содержит кнопку сброса. Чтобы сбросить Pico, вам придется отсоединить и снова подключить USB-кабель, что может быть утомительно. Это можно преодолеть, добавив кнопку сброса между выводами платы 28 (GND) и 30 (Reset), как показано на следующем рисунке (вы можете получить такую кнопку по этой ссылке: **https://bit.ly/3w8AmO2**)[9]:

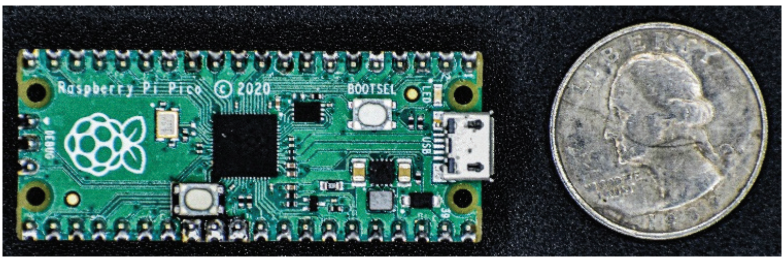


Рисунок 1.13. Кнопка сброса на Pico

Кнопка сброса упрощает перезапуск программы во время разработки. Добавление кнопки сброса необязательно, и не рекомендуется к этому приступать, если вы не чувствуете себя свободно в приемах пайки компонентов.

В следующем разделе мы напишем первую программу для Pico.

MicroPython

В этом примере мы будем использовать **Thonny IDE** (IDE означает **integrated development environment** – интегрированная среда разработки). Thonny IDE можно загрузить с **https://thonny.org**. Она доступна для операционных систем Windows, Mac и Linux.

Чтобы можно было сразу приступать к программированию Raspberry Pi, он выпускается с предустановленным Thonny. Вы можете запустить Thonny из **Menu** | **Programming** (*Меню* | *Программирование*) и прокрутить вниз раскрывающийся список, чтобы найти **Thonny Python IDE**, как показано на рис. 1.14.

[9] Вместо экзотической кнопки стоимостью, как указано по ссылке, почти два евро, можно применить обычную тактовую кнопку стоимостью около 10 рублей, доступную в источниках согласно примечанию на стр. 23. Ее не припаивают к плате Pico, а устанавливают на макетную плату между контактами платы 28 и 30. – *Прим. перев.*

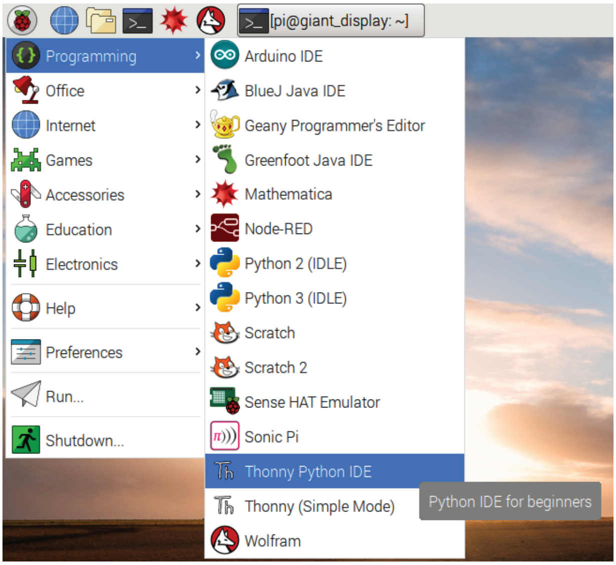


Рисунок 1.14. Расположение Thonny IDE на рабочем столе Raspberry Pi

Теперь давайте подготовим Pico, прошив двоичный файл MicroPython.

Прошивка двоичного файла MicroPython

Для прошивки файла Pico должен быть подключен к компьютеру определенным образом. Шаги по прошивке двоичного файла следующие.

1. Первый шаг – загрузить двоичный файл с **https://bit.ly/31nBMFW**.
2. Второй шаг: нажмите и удерживайте кнопку загрузки, показанную на рис. 1.15, при подключении кабеля microUSB к Pico. Убедитесь, что другой конец USB-кабеля подключен к компьютеру.

Рисунок 1.15. Кнопка загрузки Pico BOOTSEL

3. Pico должен появиться на вашем компьютере как запоминающее устройство, которое может выглядеть следующим образом:

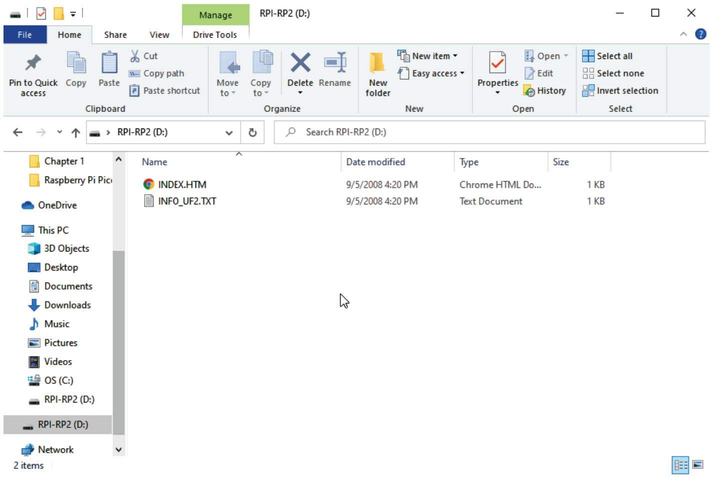


Рисунок 1.16. Raspberry Pi Pico как запоминающее устройство

4. Затем скопируйте двоичный файл на накопитель. Pico сбросит сам себя, и мы готовы писать программы на MicroPython.

Написание первой программы

Давайте запустим Thonny и отправимся на тест-драйв! Вам нужно будет предпринять следующие шаги.

1. Запустите Thonny, и вы должны увидеть окно, как показано на рис. 1.17.

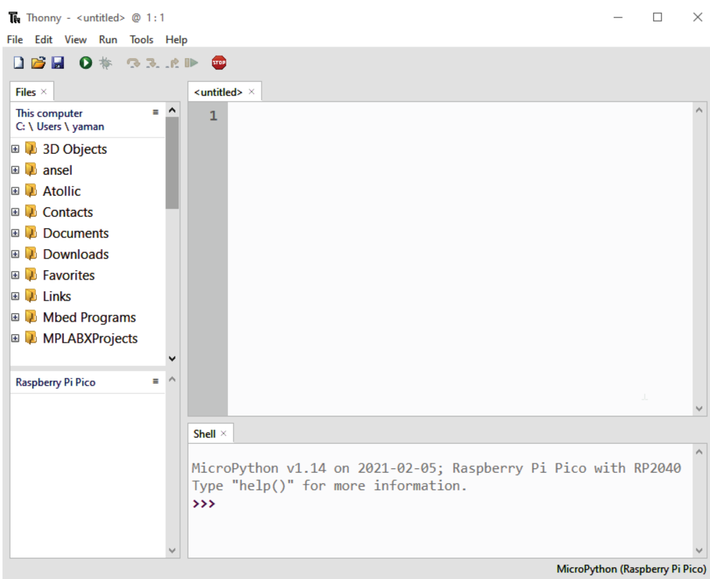


Рисунок 1.17. Thonny IDE

2. Мы собираемся запустить нашу первую программу, используя интерпретатор, работающий на вашем Pico. Интерпретаторы Python позволяют тестировать код по мере его написания. Перейдите в раздел **Run** (*Выполнить*), а затем нажмите кнопку **Select Interpreter** (*Выбрать интерпретатор*).

   Здесь выберите следующие параметры, как показано на рис. 1.18:

   - **MicroPython (Raspberry Pi Pico)**;
   - **< Try to detect port automatically >** (*Попробуйте автоматически определить порт*).

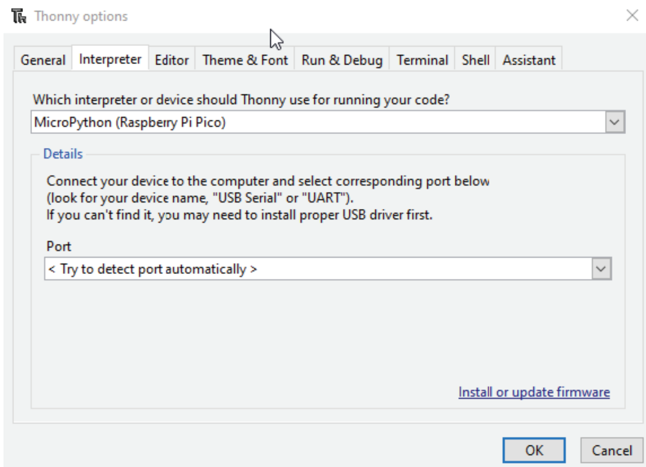


Рисунок 1.18. Выбор интерпретатора Python

3. Теперь должен загрузиться интерпретатор Python (см. следующий рисунок). Это интерпретатор MicroPython, работающий на Pico. Давайте проведем тест-драйв.

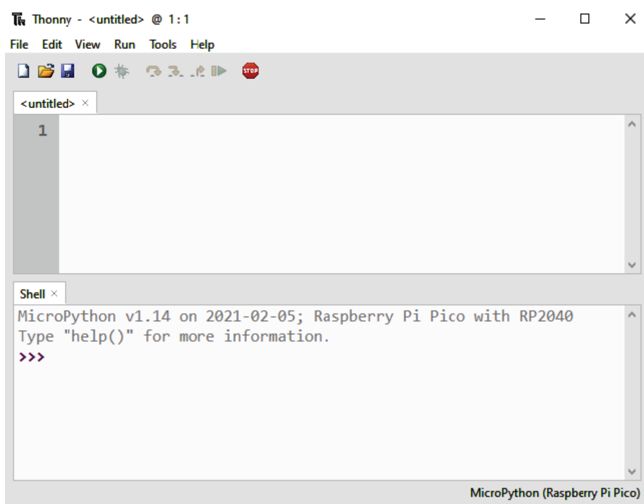


Рисунок 1.19. Интерпретатор Python

4. В командной строке после знака приглашения >>> введите следующую строку кода:

```
>>> print("Hello World")
```

Вы должны увидеть следующий результат:

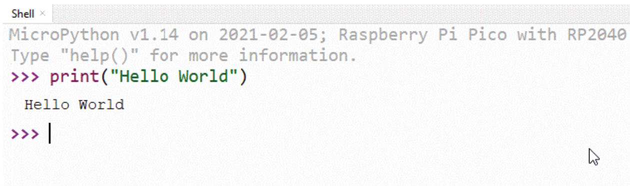


Рисунок 1.20. Результаты работы интерпретатора

Вы только что закончили писать свою первую программу на Raspberry Pi Pico! В следующем примере мы напишем сценарий, который начнет выполняться непрерывно, когда Pico питается от USB-кабеля.

Реализация примера мигания светодиода

В предыдущем разделе мы использовали интерпретатор для написания нашей программы. Интерпретатор может быть полезен при тестировании кода или получении дополнительной информации об импортируемых модулях. В этом примере мы обсудим написание скрипта, который автоматически запускается при включении Pico. Мы обсудим пример начала работы с электроникой в программе, аналогичной «Hello World», которая будет мигать светодиодом с интервалом в 1 секунду.

Pico поставляется с зеленым светодиодом на плате, и его расположение показано на рис. 1.21.

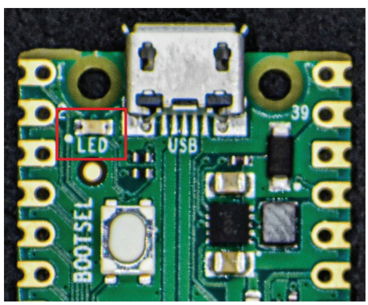


Рисунок 1.21. Расположение светодиода на плате Pico

Мы заставим этот светодиод мигать с интервалом в 1 секунду, то есть будем каждую секунду его включать и выключать. Для того чтобы написать эту программу, нам понадобятся модули `machine` и `utime`.

Согласно спецификации Pico, встроенный светодиод подключен к универсальному вводу-выводу (**general-purpose I/O, GPIO**) номер 25. Мы будем действовать следующим образом:

- для управления светодиодом используем класс `Pin` из модуля `machine`;
- для введения задержки между включением и выключением светодиода используем модуль `utime`.

Теперь вам нужно будет выполнить такие действия.

1. Давайте взглянем на следующий пример кода:

```
from machine import Pin
import utime

led = Pin(25, Pin.OUT)

while True:
led.toggle()
utime.sleep(1)
```

> **Упражнение**
>
> Выработайте практику активного использования интерпретатора во время написания кода. В интерпретаторе импортируйте модули `machine` и `utime`, попробуйте выполнить `help(machine)`, `help(utime)` и разберитесь в их возможностях.

2. Создайте новый файл и введите фрагмент кода, показанный на шаге 1. Установите место для размещения файла *Raspberry Pi Pico*, как показано на следующем рисунке:

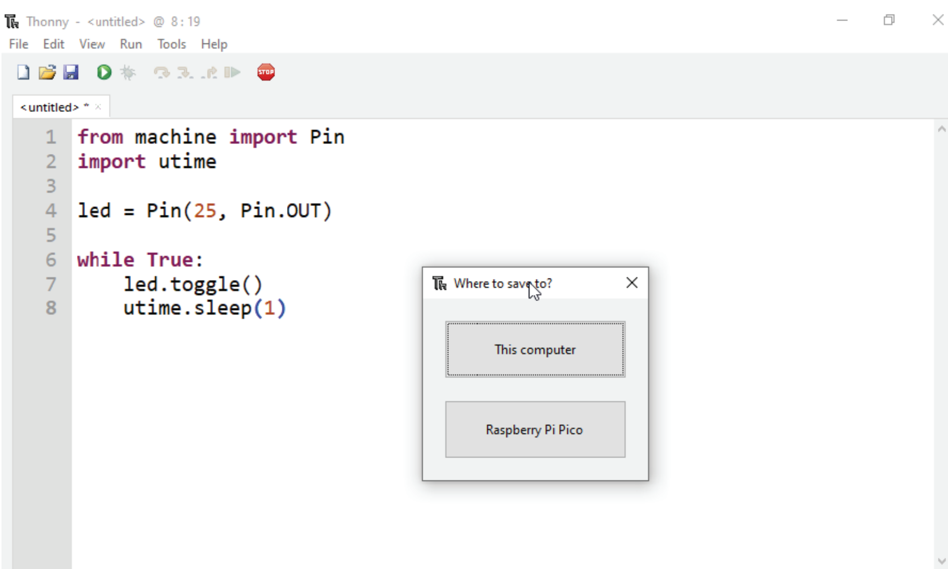


Рисунок 1.22. Расположение файла

3. Сохраните файл как *main.py*, и ваш код должен автоматически начать выполняться.
4. Попробуйте отсоединить USB-кабель и снова подключить его. Вы заметите, что скрипт запускается автоматически.
5. Чтобы остановить выполнение, нажмите на кнопку **STOP**, расположенную на панели инструментов, как показано на следующем рисунке:

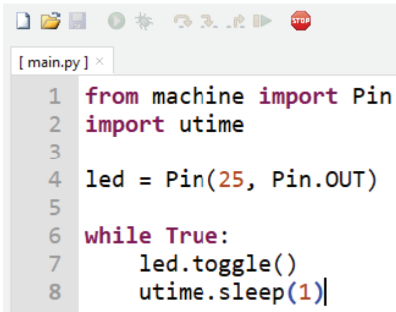


Рисунок 1.23. Кнопка **STOP** для прерывания выполнения

6. Чтобы возобновить выполнение, нажмите кнопку **Run Current Script** (*Запустить текущий сценарий*), на которую указывает зеленая кнопка воспроизведения, указанная в верхней части страницы, как показано на следующем рисунке:

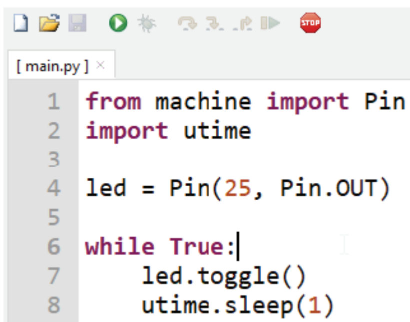


Рисунок 1.24. Кнопка **Run Current Script** для запуска выполнения скрипта

В следующем разделе мы более подробно рассмотрим код.

Описание примера кода

В этом разделе мы подробно рассмотрим пример кода, показанный ранее.

1. Мы начинаем с импорта модуля `utime` и класса `Pin` из модуля `machine` следующим образом:

```
from machine import Pin
import utime
```

> **GPIO**
>
> Вывод GPIO можно использовать в качестве входного или выходного вывода. Когда вы используете вывод GPIO в качестве выходного вывода, вы можете установить его в `HIGH` (высокий) или `LOW` (низкий) уровень. Аналогично, когда вы используете его в качестве входного вывода, вы можете определить, имеет ли сигнал на входе `HIGH` (высокий) или `LOW` (низкий) уровень. В этом примере мы включаем/выключаем светодиод, чередуя состояния `HIGH` и `LOW`.

2. Далее мы объявляем вывод 25 GPIO в качестве выходного вывода, используя объект, принадлежащий классу `Pin`. Первый аргумент в следующем фрагменте кода ссылается на номер вывода 25, в то время как второй аргумент устанавливает вывод в качестве выходного:

```
led = Pin(25, Pin.OUT)
```

3. Теперь нам нужно мигать светодиодом с интервалом в 1 секунду. Мы собираемся сделать это внутри бесконечного цикла.
4. Объект `led` имеет функцию переключения `toggle()`, которая переключает вывод между включенным (ON) и выключенным (OFF) состояниями.
5. Мы введем задержку в 1 секунду, вставив функцию ожидания после очередного переключения следующим образом:

```
while True:
    led.toggle()
    utime.sleep(1)
```

Поздравляем вас с вашим первым шагом с Raspberry Pi Pico! Теперь мы обсудим тот же пример в CircuitPython.

Пример CircuitPython

В этом разделе мы обсудим использование **CircuitPython** для программирования Pico на том же примере мигания светодиода. Мы также установим среду *Mu IDE* в программу с помощью CircuitPython.

Прошивка двоичного кода CircuitPython

В этом разделе мы будем прошивать двоичный файл CircuitPython на Pico. Процесс установки такой же, как и для MicroPython. Двоичный файл можно загрузить с сайта **https://bit.ly/31pnLI4**.

После того как вы загрузили двоичный файл, следуйте инструкциям из раздела «*MicroPython*».

Кодирование с помощью Mu

В этом разделе мы установим среду разработки Mu IDE. Ее можно загрузить с **https://bit.ly/3ruxDKW**. Mu IDE доступна для операционных систем Windows, Linux и Mac.

Установка на Raspberry Pi

На Raspberry Pi Mu IDE можно установить следующим образом:

1. Перейдите в **Menu**|**Preferences**|**Recommended Software**|**Select Mu** (*Меню*|*Настройки*|*Рекомендуемое программное обеспечение*|*Выбрать Mu*), чтобы выбрать Mu из списка устанавливаемого программного обеспечения.
2. Mu IDE можно запустить из **Menu** | **Programming** | **Mu** (*Меню* | *Программирование* | *Mu*).

В следующем разделе мы рассмотрим написание нашей первой программы с помощью Mu.

Запуск Mu

Шаги по написанию программы с Mu включают в себя следующее.

1. Подключите Pico к компьютеру / Raspberry Pi с помощью USB-кабеля.
2. Как только Mu будет запущен, нам нужно установить режим программирования. Поскольку мы программируем на CircuitPython, мы установим для него значение **CircuitPython**, как показано на следующем рисунке:

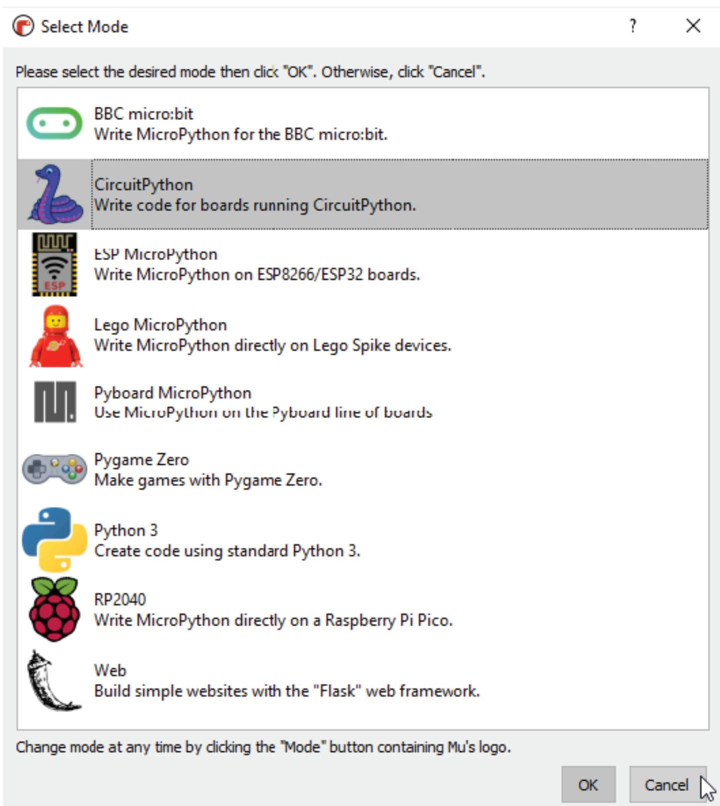


Рисунок 1.25. Выбор режима программирования

3. Вы также можете вернуться к установке режима программирования из главного окна IDE, как показано на следующем рисунке:

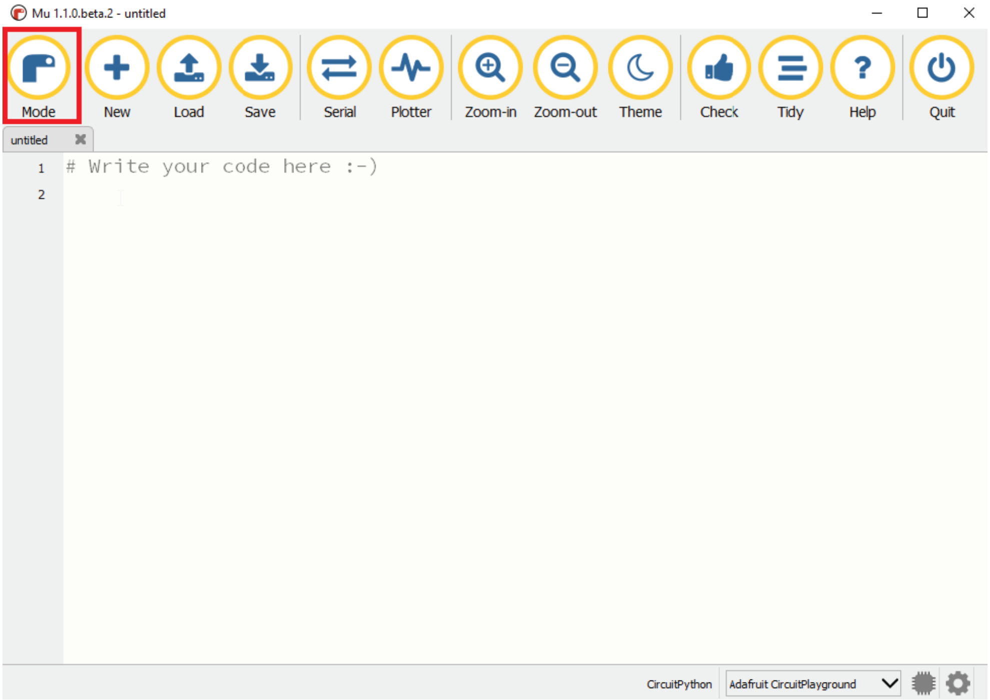


Рисунок 1.26. Изменение режима Python

Обратите внимание, что на рис. 1.25 Pico обнаруживается автоматически.

4. Нажмите кнопку **Serial**, чтобы запустить доступ к интерпретатору Python, как показано здесь:

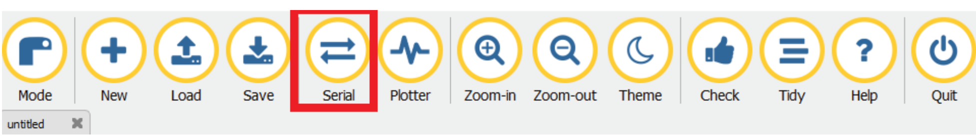


Рисунок 1.27. Расположение кнопки **Serial**

5. Запустите интерпретатор Python, нажав <**Ctrl**+**C**>, и он должен привести вас к приглашению REPL >>>, как показано в нижней части следующего рисунка:

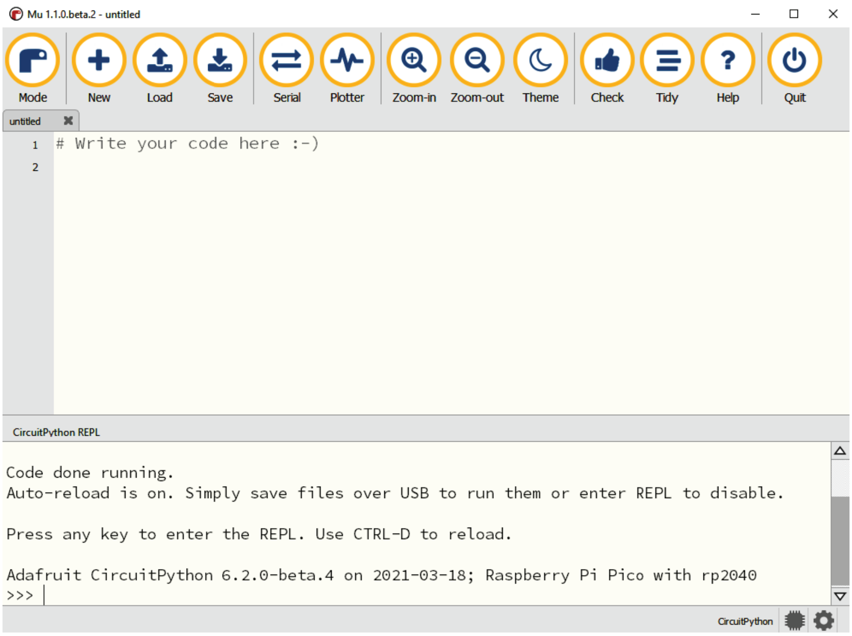


Рисунок 1.28. Интерпретатор CircuitPython на Pico

Наконец, в этой среде повторите пример «Hello World» из раздела MicroPython с помощью интерпретатора.

Второй пример мигания светодиода

Мы обсудим тот же пример мигания светодиода с использованием CircuitPython и рассмотрим различия между двумя вариантами Python. Взглянем на следующий фрагмент кода:

```
import time
import board
import digitalio

led = digitalio.DigitalInOut(board.LED)
   led.direction = digitalio.Direction.OUTPUT

   while True:
   led.value = True
   time.sleep(1)
   led.value = False
   time.sleep(1 )
```

Давайте рассмотрим пример кода следующим образом:

1. Мы начинаем с импорта модулей `time`, `board` и `digitalio`. Модуль времени `time` используется для введения задержки между включением и выключением светодиода.

2. Модуль платы `board` содержит определения контактов и периферийных устройств, специфичных для выбранной платы. В этом примере мы используем из модуля `board` константу `LED` для управления встроенным светодиодом на Raspberry Pi Pico.
3. Модуль `digitalio` обеспечивает доступ к периферийным устройствам Pico. В этом примере нам нужно объявить вывод светодиода (вывод GPIO 25) в качестве выходного вывода:

```
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT
```

4. В первой строке предыдущего фрагмента кода мы объявляем `led` экземпляром класса `DigitalInOut`.
5. Во второй строке мы устанавливаем вывод светодиода в качестве выходного вывода.
   Мы используем класс `Direction` из модуля `digitalio`.
6. Далее мы входим в бесконечный цикл, в котором включаем/выключаем светодиод следующим образом:

```
while True:
    led.value = True
    time.sleep(1)
    led.value = False
    time.sleep(1 )
```

7. В первой строке цикла `while` мы для `led.value` устанавливаем значение `True`. Это включает светодиод. За этим следует задержка в 1 секунду, что достигается вызовом `time.sleep(1)`.
8. В третьей строке цикла `while` мы устанавливаем для `led.value` значение `False`. При этом индикатор гаснет. За этим также следует задержка в 1 секунду.
9. Мы хотим, чтобы скрипт запускался после сброса. Загрузите файл *code.py*, расположенный на Pico, в данный момент указанном на вашем компьютере в качестве устройства хранения. Кнопка *Load* (*Загрузка*) расположена на верхней панели инструментов.
10. Введите пример кода, который мы обсуждали, в *code.py* и сохраните его.
11. Нажмите <**Ctrl**+**D**> в интерпретаторе CircuitPython, и вы должны заметить мигание светодиода на Pico.

Поздравляем вас с написанием вашей первой программы CircuitPython для Raspberry Pi Pico!

CircuitPython или MicroPython?

В этой главе мы рассматривали примеры как с CircuitPython, так и с MicroPython. Примеры в чем-то идентичны и имеют схожую структуру. В чем их различия и какой вариант Python вы должны использовать для своей разработки?

Короткий ответ заключается в том, что это зависит от вас. Для единообразия мы будем обсуждать все примеры в CircuitPython с использованием Mu IDE.

Обе реализации имеют широкую пользовательскую базу и библиотеки для дополнительного оборудования. CircuitPython был выделен из MicroPython компанией Adafruit. CircuitPython может быть полезен при использовании интерфейсных модулей от Adafruit Industries.

Thonny и Mu IDE

В этой главе вы, возможно, заметили, что мы использовали редактор Thonny для примера MicroPython и Mu для примера CircuitPython. Мы хотели продемонстрировать различные инструменты, доступные для Raspberry Pi Pico. Вы даже можете использовать простой текстовый редактор для своей разработки. Мы покажем вам, как сохранить и загрузить ваш код в Pico.

В следующем разделе мы обсудим дополнительное оборудование для Pico.

Подключение полезного дополнительного оборудования для Pico

В этом разделе мы покажем варианты решений для выполнения макетов с дополнительным оборудованием, выпускаемые для Pico. Отметим, что мы обсуждаем только комплекты, разработанные специально для Pico. Этот список не является исчерпывающим, и мы выбрали ряд примеров, руководствуясь их различиями. Вы можете выбрать любой вариант по вашему выбору, но сразу скажем, что это необязательно. Простой беспаечной платы может быть достаточно, и макет будет выглядеть, как показано на рис. 1.29.

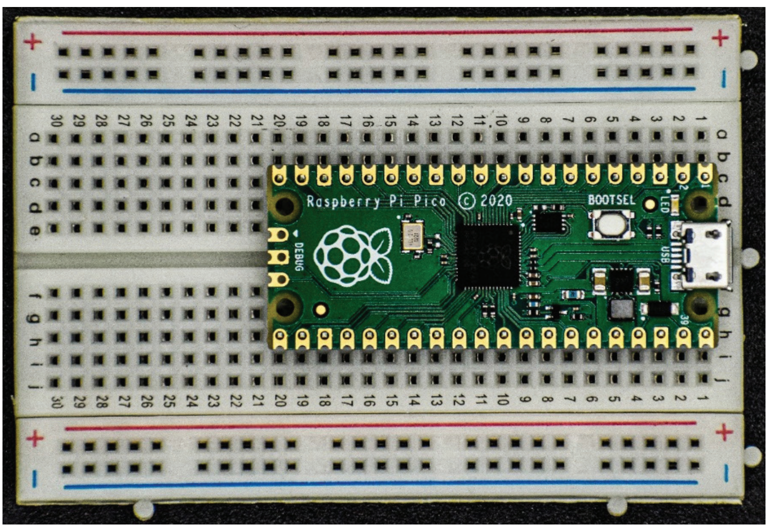


Рисунок 1.29. Pico на макетной плате

Далее мы рассмотрим варианты, в которых Pico поставляется в комплекте с различными макетными платами, облегчающими подключение.

Pico Breadboard Kit (набор Pico с макетной платой)

Как следует из названия, здесь Pico поставляется с макетной платой, а также с четырьмя светодиодами, четырьмя кнопками и зуммером. Имеется пара разъемов для установки Pico на плате и два ряда разъемов для доступа ко всем контактам, имеющимся на плате Pico. Изображение комплекта можно увидеть на рис. 1.30. Комплект можно приобрести по адресу **https://bit.ly/3tV7ala**.

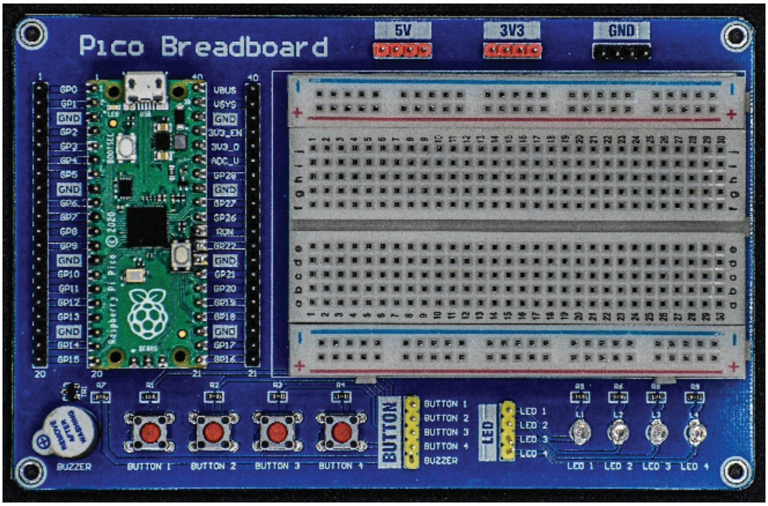


Рисунок 1.30. Pico Breadboard Kit

Pico Breadboard Kit может быть полезен абсолютному новичку в электронике.

Pico GPIO Expansion Board (плата расширения выводов GPIO Pico)

Этот комплект обеспечивает доступ ко всем выводам Pico. С обеих сторон есть два ряда штыревых и гнездовых разъемов, как показано на рис. 1.31. Эту плату можно приобрести по адресу **https://bit.ly/3rprobq**.

Два ряда штыревых и гнездовых разъемов позволяют использовать соответствующие соединительные кабели для создания прототипов.

Pico HAT Expansion (расширение Pico HAT)

Эта плата позволяет подключать любую плату Raspberry Pi **HAT** (что означает **Hardware Attached on Top** – аппаратура, устанавливаемая сверху) к Pico. В комплекте поставляется разъем 2×20, который позволяет подключать к плате оборудование HAT (см. рис. 1.32 далее). Плата также обеспечивает доступ к выводам платы Pico, которые подключены к выводам разъема HAT. Плату можно приобрести здесь: **https://bit.ly/3lYRfpu**. На указанной веб-странице также предоставляется сопоставление выводов Pico и HAT.

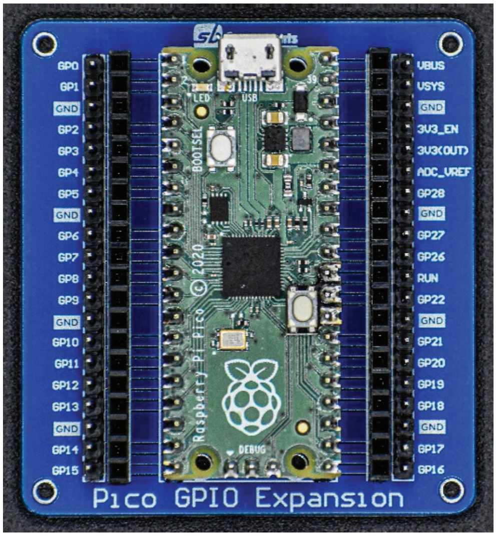


Рисунок 1.31. Pico GPIO Expansion Board

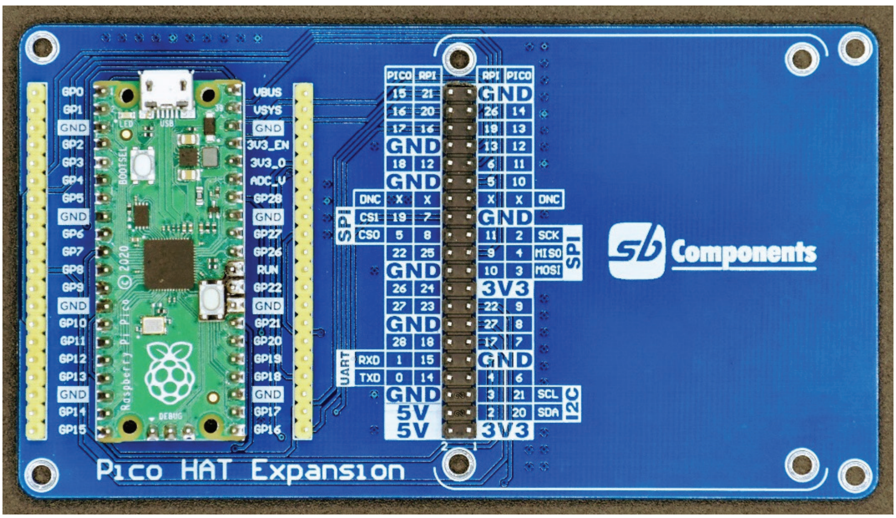


Рисунок 1.32. Pico HAT Expansion

Эта плата позволит добавить Pico в существующий проект Raspberry Pi. Это также может позволить вам использовать HAT-оборудование Raspberry Pi с Pico.

Grove Shield for Pi Pico (плата расширения Grove для Pi Pico)

Плата расширения Grove Shield for Pi Pico помогает подключить Pico к экосистеме Grove от Seeed Studio. На случай, если вы с ней незнакомы: экосистема Grove состоит из модульных плат для создания макетов и прототипов. Как вы можете видеть на рис. 1.33, плата состоит из ряда разъемов, которые позволяют подключать ее к датчикам и исполнительным механизмам. Плату можно приобрести по адресу **https://bit.ly/2NZG3MO**.

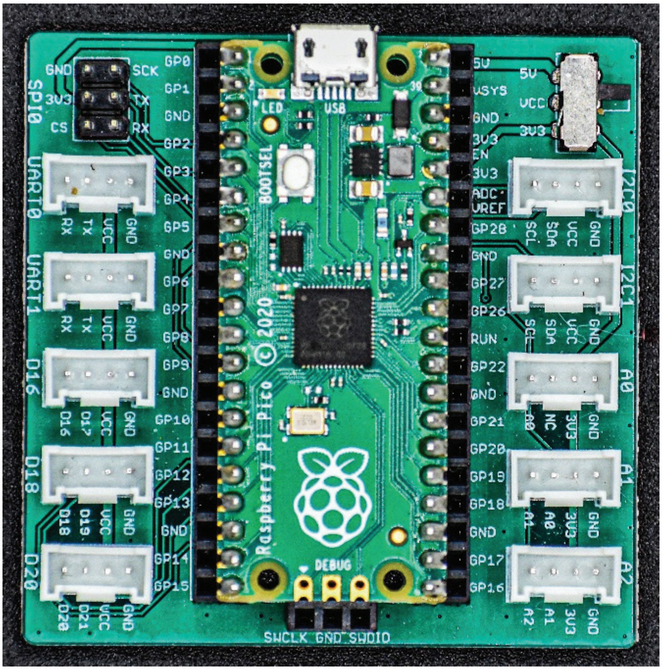


Рисунок 1.33. Grove Shield for Pi Pico

В примерах далее мы обсудим макетирование с помощью подобных плат расширения.

Pimoroni Pico Decker (четырехкратный расширитель)

Эта плата позволяет подключать до четырех плат расширения к Pico (см. рис. 1.34), но важно убедиться, что между дополнительными платами нет конфликтов контактов. Плату можно приобрести на сайте **https://bit.ly/3lXpWMt**.

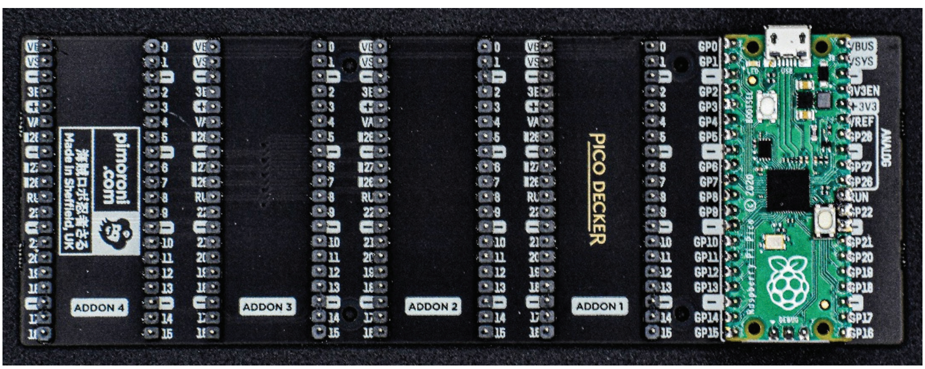


Рисунок 1.34. Pimoroni Pico Decker (четырехкратный расширитель)

В этом разделе мы рассмотрели различные комплекты, доступные для Raspberry Pi Pico. Список не является исчерпывающим – например, платы Feather/Thing Plus поставляются с совместимым оборудованием, известным как FeatherWings. Аналогичным образом платы MicroMod поставляются со своей собственной экосистемой дополнительного оборудования.

Итоги

В этой главе мы познакомили вас с Raspberry Pi Pico. Мы обсудили периферийные устройства микроконтроллера RP2040 и варианты платы Pico различных производителей оборудования. Мы также обсудили дополнительные комплекты для макетирования, доступные для Pico. Мы припаяли разъемы для Pico и добавили кнопку сброса. Мы также обсудили пример «Hello World!» и пример мигания светодиода с использованием CircuitPython и MicroPython.

Теперь, когда вы закончили настройку Pico и ознакомились с опциями, доступными для программирования Pico, мы обсудим особенности микроконтроллера RP2040 на практических примерах.

В следующей главе мы рассмотрим коммуникационные интерфейсы, доступные на Pico.

Глава 2

Последовательные интерфейсы и их приложения

В этой главе мы обсудим различные коммуникационные интерфейсы, доступные на Pico. Мы обсуждаем их все в одном месте, потому что они пригодятся в проектах, которые мы рассматриваем в следующих главах. Темы, обсуждаемые в этой главе, будут полезны для взаимодействия с датчиками, управления исполнительными механизмами, отображения сообщений и взаимодействия с модулем Wi-Fi, подключенным к локальной сети для размещения данных в облаке и извлечения их оттуда. Эта глава также поможет вам, когда дело дойдет до взаимодействия с модулями сотовой связи, которые также действуют как периферийные устройства.

Если вы знакомы с темами, обсуждаемыми в данной главе, переходите сразу к следующей главе. Если нет, мы рекомендуем поиграть с примерами кода, рассмотренными в этой главе, чтобы получить базовое представление о коммуникационных интерфейсах, доступных на Pico. В последнем разделе этой главы мы обсудим настройку специального беспроводного модуля ESP32 для Pico. Мы будем использовать этот беспроводной модуль во всех проектах, обсуждаемых в этой книге.

В данной главе мы рассмотрим следующие основные темы:

- установка необходимых библиотек;
- использование интерфейса UART для связи между двумя платами Pico;
- подключение датчиков с использованием интерфейса I2C;
- отображение данных о температуре с помощью SPI;
- настройка беспроводного модуля.

Технические требования

В этой главе используется следующее оборудование:

- 2 шт. Raspberry Pi Pico (ссылка: **https://bit.ly/3AJtoAf**);
- датчик температуры и влажности DHT20 (ссылка: **https://bit.ly/3q5KVAx**) или датчик температуры HTU21D-F (ссылка: **https://bit.ly/3gCM51T**)[10];

[10] Могут быть заменены на любой датчик температуры-влажности с I2C-интерфейсом, например на основе сенсоров SHT20, SHT21, SHT31, SHT85, указанного авторами AHT20 (см. далее) и его более качественной версии AHT21 и др. Разумеется, понадобятся соответствующие библиотеки – несмотря на близкую функциональность и одинаковый интерфейс, датчики в большинстве случаев невзаимозаменяемые. – *Прим. перев.*

- 128×32 SPI OLED-дисплей (ссылка: **https://bit.ly/3yAwHJY**)[11];
- Pico Omnibus – двойной расширитель (ссылка: **https://bit.ly/3sr2GJR**);
- беспроводной модуль Pico (ссылка: **https://bit.ly/3yPuoT9**).

> **Выбор компонентов**
>
> Возможно, вы заметили, что мы рекомендовали использовать два датчика температуры. Мы хотели предоставить альтернативные варианты, чтобы вы могли выбрать компонент в соответствии с вашим бюджетом.

Примеры кода, рассмотренные в этой главе, доступны для загрузки здесь:

https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIYWorkshop/tree/main/chapter_02.

Видеоролики с кодом в действии для этой главы можно посмотреть по адресу **https://bit.ly/3w9qPH2**.

В следующем разделе мы начнем с установки необходимых библиотек.

Установка необходимых библиотек

В этом разделе мы установим необходимые библиотеки для датчика температуры, дисплея и беспроводного модуля. Все библиотеки являются частью пакета Adafruit CircuitPython. Последний пакет можно загрузить в виде ZIP-файла с **https://circuitpython.org/libraries**. Мы используем версию пакета, предназначенную для CircuitPython 6.x.x.

После загрузки ZIP-файла извлеките его содержимое, чтобы мы могли скопировать библиотеки, необходимые нам для проекта.

> **Установка CircuitPython**
>
> Мы предполагаем, что вы установили CircuitPython на свой Pico. Если вы незнакомы с процессом установки, мы рекомендуем следовать инструкциям из главы 1 «*Начало работы с Raspberry Pi Pico*».

Датчик температуры HTU21D-F

Если вы используете датчик температуры HTU21D-F (показан на рис. 2.1), для него требуются две библиотеки, а именно *adafruit_bus_device* и *adafruit_htu21d*. Из папки библиотеки скопируйте папку *adafruit_bus_device* и двоичный файл *adafruit_htu21d.mpy* в папку *lib* вашего Pico.

В следующем разделе мы рассмотрим библиотеки, необходимые для датчика температуры DHT20.

---

[11] Для дисплеев 0,91 дюйма на основе контроллера SSD1306 имеется множество дешевых аналогов, доступных как в отечественных магазинах (см. примечание на стр. 23), так и на AliExpress. – *Прим. перев.*

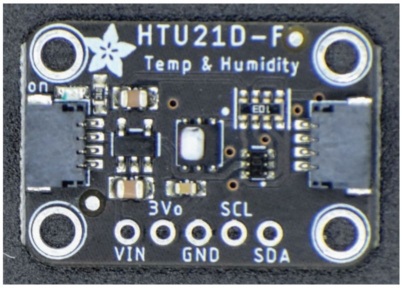


Рисунок 2.1. Датчик температуры и влажности HTU21D-F

Датчик температуры и влажности DHT20

Если вы используете датчик температуры DHT20 (показан на рис. 2.2), для него требуются две библиотеки, а именно *adafruit_bus_device* и *adafruit_ahtx0*. Из папки библиотеки скопируйте папку *adafruit_bus_device* и двоичный файл *adafruit_ahtx0.mpy* в папку *lib* вашего Pico.

Микросхема внутри датчика DHT20 представляет собой датчик температуры AHT20. Поэтому мы устанавливаем библиотеку *adafruit_ahtx0*.

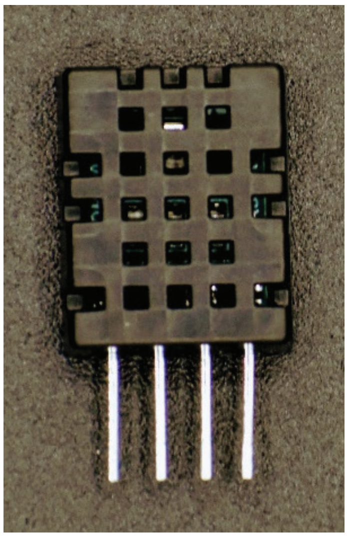


Рисунок 2.2. Датчик температуры и влажности DHT20

Далее мы установим библиотеки для OLED-дисплея.

OLED-дисплей (контроллер SSD1306)

Для OLED-дисплея требуются три библиотеки, а именно *adafruit_bus_device*, *adafruit_displayio_ssd1306* и *adafruit_display_text*. Поскольку мы установили первую из этих библиотек в предыдущем разделе, скопируем папку с именем *adafruit_display_text* и двоичный файл *adafruit_displayio_ssd1306.mpy* из пакета библиотек в папку *lib* вашего Pico.

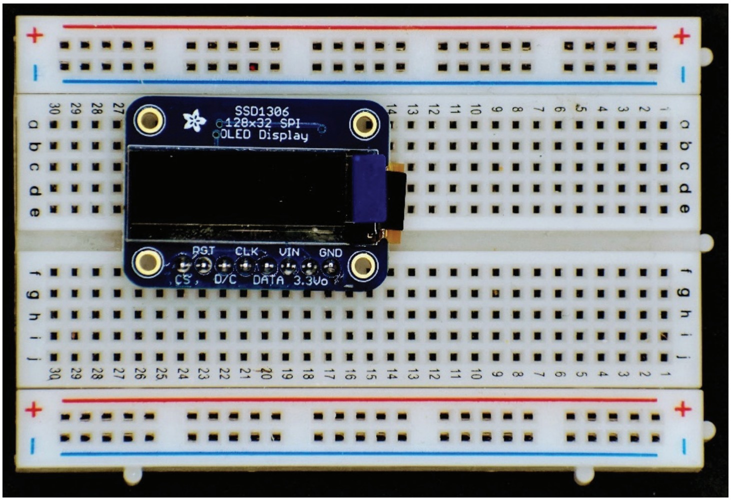


Рисунок 2.3. 128×32 SPI OLED-дисплей

Далее мы установим библиотеки для беспроводного модуля.

Беспроводной модуль

Для беспроводного модуля нам нужна библиотека *adafruit_esp32spi*, поскольку мы уже установили *adafruit_bus_device*. Скопируйте папку с тем же именем в папку *lib* вашего Pico. Нам также понадобится двоичный файл *adafruit_requests.mpy* из пакета библиотеки.

Рисунок 2.4. Беспроводной модуль ESP32 для Pico

Теперь, когда мы установили необходимые библиотеки, мы начнем с тестирования интерфейса UART.

Использование интерфейса UART для связи между двумя платами Pico

В этом разделе мы обсудим интерфейс универсального асинхронного приемника/передатчика (**Universal Asynchronous Receiver/Transmitter,**

UART) и его приложения. Мы будем использовать две платы Pico для передачи сообщения от одного Pico к другому. Второй Pico повторяет входящее сообщение. Прежде чем мы перейдем к примеру, давайте взглянем на интерфейс UART.

Интерфейс UART

Интерфейс UART представляет собой протокол связи, использующий две линии – передающую и приемную (показаны на следующем рисунке). Как следует из названия, протокол является асинхронным, то есть для связи нет опорного тактового сигнала.

Передача данных происходит с заданной заранее скоростью передачи данных в бодах[12]. Типичные скорости передачи данных в бодах включают 9600, 19 200, 38 400 и т. д.

Интерфейс UART используется для связи с датчиками, такими как датчики GPS, регистраторы данных и т. д. На рис. 2.5 показано соединение между двумя устройствами. Приемник (RX) одного устройства подключен к передатчику (TX) другого устройства, и их контакты заземления соединены друг с другом. Чтобы инициировать связь между ними, оба устройства должны работать с предварительно настроенной скоростью передачи данных в бодах, размером кадра данных и т. д. Обычно для датчиков с выходом UART эти спецификации указаны в техническом описании. В качестве упражнения мы предлагаем ознакомиться со спецификациями следующего GPS-датчика и узнать его скорость передачи данных в бодах: **https://www.adafruit.com/product/746**.

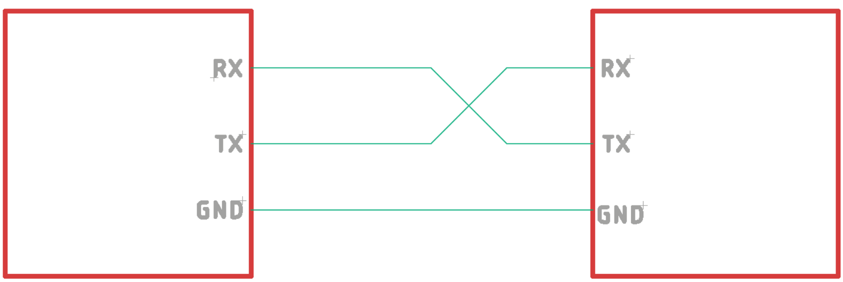


Рисунок 2.5. Настройка связи UART

[12] Строго говоря, в *битах в секунду*. Для UART сложилась традиция единицу скорости передачи именовать *бодом*, хотя в действительности речь здесь идет именно о битах в секунду. Скорость передачи, указанная в бодах (по фамилии французского телеграфиста Эмиля Бодо, изобретателя кодировки символов для телетайпов), называется *символьной* скоростью, в то время как указанная в битах в секунду – *битовой* скоростью. Количество посылок (символов) в секунду, называемое бодом, отличается от количества битов, которое может быть передано таким способом: например, модем, работающий со скоростью 2400 бод, может иметь битовую скорость 19 200 бит/с – в этом случае одна посылка (символ) несет 8 бит информации, что достигается специальными системами модуляции. Однако принято считать, что для UART символьная и битовая скорости совпадают. При подсчетах следует учитывать также, что полный UART-кадр, несущий один байт информации, имеет размер не 8, а 10 бит (с учетом стартового и стопового битов), т. е. скорость 9600 бит/с равна ровно 960 байт/с. – *Прим. перев.*

Мы рекомендуем следующие статьи для более глубокого ознакомления с интерфейсом UART: **https://radioprog.ru/post/168** (физические основы устройства), **https://www.rotr.info/electronics/mcu/arm_usart.htm** (UART, COM-порт и RS-232), а также **https://musbench.com/all/uart/** (особенности реализаций).

Настройка Pico

В этом разделе мы рассмотрим подключение двух Raspberry Pi Pico. Мы собираемся использовать порт UART0, выделенный прямоугольником на рисунке:

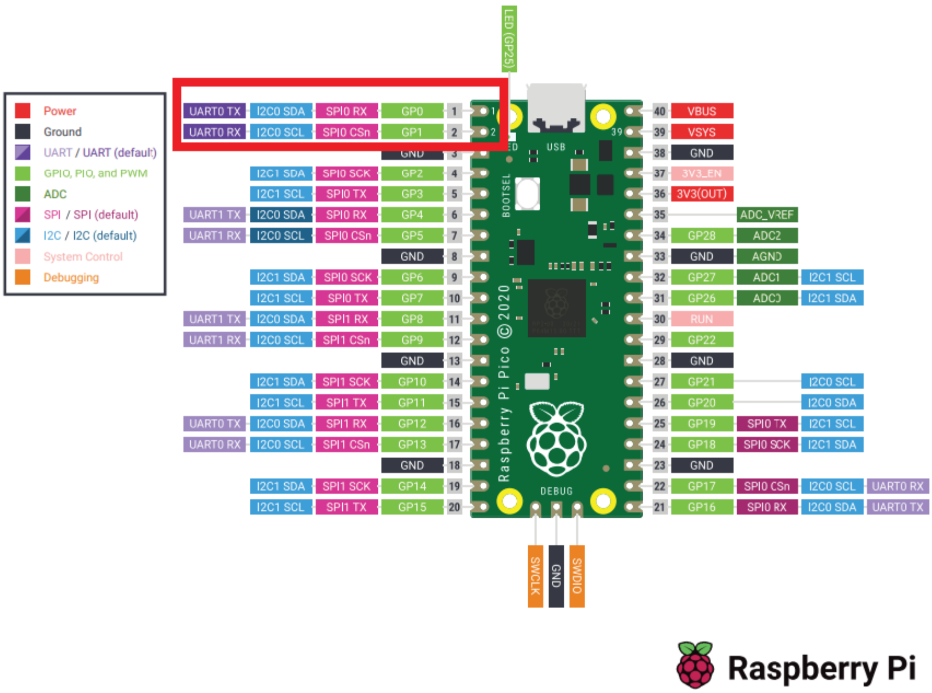


Рисунок 2.6. Разводка выводов Pico
(источник изображения: Raspberry Pi Foundation)

Как видно по рис. 2.6, Pico имеет два порта с именами UART0 и UART1[13]. В этом примере мы собираемся использовать UART0. Чтобы передавать сообщения между платами, нам нужно подключить передающий вывод TX первого Pico (вывод 1, GPIO0) к приемному выводу RX (вывод 2, GPIO1) второго Pico, как показано в схемах на рис. 2.5 и 2.7. Аналогично приемный вывод RX (вывод 2, GPIO 1) первого Pico соединен с передающим выводом TX второго Pico (вывод 1, GPIO0). Заземляющие выводы GND обеих плат соединены между собой.

[13] Но с возможностью переключения выводов RX и TX на разные пары контактов. – *Прим. перев.*

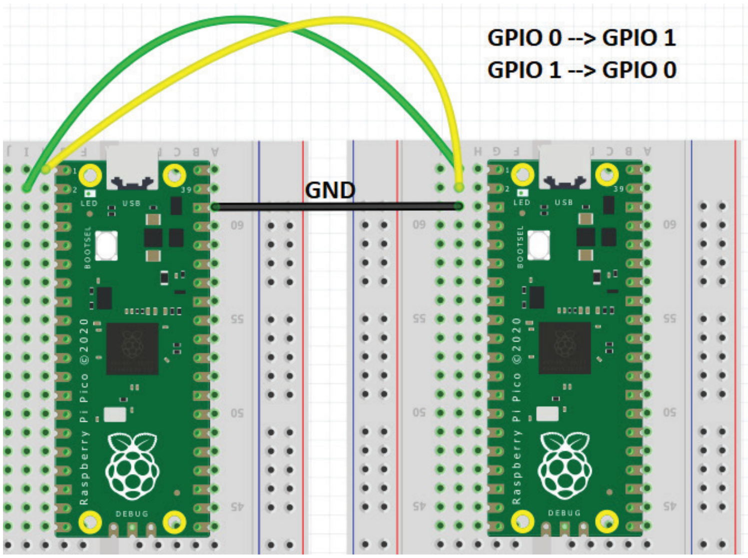


Рисунок 2.7. Схема соединений Pico UART (источник: Pico part by vanepp)

Теперь, когда мы соединили две платы Pico, давайте рассмотрим их программирование.

Программирование UART на платах Pico

В этом примере мы работаем с двумя Pico, а именно с передатчиком и приемником. Давайте рассмотрим код, используемый в обеих платах Pico. Хотя они схожи по своей структуре, между ними существуют определенные различия из-за их функции.

Программирование передатчика

Первая плата Pico принимает входные данные от пользователя через **терминал последовательного порта**. Затем захваченные входные данные передаются на второй Pico. Пример кода (доступен для скачивания здесь: **https://bit.ly/2ZI79gO**):

```
import board
import busio
uart = busio.UART(board.GP0, board.GP1, baudrate=9600)
while True:
  # capture message
  message = input("Enter a message:").encode("utf-8")
  uart.write(message)
  # receive echo
     bytes_waiting = uart.in_waiting
     if bytes_waiting:
     incoming_msg = uart.read(bytes_waiting)
     print(incoming_msg)
```

Теперь давайте рассмотрим отдельные блоки кода. Первым шагом является импорт необходимых модулей в нашу программу. В CircuitPython модуль платы `board` содержит все определения выводов, необходимые для Pico. Модуль `busio` содержит класс UART, необходимый для этого примера:

```
import board
import busio
```

В предыдущем разделе мы обсуждали контакты GPIO0 и GPIO1. Мы инициализируем класс UART для этих выводов со скоростью передачи 9600:

```
uart = busio.UART(board.GP0, board.GP1, baudrate=9600)
```

Теперь запустим бесконечный цикл, в котором мы фиксируем входящие сообщения с помощью функции `input()` следующим образом:

```
while True:
  # capture message
  message = input("Enter a message:").encode("utf-8")
```

Вводимая строка преобразуется в байты с помощью `encode`. Это происходит потому, что метод класса UART `write()` принимает только массив байтов (а не более длинных слов, представляющих текстовые символы в многоязычной кодировке UTF-8):

```
uart.write(message)
```

Следующим шагом является ожидание отклика (`echo`) от второй платы:

```
# receive echo
  bytes_waiting = uart.in_waiting
  if bytes_waiting:
  incoming_msg = uart.read(bytes_waiting)
  print(incoming_msg)
```

Свойство `in_waiting` класса UART возвращает количество байтов, ожидающих в буфере. Если в буфере есть ожидающее сообщение, мы считываем его с помощью метода `read()` класса UART. Сообщение выводится на терминал.

Сохраните образец кода в файле *code.py* в первой плате Pico (называемой передатчиком), а в следующем разделе мы рассмотрим пример кода для второй платы (выбранной приемником).

Программирование приемника

Пример кода почти идентичен предыдущему. Программа для второй платы выглядит следующим образом (доступна для скачивания здесь: **https://bit.ly/2Y99WiU**):

```
import time
import board
```

```
import busio

uart = busio.UART(board.GP0, board.GP1, baudrate=9600)

while True:
  # await incoming message
  bytes_waiting = uart.in_waiting
  if bytes_waiting:
      incoming_msg = uart.readline()
      print(incoming_msg)
      # re-transmit
      uart.write(incoming_msg)
```

Давайте рассмотрим, что происходит внутри цикла `while`.

Если в буфере есть байты (возвращаемые свойством `in_waiting`), мы считываем его с помощью метода `readline()`. Метод считывает данные до тех пор, пока не будет достигнут символ новой строки. Мы используем `readline()` вместо `read()`, потому что входящее сообщение содержит символ новой строки и помогает напечатать все сообщение, захваченное функцией `input()` в предыдущем примере кода.

Сохраните программу как файл *code.py* на втором Pico.

В следующем разделе мы протестируем наш код с помощью Mu IDE.

Тестирование кода

Чтобы протестировать наш код, мы будем использовать два экземпляра Mu IDE, для чего нам нужно знать номера последовательных портов каждого Pico. Мы продемонстрируем тест в среде Windows. Процесс должен быть аналогичным в других операционных системах.

Предпримем следующие шаги для тестирования платы.

1. Подключите первую плату Pico и запустите **Диспетчер устройств**, как показано на рис. 2.8 (из **Панели управления** Windows). Определите номер последовательного порта первой платы, как показано на рис. 2.8.

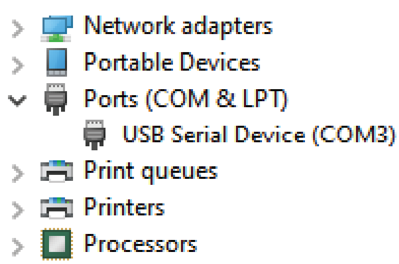


Рисунок 2.8. Обратите внимание на номер COM-порта в **Диспетчере устройств**

2. Подключите вторую плату Pico и также отметьте номер последовательного порта (как показано на рис. 2.9).

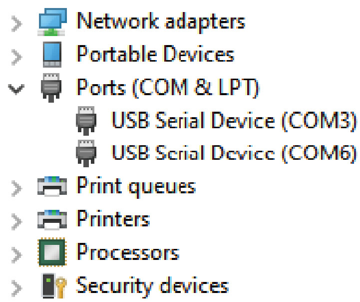


Рисунок 2.9. Обратите внимание на COM-порт второй платы Pico

3. Если вы еще этого не сделали, запустите первый экземпляр Mu IDE и убедитесь, что IDE установлена в режим CircuitPython. Это можно сделать с помощью кнопки **Mode** (*Режим*) в верхнем левом углу панели инструментов IDE (см. рис. 1.26).
4. Теперь нам нужно выбрать последовательный порт первой платы Pico. В правом нижнем углу Mu IDE находится выпадающее меню плат, подключенных к компьютеру. При наведении курсора мыши на одно из устройств серийный номер должен быть выделен (как показано на рис. 2.10). Выберите последовательный порт первой платы.

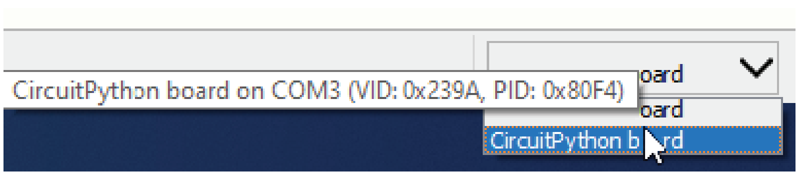


Рисунок 2.10. Выбор последовательного порта в среде Mu IDE

5. Теперь запустите второй экземпляр Mu IDE и повторите процесс для второй платы Pico.
6. Откройте терминал последовательного порта на обоих экземплярах Mu IDE с помощью кнопки **Serial** на панели инструментов:

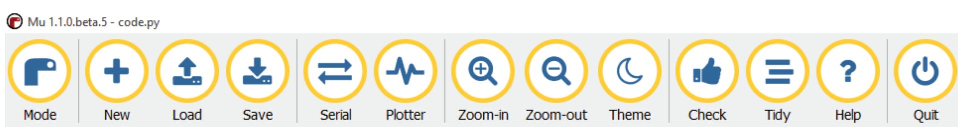


Рисунок 2.11. Кнопка **Serial** на панели инструментов

7. Когда вы откроете сеанс связи, запустив терминал последовательного порта, увидите пустое окно, как показано на следующем рисунке. Это означает, что код уже запущен. Если вы уже видите что-то, выведенное интерпретатором Python, значит, с кодом что-то не так.

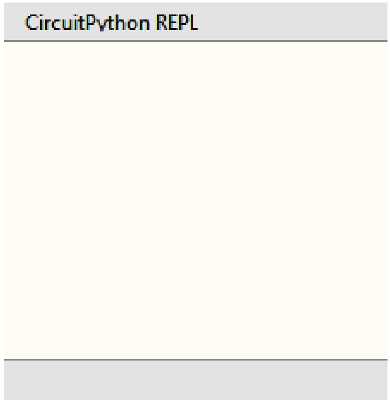
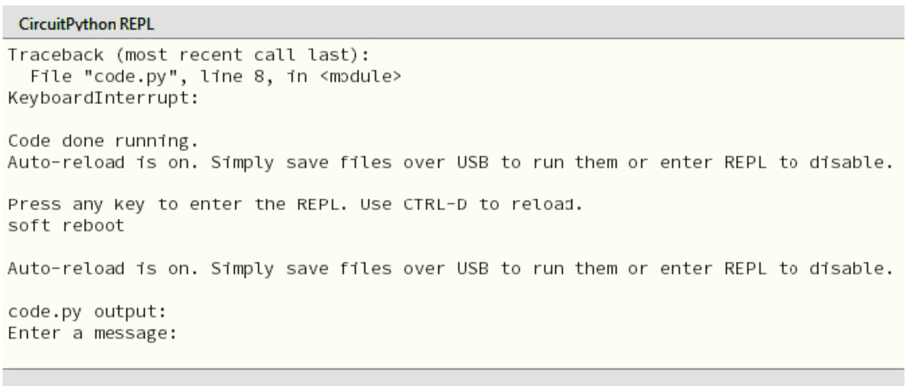


Рисунок 2.12. Код, работающий на вашем Pico, как ожидалось

8. Нажмите <**Ctrl**+**C**>, чтобы прервать выполнение, и <**Ctrl**+**D**>, чтобы перезагрузить код, как показано на следующем рисунке:




```
Traceback (most recent call last):
  File "code.py", line 8, in <module>
KeyboardInterrupt:

Code done running.
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Press any key to enter the REPL. Use CTRL-D to reload.
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

code.py output:
Enter a message:
```

Рисунок 2.13. Результаты перезагрузки кода

9. Повторите действия для второй платы Pico, и вы должны увидеть что-то вроде показанного на следующем рисунке:


```
KeyboardInterrupt:

Code done running.
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Press any key to enter the REPL. Use CTRL-D to reload.
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

code.py output:
```

Рисунок 2.14. Результаты перезагрузки второй платы Pico

10. Введите любое сообщение на первой плате (передатчик), и вы должны увидеть его на второй (приемник), как показано на следующем рисунке:

```
code.py output:
Enter a message:Hello
b'\x00'
Enter a message:World
b'Hello'
Enter a message:!
b'World'
Enter a message:
```

```
code.py output:
b'Hello'
b'Hello'
b'\x00'
b'Hello'
b'World'
b'!'
```

Рисунок 2.15. Отклик на сообщения со второй платы

11. Когда вы вводите второе сообщение на первом Pico (передатчике), то должны увидеть эхо первого сообщения, переданного обратно другой платой Pico (приемником). В показанном на рис. 2.15 отклике первым сообщением было `Hello`. Оно было передано на вторую плату и направлено обратно. Когда вводится второе сообщение `World`, на экран также выводится отраженное сообщение. В коде первого Pico функция `input()` ожидает, пока не будет что-то введено пользователем, а затем выводит на экран пришедшие сообщения из буфера UART.
12. По окончании тестирования можно закрыть сеанс нажатием кнопки **Serial** на панели инструментов.

В следующем разделе мы обсудим некоторые приложения интерфейса UART.

Приложения интерфейса UART

Мы хотели бы показать вам кое-какое оборудование, использующее интерфейс UART микроконтроллера RP2040. На рис. 2.16 ниже показана плата расширения Raspberry Pi Build HAT – RP2040. Она позволяет подключать датчики и управлять двигателями. Плата HAT спроектирована с использованием микроконтроллера RP2040 и, согласно документации, предоставленной Raspberry Pi foundation, управляется через интерфейс UART. Сталкивались ли вы с оборудованием, которым также можно управлять с помощью UART[14]?

[14] До появления современных микроконтроллеров UART (точнее, основанные на нем стандарты RS-232 и RS-485, а также COM-порт у «больших» компьютеров) был основным интерфейсом взаимодействия компьютеров с периферийными устройствами (наряду с параллельным портом LPT). По мере внедрения в периферию собственных управляющих микроконтроллеров UART был постепенно вытеснен более удобными и надежными интерфейсами: скоростным SPI (в масштабах платы), I2C (в масштабах устройства), а также USB (между устройствами на расстояниях в единицы-десятки метров). В последнее время активно развиваются также беспроводные интерфейсы для связи устройств и модулей (о чем будет идти речь далее в этой книге). Заметим, однако, что при связи микроконтроллерных устройств с «большими» компьютерами в операционной системе последних большинство «новых» интерфейсов все равно отображаются как виртуальный COM-порт. Основной недостаток традиционного UART (соединение только «точка-точка») при этом преодолевается возможностью создания множества виртуальных COM-портов. – *Прим. перев.*

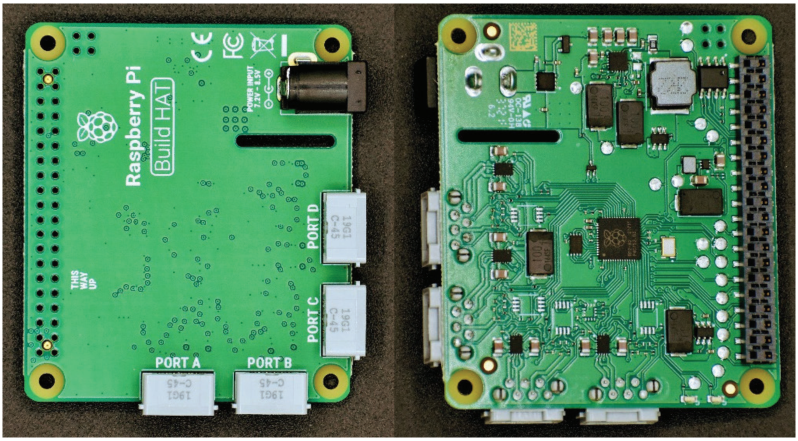


Рисунок 2.16. Raspberry Build HAT – RP2040

В следующем разделе мы обсудим интерфейс I2C.

Подключение датчиков через интерфейс I2C

В этом разделе мы будем использовать интерфейс I2C[15] (произносится как «и-два-Си») для считывания температуры и влажности с датчика HTU21D. Мы будем считывать температуру и отображать ее на OLED-дисплее.

Введение в интерфейс I2C

В предыдущем разделе мы обсуждали интерфейс UART, который является асинхронным; то есть опорный тактовый сигнал в нем отсутствует. Теперь мы обсудим интерфейс I2C, который является синхронным и обычно состоит из тактового вывода (вывода опорного тактового сигнала, обычно маркируется как SCL или CLK) и вывода данных (SDA, Data, DA). На рис. 2.17 показано схематическое представление устройств на шине I2C, где у нас есть хост-устройство, которое обычно представляет собой микроконтроллер (например, RP2040), и ведомые (периферийные) устройства (датчики), соединенные линиями тактовых импульсов и данных. Каждый датчик на шине имеет уникальный адрес, который позволяет хосту взаимодействовать с устройствами, присутствующими на шине.

В то время как интерфейс UART предназначен для связи между двумя устройствами, интерфейс I2C предназначен для связи между хостом и несколькими периферийными устройствами. Для более глубокого изучения мы рекомендуем посмотреть подробное описание интерфейса I2C, например, здесь: **http://easyelectronics.ru/interface-bus-iic-i2c.html**.

---

15 Первоначально обозначался с цифрой в верхнем индексе: I^2C, но постепенно написание было упрощено. Иногда обозначается с двойным I вместо цифры: IIC. В описаниях микроконтроллеров Atmel AVR (а также основанной на них платформе Arduino) тот же интерфейс принято именовать TWI, что означает просто *Two-Wire Interface* – двухпроводной интерфейс. – *Прим. перев.*

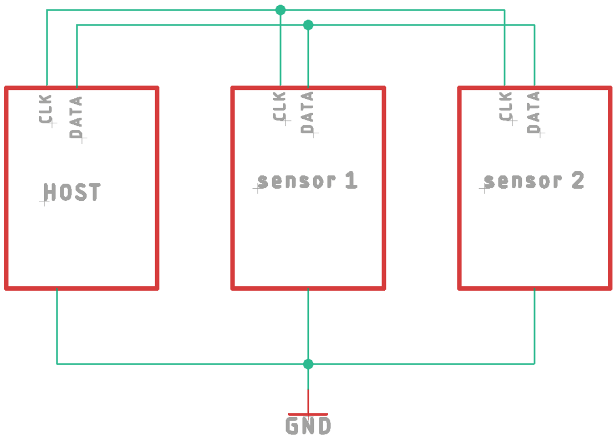


Рисунок 2.17. Схематическое представление шины I2C

Интерфейс I2C обычно состоит из одного хост-устройства и одного ведомого, но может включать в себя несколько основных и периферийных устройств. Интерфейс I2C обычно используется для связи с такими устройствами, как датчики, дисплеи и т. п. На рис. 2.18 Pico сопряжен с сотовым модулем от **https://blues.io**, называемым **Notecard**. Pico служит в качестве хост-устройства, а сотовый модуль – периферийного устройства. Это позволяет Pico собирать данные с других датчиков на той же шине I2C и размещать их в облаке. На этой основе мы создали телефон с клавиатурой, который отправляет вам текстовое сообщение. Подробнее о проекте вы можете прочитать в главе 11 «*Создаем готовый продукт*», а также здесь: **https://bit.ly/3pY7J5e**.

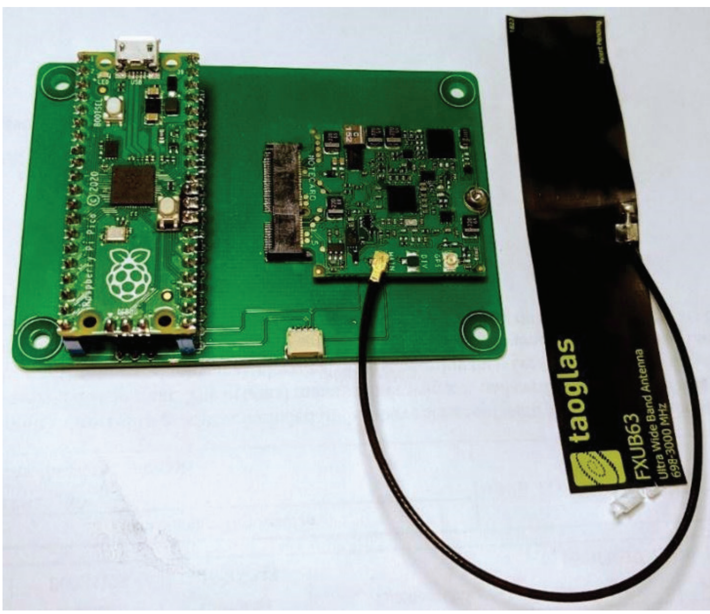


Рисунок 2.18. Pico, подключенный к сотовому модулю

В следующем разделе мы рассмотрим необходимость использования подтягивающих резисторов на шине I2C.

Подтягивающие резисторы

Линии синхронизации и передачи данных шины I2C являются выходами с открытым стоком (коллектором). Они работают, устанавливая шину в низкое состояние, но не могут сами привести ее в высокое. Следовательно, для нормальной работы необходимы подтягивающие резисторы, как показано на следующем рисунке:

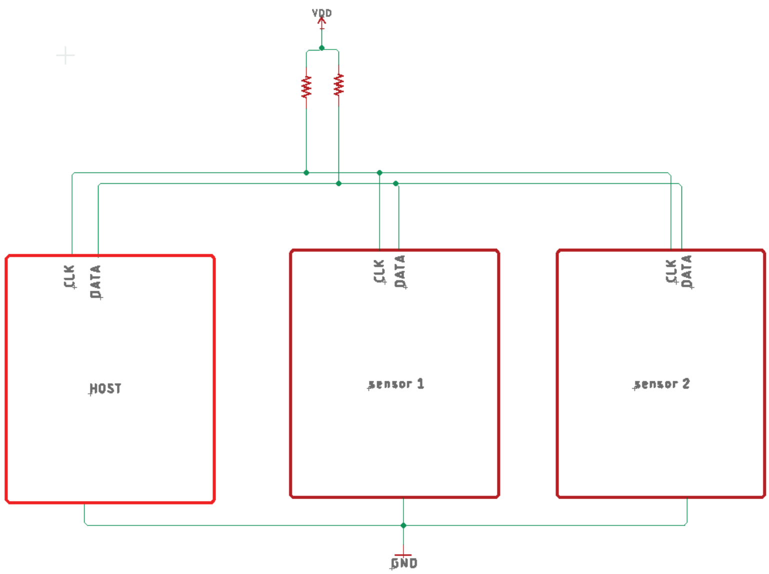


Рисунок 2.19. Подтягивающие резисторы

Значения подтягивающего резистора зависят от различных факторов. Эта информация из Texas Instruments будет полезна при расчете значений подтягивающих резисторов: **https://bit.ly/30dLM7G**. Обычно большинство модулей с интерфейсом I2C поставляются со встроенными подтягивающими резисторами. Иногда у вас может быть возможность отключить их, если в другом месте цепи есть такие резисторы. Для нормальной работы на шине I2C обязательно необходимо иметь подтягивающие резисторы[16].

Теперь, когда мы рассмотрели интерфейс I2C, мы собираемся подключить датчик температуры к Pico.

[16] Нормальное значение подтягивающих резисторов для тактового сигнала ~100 кГц и менее находится в пределах 2,4–10 кОм. В готовых модулях обычно установлены резисторы около 4,7–5,1 кОм. При количестве таких параллельно установленных модулей три и более общее сопротивление подтягивающих резисторов может снизиться до 1,5 кОм и менее, что может перегружать выходные транзисторы. Опыт показывает, что при трех параллельных модулях это не приводит к каким-либо проблемам, но при большем количестве рекомендуется лишние подтягивающие резисторы удалить с плат части модулей (оставив их на одной-двух платах). – *Прим. перев.*

Тестирование датчика температуры HTU21D-F

В этом разделе мы рассмотрим подключение датчика температуры HTU21D-F с Raspberry Pi Pico. На рис. 2.20 показаны соединения датчика температуры с Pico. В следующей таблице слева от стрелки показаны контакты на плате температурного датчика, справа – наименования контактов на Pico:

- 3,3V → VBUS;
- SCL → GP9;
- SDA → GP8;
- GND → GND.

Вот примерная схема соединений между Pico и датчиком температуры HTU21D-F:

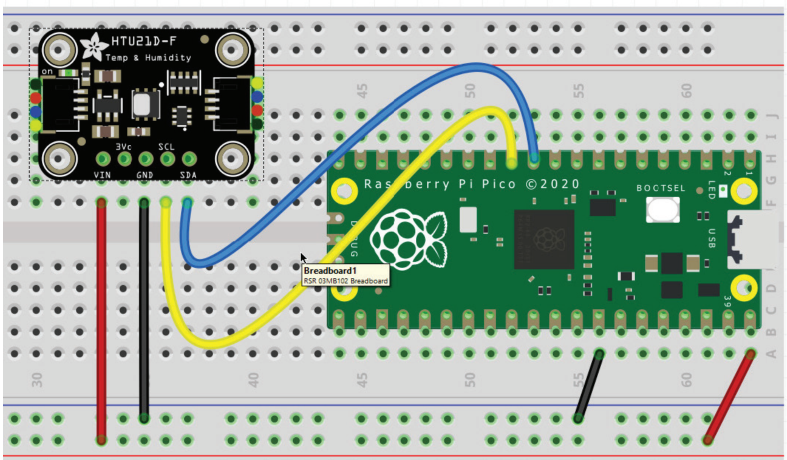


Рисунок 2.20. Соединения Pico с датчиком температуры/влажности HTU21D

Теперь давайте рассмотрим код, необходимый для считывания температуры и влажности с датчика HTU21D.

Код датчика температуры HTU21D-F

Мы предполагаем, что вы установили необходимые библиотеки для датчика температуры (см. раздел «*Установка необходимых библиотек*»). Давайте поэтапно рассмотрим код, необходимый для подключения датчика температуры (доступен здесь: **https://bit.ly/3jZD3Nn**). Пример кода взят с сайта Adafruit.

1. Первым шагом является импорт необходимых модулей:

```
import time
import board
import busio
from adafruit_htu21d import HTU21D
```

2. Следующим шагом является инициализация класса I2C из модуля `busio` и класса `HTU21D`:

```
i2c = busio.I2C(board.GP9, board.GP8)
sensor = HTU21D(i2c)
```

3. Теперь мы можем начать считывать температуру в цикле (каждые две секунды):

```
while True:
    print("\nTemperature: %0.1f C" % sensor.temperature)
    print("Humidity: %0.1f %%" % sensor.relative_humidity)
    time.sleep(2)
```

4. Собрав все это вместе, имеем следующий код:

```
import time
import busio
import board
from adafruit_htu21d import HTU21D
i2c = busio.I2C(board.GP9, board.GP8)
sensor = HTU21D(i2c)
while True:
    print("\n Temperature: %0.1f C" % sensor.temperature)
    print("Humidity: %0.1f %%" % sensor.relative_humidity)
    time.sleep(2)
```

5. Сохранив этот образец кода в Pico как *code.py*, мы получаем возможность считывать температуру и влажность с датчика (показано на рисунке с использованием Mu IDE):

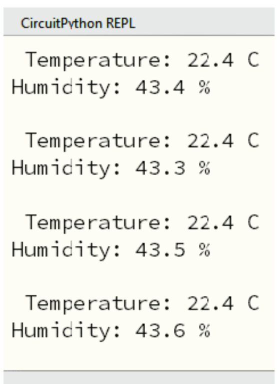


Рисунок 2.21. Данные о температуре и влажности от HTU21D

В следующем разделе мы рассмотрим тестирование датчика AHT20.

Тестирование датчика температуры DHT20

В этом разделе мы обсудим подключение датчика температуры DHT20 к Raspberry Pi Pico. В таблице (рис. 2.22) показаны выводы датчика DHT20 (ссылка на техническое описание: **https://bit.ly/31GrBQo**).

Вывод	Название	Описание
1	VDD	Напряжение питания (2,2–5,5 В)
2	SDA	Двунаправленный вывод/ввод последовательных данных
3	GND	Общий провод («земля»)
4	SCL	Двунаправленный вывод/ввод тактовых импульсов

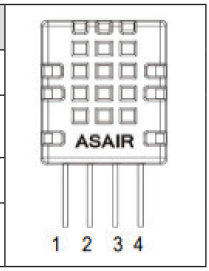


Рисунок 2.22. Разводка выводов датчика DHT20

Датчик температуры соединен с Pico, как показано на следующем рисунке. Соединения показаны в таблице, где стрелка указывает на вывод Pico, а слева указан вывод температурного датчика (его выводы см. на предыдущем рисунке):

- 3,3V → VBUS;
- SCL → GP9;
- SDA → GP8;
- GND → GND.

Схема соединений между Pico и датчиком температуры AHT20 показана на следующем рисунке. Как упоминалось ранее, датчик DHT20 содержит внутри датчик AHT20, следовательно, в любом случае мы создали схему для компонента AHT20[17].

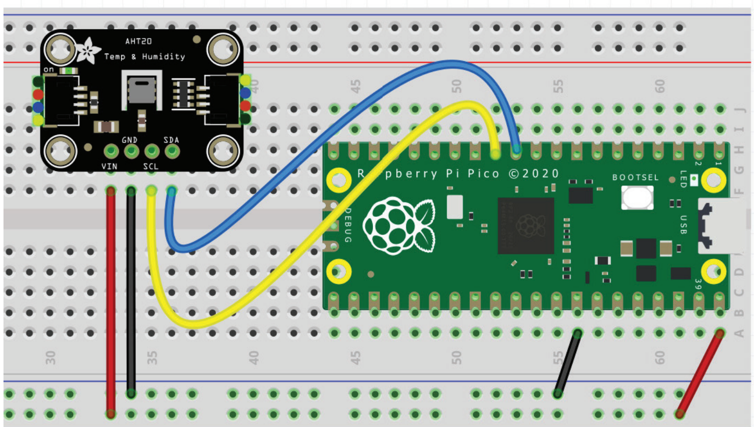


Рисунок 2.23. Подключение Pico к датчику температуры/влажности AHT20

17 Обратите внимание, что в таблице на рис. 2.22 приведена разводка выводов для модуля DHT20 производства компании Aosong Electronic (торговая марка ASAIR), и выше дана ссылка на соответствующее техническое описание. В то время как на рис. 2.23 показан модуль AHT20 производства Adafruit (как видно по рисунку, у него иная разводка выводов). Некоторая путаница в авторском изложении связана с неудачным названием модуля Adafruit – в основе и DHT20, и этого модуля лежит сенсор AHT20 от Aosong (**http://aosong.com/userfiles/files/media/Data%20Sheet%20AHT20.pdf**). Ознакомиться с модулем Adafruit AHT20 можно по ссылке: **https://learn.adafruit.com/adafruit-aht20**. – *Прим. перев.*

Теперь давайте рассмотрим код, необходимый для считывания температуры и влажности с датчика AHT20.

Код для датчика температуры/влажности AHT20

Мы предполагаем, что вы установили необходимые библиотеки для датчика температуры (см. раздел «*Установка необходимых библиотек*»). Давайте вкратце рассмотрим код Adafruit, необходимый для работы с датчиком температуры.

1. Первым шагом является импорт необходимых модулей:

```
import time
import board
import busio
from adafruit_ahtx0 import AHTx0
```

2. Следующим шагом является инициализация класса I2C из модуля `busio` и класса `AHTx0`:

```
i2c = busio.I2C(board.GP9, board.GP8)
sensor = AHTx0(i2c)
```

3. Теперь мы можем начать считывать температуру в цикле (каждые две секунды):

```
while True:
    print("\nTemperature: %0.1f C" % sensor.temperature)
    print("Humidity: %0.1f %%" % sensor.relative_humidity)
    time.sleep(2)
```

4. Собрав все это вместе, мы имеем следующий код:

```
import time
import busio
import board
from adafruit_ahtx0 import AHTx0
i2c = busio.I2C(board.GP9, board.GP8)
sensor = AHTx0(i2c)
while True:
    print("\n Temperature: %0.1f C" % sensor.temperature)
    print("Humidity: %0.1f %%" % sensor.relative_humidity)
    time.sleep(2)
```

5. Как только мы сохраним наш образец кода в Pico как *code.py*, мы получаем возможность считывать температуру и влажность с датчика (на рисунке показано с использованием Mu IDE):

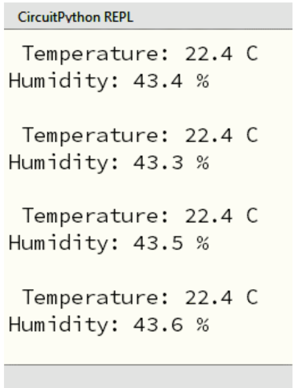


Рисунок 2.24. Данные о температуре и влажности от AHT20

Теперь, когда мы протестировали датчики температуры, кратко рассмотрим экосистему Qwiic/STEMMA.

Экосистема Qwiic/STEMMA

Экосистема Qwiic состоит из серии датчиков, дисплеев и т. д., представленных компанией SparkFun с ориентированными разъемами[18], которые обеспечивают простое подключение устройств, использующих интерфейс I2C. Вы можете узнать больше об интерфейсе Qwiic здесь: **https://www.sparkfun.com/qwiic**.

В то время как платы Qwiic от SparkFun работают только при напряжении 3,3 В, Adafruit Industries разработала платы, допускающие напряжение 5 В. Экосистема плат Adafruit, использующих интерфейс I2C, называется STEMMA. Вы можете ознакомиться с экосистемой STEMMA здесь: **https://bit.ly/3buXSM3**. Поскольку платы Qwiic и STEMMA имеют одинаковые разъемы, они перекрестно совместимы, но платы Qwiic работают только при уровне сигнала 3,3 В. Вы должны быть осторожны при смешивании разных плат.

Плата Feather RP2040

На следующем рисунке датчик температуры STEMMA HTU21D подключен к плате Adafruit Feather RP2040 с помощью кабеля Qwiic. Напомним, что плата Feather RP2040 – это вариант Raspberry Pi Pico в форм-факторе Adafruit Feather. Вы можете узнать больше об этом здесь: **https://bit.ly/3mz5G5t**.

Платы HTU21D и Feather RP2040 поставляются с разъемом Qwiic и помогают упростить макетирование, как показано на рис. 2.25. Вы заметите, что для подключения датчика к плате Feather используется простая сборка проводов, содержащая ориентированные разъемы на обоих концах.

---

[18] То есть их нельзя случайно перевернуть при подключении. Большим достоинством системы Qwiic/STEMMA являются горизонтальное расположение разъемов, не занимающих лишнего места по высоте платы (в отличие от других систем, упомянутых в сноске 7 на стр. 25). – *Прим. перев.*

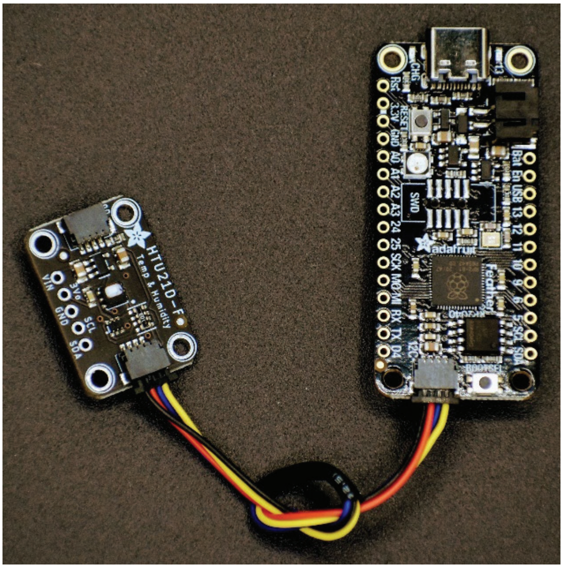


Рисунок 2.25. Feather RP2040, подключенная к HTU21D с помощью кабеля Qwiic

Разъемы Qwiic также позволяют соединять платы последовательно друг с другом. На следующем рисунке показаны две платы датчиков температуры, соединенные последовательно. Это облегчает сборку макетов без запутанных проводов на макетной плате и помогает избежать ошибок при соединениях.

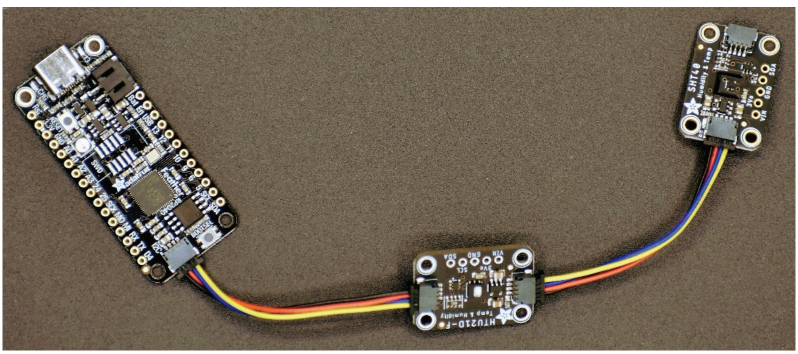


Рисунок 2.26. Feather RP2040 с двумя датчиками температуры

Знакомство с интерфейсом UART и I2C может помочь при подключении дисплеев, управлении двигателями и отладке кода. В следующем разделе мы обсудим устранение неполадок на шине I2C.

Устранение неполадок

Вы можете столкнуться со сценариями, в которых вам потребуется устранить неполадки в подключениях к шине I2C на макетной плате. Это может быть что-то простое, как перепутывание линий передачи данных и тактирования. В CircuitPython есть удобный инструмент сканирования, помогающий в поиске устройств, присутствующих на шине I2C. С его помощью можно определить, правильно ли подключено устройство и можем ли мы его идентифицировать. Посмотрим на код, доступный в документации CircuitPython (ссылка: **https://bit.ly/3H7oUrq**). Образец кода доступен для скачивания вместе с этой главой (файл *i2c_scanning.py*).

1. Первым шагом является импорт необходимых модулей и инициализация шины I2C:

```
import time
import board
import busio
i2c = busio.I2C(board.GP9, board.GP8)
```

2. Следующим шагом является установление блокировки на шине I2C. Это позволяет нам контролировать шину:

```
lwhile not i2c.try_lock():
  pass
```

3. Затем мы используем метод `scan()` для обнаружения и вывода адресов всех устройств, найденных на шине I2C:

```
print("I2C addresses found:", [hex(device_address)
      for device_address in i2c.scan()])
```

4. Собрав все это вместе, мы имеем следующий код:

```
import time
import board
import busio

i2c = busio.I2C(board.GP9, board.GP8)

while not i2c.try_lock():
  pass

try:
  while True:
      print("I2C addresses found:", [hex(device_address)
            for device_address in i2c.scan()])
      time.sleep(2)

finally:
    i2c.unlock()
```

5. Сохраните предыдущий пример кода как *code.py* на вашем Pico, и он должен вывести адреса устройств, обнаруженных на шине I2C. На рис. 2.27

код использовался для определения датчика температуры, подключенного к Pico.

```
CircuitPython REPL
I2C addresses found: ['0x38']
I2C addresses found: ['0x38']
I2C addresses found: ['0x38']
I2C addresses found: ['0x38']
I2C addresses found: ['0x38']
I2C addresses found: ['0x38']
I2C addresses found: ['0x38']
```

Рисунок 2.27. Устройства, обнаруженные на шине I2C

Это может пригодиться при устранении неполадок в соединениях макетной платы. В следующем разделе мы собираемся отобразить данные о температуре и влажности с помощью OLED-дисплея.

Отображение данных о температуре на дисплее с SPI-интерфейсом

В этом разделе мы будем использовать OLED-дисплей с драйвером SSD1306. Последовательный периферийный интерфейс (SPI) отобразит данные о температуре и влажности, которые мы получили в предыдущем разделе. Прежде чем мы начнем, давайте вкратце взглянем на шину SPI.

Последовательный периферийный интерфейс (SPI)

SPI обычно используется для сопряжения периферийных устройств, таких как карты microSD, устройства памяти, датчики и т. д. Это синхронный интерфейс, подобный интерфейсу I2C, где мы используем тактовый сигнал для синхронизации хоста и периферийного устройства. Шина SPI обычно содержит следующие выводы:

- *Chip Select* (CS) – позволяет хосту (мастеру) выбрать периферийное устройство, с которым он хочет инициировать связь. Возможность выбора микросхемы через вывод CS позволяет подключать несколько устройств к одной шине;
- *вывод данных* (DO) – вывод, используемый хост-устройством для передачи данных на периферийное устройство;
- *ввод данных* (DI) – вывод, используемый хост-устройством для приема данных с периферийного устройства[19].

[19] Стандартные названия выводов SPI для обмена данными: MOSI (Master Out Slave In; здесь DO) и MISO (Master In Slave Out; здесь DI). Они будут более привычными, например, тем, кто имел дело с Arduino. См. также далее раздел «*Настройка беспроводного модуля ESP32*». – *Прим. перев.*

Для более глубокого изучения шины SPI мы рекомендуем прочитать это описание, чтобы лучше понять устройство интерфейса: **https://3d-diy.ru/wiki/arduino-moduli/interfeys-peredachi-dannykh-spi/**.

Подключение дисплея

В этом примере мы используем OLED-дисплей 128×32 SPI от Adafruit (ссылка: **https://bit.ly/3yAwHJY**). Дисплей должен быть подключен, как показано ниже, где слева от стрелки указан вывод Pico, справа – вывод OLED-дисплея:

- 3,3V → VIN;
- GP2 → CLK;
- GP3 → DATA;
- GP11 → CS;
- GP13 → D/C;
- GP12 → RST;
- GND → GND.

Схема подключения OLED-интерфейса к Pico показана на следующем рисунке:

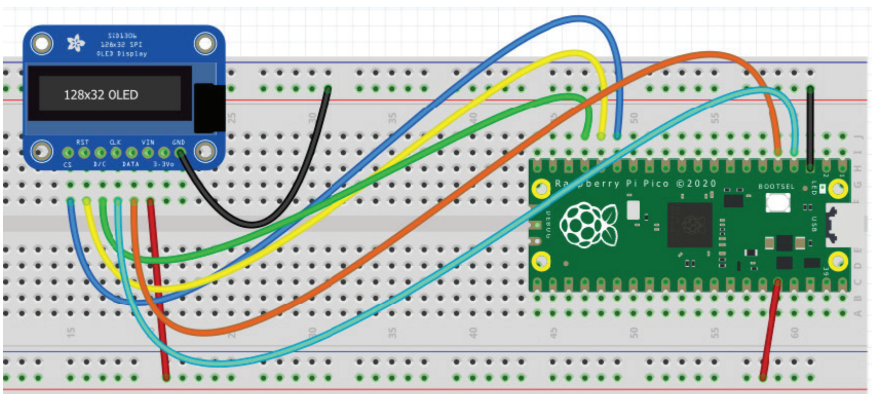


Рисунок 2.28. Схема подключения OLED-дисплея

> **Подключение датчика температуры**
>
> На этой схеме мы не показали подключение датчика температуры. Мы предполагаем, что вы не отсоединяли датчик HTU21D.

В следующем разделе рассмотрим код, необходимый для отображения данных о температуре и влажности.

Отображение данных о температуре и влажности

Пример кода, обсуждаемый в этом разделе, доступен для скачивания здесь: **https://bit.ly/3byo6NA**.

О форматировании строк

Во многих фрагментах кода далее вы заметите, что некоторые операторы не умещаются в одну строку и переносятся на вторую. Обратите внимание, что разрывов строк в этих местах нет, две строки получаются только вследствие форматирования текста.

1. Первым шагом является импорт необходимых модулей:

```
import time
import board
import busio
import displayio
import terminalio
from adafruit_display_text import label
import adafruit_displayio_ssd1306
from adafruit_htu21d import HTU21D
```

2. Следующим шагом является инициализация шины SPI, OLED-дисплея, шины I2C и датчика температуры:

```
displayio.release_displays()

spi = busio.SPI(board.GP2, MOSI=board.GP3)
i2c = busio.I2C(board.GP9, board.GP8)

oled_cs = board.GP11
oled_dc = board.GP13
oled_reset = board.GP12

display_bus = displayio.FourWire(spi, command=oled_dc,
chip_select=oled_cs, reset=oled_reset, baudrate=1000000)

WIDTH = 128
HEIGHT = 32

sensor = HTU21D(i2c)
display = adafruit_displayio_ssd1306.SSD1306(display_bus,
width=WIDTH, height=HEIGHT)
```

3. Теперь мы можем начать отображать температуру в цикле на дисплее:

```
while True:
  splash = displayio.Group()

  text = "Temperature: {:.1f} C".format(sensor.temperature)
  text_area = label.Label(terminalio.FONT, text=text, c olor=0xFFFFFF, x=0, y=4)
```

```
splash.append(text_area)
display.show(splash)

text = "Humidity: {:.1f} %".format(sensor.relative_humidity)
text_area = label.Label(terminalio.FONT, text=text, c olor=0xFFFFFF, x=0, y=17)
splash.append(text_area)
display.show(splash)

time.sleep(2)
```

Если вы сохраните предыдущий пример кода как *code.py* на вашем Pico, то увидите, что температура постоянно обновляется на OLED-дисплее. Теперь мы рассмотрим вариант платы Pico, который поставляется со встроенным дисплеем.

Плата LILYGO RP2040

В предыдущем разделе мы обсуждали подключение дисплея к Pico через интерфейс SPI, теперь рассмотрим альтернативу. Плата LILYGO RP2040 (показана на рис. 2.29) оснащена ЖК-дисплеем с диагональю 1,14 дюйма и 4 МБ внешней флеш-памяти.

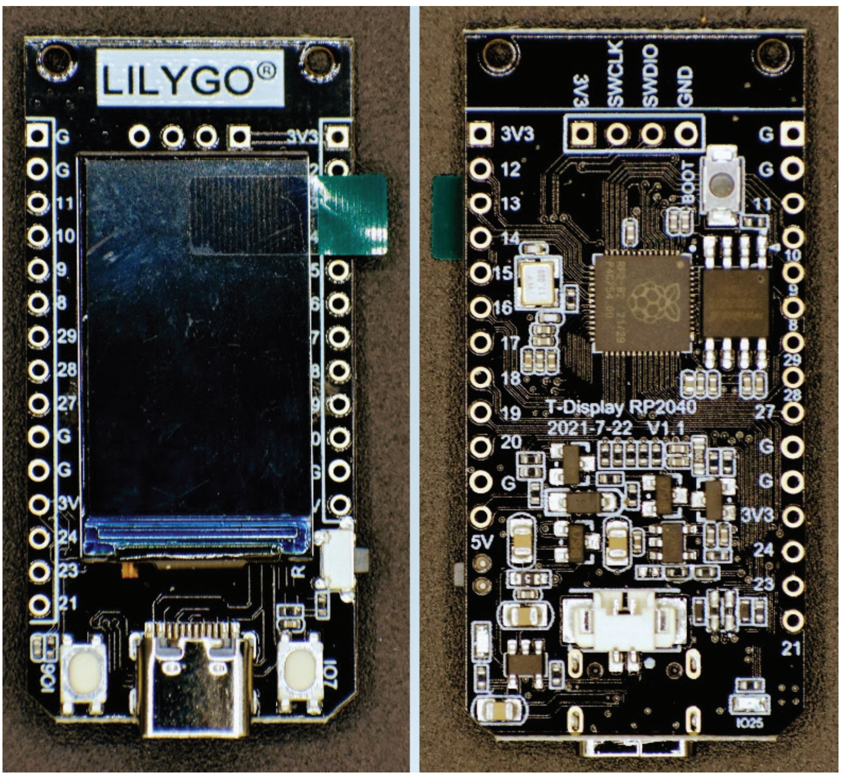


Рисунок 2.29. Плата LILYGO RP2040

Плата доступна по адресу **https://bit.ly/3EZvAWz**. Вы можете заставить дисплей на этой плате работать с приведенным ранее примером кода – нужно только заменить номера выводов (разводка показана на рисунке ниже).

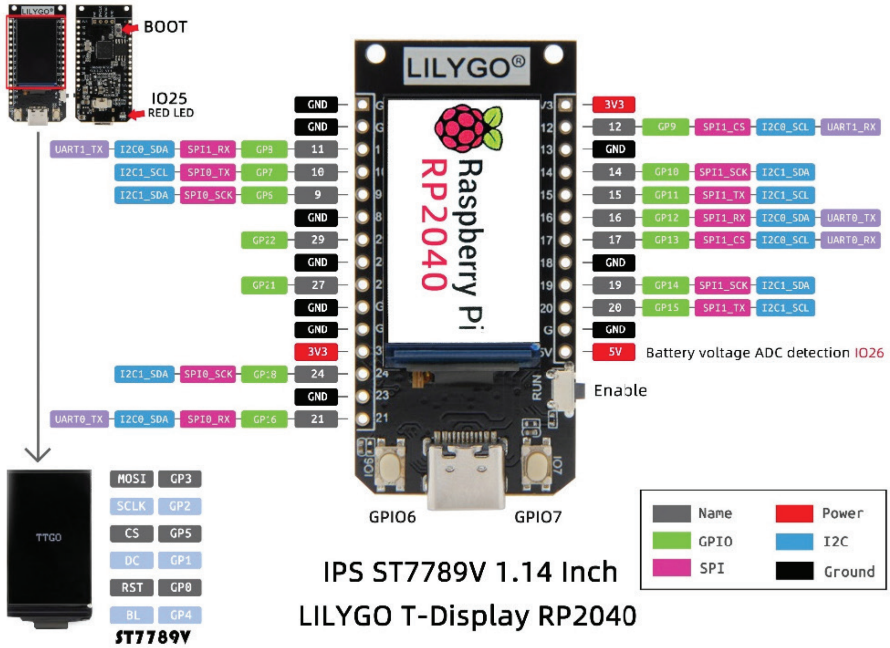


Рисунок 2.30. Выводы платы LILYGO RP2040 (источник изображения: **https://bit.ly/3EZvAWz**)

В следующем разделе мы обсудим настройку беспроводного модуля, который также использует SPI.

Настройка беспроводного модуля ESP32

В этом разделе мы обсудим настройку и тестирование беспроводного модуля ESP32. Мы будем использовать один и тот же беспроводной модуль для подключения к интернету во всех проектах.

Для удобства мы будем использовать Pico Omnibus – двойной расширитель от Pimoroni. Pico и беспроводной модуль устанавливаются на расширитель, как показано на рис. 2.31.

Если у вас нет расширителя, для связи с ESP32 используются следующие контакты:

- CS → GP7;
- Ready → GP10;
- Reset → GP11;
- SCK → GP18;
- MOSI → GP19;
- MISO → GP16.

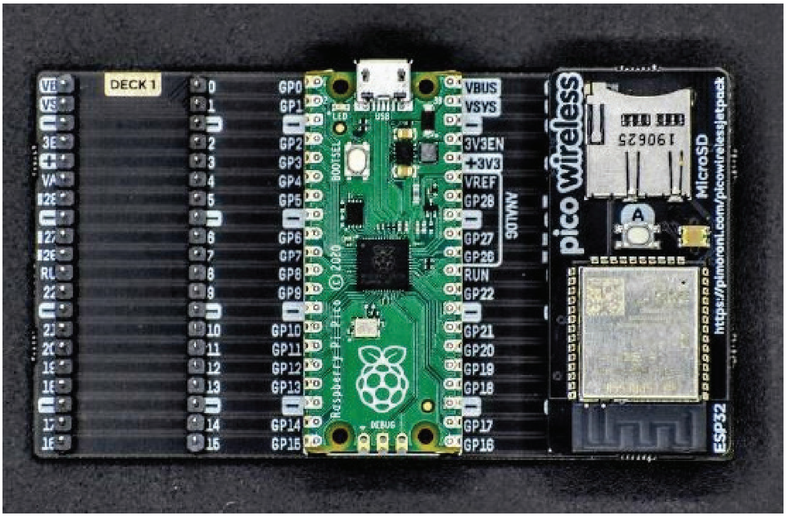


Рисунок 2.31. Pico и беспроводной модуль ESP32, установленные на Pico Omnibus – двойной расширитель

Схема подключения также показана на рис. 2.32[20]:

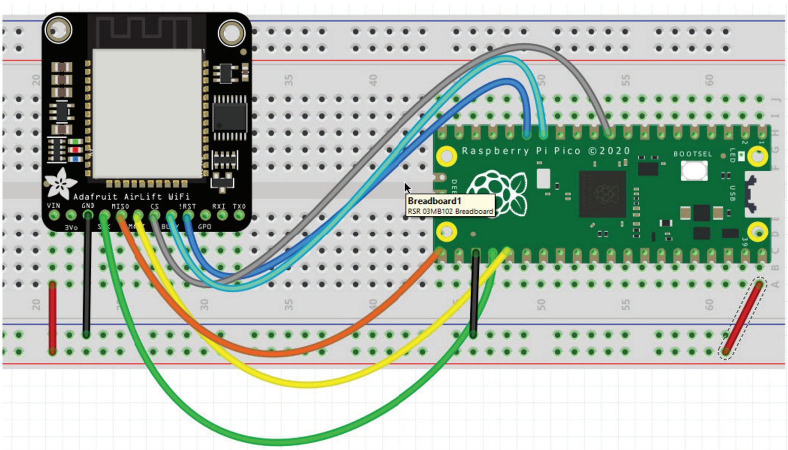


Рисунок 2.32. Сопряжение модуля ESP32 с Raspberry Pi Pico

Чтобы подключиться к сети Wi-Fi с помощью беспроводного модуля, создайте файл с именем *secrets.py* на вашем Pico и сохраните следующую информацию:

[20] Отметим, что на рис. 2.31 и 2.32 показаны, формально говоря, разные модули от разных компаний и в разном конструктивном исполнении. Однако первый вариант (рис. 2.31) также выпускается компанией Adafruit в конструктиве Feather под названием Adafruit AirLift (см. **https://learn.adafruit.com/adafruit-airlift-featherwing-esp32-wifi-co-processor-featherwing**). Вариант, показанный на рис. 2.32 с тем же названием, очевидно, имеет идентичную функциональность (модуль создан специально для подключения к Pico, как беспроводной сопроцессор). То, что все это одинаковые модули, подтверждается использованием кода именно от Adafruit далее. – *Прим. перев.*

```
secrets = {
'ssid' : 'home ssid',
'password' : 'my password',
'timezone' : "America/New_York",
}
```

Убедитесь, что вы сохранили фрагмент кода, включив в него свои учетные данные Wi-Fi. После этого можно проверить подключение к сети Wi-Fi и получение информации с какого-нибудь веб-сайта. Мы внесли изменения в код, предоставленный обучающей системой Adafruit для тестирования беспроводного модуля. Пример кода доступен при загрузке главы как *code_wifi.py* (ссылка: **https://bit.ly/3v4Co1j**).

Мы внесли следующие изменения в фрагмент кода:

```
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)
spi = busio.SPI(board.GP18, board.GP19, board.GP16)
```

Если сохранить файл как *code.py* и запустить его, вы должны увидеть следующее сообщение:

Рисунок 2.33. Тест Wi-Fi

Теперь, когда мы протестировали беспроводной модуль, в следующей главе перейдем к взаимодействию со светодиодной лентой NeoPixel[21].

Итоги

В этой главе мы обсудили соединение двух плат Pico через интерфейс UART и подключение датчиков через интерфейс I2C. Они пригодятся в проектах, которые мы собираемся рассматривать в следующих главах. Примеры, рассмотренные в этой главе, должны помочь вам в работе с периферийными устройствами в ваших проектах, поскольку мы обсуждали наиболее часто используемые интерфейсы связи.

В следующей главе мы обсудим домашнюю автоматизацию и использование Pico для управления бытовой техникой. Вы заметите, что сложность проектов возрастает с каждой главой.

[21] NeoPixel – торговая марка Adafruit для лент индивидуально адресуемых RGB-светодиодов. – *Прим. перев.*

Глава 3

Проекты домашней автоматизации

В этой главе мы обсудим проекты домашней автоматизации с использованием Raspberry Pi Pico. Мы собираемся работать над простыми проектами, которые можно было бы выполнить за выходные и тем улучшить среду обитания. Мы начнем с того, что научимся подключать датчики для обнаружения событий в доме. Затем мы начнем размещать информацию о событиях, обнаруженных датчиком, в облаке. Мы также обсудим управление бытовой техникой в домашних условиях с помощью Pico.

Кроме проектов с использованием Raspberry Pi Pico, мы также представим плату RP2040 Connect от Arduino. Примеры, обсуждаемые в этой главе, могут быть выполнены с использованием RP2040 Connect с некоторыми изменениями в коде.

В этой главе мы рассмотрим следующие основные темы:

- установка необходимых библиотек;
- подключение датчиков;
- управление приборами;
- размещение данных в облаке;
- управление светодиодными лентами;
- представляем RP2040 Connect.

Технические требования

Для этой главы требуются следующие компоненты:

- Raspberry Pi Pico (ссылка: **https://bit.ly/3AJtoAf**);
- Pico Omnibus – двойной расширитель (ссылка: **https://bit.ly/3sr2GJR**);
- беспроводной модуль Pico (ссылка: **https://bit.ly/3yPuoT9**);
- модуль релейного управления питанием для «интернета вещей» IoT Power Relay (ссылка: **https://www.sparkfun.com/products/14236**[22]);

[22] По ссылке предлагается управляемое от микроконтроллеров сдвоенное реле с американскими/японскими стандартами сетевого электропитания и розеток для подключения. Выпускается множество аналогичных продуктов в расчете на европейские/российские стандарты электросети (см. например, в каталоге «Чипа и Дипа»), однако сетевые розетки для них, как правило, придется устанавливать самостоятельно. При подборе аналогов следует иметь в виду, какой мощности нагрузку вы собираетесь подключать (рекомендуемая мощность нагрузки, которую может переключать указанный в ссылке релейный модуль по каждому из каналов, составляет до 3,5 кВт, что, конечно, перекрывает любые домашние применения). – *Прим. перев.*

- герконовый выключатель (ссылка: **https://bit.ly/3uS3dE9**)[23];
- светодиодная лента или кольцо NeoPixel (ссылка: **https://bit.ly/3pp95nk**)[24].

Примеры кода, обсуждаемые в этой главе, доступны здесь: **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIY-Workshop/tree/main/chapter_03**.

Видеоролики с кодом в действии для этой главы можно посмотреть по адресу **https://bit.ly/3kLsX2B**.

> **Установка CircuitPython**
>
> Мы предполагаем, что вы установили CircuitPython на свой Pico. Если вы незнакомы с процессом установки, мы рекомендуем следовать инструкциям из главы 1 «*Начало работы с Raspberry Pi Pico*».

Установка необходимых библиотек

В этом разделе мы установим библиотеки, необходимые для оборудования, описанного в данной главе, включая NeoPixel LED и беспроводной модуль для Raspberry Pi Pico. Все библиотеки являются частью пакета Adafruit CircuitPython. Пакет можно загрузить в виде ZIP-файла с **https://circuitpython.org/libraries**. Мы используем версию пакета, предназначенную для CircuitPython 6.x.x.

После загрузки ZIP-файла извлеките его содержимое, чтобы мы могли скопировать библиотеки, необходимые для проекта.

NeoPixel LED

Нам нужен двоичный файл *neopixel.mpy* для управления светодиодной лентой NeoPixel. Скопируйте его в папку *lib*.

Беспроводной модуль

Нам нужна библиотека *adafruit_esp32spi* для беспроводного модуля. Скопируйте папку (с тем же именем) в папку *lib* вашего Pico. Нам также понадобится двоичный файл *adafruit_requests.mpy* из пакета библиотеки.

> **Настройка беспроводного модуля**
>
> В главе 2 «*Последовательные интерфейсы и их приложения*» мы подробно обсудили настройку беспроводного модуля. Рекомендуем вам прочитать эту главу, а затем настроить беспроводной модуль.

[23] Имеется множество аналогов, доступных в России (например: **https://www.chipdip.ru/product0/9000749865**). – *Прим. перев.*

[24] Доступно в российских интернет-магазинах, например **https://www.chipdip.ru/catalog/ext?gq=NeoPixel**. – *Прим. перев.*

Подключение контактных датчиков

В этом разделе мы рассмотрим подключение контактных датчиков к Pico, в частности геркона. Все мы бывали в ситуациях, когда либо не слышали, как открывается дверь, либо не знаем, что кто-то неправильно закрыл дверь. Мы собираемся обсудить обнаружение таких ситуаций с помощью геркона.

Герконовый выключатель – это тип переключателя, который замыкает контакт в присутствии магнитного поля. На рис. 3.1 показан герконовый выключатель, продаваемый компанией Adafruit, который можно использовать в качестве дверного датчика. Датчик состоит из двух частей – магнита и геркона.

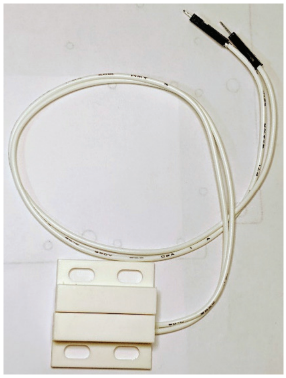


Рисунок 3.1. Герконовый выключатель

Магнит обычно устанавливается на двери, в то время как геркон устанавливается на дверной коробке. Для данного датчика магнит должен находиться в пределах ~13 мм от геркона, тогда выключатель замыкается. Мы собираемся использовать этот принцип для определения состояния двери.

Шаги по подключению датчика двери к Pico включают следующее:

1. Первым шагом является подключение датчика двери к Pico, как показано на схеме:

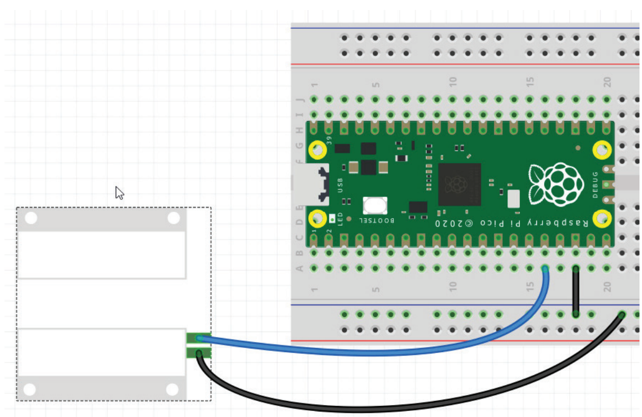


Рисунок 3.2. Схема подключения датчика двери к Pico

На схеме один провод герконового выключателя подключен к контакту GP12, другой подключен к контакту GND.

2. Мы собираемся рассмотреть пример кода с интерпретатором Python. Пример кода, обсуждаемый в этом разделе, доступен для загрузки вместе с этой главой в файле *code_door_sensor.py*. Для работы с ним необходимо запустить интерпретатор Python на Pico. Откройте терминал последовательного порта нажатием кнопки **Serial** на встроенном Mu IDE с панели инструментов в верхней части IDE.

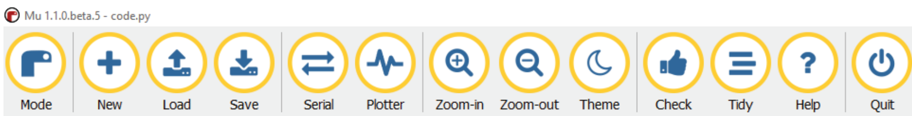


Рисунок 3.3. Расположение кнопки **Serial** на панели инструментов

3. Если вы видите текст, показанный на следующем рисунке, нажмите любую клавишу, чтобы войти в интерпретатор Python:

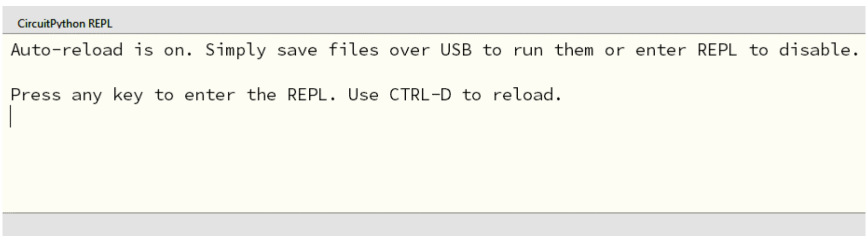


Рисунок 3.4. Терминал последовательного порта на Mu IDE

4. Если на Pico запущен код, может появиться пустой экран (как показано на следующем рисунке) или что-то напечатается. Нажмите <**Ctrl**+**C**>, чтобы прервать выполнение кода.

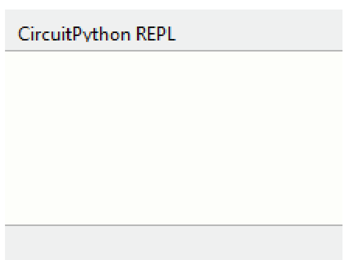


Рисунок 3.5. Терминал последовательного порта без вывода на Mu IDE

5. Интерпретатор Python будет выглядеть примерно так, как показано на следующем рисунке:

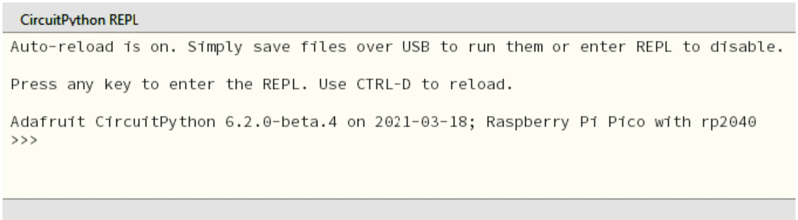


Рисунок 3.6. Интерпретатор Python на Pico

6. Первым шагом является импорт необходимых модулей. Для этого примера мы импортируем модули `board`, `time` и `digitalio`:

   - модуль `board` содержит все определения контактов для Raspberry Pi Pico. Это позволяет нам определить вывод, который мы хотим использовать в этом примере;
   - модуль `time` содержит функцию задержки. Он используется для вставки преднамеренной задержки в выполнении кода;
   - модуль `digitalio` используется для настройки функции определенного вывода. В этом примере мы будем применять модуль для настройки вывода в качестве входного контакта.

   Импорт модулей выполняется в следующих строках кода:

```
>>> import board
>>> import digitalio
>>> import time
```

7. Поскольку геркон подключен к выводу GP12, мы собираемся инициализировать его как входной вывод:

```
>>> switch = digitalio.DigitalInOut(board.GP12)
>>> switch.direction = digitalio.Direction.INPUT
```

8. Вывод GP12 (к которому мы подключили геркон) при разомкнутом герконе «висит в воздухе», и нам нужно привязать его к фиксированному состоянию. Для этого мы используем внутренний подтягивающий резистор. Подтягивающие (pull-up) резисторы позволяют привязать входные контакты к известному состоянию высокого уровня. Это позволяет предотвратить шум на контакте, который будет рассматриваться кодом как изменение состояния двери. Привязка вывода GP12 к высокому уровню осуществляется следующим образом:

```
>>> switch.pull = digitalio.Pull.UP
```

   На следующей схеме показан вывод GP12 в конфигурации с подтягивающим резистором:

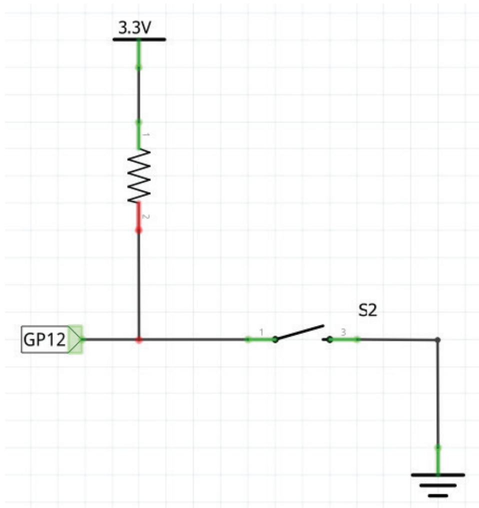


Рисунок 3.7. Конфигурация вывода GPIO с подтягивающим резистором

9. В качестве альтернативы вы можете привязать вывод GPIO к заземляющему (pull-down) резистору, и конфигурация вывода GPIO будет установлена следующим образом:

```
>>> switch.pull = digitalio.Pull.DOWN
```

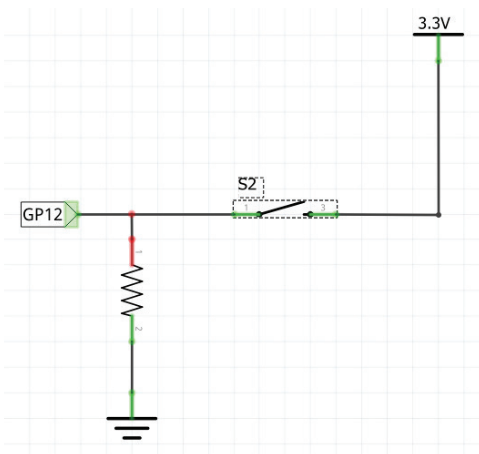


Рисунок 3.8. Конфигурация вывода GPIO с заземляющим резистором

10. Мы собираемся установить GP12 в состояние с подтягивающим резистором. Теперь давайте прочитаем состояние переключателя в бесконечном цикле и проверим наличие дверного магнита следующим образом[25]:

[25] Многоточие, использованное в этом фрагменте кода, – специальная константа в языке Python, которая может служить синтаксически правильной заменой ненаписанного кода. Иными словами, авторы хотят показать, что на месте многоточий можно располагать другие операторы, выполняющие какие-то действия, выполнение программы от этого не изменится. – *Прим. перев.*

```
>>> while True:
...     print(switch.value)
...     time.sleep(1)
```

11. Теперь терминал должен начать печатать состояние дверного датчика с интервалом в 1 секунду. Когда магнит присутствует (другими словами, дверь закрыта), программа на экране печатает *False*. Когда дверь открыта, она печатает *True*, как видно на следующем рисунке[26]:

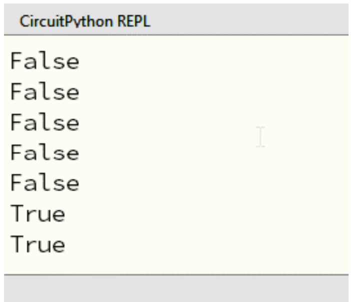


Рисунок 3.9. Состояния дверного датчика, выводимые на экран

12. Теперь, когда мы протестировали наш код с помощью интерпретатора Python, мы можем сохранить код как *code.py* и наблюдать за состоянием дверного датчика, выводимым в бесконечном цикле:

```
import board
import digitalio
import time

switch = digitalio.DigitalInOut(board.GP12)
switch.direction = digitalio.Direction.INPUT
switch.pull = digitalio.Pull.UP

while True:
    print(switch.value)
    time.sleep(1)
```

Вы когда-нибудь задумывались, где используются такие дверные датчики? Например, вы могли слышать дверной звонок во время похода в круглосуточный магазин – они используют какую-то форму дверного датчика[27]. На следую-

[26] Логическое значение *False* эквивалентно нулю (низкому уровню), *True* – единице (высокому уровню), что совпадает со значениями логических уровней на выводе при замкнутом и незамкнутом герконе соответственно. – *Прим. перев.*

[27] В магазинах и офисах контактные датчики, подобные описываемому герконовому, обычно используются в охранном режиме (как на фото рис. 3.10 ниже). В режиме повседневной эксплуатации для сигнализации о посетителях чаще применяется какой-либо бесконтактный датчик – пересечение инфракрасного луча, PIR-датчики присутствия (**https://osensorax.ru/posiciya/pir-datchik**), ультразвуковые и высокочастотные датчики. Иногда в таком режиме используют видеокамеры со специальным программным обеспечением. – *Прим. перев.*

щем фото вы можете видеть дверной датчик, используемый для мониторинга ворот с контролем доступа:

Рисунок 3.10. Датчик на воротах с контролем доступа

Мы собираемся использовать вывод состояния дверного датчика, чтобы определять состояние двери и сообщать о событиях в облако или передавать сигнал управляющим устройствам. В следующем разделе мы обсудим управление приборами внутри помещения на основе состояния дверного датчика.

Управление приборами

В этом разделе мы будем управлять устройствами с помощью продукта под названием IoT Power Relay (показано на рис. 3.11)[28]. Мы собираемся внести некоторые изменения в пример кода, рассмотренный в предыдущем разделе, чтобы включить свет, когда дверь открыта, и наоборот.

> **Приборы, управляемые силовым реле**
>
> Мы используем реле питания для проектов, поскольку оно обеспечивает безопасный способ управления электроприборами переменного тока. Рекомендуется управлять простыми резистивными нагрузками, такими как торшер[29].

На фото ниже показан модуль релейного управления питанием IoT Power Relay. Зеленая клеммная колодка в нижней части рисунка используется для управления реле.

[28] Напоминаем, что указанное реле рассчитано на американские стандарты электропитания. В сноске 22 на стр. 75 приведены общие рекомендации по замене. – *Прим. перев.*

[29] Авторы имеют в виду, что управление силовыми нагрузками, содержащими реактивные элементы (например, двигателями), требует более аккуратного подхода и специфических знаний. – *Прим. перев.*

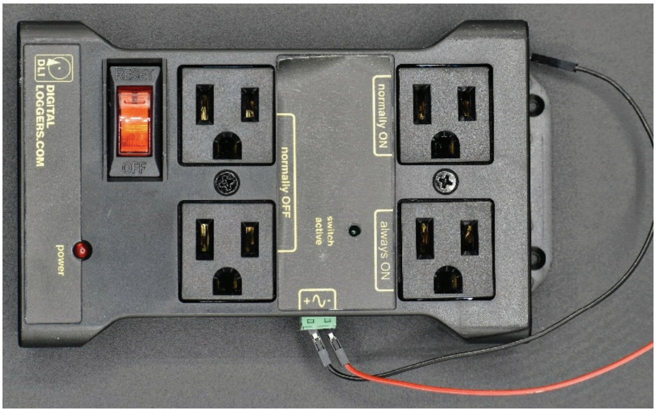


Рисунок 3.11. IoT Power Relay (источник изображения: ***sparkfun.com***)

Реле питания IoT позволяет управлять приборами переменного тока (AC) сигналом с уровнем 3,3 V. Модуль поставляется с необходимой защитой для безопасного сопряжения Pico с реле. Для управления реле мы будем использовать вывод GP11. Схема управления силовым реле показана на рис. 3.12. Положительная клемма вывода управления подключается к GP11, а отрицательная – к GND.

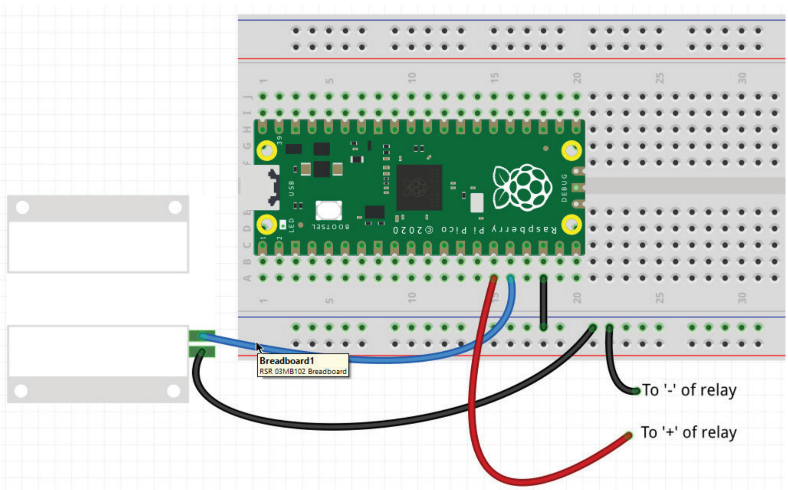


Рисунок 3.12. Схема для подключения релейного модуля

Для управления устройством мы собираемся внести некоторые изменения в пример кода, рассмотренный в предыдущем разделе. Порядок действий следующий.

1. Первым шагом является инициализация вывода GP11 в качестве выходного вывода:

```
>>> relay = digitalio.DigitalInOut(board.GP11)
>>> relay.direction = digitalio.Direction.OUTPUT
```

2. Теперь давайте изменим цикл `while`, чтобы включить реле в зависимости от состояния двери. В предыдущем разделе мы заметили: когда дверь открыта, состояние переключателя равно `True`. То есть нам нужно включить свет, если состояние переключателя равно `True`:

```
import board
import digitalio
import time

switch = digitalio.DigitalInOut(board.GP12)
switch.direction = digitalio.Direction.INPUT
switch.pull = digitalio.Pull.UP
relay = digitalio.DigitalInOut(board.GP11)
relay.direction = digitalio.Direction.OUTPUT

while True:
  if switch.value:
      relay.value = True
  else:
      relay.value = False
  time.sleep(1)
```

Когда вы включаете реле с помощью шнура питания, вы должны услышать, как реле срабатывает с тикающим звуком[30]. Попробуйте подключить к реле настольную лампу. Она должна загораться всякий раз, когда вы открываете дверь. В случае если вы незнакомы с реле, мы рекомендуем статью из Википедии: **https://ru.wikipedia.org/wiki/Электромагнитное_реле**.

Мы рассмотрели устройства управления, основанные на определении состояния сенсоров микроконтроллером RP2040 Pico. В следующем разделе мы обсудим размещение состояний датчиков дверей в облаке.

Размещение состояний датчиков в облаке

Размещение состояний датчиков в облачном хранилище применяется, например, для получения уведомлений по электронной почте и мобильной связи, когда дверь открыта. Мы предполагаем, что вы настроили беспроводной модуль, следуя инструкциям из главы 2 «*Последовательные интерфейсы и их приложения*».

Настройка Adafruit IO

В этом разделе мы обсудим настройку Adafruit IO для нашего проекта. Adafruit IO – это облачный сервис, предоставляемый компанией Adafruit Industries, который позволяет Raspberry Pi Pico быть подключенным к интернету, размещать данные с датчиков, а также управлять устройствами из любой точки сети. Настройка включает в себя следующие шаги.

[30] Разумеется, при условии, что на управляющие контакты подан высокий уровень (`True`). – *Прим. перев.*

> **Дополнительный ресурс**
>
> Мы написали этот раздел, используя **https://learn.adafruit.com** в качестве источника. Если вы столкнетесь с проблемами, обратитесь по ссылкам, приведенным далее.

1. Первый шаг – создать для себя учетную запись на сайте **https://io.adafruit.com/**. Базовый план является бесплатным, но существует также подписка, доступная следующим образом:

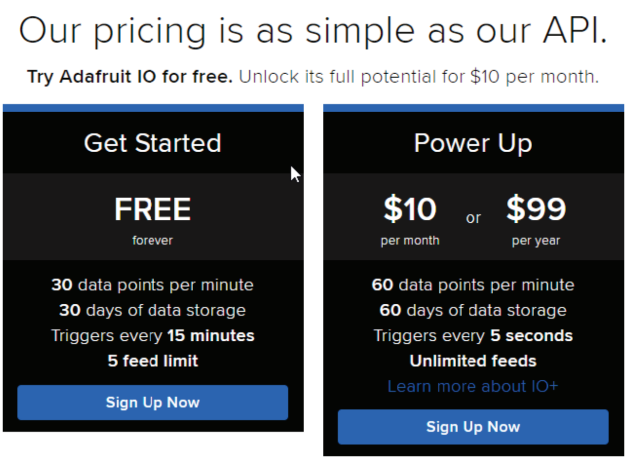


Рисунок 3.13. Регистрация на ***io.adafruit.com***

2. После того как вы создали учетную запись, извлеките свой ключ через пункт меню **My Key**, как показано на рис. 3.14. Скопируйте имя пользователя и информацию об активном ключе (ссылка: **https://bit.ly/3vK6QfK**).

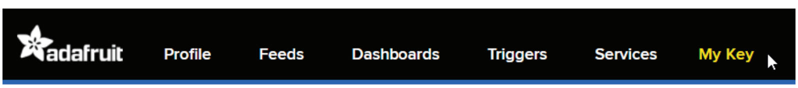


Рисунок 3.14. Извлечение ключа Adafruit IO

3. Нам нужно сохранить полученную информацию в *secrets.py* следующим образом (ссылка: **https://bit.ly/2SFbkHn**):

```
secrets = {
  'ssid' : '_your_wifi_ssid',
  'password' : '_your_wifi_password',
  'aio_username' : '_your_adafruit_io_username',
  'aio_key' : '_your_big_huge_super_long_aio_key_'
}
```

В следующем разделе мы обсудим размещение сведений о событиях в облачном сервисе Adafruit IO.

Порядок размещения

Теперь мы рассмотрим размещение данных с использованием Raspberry Pi Pico. Порядок размещения включает в себя следующее.

1. Загрузите пакет кода с **https://bit.ly/3gNPadR** и извлеките файлы.
2. Скопируйте содержимое папки *lib* в папку *lib* устройства CircuitPython.
3. Мы собираемся изменить сценарий, описанный в разделе об измерении температуры, чтобы разместить какое-либо значение в ленте температуры. Первым шагом является импорт необходимых модулей:

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as
socket
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_io.adafruit_io import IO_HTTP, AdafruitIO_RequestError
```

4. Затем мы импортируем учетные данные Wi-Fi для проекта следующим образом:

```
# Get wifi details and more from a secrets.py file
try:
  from secrets import secrets
except ImportError:
  print("WiFi secrets are kept in secrets.py, please add them there !")
  raise
```

5. Далее мы инициализируем модуль Wi-Fi:

```
# ESP32 pins
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)

spi = busio.SPI(board.GP18, board.GP19, board.GP16)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

requests.set_socket(socket, esp)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
  print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for i in esp.MAC_address])

for ap in esp.scan_networks():
  print("\t%s\t\tRSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))
```

6. Затем мы подключаемся к сети на основе информации, полученной из *secrets.py*:

```
print("Connecting to AP...")
while not esp.is_connected:
  try:
    esp.connect_AP(secrets["ssid"], secrets["password"])
  except RuntimeError as e:
    print("could not connect to AP, retrying: ", e)
    continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
```

7. Мы инициализируем соединение с сокетом[31] и инициализируем объект ввода-вывода *Adafruit IO object* следующим образом:

```
socket.set_interface(esp)
requests.set_socket(socket, esp)

aio_username = secrets["aio_username"]
aio_key = secrets["aio_key"]

# Initialize an Adafruit IO HTTP API object
io = IO_HTTP(aio_username, aio_key, requests)
```

8. Затем мы инициализируем новый поток измерений температуры, на случай если она не измерялась ранее:

```
try:
  # Get the 'temperature' feed from Adafruit IO
  temperature_feed = io.get_feed("temperature")
except AdafruitIO_RequestError:
  # If no 'temperature' feed exists, create one
  temperature_feed = io.create_new_feed("temperature")
```

9. Отправляем случайное значение температуры из потока, как показано здесь:

```
# Send random integer values to the feed
random_value = randint(0, 50)
print("Sending {0} to temperature feed...".format(random_value))
io.send_data(temperature_feed["key"], random_value)
print("Data sent!")
```

[31] Сокет (socket, букв. *розетка*, *разъем*) – элемент программного интерфейса для обеспечения обмена данными между процессами, конечная точка для отправки и приема данных по сети. Подробнее см., например, **http://rsdn.org/article/unix/sockets.xml**. – *Прим. перев.*

10. Извлекаем отправленное значение, чтобы проверить, все ли работает:

```
# Retrieve data value from the feed
print("Retrieving data from temperature feed...")
received_data = io.receive_data(temperature_feed["key"])
print("Data from temperature feed: ", received_data["value"])
```

Вы можете посетить сайт **https://io.adafruit.com/** для просмотра сформированной ленты значений температуры.

Следующие шаги

В предыдущей главе мы обсуждали подключение Pico к сети Wi-Fi и размещение случайных значений (согласно примеру Adafruit). Вы можете использовать тот же сценарий для размещения событий датчика двери, удаленного мониторинга растений и т. д. Ознакомьтесь с репозиторием этой книги для получения некоторых примеров.

Управление светодиодными лентами

В этом разделе мы будем подключать светодиодную ленту NeoPixel к Pico. NeoPixels – это последовательно соединенные трехцветные светодиоды, которые можно адресовать индивидуально. NeoPixels выпускаются в различных форм-факторах: в виде твердых полосок, гибких лент и круглых колец. Мы будем обсуждать пример с кольцом NeoPixels (ссылка: **https://bit.ly/3cn5pxj**). Из светодиодной ленты может получиться отличный проект праздничного освещения или контроллер рассеянного света, поэтому изучим ее интерфейс. Наш любимый проект с использованием Pico и NeoPixel LED – настольная лампа (ссылка: **https://bit.ly/3r6ildJ**).

В примерах, обсуждаемых в этом разделе, используются вспомогательные функции от Adafruit.

Кольцо NeoPixel требует трех контактов для подключения, а именно входа данных **Data IN** (**DIN**), питания **Power** (**VCC**) и общего провода **GND** (соединения показаны на рис. 3.15). Вывод питания подключен к выводу 3,3 В Pico, вывод Data IN подключен к GP10 Pico.

Чтобы управлять светодиодным кольцом NeoPixel с помощью Pico, нам нужно выполнить следующее.

1. Первым шагом является импорт необходимых модулей таким образом:

```
import time
import board
import neopixel
```

2. Затем мы инициализируем объект NeoPixel и устанавливаем GP10 в качестве управляющего вывода:

```
pixel_pin = board.GP10
num_pixels = 12
pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.3, auto_write=False)
```

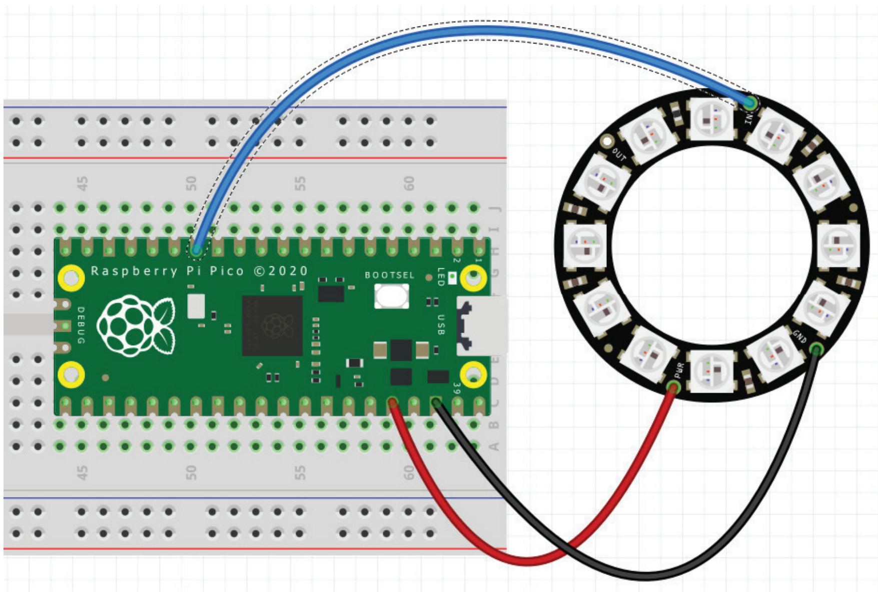


Рисунок 3.15. Схема подключения Pico к кольцу NeoPixel

Давайте воспользуемся вспомогательной функцией из Adafruit следующим образом:

```
def color_chase(color, wait):
  for i in range(num_pixels):
    pixels[i] = color
    time.sleep(wait)
    pixels.show()
  time.sleep(0.5)
```

3. Теперь войдем в бесконечный цикл `while`, переключаясь между цветами вот так:

```
while True:
  color_chase(RED, 0.1) # Increase the number to slow down the color  chase
  color_chase(YELLOW, 0.1)
  color_chase(GREEN, 0.1)
  color_chase(CYAN, 0.1)
  color_chase(BLUE, 0.1)
  color_chase(PURPLE, 0.1)
```

4. Собираем все это вместе:

```
import time
import board
import neopixel

pixel_pin = board.GP10
```

```
num_pixels = 12

pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.3, auto_write=False)

def color_chase(color, wait):
  for i in range(num_pixels):
    pixels[i] = color
    time.sleep(wait)
    pixels.show()
  time.sleep(0.5)

RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)

while True:
  color_chase(RED, 0.1) # Increase the number to slow down the color chase
  color_chase(YELLOW, 0.1)
  color_chase(GREEN, 0.1)
  color_chase(CYAN, 0.1)
  color_chase(BLUE, 0.1)
  color_chase(PURPLE, 0.1)
```

В этом примере мы используем кольцо NeoPixel с 12 светодиодами. Вам придется исправить количество светодиодов `num_pixels` перед сохранением кода на Pico. Можно соединить несколько светодиодных лент вместе и установить их по углам стены в вашем доме или настроить окружающее освещение на разные цвета в зависимости от времени суток.

В следующем разделе мы представим RP2040 Connect, вариант Raspberry PI Pico.

Введение в Arduino Nano RP2040 Connect

В этом разделе мы представим Arduino Nano RP2040 Connect от **Arduino.cc**. RP2040 Connect – это вариант Pico в форм-факторе Arduino Nano. Нам нравится эта плата, потому что она поставляется с интерфейсами Bluetooth и Wi-Fi. Плата также оснащена встроенным акселерометром, гироскопом, RGB-светодиодом и микрофоном. Это открывает огромные возможности для создания прототипов с помощью RP2040 Connect. На рис. 3.16 показана разводка выводов RP2040 Connect (плату и ее описание можно найти здесь: **https://bit.ly/2YZClrY**).

В следующем разделе мы обсудим установку CircuitPython на RP2040 Connect.

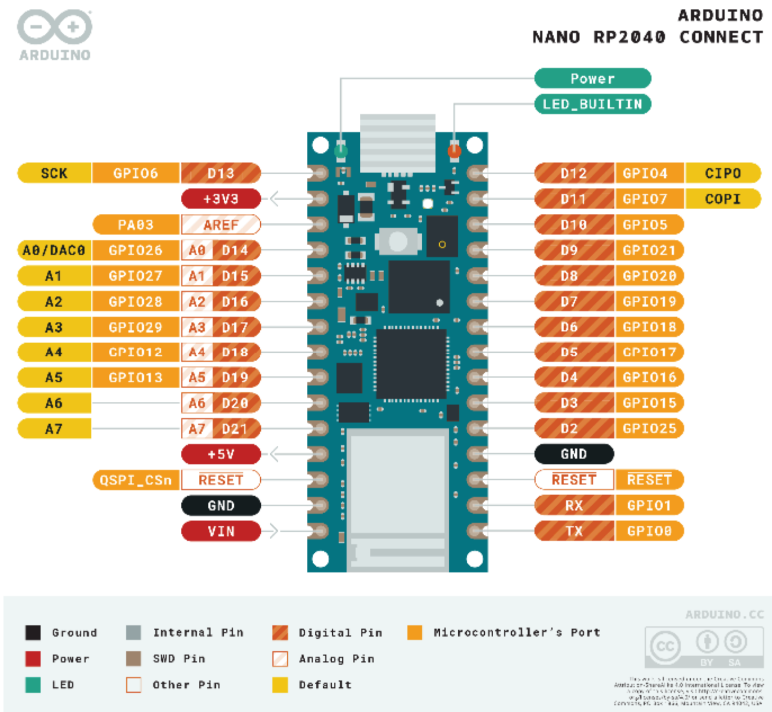


Рисунок 3.16. Разводка выводов Arduino Nano RP2040 Connect (источник изображения: **Arduino.cc**)

Установка CircuitPython на RP2040 Connect

Установка CircuitPython аналогична тому, что мы обсуждали в главе 1 «*Начало работы с Raspberry Pi Pico*», за исключением того, что в этих инструкциях будет дополнительный шаг, где мы укажем флеш-накопитель. Шаги по установке CircuitPython на RP2040 Connect следующие.

1. Установите RP2040 Connect на макетную плату, как показано на следующем фото:

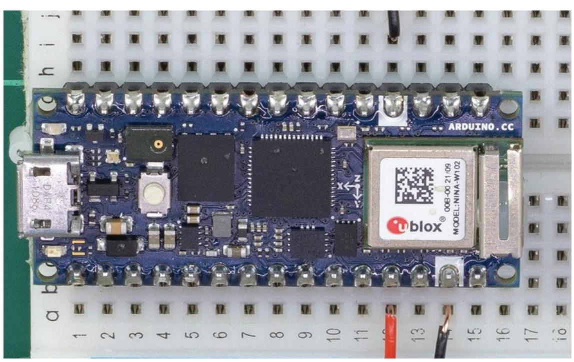


Рисунок 3.17. Установка RP2040 на макетную плату

2. Дважды нажмите на кнопку сброса, чтобы перевести устройство в режим загрузчика (на рис. 3.18 кнопка обведена прямоугольником):

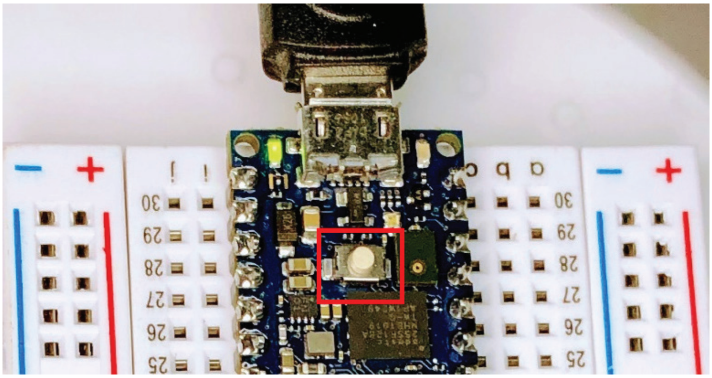


Рисунок 3.18. Кнопка сброса на RP2040 Connect

3. RP2040 Connect обнаруживается как запоминающее устройство. Загрузите двоичный файл CircuitPython с **https://bit.ly/34R29pJ** и скопируйте его на запоминающее устройство.
4. После перепрошивки устройство само себя сбрасывает и обнаруживается как запоминающее устройство с именем CIRCUITPY.

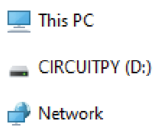


Рисунок 3.19. RP2040 Connect с установленным CircuitPython

5. Теперь мы имеем возможность запрограммировать подключение RP2040 Connect с помощью Mu IDE. На следующем рисунке видим, что IDE обнаружила устройство CircuitPython:

```
Detected new CircuitPython device: Adafruit CircuitPlayground.
```

Рисунок 3.20. RP2040 Connect, обнаруженный Mu IDE

6. Мы можем открыть последовательное соединение, чтобы начать программирование в CircuitPython. На следующем рисунке вы видите версию CircuitPython, работающую на Arduino RP2040 Connect:

```
CircuitPython REPL
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Press any key to enter the REPL. Use CTRL-D to reload.

Adafruit CircuitPython 6.3.0-rc.0 on 2021-05-25; Arduino Nano RP2040 Connect with rp2040
>>>
```

Рисунок 3.21. Интерпретатор Python, работающий на RP2040 Connect

Как вы могли заметить, процесс установки Python на RP2040 Connect очень похож на процесс установки на Raspberry Pi Pico. Давайте обсудим, как подключить этот контроллер к интернету с помощью встроенного модуля Wi-Fi, чтобы продемонстрировать программирование в CircuitPython с использованием такого подключения.

Подключение RP2040 к интернету

В этом разделе мы обсудим подключение RP2040 Connect к интернету. В основном мы собираемся внести некоторые незначительные изменения в образец кода, рассмотренный в предыдущей главе (*code_wifi.py* доступен для скачивания вместе с главой 2). Образец измененного кода доступен для загрузки в файле *code_wifi_rp2040_connect.py*.

Установка предварительных условий

Предварительные условия для настройки встроенного модуля Wi-Fi такие же, как и для беспроводного модуля ESP32, используемого с Pico[32]. Из пакета библиотеки CircuitPython, загруженного ранее, нам нужен *adafruit_esp32spi*. Скопируйте папку (с тем же именем) в папку *lib* вашего Pico. Из того же пакета также понадобится двоичный файл *adafruit_requests.mpy*. Необходимо сохранить учетные данные Wi-Fi в файле с именем *secrets.py* в следующем формате:

```
secrets = {
   'ssid' : 'SSID',
   'password' : 'Password',
   'timezone' : "America/New_York",
}
```

Не забудьте включить в файл свои учетные данные Wi-Fi. Теперь мы изменим пример кода, содержащийся в *code_wifi.py*.

Изменение кода

Эти изменения минимальны: мы просто изменим определения выводов между строками 25 и 29. Первоначальное определение было следующим:

```
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)

spi = busio.SPI(board.GP18, board.GP19, board.GP16)
```

Модификация в соответствии с определениями контактов Arduino RP2040 Connect выглядит таким образом:

```
# ESP32 pins
esp32_cs = DigitalInOut(board.CS1)
esp32_ready = DigitalInOut(board.ESP_BUSY)
```

[32] Плата Arduino Nano RP2040 Connect содержит встроенный модуль ESP32Wi-Fi. – *Прим. перев.*

```
esp32_reset = DigitalInOut(board.ESP_RESET)

spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)
```

Если сохранить файл как *code.py* и запустить его, вы должны увидеть следующее:

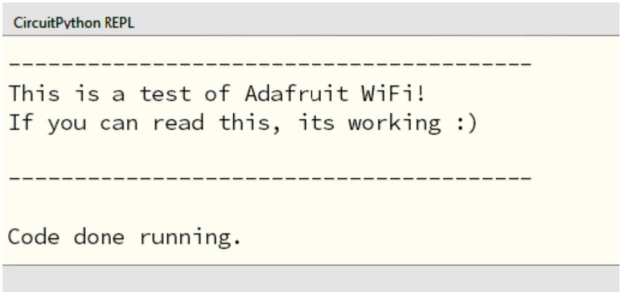


Рисунок 3.22. Тест подключения Wi-Fi RP2040 Connect

Из этого примера вы могли заключить, что программирование в CircuitPython возможно с любым вариантом Raspberry Pico, при внесении некоторых незначительных изменений в код.

> **Задание читателю**
>
> Если у вас есть RP2040 Connect, выберите любой пример кода, рассмотренный ранее в этой главе, и попробуйте использовать его с RP2040 Connect. Убедитесь, что у вас есть разводка выводов платы для внесения изменений в код.

Итоги

В этой главе мы обсуждали подключение датчиков к контактам Pico ввода/вывода общего назначения (GPIO), включение и выключение устройств с помощью блока реле, подключение светодиодных лент к Pico и публикацию событий датчиков в облаке. Мы разбирали управление светодиодной лентой, использующее вариант Pico под названием Arduino Nano RP2040 Connect.

Примеры, описанные в этой главе, были выбраны в первую очередь в качестве примеров, которые можно было бы рассматривать как законченные проекты выходного дня.

В следующей главе мы с удовольствием займемся садоводством. Скоро увидимся!

Глава 4

Весело проводите время в саду!

В этой главе мы обсудим подключение датчика влажности почвы к Raspberry Pi Pico и публикацию значений влажности и температуры почвы в ThingSpeak (**https://thingspeak.com**). ThingSpeak – это платформа для **IoT** (интернета вещей), которая позволяет визуализировать собранные данные. К концу этой главы вы должны быть в состоянии создать что-то вроде этого:

Рисунок 4.1. Удовольствие от садоводства

На показанном фото датчик влажности почвы в стандарте STEMMA от Adafruit Industries установлен рядом с небольшим растением в горшке. Датчик влажности почвы подключен к Raspberry Pi Pico. Pico монтируется на двойной расширительный блок Pico Omnibus (см. рис. 2.31). Для загрузки значений влажности почвы в ThingSpeak слева от Pico подключен беспроводной модуль Pico.

В этой главе мы обсудим следующие темы:

- почему садоводство?;
- установка необходимых библиотек;
- настройка датчика влажности почвы;
- настройка беспроводного модуля;
- настройка светодиода NeoPixel LED;
- публикация данных в ThingSpeak;
- собрать все это воедино.

Давайте начнем!

Технические требования

Для этой главы рекомендуется использовать следующее оборудование:

- Raspberry Pi Pico (**https://bit.ly/3AJtoAf**);
- емкостный датчик влажности почвы Adafruit STEMMA (с I2C-выходом) (**https://learn.adafruit.com/adafruit-stemma-soil-sensor-i2c-capacitive-moisture-sensor**)[33];
- двойной расширитель Pico Omnibus (**https://bit.ly/3sr2GJR**);
- беспроводной модуль Pico (**https://bit.ly/3yPuoT9**);
- I2C-кабель STEMMA (**https://bit.ly/3shASra**);
- светодиод NeoPixel LED (необязательно) (**https://bit.ly/3ALSazI**)[34].

Вам также понадобится растение в горшке по вашему выбору. Примеры кода для этой главы доступны по следующей ссылке: **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIY-Workshop/tree/main/chapter_04**.

Видеоролики с кодом в действии для этой главы можно посмотреть по адресу **https://bit.ly/3w3CqY9**.

> **Установка CircuitPython**
>
> Мы предполагаем, что вы установили CircuitPython на свой Pico. Если вы незнакомы с процессом установки, мы рекомендуем следовать инструкциям из главы 1 «*Начало работы с Raspberry Pi Pico*».

Почему садоводство?

Почему мы выбрали садоводство для нашего проекта в этой главе?

---

[33] Емкостные датчики влажности почвы в России доступны в ряде модификаций (например, **https://www.chipdip.ru/product/sensor-soil-moisture-capacitive**), однако выходной сигнал у них простой аналоговый и с указанной в тексте I2C-библиотекой подключить их не получится. Так что заинтересованному читателю придется либо каким-то образом приобретать указанный по ссылке датчик Adafruit, либо выстраивать свою схему подключения (что, впрочем, при использовании встроенного АЦП Pico никаких трудностей представлять не должно). – *Прим. перев.*

[34] В России аналогичный модуль одиночного трехцветного светодиода WS2812 с контроллером можно приобрести, например, в интернет-магазине **https://tixer.ru**. – *Прим. перев.*

Садоводство – одно из самых старых увлечений. У него бывают периоды подъемов, как это было во время пандемии. Садоводство может быть воспринято на многих уровнях, технологии могут улучшить его, уменьшить страх неудачи и служить образовательным инструментом. Датчики и исполнительные устройства для полива или освещения могут помочь с растениями, требующими дополнительного ухода, как некоторые комнатные или водные растения, орхидеи и деревья бонсай.

Помимо того что садоводство является обычным хобби, его также можно использовать для эффективного выращивания собственных продуктов питания в городских условиях. Поскольку все больше людей живет в городах, для них становится важным иметь возможность выращивать хотя бы часть собственных продуктов питания. Домашняя еда оздоровляет питание и образ жизни. Занятие садоводством ведет к снижению стресса и улучшению психического здоровья. Садоводство и городское сельское хозяйство в любом масштабе также могут помочь находящимся под угрозой исчезновения пчелам, бабочкам и другим видам.

Растениям требуются почва, вода, свет определенной длины волны в течение определенного периода времени и удобрения. Датчики можно использовать для измерения количества света, получаемого растением в определенном месте, влагосодержания, водородного показателя почвы (pH) и аналогичных параметров.

Датчики также могут использоваться для измерения температуры окружающей среды, содержания углекислого газа и летучих органических соединений, которые могут оказывать влияние на рост растений. Как только выходные данные датчиков будут собраны и интерпретированы с помощью таких инструментов, как Raspberry Pi Pico, растения могут быть обеспечены необходимыми ресурсами – водой, удобрениями, достаточным освещением или затенением и т. д.

В более широком масштабе Raspberry Pi Pico можно использовать для любых сельскохозяйственных технологий, включая гидропонику и вертикальное земледелие, в том числе для универсального земледелия в развивающихся странах и экономически неблагополучных сообществах. Сеть плат Pico может служить в качестве периферийных устройств, обнаруживая проблемы и обеспечивая питание или другую поддержку для выживания и роста растений. Аналогичным образом эта работа может быть распространена на городские сады или лесное хозяйство.

Установка необходимых библиотек

В этом разделе мы установим библиотеки, необходимые для подключения датчика влажности почвы, светодиода NeoPixel и беспроводного модуля к Raspberry Pi Pico. Все библиотеки являются частью пакета Adafruit CircuitPython. Последний пакет можно загрузить в виде ZIP-файла с **https://circuitpython.org/libraries**. Мы использовали версию пакета, предназначенную для CircuitPython 6.x.x.

После загрузки ZIP-файла извлеките его содержимое, чтобы мы могли скопировать библиотеки, необходимые нам для проекта.

Датчик содержания влаги в почве

Мы используем емкостный датчик влажности почвы, основанный на платформе **Adafruit seesaw**. Внешний вид датчика показан на следующем рисунке:

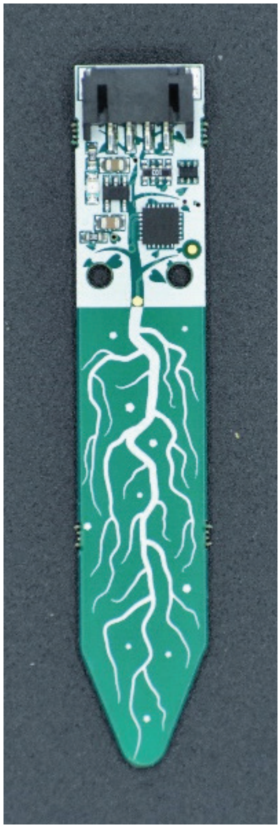


Рисунок 4.2. Емкостный датчик влажности почвы от Adafruit

Adafruit seesaw – это платформа, которая позволяет взаимодействовать через интерфейс I2C с датчиками любого типа, управлять двигателем, контролировать выход с широтно-импульсной модуляцией (ШИМ) и т. д. Это особенно полезно в сценариях, когда ваш микроконтроллер не поставляется с требуемым периферийным устройством. В этом проекте емкостный датчик влажности почвы соединен с микроконтроллером, работающим под управлением встроенного программного обеспечения *seesaw*. Мы можем считывать значения относительной влажности и содержания влаги в почвы, используя библиотеки *adafruit_seesaw*. Вы можете узнать больше о фреймворке *seesaw* здесь: **https://bit.ly/3D0qN7c**.

Для подключения датчика влажности почвы необходимы две библиотеки: *adafruit_seesaw* и *adafruit_bus_device*. В извлеченном пакете библиотеки имеются папки *adafruit_seesaw* и *adafruit_bus_device*. Скопируйте эти папки в папку *lib* вашего Pico.

Беспроводной модуль

Нам нужна библиотека *adafruit_esp32spi* для беспроводного модуля. Скопируйте папку (с тем же именем) в папку *lib* вашего Pico. Нам также понадобится двоичный файл *adafruit_reqests.mpy* из пакета библиотеки.

NeoPixel LED

Нам нужен двоичный файл *neopixel.mpy* для управления светодиодом NeoPixel (показан на следующем рисунке). Скопируйте двоичный файл в папку *lib*.

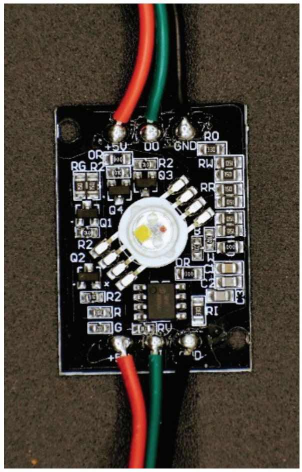


Рисунок 4.3. Красный, зеленый, синий, белый (RGBW) светодиод NeoPixel LED

В результате содержимое папки *lib* Pico должно выглядеть примерно так:

Name	Date modified	Type	Size
adafruit_bus_device	8/8/2021 6:58 PM	File folder	
adafruit_esp32spi	8/8/2021 6:58 PM	File folder	
adafruit_seesaw	8/8/2021 5:33 AM	File folder	
adafruit_requests.mpy	8/6/2021 5:10 AM	MPY File	14 KB
neopixel.mpy	8/6/2021 5:09 AM	MPY File	3 KB

Рисунок 4.4. Библиотеки, необходимые для текущего проекта

Далее мы настроим отдельные компоненты для проекта и протестируем их. Начнем с датчика влажности почвы.

Настройка датчика влажности почвы

В этом разделе мы будем подключать и тестировать датчик влажности почвы к Pico. Здесь показана схема подключения датчика:

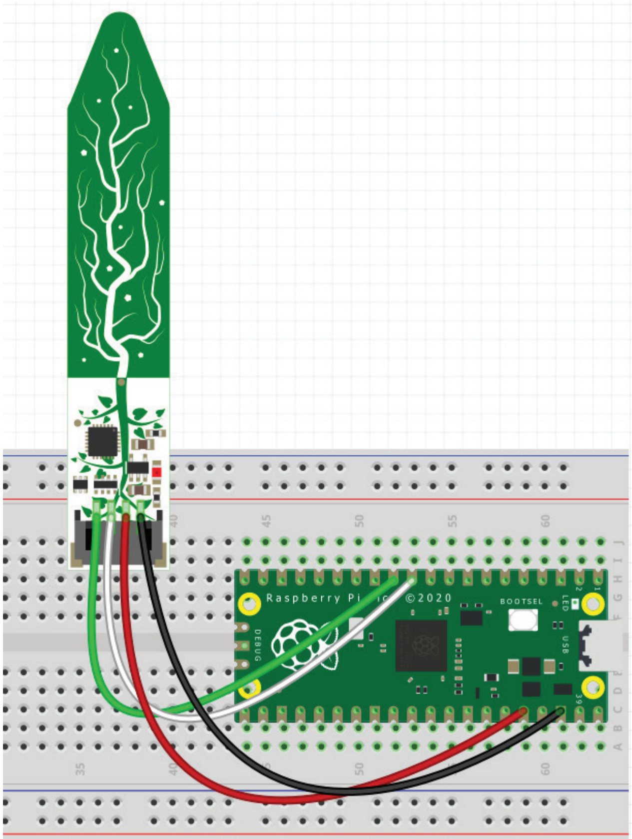


Рисунок 4.5. Схема подключения датчика влажности почвы к Pico

> **Разводка выводов Pico**
>
> Схема разводки выводов для Pico представлена в главе 1 «*Начало работы с Raspberry Pi Pico*».

Датчик влажности почвы поставляется с интерфейсом I2C, соответственно, разъем датчика системы STEMMA состоит из четырех контактов, а именно SCL, SDA, VIN и GND. Если вы незнакомы с интерфейсом I2C, мы рекомендуем прочитать главу 2 этой книги «*Последовательные интерфейсы и их приложения*». Согласно схеме рис. 4.5, контакты датчика влажности почвы подключены к Pico следующим образом:

- SCL к выводу GP9 Pico;
- SDA к выводу GP8 Pico;
- VIN к контакту 3,3 В Pico;
- выводы GND датчика и Pico соединены вместе.

> **Подтягивающие резисторы для шины I2C**
>
> Как правило, для выводов SCL и SDA шины I2C необходимы подтягивающие резисторы. Датчик влажности почвы системы STEMMA выпускается с подтягивающими резисторами на шине I2C. При использовании готовых датчиков, которые поставляются с интерфейсом I2C, ознакомьтесь со схемой и конфигурацией шины I2C датчика. Если вы незнакомы с необходимостью использования подтягивающих резисторов для шины I2C, мы рекомендуем ознакомиться с главой 2 «*Последовательные интерфейсы и их приложения*».

Теперь, когда мы подключили датчик влажности почвы, давайте напишем код для его тестирования.

> **Примеры кода**
>
> Примеры кода, используемые для тестирования отдельных компонентов далее, основаны на образцах из обучающей системы *Adafruit Learning System*, которая является отличным ресурсом для любого аппаратного проекта.

Откройте *code.py* (находится на нашем Pico) и внесите следующие изменения.

1. Первым шагом является импорт необходимых модулей следующим образом:

```
import time
import board
import busio
from adafruit_seesaw.seesaw import Seesaw
```

2. За этим следует инициализация шины I2C путем указания выводов SCL (GP9) и SDA (GP8). Затем мы создаем экземпляр объекта, принадлежащего классу `Seesaw class` для связи с датчиком почвы:

```
i2c = busio.I2C(board.GP9, board.GP8)
ss = Seesaw(i2c, addr=0x36)
```

3. Теперь мы можем прочитать уровни влажности и температуры почвы следующим образом:

```
touch = ss.moisture_read()
temp = ss.get_temp()
```

4. Мы можем считывать уровни влажности и температуры в бесконечном цикле и выводить их через последовательный порт. Собрав все это вместе, мы имеем следующее:

```
import time
import board
import busio
from adafruit_seesaw.seesaw import Seesaw

i2c = busio.I2C(board.GP9, board.GP8)
ss = Seesaw(i2c, addr=0x36)

while True:
  # read moisture level through capacitive touch pad
  touch = ss.moisture_read()

  # read temperature from the temperature sensor
  temp = ss.get_temp()

  print("temp: " + str(temp) + " moisture: " + str(touch))
  time.sleep(1)
```

Сохранив файл кода как *code.py*, вы должны увидеть выходные данные датчика влажности почвы, как показано на рис. 4.6. Поскольку это емкостный датчик, прикосновение к датчику может привести к изменению измеряемой емкости. Вы можете наблюдать изменение показаний при прикосновении к датчику:

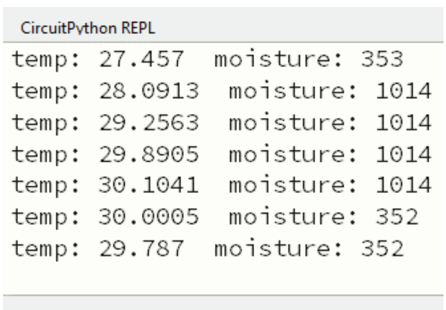


Рисунок 4.6. Выходной сигнал датчика влажности почвы

Уровень влажности почвы колеблется от 200 (очень сухой) до 2000 (очень влажный)[35]. Уровни зависят от нескольких факторов, включая тип почвы, температуру и т. д.

В следующем разделе мы собираемся настроить беспроводной модуль.

[35] Следует учитывать, что используемый здесь почвенный датчик влажности Adafruit STEMMA Soil Sensor показывает не привычную по метеорологическим датчикам относительную влажность (humidity), а влагосодержание (moisture) в условных единицах. В лабораторных условиях влагосодержание обычно рассчитывается в массовых долях воды относительно количества *твердого вещества* (а не общей массы образца), поэтому может колебаться от нуля до многих сотен процентов. Показания электронных датчиков влагосодержания зависят от многих дополнительных величин (плотности и пористости почвы, ее состава, содержания солей и т. д.), поэтому чаще всего выражаются в неких условных единицах, как и в данном случае. – *Прим. перев.*

Настройка беспроводного модуля

В этом разделе мы обсудим настройку и тестирование беспроводного модуля ESP32. Для удобства будем использовать двойной расширитель **Pico Omnibus** от Pimoroni. Pico и беспроводной модуль монтируются на расширитель, как показано на рисунке:

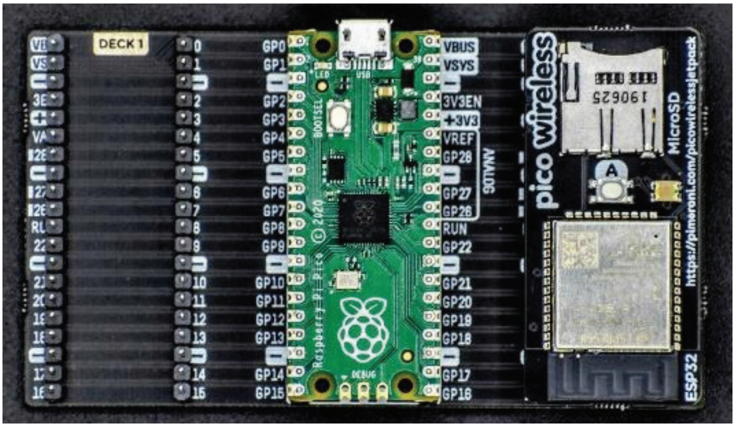


Рисунок 4.7. Pico и беспроводной модуль, установленные на Pico Omnibus

> **Соединения на макетной плате**
>
> Если у вас нет расширителя, для связи с ESP32 используются следующие контакты: CS → GP7; Ready → GP10; Reset → GP11; Clock → GP18; *Master Out Slave In (MOSI)* → GP19; *Master In Slave Out (MISO)* → GP16.
>
> Схема соединений показана на рис. 4.8.

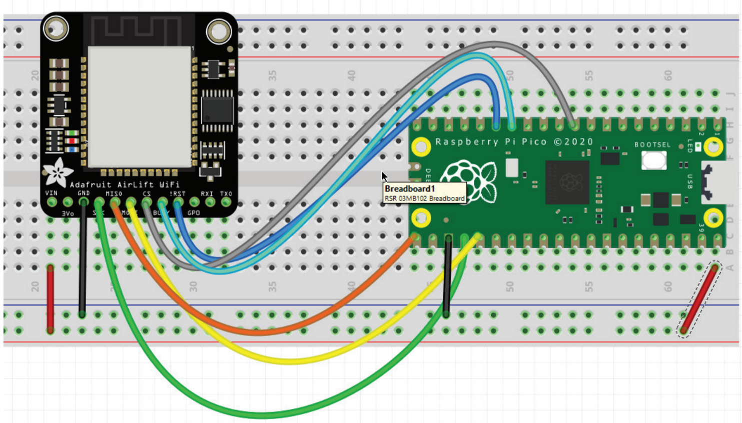


Рисунок 4.8. Подключение беспроводного модуля ESP32 к Raspberry Pi Pico

Чтобы подключиться к сети Wi-Fi с помощью пакета *wireless pack*, создайте файл с именем *secrets.py* на вашем Pico и сохраните в нем следующую информацию:

```
secrets = {
   'ssid' : 'home ssid',
   'password' : 'my password',
   'timezone' : "America/New_York",
}
```

Убедитесь, что вы верно указали свои учетные данные Wi-Fi. Теперь можно проверить, сможем ли мы подключиться к сети Wi-Fi и получать информацию с веб-сайта. Мы внесли изменения в код *Adafruit Learning System* для тестирования беспроводного пакета. Пример кода доступен при загрузке главы как *code_wifi.py* (см. **https://bit.ly/3v4Co1j**).

```
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)
spi = busio.SPI(board.GP18, board.GP19, board.GP16)
```

Сохраните файл как *code.py* и запустите его, вы должны увидеть следующий вывод:

Рисунок 4.9. Тест Wi-Fi

Теперь, когда мы протестировали беспроводной пакет, мы в следующем разделе перейдем к взаимодействию с NeoPixel.

Настройка светодиода NeoPixel

В этом разделе мы рассмотрим интерфейс и протестируем светодиод NeoPixel. Мы добавляем светодиод в проект, потому что его можно использовать в качестве наглядной визуализации того, что ваша почва слишком сухая. Если вы еще незнакомы с индивидуально адресуемыми RGB-светодиодами NeoPixel, то более подробную информацию вы можете найти здесь: **https://bit.ly/3sin0x5**.

NeoPixel LED подключен к выводу GP0 Pico, как показано на рис. 4.7. Соединения между Pico и NeoPixel LED следующие:

- GP0 → вывод DIN (DI) NeoPixe LED;
- VBUS → вывод +;
- выводы GND соединены между собой.

Здесь показана схема соединений на макетной плате:

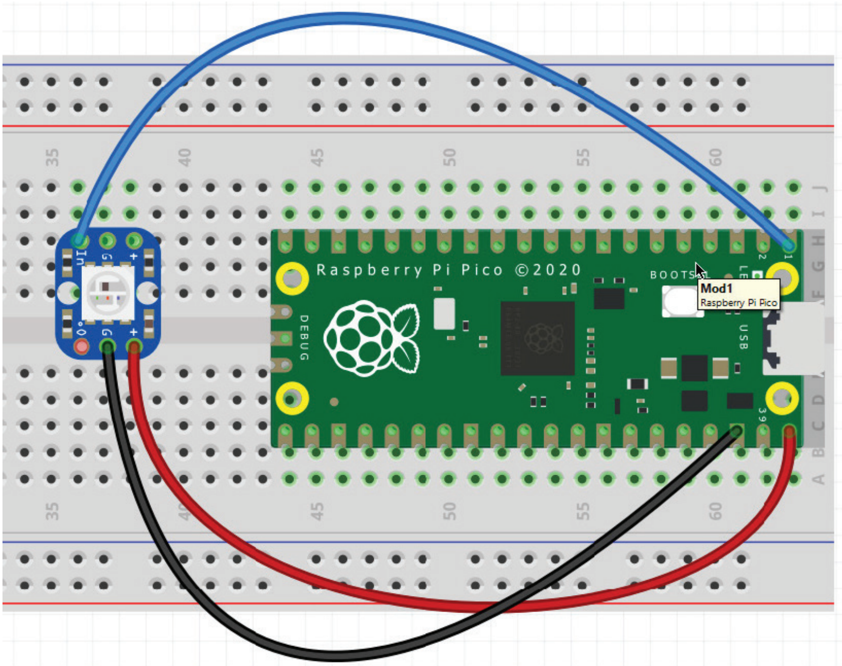


Рисунок 4.10. Подключение NeoPixel LED к Pico

Сейчас давайте проверим, все ли в рабочем состоянии. Первым шагом является импорт необходимых модулей следующим образом:

```
import time
import board
import neopixel
```

Теперь инициализируем объект, принадлежащий классу Neopixel, выполнив следующий код:

```
pixel_pin = board.GP0
num_pixels = 1

ORDER = neopixel.RGBW
pixels = neopixel.NeoPixel
(pixel_pin, num_pixels, brightness=0.2, auto_write=False, pixel_order=ORDER)
```

Мы можем создать эффект мигания нашего NeoPixel LED следующим образом:

```
while True:
   pixels.fill((0, 0, 255, 0))
   pixels.show()
   time.sleep(0.250)
   pixels.fill((0, 0, 0, 0))
   pixels.show()
   time.sleep(0.250)
```

Собрав все это вместе, мы имеем вот что:

```
import time
import board
import neopixel

pixel_pin = board.GP0
num_pixels = 1

ORDER = neopixel.RGBW
pixels = neopixel.NeoPixel
(pixel_pin, num_pixels, brightness=0.2, auto_write=False, pixel_order=ORDER )

while True:
   pixels.fill((0, 0, 255, 0))
   pixels.show()
   time.sleep(0.250)
   pixels.fill((0, 0, 0, 0))
   pixels.show()
   time.sleep(0.250)
```

На этом этапе макет вашего проекта должен выглядеть примерно так:

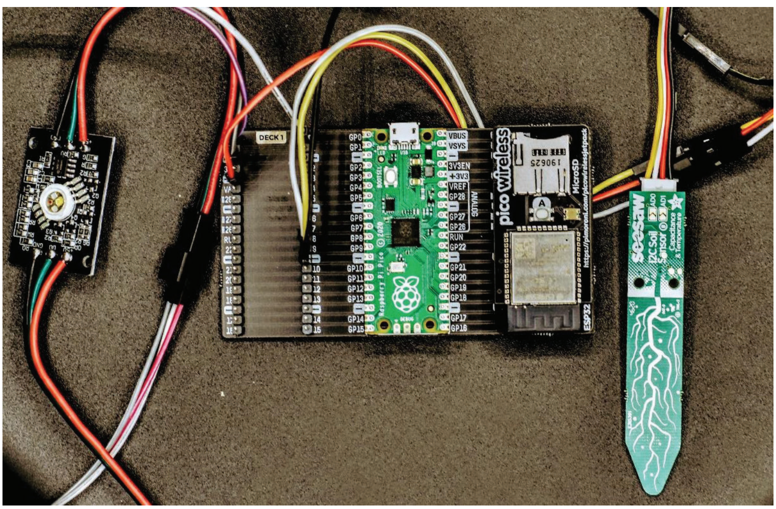


Рисунок 4.11. Макет проекта датчика влажности почвы

В следующем разделе мы рассмотрим размещение данных об уровне влажности почвы на платформе ThingSpeak и сведем наш проект воедино.

Размещение данных в ThingSpeak

В этом разделе мы обсудим размещение наших данных в ThingSpeak. Это программное обеспечение с открытым исходным кодом, которое позволяет загружать данные с устройств, подключенных к интернету, как, например, в нашем текущем проекте. Программа позволяет собирать данные с датчиков и затем анализировать их. Мы выбрали ThingSpeak в качестве примера из различных наборов инструментов, доступных разработчикам. Создайте бесплатную учетную запись на **https://thingspeak.com** и новый канал, как показано на следующем рисунке:

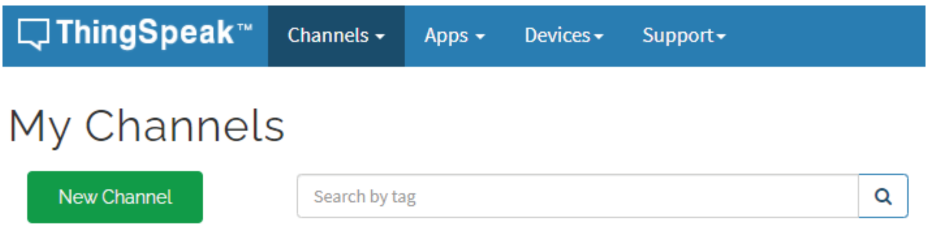


Рисунок 4.12. Создание нового канала

Дайте название своему каналу, заполните поле 1 (как показано на рис. 4.13) и сохраните канал:

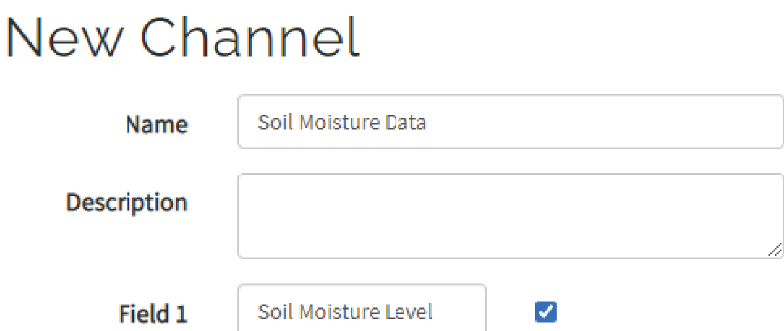


Рисунок 4.13. Присвоение имени каналу

Теперь канал создан. Запишите значение ключа канала (**Write API Key**, вкладка **API Key**), как показано на следующем рисунке:

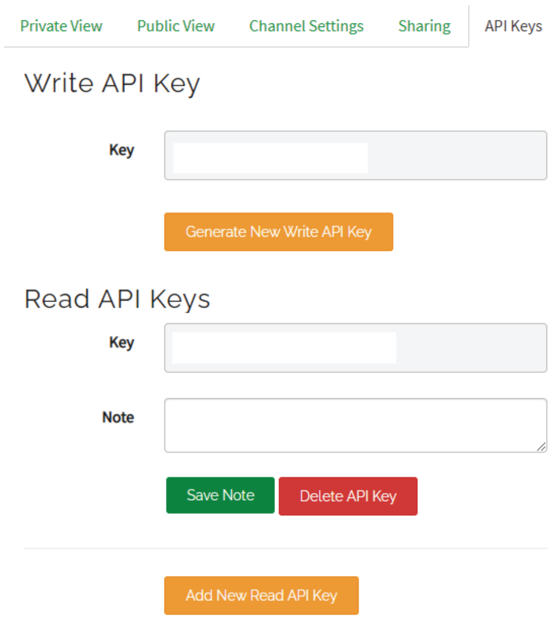


Рисунок 4.14. Запись значения ключа канала

Теперь давайте изменим *secrets.py*. Чтобы добавить ключ для ThingSpeak, внесите следующие исправления:

```
secrets = {
   'ssid' : 'my_network',
   'password' : 'password',
   'timezone' : "America/New_York",
   'thingspeak_token' : 'ABCD1234',
}
```

Окончательный пример кода доступен для загрузки вместе с этой главой в файле *code_thingspeak.py* (см. **https://bit.ly/3avHvy0**). Давайте обсудим наиболее важные элементы кода.

- Начнем с импорта модулей, необходимых для проекта, как показано в следующем фрагменте кода:

```
import time
import board
import busio
import neopixel
from adafruit_seesaw.seesaw import Seesaw
from digitalio import DigitalInOut
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
```

- Импортируем учетные данные, необходимые для проекта, из *secrets.py* следующим образом:

```
try:
  from secrets import secrets
except ImportError:
  print("WiFi secrets are kept in secrets.py, please add them there !")
raise
```

- Объявляем URL-адрес, который используется для публикации данных в ThingSpeak. URL-адрес доступен на вкладке *API Keys* вашей учетной записи ThingSpeak. Код проиллюстрирован здесь:

```
URL = "https ://api.thingspeak.com/update?api_key={token}&field1={value}"
```

- Теперь мы инициализируем датчик грунта, подключенный через интерфейс I2C:

```
i2c = busio.I2C(board.GP9, board.GP8)
ss = Seesaw(i2c, addr=0x36)
```

- Затем инициализируем NeoPixel, который подключен к выводу GP0:

```
pixel_pin = board.GP0
num_pixels = 1
ORDER = neopixel.RGBW
pixels = neopixel.NeoPixel
(pixel_pin, num_pixels, brightness=0.2, auto_write=False, pixel_order=ORDER )
```

- Потом инициализируем модуль беспроводной связи ESP32. Ранее в этой главе мы обсуждали контакты, используемые для подключения беспроводного модуля. Вот соответствующий код:

```
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)
spi = busio.SPI(board.GP18, board.GP19, board.GP16)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
```

- Теперь подключаем модуль к беспроводной сети следующим образом:

```
requests.set_socket(socket, esp)
if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
  print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for i in esp.MAC_address])

print("Connecting to AP...")
while not esp.is_connected:
  try:
    esp.connect_AP(secrets["ssid"], secrets["password"])
```

```
  except RuntimeError as e:
    print("could not connect to AP, retrying: ", e)
    continue

print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
```

- Входим в основной цикл программы, где считываем уровень влажности (*moisture*) каждые 2 секунды:

```
while True:
  # read moisture level
  touch = ss.moisture_read()
  # read temperature from the temperature sensor
  temp = ss.get_temp()
  print("temp: " + str(temp) + " moisture: " + str(touch))
```

Если уровень влажности почвы ниже определенного порога, мы создаем световой эффект с помощью NeoPixel LED. Это должно напомнить, что пришло время поливать растение. В следующем фрагменте кода пороговое значение было установлено равным 500, но оно может варьироваться в зависимости от типа почвы, используемой для вашего растения. Вам нужно настроить этот порог в зависимости от того, что вы измеряете, вставив датчик почвы в растение в горшке. Необходимо также измерить уровень влажности почвы, чтобы установить необходимые пороговые значения:

```
if touch < 500:
  pixels.fill((0, 0, 255, 0))
  pixels.show()
  time.sleep(0.250)
  pixels.fill((0, 0, 0, 0))
  pixels.show()
  time.sleep(0.250)
```

- Данные отправляются каждый раз, когда уровень влажности почвы падает ниже порогового значения:

```
try:
  r = requests.get(URL.format(token=secrets["thingspeak_token"], value =str(touch)))
except:
  print("Failed to publish value")
else:
  print("-" * 40)
  print(r.json())
  print("-" * 40)
  r.close()
```

Данные отправляются с интервалом в 2 секунды – такой короткий интервал поможет определить продолжительность времени, в течение которого уровень влажности почвы был ниже желаемого уровня. После того как мы сохранили код в Pico и убедились, что данные успешно размещены в ThingSpeak, можно собрать все воедино.

Собираем все вместе

После того как вы отрегулировали пороговые значения, вставьте датчик почвы рядом с вашим растением в горшке и включите Pico. У вас должно быть что-то, похожее на это:

Рисунок 4.15. Удовольствие от садоводства

Вы также можете отслеживать свои опубликованные данные на ThingSpeak, как показано на следующем рисунке:

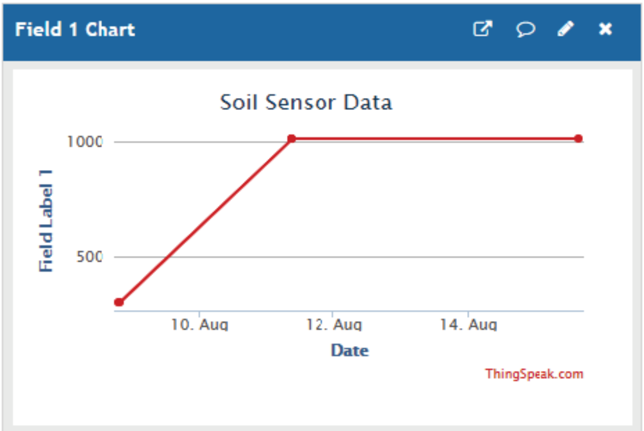


Рисунок 4.16. Данные датчика влажности почвы

ThingSpeak – это аналитическая платформа, которая позволяет вам собирать данные с ваших датчиков. Вы можете расширить данный проект, создав панель мониторинга для всех датчиков в вашем саду и удаленно управляя клапанами полива с панели мониторинга. Такие платформы, как ThingSpeak, позволяют масштабировать от одного датчика до множества датчиков и исполнительных механизмов.

Итоги

В этой главе мы обсуждали подключение датчика почвы к Pico и мониторинг уровня влажности почвы в горшках с растениями. Мы также обсудили размещение уровня влажности на платформе ThingSpeak и включение светодиода, когда влажность падает ниже определенного порога. Этот проект демонстрирует использование Pico в садоводстве. Вы могли бы продолжить данный проект, добавив автоматический оросительный клапан, когда уровень влажности опускается ниже определенного порога. Вы также можете изменить поведение NeoPixel на основе измеренных уровней влажности – например, вы можете заставить светодиод NeoPixel светиться янтарным цветом, когда ваше растение нуждается в поливе[36].

В следующей главе мы обсудим создание метеостанции с использованием Pico!

[36] Для свечения разными цветами достаточно поменять значения RGB в функции `pixels.fill`. Например, янтарному свечению должны соответствовать примерные значения `pixels.fill((255, 192, 0, 0))`. – *Прим. перев.*

Часть II

Обучение через созидание

Эта часть посвящена получению опыта разработки путем создания трех различных проектов различной сложности с использованием Raspberry Pi Pico. Мы начнем с метеостанции и закончим настенным дисплеем для отслеживания качества воздуха.

Часть II содержит следующие главы:

- глава 5 «*Строим метеостанцию*»;
- глава 6 «*Проектируем настенный семисегментный дисплей*»;
- глава 7 «*Разрабатываем устройство слежения за качеством воздуха*».

Глава 5

Строим метеостанцию

В этой главе вы построите метеостанцию для проведения собственных научных экспериментов с использованием Raspberry Pi Pico. Мы обсудим подключение и тестирование различных датчиков с Pico, а также размещение Pico на открытом воздухе.

К концу данной главы вы построите нечто похожее на то, что показано на следующем рисунке:

Рисунок 5.1. Raspberry Pi Pico, подключенный к метеостанции

В этом проекте мы рассмотрим взаимодействие с Pico датчиков для сбора данных об окружающей среде. Мы протестируем все датчики, обсудив код, необходимый для их подключения, и соберем нашу метеостанцию воедино.

Рассмотрение в данной главе включает следующие темы:

- проведение гражданских научных экспериментов;
- установка необходимых библиотек;
- тестирование датчиков;
- сборка и тестирование метеостанции.

Защита проекта от атмосферных воздействий

В этой главе мы хотим продемонстрировать проведение гражданских научных экспериментов с использованием микроконтроллера RP2040. Мы не будем обсуждать источники питания или защиту вашей метеостанции от атмосферных воздействий, поскольку хотим сосредоточиться на взаимодействии датчиков и сборе данных. Защита от атмосферных воздействий и выбор источника питания требуют дополнительного тестирования. Этот проект находится в стадии разработки, и если он в существующем виде покажет себя достаточно работоспособным, мы опубликуем результаты в репозитории этой книги[37].

Технические требования

Для этой главы рекомендуется использовать следующее оборудование. Мы оставим выбор полного комплекта датчиков для метеостанции на ваше усмотрение:

- Raspberry Pi Pico (**https://bit.ly/3AJtoAf**);
- двойной расширитель Pico Omnibus (**https://bit.ly/3sr2GJR**);
- AM2315 – датчик температуры/влажности I2C в корпусе (**https://bit.ly/3FNFWd**[38]);
- две переходные платы RJ11 (**https://www.chipdip.ru/product1/8004744987**);
- УФ-датчик VEML6075 (**https://www.sparkfun.com/products/15089**);
- датчик давления BME280 (**https://bit.ly/3lPb3xh**[39]);
- датчик влажности почвы Adafruit STEMMA (см. предыдущую главу);
- дополнительно: комплект измерителя погоды[40] (**https://bit.ly/3johPZC**).

Примеры кода, рассмотренные в этой главе, доступны здесь: **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIYWorkshop/tree/main/chapter_05**.

Видеоролики с кодом в действии для этой главы можно посмотреть по адресу: **https://bit.ly/3P1uTSz**.

[37] Следует твердо усвоить, что представленные в этой книге электронные схемы, собранные на беспаечной макетной плате или с помощью конструкций, подобных Pico Omnibus, – это лишь макеты реальных приборов (авторы не зря часто употребляют слова «макетирование» и «прототипирование»). Для доводки изделия «до ума» требуется еще очень многое, независимо от того, должно оно работать на открытом воздухе или в комнатных условиях. Причем именно этот этап «доводки до ума» обычно оказывается самым затратным по деньгам и по времени: стоимость готового изделия может в разы и десятки раз превышать стоимость комплектующих, а сам процесс требует нетривиальных знаний и умений. Подобные книги и пособия не зря обычно останавливаются на этапе макетирования, так как конструирование электронных приборов – отдельная и весьма обширная тема. В некоторой (очень небольшой) степени авторы касаются этой темы в конце главы. – *Прим. перев.*

[38] В российских условиях приобрести датчик AM2315 проще всего на AliExress (см., например, **https://sellercheck.ru/m/am2315**). – *Прим. перев.*

[39] BME-280 доступен в российских интернет-магазинах в нескольких вариантах (см., например, **https://www.chipdip.ru/product/bme280-environmental-sensor**), отличающихся от указанного авторами датчика Adafruit только разводкой выводов. – *Прим. перев.*

[40] «Измерителем погоды» (weather meter) компания SparkFun назвала комплект из датчика осадков и датчиков параметров ветра (измерителя скорости – анемометра и указателя направления – флюгера). – *Прим. перев.*

Проведение гражданских научных экспериментов

Гражданская наука – это широкое понятие, которое включает в себя множество отдельных лиц и коллективов, добровольно занимающихся сбором и/или анализом данных по различным проектам, представляющим общественный интерес. Участники этих проектов могут быть любого пола, возраста и образования, и они могут находиться на разных континентах. Области, в которых гражданская наука может быть эффективной, включают мониторинг погоды и климата, отслеживание дикой природы, устойчивое развитие, глобальное потепление, здоровье населения и умные города, а также общенаучные вопросы. Ожидается, что характер, масштабы и количество гражданских научных начинаний в ближайшие годы значительно возрастут. Типы данных, собираемых гражданской наукой, могут включать дискретные или непрерывные параметры, такие как температура, содержание летучих органических соединений и другой химии в атмосфере, изображения и аудиофайлы.

В целом гражданские научные проекты могут быть эффективными в решении крупномасштабных проблем, требующих больших объемов данных, полученных недорогим способом. Когда для гражданских научных проектов нужно оборудование, расширение исследований может стать непомерно дорогостоящим, если не позаботиться о минимизации удельных затрат. Если позаботиться о выборе недорогих компонентов, таких как Raspberry Pi Pico, масштаб проектов может вырасти и охватить более широкий круг участников. Метеостанция, продемонстрированная в этой главе, может служить примером того, как создавать недорогие решения для получения данных в рамках гражданской науки.

Установка необходимых библиотек

Мы собираемся установить все библиотеки, необходимые для этой главы. Первым шагом является загрузка пакета CircuitPython Library. Вы можете загрузить пакет библиотеки в виде ZIP-файла отсюда: **https://circuitpython.org/libraries**. Извлеките содержимое ZIP-файла, чтобы начать установку библиотеки.

> **Установка CircuitPython**
>
> Мы предполагаем, что вы установили CircuitPython на свой Pico. Если вы незнакомы с процессом установки, рекомендуем следовать инструкциям из главы 1 «*Начало работы с Raspberry Pi Pico*».

Датчик AM2315

Нам нужны две библиотеки для датчика температуры и влажности AM2315 (показан на рис. 5.2 далее), а именно *adafruit_bus_device* и *adafruit_ahtx0*. В извлеченном пакете есть папка с именем *adafruit_bus_device* вместе с файлом *adafruit_ahtx0.mpy*. Скопируйте ее в папку *lib* Pico.

Выбор библиотеки

Согласно руководству, представленному Adafruit, вы можете считывать температуру и влажность с датчика AM2315 с помощью библиотеки *adafruit_am2320* (**https://bit.ly/3psAUgE**). Однако при использовании библиотеки у нас возникли проблемы с инициализацией. Поскольку адрес I2C датчика был идентичен адресу AHT20, мы решили попробовать библиотеку *adafruit_ahtx0*, и мы смогли прочитать данные датчика[41].

Датчик BME280

Нам нужны две библиотеки для датчика BME280, а именно *adafruit_bus_device* и *adafruit_bme280*. Поскольку мы уже скопировали первую в предыдущем разделе, скопируйте вторую в папку *lib* Pico.

Датчик ультрафиолетового излучения VEML6075

Для датчика VEML6075 требуются две библиотеки, а именно *adafruit_bus_device* и *adafruit_veml6075*. Поскольку мы уже скопировали первую, нам нужно скопировать *adafruit_veml6075.mpy* в папку *lib* Pico.

Тестирование датчиков

В этом разделе мы настроим и протестируем отдельные компоненты, используемые в проекте, прежде чем собрать их вместе.

Тестирование датчика BME280

В этом разделе мы протестируем датчик BME280 (показан на рисунке ниже):

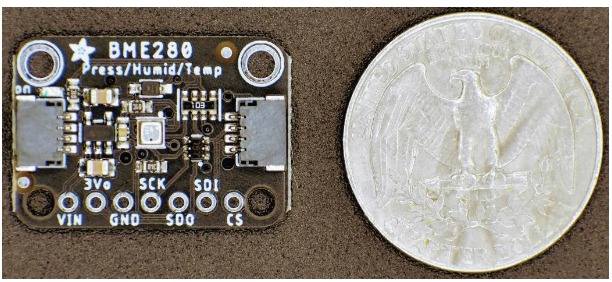


Рисунок 5.2. Датчик атмосферного давления BME280

[41] Согласно исходным текстам библиотек, указанных авторами, утверждение об идентичности адресов не соответствует действительности. Более того, у датчиков AM2315 (AM2320) и AHT20 существенно отличаются форматы представления данных (см. процедуры обработки считанных значений температуры и влажности в текстах библиотек и описаниях датчиков). Читателю предлагается самостоятельно разобраться с этим случаем, не принимая на веру рекомендации авторов (в частности, может пригодиться I2C-сканер, описанный на стр. 67). Заметим также, что на ***github.com*** для такого удобного датчика, как AM2315, имеется большое количество вариантов выбора Python-библиотек от разных источников. – *Прим. перев.*

Датчик давления BME280 можно использовать для измерения температуры, влажности и атмосферного давления[42]. Он поставляется с интерфейсом I2C, и его адрес на шине I2C равен 0x77.

Датчик соединяется с Pico, как показано на следующей схеме:

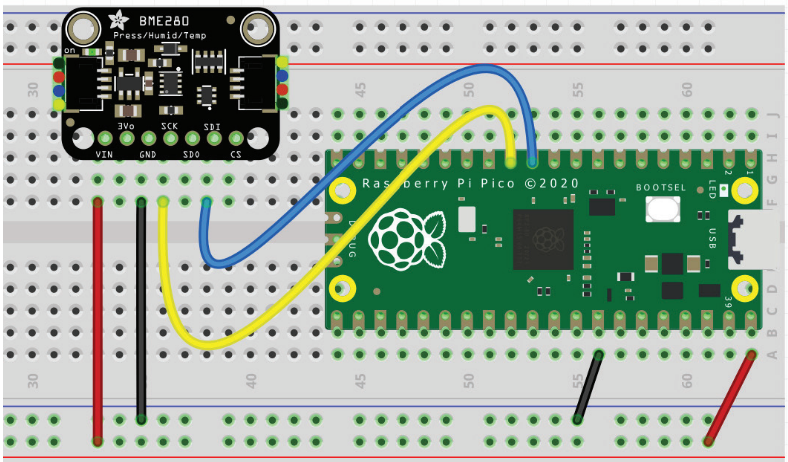


Рисунок 5.3. Схема соединений датчика BME280

Согласно схеме датчик соединен с Pico следующим образом (слева указаны контакты на плате BME280, стрелка указывает на контакты Raspberry Pi Pico):

- VIN → VBUS;
- SCL → GP9;
- SDA → GP8;
- GND → GND.

Подтягивающие резисторы для шины I2C

Как правило, для выводов SCL и SDA шины I2C необходимы подтягивающие резисторы. Используемая нами плата датчика BME280 поставляется уже с установленными подтягивающими резисторами на шине I2C. При использовании готовых датчиков, которые поставляются с интерфейсом I2C, ознакомьтесь со схемой и изучите конфигурацию шины I2C датчика. Если вы незнакомы с необходимостью использования подтягивающих резисторов для шины I2C, мы рекомендуем ознакомиться с главой 2 «*Последовательные интерфейсы и их приложения*».

[42] При использовании датчика BME280 следует учесть примечание, которое делает Bosh (производитель сенсоров BME280 и BMP280) по поводу измерения температуры: «*...значение температуры зависит от температуры печатной платы, самонагрева сенсорного элемента и окружающей температуры* (внутри корпуса – *Ю.Р.*) *и обычно превышает температуру окружающей среды*» (стр. 10 официального руководства Bosh по BME280). С основной функцией измерения атмосферного давления датчики BME280 и BMP280 справляются на отлично, а вот дополнительные величины требуют как минимум индивидуальной калибровки, причем после установки датчика в конкретную конструкцию. – *Прим. перев.*

Qwiic-разъемы

Датчик BME280 поставляется с парой разъемов Qwiic/STEMMA. Если у вас есть соединительный кабель Qwiic (ссылка: **https://www.sparkfun.com/products/17261**), вы можете подключить его, как показано на следующем рисунке. Если вы незнакомы с экосистемой Qwiic/STEMMA, мы рекомендуем прочитать главу 2 «*Последовательные интерфейсы и их приложения*». Разъемы Qwiic упрощают создание прототипов и макетов за счет последовательного соединения секционных плат:

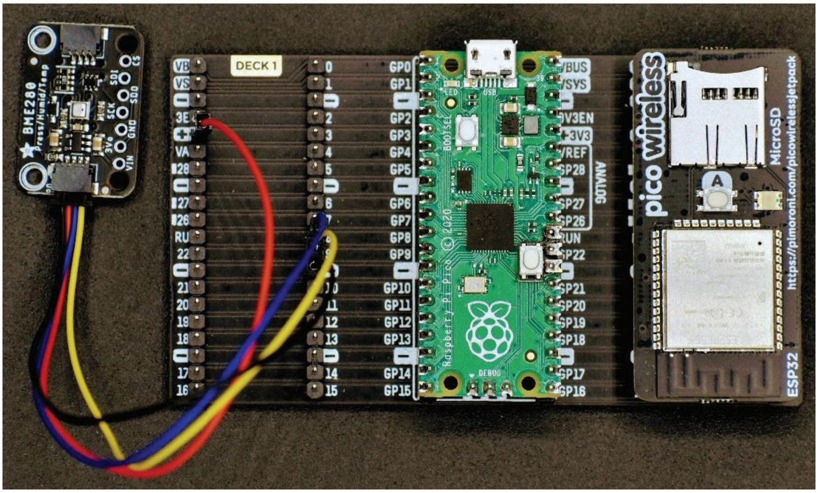


Рисунок 5.4. Датчик BME280, подключенный к Raspberry Pi Pico с помощью кабеля Qwiic

Давайте рассмотрим код, необходимый для взаимодействия с датчиком. Обсуждаемый пример кода доступен для загрузки вместе с этой главой в виде *code_bme280.py*.

1. Первым шагом к считыванию данных с датчика является импорт необходимых модулей. Библиотека *adafruit_bme280* поставляется с двумя подмодулями, а именно *basic* и *advanced*. В этом примере мы будем использовать базовый модуль. Для удобства чтения мы собираемся импортировать *adafruit_bme280.basic* как *adafruit_bme280*:

```
import board
import busio
import time
from adafruit_bme280 import basic as adafruit_bme280
```

2. Следующим шагом является инициализация шины I2C и передача объекта `I2C` в качестве аргумента классу `BME280`:

```
i2c = busio.I2C(board.GP9, board.GP8)
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)
```

3. Используйте местный прогноз погоды, чтобы установить базовую величину давления на уровне моря в вашей местности[43]:

```
bme280.sea_level_pressure = 1007.1
```

4. Теперь мы можем считывать температуру, давление и влажность:

```
while True:
  print("\nTemperature: %0.1f C" % bme280.temperature)
  print("Humidity: %0.1f %%" % bme280.relative_humidity)
  print("Pressure: %0.1f hPa" % bme280.pressure)
  print("Altitude = %0.2f meters" % bme280.altitude)
  time.sleep(2)
```

5. После выполнения кода вы должны увидеть выходные данные датчика следующим образом:

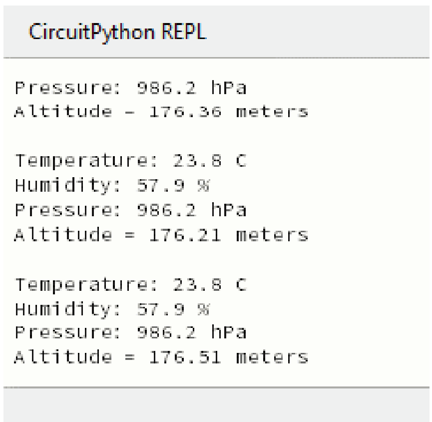


Рисунок 5.5. Выходной сигнал датчика BME280 в Mu IDE

Теперь, когда мы протестировали датчик BME280, обсудим тестирование датчика AM2315 в следующем разделе.

Тестирование датчика температуры/влажности AM2315

В этом разделе мы протестируем датчик температуры/влажности AM2315 (показан на следующем рисунке):

[43] Без знания величины «атмосферного давления, приведённого к уровню моря» (в авиации используется аббревиатура QNH) функция вычисления высоты места (altitude) не работает. Из бытового прогноза погоды вы эту величину, разумеется, не узнаете – только из специального авиационного, но ее можно приблизительно рассчитать, зная высоту некоей опорной точки на местности. Упрощенную методику расчетов можно найти здесь: **https://helpiks.org/3-25355.html**. Учтите, что каждый гПа отклонения в расчетах от действительной величины дает ошибку измерения альтитуды в 8 метров высоты. – *Прим. перев.*

Рисунок 5.6. Датчик AM2315

Поскольку датчик AM2315 поставляется с интерфейсом I2C, подключается он так же, как датчик BME280 (см. схему, показанную на рис. 5.5). Давайте кратко рассмотрим код, необходимый для взаимодействия с датчиком. Пример кода, обсуждаемый в этом разделе, доступен для загрузки вместе с этой главой в виде файла *code_ahtx0.py*.

1. Как всегда, тестирование датчика начнем с импорта необходимых модулей:

```
import time
import board
import busio
import adafruit_ahtx0
```

2. Мы инициализируем шину I2C и передаем объект классу `AHTx0`. Как обсуждалось ранее, мы используем библиотеку *adafruit_ahtx0*, поскольку используемая микросхема датчика температуры совместима с семейством *ahtx0*:

```
i2c = busio.I2C(board.GP9, board.GP8)
temp_sensor = adafruit_ahtx0.AHTx0(i2c)
```

3. Теперь пришло время прочитать значения температуры и влажности.

```
while True:
  print("\nTemperature: %0.1f C" % sensor.temperature)
  print("Humidity: %0.1f %%" % sensor.relative_humidity)
  time.sleep(2)
```

4. После сохранения этого примера кода в файле *code.py* мы получим возможность видеть значения температуры и влажности на выходе последовательного порта:

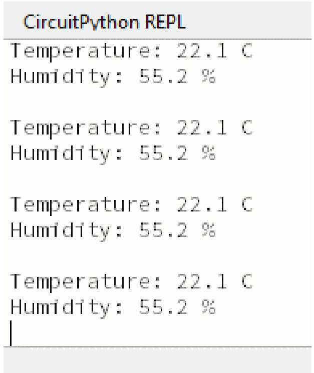


Рисунок 5.7. Значения температуры и влажности датчика AM2315

В следующем разделе мы обсудим тестирование УФ-датчика VEML6075.

Тестирование датчика VEML6075

В этом разделе мы рассчитаем УФ-индекс[44] с помощью датчика VEML6075 (показан на рис. 5.8). Библиотеки, которые мы установили ранее в этой главе, обеспечивают вывод УФ-индекса с использованием необработанных значений датчика.

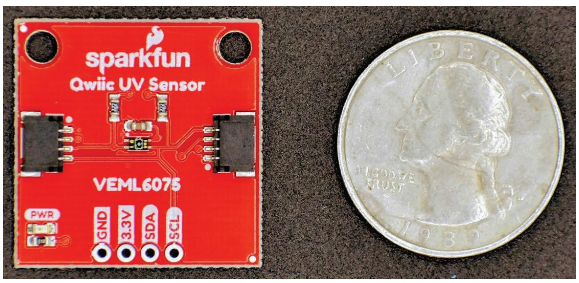


Рисунок 5.8. УФ-датчик VEML6075

Пример кода, обсуждаемый в этом разделе, доступен для загрузки вместе с этой главой в виде *code_veml6075.py*. Подключение датчика VEML6075 идентично подключению датчика BME280 (рис. 5.5).

Давайте рассмотрим код, необходимый для взаимодействия с датчиком.

1. Как всегда, тестирование датчика начнем с импорта необходимых модулей:

```
import time
import board
import busio
import adafruit_veml6075
```

[44] УФ-индекс – безразмерный показатель, характеризующий уровень ультрафиолетового излучения в спектре солнечного света. Подробнее об этом можно ознакомиться в статье Википедии «*УФ-индекс*». – *Прим. перев.*

2. Инициализируем шину I2C и передаем объект классу VEML6075:

```
i2c = busio.I2C(board.GP9, board.GP8)
veml = adafruit_veml6075.VEML6075(i2c, integration_time=100)
```

3. Теперь читаем значения УФ-индекса:

```
while True:
  print(veml.uv_index)
  time.sleep(1)
```

4. Вы можете проверить датчик с помощью ультрафиолетового фонарика или подвергнуть его воздействию солнечного света. На следующем снимке экрана вы можете увидеть скачок значений УФ-индекса, когда датчик подвергается воздействию ультрафиолетового фонарика:

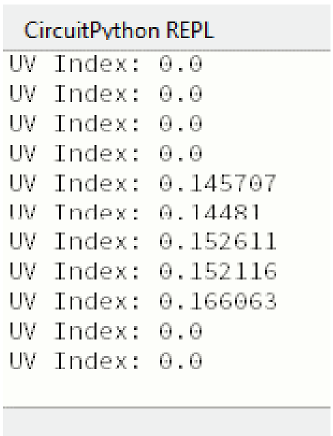


Рисунок 5.9. Скачок значений УФ-индекса при использовании УФ-фонарика

> **Осторожно!**
>
> Ультрафиолетовые фонарики – не игрушки; не светите ими прямо в глаза.

В следующем разделе мы протестируем датчики, входящие в состав измерителя погоды.

Тестирование датчиков измерителя погоды

В этом разделе мы рассмотрим тестирование датчиков, входящих в комплект измерителя погоды, а именно датчика осадков, анемометра и флюгера.

Выбор измерителя погоды

Измеритель погоды, который мы выбрали, довольно дорогой для проекта «Сделай сам». С помощью 3D-печати можно создать свой собственный анемометр, флюгер и т. д.[45]. Кроме того, вы можете создать настольную метеостанцию, которая использует общедоступные источники получения прогноза погоды.

Тестирование датчика осадков

Как следует из названия, датчик осадков (показан на следующем рисунке) используется для определения количества осадков. Датчик осадков представляет собой опрокидывающийся ковш с концевым выключателем, замыкающимся при наклоне. Согласно техническому описанию, переключатель замыкается на каждые 0,011 дюйма осадков (примерно 2,8 мм). Ковш опрокидывается и одновременно размыкает выключатель. Это позволяет рассчитать общее количество осадков за заданный промежуток времени.

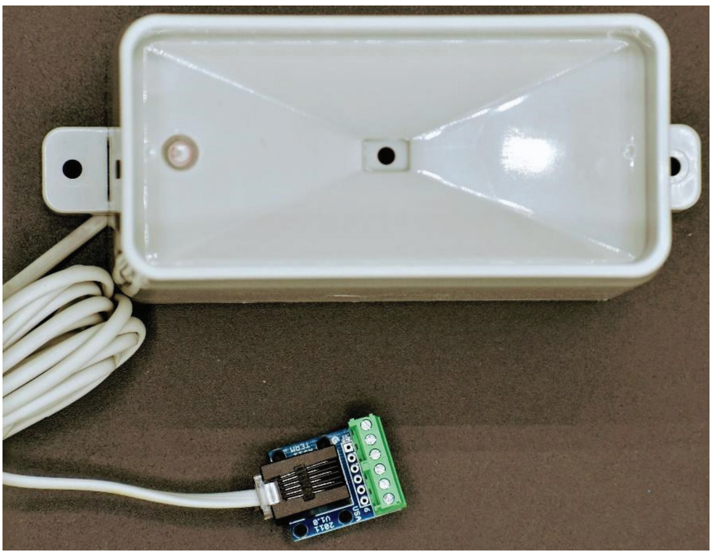


Рисунок 5.10. Датчик осадков, подключенный к переходной плате RJ11

Датчик осадков соединяется с Pico, как показано на следующем рисунке.

[45] Авторы ошибаются, причем дважды: во-первых, предлагаемый компанией SparkFun Weather Meter Kit за 80 долларов – очень невысокая цена для такого комплекта, даже если учесть издержки на доставку в Россию (профессиональные измерители такого рода зашкаливают по стоимости за многие тысячи долларов). Во-вторых, самостоятельное изготовление анемометра и флюгера с хорошими параметрами и приемлемой надежностью работы либо окажется неподъемной задачей для любительских условий, либо конечный результат многократно превысит по стоимости предлагаемый комплект. Обе эти особенности есть следствие одной и той же причины: наличия в конструкции датчиков большого количества весьма тонко выполненных механических частей. – *Прим. перев.*

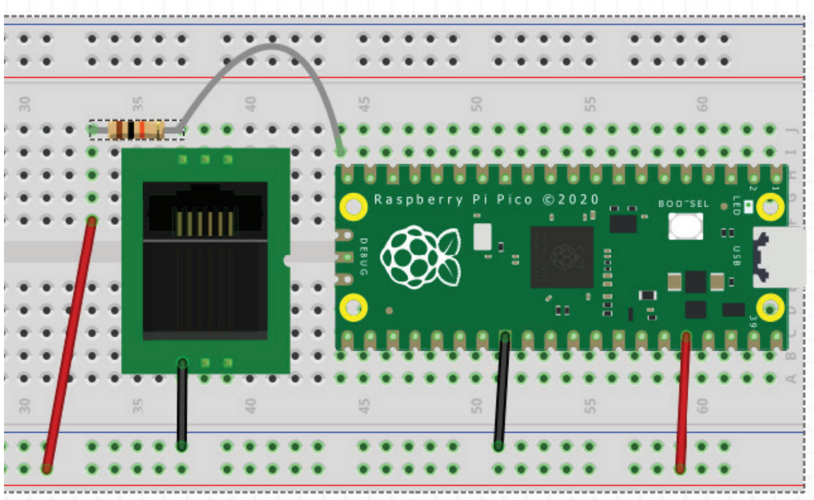


Рисунок 5.11. Схема подключения датчика осадков

Согласно техническому описанию, переключатель наклона датчика осадков подключен к третьему и четвертому контактам разъема RJ11. Один из выводов переключателя подключается к земле, другой – к выводу GPIO 15 и подтягивается к питанию с помощью резистора 10 КОм. Рассмотрим код, необходимый для подключения датчика осадков.

Мы будем использовать модуль *countio* для подсчета переходов состояний контактов датчика. Модуль *countio* подсчитывает количество падающих фронтов. Поскольку мы подключили вывод GPIO 15 и вывод переключателя к 3,3 В через резистор, падающий фронт обнаруживается каждый раз, когда переключатель замыкается. Пример кода, обсуждаемый в этой главе, доступен для загрузки вместе с данной главой в виде *code_rainfall.py*.

1. Начинаем с импорта всех необходимых модулей:

```
import time
import board
import countio
```

2. Инициализируем счетчик на выводе GP15 (вывод GPIO 15) и начинаем подсчет импульсов:

```
with countio.Counter(board.GP15) as pin_counter:
  while True:
    if pin_counter.count:
      print("Rainfall Detected")
      pin_counter.reset()
      time.sleep(2)
```

3. Сохраните этот код как *code.py* и введите в датчик пару капель воды. Вы должны увидеть следующий результат:

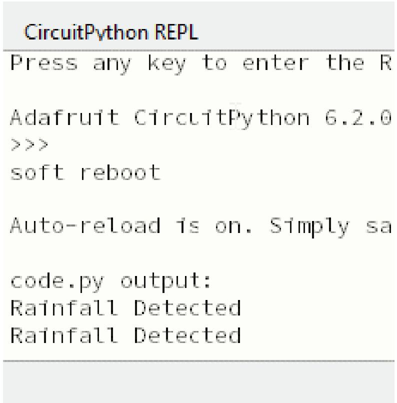


Рисунок 5.12. Датчик осадков

В следующем разделе мы проверим анемометр и флюгер.

Проверка анемометра и флюгера

В этом разделе мы проверим анемометр и флюгер. Оба датчика подключены к одному и тому же разъему RJ11 (как показано на рисунке):

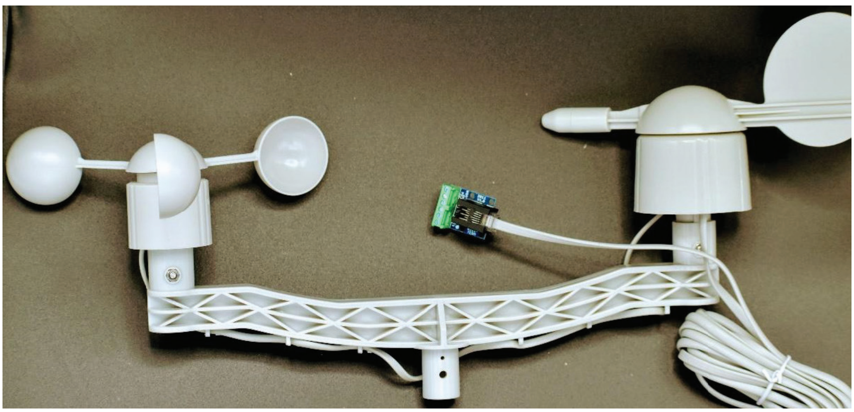


Рисунок 5.13. Анемометр и флюгер, подключенные к переходной плате RJ11

Давайте посмотрим, как определить скорость и направление ветра.

Анемометр

Анемометр – это прибор (показан в левой части рис. 5.13), который используется для измерения скорости ветра. Согласно техническому описанию, анемометр поставляется с герконовым замыкателем. Когда ветер перемещает вращающийся элемент, магнит периодически замыкает контакты.

Одно замыкание переключателя в секунду соответствует скорости ветра 1,492 мили (примерно 2,5 км) в час. Схема, необходимая для считывания скорости ветра, идентична схеме на рис. 5.11, за исключением того, что вывод анемометра подключен к выводу 14 GPIO.

Обсуждаемый далее пример кода, необходимого для измерения скорости ветра, доступен в виде *code_anemometer.py*.

1. Первым шагом является импорт всех необходимых модулей. Как и в предыдущем примере, мы будем использовать модуль `countio`:

```
import time
import board
import countio
```

2. Мы инициализируем счетчик на выводе 14 и подсчитываем количество импульсов в секунду, используя функцию `time.monotonic()`, возвращающую время в секундах. Мы используем это время для расчета скорости ветра:

```
with countio.Counter(board.GP14) as pin_counter:
  last_tick = time.monotonic()
  while True:
    if pin_counter.count and ((time.monotonic() -
    last_tick) > 1.0):
    print("Wind Speed = {0} mph".format(pin_counter.count * 1.492))
    pin_counter.reset()
    last_tick = time.monotonic()
```

3. Сохраните образец кода в виде *code.py* и попробуйте покрутить анемометр; вы должны начать видеть показания скорости ветра в милях в час (mph):

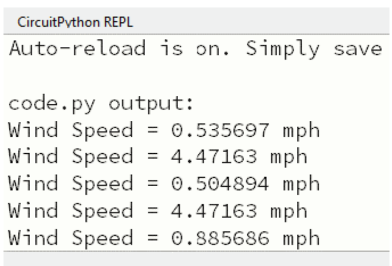


Рисунок 5.14. Подсчет скорости ветра

Мы можем проверить скорость ветра, установив снаружи комплект измерителя погоды и сравнив его с данными из общедоступных источников данных[46]. В следующем разделе собираемся протестировать флюгер.

[46] Скорее всего, в условиях, показанных на фото (рис. 5.1 или 5.18), вы ничего пригодного для сравнения не получите. Ветер в приземном слое замедляется и образует вихри, на него влияют различные препятствия (на фото это довольно высокий забор и не попавшие в кадр строения, деревья и кусты) – у вас будут в разы меняться показания от измерения к измерению, причем в среднем они будут гораздо меньше указанного в сводке погоды для данной местности. В профессиональных метеостанциях датчики ветра выносятся на мачту высотой (стандартно) 100 м – именно на такой высоте, при равнинном рельефе и условии отсутствия высоких сооружений поблизости, влияние поверхности на скорость потоков воздуха снижается до приемлемых величин. – *Прим. перев.*

Проверка датчика направления

В этом разделе мы собираемся протестировать флюгер – инструмент, используемый для определения направления ветра. Флюгер использует герконовые переключатели в сочетании с резисторами, образующими делитель напряжения с добавочным внешним сопротивлением. Выходное напряжение делителя служит для определения направления ветра.

Флюгер соединен с Pico, как показано на схеме рис. 5.15. Выход делителя напряжения подключается к выводу 26 GPIO:

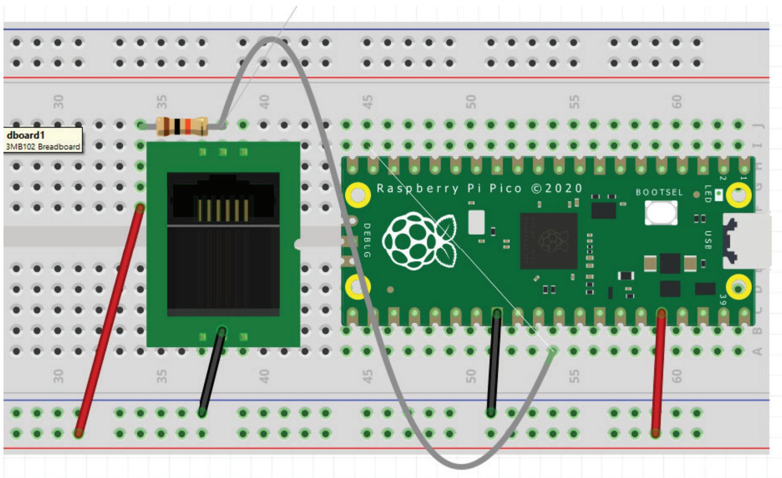


Рисунок 5.15. Подключение флюгера к Pico

Пример кода, обсуждаемый в этом разделе, находится в файле *code_windvane.py*. Давайте взглянем на код:

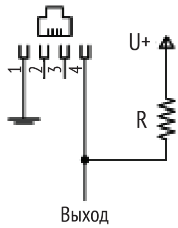
1. Первым шагом является загрузка необходимых модулей. Поскольку флюгер имеет аналоговый выход, мы импортируем модуль `analogio`:

```
import time
import board
import analogio
```

2. GP26 инициализируем как вход аналого-цифрового преобразователя (АЦП):

```
vane = analogio.AnalogIn(board.GP26)
```

3. Направление рассчитывается на основе значений сопротивления, указанных в техническом описании. Если для создания делителя напряжения использовать резистор 10 КОм, то направление можно определить в соответствии с таблицей:



Направление (градусы)	Сопротивление (Ом)	Напряжение (U = 5 В; R = 10 КОм)
0	33 КОм	3,84 В
22,5	6,57 КОм	1,98 В
45	8,2 КОм	2,25 В
67,5	891 Ом	0,41 В
90	1 КОм	0,45 В
112,5	688 Ом	0,32 В
135	2,2 КОм	0,90 В
157,5	1,41 КОм	0,62 В
180	3,9 КОм	1,40 В
202,5	3,14 КОм	1,19 В
225	16 КОм	3,08 В
247,5	14,12 КОм	2,93 В
270	120 КОм	4,62 В
292,5	42,12 КОм	4,04 В
315	64,9 КОм	4,78 В
337,5	21,88 КОм	3,43 В

Рисунок 5.16. Зависимость направления от значений выходного сопротивления (из технического описания флюгера)

4. Разрешение АЦП RP2040 составляет 12 бит, но библиотека CircuitPython масштабирует его до 16 бит. АЦП выводит значение от 0 до 65 535, где 0 представляет 0 В и 65 535 представляют 3,3 В. Для вычисления напряжения на основе необработанных (`raw`) значений АЦП мы напишем вспомогательную функцию `get_voltage`:

```
def get_voltage(raw):
  return (raw * 3.3) / 65536
```

5. Теперь для определения направления ветра можно использовать рассчитанное функцией напряжение. Согласно информации из технического описания (см. рис. 5.16) составляем функцию `win_dir`:

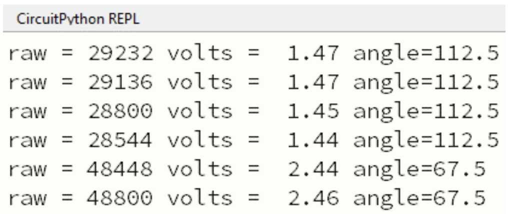
```
def wind_dir(vin):
  if vin < 150: windDir="202.5"
  elif vin < 300: windDir = "180"
  elif vin < 400: windDir = "247.5"
  elif vin < 600: windDir = "225"
  elif vin < 900: windDir = "292.5"
  elif vin < 1100: windDir = "270"
  elif vin < 1500: windDir = "112.5"
  elif vin < 1700: windDir = "135"
  elif vin < 2250: windDir = "337.5"
  elif vin < 2350: windDir = "315"
  elif vin < 2700: windDir = "67.5"
  elif vin < 3000: windDir = "90"
  elif vin < 3200: windDir = "22.5"
  elif vin < 3400: windDir = "45"
```

```
    elif vin < 4000: windDir = "0"
    else: windDir = "0"
    return windDir
```

6. Таким образом, мы считываем значение АЦП, преобразуем в напряжение и вычисляем направление:

```
while True:
    raw = vane.value
    volts = get_voltage(raw)
    print("raw = {:5d} volts = {:5.2f} angle={}".format(raw, volts, wind _dir(volts * 1000)))
    time.sleep(0.5)
```

7. Сохраним образец кода как *code.py* и, поворачивая флюгер вокруг своей оси, можем видеть следующий результат:



```
CircuitPython REPL
raw = 29232 volts =  1.47 angle=112.5
raw = 29136 volts =  1.47 angle=112.5
raw = 28800 volts =  1.45 angle=112.5
raw = 28544 volts =  1.44 angle=112.5
raw = 48448 volts =  2.44 angle=67.5
raw = 48800 volts =  2.46 angle=67.5
```

Рисунок 5.17. Значения направления ветра

Теперь, когда мы протестировали датчики измерителя погоды, пришло время собрать все воедино.

Тестирование беспроводного модуля

Беспроводной модуль можно использовать для загрузки данных в облако или получения информации о погоде через интернет. Мы рекомендуем ознакомиться с главой 2 «*Последовательные интерфейсы и их приложения*», чтобы настроить и протестировать беспроводной модуль.

Сборка и тестирование метеостанции

Мы рекомендуем следовать отличному руководству от SparkFun по сборке комплекта: **https://bit.ly/3yTgZZu**. Мы собрали свой комплект и попробовали установить его на нашем заднем дворе, как показано на рис. 5.18. Для нашей установки мы использовали Pico и для удобства несущую плату Pico Omnibus, чтобы избежать запутанных проводов на открытом воздухе.

Мы протестировали нашу метеостанцию без учета следующих соображений:

- выбор источника питания для метеостанции;
- монтаж несущей платы на метеостанцию;
- защита несущей платы от атмосферных воздействий внутри корпуса;
- надежная установка метеостанции.

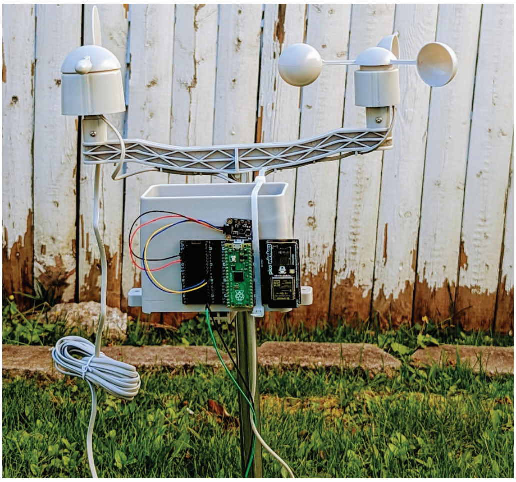


Рисунок 5.18. Установка метеостанции на заднем дворе

Потребуется тщательное тестирование, чтобы выявить и устранить все проблемы. Первым шагом является устранение любых проблем, связанных с подключением датчиков. Затем нам нужно убедиться, что корпус, который мы выберем для несущей платы, выдержит зимние погодные условия, включая сильный ветер и снегопад. Источник питания также должен быть способен выдерживать такие условия. Есть интересная статья от SparkFun об уроках, извлеченных из их пробного развертывания: **https://learn.sparkfun.com/tutorials/weatherstation-wirelessly-connected-to-wunderground#lessons-learned**[47].

Мы намерены предпринять в рамках этого проекта следующее.

Следующие шаги

На момент написания этой главы мы не встречали какой-либо дополнительной аппаратной платы для Pico, специально предназначенной для установки датчиков метеостанции. Мы планируем создать свою собственную печатную плату (PCB) и поделимся нашими успехами в репозитории GitHub этой книги.

Мы хотели бы обратить внимание, что будем публиковать там все изменения в проекте, включая любые недостатки, которые мы, возможно, пропустили во время сборки.

[47] К сожалению, и авторы книги, и компания SparkFun упускают в этих обсуждениях важную тему достоверности показаний датчиков, без чего все усилия по обеспечению надежности работы могут оказаться бесполезными (см. также сноски 42 и 46 в этой главе). Для обеспечения достоверных измерений требуется индивидуальная калибровка и периодическая поверка датчиков, что формально требует доступа к сертифицированной метрологической лаборатории и представляет собой весьма затратную во всех отношениях процедуру (и еще не каждый датчик из указанных в этой книге сможет ее пройти). Но, конечно, получение официального заключения необходимо, только если вы собираетесь использовать результаты для каких-то серьезных целей (научной статьи, использования в суде в качестве доказательств и т. п.). Для домашнего использования вполне достаточно провести простейшую калибровку самостоятельно с помощью доступных средств. Что, однако, все равно невозможно без знания элементарных основ метрологии: классификации ошибок измерений, причин их возникновения и способов устранения, представления об обработке данных и образцовых средствах измерения для каждого поверяемого параметра и т. д. – *Прим. перев.*

Итоги

В этом проекте мы обсуждали проведение гражданского научного эксперимента путем создания собственной метеостанции с использованием Raspberry Pi Pico. Мы протестировали датчики и собрали метеостанцию вместе. Мы также обсудили проблемы неполадок, связанных с развертыванием станции на открытом воздухе, и необходимость защиты нашей установки от атмосферных воздействий.

Поскольку эксперименты гражданской науки выигрывают от обмена данными, мы планируем опубликовать статус нашей метеостанции в общедоступном потоке данных.

В следующей главе мы собираемся повеселиться с дисплеями!

Глава 6

Проектируем настенный семисегментный дисплей

В этой главе мы позабавимся с некоторыми дисплеями в ретро-стиле. Мы построим настенный дисплей, управляемый Pico. Создавать средства отображения с использованием светодиодных семисегментных индикаторов – это увлекательный опыт. К концу данной главы вы должны быть в состоянии создать что-то вроде дисплея, показанного на рисунке:

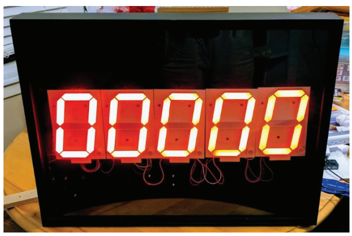


Рисунок 6.1. Большой настенный семисегментный дисплей

В этой главе мы рассмотрим создание подобных экранов, а также варианты создания дисплеев различных размеров и применений. Темы, обсуждаемые в данной главе, включают следующее:

- мотивации проекта;
- установка необходимых библиотек;
- выбор семисегментных индикаторов;
- подключение настенного семисегментного дисплея;
- написание драйверов для семисегментного дисплея;
- использование дисплея;
- сборка всего этого воедино.

Технические требования

Для этой главы рекомендуется использовать следующее оборудование:

- Raspberry Pi Pico (ссылка: **https://bit.ly/3AJtoAf**);
- Pico Omnibus – двойной расширитель (ссылка: **https://bit.ly/3sr2GJR**);
- беспроводной модуль Pico (ссылка: **https://bit.ly/3yPuoT9**);

- 6,5-дюймовый семисегментный индикатор с общим анодом (5 шт.; ссылка: **https://bit.ly/3z6ii76**)[48];
- драйвер семисегментных индикаторов SparkFun Large Digit Driver (5 шт.; ссылка: **https://bit.ly/3hpLC2F**)[49];
- соединительные провода Male to Female, Male to Male и Female to Female (ссылка: **https://iarduino.ru/shop/Provoda-kabeli/**);
- источник питания 12 В (ссылка: **https://www.chipdip.ru/product/gst25e12-p1j**);
- закрытый корпус с прозрачным окном из вашего местного магазина хобби (что-то вроде коробки, показанной на рисунке выше[50]).

Примеры кода для этой главы доступны по адресу **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIY-Workshop/tree/main/chapter_06**.

Видеоролики с кодом в действии для этой главы можно посмотреть по адресу **https://bit.ly/3MN0lBO**.

> **Семисегментные цифровые индикаторы**
>
> Для этого проекта мы использовали пять семисегментных индикаторов для отображения любого числа длиной до пяти цифр. Код, обсуждаемый в данной главе, может быть легко адаптирован к количеству цифр, необходимому для вашего проекта.

В следующем разделе мы обсудим обоснования для проекта в этой главе.

О мотивации проекта

У нас есть общий интерес к созданию наглядных средств отображения для улучшения качества жизни. Эти средства могут использоваться для вывода рассчитанных данных и служить для информирования пользователей или для других целей.

Ранее мы уже использовали это в медицинских учреждениях. Например, людям с хроническими заболеваниями или склонностью к развитию хронических заболеваний, таких как диабет либо ожирение, может быть очень удобно иметь такие дисплеи. Мы разрабатывали решения, которые фиксировали ежедневное количество выполненных шагов или, наоборот, оставшееся необходимое количество, что при размещении на видном месте помогает не пропустить напоминание. С использованием Pico такие наглядные средства могут быть разработаны и для других применений. За пределами сферы здравоохранения подобные дисплеи можно, например, использовать для показа результатов соревнующихся команд.

---

[48] О выборе дисплея в российских условиях см. далее стр. 137. – *Прим. перев.*

[49] В российских условиях можно приобрести в «Чипе и Дипе» (**https://www.chipdip.ru/product0/8007708141**). – *Прим. перев.*

[50] Готовый корпус таких больших размеров нужной конфигурации приобрести затруднительно, проще самостоятельно склеить из оргстекла и покрасить изнутри черной краской. Съемная передняя стенка остается прозрачной, однако для проекта дисплея из этой главы ее лучше выполнить из дымчатого (затемненного) полупрозрачного оргстекла. См. также раздел «*Собираем все вместе*» в конце главы. – *Прим. перев.*

Для нашей предыдущей книги «*Программирование на Python с помощью Raspberry Pi*» мы создали на основе Raspberry Pi Zero панель мониторинга личного здоровья, которая отслеживала физическую активность в виде шкального индикатора. Индикатор состоит из восьми светодиодов RGB, и очередной светодиод загорался зеленым цветом на каждые 1250 шагов, отслеживаемых с помощью фитнес-трекера. На рисунке показана такая панель мониторинга, похожая на индикатор выполнения в Windows:

Рисунок 6.2. Панель мониторинга личного здоровья на основе Raspberry Pi Zero

Поскольку панель мониторинга отслеживает только ход физической активности и не предоставляет никакой информации о фактическом количестве шагов, оставшихся за день, мы решили создать панель мониторинга большего размера. Это позволило бы планировать соответствующую физическую активность в зависимости от имеющегося времени. Поэтому мы создали большую цифровую панель и решили поделиться нашей работой на выставке Maker Fair в Кливленде:

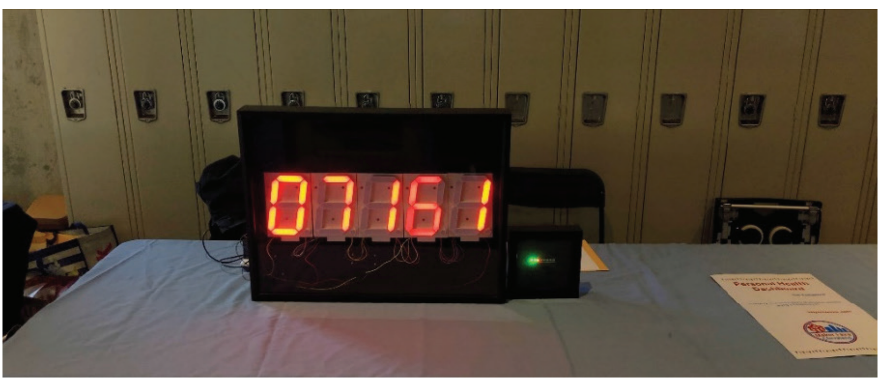


Рисунок 6.3. Экспозиция цифровой панели на выставке Maker Faire в Кливленде

На рис. 6.3 показаны обе панели мониторинга рядом друг с другом. Вы, наверное, заметили, что на большом дисплее высвечивается цифра 7161. Это означает, что человеку, носящему фитнес-трекер, осталось сделать 7161 шаг, чтобы выполнить свою ежедневную программу. Вместо того чтобы вести отсчет количества ежедневно сделанных шагов, панель мониторинга ведет обратный отсчет от заданного количества. Это позволяет планировать свою физическую активность в соответствии с шагами, оставшимися на день.

Мы решили переделать дизайн этого дисплея с применением Pico и хотели поделиться нашим решением в этой главе.

Возможные варианты использования

Некоторые потенциальные варианты использования проекта включают следующее:

- отслеживание физической активности;
- отслеживание количества человек в пространстве;
- отслеживание подписчиков на платформах социальных сетей;
- отслеживание результатов конкурсов.

В данной главе мы покажем, как создать дисплей для любой из этих целей.

Установка необходимых библиотек

В этом разделе мы установим библиотеки, необходимые для проекта. К ним относятся драйверы для беспроводного модуля и семисегментного дисплея.

> **Установка CircuitPython**
>
> Мы предполагаем, что вы установили CircuitPython на свой Pico. Если вы незнакомы с процессом установки, рекомендуется следовать инструкциям из главы 1 «*Начало работы с Raspberry Pi Pico*».

Все библиотеки являются частью пакета Adafruit CircuitPython. Последняя версия пакета может быть загружена в виде ZIP-файла с **https://circuitpython.org/libraries**. Мы использовали версию пакета, предназначенную для CircuitPython 6.x.x.

После загрузки ZIP-файла извлеките его содержимое, чтобы вы могли скопировать библиотеки, необходимые нам для проекта.

Беспроводной модуль

Нам нужна библиотека *adafruit_esp32spi* для беспроводного модуля. Скопируйте папку с тем же именем в папку *lib* вашего Pico. Нам также понадобится двоичный файл *adafruit_reqests.mpy* из пакета библиотеки.

> **Настройка беспроводного модуля**
>
> В главе 2 «*Последовательные интерфейсы и их приложения*» мы подробно обсудили настройку беспроводного модуля. Мы рекомендуем вам прочитать эту главу, а затем настроить беспроводной модуль.

Выбор семисегментных индикаторов

Для этой главы мы выбрали 6,5-дюймовые семисегментные индикаторы. Мы выбрали их из-за больших размеров, но для каждой цифры нужен отдельный драйвер. На следующем рисунке показан семисегментный индикатор с драйвером, припаянным к его задней панели:

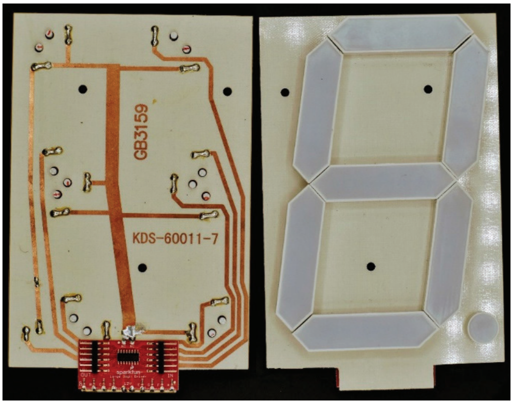


Рисунок 6.4. Семисегментный индикатор высотой 6.5 дюйма с драйвером на задней панели

> **О выборе индикаторов в российских условиях**
>
> Предлагаемые индикаторы довольно дорогие, поэтому стоит предложить несколько альтернатив. В российских условиях их можно заменить на чаще встречающиеся в продаже, более дешевые и доступные в разных цветах свечения 5-дюймовые индикаторы (с высотой цифры 127 мм) фирм Ningbo Foryard Optoelectronics FYS-50011Bx (**http://www.foryard-led.ru/led_numeric_display/single_digit_display/5.00inch**) или G-Nor Optoelectronics GNS-50011Bx (**http://www.g-nor.com/Pr_d_gci_96_id_65.html**). В обозначении «B» означает общий анод, а вместо последней «x» подставляется обозначение цвета свечения. Схема включения и управление этими индикаторами такие же, как описано в тексте, иная только разводка выводов, и, соответственно, установка платы драйвера отличается от показанного на рис. 6.4. При замене на индикаторы еще меньшего размера, возможно, следует внимательно ознакомиться с характеристиками индикаторов и драйвера – выходной ток может быть слишком велик, так как он управляется простыми резисторами, рассчитанными под определенное падение напряжения на сегменте.

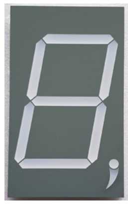


Рисунок 6.5. Цифровой сегментный индикатор FYS-50011Bx

Мы также можем порекомендовать семисегментный дисплей от Adafruit Industries. Он поставляется с интерфейсом I2C, что позволяет легко подключить его к Pico. Adafruit предоставляет специальные драйверы для этого дисплея, так как он использует динамическую индикацию[51]:

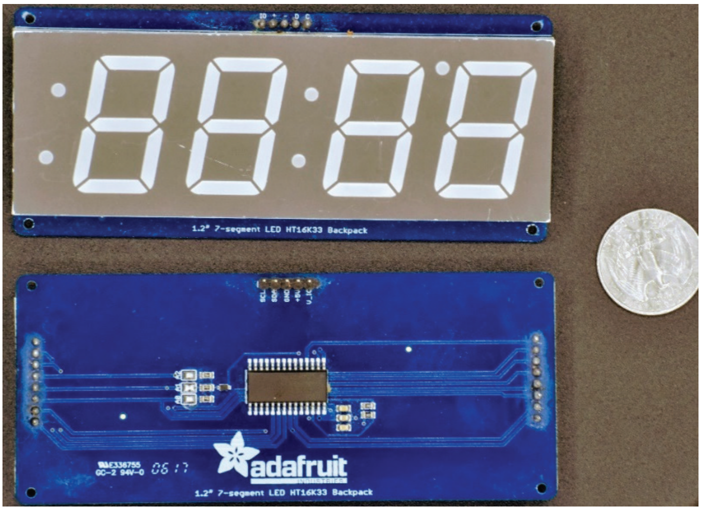


Рисунок 6.6. Семисегментный дисплей HT16K33 от Adafruit Industries

> **Оптимизация схемы управления**
>
> Переделка программы и схемы под динамическую индикацию вместо управления каждой цифрой в отдельности значительно упростит выполнение соединений и в целом удешевит проект (например, потребуется всего один драйвер на все индикаторы, как видно по рис. 6.6). В продаже имеются специальные сдвоенные, строенные и счетверенные цифровые индикаторы, что также обходится дешевле, чем аналогичное количество одинарных. Однако в этом случае, вероятно, придется отказаться от предлагаемого драйвера от SparkFun и пристраивать собственный более универсальный. Ознакомиться с принципами динамической индикации и вариантами схемотехнических решений можно по этой ссылке: **https://habr.com/ru/post/591945/**.

---

[51] Однако употреблять этот дисплей для каких-либо целей, кроме построения часов, неудобно из-за наличия промежутка между цифрами, занятого разделительным двоеточием. – *Прим. перев.*

Поскольку мы являемся поклонниками большого семисегментного дисплея, для создания нашего проекта мы выбрали индикаторы от SparkFun (показанные на рис. 6.4).

Подключение настенного семисегментного дисплея

В этом разделе мы подключим дисплей к Raspberry Pi Pico, используя двойной расширитель Pico Omnibus от Pimoroni. Это позволяет подключить беспроводной модуль и семисегментный дисплей. Шаги по подключению дисплея следующие:

1. Приготовьте отдельные семисегментные цифры. Драйвер должен быть припаян к обратной стороне каждого индикатора (рис. 6.4). Пайка драйвера – это очень простой шаг, если следовать инструкциям, изложенным в SparkFun (ссылка: **https://bit.ly/3hLUobk**).
2. Затем подключаем семисегментный драйвер к следующим выводам Pico:
   - Latch → GP17;
   - Clock → GP18;
   - Serial → GP19;
   - 5V → VBUS;
   - 12V → внешнее питание 12 В.

На следующем рисунке показана схема соединений между семисегментным драйвером и Raspberry Pi Pico:

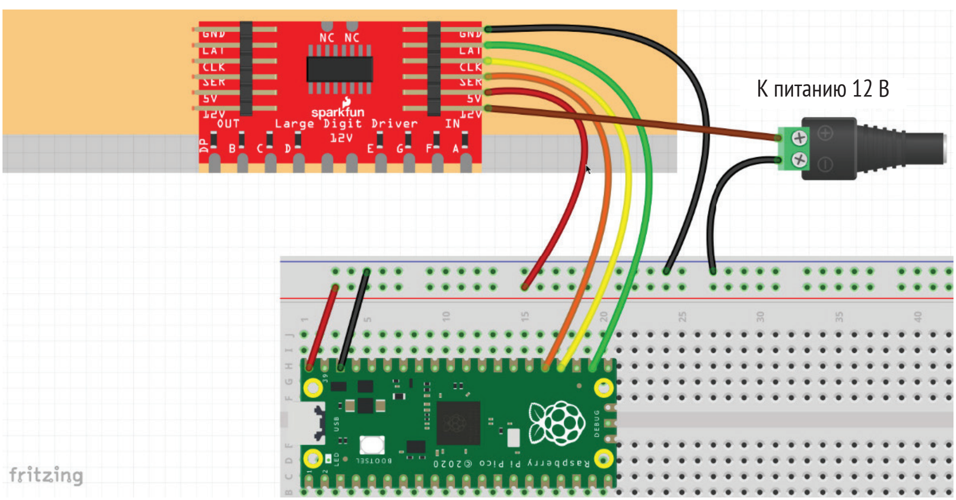


Рисунок 6.7. Схема соединений Pico с семисегментным драйвером

3. Драйвер имеет два набора контактов – входные (*In*) и выходные (*Out*). Pico соединяется со стороной входов (*In*).

4. Следует позаботиться о том, чтобы общий провод (минус) внешнего источника питания и контакт GND Pico были соединены друг с другом (см. схему на рис. 6.7).
5. Следующий шаг – последовательно соединить столько цифр, сколько вы хотите. Выходные контакты (*Out*) подключены ко входам (*In*) следующей цифры, как показано на схеме:

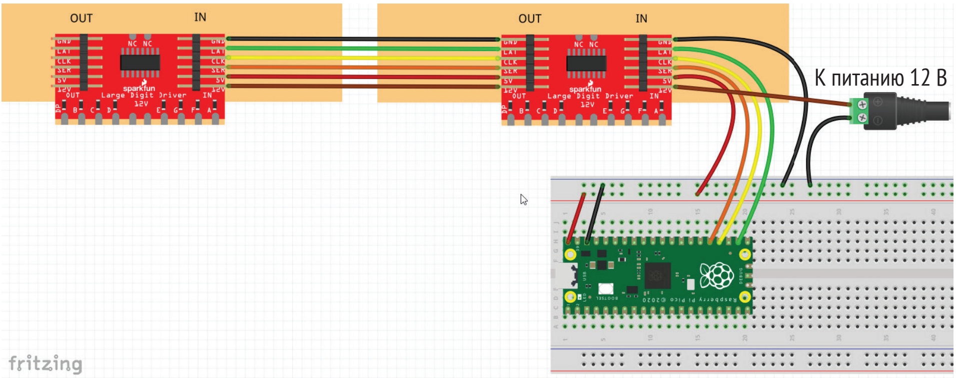


Рисунок 6.8. Два семисегментных драйвера, соединенных последовательно

6. Для нашего проекта подключается до пяти цифр:

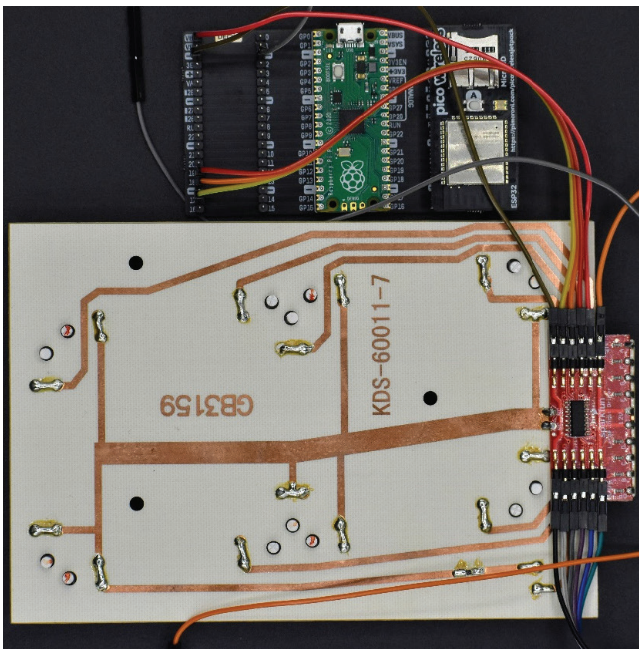


Рисунок 6.9. Настенный семисегментный дисплей, подключенный к Raspberry Pi Pico

В следующем разделе мы обсудим драйверы этого дисплея.

Создание драйверов для семисегментного дисплея

В предыдущем разделе мы подключили дисплей. Теперь нам нужно написать драйверы для его тестирования. Мы перенесли код C++, предоставленный SparkFun, на CircuitPython.

1. Первым шагом является импорт всех необходимых модулей для драйвера:

```
import board
import busio
import time
from digitalio import DigitalInOut, Direction, Pull
```

2. Мы объявляем контакты защелки данных (`latch`), тактовых импульсов (`clock`) и данных (`data`), а также устанавливаем их в качестве выходных контактов с исходным низким уровнем:

```
latch = DigitalInOut(board.GP22)
clock = DigitalInOut(board.GP26)
data = DigitalInOut(board.GP27)

latch.direction = Direction.OUTPUT
clock.direction = Direction.OUTPUT
data.direction = Direction.OUTPUT

latch.value = False
clock.value = False
data.value = False
```

3. Функция `post_number` используется для отображения знака в выбранной позиции. Это достигается путем включения соответствующих сегментов. Например, для того чтобы отобразить цифру 1, нам нужно включить сегменты B и C. На следующем рисунке показаны наименования сегментов семисегментного индикатора:

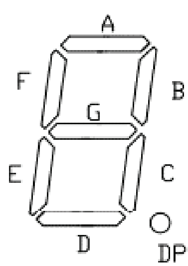


Рисунок 6.10. Наименования сегментов семисегментного индикатора

4. Плата драйвера состоит из 8-разрядного сдвигового регистра. Отображаемый знак (number) выводится с помощью последовательного сдвига кода цифры:

```
def post_number(number, decimal=False):
  a = 1<<0
  b = 1<<6
  c = 1<<5
  d = 1<<4
  e = 1<<3
  f = 1<<1
  g = 1<<2
  dp = 1<<7

  if number == 1: segments = b | c
  elif number == 2: segments = a | b | d | e | g
  elif number == 3: segments = a | b | c | d | g
  elif number == 4: segments = f | g | b | c
  elif number == 5: segments = a | f | g | c | d
  elif number == 6: segments = a | f | g | e | c | d
  elif number == 7: segments = a | b | c
  elif number == 8: segments = a | b | c | d | e | f | g
  elif number == 9: segments = a | b | c | d | f | g
  elif number == 0: segments = a | b | c | d | e | f
  elif number == " ": segments = 0
  elif number == "c": segments = g | e | d
  elif number == "-": segments = g
  else: segments = 0

  if decimal: segments |= dp

  for i in range(8):
    clock.value = False
    data.value = segments & (1 << (7 - i))
    clock.value = True
```

5. Как упоминалось ранее, разряды дисплея последовательно соединены друг с другом. Поскольку в этом проекте мы используем пять разрядов, то можем отображать нужные цифры в каждой позиции, сдвигая на соответствующее число разрядов. Функция show_number используется для извлечения цифр из числа и их отображения:

```
def show_number(x):

  for num in range(5):
    remainder = int(x % 10)
    post_number(remainder, False)
```

```
        x = int(x / 10)

    latch.value = False
    latch.value = True
```

Поскольку наш дисплей состоит из пяти цифр, мы повторяем цикл `for` пять раз. Это гарантирует, что при необходимости мы добавим к числу ведущие нули. На следующем рисунке показаны две цифры дисплея, показывающие число 47 во время тестирования:

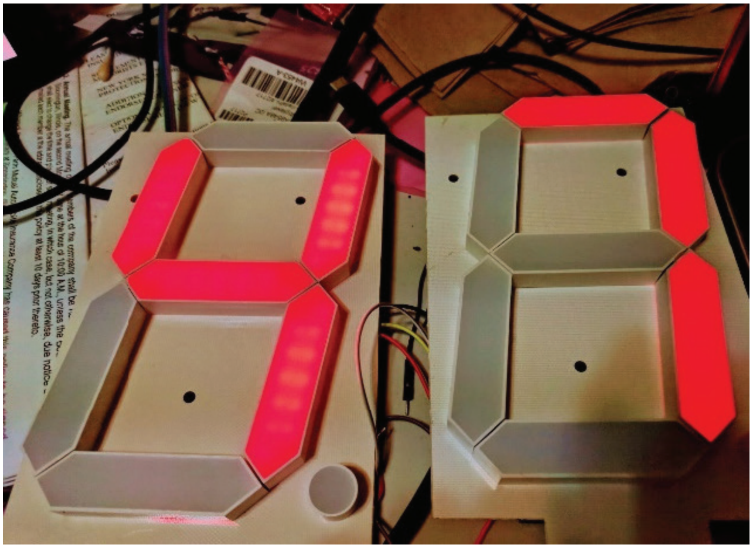


Рисунок 6.11. Дисплей, подключенный к Pico

Драйвер, который мы обсуждали в этом разделе, доступен для загрузки вместе с остальными материалами этой главы в файле *seven_segment.py*. Сохраните файл в Pico, так как мы будем использовать его в дальнейших примерах. В следующем разделе мы покажем, как собрать все воедино.

Использование дисплея

В этом разделе мы обсудим обновление информации на дисплее двумя способами: с помощью простого веб-сервера, работающего на Pico, и через последовательный порт.

Итак, начнем!

Простой веб-сервер

В этом примере мы разместим простой веб-сервер на Pico, чтобы можно было обновлять информацию с любого устройства в локальной сети с помощью браузера. Данный пример основан на примере *esp32spi_wsgiserver*, доступного на сайте Adafruit (ссылка: **https://bit.ly/3itHrmY**). Мы внесли несколько изменений в представленный код, чтобы адаптировать его к нашему примеру. Образец измененного кода доступен для загрузки из репозитория этой главы в виде файла *code_server.py* (ссылка: **https://bit.ly/3hqWA7X**).

Изменения, внесенные в пример Adafruit, включают следующее.

- Импорт драйвера *seven_segment*, чтобы иметь возможность обновлять информацию при поступлении запроса:

```
import seven_segment
```

- Обновляем выводы GPIO для управления беспроводным пакетом ESP32:

```
# If you have an externally connected ESP32:
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)
spi = busio.SPI(board.GP18, board.GP19, board.GP16)
esp = adafruit_esp32spi.ESP_SPIcontrol(
  spi, esp32_cs, esp32_ready, esp32_reset
)
```

- Регистрируем метод с именем `display` для обновления информации при поступлении входящего запроса `GET`:

```
web_app = SimpleWSGIApplication(static_dir=static)
web_app.on("GET", "/display", display)
```

Метод `display` вызывается всякий раз, когда в запросе вызывается путь `/display`. Метод `display` включает в себя следующее:

```
def display(environ): # pylint: disable=unused-argument
  if "QUERY_STRING" in environ.keys():
    query = environ['QUERY_STRING']
    if "value=" in query:
      value = query[6:]
      print(value)
    seven_segment.show_number(int(value))
  return web_app.serve_file("static/index.html")
```

- В методе `display` мы проверяем, содержит ли запрос какие-либо параметры. Если есть параметр с именем `value`, мы его анализируем и преобразуем в целое число.

- Затем мы используем драйвер `seven_segment` для обновления информации на дисплее.

- Нам также необходимо вернуть ответ. Метод возвращает статическую веб-страницу, которая гласит `Success!`. Статический HTML-файл доступен вместе с примерами кода, приведенными в этой главе.

- Пришло время попробовать сервер. После того как вы подключили дисплей, Pico должен содержать следующие файлы (см. рис. 6.12):

- семисегментный драйвер для дисплея (*seven_segment.py*);
- *secrets.py* для получения учетных данных Wi-Fi;
- HTML-файлы в папке `static`.

Файлы должны выглядеть так, как показано на рисунке:

Name	Date modified	Type	Size
.fseventsd	12/31/2019 11:00 PM	File folder	
lib	12/31/2019 11:00 PM	File folder	
static	9/9/2021 12:09 AM	File folder	
.metadata_never_index	12/31/2019 11:00 PM	METADATA_NEVE...	0 KB
.Trashes	12/31/2019 11:00 PM	TRASHES File	0 KB
boot_out.txt	12/31/2019 11:00 PM	Text Document	1 KB
code.py	10/3/2021 5:01 PM	Python File	7 KB
secrets.py	9/9/2021 12:08 AM	Python File	1 KB
seven_segment.py	10/3/2021 1:56 PM	Python File	2 KB

Рисунок 6.12. Файлы для примера веб-сервера

- После того как вы загрузили образец кода в виде *code.py* и открыли терминал последовательного порта, Pico использует учетные данные, предоставленные в файле *secrets.py* для подключения к беспроводной сети. Запустится сервер и предоставит IP-адрес Pico, как показано на следующем рисунке:

```
code.py output:
ESP32 SPI simple web server test!
MAC addr: ['0x0', '0x65', '0x9a', '0x57', '0xdd', '0xc4']
MAC addr actual: ['0xc4', '0xdd', '0x57', '0x9a', '0x65', '0x0']
open this IP in your browser:  192.168.86.32
```

Рисунок 6.13. IP-адрес веб-сервера Pico

- IP-адрес Pico мы используем для обновления дисплея. Согласно рис. 6.13, IP-адрес равен 192.168.86.32. Давайте сделаем запрос `GET` для отображения числа 789.
- Запустите браузер с устройства, подключенного к той же сети, что и Pico. Введите в адресную строку следующее: `http://192.168.86.32/display?value=789`.
- Если запрос GET прошел успешно, браузер должен вернуть статическую веб-страницу, как показано на рис. 6.14:

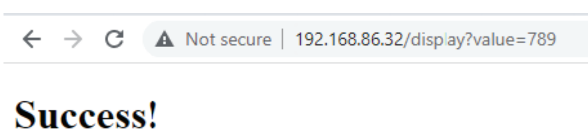


Рисунок 6.14. Ответ веб-сервера Pico

- Дисплей при этом должен обновиться до 00789 (обратите внимание на ведущие нули), как показано на рис. 6.15:

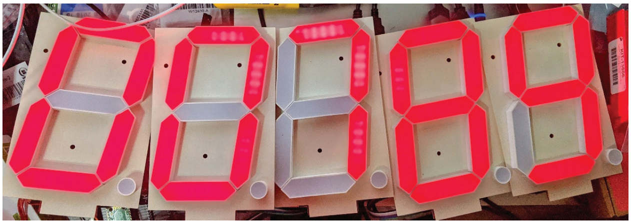


Рисунок 6.15. Дисплей, показывающий номер 00789

В случае возникновения каких-либо проблем проверьте выходные данные последовательного порта на наличие ошибок. Мы рекомендуем устранять неполадки в следующей последовательности:

- убедитесь, что вы загрузили все файлы, как описано в этом разделе;
- убедитесь, что все подключено правильно, включая беспроводной блок и контакты дисплея;
- убедитесь, что Pico подключен к беспроводной сети.

Далее мы обсудим управление дисплеем через последовательный порт.

Пример управления через последовательный порт

Если вы еще незнакомы с последовательными портами, мы рекомендуем ознакомиться с главой 2 «*Последовательные интерфейсы и их приложения*».

Метод управления через последовательный порт полезен, когда Pico подключается через USB к Raspberry Pi. В этом примере Pico используется только для управления дисплеем. Пример кода, обсуждаемый в данном разделе, доступен для загрузки вместе с материалами этой главы как *code_serial.py*.

- Мы используем семисегментные драйверы дисплея, рассмотренные ранее в этой главе, но добавьте цикл `while` для захвата вводимых через последовательный порт данных:

```
while True:
  input_string = input("Enter a value to update?")
  if input_string[0] == "S":
    value_string = input_string[1:]
    try:
      value = int(value_string)
    except:
      print("Enter a valid number")
  else:
      show_number(value)
```

- Здесь мы фиксируем вводимые пользователем данные с помощью функции `input()`. Если пользовательский ввод имеет определенный формат (например, если первый символ ввода *S*), извлекаем остальную часть строки и пытаемся преобразовать ее в целое число.

- Если попытка преобразования завершается неудачей, мы запрашиваем другой ввод. В противном случае обновляем дисплей с помощью полученного числа.
- В этом примере используется вызов функции ввода, который блокирует выполнение программы до тех пор, пока не появятся вводимые данные. Если вы хотите избежать таких блокирующих вызовов функций, мы рекомендуем познакомиться с модулем *supervisor* (ссылка: **https://bit.ly/3ozTgvT**) в CircuitPython. Это позволяет вызывать функцию ввода только при наличии доступных данных. В репозитории данной главы вы найдете файл *code_supervisor.py*, в котором имеется пример использования модуля супервизора.

Мы решили обсудить пример последовательного порта, потому что дисплей, который мы изначально создали для Maker Fair, использовал Arduino и Raspberry Pi. Raspberry Pi извлекал данные из облака, в то время как Arduino использовался для управления дисплеем. Мы хотели продемонстрировать управление дисплеем через последовательный порт в таких приложениях.

Отслеживание физической активности

Чтобы отображать количество ежедневных шагов с фитнес-трекера, нам нужно, чтобы Pico подключался к облаку с использованием API, предоставляемого производителем. Такие API-интерфейсы обычно поставляются с механизмом аутентификации для доступа к данным. Например, фитнес-трекеры Fitbit поставляются с API, использующим механизм OAuth 2.0 (ссылка на документацию Fitbit API: **https://bit.ly/3mmkHX3**). В настоящее время мы изучаем возможность применения API Fitbit для получения наших данных о физической активности, а также обработку обновлений. Мы опубликуем наши результаты в репозитории GitHub этой главы.

В следующем разделе обсудим сборку дисплея в корпусе.

Собираем все вместе

После тестирования дисплея его следует собрать в корпусе. Сначала размещаем дисплей на листе фанеры, как показано на рис. 6.16:

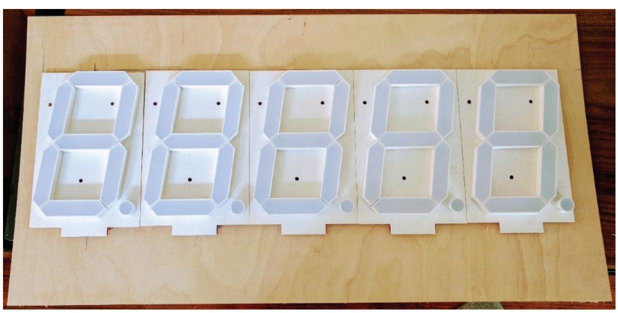


Рисунок 6.16. Планирование размещения семисегментного дисплея

Это позволит нам определить размеры корпуса, необходимого для проекта. Корпус представляет собой прямоугольную коробку с одной прозрачной стенкой[52]. Такие коробки используются для демонстрации сувениров, медалей и т. д. Они иногда продаются в магазинах, которые торгуют товарами для декоративно-прикладного искусства, рукоделия и других связанных с этим хобби.

Мы приобрели такой корпус и собрали цифры на его задней панели, как показано на рис. 6.17:

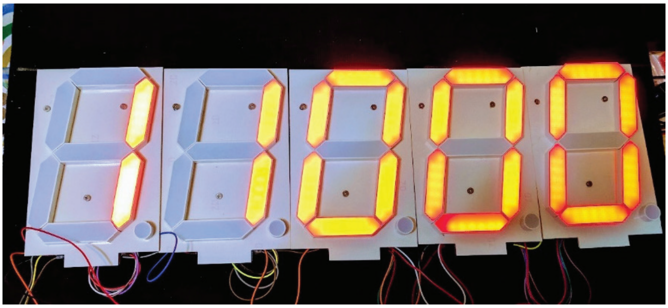


Рисунок 6.17. Семисегментные цифры, собранные на задней стенке корпуса

Поскольку плата Pico Omnibus не имеет монтажных отверстий, мы использовали двустороннюю клейкую ленту, чтобы приклеить ее к задней стороне корпуса. У вас должно получиться что-то вроде показанного на рис. 6.18.

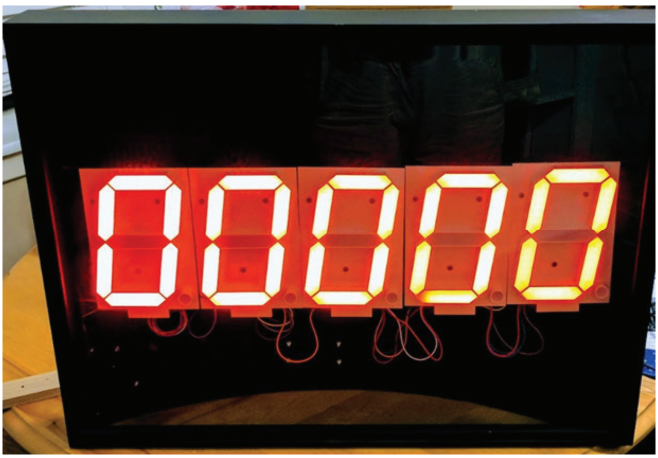


Рисунок 6.18. Настенный семисегментный дисплей в сборе

Обычно мы питаем дисплей с помощью адаптера постоянного тока напряжением 12 В[53]. Когда для получения данных на дисплее требуется подключение к интернету, мы создаем точку доступа Wi-Fi, которая имеет учетные данные, сохраненные в *secrets.py*.

[52] См. также сноску 50 на стр. 134. – *Прим. перев.*

[53] Несмотря на большие размеры, каждый сегмент такого дисплея потребляет не более 20 мА, т. е. общее максимальное потребление (при всех горящих восьмерках) равно 5 (цифр) × 7 (сегментов) × 20 (мА) = 700 мА. Итого потребности такого дисплея перекроет обычный 12-вольтовый адаптер с током 1 А. – *Прим. перев.*

Итоги

В этой главе мы обсуждали создание большого настенного семисегментного дисплея. Описаны драйверы, предназначенные для управления дисплеем. За этим последовал обзор использования дисплея, с подключением к интернету и отображением цифр. Мы также обсудили планирование макета и сборку дисплея в корпус.

В следующей главе мы обсудим создание устройства для отслеживания качества воздуха.

Глава 7

Разрабатываем устройство слежения за качеством воздуха

В предыдущей главе мы создали устройство с большим настенным семисегментным дисплеем для отслеживания физической активности. В этой главе мы собираемся создать похожий аппарат, который наглядно демонстрирует качество воздуха. К концу главы вы сможете создать нечто, подобное показанному на рисунке:

Рисунок 7.1. Наглядная демонстрация качества воздуха

Цель этого проекта – продемонстрировать качество воздуха в нашем ближайшем окружении. Мы будем проектировать устройство двумя разными способами. В первом способе осуществим отображение данных из общедоступных источников. Во втором подходе обсудим использование датчика углекислого газа, подключенного к Raspberry Pi Pico.

Темы, обсуждаемые в этой главе, включают следующее:

- мотивация проекта;
- установка необходимых библиотек;
- использование общедоступных источников данных о качестве воздуха;

- подключение датчика CO_2 к Pico;
- подключение шагового двигателя;
- построение дисплея.

Технические требования

Для этой главы рекомендуется использовать следующее оборудование:

- Raspberry Pi Pico (ссылка: **https://bit.ly/3AJtoAf**);
- Pico Omnibus – двойной расширитель (ссылка: **https://bit.ly/3sr2GJR**);
- беспроводной модуль Pico (ссылка: **https://bit.ly/3yPuoT9**);
- шаговый двигатель от автомобильного датчика (описание: **https://diyodemag.com/education/what_the_tech_gauge_stepper**)[54];
- плата драйвера шагового двигателя DRV8833 (ссылка: **https://iarduino.ru/shop/Expansion-payments/drayver-motorov-dvuhkanalnyy-drv8833.html** или **https://www.chipdip.ru/product1/8008724451**)[55];
- соединительные провода Male to Female, Male to Male и Female to Female (ссылка: **https://iarduino.ru/shop/Provoda-kabeli/**);
- дополнительно: датчик датчик CO_2 Adafruit SCD-30 (ссылка: **https://bit.ly/3l7MfQ6**)[56];
- закрытый корпус с прозрачным окном (аналогично предыдущему проекту).

Примеры кода для этой главы доступны по адресу **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIY-Workshop/tree/main/chapter_07**.

Видеоролики с кодом в действии для данной главы можно посмотреть по адресу **https://bit.ly/3OXiHC5**.

> **Установка CircuitPython**
>
> Мы предполагаем, что вы установили CircuitPython на свой Pico. Если вы незнакомы с процессом установки, то рекомендуем следовать инструкциям из главы 1 «*Начало работы с Raspberry Pi Pico*».

Мотивация проекта

Хорошее качество воздуха является главным правом для людей во всем мире. Растущая урбанизация и глобальное загрязнение продолжают снижать качество воздуха, что серьезно сказывается на качестве жизни граждан во всем мире, будь то Пешавар, Нью-Дели или где-либо еще. Гражданские ученые участвуют в

[54] В российских условиях шаговый двигатель можно поискать здесь: **http://www.autotuninggroup.ru/category/shagovye-dvigateli-motory-dlja-panelej-priborov/offset10/** или заменить подходящей моделью отсюда: **https://sellercheck.ru/c/202002642?seller=1230304**. – *Прим. перев.*

[55] По указанным адресам можно приобрести аналог описанной далее авторами управляющей платы от Adafruit, у которых названия контактов могут незначительно отличаться. – *Прим. перев.*

[56] Целесообразно заменить на **Sensirion SCD30** (в основе это один и тот же датчик, поэтому можно подключать в соответствии с рекомендациями авторов). Приобрести его можно на AliExpress или в «Чипе и Дипе» (**https://www.chipdip.ru/product0/8009717176**). – *Прим. перев.*

мониторинге качества воздуха, чтобы помочь лучше понять уровни загрязнения и причины, которые способствуют плохому качеству воздуха. Параметры, представляющие интерес, включают уровни кислорода (O_2) и двуокиси углерода (CO_2), наличие летучих органических соединений (VOC), твердых частиц (PM), образующихся в результате сжигания угля, древесины и других материалов, а также соединений серы, азота и иных элементов.

Датчики, контролирующие качество воздуха, становятся все более дешевыми, в то же время повышают разрешающую способность и точность. Это позволит гражданским ученым разрабатывать решения, позволяющие осуществлять мониторинг качества воздуха. Данные, собранные в результате таких усилий, помогают людям понимать происходящее, реагировать на загрязнение и предотвращать его, тем самым улучшая качество жизни.

Установка необходимых библиотек

В этом разделе мы установим библиотеки, необходимые для данной главы, включая управление шаговым двигателем, датчиком CO_2 и беспроводным модулем на Raspberry Pi Pico. Все библиотеки являются частью пакета Adafruit CircuitPython. Последний пакет можно загрузить в виде ZIP-файла с **https://circuitpython.org/libraries**. Мы использовали версию пакета, предназначенную для CircuitPython 6.x.x. После загрузки ZIP-файла извлеките его содержимое, чтобы иметь возможность скопировать библиотеки, необходимые нам для проекта.

Шаговый двигатель

Нам нужна библиотека *adafruit_motor* для управления шаговым двигателем (показан на следующем рисунке). Шаговый двигатель оснащен стрелкой, которая используется нами для указания соответствующей категории качества воздуха. Скопируйте библиотеку *adafruit_motor* в папку *lib* вашего Pico.

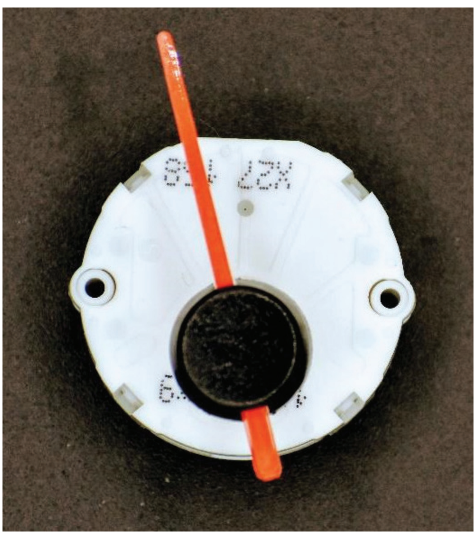


Рисунок 7.2. Шаговый двигатель от автомобильного датчика

Далее мы рассмотрим установку библиотек для беспроводного модуля.

Беспроводной модуль

Нам нужна библиотека *adafruit_esp32spi* для беспроводного модуля. Скопируйте папку с тем же именем в папку *lib* вашего Pico. Нам также понадобится двоичный файл *adafruit_reqests.mpy* из пакета библиотеки.

> **Настройка беспроводного модуля**
>
> В главе 2 «*Последовательные интерфейсы и их приложения*» мы подробно обсудили настройку беспроводного модуля. Мы рекомендуем вам прочитать эту главу, а затем настроить беспроводной модуль.

Датчик CO_2 SCD30

Поскольку датчик дорогой, это необязательный шаг. Для датчика CO_2 (показан на рис. 7.3) необходимы следующие библиотеки: *adafruit_bus_device*, *adafruit_register* и *adafruit_scd30*. Скопируйте папки *adafruit_bus_device* и *adafruit_register* вместе с файлом *adafruit_scd30.mpy* в папку *lib* вашего Pico.

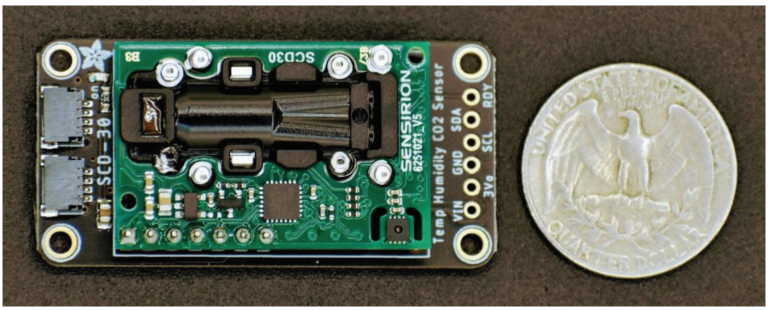


Рисунок 7.3. Датчик CO_2 SCD30

Теперь, когда мы установили необходимые библиотеки, обсудим получение данных о качестве воздуха из общедоступных источников.

Использование общедоступных источников для получения данных о качестве воздуха

В этом разделе мы обсудим получение данных о качестве воздуха с использованием **API AirNow**, предоставляемых **Агентством охраны окружающей среды США** (**US EPA**). Напоминаем, что API – аббревиатура от **Application Programming Interface** (*интерфейс прикладного программирования*). Здесь это относится к интерфейсу, предоставляемому для получения данных о качестве воздуха. API должно сопровождаться техническими требованиями к доступу. Это программа, которая объединяет данные от федеральных, государственных и местных агентств для составления отчета об индексе качества воздуха (**Air Quality Index**, **AQI**). Мы собираемся протестировать API, а затем обсудить код, необходимый для получения данных о качестве воздуха с помощью Pico.

> **Источники данных за пределами Соединенных Штатов**
>
> Если вы живете за пределами Соединенных Штатов, мы рекомендуем использовать местный источник данных в вашем регионе[57]. Если такового не существует, то рекомендуем ознакомиться с интерфейсом датчика CO_2 в разделе «*Подключение датчика* CO_2 *к Pico*» этой главы.

Шаги для получения данных о качестве воздуха с использованием общедоступных данных следующие:

1. Мы рекомендуем зарегистрировать бесплатную учетную запись на сайте **https://airnowapi.org**. После входа в учетную запись вы можете найти персональный ключ API в правом верхнем углу на вкладке **Web Services**, как показано на рисунке:

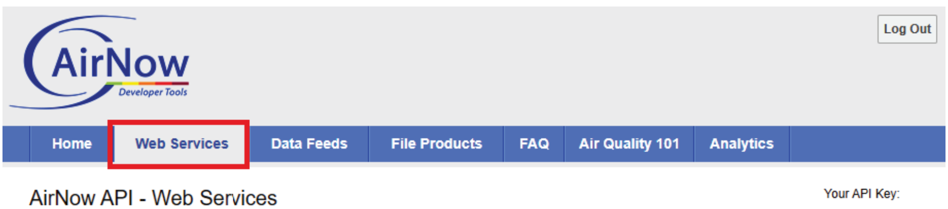


Рисунок 7.4. Расположение ключа API AirNow на вкладке **Web Services**

2. Мы предполагаем, что вы подготовили Pico для настройки беспроводного модуля. Это включает в себя настройку учетных данных Wi-Fi в файле *secrets.py*. Необходимо включить ключ API AirNow в следующем формате:

```
secrets = {
  'ssid' : 'SSID',
  'password' : 'password',
  'timezone' : "America/New_York",
  'api_key' : "ABCDEFGHIJK12345"
}
```

3. Сохраненные учетные данные мы будем использовать для получения местных данных о качестве воздуха. Теперь вернемся к **https://airnowapi.org**. Мы собираемся протестировать API с помощью инструментов, доступных на сайте, прежде чем формировать код, необходимый для получения данных.

[57] Узнать об источниках данных по загрязнению воздуха в России можно на сайте **https://www.iqair.com/ru/russia**. – *Прим. перев.*

4. В вашем аккаунте AirNow перейдите в раздел **Forecasts** (*Прогнозы*) на вкладке **Web Services**, как показано на следующем рисунке:

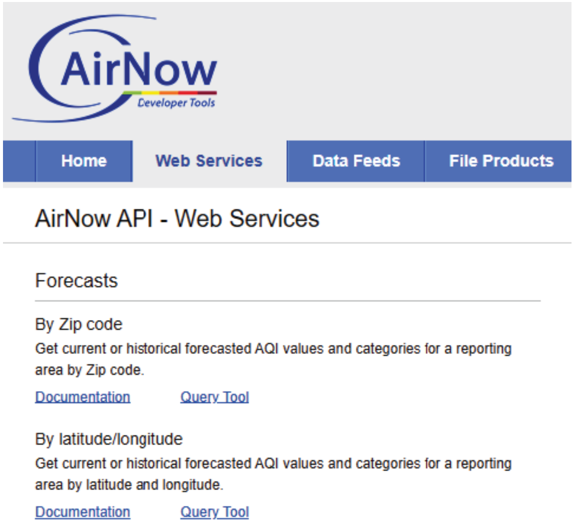


Рисунок 7.5. Раздел **Forecasts** на вкладке **Web Services**

5. URL-адрес, который будет использоваться в коде, запущенном на Pico, мы получим через механизм запроса (пункт **Query Tool**, показанный на рисунке выше). Следует выбрать почтовый индекс (**Zip code**)[58] и формат, в котором API будет посылать данные. В этом случае мы собираемся использовать формат `json`.
6. На следующем рисунке показаны параметры конструктора запросов:

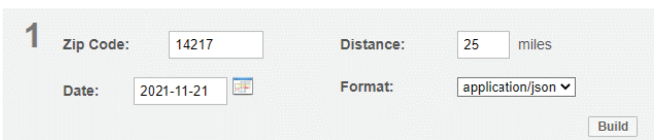


Рисунок 7.6. Параметры конструктора запросов

7. Нажатие на кнопку **Build** (*Построить*) генерирует URL-адрес, как показано на следующем рисунке:

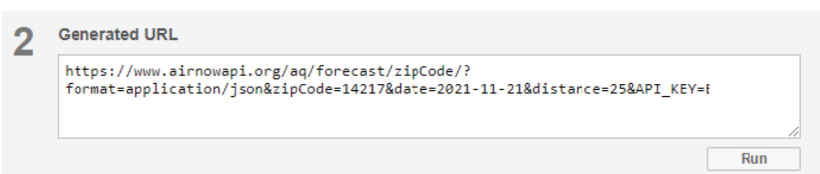


Рисунок 7.7. Результат запроса

[58] Почтовые индексы в США часто используются для идентификации определенной территории наряду с ее названием или географическими координатами. – *Прим. перев.*

8. Сгенерированный URL-адрес должен выглядеть примерно так:

```
https://www.airnowapi.org/aq/forecast/zipCode/?format=application/
json&zipCode=14217&date=2021-11-21&distance=25&API_KEY=ABCEDFGHIJKL12356
```

9. Скопируйте URL-адрес, поскольку он понадобится для нашего кода. Теперь нажмите **Run** (*Выполнить*), и вы увидите данные о качестве воздуха для заданного почтового индекса:

3 **Output**


```
[{"DateIssue":"2021-11-19 ","DateForecast":"2021-11-21 ","ReportingArea":"Western
New York
Region","StateCode":"NY","Latitude":42.6595,"Longitude":-79.0628,"ParameterName":"PM
2.5","AQI":39,"Category":
{"Number":1,"Name":"Good"},"ActionDay":false,"Discussion":""},{"DateIssue":"2021-11-
19 ","DateForecast":"2021-11-22 ","ReportingArea":"Western New York
Region","StateCode":"NY","Latitude":42.6595,"Longitude":-79.0628,"ParameterName":"PM
2.5","AQI":34,"Category":
{"Number":1,"Name":"Good"},"ActionDay":false,"Discussion":""}]
```


Рисунок 7.8. Данные о качестве воздуха

10. Давайте подробнее рассмотрим ответ:

```
{"DateIssue":"2021-11-19 ","DateForecast":"2021-11-21",
"ReportingArea":"Western New York Region","StateCode":"NY","Latitude":42.6595,
"Longitude":-79.0628,"ParameterName":"PM2.5","AQI":39,"Category":{"Number":1,
"Name":"Good"},"ActionDay":false,"Discussion":""}
```

В предыдущем фрагменте ответа мы видим, что AQI равен 39 (выделено в ответе), а категория Good (*Хорошая*). Категория относится к одной из шести градаций качества воздуха (показаны на следующем рисунке). Категории помогают понять, как измеренное качество воздуха связано со здоровьем населения.

AQI Numbers	AQI Category (Descriptor)	AQI Color	Hexadecimal Color Value	Category Number
0 - 50	Good	Green	(00e400)	1
51 - 100	Moderate	Yellow	(ffff00)	2
101 - 150	Unhealthy for Sensitive Groups	Orange	(ff7e00)	3
151 - 200	Unhealthy	Red	(ff0000)	4
201 - 300	Very Unhealthy	Purple	(8f3f97)	5
301 - 500	Hazardous	Maroon	(7e0023)	6

Рисунок 7.9. Категории качества воздуха. Источник: **airnowapi.org**

В ответе на запрос указано, что качество воздуха хорошее. При выполнении того же запроса на Pico следует извлечь категорию качества воздуха и использовать ее для приведения в действие шагового двигателя, указывающего на соответствующую категорию. Такое наглядное представление может быть использовано для информирования людей о качестве воздуха в окружающей среде.

Мы рекомендуем попробовать разные почтовые индексы на континентальной части Соединенных Штатов, чтобы проверить качество воздуха в разных местах. Агентство по охране окружающей среды США дает отдельные рекомендации в районах, подверженных лесным пожарам.

Теперь давайте обсудим выполнение запроса на Pico.

Выполнение запроса на Pico

В этом разделе мы обсудим код, необходимый для выполнения запроса на Pico с использованием беспроводного модуля ESP32. Убедитесь, что вы установили необходимые библиотеки, описанные ранее в этой главе. В этом примере мы собираемся сделать выполнение кода интерактивным, когда принимаются вводимые пользователем данные для местных почтовых индексов и дат. Код доступен для скачивания вместе с этой главой в виде файла *code_airnow.py*.

1. Первым шагом является импорт необходимых модулей:

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
```

2. Следующий шаг – импортировать учетные данные Wi-Fi из *secrets.py*:

```
try:
  from secrets import secrets
except ImportError:
  print("WiFi secrets are kept in secrets.py, please add them there!")
raise
```

3. Далее мы объявляем переменную `API_URL` без почтового индекса, даты и ключа API. Скопируйте URL-адрес из предыдущего раздела и сохраните его в следующем формате:

```
API_URL  = "https://www.airnowapi.org/aq/forecast/zipCode/?format=application/
json&zipCode={0}&date={1}&distance=25&API_KEY={2}"
```

4. Мы также загрузим ключ API, хранящийся в *secrets.py*, следующим образом:

```
API_KEY = secrets["api_key"]
```

5. Следующий шаг – объявление контактов, необходимых для беспроводного модуля, и инициализация последовательного периферийного интерфейса (SPI):

```
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)
spi = busio.SPI(board.GP18, board.GP19, board.GP16)
```

6. Затем инициализируем модуль беспроводной связи ESP32:

```
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
requests.set_socket(socket, esp)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
  print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for I in esp.MAC_address])
```

7. Теперь подключаемся к беспроводной сети, используя учетные данные, сохраненные в *secrets.py*:

```
print("Connecting to AP...")
while not esp.is_connected:
  try:
    esp.connect_AP(secrets["ssid"], secrets["password"])
  except RuntimeError as e:
    print("could not connect to AP, retrying: ", e)
    continue

print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
```

8. Как только беспроводной модуль подключится к интернету, пользователю необходимо будет ввести данные для почтового индекса и даты:

```
zipcode = input("Enter a valid 5-digit zipcode: ")
date = input("Enter today's date in the following format YYYY-MM-DD: ")
```

9. С помощью полученных входных данных, зная `API_KEY`, мы можем отформатировать наш `API_URL`:

```
full_url = API_URL.format(zipcode, date, API_KEY)
```

10. Теперь можем получить местные данные о качестве воздуха, используя конструкцию `try-except-else`. Если запрос `GET` прошел успешно, выполняется код в разделе `else`. Какие-либо ошибки зафиксируются блоком `except`:

```
try:
  response = requests.get(full_url)
except Exception as e:
  print(e)
else:
  data = response.json()[0]
  print("-" * 40)
  print("Reporting Area: ", data["ReportingArea"])
  print("AQI: ", data["AQI"])
  print("Category: ", data["Category"]["Name"])
```

```
  print("-" * 40)
  response.close()
```

11. В блоке `else` мы получаем данные и из ответа извлекаем следующую информацию: обозначение местности (`Reporting Area`), индекс качества воздуха (`AQI`) и наименование (`Name`) категории качества воздуха (`Category`).

12. Собрав все это вместе, мы имеем следующее (код доступен для скачивания в файле *code_airnow.py*):

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as
socket
from adafruit_esp32spi import adafruit_esp32spi

# Get Wi-Fi details and more from a secrets.py file
try:
  from secrets import secrets
except ImportError:
  print("WiFi secrets are kept in secrets.py, please add them there !")
  raise

API_URL  = "https://www.airnowapi.org/aq/forecast/zipCode/?format=application/
json&zipCode={0}&date={1}&distance=25&API_KEY={2}"
API_KEY = secrets["api_key"]

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)

spi = busio.SPI(board.GP18, board.GP19, board.GP16)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs,
esp32_ready, esp32_reset)
requests.set_socket(socket, esp)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
  print("ESP32 found and in idle mode")

print("Connecting to AP...")
while not esp.is_connected:
  try:
    esp.connect_AP(secrets["ssid"], secrets["password"])
```

```
  except RuntimeError as e:
    print("could not connect to AP, retrying: ", e)
    continue

print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))

zipcode = input("Enter a valid 5-digit zipcode: ")
date = input("Enter today's date in the following format YYYY-MM-DD: ")

full_url = API_URL.format(zipcode, date, API_KEY)

try:
  response = requests.get(full_url)
except Exception as e:
  print(e)
else:
  data = response.json()[0]
  print("-" * 40)
  print("Reporting Area: ", data["ReportingArea"])
  print("AQI: ", data["AQI"])
  print("Category: ", data["Category"]["Name"])
  print("-" * 40)
  response.close()

print("Done!")
```

13. Прежде чем сохранить представленный код как *code.py* на Pico и запустить его, мы должны быть в состоянии предоставить входные данные для программы: почтовый индекс и дату.

```
Enter a valid 5-digit zipcode: 94103
Enter today's date in the following format YYYY-MM-DD: 2021-10-23
```

Рисунок 7.10. Предоставление пользовательских входных данных для программы AirNow API

14. Почтовый индекс, указанный здесь в качестве входных данных, принадлежит городу Сан-Франциско. Мы должны увидеть прогноз индекса AQI, как показано на следующем рисунке:

```
----------------------------------------
Reporting Area:  San Francisco
AQI:  42
Category:  Good
----------------------------------------
```

Рисунок 7.11. Прогноз индекса качества воздуха для Сан-Франциско на 23 ноября 2021 года

15. Мы провели проверку качества воздуха для различных почтовых индексов. Например, прогноз качества воздуха в тот же день в Буффало, штат Нью-Йорк, был следующим:

```
---------------------------------------
Reporting Area:  Western New York Region
AQI:  29
Category:  Good
---------------------------------------
```

Рисунок 7.12. Прогноз индекса качества воздуха для Буффало, штат Нью-Йорк, на 23 ноября 2021 года

На предыдущих двух рисунках вы можете заметить существенную разницу в показателях между двумя городами, несмотря на то что оба они находятся в категории `Good` (хорошо). Этот инструмент может быть использован для информирования людей о качестве воздуха в их местности.

Изначально мы создавали данный проект для выставки Maker Faire, где использовали клавиатуру, позволяющую участнику ввести свой почтовый индекс, чтобы узнать о качестве воздуха в его местности.

Мы также продемонстрировали плохие прогнозы, показав почтовые индексы, где прогнозы качества воздуха сопровождались предупреждениями. Синоптики используют эту информацию, чтобы предупредить людей при аллергии в весенний сезон.

В этом примере код выполняется только один раз для каждого пользовательского ввода. Это связано с тем, что мы хотели, чтобы выполнение программы было интерактивным. Если этот проект будет использоваться в стационарной установке, мы единожды закодируем почтовый индекс и будем регулярно получать данные о качестве воздуха в нужной местности. В этом случае нам также необходимо получать дату и время в локальной сети с помощью беспроводного модуля. Мы продемонстрировали это в примере кода *code_airnow_loop.py* (доступно для скачивания вместе с данной главой).

Скорость вызовов API AirNow для определения качества воздуха по почтовому индексу ограничена 500 вызовами в час. Вы должны быть осторожны при использовании API, поскольку это общедоступный сервис. Теперь, когда мы можем получать данные о качестве воздуха, полученные данные можно использовать для создания показывающего устройства, о котором говорилось в начале главы. Однако сначала поработаем над подключением датчика CO_2 к Pico.

Подключение датчика CO_2 к Pico

В этом разделе мы обсудим подключение датчика CO_2 к Raspberry Pi Pico. Этот раздел предназначен для тех, у кого нет доступа к местному источнику данных о качестве воздуха. Датчик SCD30 имеет диапазон измерений 400–10 000 промилле. Мы выбрали этот датчик, потому что он поставляется со встроенным датчиком температуры, который обеспечивает температурную компенсацию при расчете концентрации CO_2. Спецификация датчика доступна здесь:

https://bit.ly/3tyJZ9C[59]. Мы измерим концентрацию CO_2 и опубликуем ее через **ThingSpeak** – сервис, который обсуждали в главе 4 «*Весело проводите время в саду!*». Это позволит нам предоставить широкий доступ к информации о местных уровнях качества воздуха.

Если у вас есть доступ к общедоступному источнику данных, вы можете пропустить этот раздел и перейти к следующему разделу, где мы тестируем двигатели.

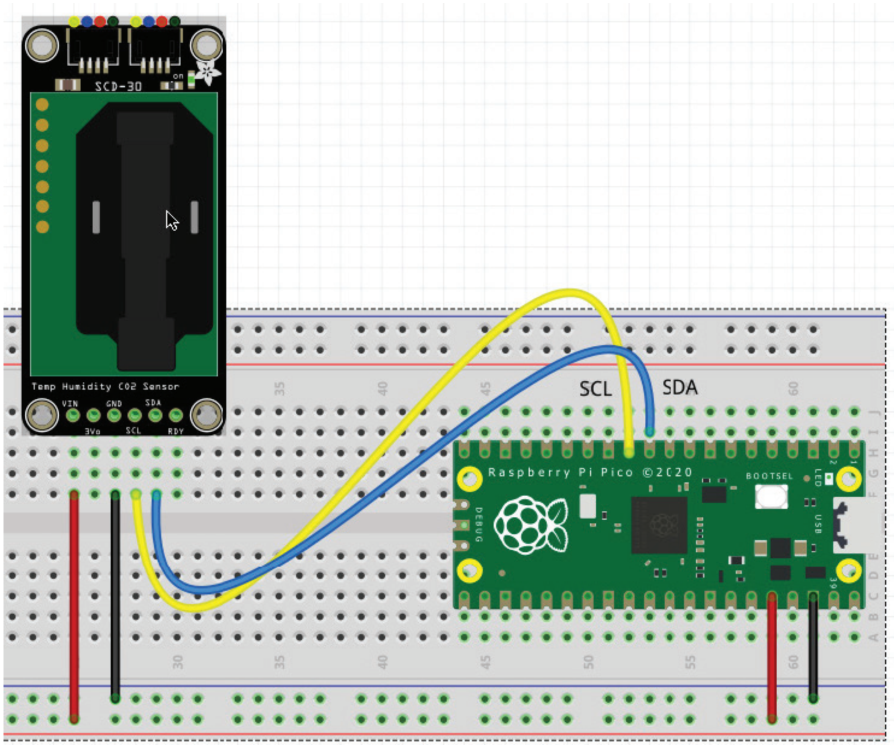


Рисунок 7.13. Подключение Pico к датчику CO_2 SCD30

Мы предполагаем, что вы установили необходимые библиотеки для взаимодействия с датчиком, используя инструкции, приведенные ранее в этой главе. Датчик углекислого газа имеет интерфейс I2C и соединен с Pico, как показано на рис. 7.13. Наименования контактов слева относятся к Pico, в то время как стрелка указывает на контакт датчика CO_2:

- GP9 → SCL;
- GP8 → SDA;
- 3.3V → VIN;
- выводы GND соединены между собой.

[59] Документация от компании Sensirion, производителя датчиков SCD30, доступна по следующему адресу: **https://sensirion.com/media/documents/4EAF6AF8/61652C3C/Sensirion_CO2_Sensors_SCD30_Datasheet.pdf**. – *Прим. перев.*

Подтягивающие резисторы

Плата датчика SCD30 поставляется с подтягивающими резисторами, необходимыми для интерфейса I2C.

На следующем рисунке показана схема подключения датчика CO_2 к Raspberry Pi Pico.

Теперь давайте рассмотрим код, необходимый для взаимодействия с датчиком. Пример кода, обсуждаемый здесь, доступен для загрузки вместе с этой главой в виде файла *code_scd30.py*.

1. Первым шагом является импорт необходимых модулей:

```
import board
import busio
import adafruit_scd30
```

2. Далее мы инициализируем интерфейс I2C:

```
i2c = busio.I2C(board.GP9, board.GP8)
scd = adafruit_scd30.SCD30(i2c)
```

3. Теперь войдем в цикл `while`, где получим концентрацию CO_2, если доступен результат очередного измерения:

```
while True:
  if scd.data_available:
    print("Data Available!")
    print("CO2: %d PPM" % scd.CO2)
    print("Temperature: %0.2f degrees C" % scd.temperature)
    print("Humidity: %0.2f %% rH" % scd.relative_humidity)
```

4. Собрав все это вместе, мы имеем следующий код:

```
import time
import board
import busio
import adafruit_scd30

i2c = busio.I2C(board.GP9, board.GP8)
scd = adafruit_scd30.SCD30(i2c)

while True:
  if scd.data_available:
    print("Data Available!")
    print("CO2: %d PPM" % scd.CO2)
    print("Temperature: %0.2f degrees C" % scd.temperature)
    print("Humidity: %0.2f %% rH" % scd.relative_humidity)
  time.sleep(2.0)
```

5. Сохранив пример кода как *code.py* на Pico, мы должны получить следующий результат:

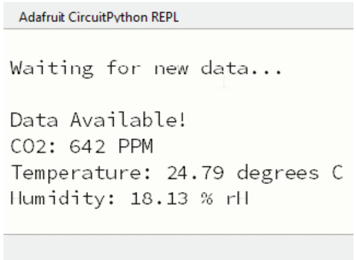


Рисунок 7.14. Вывод результатов с датчика CO_2

> **Простота создания макетов и прототипов**
>
> Обратите внимание на простоту обращения к датчикам и считывания данных с помощью CircuitPython. Это облегчает создание наших проектов.

Теперь мы обсудим размещение данных датчиков с помощью **ThingSpeak** – платформы, которую мы использовали ранее, в главе 4 «*Весело проводите время в саду!*». Мы рекомендуем вам создать бесплатную учетную запись на ThingSpeak (если вы еще этого не сделали).

Давайте рассмотрим шаги, связанные с публикацией данных в ThingSpeak.

1. Первый шаг – создать новый канал на ThingSpeak (как показано на следующем рисунке). Вы можете создать не более четырех каналов, используя бесплатную учетную запись.

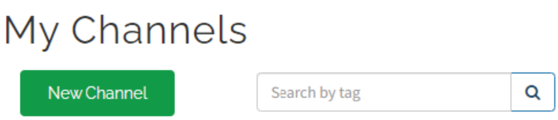


Рисунок 7.15. Создание нового канала на ThingSpeak

2. Создайте поле с названием `CO2 concentration` (концентрация CO_2), как показано на рис. 7.16:

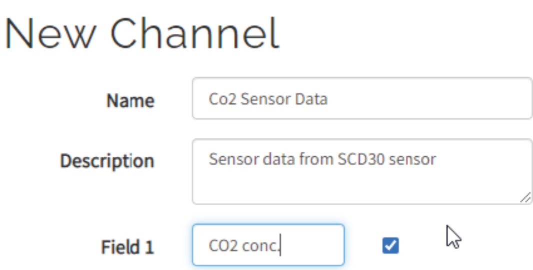


Рисунок 7.16. Создание нового канала

3. После создания канала скопируйте ключ записи **API key** и URL-адрес из меню **Write API key**, доступного на вкладке **API Keys** вашего канала.

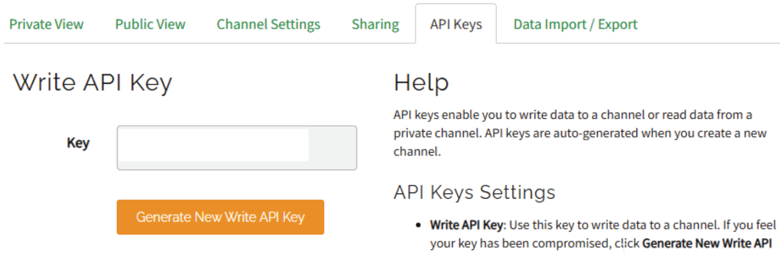


Рисунок 7.17. Скопируйте ключ API для записи

4. Кроме того, скопируйте URL-адрес, необходимый для публикации данных датчика. Он доступен на той же вкладке:

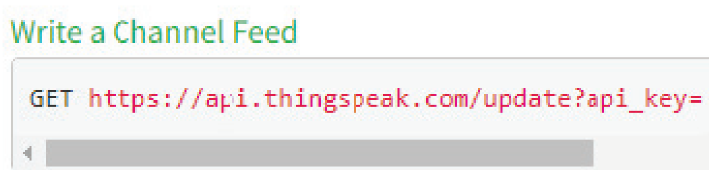


Рисунок 7.18. Скопируйте канал URL-адреса записи

5. Теперь мы изменим *secrets.py*, чтобы включить ключ API ThingSpeak:

```
secrets = {
  'ssid' : 'SSID',
  'password' : 'password',
  'timezone' : "America/New_York",
  'api_key' : "ABCDEFGHIJK12345"
}
```

> **Ключ API**
>
> Мы предполагаем, что вы следуете этому примеру из-за отсутствия местных источников данных. Поэтому в поле `api_key` мы храним ключ API записи.

6. Теперь рассмотрим код, необходимый для публикации данных в ThingSpeak. Пример этого кода доступен для загрузки вместе с этой главой в файле *code_scd_thingspeak.py*:

```
import board
import busio
import time
import adafruit_scd30
from digitalio import DigitalInOut
```

```
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
```

7. Следующий шаг – импортировать учетные данные Wi-Fi из *secrets.py*:

```
try:
  from secrets import secrets
except ImportError:
  print("WiFi secrets are kept in secrets.py, please add them there !")
  raise
```

8. Далее мы объявляем переменную API_URL без почтового индекса, даты или ключа API. Скопируйте URL-адрес из раздела «*Подключение датчика CO_2 к Pico*» и сохраните его в следующем формате:

```
API_URL = "https://api.thingspeak.com/update?api_key={0}&field1={1}"
```

9. Ключ API, хранящийся в *secrets.py*, загрузим таким образом:

```
API_KEY = secrets["api_key"]
```

10. Следующий шаг – объявление контактов, необходимых для беспроводного модуля, и инициализация SPI:

```
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)
spi = busio.SPI(board.GP18, board.GP19, board.GP16)
```

11. Затем мы инициализируем беспроводной модуль ESP32:

```
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32 _reset)
requests.set_socket(socket, esp)
```

12. Потом инициализируем интерфейс I2C и датчик SCD30:

```
i2c = busio.I2C(board.GP9, board.GP8)
scd = adafruit_scd30.SCD30(i2c)
```

13. Далее вводим цикл `while`, в котором мы подключаемся к беспроводной сети:

```
while True:
  while not esp.is_connected:
    try:
      print("Connecting to AP...")
      esp.connect_AP(secrets["ssid"], secrets["password"])
    except RuntimeError as e:
      print("could not connect to AP, retrying: ", e)
```

```
      continue
    else:
      print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi )
      print("My IP address is", esp.pretty_ip(esp.ip_address))
```

14. Затем, по мере готовности датчика, считываем концентрацию CO_2:

```
  if  scd.data_available:
    print("Data Available!")
    print("CO2: %d PPM" % scd.CO2)
    print("Temperature: %0.2f degrees C" % scd.temperature)
    print("Humidity: %0.2f %% rH" % scd.relative_humidity)
```

15. Потом публикуем данные в ThingSpeak:

```
    full_url = API_URL.format(API_KEY, str(scd.CO2))
    try:
      response = requests.get(full_url)
    except Exception as e:
      print(e)
    else:
      print("-" * 40)
      print(response.json())
      print("-" * 40)
      response.close()
      print("Done!")
  time.sleep(20)
```

16. В итоге мы имеем следующий код:

```
import board
import busio
import time
import adafruit_scd30
from digitalio import DigitalInOut
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi

# Get wifi details and more from a secrets.py file
try:
  from secrets import secrets
except ImportError:
  print("WiFi secrets are kept in secrets.py, please add them there !")
  raise

API_URL = "https://api.thingspeak.com/update?api_key={0}&field1={1}"
API_KEY = secrets["api_key"]
```

```
# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.GP7)
esp32_ready = DigitalInOut(board.GP10)
esp32_reset = DigitalInOut(board.GP11)

spi = busio.SPI(board.GP18, board.GP19, board.GP16)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready,  esp32_reset)
requests.set_socket(socket, esp)

i2c = busio.I2C(board.GP9, board.GP8)
scd = adafruit_scd30.SCD30(i2c)

while True:
  while not esp.is_connected:
    try:
      print("Connecting to AP...")
      esp.connect_AP(secrets["ssid"], secrets["password"])
    except RuntimeError as e:
      print("could not connect to AP, retrying: ", e)
      continue
    else:
      print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp .rssi)
      print("My IP address is", esp.pretty_ip(esp.ip_address))

  if scd.data_available:
    print("Data Available!")
    print("CO2: %d PPM" % scd.CO2)
    print("Temperature: %0.2f degrees C" % scd.temperature)
    print("Humidity: %0.2f %% rH" % scd.relative_humidity)
    full_url = API_URL.format(API_KEY, str(scd.CO2))

    try:
      response = requests.get(full_url)
      except Exception as e:
      print(e)
    else:
      print("-" * 40)
      print(response.json())
      print("-" * 40)
      response.close()
      print("Done!")
  time.sleep(20)
```

17. После того как код будет сохранен в файле *code.py* на Pico и запущен, мы должны увидеть концентрацию CO_2 и подтверждение публикации

данных в ThingSpeak. Мы должны получить результат, аналогичный показанному на рисунке:

```
Data Available!
CO2: 634 PPM
Temperature: 27.86 degrees C
Humidity: 23.46 % rH
-----------------------------
339
-----------------------------
Done!
```

Рисунок 7.19. Данные о температуре и влажности, опубликованные в ThingSpeak

18. Мы также должны иметь возможность видеть тенденции концентрации CO_2 с течением времени, как показано на следующем рисунке:

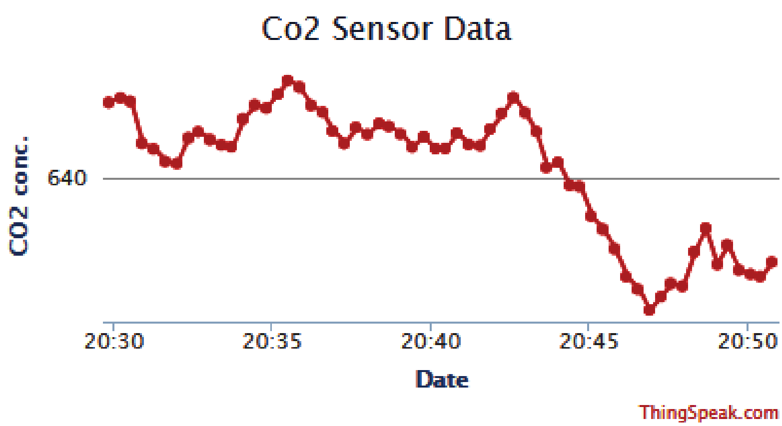


Рисунок 7.20. Публикация данных о концентрации CO_2 в ThingSpeak

Мы должны иметь возможность делиться этими данными в общедоступной информационной панели. Аналогичным образом мы можем добавить датчики летучих органических соединений (VOC), твердых частиц (PM) и другие и получить панель мониторинга, которая демонстрирует все эти параметры на большом экране.

В следующем разделе мы обсудим управление шаговым двигателем для нашего выставочного экземпляра.

Подключение шагового двигателя

В этом разделе мы обсудим подключение шагового двигателя к Raspberry Pi Pico с помощью драйвера шагового двигателя DRV8833. Это позволит нам создать дисплей, в котором шаговый двигатель будет указывать на соответствующую категорию качества воздуха.

Выбор двигателя

Для этого проекта мы выбрали шаговый двигатель, который используется в автомобильных датчиках. Это связано с тем, что общий диапазон перемещения такого двигателя составляет 315°. Он поставляется со стрелкой, которая полезна при создании нашего наглядного пособия. Выбор двигателя для данной цели включает в себя несколько факторов. Мы рекомендуем следующую статью из SparkFun: **https://bit.ly/3qyYxnX**[60].

Первый шаг – подключить драйвер шагового двигателя к Raspberry Pi Pico. На рис. 7.21 показаны соединения драйвера шагового двигателя с Pico. Далее слева от стрелки показан вывод Raspberry Pi Pico, справа – вывод драйвера шагового двигателя DRV8833:

- 3.3V → VM, SLP;
- GP12 → AIN1;
- GP13 → AIN2;
- GP14 → BIN1;
- GP15 → BIN2;
- GND → GND.

На следующем рисунке показана схема подключения драйвера шагового двигателя к Raspberry Pi Pico:

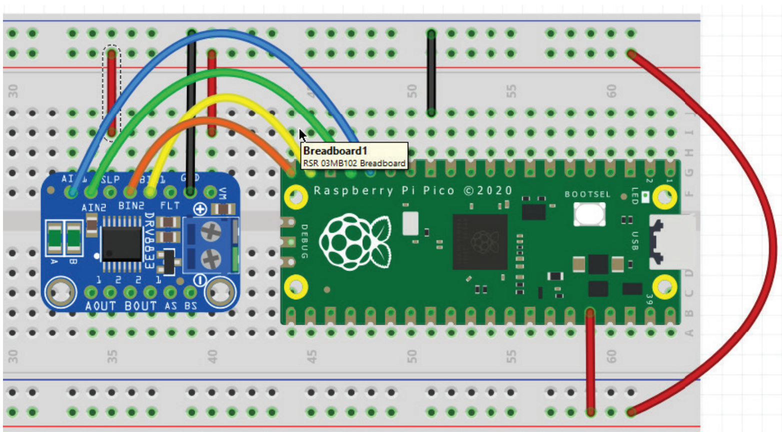


Рисунок 7.21. Подключение DRV8833 к Raspberry Pi Pico

Шаговый двигатель имеет две пары выводов обмоток. Первая пара подключена к выводам драйвера AOUT, вторая – к BOUT. В случае если вы незнакомы с шаговыми двигателями, мы рекомендуем ознакомиться с этой статьей: **https://bit.ly/3A6mfuN**[61].

Соединения драйвера с шаговым двигателем на макетной плате показаны на следующем рисунке:

---

[60] О выборе двигателя см. также сноску 54 на стр. 151. – *Прим. перев.*

[61] Для начального ознакомления с устройством шагового двигателя рекомендуется, например, следующая статья на русском языке: **http://electrik.info/motors/1432-arduino-i-shagovyy-dvigatel.html**. – *Прим. перев.*

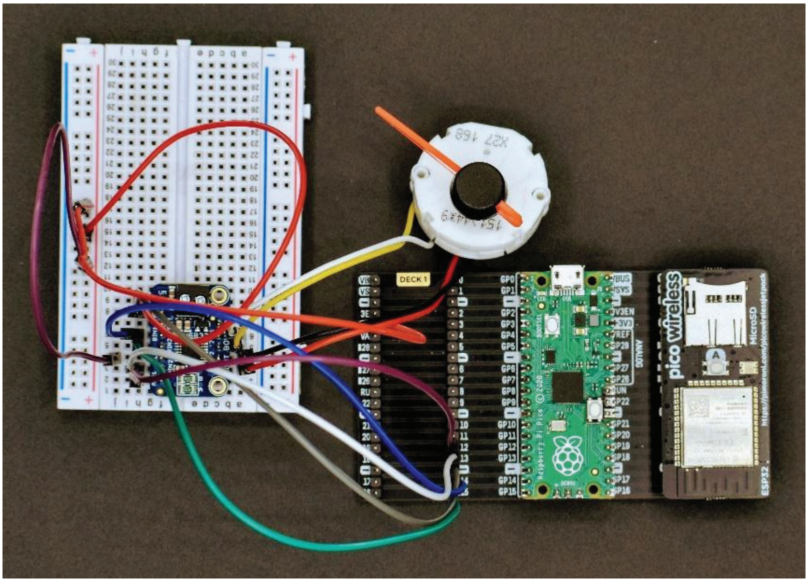


Рисунок 7.22. Подключение шагового двигателя к драйверу DRV8833

Теперь давайте рассмотрим код, необходимый для приведения в действие шагового двигателя. Код, обсуждаемый в этом разделе, доступен для загрузки вместе с этой главой в файле *code_drv8833.py*. Мы предполагаем, что вы установили необходимые для шагового двигателя драйверы, описанные ранее в этой главе.

1. Первым шагом является импорт необходимых модулей:

```
import time
import board
import digitalio
from adafruit_motor import stepper
```

2. Следующий шаг – определить контакты, используемые для управления шаговым двигателем, и установить их в качестве выходных:

```
coils = (
  digitalio.DigitalInOut(board.GP12), # A1
  digitalio.DigitalInOut(board.GP13), # A2
  digitalio.DigitalInOut(board.GP14), # B1
  digitalio.DigitalInOut(board.GP15), # B2
)
for coil in coils:
  coil.direction = digitalio.Direction.OUTPUT
```

3. Инициализируем класс `StepperMotor` из библиотеки драйверов шаговых двигателей:

```
motor = stepper.StepperMotor(coils[0], coils[1], coils[2], coils[3], microsteps=None)
```

4. Теперь проворачиваем шаговый двигатель вперед и назад на 400 шагов в каждую сторону с задержкой 10 мс после каждого шага:

```
 DELAY = 0.01
 STEPS = 400
 for step in range(STEPS):
 motor.onestep()
 time.sleep(DELAY)

 for step in range(STEPS):
   motor.onestep(direction=stepper.BACKWARD)
   time.sleep(DELAY)
```

5. В конце программы мы освобождаем контакты GPIO, используемые для управления шаговым двигателем, вызывая функцию `motor.release()`.

6. Собрав все это вместе, имеем следующее:

```
import time
import board
import digitalio
from adafruit_motor import stepper

DELAY = 0.01
STEPS = 400

coils = (
 digitalio.DigitalInOut(board.GP12), # A1
 digitalio.DigitalInOut(board.GP13), # A2
 digitalio.DigitalInOut(board.GP14), # B1
 digitalio.DigitalInOut(board.GP15), # B2
)

for coil in coils:
 coil.direction = digitalio.Direction.OUTPUT

 motor = stepper.StepperMotor(coils[0], coils[1], coils[2], coils[3], mi crosteps=None)

for step in range(STEPS):
 motor.onestep()
 time.sleep(DELAY)

for step in range(STEPS):
motor.onestep(direction=stepper.BACKWARD)
time.sleep(DELAY)

motor.release()
```

Если соединения между драйвером шагового двигателя и Pico, а также между драйвером и самим шаговым двигателем правильные, мы должны увидеть, как стрелка на валу шагового двигателя перемещается вперед и назад.

В следующем разделе мы обсудим сборку всего дисплея, чтобы можно было использовать стрелку для указания категории качества воздуха.

Устройство дисплея

Пришло время собрать дисплей внутри корпуса. Мы напечатали различные категории качества воздуха, расположив их полукругом, и проверили размещение в корпусе, как показано на рис. 7.23. О конструкции корпуса см. главу 6 «*Проектируем настенный семисегментный дисплей*».

Рисунок 7.23. Подгонка шкалы внутри корпуса

Шаговый двигатель располагаем в середине корпуса поверх шкалы качества воздуха[62]. После установки шагового двигателя пришло время откалибровать положения шагового двигателя и рассчитать количество шагов, необходимых для установки стрелки на каждой категории.

На рис. 7.24 показан дисплей, указывающий на хорошее качество воздуха (категория *Good*).

В следующем разделе мы обсудим, что делать с интерактивным дисплеем.

Создание интерактивного дисплея

Если вы планируете использовать его в выставочных целях, то можете сделать его интерактивным, добавив к дисплею клавиатуру (ссылка: **https://www.sparkfun.**

[62] Несомненно, читатель догадается, что лучше разместить напечатанную шкалу не на задней стенке (см. следующий рисунок), а поверх шагового двигателя таким образом, чтобы вал «протыкал» шкалу и видна была только стрелка. – *Прим. перев.*

com/products/15290)[63]. Вы можете попросить посетителей ввести свой почтовый индекс и узнать больше о качестве воздуха в их окружении. Вы также можете получить некоторые прогнозы качества воздуха в день выставки и поговорить о том, как это влияет на людей, живущих в этой местности. Например, при плохом качестве воздуха местные власти могут ввести временные ограничения на мероприятия на свежем воздухе. Это может стать отличным инструментом для мотивации людей к электронике и гражданской науке.

Рисунок 7.24. Готовый дисплей

Итоги

В этой главе мы обсуждали получение данных о качестве воздуха из общедоступных источников данных. Мы также обсудили возможность подключения датчика CO_2 к Raspberry Pi Pico для отслеживания качества воздуха самостоятельно. Затем разобрали управление шаговым двигателем с помощью Pico, который можно использовать для указания на одну из шести различных категорий качества воздуха. Это позволяет нам создавать дисплей, который отслеживает качество воздуха в нашем окружении.

В следующей главе мы собираемся весело провести время с радиоприемниками! Присоединяйтесь к нам в этом приключении.

[63] Указанную 12-клавишную клавиатуру (кнопочный номеронабиратель, тастатура) с I2C-интерфейсом в российских условиях можно приобрести в «Чипе и Дипе» (**https://www.chipdip.ru/product1/8004461027**). – *Прим. перев.*

Часть III

Темы повышенной сложности

В этой части мы приступим к обсуждению более сложных тем с использованием Raspberry Pi Pico. С учетом опыта, накопленного в предыдущих частях, мы собираемся обсудить различные темы, начиная от сопряжения беспроводных модулей и заканчивая созданием собственного продукта с использованием Pico.

Часть III содержит следующие главы:

- глава 8 «*Беспроводная связь*»;
- глава 9 «*Строим робота!*»;
- глава 10 «*Знакомство с приложениями TinyML*»;
- глава 11 «*Создаем готовый продукт*»;
- глава 12 «*Дополнительные возможности работы с Pico*».

Глава 8

Беспроводная связь

В этой главе мы обсудим организацию беспроводной связи с помощью Raspberry Pi Pico. До сих пор мы обсуждали публикацию данных датчиков с помощью модуля Wi-Fi. Что делать, если мы хотим установить проект в месте, где у нас нет доступа к сети Wi-Fi? В этой главе разберем различные варианты выхода из подобной ситуации.

Мы рассмотрим следующие темы:

- установка необходимых библиотек;
- подключение модуля Bluetooth с низким энергопотреблением (Bluetooth LE);
- работа с модулем Sigfox;
- работа с модулем LoRa.

> **Сотовые модули**
>
> Мы будем обсуждать подключение сотовых модулей к Raspberry Pi Pico в главе 11 «*Создаем готовый продукт*», где рассматривается создание законченного устройства.

Технические требования

Для этой главы рекомендуется использовать следующее оборудование. Компоненты следует приобретать в соответствии с типом беспроводной связи, который вы будете использовать в своем проекте:

- Raspberry Pi Pico (**https://bit.ly/3AJtoAf**);
- дополнительно: Adafruit Bluefruit LE SPI Friend (**https://bit.ly/3cXjukF**)[64];
- дополнительно: плата приемопередатчика Adafruit RFM95W (**https://bit.ly/3y2UDWo**)[65];

---

[64] В России Adafruit Bluefruit LE SPI Friend можно заказать через многие интернет-магазины, например **https://zener.ru/shop/product/2633/**. – *Прим перев.*

[65] Платы LoRa оригинального производителя (фирмы HOPE) на основе модуля RFM95W доступны, например, в «Чипе и Дипе» (см. **https://www.chipdip.ru/product1/8002728468**), а также встречаются во многих других отечественных интернет-магазинах. В тексте далее идет речь о модулях Adafruit RFM95W для американского ISM-диапазона 915 МГц, для России и Европы следует ориентироваться на 433 и 868 МГц (см. также пояснение в сноске 71 на стр. 192). Отметим, что во всех случаях антенны и разъемы для них приобретаются отдельно (подробнее см. стр. 200). – *Прим. перев.*

- соединительные провода (ссылка: **https://iarduino.ru/shop/Provoda-kabeli/**);
- дополнительно: датчик CO_2 Adafruit SCD-30 (ссылка: **https://bit.ly/3l7MfQ6**)[66];
- беспаечная макетная плата.

Примеры кода для этой главы доступны по адресу: **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIY-Workshop/tree/main/chapter_08**.

Видеоролики с кодом в действии главы можно посмотреть по адресу **https://bit.ly/3Fgslvi**.

> **Установка CircuitPython**
>
> Мы предполагаем, что вы установили CircuitPython на свой Pico. Если вы незнакомы с процессом установки, рекомендуем следовать инструкциям из главы 1 «*Начало работы с Raspberry Pi Pico*».
>
> **Дополнительные компоненты**
>
> Мы написали эту главу в стиле «выбери нужное». Вам потребуются компоненты только для того типа беспроводной радиосвязи, который вы планируете реализовать.

В следующем разделе обсудим установку библиотек, необходимых для трех примеров беспроводной радиосвязи в этой главе.

Установка необходимых библиотек

В этом разделе мы установим библиотеки, необходимые для примеров, обсуждаемых в данной главе, включая модули Bluetooth, LoRa и Sigfox. Все библиотеки являются частью пакета Adafruit CircuitPython. Пакет можно загрузить в виде ZIP-файла с **https://circuitpython.org/libraries**. Мы использовали версию пакета, предназначенную для CircuitPython 6.x.x.

После загрузки ZIP-файла извлеките его содержимое, чтобы мы могли скопировать библиотеки, необходимые для проекта.

Adafruit Bluefruit LE SPI Friend

В остальной части этой главы мы будем ссылаться на Adafruit Bluefruit LE SPI Friend (показан на рис. 8.1) в качестве модуля Bluetooth.

Для модуля Bluetooth необходимы библиотеки *adafruit_bus_device* и *adafruit_bluefruitspi*. Из пакета библиотек скопируйте папку *adafruit_ bus_device* и двоичный файл *adafruit_bluefruitspi.mpy* в папку *lib* на Pico.

Дополнительно: модуль LoRa

Это необязательный шаг. Вам необходимо установить библиотеки, связанные с модулем Adafruit RFM95W (показан на рис. 8.2), только если вы будете использовать беспроводную связь LoRa.

---

[66] См. сноску 56 на стр. 151. – *Прим. перев.*

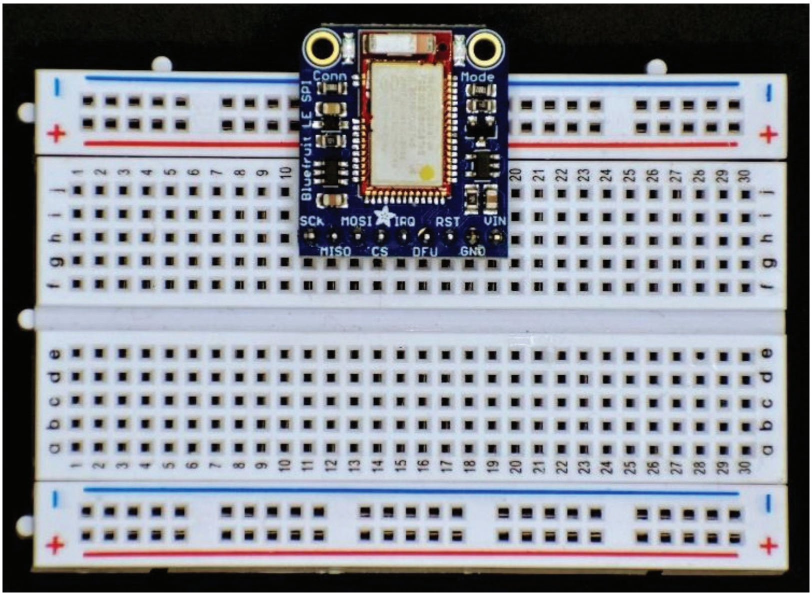


Рисунок 8.1. Adafruit Bluefruit LE SPI Friend

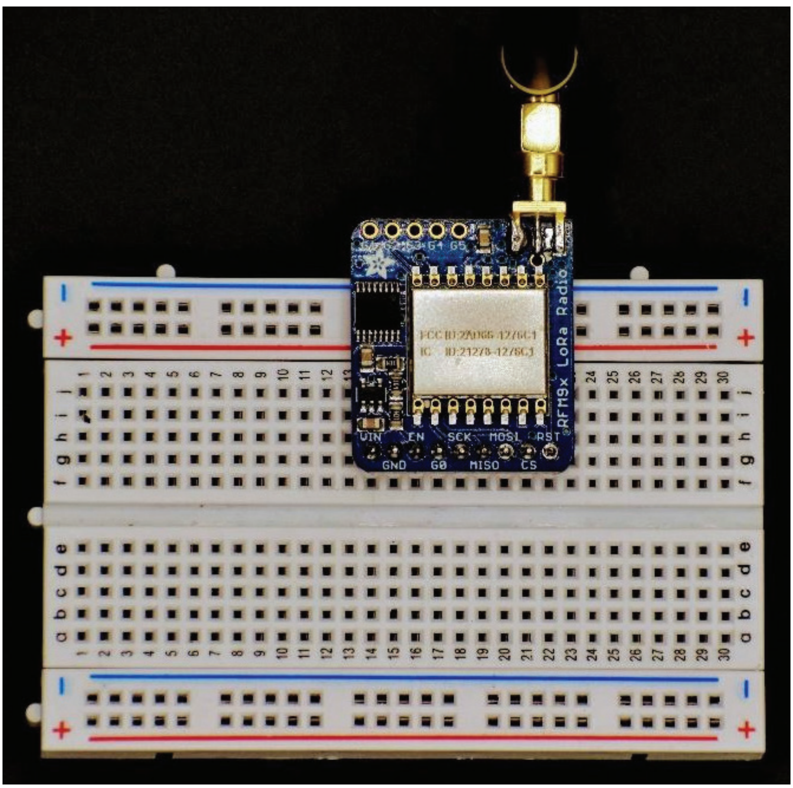


Рисунок. 8.2. Модуль Adafruit RFM95W (LoRa)

Модулю LoRa необходимы библиотеки *adafruit_bus_device* и *adafruit_rfm9x*. Скопируйте папку *adafruit_bus_device* вместе с двоичным файлом *adafruit_rfm9x* в папку *lib* вашего Pico.

Дополнительно: датчик CO_2

Это необязательный шаг. Вам нужно установить библиотеки для датчика CO_2 только в том случае, если вы собираетесь использовать датчик. Датчику CO_2 (показан на рисунке ниже) необходимы *adafruit_bus_device*, *adafruit_register* и *adafruit_scd30*. Скопируйте папки *adafruit_bus_device* и *adafruit_register* вместе с файлом *adafruit_scd30.mpy* в папку *lib* вашего Pico.

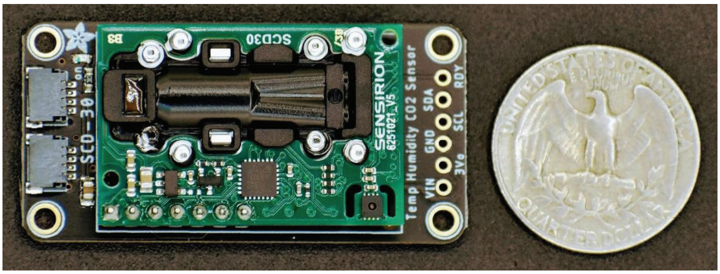


Рисунок 8.3. Датчик CO_2 SCD30

В следующем разделе мы обсудим подключение модуля Bluetooth к Raspberry Pi Pico.

Подключение модуля Bluetooth с низким энергопотреблением

В этом разделе мы обсудим подключение Adafruit Bluefruit LE SPI Friend (далее именуемого модулем Bluetooth) к Raspberry Pi Pico.

В случае если вы незнакомы с технологией Bluetooth Low Energy, это технология беспроводной сети с низким энергопотреблением, предназначенная для беспроводных датчиков и других периферийных устройств, которые могут работать от батарейки-монетки. Вы можете узнать больше о Bluetooth Low Energy (BLE) из руководства по адресу **https://bit.ly/3BkLPNu**[67].

Плата Adafruit Bluefruit LE, используемая в этом разделе, поставляется с модулем Raytac MDBT40 BLE, который сертифицирован FCC и CE, то есть обязательными сертификатами для использования в Соединенных Штатах и Европе соответственно. Модуль разработан на основе чипсета Nordic nRF51822. Сообществом Adafruit реализована прошивка, позволяющая подключать модуль Bluetooth LE в качестве периферийного устройства. Для получения дополнительной информации, включая техническое описание модуля, мы рекомендуем эту статью от Adafruit: **https://bit.ly/3sFaWX4**.

Мы продемонстрируем приложения, в которых можем использовать модуль Bluetooth в качестве периферийного устройства для Pico. Мы будем подклю-

[67] По-русски подробности о технологии Bluetooth LE можно узнать, например, отсюда: **https://habr.com/ru/post/532298/**. – *Прим. перев.*

чать модуль Bluetooth к Pico с помощью интерфейса SPI. В первом примере передадим некоторые тестовые строки через Bluetooth к мобильному устройству. Переданное сообщение передается обратно в модуль, считывается Pico и выводится на экран. Во втором примере мы обсудим отображение данных датчиков в мобильном приложении. Давайте начнем!

Соединения между Pico и модулем Bluetooth следующие (слева от стрелки наименования относятся к контактам Pico, справа – к контактам модуля Bluetooth):

- VBUS → VIN;
- GP0 → IRQ;
- GP1 → RST;
- GP5 → CS;
- GP2 → SCK;
- GP3 → MOSI;
- GP4 → MISO;
- GND соединяются между собой.

Схема подключения модуля Bluetooth к Raspberry Pi Pico показана на следующем рисунке:

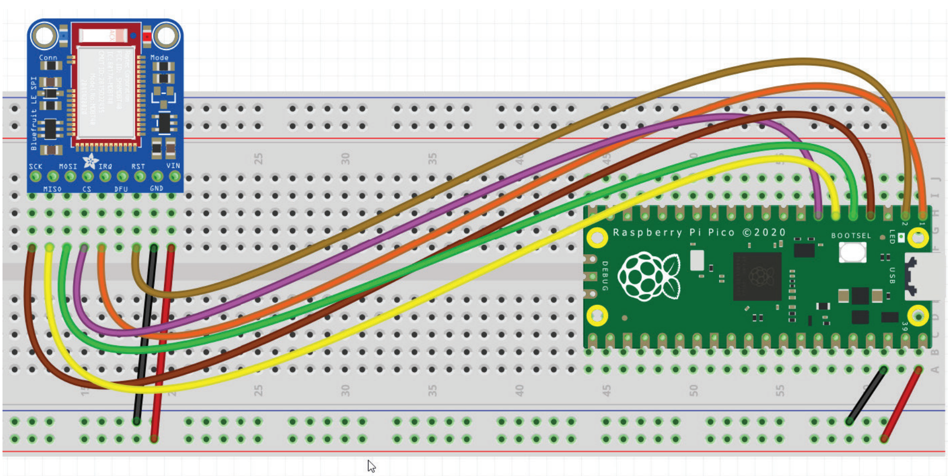


Рисунок 8.4. Подключение модуля Bluetooth к Raspberry Pi Pico

Теперь давайте рассмотрим код, необходимый для взаимодействия с модулем Bluetooth. Пример, приведенный в этом разделе, основан на документации к библиотеке *adafruit_bluefruitspi*. Пример доступен для загрузки в папке *downloads* репозитория GitHub данной главы в виде файла *bluefruit_spi_test.py*. Эти шаги включают в себя следующее:

1. Первым шагом является импорт необходимых модулей для программы:

```
import time
import busio
import board
```

```
from digitalio import DigitalInOut
from adafruit_bluefruitspi import BluefruitSPIr
```

2. Следующий шаг – инициализация интерфейса SPI и настройка модуля Bluetooth:

```
spi_bus = busio.SPI(board.GP2, MOSI=board.GP3,MISO=board.GP4)
cs = DigitalInOut(board.GP5)
irq = DigitalInOut(board.GP0)
rst = DigitalInOut(board.GP1)
bluefruit = BluefruitSPI(spi_bus, cs, irq, rst, debug=False)
```

3. Инициализация модуля Bluetooth производится с помощью AT-команд. Так называется набор команд, используемых для управления модемами (**AT** – сокращение от слова **attention** – внимание). В данном случае AT-команды применяются для управления модулем Bluetooth:

```
print("Initializing the Bluefruit LE SPI Friend module")
bluefruit.init()
bluefruit.command_check_OK(b'AT+FACTORYRESET', delay=1)
```

4. Запрашиваем информацию у модуля Bluetooth:

```
print(str(bluefruit.command_check_OK(b'ATI'), 'utf-8'))
```

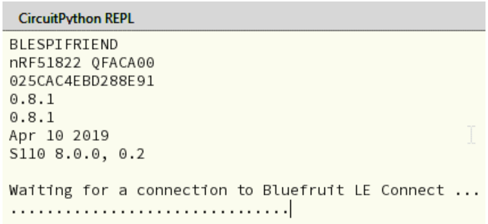
5. Модули Bluetooth нуждаются в указании имени для подключения к устройству. Поскольку мы собираемся просматривать данные на мобильном телефоне, необходимо задать имя, которое легко идентифицировать в мобильном приложении. Мы выбрали имя `PicoBLE`.

```
bluefruit.command_check_OK(b'AT+GAPDEVNAME=PicoBLE')
```

6. Теперь входим в наш основной цикл, где ждем подключения устройства к модулю Bluetooth. Поскольку ожидаем подключения, то будем выводить символ «.» через терминал последовательного порта каждые 500 мс (как показано на рис. 8.5 далее):

```
while True:
  print("Waiting for a connection to Bluefruit LE Connect ...")
  dotcount = 0
  while not bluefruit.connected:
    print(".", end="")
    dotcount = (dotcount + 1) % 80
    if dotcount == 79:
    print("")
    time.sleep(0.5)
```

Это помогает пользователю понять, что на Pico запущена программа:



```
CircuitPython REPL
BLESPIFRIEND
nRF51822 QFACA00
025CAC4EBD288E91
0.8.1
0.8.1
Apr 10 2019
S110 8.0.0, 0.2

Waiting for a connection to Bluefruit LE Connect ...
..............................
```

Рисунок 8.5. Ожидание соединения

7. Когда устройство соединится с модулем Bluetooth, через терминал последовательного порта выводится отладочное сообщение (как показано на рис. 8.6 далее):

```
print("\n *Connected!*")
connection_timestamp = time.monotonic()
while True:
  if time.monotonic() - connection_timestamp > 3:
    connection_timestamp = time.monotonic()
    if not bluefruit.connected:
      break
  resp = bluefruit.uart_rx()
  if not resp:
    continue
```

Здесь мы ждем любых входящих сообщений от подключенного устройства. Программа гарантирует, что мы по-прежнему подключены к соответствующему устройству. Если нет, мы выходим из цикла и ожидаем подключения с другого устройства.

```
code.py output:
Initializing the Bluefruit LE SPI Friend mocule
BLESPIFRIEND
nRF51822 QFACA00
025CAC4E3D288E91
0.8.1
0.8.1
Apr 10 2019
S110 8.0.0, 0.2

Waiting for a connection to Bluefruit LE Cornect ...
.............
 *Connected!*
```

Рисунок 8.6. Модуль Bluetooth подключен к устройству

8. В предыдущем фрагменте кода читаются сообщения с использованием метода `uart_rx()`. Если на модуль Bluetooth поступает входящее сообщение, мы выводим его на терминал последовательного порта и отсылаем сообщение обратно с использованием метода `uart_tx()`:

```
print("Read %d bytes: %s" % (len(resp), resp))
print("Writing reverse...")
send = []
for i in range(len(resp), 0, -1):
```

```
    send.append(resp[i-1])
  print(bytes(send))
  bluefruit.uart_tx(bytes(send))
```

9. После сохранения кода в Pico как *code.py*, подключения модуля Bluetooth и выполнения кода мы сможем увидеть, что модуль Bluetooth инициализирован и ожидает подключения:

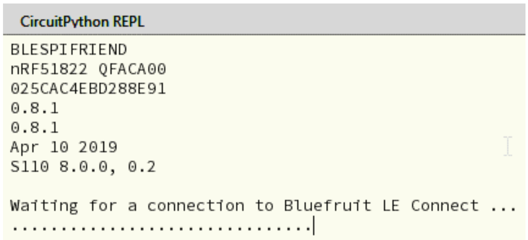


Рисунок 8.7. Модуль Bluetooth инициализирован и ожидает подключения

10. Загрузите приложение **Bluefruit LE Connect** с Adafruit на свой смартфон. Оно доступно как для устройств Android, так и для устройств Apple. Скриншот приложения показан на следующем рисунке:

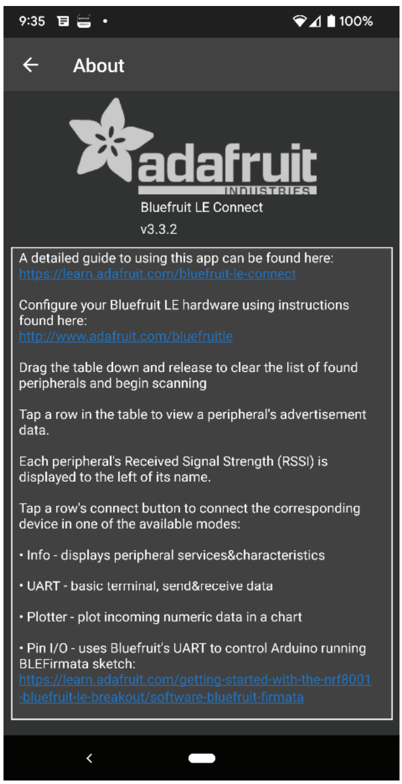


Рисунок 8.8. Приложение Adafruit Bluefruit LE Connect

11. После запуска приложения на своем телефоне вы должны увидеть экран со списком устройств, находящихся поблизости от вас. Среди них определится `PicoBLE`, как показано на следующем рисунке:

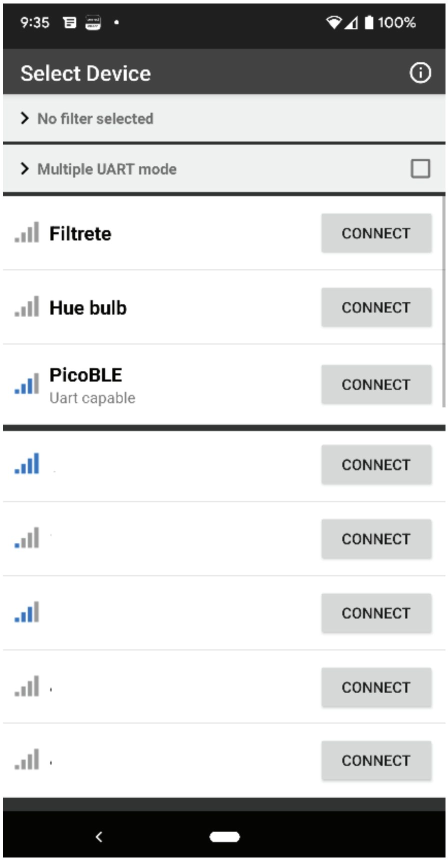


Рисунок 8.9. PicoBLE обнаружено на устройстве Android

12. После подключения к устройству мы должны увидеть следующий результат на терминале последовательного порта Pico:

```
code.py output:
Initializing the Bluefruit LE SPI Friend module
BLESPIFRIEND
nRF51822 QFACA00
025CAC4E3D288E91
0.8.1
0.8.1
Apr 10 2019
S110 8.0.0, 0.2

Waiting for a connection to Bluefruit LE Connect ...
.............
 *Connected!*
```

Рисунок 8.10. Интерфейс модуля Bluetooth Pico

13. После подключения к модулю Bluetooth нам будут представлены различные опции модуля, такие как **UART** и **Plotter**[68] (как показано на рис. 8.11 далее). В этом примере мы будем использовать интерфейс UART. На следующем рисунке также показан уровень сигнала нашего модуля PicoBLE. Величина **RSSI** (**Received Signal Strength Indicator**, *индикатор уровня принимаемого сигнала*) показывает качество сигнала для Bluetooth-соединения между телефоном и модулем:

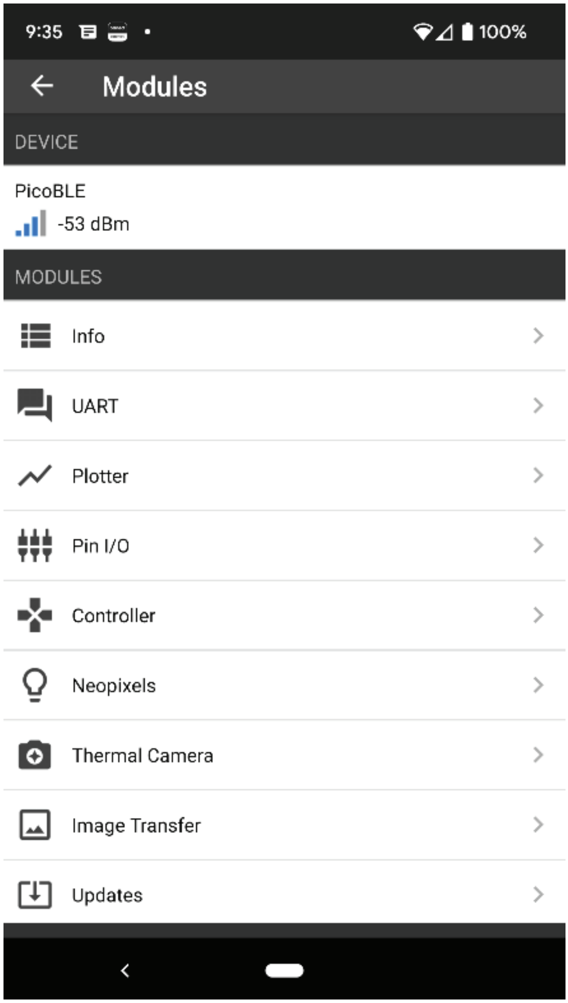


Рисунок 8.11. Опции Bluetooth-модуля на устройстве Android

14. Выбрав интерфейс UART, мы передадим сообщение «Hello World» на модуль Bluetooth (как показано на следующем рисунке). Обратите внимание, что модуль Bluetooth воспроизводит сообщение в обратном порядке:

[68] В данном случае Plotter – опция программы для вывода информации в графической форме. – *Прим. перев.*

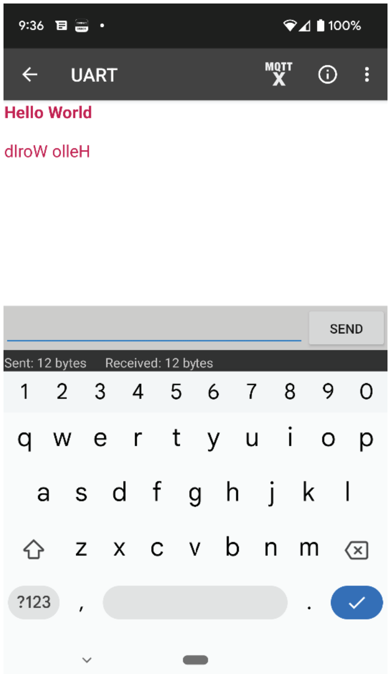


Рисунок 8.12. Передача сообщения модулю Bluetooth и получение отклика

15. На Pico мы увидим отладочные сообщения через терминал последовательного порта. На рис. 8.13 обратите внимание, что модуль Bluetooth получил 12 байт (`Привет World\n`[69]) и повторяет сообщение в обратном порядке:

```
Waiting for a connection to Bluefruit LE Connect ...
.............
 *Connected!*
Read 12 bytes: b'Hello World\n'
Writing reverse...
b'\ndlroW olleH'
```

Рисунок 8.13. Отладочные сообщения на Pico

Теперь мы подключили наш Pico к мобильному устройству с помощью модуля Bluetooth. В следующем разделе обсудим отображение на телефоне показаний датчиков, размещенных в облаке с помощью Adafruit IO.

Размещение показаний датчика через модуль Bluetooth

В этом разделе мы обсудим публикацию данных, собранных модулем Bluetooth, с помощью Adafruit IO. Если вы помните, мы использовали этот сервис в главе 3 «*Проекты домашней автоматизации*» для размещения данных

[69] Строго говоря, в данном сообщении, с учетом символа перевода строки («\n»), не 12 байт, а 13 или даже 14 («\r\n»: возврат каретки + новая строка). – *Прим. перев.*

в облаке. Чтобы зарегистрировать учетную запись Adafruit IO, рекомендуется следовать инструкциям в указанной главе. На следующей схеме показан поток данных из Pico в Adafruit IO:

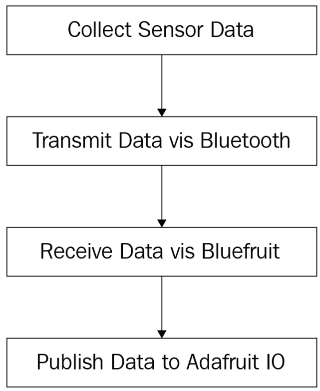


Рисунок 8.14. Размещение данных в Adafruit IO

Данные, собранные Pico, передаются по Bluetooth, если он подключен. Данные, собранные смартфоном, отправляются в Adafruit IO.

Примерный сценарий

Давайте рассмотрим сценарий, в котором мы установили датчики углекислого газа или датчики температуры, подключенные к Raspberry Pi Pico, в разных местах для сбора данных. Мы хотим собирать данные с датчиков с помощью мобильного устройства всякий раз, когда оказываемся поблизости от датчика, и отправлять данные в облако.

В этом разделе мы обсудим размещение данных, собранных с датчиков, с помощью мобильного приложения. Для подключения датчика углекислого газа рекомендуется следовать инструкциям из главы 7 «*Разрабатываем устройство слежения за качеством воздуха*», однако вы можете выбрать любой датчик. В конце концов, для пробы можно просто отправить случайно сгенерированные данные.

Программирование Pico

Рассмотрим пример кода, необходимого для сбора данных с датчика углекислого газа и передачи их по Bluetooth при подключении к мобильному устройству. Пример, обсуждаемый в этом разделе, идентичен рассмотренному в предыдущем разделе. Мы обсудим только различия между этими двумя примерами. Код из этого раздела доступен для загрузки в репозитории GitHub этой главы в файле *bluefruit_SCD30.py*.

Этапы, связанные с передачей данных датчика углекислого газа, следующие:

1. Помимо импорта необходимых модулей, рассмотренных в предыдущем разделе, мы также импортируем модуль *adafruit_scd30* в строке номер 6:

```
import adafruit_scd30
```

2. Теперь инициализируем интерфейс I2C на линиях 14 и 15, а также датчик углекислого газа:

```
i2c = busio.I2C(board.GP9, board.GP8)
scd = adafruit_scd30.SCD30(i2c)
```

3. Другое важное изменение произошло по условию доступности датчика SCD30. Если мы подключены к устройству Bluetooth и доступны новые данные от датчика углекислого газа, то передаем данные с помощью метода `uart_tx`:

```
if scd.data_available:
  data = str(scd.CO2) + "\n"
  bluefruit.uart_tx(bytes(data, "utf-8"))
time.sleep(2.0)
```

4. Теперь мы имеем возможность видеть данные на экране смартфона:

Рисунок 8.15. Данные датчика CO_2 на мобильном устройстве

5. Теперь мы также имеем возможность отображать собранные данные на графике (вкладка **Plotter**):

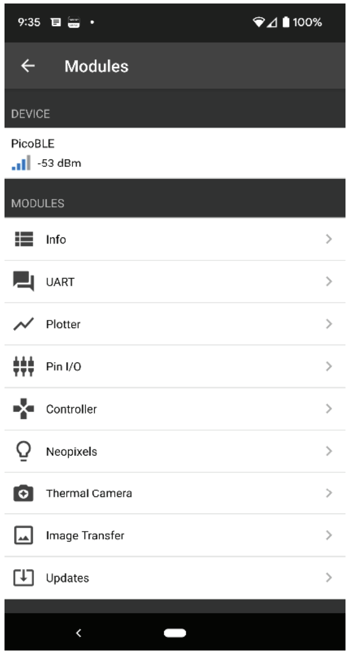


Рисунок 8.16. Вкладка **Plotter** в приложении Bluefruit

6. Вкладка **Plotter** дает возможность видеть график поступающих данных в реальном времени, как показано на следующем рисунке:

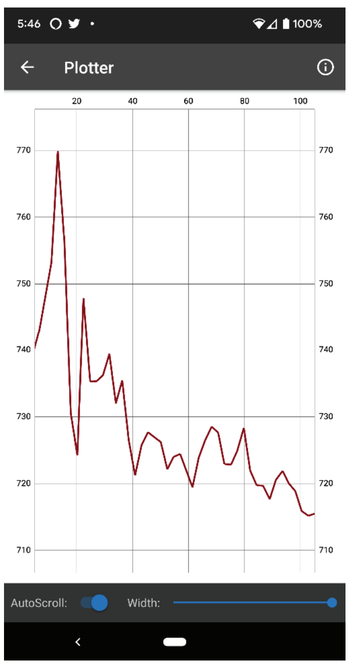


Рисунок 8.17. График собранных данных в реальном времени

7. Теперь давайте рассмотрим публикацию собранных данных с помощью MQTT. **MQTT** расшифровывается как **MQ Telemetry Transport** («MQ» – наименование серии продуктов, разработанных фирмой IBM). MQTT – протокол, разработанный для маломощных устройств, таких как Raspberry Pi Pico, для размещения данных в интернете. Мы предполагаем, что вы зарегистрировали учетную запись на ***io.adafruit.com***, следуя инструкциям из главы 3 «*Проекты домашней автоматизации*».
8. Перейдите к ***io.adafruit.com*** и создайте новый поток (на вкладке **Feeds**) под названием `Collected Data` (собранные данные), как показано на следующем рисунке:

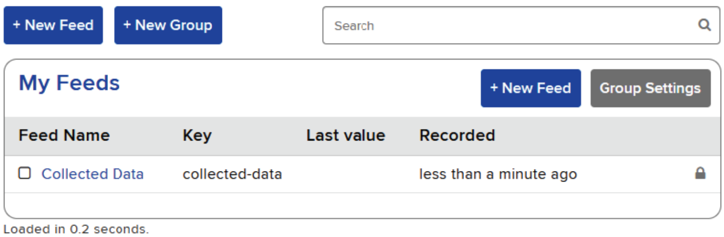


Рисунок 8.18. Создание нового потока данных

9. Вы можете найти сведения о созданном потоке в разделе **Feed Info**, как показано на рисунке:

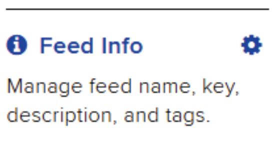


Рисунок 8.19. Информация о потоке

10. Теперь нам нужно имя канала MQTT, как показано на рис. 8.20.

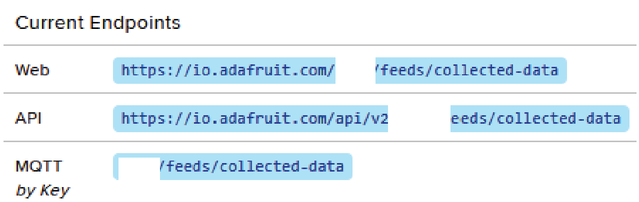


Рисунок 8.20. Имя канала MQTT

11. Сейчас мы возвращаемся к мобильному приложению и нажимаем кнопку **MQTT**:

Рисунок 8.21. Нажмите кнопку **MQTT**

12. На открывшейся странице мы введем имя канала MQTT в поле **Uart RX**, как показано на рис. 8.22. Нам также необходимо предоставить ключ учетной записи Adafruit IO. Эта информация доступна на вкладке **My Key** учетной записи. На странице есть QR-код, который позволяет захватить учетные данные с помощью камеры смартфона. При нажатии кнопки сканирования QR-кода **Scan Code** запустится камера смартфона, после сканирования поле **Pass/Key** заполняется автоматически. Вам просто нужно ввести свое имя пользователя. Мы не показываем вкладку **My Key** на странице Adafruit IO по соображениям конфиденциальности.

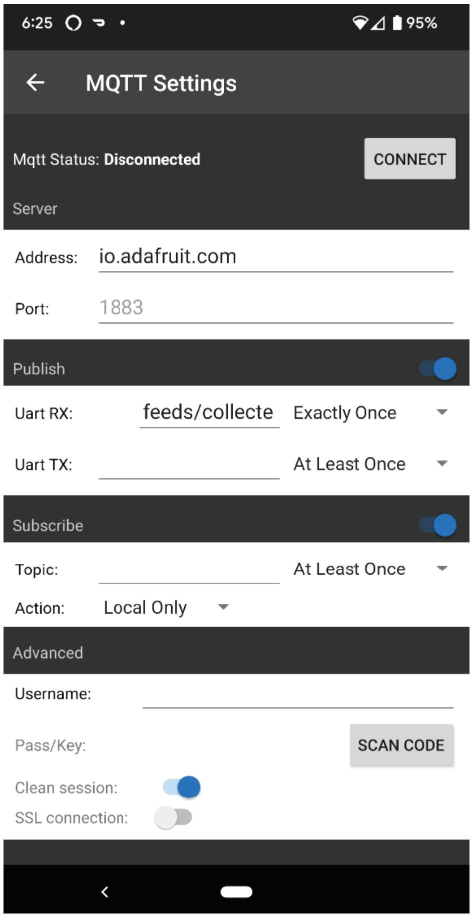


Рисунок 8.22. Введите имя канала, имя пользователя и QR-код захвата, нажав **Scan Code**

13. Теперь нажмите кнопку **Connect** (показана на рис. 8.22 выше), и вы должны подключиться к MQTT-серверу Adafruit. Входящие данные автоматически публикуются на сервере MQTT.

14. Теперь вы имеете возможность просматривать данные на странице **Feeds** (*Каналы*) вашей учетной записи Adafruit IO:

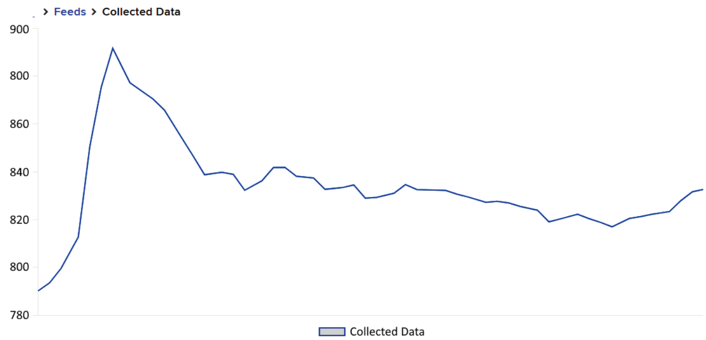


Рисунок 8.23. Собранные данные датчика углекислого газа

Теперь вы можете собирать данные с помощью модуля Bluetooth. В следующем разделе мы рассмотрим взаимодействие Raspberry Pi Pico с модулем Sigfox.

Подключение модуля Sigfox

В этом разделе мы обсудим взаимодействие модуля Sigfox с Raspberry Pi Pico. Прежде всего давайте быстро рассмотрим модуль Sigfox[70].

Что такое Sigfox?

Sigfox – разновидность маломощной глобальной сети (LPWAN), управляемой одноименной французской компанией. Радиус действия радиостанций Sigfox составляет около 10–30 км, а оператор сети ограничивает общее количество передач 140 сообщениями в день, при этом каждое сообщение ограничено 12 байтами. Этот тип сети подходит для беспроводных узлов с питанием от батарей, развернутых в больших масштабах. Sigfox обычно работает в нелицензионных ISM-диапазонах[71]. Вы можете узнать больше о Sigfox здесь: **https://build.sigfox.com/sigfox**.

> **Проблемы Sigfox**
>
> На момент написания этой главы Sigfox подала заявление о защите от банкротства. Мы не знаем, что это означает для будущего Sigfox networks[72].

[70] Способ использования Sigfox, приведенный далее в этом разделе, ориентирован на территорию США. Поэтому раздел следует рассматривать только в качестве примера обращения с подобными сервисами (учитывая также проблемы компании Sigfox, см. примечание в следующем абзаце). – *Прим. перев.*

[71] ISM (Industrial, Scientific and Medical) – диапазоны частот, не требующие для их использования получения лицензии (обычно с ограничениями на мощность и дальность передачи). В США один из самых популярных ISM-диапазонов составляет 915 МГц, в России и Европе – 433 и 868 МГц. – *Прим. перев.*

[72] О проблемах Sigfox в России см. материал по адресу **https://www.comnews.ru/content/220907/2022-06-27/2022-w26/sigfox-ne-pokoril-rossiyu**. О некоторых менее проблемных аналогичных сервисах также упоминается в этой статье. – *Прим. перев.*

Сетям Sigfox требуется годовая лицензия от оператора для подключения к их сети. В Соединенных Штатах одна лицензия на передачу 140 сообщений в день стоит около 19 долларов США. Также важно выяснить, доступна ли сеть Sigfox в вашем регионе. Вы можете узнать, доступно ли покрытие в вашем регионе, здесь: **https://www.sigfox.com/en/coverage**.

Например, покрытие Sigfox в северной части штата Нью-Йорк в Соединенных Штатах показано на следующем рисунке:

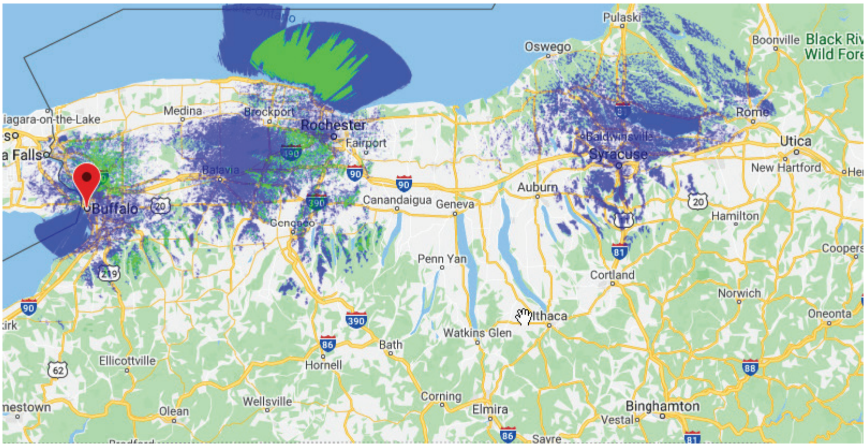


Рисунок 8.24. Охват Sigfox в северной части штата Нью-Йорк

В следующем разделе мы рассмотрим модуль Sigfox, используемый в примере.

Модуль Sigfox

В данном разделе мы обсудим модуль RC1692 Sigfox, используемый в этом примере (показан на следующем рисунке). RC1692 (доступен в **https://bit.ly/3DtRIap**) представляет собой радиоприемник Sigfox, работающий в полосе частот 902–928 МГц. Это модуль, сертифицированный FCC, который предназначен для использования в Соединенных Штатах и нескольких других странах (более подробную информацию вы можете найти в документации по адресу **https://bit.ly/32Wr9xX**).

Мы не смогли найти дешевых наборов для радиомодулей Sigfox, поэтому разработали собственную печатную плату в форм-факторе Adafruit Featherwing. Вы можете изготовить собственный экземпляр, используя файлы чертежей, доступные по адресу: **https://bit.ly/3pxf3mq**.

> **Дополнительная плата, совместимая с Pico**
>
> В этом примере используется Featherwing, но в разработке также находится плата, совместимая с Pico, которая может быть использована для проектов в будущем. Мы загрузим файлы чертежей в репозиторий данной книги.

Мы спроектировали плату модуля Sigfox с учетом фактора Featherwing, потому что создали плату до того, как появился Raspberry Pi Pico, но соединение его с этим радиомодулем довольно простое, поскольку модуль имеет интерфейс UART.

Наш Sigfox Featherwing содержит шестиконтактный разъем, предназначенный для стандартного адаптера FTDI (например, такого: **https://bit.ly/32Ws15H**). Это позволяет вам зарегистрировать радиостанцию в сети Sigfox. На следующем рисунке изображена плата Sigfox Featherwing, расположенная на макетной плате:

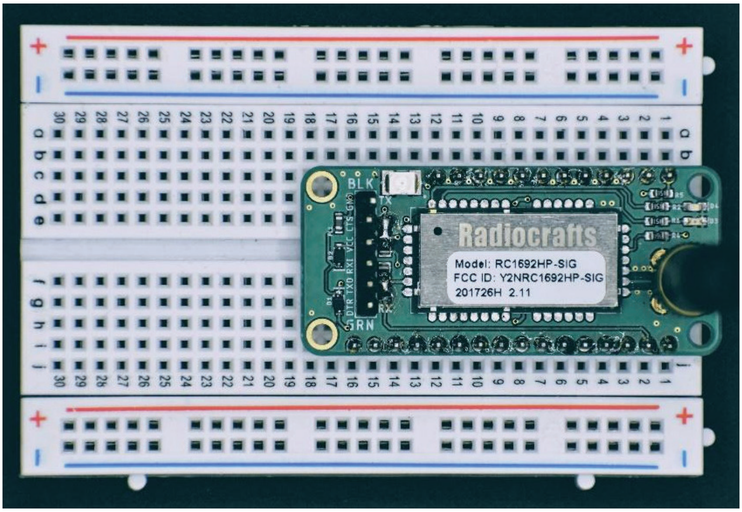


Рисунок 8.25. Плата RC1692 Sigfox Featherwing

В следующем разделе мы обсудим настройку Sigfox radio, прежде чем подключать его к Raspberry Pi Pico.

Настройка модуля Sigfox

В этом разделе мы обсудим настройку модуля Sigfox. Она включает в себя следующие шаги:

1. Первый шаг – убедиться, что в вашем регионе есть оператор сети Sigfox. Вы можете найти информацию о покрытии в вашем регионе, воспользовавшись ссылкой, размещенной ранее в этом разделе.
2. Если Sigfox доступен в вашем регионе, вы можете приобрести годовую лицензию на сайте **https://buy.sigfox.com**. Цена зависит от оператора и опций, доступных в вашем регионе.
3. Вам также необходимо создать учетную запись на **https://backend.sigfox.com/auth/login**. Приобретенная лицензия автоматически добавляется в вашу учетную запись Sigfox.
4. Теперь вы можете создать учетную запись на вкладке в верхней части веб-страницы, как показано на следующем рисунке:

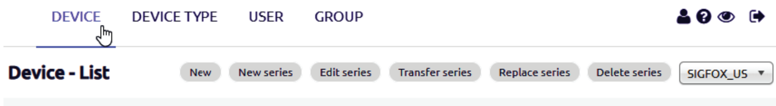


Рисунок 8.26. Добавление нового устройства в сервис Sigfox

5. После нажатия на кнопку **New** вы попадете на страницу, где можно добавить информацию, относящуюся к новому устройству.
6. Мы зарегистрируем устройство в качестве прототипа, и нам также понадобятся некоторые уникальные идентификаторы, принадлежащие модулю. Мы будем использовать программное обеспечение *SIG-CCT* с веб-сайта производителя: **http://www.radiocrafts.com**.
7. Чтобы получить информацию, необходимую для регистрации модуля, следуем инструкциям производителя (**https://bit.ly/3rDMXsw**).
8. После регистрации устройства следует подключить Sigfox Featherwing к Pico. Соединения с модулем показаны ниже (имя слева от стрелки относится к контакту на Raspberry Pi Pico, справа – к контакту на плате Sigfox Featherwing):

   - VBUS → VCC;
   - GP4 (TX) → RX;
   - GP5 (RX) → TX;
   - GP19 → RESET;
   - GND соединяются между собой.

   На следующем рисунке показано расположение контактов Raspberry Pi Pico для соединения с Sigfox Featherwing:

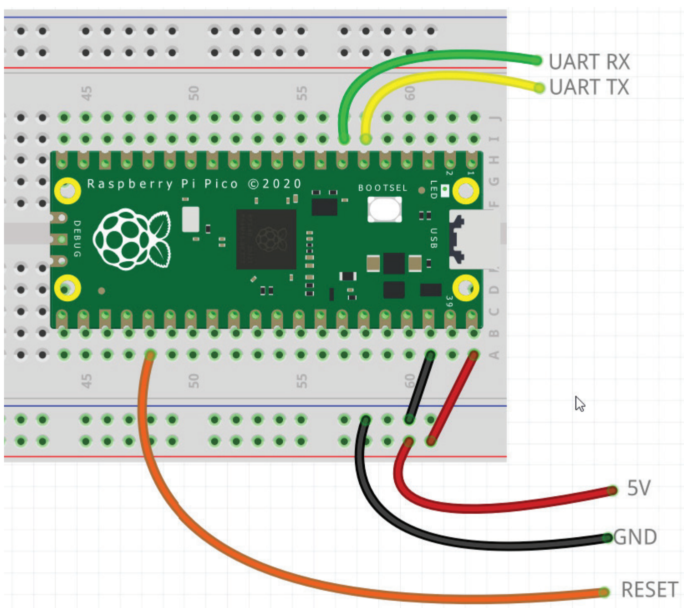


Рисунок 8.27. Соединения Pico с модулем Sigfox

В следующем разделе мы обсудим код, необходимый для управления и передачи данных с помощью модуля Sigfox.

Пример кода

Давайте рассмотрим код, необходимый для управления модулем Sigfox. Обсуждаемый здесь код доступен для загрузки в папке *downloads* репозитория GitHub этой главы в виде файла *uart_sigfox.py*.

1. Как всегда, первым шагом является импорт необходимых модулей:

```
import time
import board
import busio
import digitalio
import binascii
```

2. Поскольку нам нужен интерфейс UART, мы собираемся его инициализировать на контактах GP4 и GP5 следующим образом:

```
uart = busio.UART(tx=board.GP4, rx=board.GP5, baudrate=19200)
```

3. Нам нужно сбросить модуль Sigfox во время инициализации, что делается с помощью вывода *Reset*. Для этого мы объявим GP19 в качестве выходного вывода и на время выставим его в низкий уровень:

```
pin = digitalio.DigitalInOut(board.GP19)
pin.direction = digitalio.Direction.OUTPUT
pin.value = False
time.sleep(5.0)
pin.value = True
```

4. Теперь мы будем передавать данные дважды с 15-секундной задержкой между передачами. Каждая передача должна начинаться с размера полезной информации. В следующем фрагменте кода первый элемент данных ссылается на размер – например, 2 байта. Мы отправляем 2 случайных байта 0x10 и 0x20:

```
data = [2, 0x10, 0x20]
uart.write(bytes(data))
time.sleep(10)
uart.write(bytes(data))
print("Done")
```

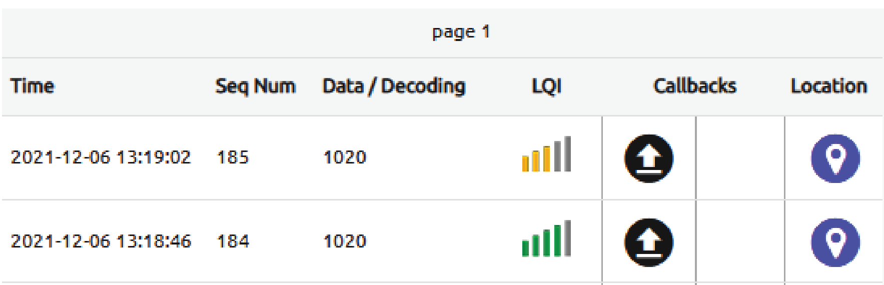
Ограничения на сообщения Sigfox

Как упоминалось ранее, Sigfox ограничивает количество сообщений 140 сообщениями в день. Не используйте бесконечный цикл и не позволяйте вашему коду выполняться непрерывно. Предполагается, что беспроводной Sigfox-модуль отправляет сообщения один или два раза в день.

5. Объединив предыдущие фрагменты кода, получаем следующее:

```
import time
import board
import busio
import digitalio
import binascii

pin = digitalio.DigitalInOut(board.GP19)
pin.direction = digitalio.Direction.OUTPUT

uart = busio.UART(tx=board.GP4, rx=board.GP5, baudrate=19200)
data = [2, 0x10, 0x20]

pin.value = False
time.sleep(5.0)
pin.value = True

uart.write(bytes(data))
time.sleep(10)
uart.write(bytes(data))
print("Done")
```

Сохранив этот пример кода как *code.py* и запустив его, мы сможем передавать данные в сеть Sigfox. Всякий раз, когда данные передаются успешно, синий светодиод на модуле Sigfox мигает пару раз. Мы можем просмотреть данные на сервере Sigfox, как показано на следующем рисунке, где два переданных случайных байта показываются на панели мониторинга:



page 1

Time	Seq Num	Data / Decoding	LQI	Callbacks	Location
2021-12-06 13:19:02	185	1020			
2021-12-06 13:18:46	184	1020			

Рисунок 8.28. Посланные данные на панели Sigfox

Вы можете просмотреть приблизительное местоположение передачи данных, нажав на кнопку в колонке **Location**. В данном случае данные передаются из Буффало, штат Нью-Йорк (рис. 8.39). Теперь мы рекомендуем подключить выбранный вами датчик и передавать реальные данные. Имейте в виду, что у вас есть только 12 байт на сообщение.

В следующем разделе мы обсудим подключение узлов LoRa с использованием Raspberry Pi Pico.

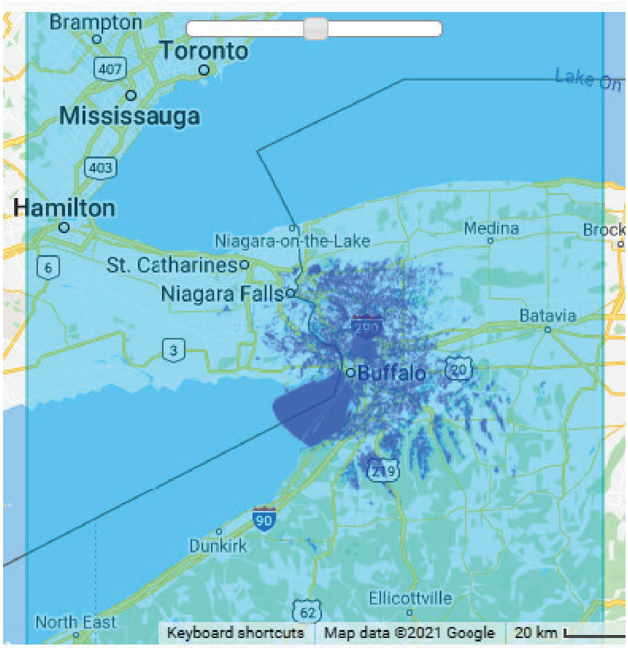


Рисунок 8.29. Местоположение передачи данных

Подключение модулей LoRa

В этом разделе мы собираемся подключить два модуля LoRa к Raspberry Pi Pico. Это позволит нам установить беспроводную связь между двумя устройствами.

Что такое LoRa?

Прежде чем мы начнем с нашего примера, давайте начнем с краткого введения в LoRa. **LoRa** расшифровывается как **Long Range Radio** (*дальнобойное радио*) и представляет собой беспроводной протокол, предназначенный для связи на большие расстояния с низким энергопотреблением. Он был разработан организацией под названием Semtech. По назначению LoRa и Sigfox (обсуждавшийся в предыдущем разделе) похожи, основное отличие заключается в том, что первый позволяет вам настроить свою собственную сеть, в то время как второй управляется отдельным оператором, который требует покупки лицензии для подключения к его сети. В следующей таблице показаны различия между радиосвязью LoRa и Sigfox[73]:

[73] Кроме отличий, перечисленных в таблице, важное различие между LoRa и сервисами, подобными Sigfox, заключается в разных масштабах задач, для которых они предназначены. LoRa действует на меньших расстояниях, измеряемых единицами километров в пределах прямой видимости: фактически в масштабе фермерского хозяйства, территории предприятия или жилого квартала. Тогда как упомянутые сервисы позволяют связываться между разными городами и целыми регионами. – *Прим. перев.*

Таблица 8.1. Различия между LoRa и Sigfox

LoRa	Sigfox
Вы можете настроить свою собственную сеть в любом месте по вашему выбору	Сетью управляет лицензированный оператор. Ограничено местом, где работают сети Sigfox
Вам необходимо настроить свой собственный шлюз и серверную инфраструктуру. Нет необходимости платить ежегодный лицензионный сбор	Sigfox предлагает годовую лицензию. Сообщение может быть перенаправлено из Sigfox в ваше облачное хранилище*
Никаких ограничений на количество сообщений	Ограничено 140 сообщениями в день
* Для LoRa и других способов локальной связи в размещении данных во внешнем хранилище тоже нет особенных проблем, как мы видели на примерах с Bluetooth-модулями. – *Прим. перев.*	

Существует некоммерческая организация под названием LoRa Alliance, которая разработала стандарт для взаимодействия между различными сетями LoRa. Радиостанции LoRa используют бесплатные ISM-диапазоны, которые находятся на частоте 915 МГц в Соединенных Штатах, на частотах 433 и 868 МГц в Европе и России. Вы можете узнать больше о LoRa здесь: **https://wireless-e.ru/standarty/tehnologiya-lora-v-voprosah-i-otvetah/**.

Примерный сценарий

Давайте рассмотрим сценарий, в котором датчик устанавливается в месте, где не обязательно имеется сеть Wi-Fi. В этом месте поблизости нет никаких зданий, поэтому понадобится устройство с батарейным питанием. Именно в таком случае могут подойти модули LoRa. Вы также можете создавать беспроводные узлы сети, поддерживаемые модулем LoRa. В то время как один узел устанавливается на месте, другой может быть установлен внутри помещения. Второй узел может быть подключен к модулю Wi-Fi (как показано на схеме рис. 8.30). Это позволяет собирать данные датчиков и размещать их в облаке с помощью Wi-Fi:

Рисунок 8.30. Поток данных между двумя узлами сети LoRa

Давайте начнем.

В этом примере используются две платы Raspberry Pi Pico и две платы радиомодуля LoRa RFM95W. Наш первый шаг – подключить каждый модуль RFM95W к Raspberry Pi Pico.

RFM95W – это приемопередатчик LoRa, способный передавать сигналы на расстояние до 2 км в прямой видимости[74]. Вы можете узнать больше о различных радиомодулях здесь: **https://bit.ly/3BiR7Ja**.

Прежде чем мы приступим к работе, возможно, потребуется установка разъемов на плату RFM95W. Мы рекомендуем следовать инструкциям по адресу **https://bit.ly/339Zwlh**[75]. Плата подключается к Pico следующим образом (слева – контакты Pico, справа – контакты на плате RFM95W):

- GP5 → CS;
- GP6 → RESET;
- GP2 → SCK;
- GP3 → MOSI;
- GP4 → MISO;
- VBUS → VIN;
- GND соединяются между собой.

На следующем рисунке показана схема подключения платы RFM95W к Raspberry Pi Pico:

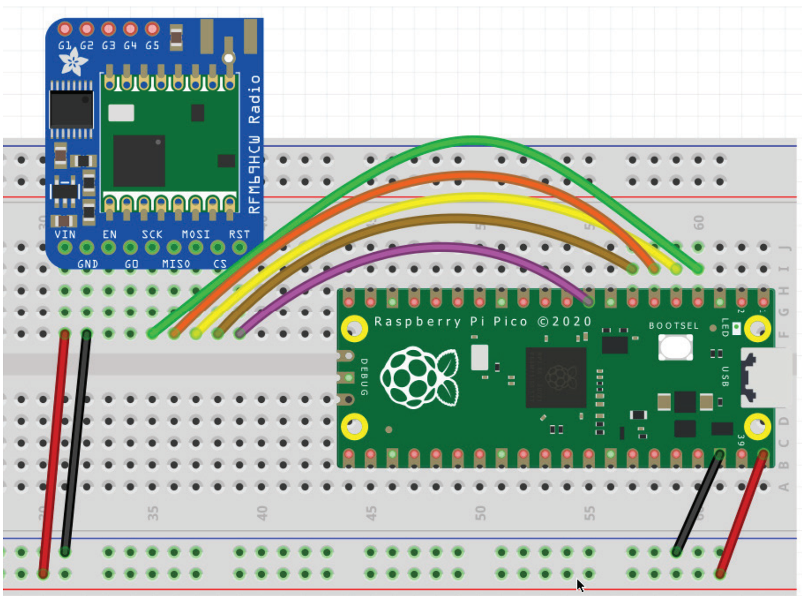


Рисунок 8.31. Схема подключения платы Adafruit RFM95W к Raspberry Pi Pico

[74] Расстояние, на котором можно установить LoRa-связь, сильно зависит от антенны, от высоты ее установки над поверхностью земли, от препятствий, встречающихся на пути радиосигнала (заметим, что понятие «прямой видимости» довольно неопределенное), и от настроек радиочасти устройства. Для частоты 868 МГц можно ориентироваться на дальность от примерно 500–1000 м при установке приемной антенны на уровне ~1,5–2 м до 4–5 км при установке на мачте ~10 м. – *Прим. перев.*

[75] Обратите внимание по этой ссылке на установку SMA-разъема для внешней антенны (показан на фото рис. 8.32) – более простые конструкции антенн (отрезок провода отмеренной длины, спиральная антенна), скорее всего, не обеспечат надлежащей дальности приемопередачи. – *Прим. перев.*

Макет соединений беспроводного узла на Adafruit RFM95W показан на следующем фото:

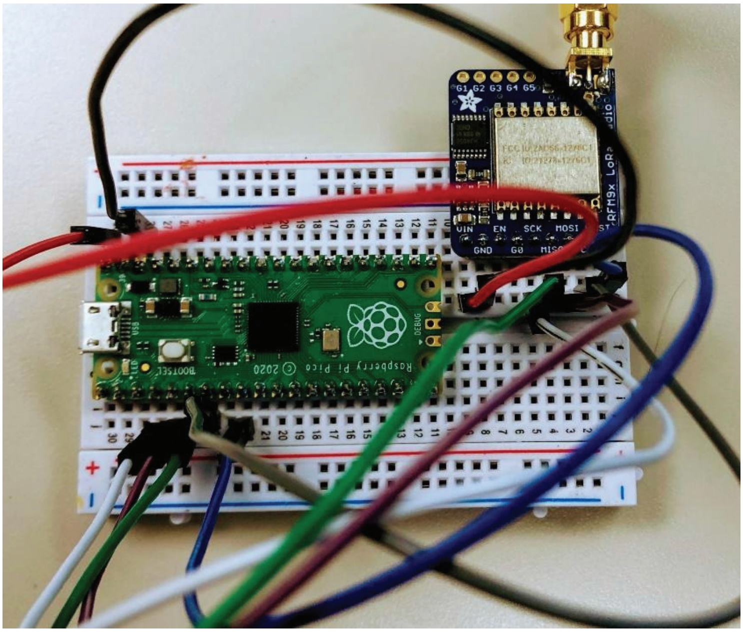


Рисунок 8.32. Макет беспроводного узла

Давайте посмотрим на код, необходимый для передачи сообщения с использованием модуля RFM95W. Код, приведенный далее, доступен для загрузки в папке *downloads* репозитория GitHub этой главы в файле *code_lora.py*:

1. Первый шаг – импорт необходимых модулей:

```
import board
import busio
import digitalio
import adafruit_rfm9x
```

2. Следующий шаг – объявить радиочастоту. Поскольку в США мы используем диапазон 915 МГц, то определим эту частоту с помощью переменной `RADIO_FREQ_MHZ`. Это следует изменить в соответствии с утвержденной частотой для ISM-диапазонов в вашей стране – например, в Европе эти частоты составляют либо 433, либо 868 МГц.

```
RADIO_FREQ_MHZ = 915.0
```

3. Теперь инициализируем интерфейс SPI и радиочасть:

```
CS = digitalio.DigitalInOut(board.GP5)
RESET = digitalio.DigitalInOut(board.GP6)
spi = busio.SPI(board.GP2, MOSI=board.GP3, MISO=board.GP4)
rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, RADIO_FREQ_MHZ)
```

4. Мы будем использовать встроенный светодиод для индикации наличия входящего пакета данных:

```
LED = digitalio.DigitalInOut(board.GP25)
LED.direction = digitalio.Direction.OUTPUT
```

5. Следующим шагом является настройка мощности передачи[76]:

```
rfm9x.tx_power = 23
```

Теперь мы запустим основной цикл, в котором делаем тестовую передачу. После тестовой передачи мы ждем входящих сообщений. Если входящих сообщений нет, выключаем индикатор. Если есть входящие сообщения, мы расшифровываем их и выводим на экран:

```
while True:
  rfm9x.send(bytes("Hello world!\r\n", "utf-8"))
  packet = rfm9x.receive()
  if packet is None:
    LED.value = False
    print("Received nothing! Listening again...")
  else:
    LED.value = True
    print("Received (raw bytes): {0}".format(packet))

    packet_text = str(packet, "ascii")
    print("Received (ASCII): {0}".format(packet_text))

    rssi = rfm9x.last_rssi
    print("Received signal strength: {0} dB".format(rssi))
```

7. Сохраним показанный пример кода в виде *code.py* на обоих Pico и будем получать следующий результат всякий раз, когда будет приходить сообщение от другого Pico:

```
Received (raw bytes): bytearray(b'Hello world!\r\n')
Received (ASCII): Hello world!

Received signal strength: -49 dB
Received nothing! Listening again...
Received nothing! Listening again...
Received (raw bytes): bytearray(b'Hello world!\r\n')
```

Рисунок 8.33. Сообщение, полученное через LoRa

Если вы подключаете датчик к Pico, как описано в примере Bluetooth, вы должны иметь возможность собирать данные с помощью одного беспроводного узла и передавать их на другой. Мы поделимся примером в репозито-

[76] Согласно описанию библиотеки *adafruit_rfm9x*, мощность передатчика модулей (`tx_power`) может быть указана в диапазоне от 5 до 23 (децибел, дБ). По умолчанию установлена средняя величина 13 дБ. – *Прим. перев.*

рии этой книги. Если второй узел дополнительно подключен к модулю Wi-Fi, можно опубликовать собранные данные в такой службе, как ThingSpeak или Adafruit IO.

Теперь если у вас есть несколько узлов, собирающих данные, вы можете установить шлюз LoRa для сбора данных со всех беспроводных узлов. Мы предоставили примеры с более подробной информацией в репозитории GitHub этой книги.

Итоги

В этой главе мы начали с подключения модуля Bluetooth к Raspberry Pi Pico. Затем обсудили размещение данных датчиков через Bluetooth с помощью мобильного устройства на внешнем хранилище. Мы также подключили модуль Sigfox к Pico. Наконец, построили два узла LoRa, используя Pico.

В следующей главе обсудим создание робота с использованием Pico.

Глава 9

Строим робота!

В этой главе мы немного позабавимся созданием робота с помощью Raspberry Pi Pico, чуть-чуть углубившись в программирование с использованием MicroPython. Создание робота будет проводиться с применением готового набора для робототехники. Начнем с рассмотрения компонентов набора, протестировав их; затем мы попробуем создать робота, следующего за линией, и робота, избегающего препятствий.

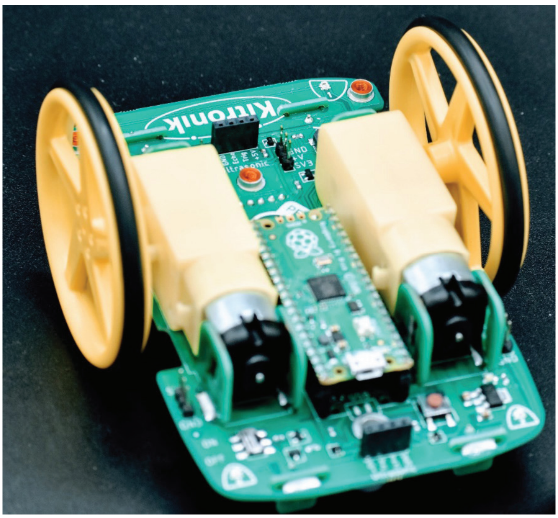


Рисунок 9.1. Автономная робототехническая платформа Kitronik для Raspberry Pi Pico

Это может быть развлекательный проект выходного дня, который можно было бы усовершенствовать несколькими путями для участия в соревнованиях роботов. Мы дадим вам несколько идей ближе к концу данной главы.

Темы, обсуждаемые в этой главе, включают следующее:

- установка необходимых компонентов;
- управление светодиодами;
- выбор двигателя и способы управления различными типами;
- тестирование датчиков;
- тестирование робота.

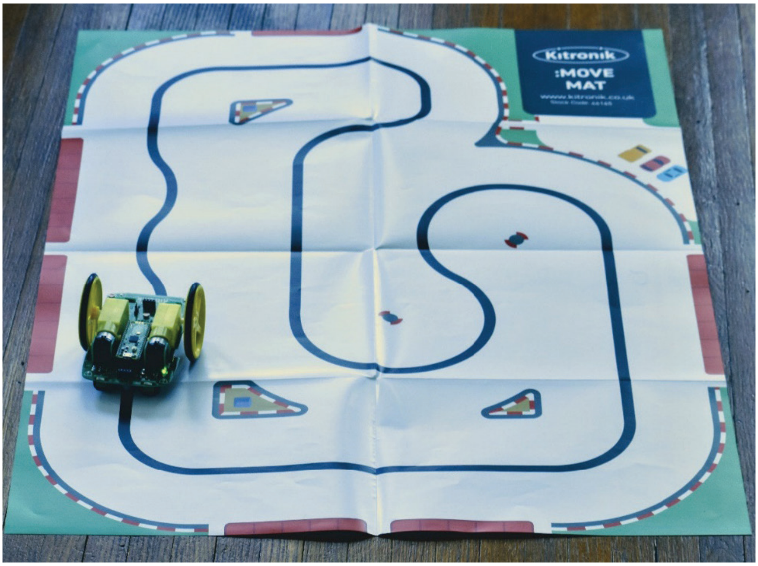


Рисунок 9.2. Робот, следующий за линией, построенный из комплекта Kitronik

Технические требования

Для этой главы рекомендуется использовать следующее оборудование:

- Raspberry Pi Pico (ссылка: **https://bit.ly/3AJtoAf**);
- Kitronik Autonomous Robotics Platform for Pico (**https://kitronik.co.uk/products/5335-autonomous-robotics-platform-for-pico**[77]).

Примеры кода для этой главы доступны по адресу **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIY-Workshop/tree/main/chapter_09**.

Видеоролики с кодом в действии для главы можно посмотреть по адресу **https://bit.ly/3kNCFkM**.

В следующем разделе мы установим необходимую библиотеку для нашего робота.

Установка необходимых компонентов

Прежде чем начнем, нам нужно установить IDE Thonny, а также установить MicroPython на Raspberry Pi Pico.

> **Установка MicroPython**
>
> Если вы незнакомы с процессом установки MicroPython на Pico, то можете следовать процессу установки, описанному в главе 1 «*Начало работы с Raspberry Pi Pico*».

[77] К сожалению, напрямую в российских интернет-магазинах указанный набор не наблюдается. – *Прим. перев.*

Необходимая библиотека для комплекта Kitronik Robotics kit доступна для загрузки в виде ZIP-файла из хранилища по адресу **https://bit.ly/3K35aoM**.

После извлечения содержимого ZIP-архива можно установить библиотеку с помощью IDE Thonny следующим образом.

1. В среде IDE Thonny перейдите в папку, содержащую извлеченные файлы библиотеки, как показано на рис. 9.3.

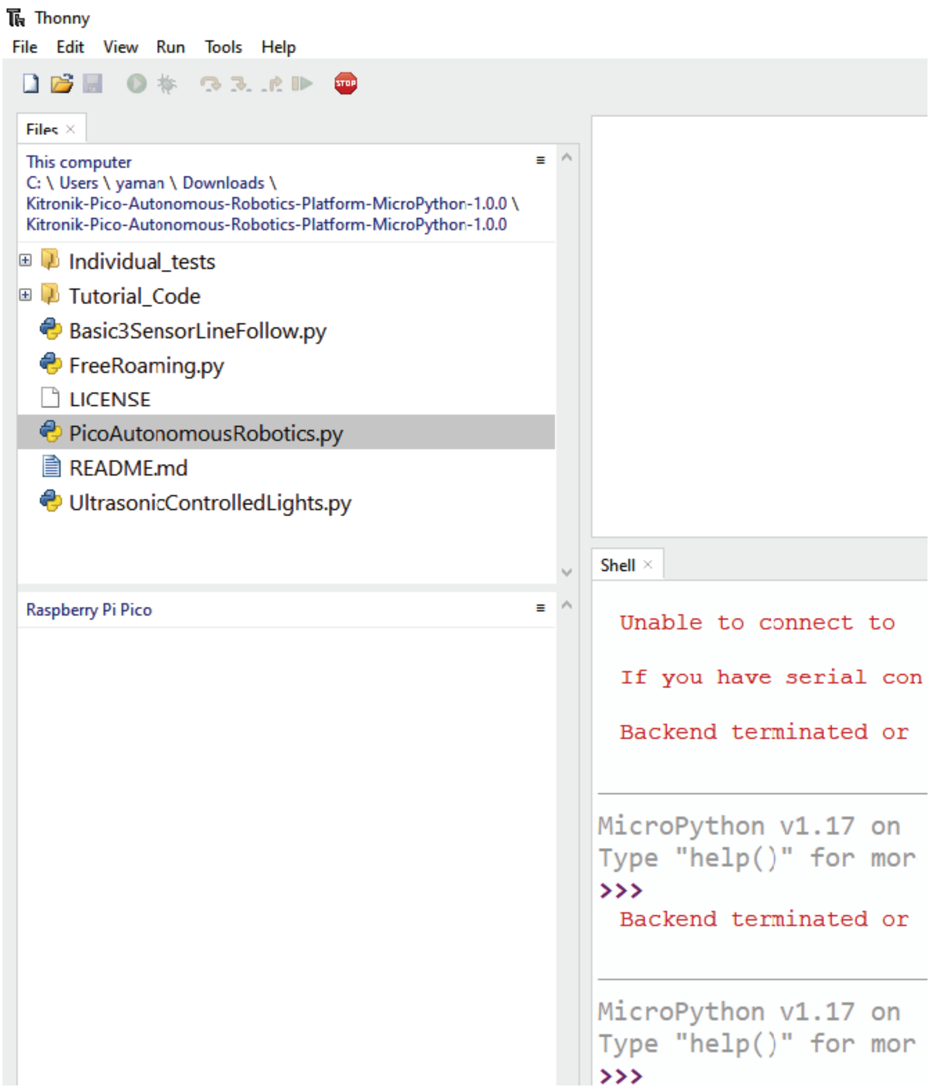


Рисунок 9.3. Расположение извлеченных файлов в среде разработки Thonny IDE

2. Теперь щелкните правой кнопкой мыши на *PicoAutonomousRobotics.py* и загрузите его в Pico, как показано на следующем рисунке:

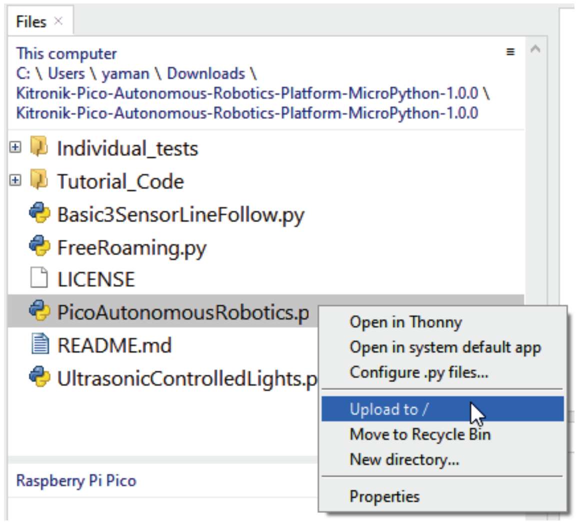


Рисунок 9.4. Загрузка библиотеки в Pico

3. Проверьте, что библиотека скопирована в Pico:

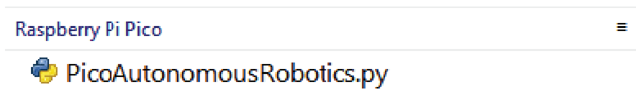


Рисунок 9.5. Библиотека, скопированная в Pico

В следующих разделах мы протестируем различные функции библиотеки. Но для начала нужно установить батареи в комплект для робота.

Установка батарей

Установка батарей показана на следующем рисунке. Роботу нужны четыре батарейки типа AA.

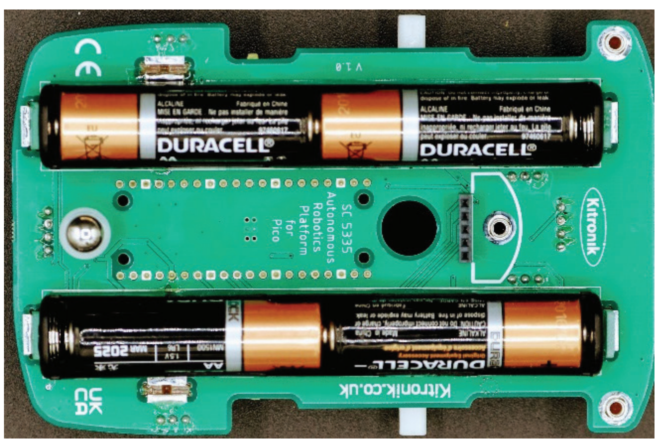


Рисунок 9.6. Установка батарей

Убедитесь, что движковый переключатель установлен в положение **OFF** (выключено), как показано на рис. 9.7.

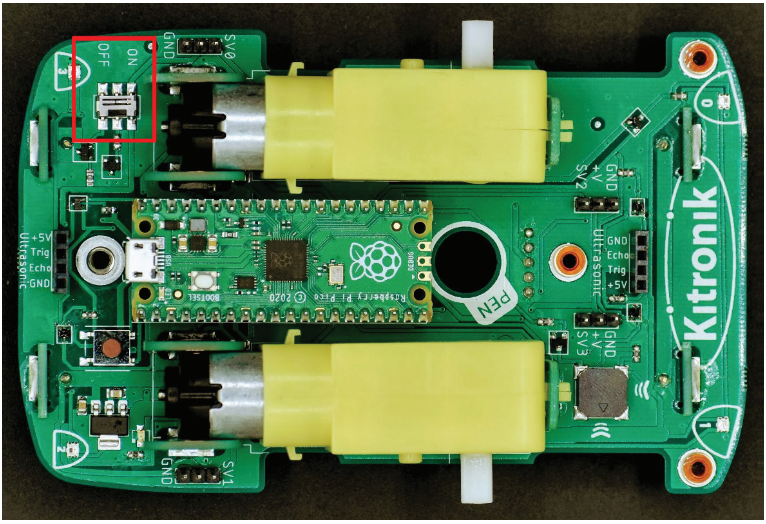


Рисунок 9.7. Расположение переключателя на платформе Kitronik Robotics

Мы начнем с тестирования RGB-светодиодов в следующем разделе.

Управление светодиодами

В этом разделе мы начнем с тестирования RGB-светодиодов, входящих в состав набора Kitronik Robotics. На корпусе расположены четыре RGB-светодиода (выделены на рис. 9.8). Светодиоды можно использовать для индикации действий робота. Мы будем зажигать их по кругу.

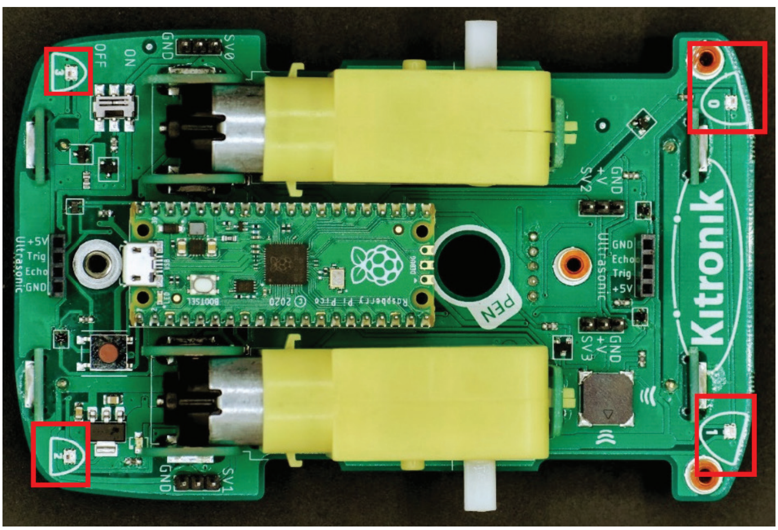


Рисунок 9.8. Расположение светодиодов на робототехнической платформе

Пример кода, обсуждаемый в этом разделе, выглядит следующим образом (файл *rgb_led_test.py*).

1. В среде IDE Thonny создайте файл с именем *main.py* и сохраните его в Pico. Как показано на следующем рисунке, при нажатии на кнопку **Save** (*Сохранить*) появляется диалоговое окно:

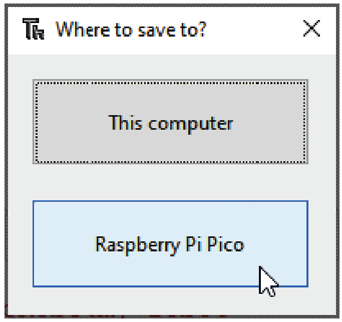


Рисунок 9.9. Сохранение *main.py* в Raspberry Pi Pico

2. Теперь рассмотрим код, предназначенный для управления светодиодами по кругу. Первым шагом является импорт необходимых модулей. Нам понадобится класс `KitronikPicoRobotBuggy` из библиотеки, которую мы установили ранее, а также модуль `time`:

```
from PicoAutonomousRobotics import KitronikPicoRobotBuggy
from time import sleep
```

3. Следующим шагом является инициализация объекта `my_robot`, принадлежащего классу `KitronikPicoRobotBuggy`. Мы также инициализируем переменную состояния с именем `LEDState`. Она будет использоваться для переключения цветов светодиодов между различными состояниями:

```
my_robot = KitronikPicoRobotBuggy()
LEDState = 0
```

4. Теперь мы объявляем функцию с именем `LEDPattern()`, где переключаем цвет светодиода на основе переменной состояния `LEDState`. Мы используем метод `setLED` для установки цвета светодиода. Этот метод принимает два аргумента – положение светодиода и цвет. Положение указано на рис. 9.8 , где прямоугольник выделяет светодиод и номер позиции.

```
def LEDPattern():
  global LEDState
  if LEDState == 0:
    my_robot.setLED(0,my_robot.BLUE)
    my_robot.setLED(1,my_robot.GREEN)
    my_robot.setLED(2,my_robot.YELLOW)
    my_robot.setLED(3,my_robot.RED)
    LEDState = 1
```

5. В предыдущем фрагменте кода мы установили цвета светодиодов и переменную `LEDState` равной 1. Когда `LEDState` равен 1 во время последующего выполнения, мы устанавливаем новые цвета светодиодов, как показано в следующем фрагменте, и изменяем переменную состояния `LEDState` на 2:

```
elif LEDState == 1:
  my_robot.setLED(0,my_robot.BLUE)
  my_robot.setLED(1,my_robot.GREEN)
  my_robot.setLED(2,my_robot.YELLOW)
  my_robot.setLED(3,my_robot.RED)
  LEDState = 2
```

6. Перед выходом из функции мы вызовем метод `show()` для отображения новых цветов светодиодов:

```
my_robot.show()
```

7. В бесконечном цикле `while` мы вызовем функцию `LEDPattern()` с задержкой в 1 секунду:

```
while True:
  LEDPattern()
  sleep(1)
```

После нажатия кнопки *Run* светодиоды начнут менять цвет по кругу[78]. На следующем рисунке показаны цвета светодиодов. Мы должны отметить, что цвета светодиодов могут быть плохо заметны на фото.

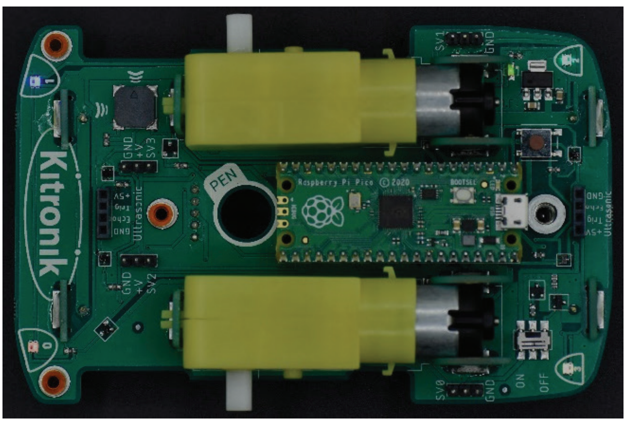


Рисунок 9.10. Светодиоды, меняющие цвет

Теперь, когда мы протестировали светодиоды RGB, обсудим двигатель.

78 В представленном коде, во-первых, имеется ошибка (цвета для состояния LEDState = 0 совпадают с цветами для состояния LEDState = 1), во-вторых, он недоделан: не представлены состояния для LEDState = 2 (и, очевидно, LEDState = 3), нет возврата к первому состоянию (LEDState = 0). Более компактный и выверенный пример кода смены цветов можно найти по указанной в разделе «*Технические требования*» этой главы ссылке на Kitronik Robotics Platform, если обратиться там к раскрывающемуся пункту *Related Resources*. – *Прим. перев.*

Выбор двигателя и способы управления различными типами

Для обычного настольного робота применяют различные варианты двигателей:

- двигатели постоянного тока (DC);
- шаговые двигатели;
- серводвигатели.

Давайте обсудим особенности этих типов двигателей.

Двигатели постоянного тока

Двигатели постоянного тока широко используются в игрушках из-за их низкой стоимости и небольших размеров. Они идеально подходят для колесных роботов. На рис. 9.11 показаны такие двигатели, используемые для привода колес в нашей платформе. Эти двигатели оснащены встроенным редуктором для уменьшения числа оборотов в минуту (об/мин), что делает их подходящими для колесного робота.

Двигатели постоянного тока обычно включаются по схеме H-моста, что позволяет контролировать скорость и направление вращения.

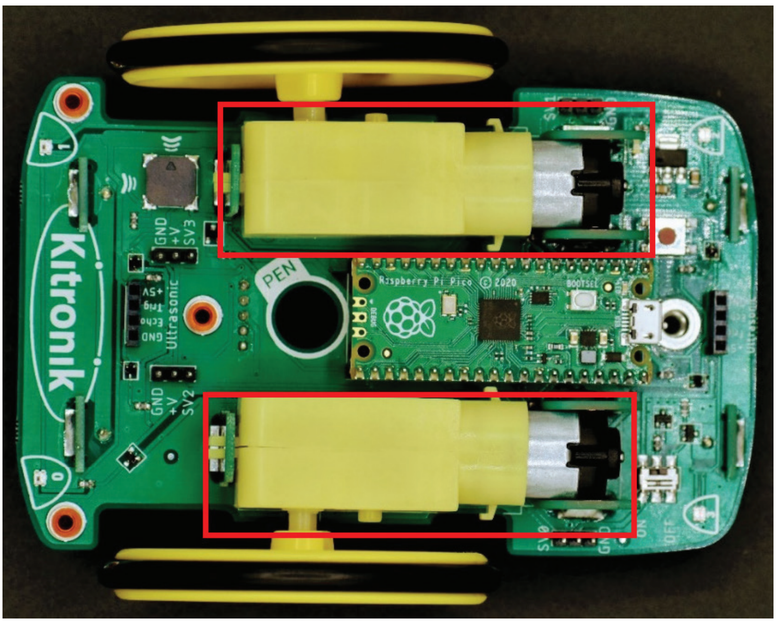


Рисунок 9.11. Двигатели постоянного тока на платформе робота

Шаговые двигатели

Шаговые двигатели идеально подходят для точного позиционирования. Например, они могут использоваться для приведения в действие механизмов линейного перемещения с помощью вала с резьбой[79] и установки исполнительного органа в требуемом положении. Поэтому шаговые двигатели вы можете встретить в таких применениях, как плоттеры или 3D-печать. Они состоят из пар катушек, на которые питание подается импульсами по очереди. На следующем рисунке два шаговых двигателя используются для управления игрушкой прохождения лабиринтов:

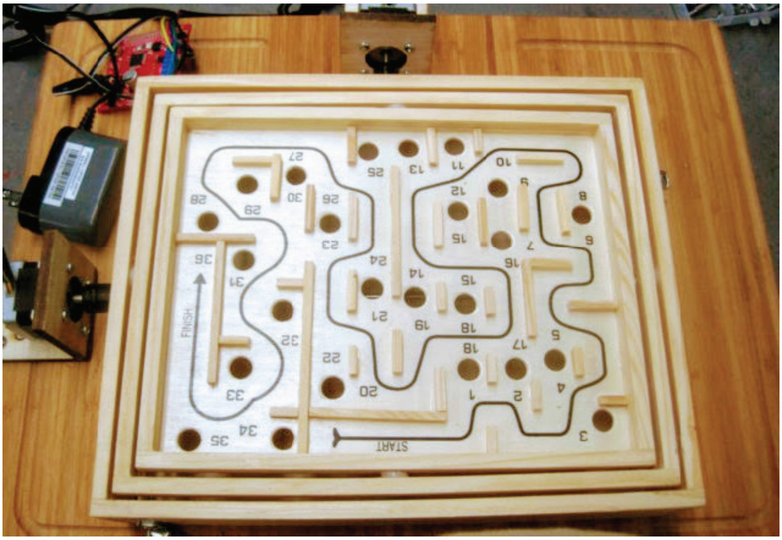


Рисунок 9.12. Робот для прохождения лабиринтов

Серводвигатели

Серводвигатели обычно управляются импульсом, длительность которого определяет угол поворота. Серводвигатели используются в таких конструкциях, как манипуляторы (показан на рис. 9.13), где импульсный сигнал применяется для удержания или высвобождения объекта:

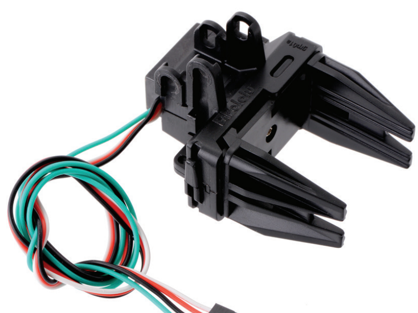


Рисунок 9.13. Серводвигатель с манипулятором-захватом[80]

[79] То есть червячной передачи. – *Прим. перев.*

[80] Используемый авторами серводвигатель с манипулятором можно найти отдельно по адресу **https://www.pololu.com/product/3551**. – *Прим. перев.*

Серводвигатель для этой цели поставляется с четвертым кабелем (голубой провод), который обеспечивает обратную связь по напряжению в зависимости от положения. Это помогает точнее скорректировать положение серводвигателя.

Робототехническая платформа, которую мы выбрали для этого проекта, может приводить в действие до четырех серводвигателей. Контакты для подключения серводвигателей выделены красными прямоугольниками на следующем рисунке:

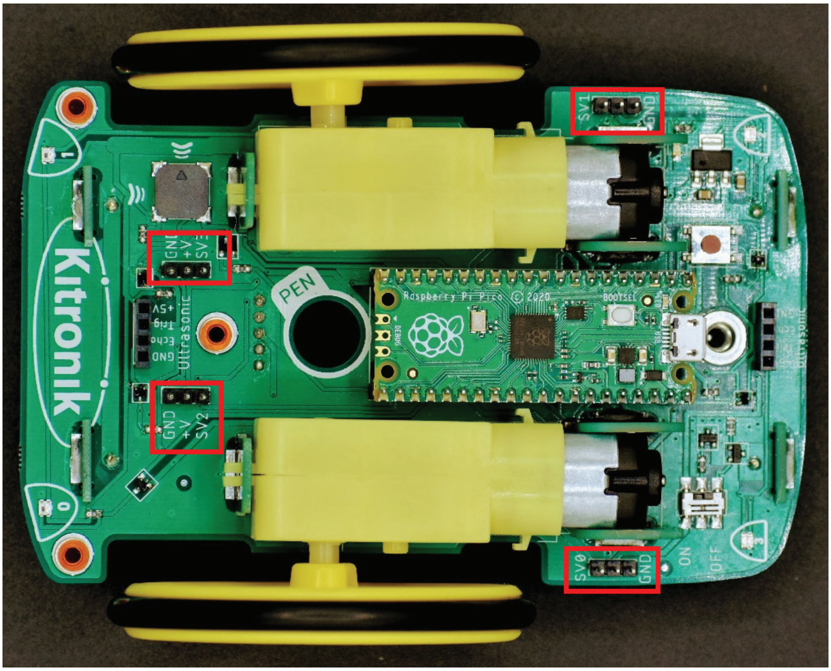


Рисунок 9.14. Контакты для подключения серводвигателей

Для более подробного ознакомления с различными двигателями и принципами их работы мы рекомендуем следующую статью: **https://learn.sparkfun.com/tutorials/motors-andselecting-the-right-one/all**.

Теперь обсудим управление двигателем для приведения робота в действие.

Управление двигателем постоянного тока

В этом разделе мы собираемся протестировать двигатель постоянного тока, используемый для привода робота, перемещая его вперед и назад в течение 10 секунд. Пример кода, обсуждаемый в данном разделе, доступен для загрузки вместе с этой главой в виде *dc_motor_test.py*.

Прежде чем мы начнем, убедитесь, что питание робота отключено (выключатель показан на рис. 9.7). Это делается для того, чтобы у нас не было сбежавшего робота. Конечно, для целей тестирования вы можете просто снять колеса робота.

Давайте начнем.

1. В среде IDE Thonny создайте файл с именем *main.py* и сохраните его в Pico.

2. Первым шагом является импорт необходимых модулей. Нам понадобится класс `KitronikPicoRobotBuggy` из библиотеки, которую мы установили ранее, а также модуль `time`:

```
from PicoAutonomousRobotics import KitronikPicoRobotBuggy
from time import sleep
```

3. Следующим шагом является инициализация объекта `buggy`, принадлежащего классу `KitronikPicoRobotBuggy`. Мы также инициализируем переменную состояния, именуемую `state`:

```
buggy = KitronikPicoRobotBuggy()
state = 0
```

4. Когда переменная состояния равна 0, двигатель будет перемещаться в прямом направлении. Для вращения двигателя используется метод `motorOn`. Метод принимает три аргумента, а именно: двигатель (левый «l» или правый «r»), направление (вперед «f» или назад «r») и скорость (значение от 0 до 100):

```
def MotorControl():
global state

if state == 0:
  buggy.motorOn("l","f",100)
  buggy.motorOn("r","f",100)
  state = 1
```

5. Для выключения двигателя используется метод `motorOff`. Нужно только указать двигатель, который необходимо выключить (левый «l» или правый «r»):

```
buggy.motorOff("l")
buggy.motorOff("r")
state = 0
```

6. Собрав все это вместе, мы имеем:

```
from PicoAutonomousRobotics import KitronikPicoRobotBuggy
from time import sleep

buggy = KitronikPicoRobotBuggy()
state = 0

def MotorControl():
  global state

  if state == 0:
```

```
        buggy.motorOn("l","f",100)
        buggy.motorOn("r","f",100)
        state = 1
    elif state == 1:
        buggy.motorOn("l","r",100)
        buggy.motorOn("r","r",100)
        state = 2
    else:
        buggy.motorOff("l")
        buggy.motorOff("r")
        state = 0

while True:
MotorControl()
sleep(10)
```

Запустите код на выполнение, нажав на кнопку **Run**, и двигатели должны вращаться вперед и назад каждые 10 секунд.

В следующем разделе мы обсудим управление серводвигателями.

Управление серводвигателем

В этом разделе мы обсудим управление серводвигателями через контакты, доступные на платформе Kitronik Robotics. Как упоминалось ранее, платформа поставляется с возможностью подключения до четырех двигателей (как показано на рис. 9.14).

Мы собираемся настроить управление манипулятором-захватом с помощью серводвигателя. Пример кода, обсуждаемый в данном разделе, доступен для загрузки вместе с этой главой как *servo_motor_test.py*.

Серводвигатель с манипулятором подключен к порту 2 платформы Kitronik Robotics, как показано на следующем рисунке:

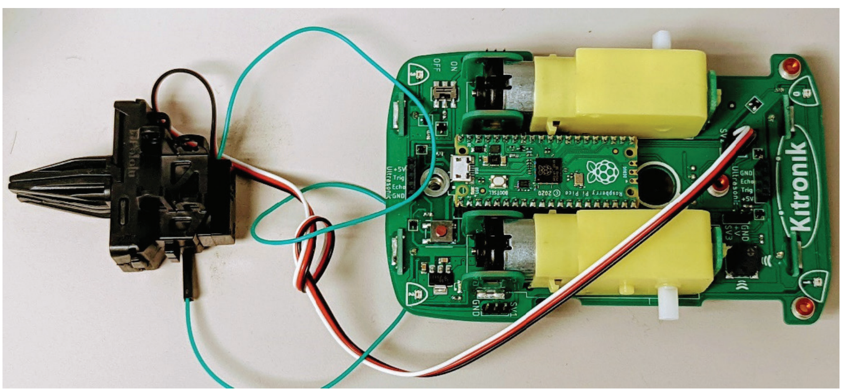


Рисунок 9.15. Управление манипулятором-захватом с помощью серводвигателя

Давайте обсудим код, необходимый для открытия и закрытия манипулятора-захвата.

1. В среде IDE Thonny создайте файл с именем *main.py* и сохраните его в своем Pico.
2. Первым шагом является импорт необходимых модулей. Нам понадобится класс `KitronikPicoRobotBuggy` из библиотеки, которую мы установили ранее, а также модуль `time`:

```
from PicoAutonomousRobotics import KitronikPicoRobotBuggy
from time import sleep
```

3. Следующим шагом является инициализация объекта `buggy`, принадлежащего классу `KitronikPicoRobotBuggy`:

```
buggy = KitronikPicoRobotBuggy()
```

4. Для поворота серводвигателя в положение 180 градусов (полностью закрытый захват) используем метод `goToPosition()`. Метод принимает два аргумента: номер порта серводвигателя (в данном случае серводвигатель подключен к порту 2) и угол поворота:

```
buggy.goToPosition(2,180)
sleep(5)
```

   На рисунке показан манипулятор в полностью закрытом положении:

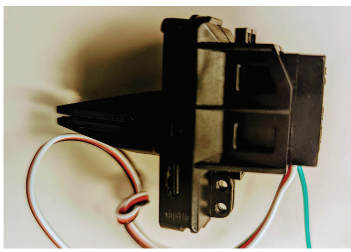


Рисунок 9.16. Захват в полностью закрытом положении

5. Чтобы полностью распахнуть захват, мы снова вызываем тот же метод, установив положение сервопривода 0 градусов:

```
buggy.goToPosition(2,0)
sleep(5)
```

   На рисунке показан захват в полностью распахнутом положении:

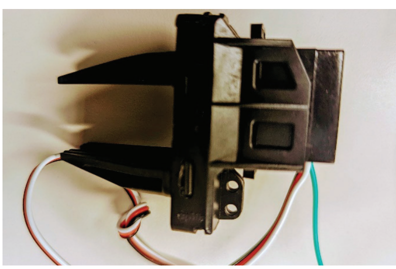


Рисунок 9.17. Захват в полностью распахнутом положении

6. Собрав все это вместе, имеем следующее:

```
from PicoAutonomousRobotics import KitronikPicoRobotBuggy
from time import sleep

buggy = KitronikPicoRobotBuggy()

while True:
  buggy.goToPosition(2,180)
  sleep(5)
  buggy.goToPosition(2,0)
  sleep(5)
```

Нажав на кнопку **Run** для выполнения кода, можем наблюдать, как захват закрывается и открывается каждые 5 секунд.

В следующем разделе мы протестируем датчики, входящие в комплект поставки Robotics kit.

Тестирование датчиков

В этом разделе мы протестируем ультразвуковой датчик расстояния и датчик для следования за линией, которые входят в комплект Kitronik Robotics. Начнем с тестирования ультразвукового датчика.

Ультразвуковой датчик

Ультразвуковые датчик расстояния обычно используется в системах обхода препятствий. Он использует принцип измерения времени распространения звука для расчета расстояния между объектами. Датчик передает звуковой сигнал на известной частоте в ультразвуковом диапазоне. Звуковой сигнал отражается от поверхности препятствий обратно к датчику. Время, прошедшее с момента передачи сигнала до его прихода обратно, используется для вычисления расстояния между датчиком и объектом. Так как

Скорость = *Расстояние* / *Время*,

следовательно, расстояние можно рассчитать по следующей формуле:

Расстояние = (*Скорость звука* × *Время*) / 2.

Деление на два необходимо, потому что звук дважды проходит между препятствием и датчиком.

Вы можете узнать об ультразвуковых датчиках для робототехники больше из этой публикации: **https://mirrobo.ru/micro/ultrazvukovye-datchiki**.

Ультразвуковой датчик может быть установлен как на передней, так и на задней панели робота. В этом примере мы установим его спереди, как показано на рисунке:

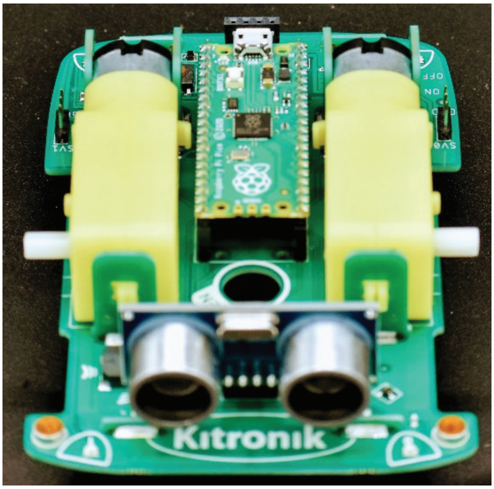


Рисунок 9.18. Ультразвуковой датчик, установленный на роботе

Вид на установленный датчик сверху показан на следующем рисунке:

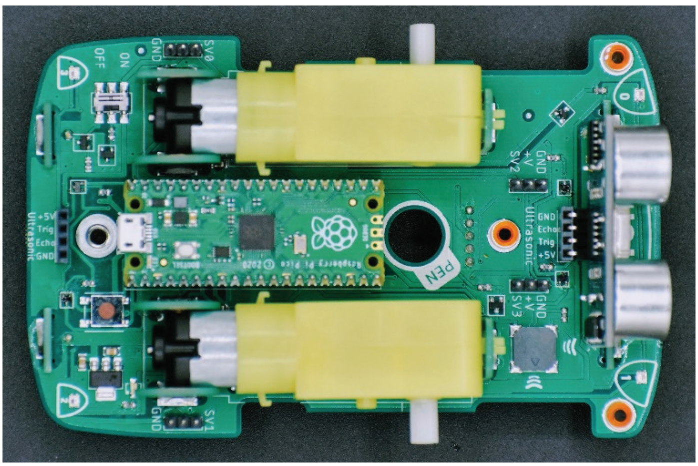


Рисунок 9.19. Вид установленного ультразвукового датчика сверху

Давайте рассмотрим пример кода для тестирования ультразвукового датчика. Пример кода, обсуждаемый в этом разделе, доступен для загрузки вместе с данной главой в виде файла *ultrasonic_test.py*.

1. В среде IDE Thonny создайте файл с именем *main.py* и сохраните его в Pico.

2. Первым шагом является импорт необходимых модулей. Нам понадобится класс `KitronikPicoRobotBuggy` из библиотеки, которую мы установили ранее, а также модуль `time`:

```
from PicoAutonomousRobotics import KitronikPicoRobotBuggy
from time import sleep
```

3. Следующим шагом является инициализация объекта `buggy`, принадлежащего классу `KitronikPicoRobotBuggy`:

```
buggy = KitronikPicoRobotBuggy()
```

4. Теперь мы можем получить расстояние в сантиметрах, используя метод `getDistance()` с указанием расположения датчика. Метод принимает «f» для датчика впереди и «r» для датчика сзади:

```
while True:
  frontDistance = buggy.getDistance("f")
  print(frontDistance)
  sleep(2)
```

5. Мы можем использовать информацию о расстоянии для управления роботом. Собрав все вместе, мы имеем следующее:

```
from PicoAutonomousRobotics import KitronikPicoRobotBuggy
from time import sleep

buggy = KitronikPicoRobotBuggy()

while True:
  frontDistance = buggy.getDistance("f")
  print(frontDistance)
  sleep(2)
```

6. После запуска кода с помощью кнопки **Run**, направляя робота на различные объекты или используя руку для имитации препятствия, мы должны увидеть следующий вывод:

```
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
 17.06425
 15.41785
 42.0861
 109.8457
 13.66855
 5.3165
 187.9983
```

Рисунок 9.20. Выходной сигнал ультразвукового датчика расстояния

На рисунке расстояние до различных объектов измеряется в сантиметрах. Выходной сигнал датчика может быть зашумлен, потому иногда отображаются очень большие значения, как, например, 109 и 187 см в данном случае. Это происходит, например, при размещении пальцев прямо рядом с датчиком[81].

[81] А также если препятствие небольшое и не перекрывает всю пространственную диаграмму направленности излучателя/приемника – в этом случае датчик с запаздыванием ловит отражения от более удаленных объектов, «вклинивающиеся» между достоверными измерениями. Добавляет вероятность ложных срабатываний и наклон поверхности препятствия – в этом случае приемник может ловить вторичные сигналы от объектов сбоку за счет звуковой волны, отраженной от препятствия в сторону. – *Прим. перев.*

Поэтому важно отфильтровывать такие значения для надежной навигации робота.

Далее протестируем датчики для следования за линией.

Датчики отслеживания линии

В этом разделе мы обсудим тестирование датчиков для отслеживания линии. Как следует из названия, эти датчики используются для того, чтобы заставить робота следовать вдоль линии. Линия может быть темной на более светлом фоне или наоборот. Датчик обычно состоит из инфракрасного светодиода и фотодиода и использует для управления роботом свойство более светлых поверхностей отражать больше света, чем более темных.

Давайте рассмотрим сценарий, в котором мы заставляем робота следовать по темной линии на более светлом фоне. Наш робот, следующий за линией, имеет три датчика, а именно: левый, правый и центральный. Идея состоит в том, чтобы управлять роботом таким образом, чтобы центральный датчик оставался строго над темной линией. При этом левый и правый датчики находятся на более светлой поверхности, чем центральный. При таких условиях в действие приводятся оба двигателя в прямом направлении.

Робот использует дифференциальный механизм рулевого управления, при котором один из двигателей выключается, чтобы повернуть робота в его направлении. Допустим, левый датчик попал на темную линию. Мы выключаем левый мотор и поворачиваем влево до тех пор, пока левый датчик не окажется снова на более светлой поверхности. Вы можете узнать больше о датчиках слежения за линией на сайте **https://bit.ly/3LSF49u**[82].

Датчики отслеживания линии установлены на нижней стороне робота, как показано на следующем рисунке:

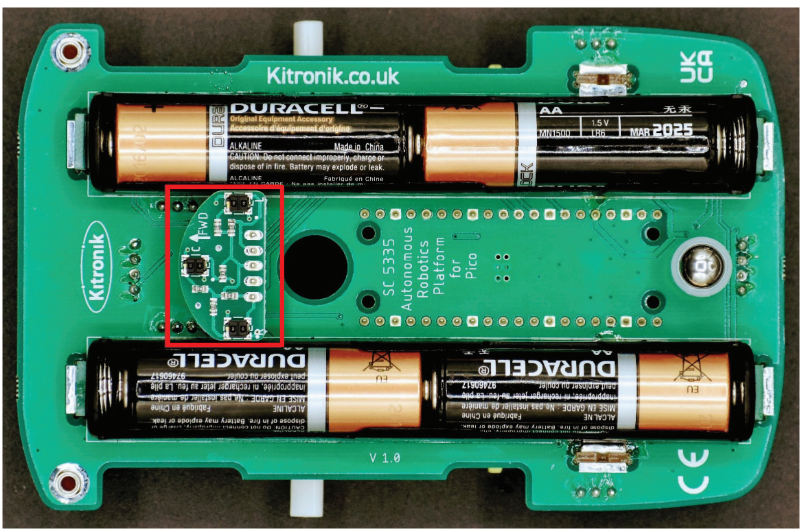


Рисунок 9.21. Установка датчиков линии

[82] Пересказ этой статьи на русском языке см. «Робот, ездящий по линии под управлением Arduino» на сайте **wiki.amperka.ru** (**http://wiki.amperka.ru/робототехника:робот-с-датчиками-линии-на-arduino**). – *Прим. перев.*

Рассмотрим код, необходимый для тестирования датчиков следования за линией. Он доступен для скачивания вместе с этой главой в файле *line_following_sensor.py*.

1. В среде IDE Thonny создайте файл с именем *main.py* и сохраните его в Pico.

2. Первым шагом является импорт необходимых модулей. Нам понадобится класс `KitronikPicoRobotBuggy` из библиотеки, которую мы установили ранее, а также модуль `time`:

```
from PicoAutonomousRobotics import KitronikPicoRobotBuggy
from time import sleep
```

3. Следующим шагом является инициализация объекта `buggy`, принадлежащего классу `KitronikPicoRobotBuggy`:

```
buggy = KitronikPicoRobotBuggy()
```

4. Теперь мы можем извлекать исходные значения датчиков отслеживания линии следующим образом:

```
while True:
  print ("L", buggy.getRawLFValue("l"), " R", buggy .getRawLFValue("r"), " C",
buggy.getRawLFValue("c"))
  sleep(1)
```

5. Собрав все это вместе, имеем следующее:

```
from PicoAutonomousRobotics import KitronikPicoRobotBuggy
from time import sleep
buggy = KitronikPicoRobotBuggy()

while True:
  print ("L" , buggy.getRawLFValue("l"), " R", b uggy.getRawLFValue("r"), " C",
buggy.getRawLFValue("c"))
  sleep(1)
```

6. При выполнении этого примера кода для робота, размещенного на поверхности с линией, мы получим следующий результат:

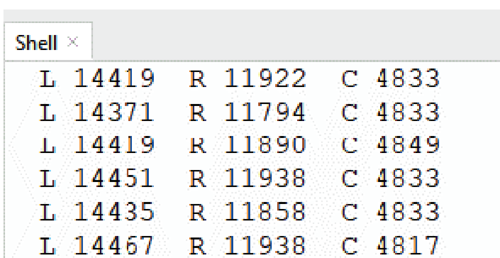


Рисунок 9.22. Выход датчиков отслеживания линии

На рис. 9.22 показан необработанный выходной сигнал левого, правого и центрального датчиков. Центральный датчик («С») показывает значение, заметно меньшее, чем левый и правый датчики. Это означает, что левый и правый датчики находятся над более светлой поверхностью фона, в то время как центральный датчик находится над темной поверхностью линии.

Необработанные значения выходного сигнала датчиков можно использовать для управления роботом, чтобы он следовал за темной линией на светлом фоне. В следующем разделе мы обсудим тестирование робота.

Тестирование робота

Теперь, когда мы протестировали компоненты нашего робота, пришло время попробовать его на практике.

Первым шагом является установка колес робота (как показано на следующем рисунке):

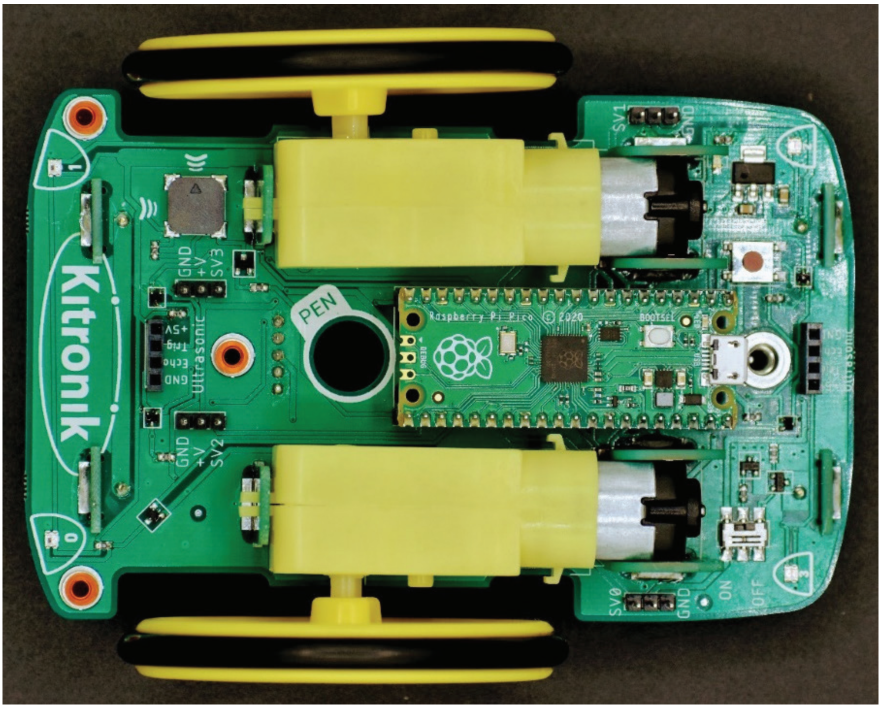


Рисунок 9.23. Установка колес на робота

Примеры кода для роботов, следующих за линией и избегающих препятствий, доступны для загрузки вместе с этой главой в файле *line_following.py* и *obstacle_avoidance.py*. Мы протестировали робота, следующего за линией, с помощью специального коврика от Kitronik (**https://bit.ly/3GAn0xM**), показанного на следующем рисунке:

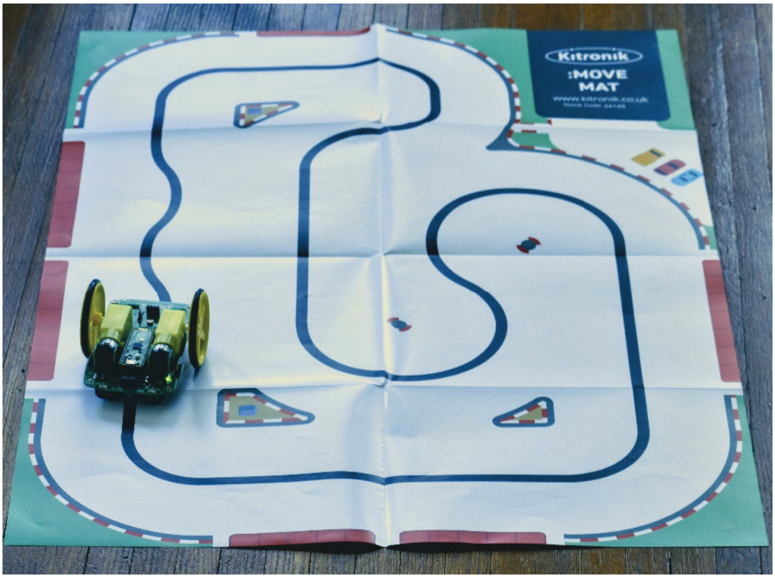


Рисунок 9.24. Робот, следующий за линией на поверхности

Соревнования по робототехнике

Куда пойти дальше? Если вас интересует создание роботов, мы рекомендуем ознакомиться с конкурсами, организованными ближайшим клубом робототехники. Существуют и другие типы роботов, например роботы, разгадывающие лабиринты, и роботы, занимающиеся борьбой сумо. Некоторые клубы даже организуют соревнования по борьбе сумо среди роботов. Наше любимое соревнование – конкурс Micromouse, в котором робот должен разгадать лабиринт и найти кратчайший путь домой. Вы можете узнать больше о конкурсах Micromouse на сайте **https://en.wikipedia.org/wiki/Micromouse**. В Соединенных Штатах профессиональная техническая ассоциация инженеров IEEE проводит региональные конкурсы Micromouse на студенческих мероприятиях.

Итоги

В этой главе мы обсуждали создание робота с использованием готового комплекта Kitronik Robotics. Мы обсудили отдельные компоненты робота, включая тестирование светодиодов, выбор двигателя и управление двигателями постоянного тока и серводвигателями. Мы рассмотрели драйверы для управления двигателями и подключения датчиков, а также кратко протестировали компоненты робота, следующего за линией, и робота, избегающего препятствий.

В следующей главе разберем создание приложений TinyML на Pico.

Глава 10

Знакомство с приложениями TinyML

В предыдущих главах мы рассмотрели проекты от создания метеостанции до робота, следующего за линией. В этой главе обсудим разработку приложений TinyML с использованием Pico. Начнем с представления TinyML и его потенциальных применений. Затем обсудим пример, который включает в себя классификацию изображений.

В этой главе мы рассмотрим следующие темы:

- представляем TinyML;
- распознавание ключевых слов в звуковых образцах;
- классификация изображений;
- разработка периферийных устройств.

Технические требования

Для этой главы требуется следующее оборудование:

- Raspberry Pi Pico (**https://bit.ly/3AJtoAf**);
- модуль камеры Arducam (**https://amzn.to/32KPqVN**[83]);
- плата микрофона Adafruit PDM (**https://bit.ly/3tGCzRt**[84]).

Дополнительное оборудование

Мы также рекомендуем следующее дополнительное оборудование:

- модуль Arduino RP2040 Connect (**https://bit.ly/2YZClrY**[85]);
- плата Arducam Pico4ML TinyML (**https://bit.ly/3DfKJTW**[86]);

[83] Модуль камеры Arducam OV5642 можно приобрести в российском интернет-магазине **amperkot.ru** (напрямую или через OZON), а также на AliExpress. Отметим, что типов камер и модулей Arducam великое множество, не ошибитесь при выборе: авторы ориентируются на самый простой и дешевый вариант. – *Прим. перев.*

[84] Микрофон Adafruit PDM MEMS можно приобрести в российских интернет-магазинах, например по адресу **https://shopozz.ru/items/253444612481-adafruit-pdm-mems-microphone-breakout-3492**. – *Прим. перев.*

[85] Модуль Arduino Nano RP2040 Connect (аналог Raspberry Pi Pico в форм-факторе Arduino Nano) можно приобрести в основных российских интернет-магазинах электронных комплектующих (**chipdip.ru**, **iarduino.ru**, **amperka.ru**, **tmelectronics.ru** и др.), а также на AliExpress. – *Прим. перев.*

[86] Можно заказать из США по ссылке **https://shopozz.ru/items/175375315676-arducam-pico4ml-tinyml-dev-kit-rp2040-board-w-hm01b0-camera-lcd-screen-new**. – *Прим. перев.*

- набор для машинного обучения Arduino Tiny Machine Learning (**https://bit.ly/3M7m4nK**[87]).

Примеры кода из этой главы можно найти в репозитории GitHub этой книги по адресу **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIYWorkshop/tree/main/chapter_10**.

Видеоролики с кодом в действии для данной главы можно посмотреть по адресу **https://bit.ly/3yklorx**.

Теперь давайте кратко ознакомимся с TinyML.

Введение в TinyML

Что такое TinyML? **TinyML** расшифровывается как **Tiny Machine Learning** (*упрощенное машинное обучение*) – концепция машинного обучения, объединяющая облегченные и оптимизированные варианты, ориентированные на создание интеллектуальных IoT-устройств. Это быстро растущая область, в которой инструменты машинного обучения используются на аппаратных средствах с ограниченными ресурсами, таких как микроконтроллер RP2040, для интерпретации данных датчиков. Ограничения относятся к небольшому объему памяти и вычислительной мощности микроконтроллеров по сравнению с сетевым сервером, имеющим огромную вычислительную мощность и встроенный графический процессор.

TinyML позволяет интерпретировать данные на микроконтроллере, работающем от батарейки-монетки. Устройство, которое само может интерпретировать данные датчиков с помощью инструментов TinyML, вместо того чтобы передавать данные в облако, называется **фронтальным устройством** (*edge device*).

Давайте проиллюстрируем эту концепцию на примере. На схеме ниже (рис. 10.1) показан поток данных в типичном IoT-приложении на основе устройства, собирающего данные с различных датчиков и отправляющего их в облако через шлюз (gateway). Обработка происходит в облаке, и сервер, работающий в облаке, отправляет через шлюз команду включать или выключать устройство:

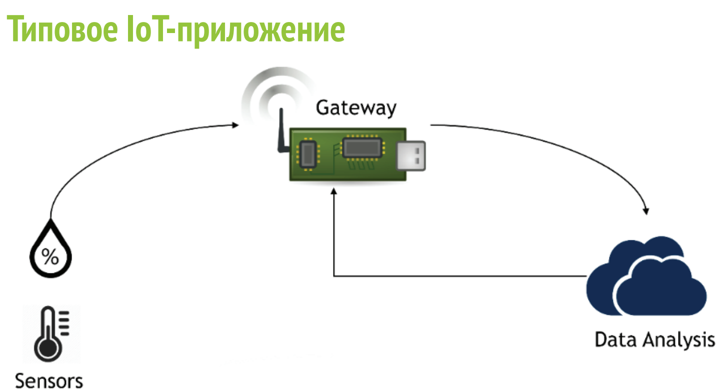


Рисунок 10.1. Типичное приложение для интернета вещей

[87] В России можно приобрести, например, по адресу **https://tmelectronics.ru/product/AKX00028**. – *Прим. перев.*

При наличии фронтальных устройств этот поток данных замыкается локально на уровне шлюза. Некоторые данные все еще поступают в облако, но мы загружаем только те события, которые были обнаружены как аномальные. Эти события включают в себя любые аномалии, такие как утечки в системе, наличие или отсутствие объектов, прогнозирование отказов компонентов по известным шумам и многое другое. Это позволяет вам сразу реагировать на аномалию, вместо того чтобы ждать ответа от облака:

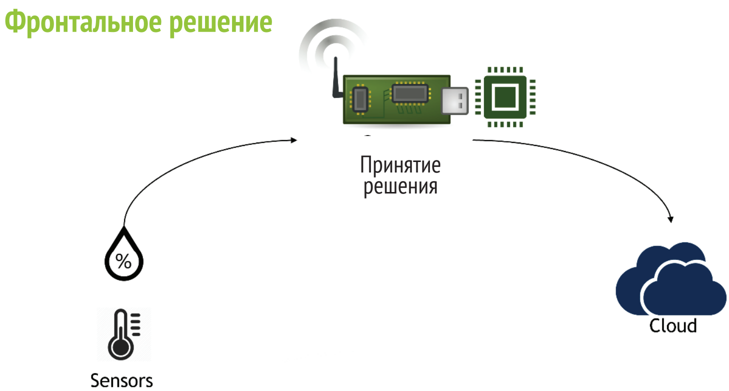


Рисунок 10.2. Приложение на основе фронтальных устройств

Фронтальные устройства обладают определенными преимуществами:

- в случае устройств с сотовой связью фронтальные устройства могут снизить накладные расходы на перекачку данных. Это связано с тем, что мы загружаем данные только при обнаружении аномалии, а не передаем каждый отдельный результат измерения;
- фронтальные устройства могут помочь продлить срок службы батареи, поскольку их можно оставлять в состоянии длительного ожидания, пока не будет обнаружено определенное событие.

Фронтальные устройства обнаруживают аномалии, используя данные, собранные в полевых условиях. В результате фронтальным устройствам требуются некоторые предварительные наборы данных для разработки конкретного продукта. В последнее время наблюдается прогресс накопления подобных данных для решения конкретных проблем, таких, например, как износ подшипников.

Назовем некоторые примеры приложений TinyML:

- автоматический визуальный осмотр: проверка на наличие дефектов в производстве;
- прогнозирующее техническое обслуживание: использование акселерометра для обнаружения необычных вибраций и скорой поломки;
- мониторинг флоры/фауны: устройства с поддержкой TinyML можно использовать для обнаружения звуков транспортных средств или бензопил, чтобы предотвратить браконьерство в лесах и в отношении диких животных;

- классификация изображений: наблюдение за дикой природой с помощью камер-ловушек.

> **Умные датчики**
>
> С появлением мощных процессоров производители полупроводников начали разрабатывать датчики с ядрами ML. Это позволяет интегрировать датчик сверхнизкого потребления, который может обнаруживать определенные события. Вот ссылка на инклинометр[88] с интегрированным ML: **https://www.st.com/en/mems-and-sensors/iis2iclx.html**.

Прежде чем перейти к примеру приложения TinyML, давайте кратко обсудим, как TinyML относится к области искусственного интеллекта (ИИ). ИИ означает, что машина может принимать решения, как человек. TinyML – это небольшое подмножество области машинного обучения (ML), которое, в свою очередь, является подмножеством области ИИ. ML относится к обучению машины делать выводы из собранных данных и самостоятельно принимать решения или давать прогнозы.

В области ML существуют различные наборы инструментов, и один из них называется **глубокое обучение** (*deep learning*). Это техника, которая смоделирована по образцу человеческого мозга, где информация обрабатывается сетями из особым образом устроенных нервных клеток – нейронов. Подобная сеть называется нейронной сетью.

Нейронная сеть состоит из взаимосвязанной системы узлов, как и клетки мозга, называемых нейронами. Нейроны организованы во входной, выходной и один или несколько промежуточных слоев, как показано на следующей схеме. Каждому узлу нейронной сети присвоен соответствующий вес:

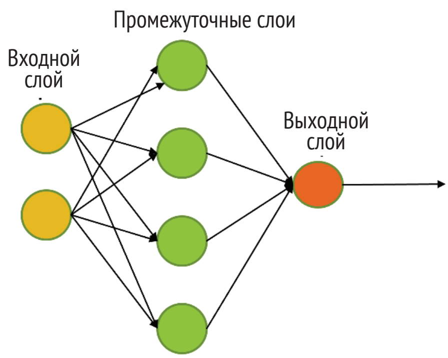


Рисунок 10.3. Модель нейронной сети

Нейронные сети обычно требуют огромных вычислительных мощностей, и, как говорилось ранее, чаще выполняются на сервере. Инструменты TinyML позволяют запускать модифицированную нейронную сеть на микроконтроллере RP2040.

[88] Инклинометр – датчик угла наклона. – *Прим. перев.*

В пример можно привести классификацию изображений, когда мы пытаемся определить, принадлежит ли изображение к определенному классу. В этом случае нейронная сеть принимает изображение в качестве входных данных и выдает вероятность того, что изображение принадлежит к выбранному классу.

Вы можете узнать больше о нейронных сетях по адресу **https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414**.

На следующей схеме показаны различные этапы развертывания нейронной сети на микроконтроллере:

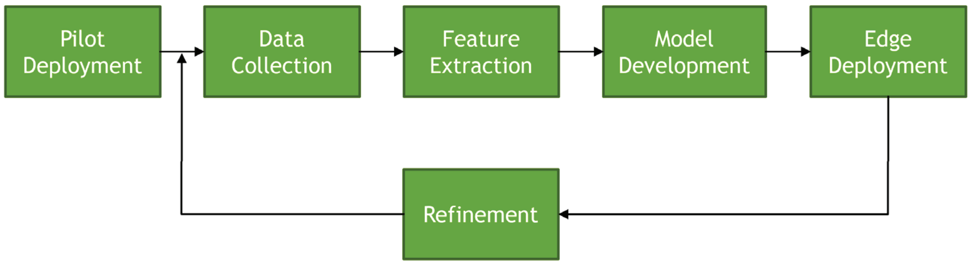


Рисунок 10.4. Этапы развертывания нейронной сети на микроконтроллере

Давайте рассмотрим эти этапы более подробно.

1. Первый шаг – это сбор данных (**Data Collection**). Он достигается путем непосредственного сбора данных с устройств, используемых в приложении, или предварительного пилотного развертывания (**Pilot Deployment**) специальных устройств для сбора данных.
2. Следующий шаг называется извлечением свойств (**Feature Extraction**). На этом этапе мы определяем особенности или аномалии. Если человек может обнаружить особенность или аномалию, просмотрев данные, нейронную сеть можно обучить делать то же самое.
3. Далее начинается развертывание модели (**Model Deployment**). Первый этап – обучение нейронной сети с использованием собранных данных. Собранные данные разделяются на наборы данных для обучения и проверки. После обучения функционирование нейронной сети проверяется с использованием набора данных для проверки. Параметры нейронной сети настраиваются в течение нескольких циклов обучения.
4. Как только нейронная сеть будет готова, мы преобразуем ее в формат, подходящий для микроконтроллера. Этот процесс называется *квантованием* (quantization). Это происходит потому, что веса в нейронной сети заданы числами с плавающей запятой. Во время квантования веса преобразуются в целые числа. Это ускоряет производительность нейронной сети, работающей на микроконтроллере.
5. После квантования модель разворачивается на микроконтроллере фронтального устройства (этап **Edge Deployment**) и измеряется ее производительность.

6. Модель дополнительно настраивается (**Refinement**) на основе измерений производительности. Это связано с падением производительности из-за квантования.

> **Онлайн-курс по TinyML**
>
> Если вы заинтересованы в продолжении обучения в этой области, мы рекомендуем следующий курс от Гарвардского университета: **https://www.edx.org/professional-certificate/harvardx-tiny-machinelearnin**.

В этой главе мы продемонстрируем два примера развертывания нейронной сети для классификации изображений и распознавания ключевых слов. В следующем разделе мы кратко обсудим набор для разработки аппаратного обеспечения, предназначенный для приложений TinyML.

Представляем Arducam Pico4ML

Arducam Pico4ML – это плата на основе микроконтроллера RP2040 от ArduCAM:

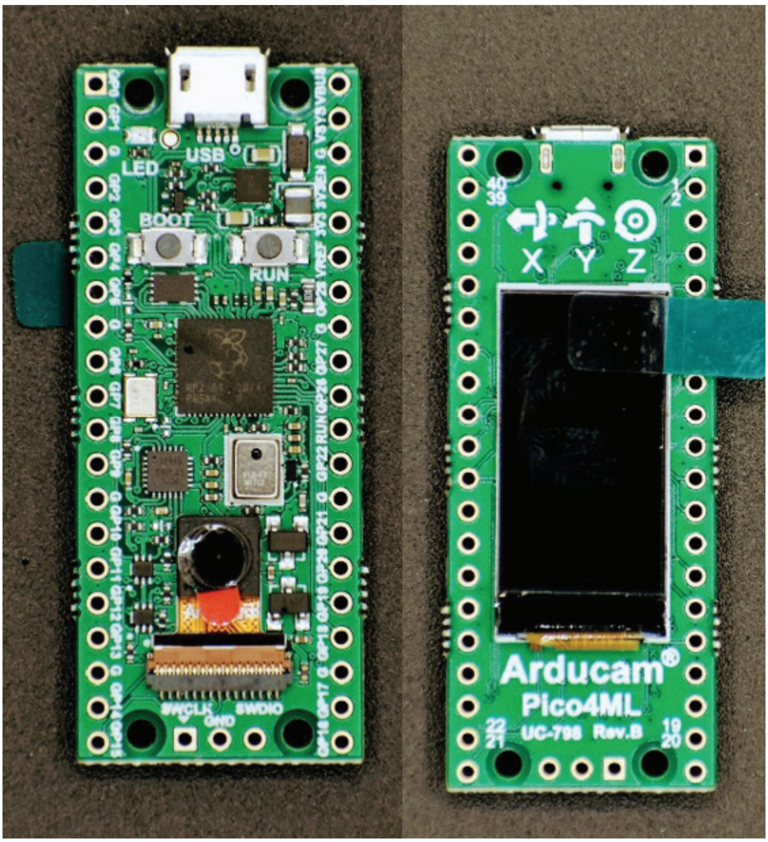


Рисунок 10.5. Плата Pico4ML

Плата оснащена следующими датчиками:

- модуль камеры HiMax, поддерживающий разрешение до 320×240;
- микрофон, снабженный цифровым выходом с модуляцией плотности импульсов (PDM);
- 9-осевой инерциальный измерительный блок (**Inertial Measurement Unit**, **IMU**) ICM-20948. IMU представляет собой комбинацию гироскопа, акселерометра и компаса. Его можно использовать в приложениях распознавания жестов.

Эти три датчика позволяют использовать плату Pico4ML в приложениях классификации изображений, определения ключевых слов и распознавания жестов соответственно. Производитель платы предоставляет примеры, которые помогут пользователям начать работу. Их можно найти по адресу **https://github.com/ArduCAM/RPI-Pico-Cam**.

Примеры, рассматриваемые далее, могут быть выполнены с использованием платы Pico4ML. После них мы представим приложение для классификации изображений.

Распознавание ключевых слов в звуковых образцах

В этом разделе мы обсудим распознавание ключевых слов в образцах звука (аудиосемплах) с использованием Pico. Это похоже на голосовых помощников, разработанных многими технологическими компаниями, таких как Google Assistant, Apple Siri и другими, которые реагируют на ключевое слово активации, например «Привет, Алекса». Мы будем использовать платформу Edge Impulse (**https://edgeimpulse.com**) для работы над этим примером. Чтобы следовать изложению примера, зарегистрируйтесь в учетной записи разработчика на Edge Impulse.

О платформе Edge Impulse

Платформа Edge Impulse позволяет разрабатывать приложения TinyML и развертывать их на микроконтроллере. Edge Impulse – интерактивный инструмент, который позволяет разрабатывать приложения без предварительных знаний о машинном обучении.

Если вы заинтересованы в том, чтобы стать сертифицированным разработчиком, мы рекомендуем эти курсы от Edge Impulse на Coursera: **https://www.coursera.org/edgeimpulse**.

Мы работали над этим примером, используя в качестве руководства следующий учебник: **https://docs.edgeimpulse.com/docs/tutorials/audio-classification**.

Есть также наш проект, который вы можете клонировать и следить за изменениями. Его можно найти по адресу **https://studio.edgeimpulse.com/public/48695/latest**.

Кнопку **Clone this project** (*Клонировать этот проект*) можно найти в правом верхнем углу, как показано на следующем скриншоте. Для клонирования проекта вам понадобится учетная запись разработчика, зарегистрированная вами ранее.

Клонирование проекта в вашу учетную запись займет несколько минут:

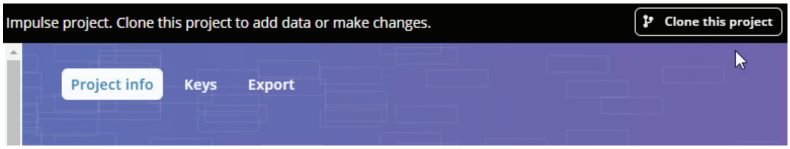


Рисунок 10.6. Клонирование проекта

После завершения должно отобразиться следующее сообщение:

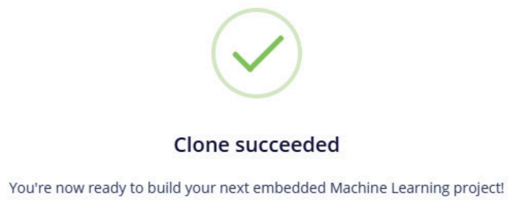


Рисунок 10.7. Клонирование выполнено успешно

Давайте рассмотрим шаги по разработке примера распознавания ключевых слов с использованием Edge Impulse.

1. После запуска проекта панель мониторинга должна выглядеть примерно так:

Рисунок 10.8. Панель мониторинга проекта для примера распознавания ключевых слов

2. Первым шагом в процессе разработки является сбор данных (в соответствии с рис. 10.3). Нажмите на кнопку **LET'S COLLECT SOME DATA** (*Давайте соберем некоторые данные*), как показано на рис. 10.8.

3. Вам будут представлены различные варианты сбора аудиоданных для нашего приложения.

   Самый простой способ сбора данных – с помощью смартфона или микрофона:

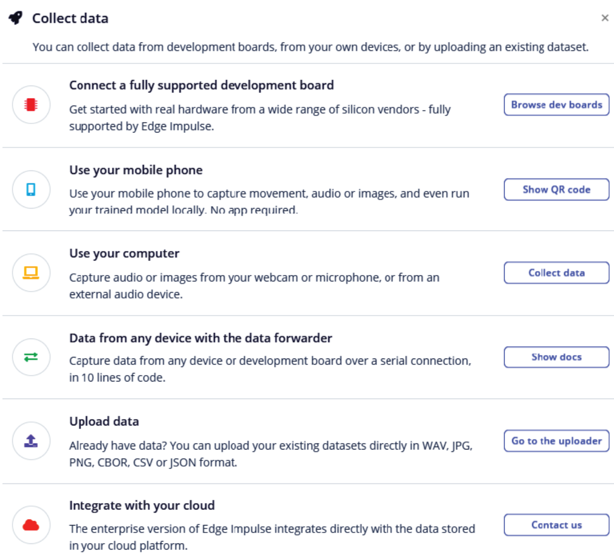


Рисунок 10.9. Варианты сбора данных

4. Например, когда вы нажимаете **Use your computer** (*Использовать свой компьютер*), вы должны иметь возможность собирать и помечать звуковые образцы со своего рабочего стола:

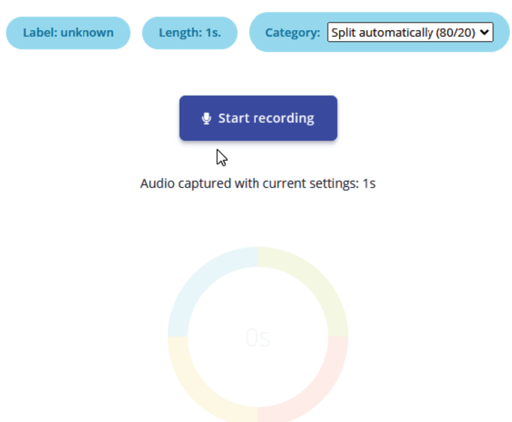


Рисунок 10.10. Сбор аудиосемплов с помощью компьютера

5. Для этого проекта мы использовали набор аудиоматериалов, доступный по адресу **https://cdn.edgeimpulse.com/datasets/keywords2.zip**. Набор данных содержит звуковые образцы, относящиеся к четырем классам: слова «**yes**», «**no**», неизвестные образцы **других слов** и **фоновый шум**.

6. Теперь, когда мы загрузили аудиосемплы в наш проект и пометили их, нам нужно их пометить, чтобы задать класс каждого образца. На следующем рисунке показаны аудиосемплы и соответствующие им метки:

SAMPLE NAME	LABEL	ADDED	LENGTH	
yes.ffb86d3c_nohash_0.wav....	yes	Today, 15:00:30	1s	⋮
yes.ff63ab0b_nohash_0.wav....	yes	Today, 15:00:30	1s	⋮

Рисунок 10.11. Наборы аудиоматериалов, загруженные в Edge Impulse

7. Следующий шаг – создать **impulse**[89]. Impulse получает звуковой образец, обрабатывает его и извлекает характерные признаки, чтобы определить, принадлежит ли образец звука к одному из четырех классов в нашем наборе аудиоданных. Impulse можно создать на вкладке **Create impulse** слева. Затем нажмите на блок **Add a Time series data** (*Добавить временной ряд данных*). Перед сохранением Impulse необходимо добавить блок обработки звука **Audio (MFCC)** и блок классификации **Classification (Keras)**:

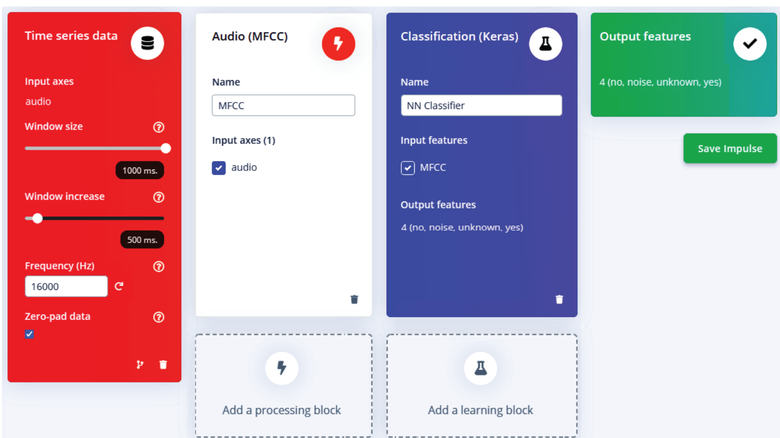


Рисунок 10.12. Создание Impulse для распознавания ключевых слов

8. Теперь давайте перейдем на вкладку **MFCC**[90] слева. В этом проекте распознавание ключевых слов может быть выполнено с помощью спектрограммы. Спектрограмма – это наглядное представление всех частот в

[89] Так в терминологии сервиса Edge Impulse называется обработчик аудиосемплов с заданными параметрами. – *Прим. перев.*

[90] MFCC (Mel-frequency cepstral coefficients) – сложный алгоритм анализа звуковых фрагментов с целью выделения характерных частот для распознавания речи. Подробнее об MFCC на русском языке см. **https://habr.com/ru/post/140828/**. – *Прим. перев.*

звуковом образце. Спектрограмма MFCC выделяет частоты, которые часто встречаются в человеческой речи. На данный момент мы оставим параметры по умолчанию и сохраним их:

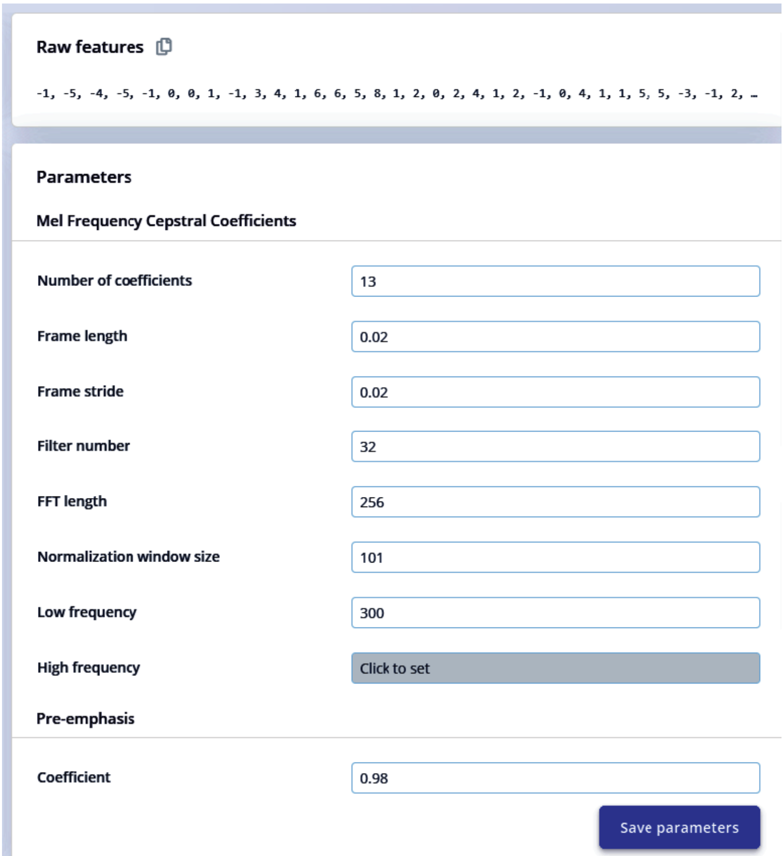


Рисунок 10.13. Сохранение параметров MFCC

9. Кроме всего прочего, можно увидеть производительность блока обработки MFCC на микроконтроллере RP2040:

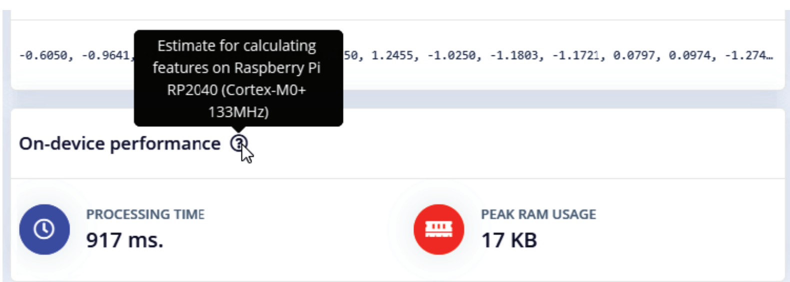


Рисунок 10.14. Производительность обработки на микроконтроллере RP2040

10. Теперь перейдите в раздел **Generate features** (*Генерация признаков*) и нажмите кнопку с таким названием:

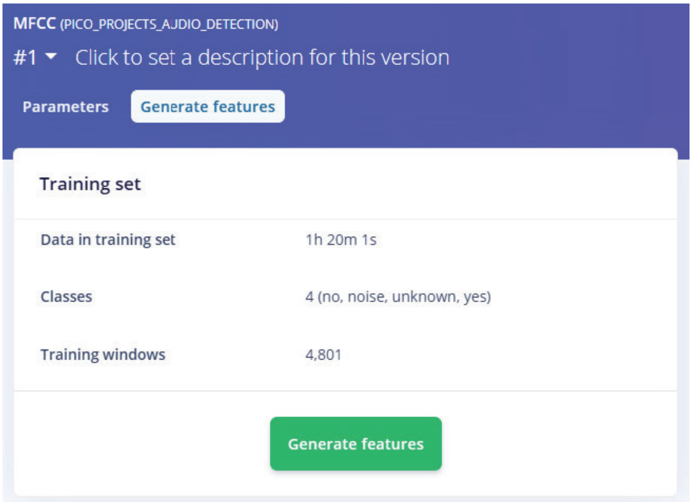


Рисунок 10.15. Генерация признаков

11. После генерации признаков вы можете визуализировать результат классификации вашего набора данных, как показано на рисунке ниже. Это помогает определить, можно ли запускать алгоритмы машинного обучения на своих данных. На рис. 10.16 каждый класс представлен цветом. Вы можете заметить, что выборки, принадлежащие к четырем классам, четко различаются между собой. Это дает уверенность в том, что мы сможем распознать ключевые слова «*yes*» или «*no*»:

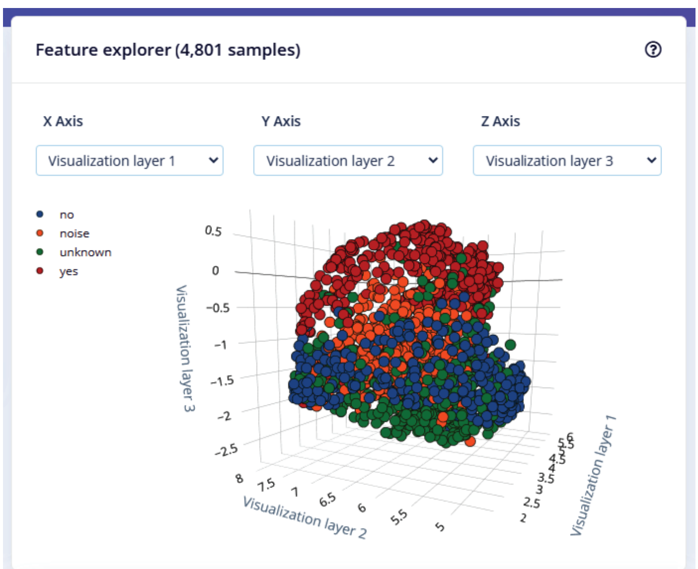


Рисунок 10.16. Результат классификации признаков

12. Далее посмотрим на классификатор нейронной сети, где обучаем нашу модель, используя параметры по умолчанию и текущий набор данных:

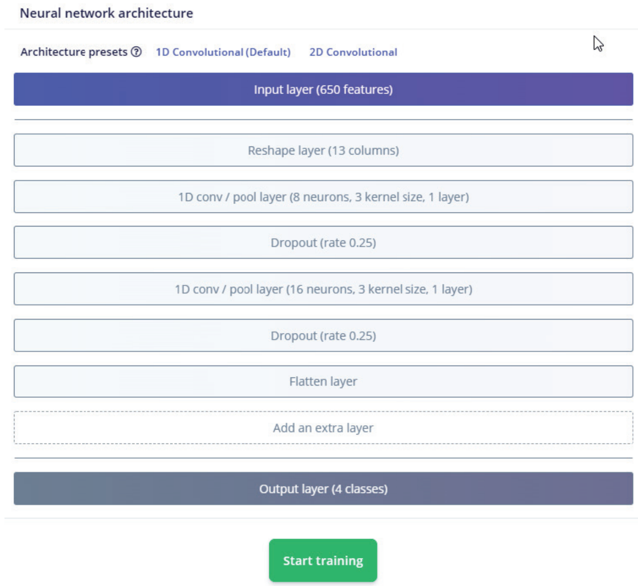
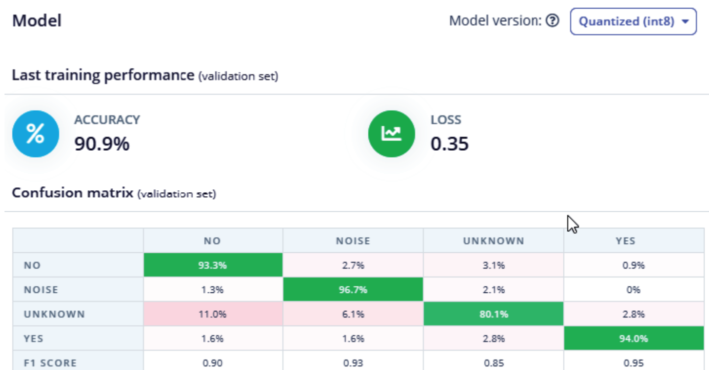


Рисунок 10.17. Начало обучения

13. Обучение может занять некоторое время, но как только оно будет завершено, мы сможем наблюдать за результатами. Результат обучения должен выглядеть примерно следующим образом:



	NO	NOISE	UNKNOWN	YES
NO	93.3%	2.7%	3.1%	0.9%
NOISE	1.3%	96.7%	2.1%	0%
UNKNOWN	11.0%	6.1%	80.1%	2.8%
YES	1.6%	1.6%	2.8%	94.0%
F1 SCORE	0.90	0.93	0.85	0.95

Рисунок 10.18. Результаты обучения

На рисунке выше показана матрица ошибок (confusion matrix). Матрица ошибок предоставляет информацию об эффективности нашей нейронной сети относительно представленного набора данных. В таблице приведена статистика эффективности модели с точки зрения идентификации аудиосемпла, принадлежащего к каждому классу. В таблице также указана величина показателя F1, используемого для измерения эффективности модели без необходимости запускать ее на Pico. Оценка F1 может находиться в диапазоне от 0 до 1. Более высокий балл указывает на лучшую эффективность модели.

14. Далее давайте посмотрим на вкладку **EON Tuner** с левой стороны. Тюнер автоматически выбирает наиболее производительную архитектуру в зависимости от целевого аппаратного приложения. Для данного проекта целевым оборудованием является Raspberry Pi RP2040. Мы можем установить его, нажав на значок шестеренки на вкладке **Target**:

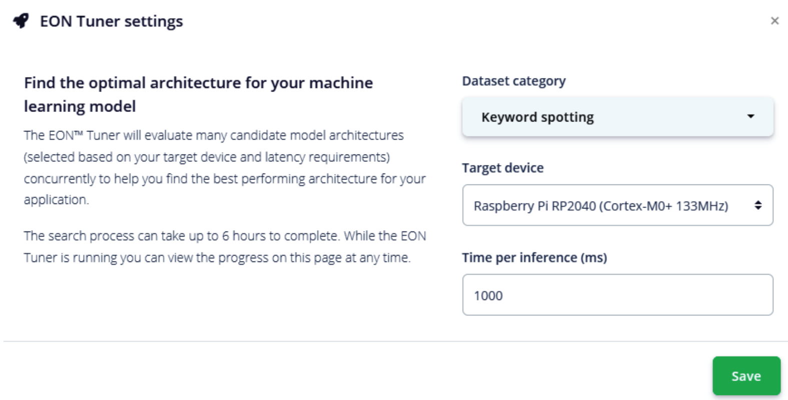


Рисунок 10.19. Установка целевого оборудования на Raspberry Pi RP2040

15. Как только оборудование задано, мы можем запустить EON Tuner, который выбирает наилучшую архитектуру для нашего приложения. Это может занять некоторое время:

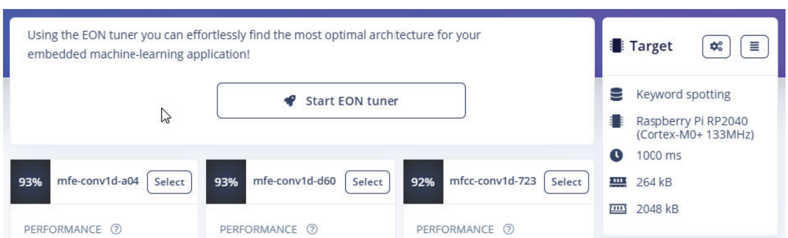
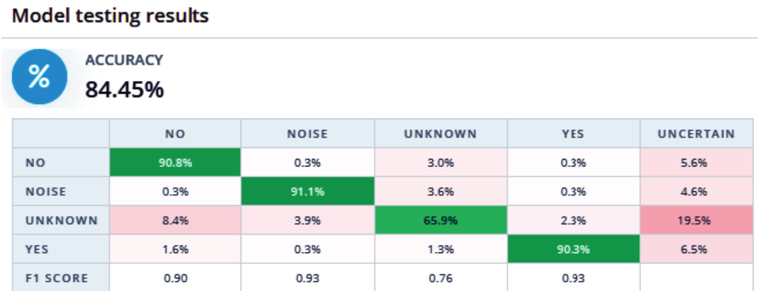


Рисунок 10.20. Запуск EON Tuner

16. Теперь мы можем протестировать модель на вкладке **Model Testing** (*Тестирование модели*). Когда мы загрузили набор данных на платформу Edge Impulse, данные были разделены в соотношении 80:20, где 80 % загруженных выборок используются для обучения, а остальные 20 % – для оценки точности полученной модели. Разделение набора данных помогает определить, не была ли модель перетренирована на наборе данных (это явление известно как переобучение). На следующем рисунке показана производительность нашей модели по сравнению с тестовым набором данных:



Model testing results

ACCURACY
84.45%

	NO	NOISE	UNKNOWN	YES	UNCERTAIN
NO	90.8%	0.3%	3.0%	0.3%	5.6%
NOISE	0.3%	91.1%	3.6%	0.3%	4.6%
UNKNOWN	8.4%	3.9%	65.9%	2.3%	19.5%
YES	1.6%	0.3%	1.3%	90.3%	6.5%
F1 SCORE	0.90	0.93	0.76	0.93	

Рисунок 10.21. Эффективность модели на тестовом наборе данных

17. Поскольку мы довольны эффективностью нашей модели, пришло время протестировать ее в реальности, развернув на Pico. Для этого перейдите на вкладку **Deployment** (*Развертывание*). Для запуска модели на Pico следует создать библиотеку Arduino. После выбора библиотеки Arduino нажмите на кнопку **Build** (*Сборка*) в нижней части страницы:

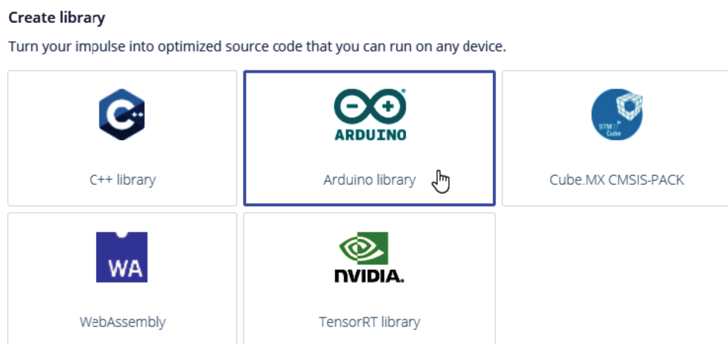


Рисунок 10.22. Создание библиотеки Arduino

> **Программирование Pico с помощью Arduino**
>
> Если вы незнакомы со средой разработки Arduino, мы рекомендуем ознакомиться с главой 12 «*Дополнительные возможности работы с Pico*». Убедитесь, что установлены необходимые пакеты для плат RP2040 Connect и Pico.

18. Как только создание библиотеки будет завершено, для загрузки с платформы Edge Impulse станет доступен соответствующий ZIP-файл. ZIP-файл будет содержать библиотеки и примеры кода, необходимые для запуска модели на Raspberry Pi Pico.
19. В среде Arduino IDE перейдите в раздел **Sketch** | **Include Library** | **Add .ZIP Library** (*Скетч* | *Включить библиотеку* | *Добавить .ZIP-библиотеку*) и укажите загруженный ZIP-файл:

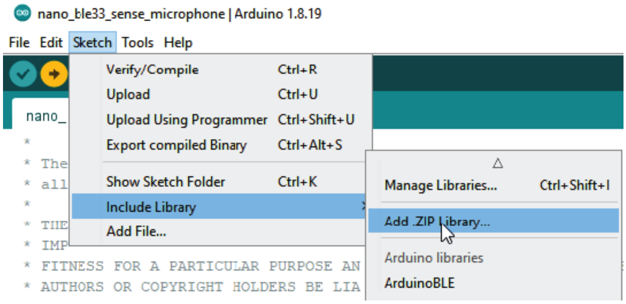


Рисунок 10.23. Добавление библиотеки

20. Пример кода будет доступен в пункте **File** | **Examples** (*Файл* | *Примеры*) под названием **Pico_Projects_Audio_Detection_inferencing** | **nano_ble33_sense_microphone**.

    `Pico_Projects_Audio_Detection_inferencing` – это название нашего проекта в Edge Impulse:

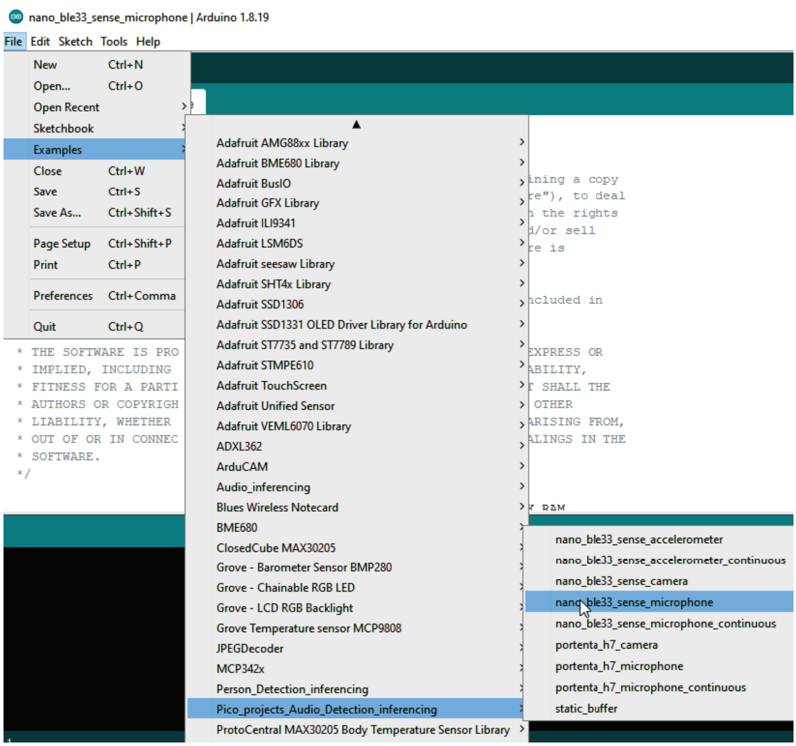


Рисунок 10.24. Расположение примера кода

21. Теперь нам нужно проверить, успешно ли компилируется наш пример кода для микроконтроллера RP2040. Мы можем проверить это, установив его на Arduino RP2040 Connect. Это связано с тем, что пример кода предназначен для использования PDM-микрофона, имеющегося на

плате Arduino Nano RP2040 Connect. Проверим, что код успешно компилируется, прежде чем вносить изменения:

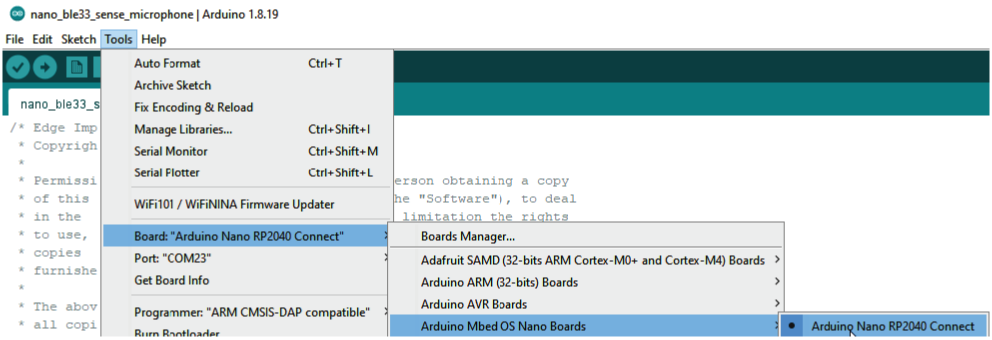


Рисунок 10.25. Установка Arduino Nano RP2040 Connect как целевой платы для компилирования

22. Вы можете проверить, что код компилируется, нажав на кнопку **Verify** (*Проверить*) (предполагается, что вы либо знакомы с Arduino IDE, либо ознакомились с ним сейчас, изучив главу 12 «*Дополнительные возможности работы с Pico*»).

23. Если у вас подключен Arduino RP2040, вы можете загрузить скомпилированный двоичный файл, нажав на кнопку **Upload** (*Загрузить*). Теперь вы готовы протестировать код, запустив монитор последовательного порта в Arduino IDE. Код связан с микрофоном PDM, который доступен на Arduino RP2040 Connect. Вы можете перейти к шагу 28, если хотите протестировать готовое приложение на Arduino RP2040 Connect. Пример кода при подключении Arduino RP2040 называется *arduino_rp2040_connect_sense_microphone.ino* и может быть загружен из репозитория GitHub этой книги.

24. Если вы работаете с Pico, то вам нужно подключить разъем PDM-микрофона к Pico следующим образом (слева от стрелки указан вывод на разъеме микрофона, справа – вывод Pico):

    - 3.3V → 3.3V;
    - GND соединяются между собой;
    - SEL → 3.3V;
    - CLK → GP3;
    - DAT → GP2.

25. На рис. 10.26 показана схема соединений Pico с разъемом микрофона.

26. Нам нужно изменить целевой микроконтроллер на Pico. Нам также необходимо внести некоторые изменения в существующий пример кода (необходимо включить два определения макросов). Как только код будет скомпилирован и загружен в Pico, можно будет запустить проект. Пример кода для этого называется *Pico_microphone.ino* и может быть найден в репозитории GitHub этой книги.

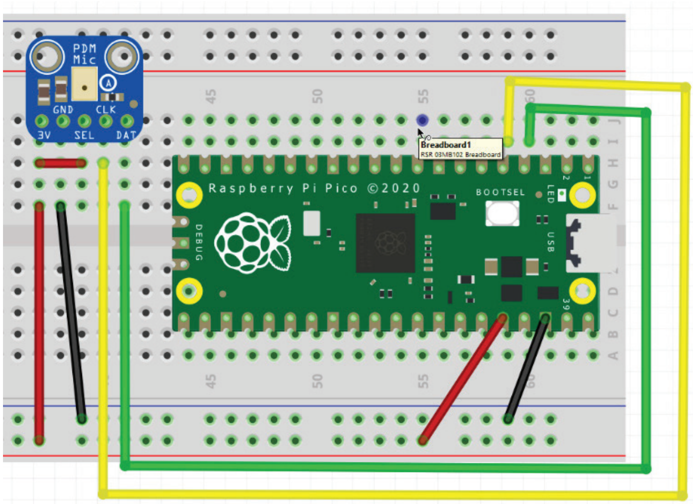


Рисунок 10.26. Схема подключения платы PDM-микрофона

27. После запуска монитора последовательного порта вы должны увидеть результаты работы модели, выдающей оценку прогноза для представленного звукового образца. На следующем рисунке модель предсказывает, что представленный образец является шумом с уверенностью 99 %:

```
    no: 0.00391
    noise: 0.91016
    unknown: 0.08203
    yes: 0.00391
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 1477 ms., Classification: 25 ms., Anomaly: 0 ms.):
    no: 0.01562
    noise: 0.85547
    unknown: 0.12500
    yes: 0.00391
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 1476 ms., Classification: 25 ms., Anomaly: 0 ms.):
    no: 0.00000
    noise: 0.99219
    unknown: 0.00781
    yes: 0.00000
Starting inferencing in 2 seconds...
```

☑ Autoscroll ☐ Show timestamp

Рисунок 10.27. Шум в звуковой выборке

28. Когда мы произносим слово «*yes*», модель может с 99%-ной уверенностью предсказать, что образец содержал это слово.

```
    no: 0.80859
    noise: 0.01172
    unknown: 0.02734
    yes: 0.15234
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 1490 ms., Classification: 25 ms., Anomaly: 0 ms.):
    no: 0.00781
    noise: 0.98047
    unknown: 0.00781
    yes: 0.00391
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 1485 ms., Classification: 25 ms., Anomaly: 0 ms.):
    no: 0.00000
    noise: 0.00000
    unknown: 0.00000
    yes: 0.99609
Starting inferencing in 2 seconds...
```

Рисунок 10.28. Образец аудиозаписи, содержащий «*yes*» (достоверность 99 %)

29. Когда мы произносим слово «*no*», модель может с уверенностью 78 % предсказать, что образец содержит это слово:

```
    unknown: 0.02734
    yes: 0.00000
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 1483 ms., Classification: 25 ms., Anomaly: 0 ms.):
    no: 0.00000
    noise: 0.00000
    unknown: 0.00000
    yes: 0.99609
Starting inferencing in 2 seconds...
Recording...
Recording done
Predictions (DSP: 1482 ms., Classification: 25 ms., Anomaly: 0 ms.):
    no: 0.78906
    noise: 0.00781
    unknown: 0.16797
    yes: 0.03125
Starting inferencing in 2 seconds...
Recording...
Recording done
```

☑ Autoscroll ☐ Show timestamp

Рисунок 10.29. Звуковой образец, содержащий «*no*» (достоверность 78%)

Таким образом, мы успешно создали приложение TinyML, которое может определять ключевое слово по звуковому образцу. В следующем разделе мы обсудим готовый пример, который определяет, присутствует ли человек в кадре, снятом камерой.

Классификация изображений

В этом разделе мы обсудим пример классификации изображений с использованием Pico. Мы будем использовать программу в виде готового двоичного файла, который уже был скомпилирован и может быть загружен, чтобы определить, присутствует ли человек в кадре, снятом камерой. Приступим.

1. Сначала загрузите себе скомпилированный двоичный файл с **https://github.com/ArduCAM/RPI-Pico-Cam/blob/master/tflmicro/bin/person_detection_screen_int8.uf2**.
2. Для загрузки двоичного файла необходимо перевести Pico в режим загрузчика. Мы рекомендуем ознакомиться с главой 1 «*Начало работы с Raspberry Pi Pico*», если вы незнакомы с этим процессом.
3. Скопируйте загруженный двоичный файл в Pico.
4. Теперь нам нужно подключить модуль камеры к Pico следующим образом (слева от стрелки указаны названия контактов на модуле камеры, справа – контакты на Pico:
   - CS → GP5;
   - MOSI → GP3;
   - MISO → GP4;
   - SCK → GP2;
   - GND соединяются между собой;
   - VCC → 3.3V;
   - SDA → GP8;
   - SCL → GP9.
5. Теперь все готово для тестирования этого приложения с помощью Pico. Для тестирования мы использовали плату Pico4ML. Это связано с тем, что плата поставляется со встроенным модулем камеры. Мы навели камеру на куклу-болванчика и увидели следующий результат, который показан на рис. 10.30.
6. Pico может с уверенностью 93,8 % предсказать, что захваченный кадр содержит человека. Это происходит потому, что модель пытается обнаружить характерные формы лица в кадре, чтобы определить, присутствует ли в кадре человек[91].
7. Pico также выводит статистику сделанных заключений через последовательный порт (см. рис. 10.31).

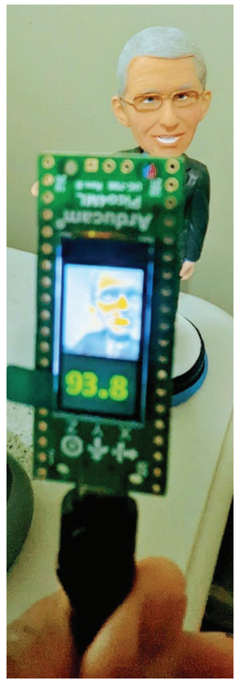


Рис. 10.30. Pico4ML распознает болванчика

На этом мы закончим с примером классификации изображений.

[91] Обратите внимание, что человек при этом в кадре отсутствует: его заменяет кукла. Хотя человеческий интеллект мгновенно отличает кукольное лицо от лица живого человека, научить программы подобным нюансам даже сложнее, чем уверенному распознаванию личности по видеосъемке. – *Прим. перев.*

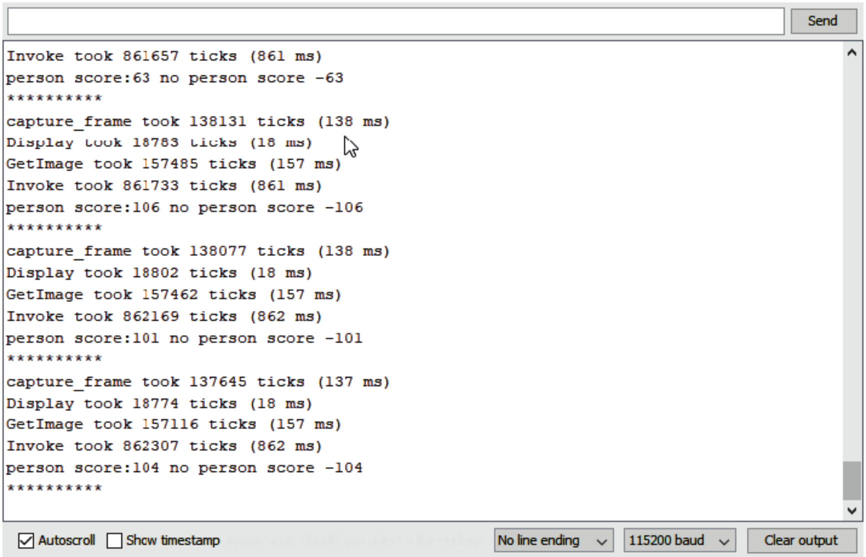


Рисунок 10.31. Вывод через последовательный порт из Pico

Разработка фронтальных устройств

Есть несколько факторов, которые следует учитывать, прежде чем приступить к разработке продукта, использующего TinyML, а именно:

- наборы данных, доступные для разработки вашего продукта. Лучше всего начать с использования существующих наборов данных или данных из существующего приложения;
- процесс разработки продукта с использованием TinyML требует большого количества проб и ошибок для уточнения параметров;
- приложения TinyML подходят для привлечения новых источников дохода и повышения производительности;
- следует также учитывать необходимость периодической перенастройки модели для устранения проблем, выявленных в системе.

Подведем итоги этой главы.

Итоги

В этой главе мы познакомились с направлением TinyML, его приложениями и началом разработки. Мы рассмотрели пример распознавания ключевых слов с использованием платформы Edge Impulse и его тестирование с помощью Pico. Мы также кратко рассмотрели пример классификации изображений, в котором использовали готовый двоичный файл программы, определяющей присутствие человека в кадре. Наконец, мы завершили эту главу обсуждением некоторых факторов, которые следует учитывать при разработке фронтального устройства.

В следующей главе мы обсудим разработку продукта на основе Pico.

Глава 11

Создаем готовый продукт

В этой главе мы рассмотрим создание продукта с использованием Raspberry Pi Pico. Мы разработаем свою печатную плату для Pico, к которой будем подключать модуль сотовой связи для отправки сообщений в облако. Мы также обсудим, как собрать телефон на основе Pico (показан на следующем рисунке). Мы специально выбрали процесс разработки продукта, связанный с проектированием печатной платы.

Рисунок 11.1. Pico-телефон

В этой главе мы рассмотрим следующие темы:

- принципы работы Pico-телефона;
- требования, которые следует учитывать;
- проектирование печатной платы;
- подготовка платы к эксплуатации;
- развитие проекта.

Технические требования

Файлы программного и аппаратного обеспечения для этой главы можно найти по адресу **https://bit.ly/35088w4**.

Для проектов в этой главе требуется следующее оборудование:

- Raspberry Pi Pico (**https://bit.ly/3AJtoAf**);
- Blues Wireless Notecard (**https://blues.io/products/notecard/**);

- антенна сотовой связи (**https://gsm-repiteri.ru/prodazha/antenny-ufl-ipx-dlya-mobilnyh-routerov/antenna-gsm-3g-4g-s-razyomom-u-fl-ipx**[92]);
- 12-клавишная Qwiic-клавиатура (**https://www.sparkfun.com/products/15290**[93]);
- изготовленная печатная плата;
- компоненты для печатной платы: спецификация доступна в репозитории GitHub этой книги;
- принадлежности для пайки, см. далее.

Видеоролики с кодом в действии для этой главы можно посмотреть по адресу **https://bit.ly/38Th2gy**.

> **Установка CircuitPython**
>
> Мы предполагаем, что вы установили CircuitPython на свой Pico. Если вы незнакомы с процессом установки, то рекомендуем следовать инструкциям из главы 1 «*Начало работы с Raspberry Pi Pico*».

В следующем разделе мы кратко обсудим Pico-телефон и его истоки.

Истоки Pico-телефона

Мы создали телефон Pico в 2021 году и опубликовали его как проект Hackster (**https://bit.ly/3pY7J5e**). Проект возник, когда мы хотели оценить сотовый модуль. Этот проект был выбран здесь для описания процесса разработки продукта, потому что он был одним из двадцати одного рекомендуемого проекта 2021 года (**https://bit.ly/3CXh3ec**), а также был описан в мартовском выпуске журнала Hackspace за 2022 год (**https://bit.ly/3CYAJ1l**), который издается фондом Raspberry Pi. Журнал можно скачать бесплатно.

> **Изменения в этой главе по сравнению с описанием проекта Hackster**
>
> В то время как проект Hackster обсуждает отправку шутки на номер телефона, здесь мы рассмотрим аспекты разработки продукта. В частности, мы обсудим определение требований к продукту, создание прототипа и многое другое.

В следующем разделе разберем определение требований к продукту.

[92] Приведенная ссылка на российский интернет-магазин указывает на недорогую встраиваемую GSM-антенну с разъемом типа u.FL, подходящим к используемой плате Blues Wireless Notecard (см. рис. 11.2). Отметим, что в тексте антенна и ее подключение более не упоминаются. – *Прим. перев.*

[93] О приобретении клавиатуры SparkFun в российских условиях см. ссылку 63 на стр. 174. В случае ее недоступности, а также в целях удешевления обычную недорогую матричную клавиатуру (**https://www.chipdip.ru/product/ak-304-n-bbw**) несложно доработать самостоятельно с помощью I2C-расширителя PCF8574 (см. дополнительный модуль красного цвета на рис. 11.3, описание доработки можно посмотреть по адресу **https://arduino.ru/forum/programmirovanie/podklyuchenie-matrichnoi-klaviatury-po-i2c**). – *Прим. перев.*

Определение требований

Прежде чем мы начнем разрабатывать продукт, нам нужно определить требования к нему. Нужно понять, возможно ли вообще создать продукт.

Давайте определим некоторые требования к нашему продукту с точки зрения его назначения:

- продукт должен содержать сетевой компонент для соединения с сотовой сетью и загрузки данных в облако;
- устройство должно иметь клавиатуру для ввода данных пользователем;
- в первой версии продукта удобно использовать Raspberry Pi Pico. Это позволит вам применять микроконтроллер RP2040 в будущих версиях. В первой версии будет использоваться готовая плата Pico, в то время как в будущих версиях будет использоваться микроконтроллер RP2040, встроенный непосредственно в печатную плату;
- питание устройства осуществляется с помощью USB-кабеля (через USB-разъем Pico) или с помощью разъема питания постоянного тока.

В реальном мире к продукту предъявлялось бы больше требований. В этой главе мы постараемся сделать все попроще.

Выбор компонентов

Теперь, когда мы разобрались с требованиями, давайте обсудим выбор компонентов для продукта.

- Мы собираемся использовать Raspberry Pi Pico. Важно знать периферийные устройства, доступные на Pico, чтобы определить варианты подключения к сети. Обычно требуется тщательный поиск, чтобы найти микроконтроллер, отвечающий всем требованиям нашего продукта. Параметры, которые следует рассмотреть, включают цену, занимаемую площадь на печатной плате и доступность.
- Еще одна причина выбрать Pico для нашего продукта – это RP2040-микроконтроллер. Когда мы переносим наш дизайн с Pico на «голый» микроконтроллер RP2040, это помогает снизить стоимость продукта, поскольку отдельный микроконтроллер очень дешев, а нам все равно понадобились бы другие компоненты для построения минимально необходимой схемы[94]. Мы обсудим перенос нашего продукта с Pico на дизайн на базе микроконтроллера RP2040 в разделе «*Продвижение проекта*».
- Если вы незнакомы с процессом поиска и выбора компонентов, то рекомендуем еженедельно обновляемый раздел сайта Adafruit Industries под

[94] Это утверждение подлежит тщательной проверке: в рознице набор отдельных компонентов запросто может оказаться дороже готового модуля; увеличивается сложность (соответственно, площадь и стоимость) печатной платы, а также сложность ее монтажа в сравнении с профессионально выполненными платами готовых модулей. Это касается, в частности, и самого Raspberry Pi Pico, даже с учетом его избыточности для конкретного проекта. – *Прим. перев.*

названием *Great Search*. Например, здесь показан фрагмент видео, в котором демонстрируется выбор микроконтроллера: **https://bit.ly/3rfDYg3**[95].

Проверка наличия компонента

На этапе выбора компонента важно убедиться, что рассматриваемый компонент имеется в наличии в количестве, необходимом вам на этапе производства. Это особенно актуально во время дефицита полупроводников из-за последствий пандемии. Было бы идеально выбрать компоненты с возможностью замены в случае его нехватки.

- Необходимо определить варианты сетевого подключения для нашего продукта. Различные варианты подключения включают Wi-Fi, Bluetooth и сотовую связь. Мы выбрали сотовую связь по следующим причинам:
  - мы хотим, чтобы наш продукт был подготовлен для использования и был простым в эксплуатации. Устройства с подключением к сотовой связи могут быть полностью подготовлены перед отправкой заказчику;
  - если мы используем Wi-Fi, нам нужно будет предоставить клиенту подробные инструкции по настройке устройства. Ему также нужно будет убедиться, что устройство установлено в месте, где есть надежное подключение к Wi-Fi.
- Однако сотовые сети также имеют определенные ограничения: покрытие сети, периодические расходы и многое другое. Поэтому придется оговорить, что наш продукт будет использоваться в местах, где доступна сотовая связь.
- Для подключения к сотовой связи мы решили использовать плату Notecard компании Blues Inc. (**https://blues.io**):

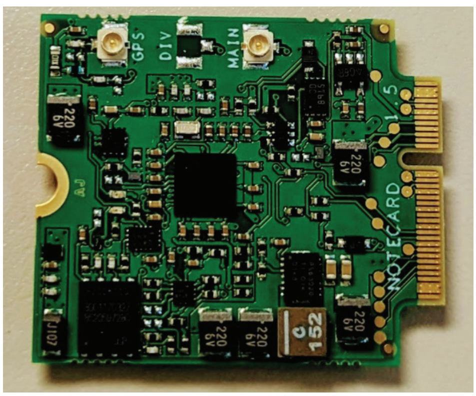


Рисунок 11.2. Notecard от Blues Inc.

[95] В российских условиях поиск компонентов рекомендуется начинать с каталогов интернет-магазинов «Чип и Дип» (**https://www.chipdip.ru**), iArduino.ru (**https://iarduino.ru**) и Amperka.ru (**https://amperka.ru**). В отличие от многих других, на этих сайтах большинство ключевых товаров снабжены ссылками на документацию, а в последних двух – еще и примерами использования. – *Прим. перев.*

Мы выбрали ее по следующим причинам:

- плата поставляется с 10 оплаченными годами и 500 МБ бесплатных данных сотовой связи[96]. Это позволяет разработать продукт, не требующий текущих затрат;
- платой можно управлять через интерфейс I2C или интерфейс UART. Pico имеет оба этих интерфейса;
- плата поставляется с библиотеками Python, которые позволяют программировать Pico с помощью CircuitPython;
- нам нужна клавиатура для пользовательского ввода. Мы выбрали для нашего продукта 12-клавишную телефонную клавиатуру (кнопочный номеронабиратель) с I2C-интерфейсом, потому что она помогает устранить необходимость в сканировании клавиш;

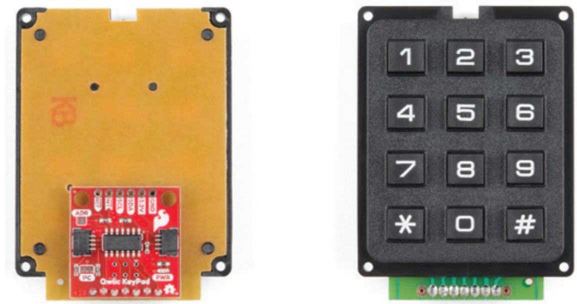


Рисунок 11.3. Телефонная Qwiic-клавиатура SparkFun (источник изображения: SparkFun; лицензия: CC by 2.0)

- мы также будем использовать в нашей конструкции разъем Qwiic/Stemma[97], который позволит подключать любое периферийное устройство I2C к нашему продукту.

Теперь, когда мы определили основные компоненты, поговорим про макетирование для обоснования концепции продукта.

Построение макета

В этом разделе мы обсудим построение макета. Это включает в себя проверку работоспособности основных компонентов.

> **Прототип и макет**
>
> Построение макета выполняется для доказательства того, что продукт отвечает функциональным требованиям. Макет может не выглядеть как конечный продукт, и он предназначен для проверки соответствия компонентов требованиям нашего продукта. Прототип обычно выглядит уже как конечный продукт. Прототип может не иметь те же размеры или оформление, но используется для точной настройки функциональности конечного продукта.

---

[96] Очевидно, эти бонусы действительны только на территории США. – *Прим. перев.*

[97] См. раздел «*Экосистема Qwiic/STEMMA*» на стр. 65. – *Прим. перев.*

Для этого продукта нам необходимо протестировать следующие функции:

- проверка интерфейса Notecard и самой сотовой связи;
- проверка клавиатуры и ввода с нее.

Чтобы смакетировать эти действия, нам необходимо определить и установить необходимые библиотеки. Мы собираемся сделать это в следующем разделе.

Установка необходимых библиотек

При разработке макета нам также нужно изучить программное обеспечение, необходимое для создания нашего продукта. Для начала следует определить доступность библиотек для взаимодействия с периферийными устройствами. Это помогает определить общее время и возможные затраты ресурсов на написание собственных библиотек, если это будет нужно. В данном случае доступны библиотеки как для сотового модуля, так и для клавиатуры. Давайте рассмотрим установку библиотек для обоих периферийных устройств.

Модуль сотовой связи

Библиотека сотового модуля Notecard доступна по адресу **https://bit.ly/37DlpeO**. Извлеките содержимое последней версии и скопируйте его из папки *notecard* в папку *lib* Raspberry Pi Pico.

Клавиатура

Библиотека для клавиатур Qwiic доступна по адресу **https://bit.ly/3L3EHZu**. Извлеките содержимое последней версии и скопируйте двоичный файл *sparkfun_qwiickeypad.mpy* из папки *lib* в папку *lib* вашего Raspberry Pi Pico.

Теперь, когда мы установили необходимые библиотеки, проверим их функциональность отдельно. В следующем разделе проверим функциональность Notecard.

Тестирование модуля Notecard

В этом разделе мы подключим Notecard к Raspberry Pi Pico и проверим его функциональность. Notecard – это модуль в форм-факторе M.2[98], для которого требуется несущая плата (показана на рис. 11.4).

Мы будем использовать следующую несущую плату для тестирования интерфейса Notecard: **https://blues.io/products/notecarrier/notecarrier-a/**.

Notecard подключается к Raspberry Pi Pico следующим образом (схема показана на рис. 11.5):

- GP9 → SCL;
- GP8 → SDA;
- 3.3V → VIN;
- выводы GND соединяются вместе.

[98] О формате M.2 см. сноску 6 на стр. 24. – *Прим. перев.*

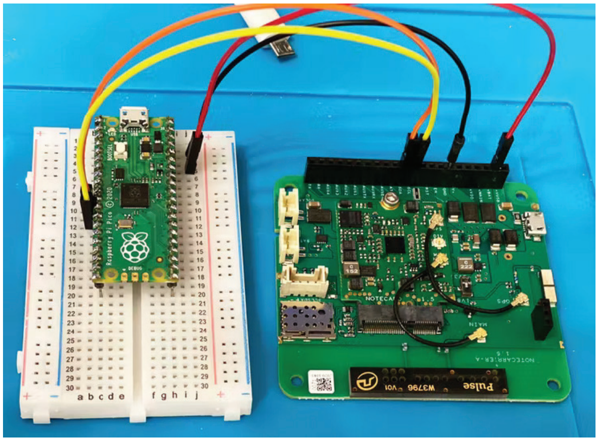


Рисунок 11.4. Модуль Notecard, подключенный к Pico (источник изображения: **hackster.io**; лицензия: MIT)

Схема соединений выглядит следующим образом:

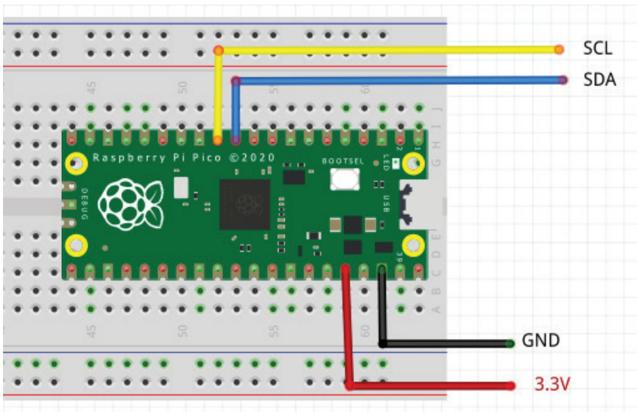


Рисунок 11.5. Схема соединений по интерфейсу I2C

Прежде чем начать тестировать Notecard, необходимо создать проект на сайте ***https://notehub.io***. Создание проекта позволяет нам загружать данные из Notecard в облако. Мы рекомендуем следовать инструкциям по адресу **https://bit.ly/3D1qpFZ**, чтобы создать новый проект. После создания нового проекта запишите идентификатор продукта, поскольку это необходимо для тестирования.

Давайте рассмотрим код, необходимый для проверки модуля Notecard. Пример кода, который обсуждается в этом разделе, можно загрузить из репозитория GitHub данной главы в файле *code_notecardtest.py*.

1. Сначала импортируем необходимые модули `board`, `busio` и `notecard`:

```
import board
import busio
import notecard
```

2. Затем мы должны инициализировать интерфейс I2C, через который подключен модуль Notecard:

```
i2c = busio.I2C(board.GP9, board.GP8)
card = notecard.OpenI2C(i2c, 0, 0, debug=True)
```

3. В предыдущем фрагменте кода мы инициализировали Notecard с помощью `debug=True`. При этом выводится вся исходящая/входящая информация.

4. Теперь нам нужно определить идентификатор продукта, который мы извлекли из созданного проекта **https://notehub.io/sign-in?flow=bccb016e-6b5f-42e8-8f61-b9b4d463598b** ранее:

```
productUID = "com.example.name:project_name"
```

5. Введем идентификатор продукта и подключим Notecard к Notehub:

```
req = {"req": "hub.set"}
req["product"] = productUID
req["mode"] = "periodic"
req["inbound"] = 120
req["outbound"] = 60
req["sync"] = True
rsp = card.Transaction(req)
while True:
  pass
```

6. Сложив все это вместе, получаем следующее:

```
import board
import busio
import notecard
productUID = "com.example.name:project_name"
# Create bus object using our board's I2C port
i2c = busio.I2C(board.GP17, board.GP16)
# Create relay object
card = notecard.OpenI2C(i2c, 0, 0, debug=True)
req = {"req": "hub.set"}
req["product"] = productUID
req["mode"] = "periodic"
req["inbound"] = 120
req["outbound"] = 60
req["sync"] = True
rsp = card.Transaction(req)
while True:
  pass
```

7. Сохранив приведенный пример кода как *code.py*, мы должны увидеть следующий результат:

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

code.py output:
{"inbound": 120, "outbound": 60, "mode": "periodic", "req": "hub.set", "product": '        , "sync": true}
{}
```

Рисунок 11.6. Результаты работы программы Notecard

8. Модуль Notecard подключится к облаку и синхронизируется в первый раз. Он также опубликует свое приблизительное местоположение в интерфейсе Notehub, как показано на следующем рисунке:

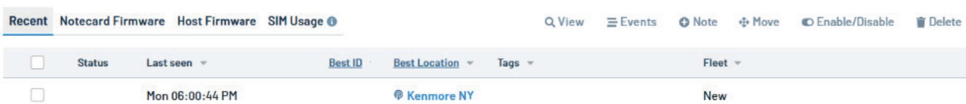


Рисунок 11.7. Модуль Notecard, подключенный к интерфейсу Notehub

Если у вас возникли проблемы с настройкой, мы рекомендуем прочитать следующее сообщение на сайте ***hackster.io***: **https://bit.ly/3tvswyG**.

В следующем разделе мы протестируем клавиатуру.

Тестирование клавиатуры

В этом разделе мы проверим Qwiic-клавиатуру SparkFun. Клавиатура подключается к Raspberry Pi Pico следующим образом (схема показана на рис. 11.5 ранее):

- GP9 → SCL;
- GP8 → SDA;
- 3.3V → VIN;
- выводы GND соединяются вместе.

Давайте посмотрим на код, который нужен для тестирования нашей клавиатуры. Пример кода, который мы будем обсуждать в этом разделе, можно загрузить из репозитория GitHub данной книги в файле *code_keypad_test.py*. Он включает следующие действия:

1. Как всегда, первым шагом является импорт необходимых модулей:

```
import sys
from time import sleep
import board
import busio
import sparkfun_qwiickeypad
```

2. Затем мы должны инициализировать интерфейс I2C и клавиатуру:

```
i2c = busio.I2C(board.GP9, board.GP8)
keypad = sparkfun_qwiickeypad.Sparkfun_QwiicKeypad(i2c)
```

3. Теперь проверим, подключена ли клавиатура и правильно ли она инициализирована:

```
if keypad.connected:
  print("Keypad connected. Firmware: ", keypad.version)
else:
  print("Keypad does not appear to be connected. Please check  wiring.")
  sys.exit()
```

4. Значения с клавиатуры хранятся в стеке FIFO[99]. Мы можем получить значения нажатых клавиш следующим образом:

```
button = -1
while True:
  # request a button
  keypad.update_fifo()
  button = keypad.button
  # Display the button value
  if button > 0:
    print("Button '" + chr(button) + "' was pressed.")
  sleep(0.100)
```

5. Собрав все это вместе, имеем следующее:

```
import sys
from time import sleep
import board
import busio
import sparkfun_qwiickeypad

i2c = busio.I2C(board.GP9, board.GP8)
# Create keypad object
keypad = sparkfun_qwiickeypad.Sparkfun_QwiicKeypad(i2c)

print("Qwiic Keypad Simple Test")

# Check if connected
if keypad.connected:
  print("Keypad connected. Firmware: ", keypad.version)
else:
  print("Keypad does not appear to be connected. Please check  wiring.")
  sys.exit()
```

[99] FIFO (*first in, first out*, «первым пришёл – первым ушёл») – способ организации обработки данных, при котором данные, пришедшие первыми, будут и обслуживаться первыми, пришедшие следующими ждут, пока обслуживание первых не будет закончено. Такой порядок обработки называется *очередью*, называть его *стеком* (как это делают авторы) неправильно, потому что стек – это альтернативная организация обработки по принципу LIFO («последним пришел – первым ушел»). – *Прим. перев.*

```
print("Press any button on the keypad.")

# button value -1 is error/busy, 0 is no key pressed
button = -1

# while no key is pressed
while True:
  # request a button
  keypad.update_fifo()
  button = keypad.button
  # Display the button value
  if button > 0:
    print("Button '" + chr(button) + "' was pressed.")
  # wait a bit before trying again
  sleep(0.100)
```

После сохранения этого примера, как *code.py*, мы сможем увидеть следующий результат:

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

code.py output:
Qwicc Keypad Simple Test
Keypad connected. Firmware:  v1.0
Press any button on the keypad.
Button '2' was pressed.
Button '3' was pressed.
Button '1' was pressed.
Button '4' was pressed.
Button '5' was pressed.
Button '6' was pressed.
```

Рисунок 11.8. Тестовый результат для клавиатуры

Теперь, когда мы протестировали отдельные компоненты нашего продукта, давайте поработаем над проектированием печатной платы.

Проектирование печатной платы

В этом разделе мы рассмотрим проектирование печатной платы. Это название относится к плате, на которой соединения между компонентами проложены с использованием медной фольги[100]. Эти соединения называются дорожками, и на рис. 11.9 показаны дорожки на типичной печатной плате.

100 Название «печатная плата» возникло из-за того, что при формировании рисунка дорожек ранее использовались методы, близкие к типографской печати (в частности, шелкография и другие способы). – *Прим. перев.*

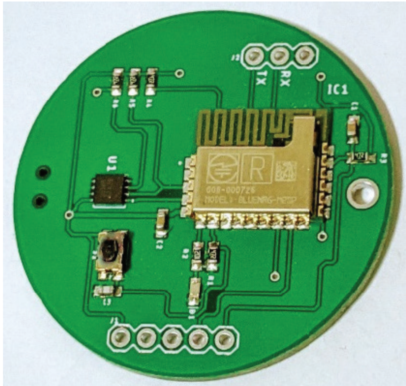


Рисунок 11.9. Дорожки на типичной печатной плате

Мы рекомендуем учебное руководство по печатным платам от SparkFun: **https://learn.sparkfun.com/tutorials/pcb-basics/all**. В случае если вы незнакомы с проектированием печатных плат, мы рекомендуем пройти следующий курс: **https://teachmepcb.com**[101].

Программное обеспечение для проектирования печатных плат

Существуют различные инструменты проектирования, доступные для проектирования печатных плат. Для проекта, описанного в этой главе, мы использовали Autodesk Eagle. Однако доступны и иные инструменты, в том числе KiCad, Altium и другие. KiCad – это проект с открытым исходным кодом, который можно использовать бесплатно. Мы использовали Eagle, но для KiCad доступно множество учебных ресурсов[102].

Любое проектирование печатной платы состоит из шести основных этапов, а именно:

- составление схемы;
- раскладка печатной платы[103];

101 На русском языке для подробного ознакомления с технологией рекомендуется учебник: *Пирогова Е. В.* Проектирование и технология печатных плат. М.: ФОРУМ: ИНФРА-М, 2005. Для начальных знаний достаточно тематических статей на «Хабре» (например, **https://habr.com/ru/post/414141/** и других) и ознакомления с требованиями по проектированию на сайтах изготовителей (например, в Центре поддержки ООО «Резонит»: **https://www.rezonit.ru/directory**). – *Прим. перев.*

102 В российских любительских кругах популярна Sprint Layout 6 – немецкая программа для разводки печатных плат с интерфейсом, переведенным на русский (эта версия распространяется бесплатно). По сравнению с более профессиональными пакетами, Sprint Layout лишена функций подготовки принципиальной схемы и автоматического отслеживания соединений, зато гораздо проще в освоении. Русскоязычный сайт программы **https://sprint-layout.ru**; подробное пособие по работе в программе Sprint Layout доступно, например, здесь: **http://www.radioscanner.ru/files/download/file19172/proektirovanie_pecatnih_plat_v_programme_sprint_layout_6.pdf**. – *Прим. перев.*

103 Вообще говоря, следующий пункт («Требования к корпусу изделия») предшествует раскладке вместе с упущенным авторами этапом компоновки компонентов на плате, который они объединяют с раскладкой. – *Прим. перев.*

- требования к корпусу изделия;
- генерация файлов Gerber;
- изготовление печатной платы;
- монтаж платы.

Давайте обсудим каждый из них более подробно.

Подготовка принципиальной схемы

Схему, которую мы обсудим в этом разделе, можно загрузить с репозитория GitHub этой главы как *blues-hardware.sch*[104]. На данном этапе мы составляем принципиальную схему для нашего продукта. Мы указываем все соединения, которые обсуждали ранее в этой главе. Схему можно загрузить вместе с PDF-файлом по адресу: **https://github.com/PacktPublishing/Raspberry-Pi-Pico-DIY-Workshop/tree/main/chapter_11/hardware**.

После завершения схемы выполняется проверка электрических соединений, чтобы убедиться в отсутствии очевидных ошибок, таких как забытое соединение.

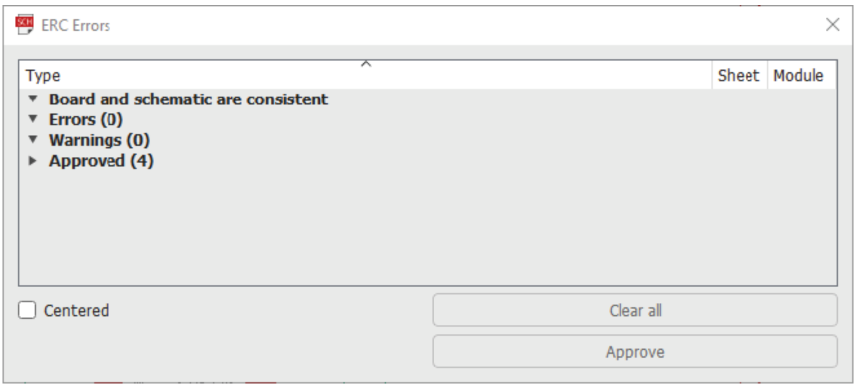


Рисунок 11.10. Проверка электрических соединений с помощью Autodesk Eagle

Теперь, когда составление схемы закончено, мы обсудим раскладку печатной платы.

Раскладка печатной платы

Макет печатной платы, который мы обсудим в этом разделе, можно загрузить из репозитория GitHub данной главы как *blues-hardware.brd*. На этапе раскладки печатной платы мы прокладываем все соединения, которые были определены на этапе составления схемы. Мы также определяем форму и размеры нашей печатной платы. Программное обеспечение помогает с маршрутизацией, отслеживая количество оставшихся соединений. После завершения прокладки соединений готовый рисунок платы в программе Eagle должен выглядеть примерно следующим образом:

104 Обращаем внимание, что типы файлов проекта платы, указанные авторами здесь и далее, относятся к используемой ими программе Eagle и могут не восприниматься другими программными пакетами для раскладки. – *Прим. перев.*

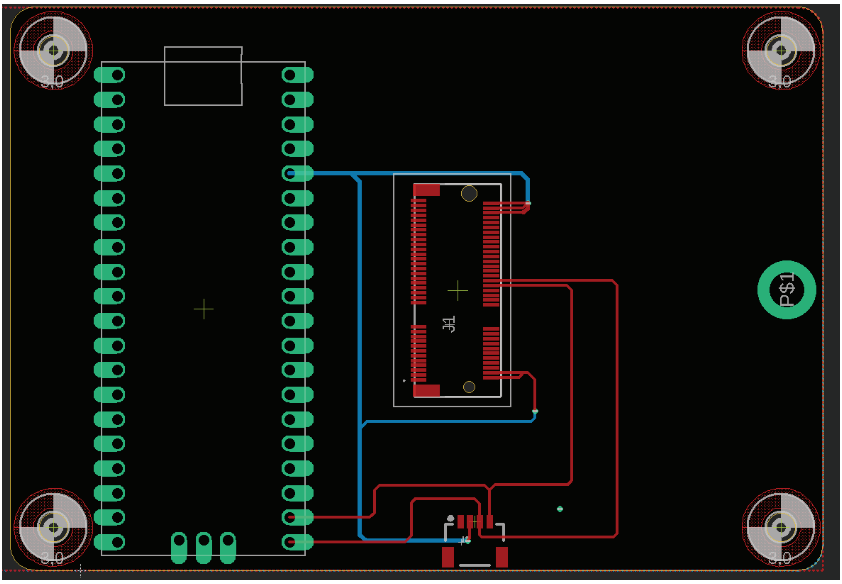


Рисунок 11.11. Раскладка платы в программе Autodesk Eagle

После завершения раскладки необходимо выполнить проверку с точки зрения норм проектирования, установленных производителем платы, чтобы убедиться, что она не нарушает никаких ограничений. На следующем рисунке показан результат подпрограммы проверки правил проектирования для нашего продукта:

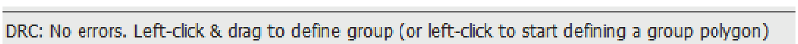


Рисунок 11.12. Результат проверки правил проектирования

Правила проектирования доступны у производителей печатных плат в соответствии с их минимальными требованиями к печатной плате. Как правило, производитель указывает ряд вариантов, которые позволяют последовательно обойти ограничения, но при этом удорожают плату (односторонняя, двухсторонняя или многослойная плата, минимальный шаг сетки расположения отверстий, металлизация отверстий и т. д.).

В следующем разделе мы рассмотрим выбор корпуса.

Выбор корпуса

В нашем текущем проекте форма печатной платы представляет собой прямоугольник (с размерами 83,9 мм × 58,2 мм), и мы рассматривали корпус как необязательный компонент. При проектировании продукта необходимо учитывать требования к корпусу. Некоторые производители готовых корпусов прилагают рекомендуемый шаблон печатной платы. Например, следующий производитель корпусов предоставляет файлы шаблонов, которые можно напрямую импортировать в Eagle для проектирования: **https://www.polycase.com/**.

В следующем разделе мы обсудим генерацию файлов Gerber.

Генерация файлов Gerber

После того как печатная плата разложена, пришло время сгенерировать файлы Gerber для отправки производителю. Gerber – это открытый стандартный формат, содержащий всю информацию, необходимую для изготовления печатной платы. Файлы Gerber для нашего продукта можно загрузить из репозитория GitHub этой главы в архиве *blues-hardware.zip*.

Файлы Gerber генерируются скриптами, известными как CAM-процессор. Мы использовали CAM, доступный по следующему адресу: **https://docs.oshpark.com/designtools/eagle/generating-gerbers/**[105].

Как только Gerber-файлы будут сгенерированы, пришло время отправить платы на изготовление.

Изготовление плат

Следующий шаг – изготовление плат. Существует много недорогих производителей печатных плат, но надо учитывать, что большинство их рассчитано на производство больших количеств и за несколько экземпляров могут и не взяться. В России для любительских нужд рекомендуется ООО «Резонит» (**https://www.rezonit.ru**), а также китайские производители вроде PCWay (русскоязычная версия сайта: **https://www.pcbway.ru/**) – у последних может быть дешевле (особенно при достаточно больших партиях), но из-за затрат времени на пересылку заказ выполняется гораздо дольше. Процесс довольно прост: вы загружаете свои файлы Gerber и оплачиваете затраты на изготовление.

> **О стоимости производства печатных плат**
>
> Необходимо учитывать, что процесс производства печатных плат складывается из двух этапов: подготовка к производству (изготовление шаблонов согласно присланным вами Gerber-файлам) и собственно изготовление. Первый этап неизменен при любых заказываемых количествах и при небольших объемах вносит бóльшую часть в стоимость заказа. Ошибки в раскладке при этом могут иногда учитываться и исправляться, но только в части несоответствия производственным нормам. При наличии ошибок в раскладке (см. далее) этап подготовки приходится оплачивать заново полностью. Зато при повторе заказа первый этап оказывается уже не нужен. В любом случае, чем больше экземпляров вы заказываете, тем меньше стоимость в расчете на одну плату.

После изготовления они отправляют вам печатные платы. Печатная плата для нашего продукта выглядит следующим образом:

105 Инструкции по генерации Gerber-файлов из различных программ раскладки, в том числе из Eagle и Sprint Layout, на русском языке можно найти, например, в Центре поддержки ООО «Резонит» (**https://www.rezonit.ru/directory**). – *Прим. перев.*

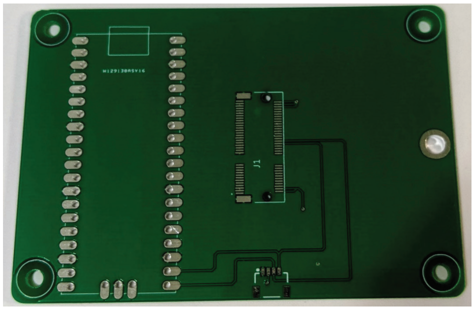


Рисунок 11.13. Готовая печатная плата, полученная с производства

В следующем разделе мы обсудим сборку печатной платы.

Монтаж печатной платы

Наш продукт пока представляет собой кучу компонентов как для установки в сквозные отверстия, так и для поверхностного монтажа. Мы припаяли разъем M.2 с помощью печи для оплавления припоя[106] и вручную припаяли разъемы для установки Pico. После монтажа платы выглядят следующим образом:

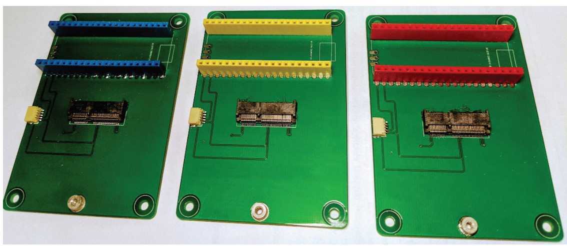


Рисунок 11.14. Печатные платы после монтажа

После того как плата смонтирована, необходимо установить Raspberry Pi Pico и Notecard следующим образом:

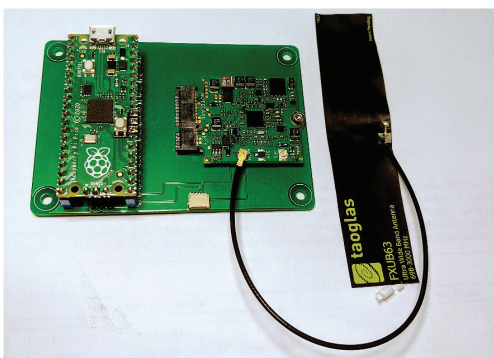


Рисунок 11.15. Плата в сборе

[106] Печь для пайки компонентов поверхностного монтажа в любительских условиях может быть заменена на паяльный (монтажный) фен, который компактнее, удобнее в домашних условиях и гораздо дешевле (см., например, **https://www.ozon.ru/category/payalnye-feny-10061**). – *Прим. перев.*

Теперь, когда мы собрали плату, рассмотрим вопрос подготовки платы к эксплуатации.

Подготовка платы к эксплуатации

Подготовка платы – это процесс, в ходе которого мы шаг за шагом проверяем функциональность ее дизайна. Это похоже на действия, которые мы обсуждали ранее для тестирования Notecard и Qwiic-клавиатуры SparkFun. В процессе проверки мы обнаружили недостающее соединение на плате. Мы заменили его отрезком провода (как показано на следующем рисунке) и проверили его функциональность:

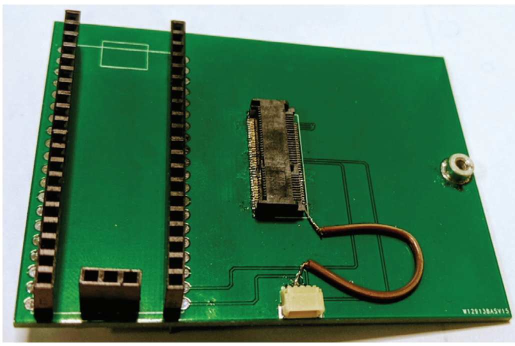


Рисунок 11.16. Исправление отсутствующего соединения с помощью отрезка провода

В случае обнаружения подобных ошибок после тестирования необходимо пересмотреть разводку печатной платы, добавив недостающее соединение.

Окончательная сборка и тестирование

Пришло время создать наш конечный продукт. Мы приобрели старый стационарный телефон с клавиатурой и удалили его внутреннюю схему. На место клавиатуры приклеили нашу клавиатуру и поместили печатную плату внутрь телефона.

Pico-телефон в сборе выглядит следующим образом:

Рисунок 11.17. Pico-телефон в сборе

Мы рекомендуем прочитать нашу статью в Hackster, чтобы узнать, как мы использовали его для текстовых сообщений, когда человек вводил свой номер телефона: **https://bit.ly/3pY7J5e**.

Давайте подведем итог тому, что мы сделали в этой главе.

Развитие проекта

До сих пор мы использовали для разработки нашего продукта плату Pico как есть. Однако нецелесообразно использовать плату Pico для всех проектов. Могут возникать, например, следующие вопросы:

- размер платы Pico может не подходить для вашего продукта;
- возможно, вам не понадобятся все контакты, доступные на плате Pico;
- микроконтроллер RP2040 стоит гораздо дешевле готовой платы Pico, и у вас уже есть необходимые дополнительные компоненты.

В следующем разделе мы обсудим некоторые варианты замены готовой платы Raspberry Pi Pico в нашем дизайне.

Замена Pico

Чтобы заменить Pico своим дизайном, основанным на микроконтроллере RP2040, нам необходимо следовать образцовому дизайну (*reference design*), а также рекомендациям по использованию (*application notes*) для микроконтроллера RP2040. Компания Raspberry Pi Foundation предоставляет подобный документ под названием «Hardware design with RP2040», доступный по адресу **https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf**). В документе подробно описаны все компоненты, необходимые для правильного построения схемы с RP2040.

Мы включим схему, в которой используется микроконтроллер RP2040 вместо Pico, в репозиторий GitHub этой главы. Есть две альтернативы созданию вашего дизайна с нуля. Первый из них называется PGA2040 от фирмы Pimoroni.

Pimoroni PGA2040

PGA2040 (показан на рис. 11.18) представляет собой плату, содержащую минимальную схему, необходимую для встраивания микроконтроллера RP2040 в ваш проект. Выводы микроконтроллера RP2040 доступны в виде сетки отверстий. Pimoroni предоставляет файлы Autodesk Eagle для использования в ваших проектах. PGA2040 можно приобрести на сайте **https://shop.pimoroni.com/products/pga2040**.

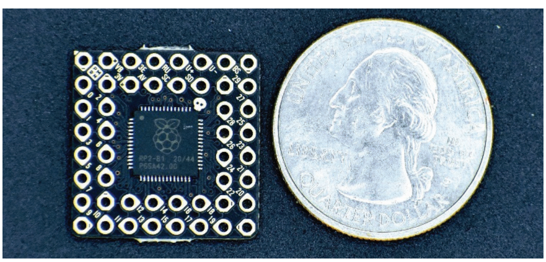


Рисунок 11.18. Плата PGA2040

Альтернативой PGA2040 является RP2040 Stamp. Мы обсудим этот продукт в следующем разделе.

RP2040 Stamp

RP2040 Stamp (показан на рис. 11.19 далее) представляет собой переходную плату размером около 1 квадратного дюйма. Как и в случае с PGA2040, Stamp предоставляет доступ ко всем выводам микроконтроллера. Вы можете заставить Stamp работать сразу, подав на плату питание 5 В. Создатель Stamp предоставил файлы проектов как для Autodesk Eagle, так и для KiCad. RP2040 Stamp доступен по адресу **https://shop.pimoroni.com/products/rp2040-stamp**.

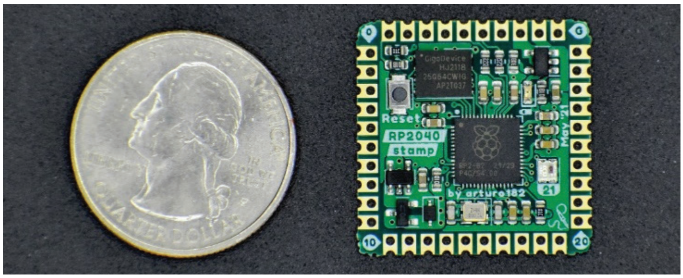


Рисунок 11.19. RP2040 Stamp

На следующем рисунке показан Stamp, распаянный на несущей плате, выполненной в форм-факторе Arduino. Это позволяет использовать Stamp с платами расширений Arduino:

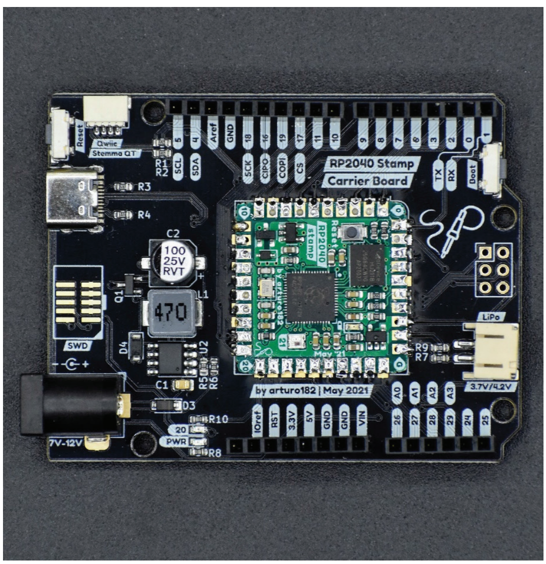


Рисунок 11.20. Stamp на несущей плате в форм-факторе Arduino

В этом разделе мы рассмотрели две альтернативы, которые можно использовать вместо Pico. Подведем итоги этой главы.

Итоги

В этой главе мы обсуждали создание законченного продукта с использованием Raspberry Pi Pico. Мы обсудили общую концепцию и проверку отдельных компонентов. Затем выполнили разводку печатной платы с помощью Autodesk Eagle. За этим последовали изготовление, монтаж, сборка и установка платы в корпус.

Мы надеемся, что вам понравилось заглядывать за кулисы и разбираться в этапах создания продукта. В следующей главе мы обсудим различные нюансы и хитрости, которые вам следует иметь в виду, прежде чем мы завершим эту книгу.

Глава 12

Дополнительные возможности работы с Pico

В этой заключительной главе мы рассмотрим некоторые темы, которые были лишь мельком затронуты в предыдущих главах. Мы также обсудим некоторые моменты, которые помогут вам лучше выполнять проекты.

В этой главе мы рассмотрим следующие темы:

- обновление встроенного ПО вашего Pico;
- программирование Pico с помощью Arduino IDE;
- программирование на C/C++ с использованием Pico SDK;
- инструменты, помогающие в создании прототипов и разработке продукта.

Технические требования

Для этой главы не существует строгих требований к оборудованию или программному обеспечению. Вам понадобится Raspberry Pi Pico для обсуждения этих тем. О его приобретении см. врезку «*О приобретении компонентов в России*» на стр. 23.

Видеоролики с кодом в действии для этой главы можно посмотреть по адресу **https://bit.ly/3kH2Cm7**.

Мы начнем с обновления встроенного ПО вашего Pico.

Обновление прошивки Pico

Время от времени вам нужно будет обновлять встроенное ПО Raspberry Pi Pico. Обновление может потребоваться, если вы хотите использовать последнюю версию CircuitPython или MicroPython на вашем Pico, которые либо включают новые функции в Pico, либо исправляют ошибки, выявленные в предыдущих версиях. Для обновления встроенного ПО на Pico необходимо выполнить следующие действия.

1. Мы будем считать, что вы используете либо CircuitPython, либо MicroPython. Рассмотрим пример, в котором мы хотим обновить до последней версии CircuitPython. Первый шаг – загрузить последнюю вер-

сию прошивки CircuitPython, доступную для Raspberry Pi Pico, с сайта **https://circuitpython.org/board/raspberry_pi_pico**.

2. Допустим, что Pico, который необходимо обновить, работает под управлением CircuitPython версии 6.3.0. Эту информацию можно найти в интерпретаторе CircuitPython, запущенном на Pico, как показано на следующем рисунке:

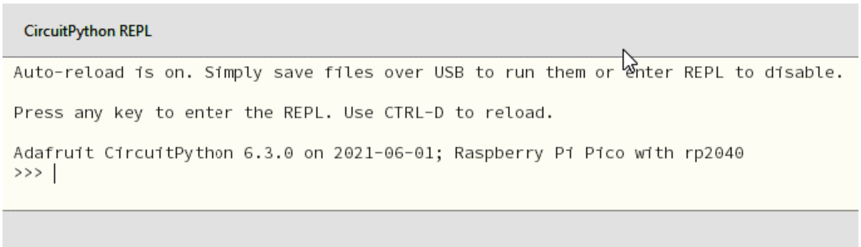


Рисунок 12.1. Версия микропрограммы CircuitPython

На момент написания этой книги последней версией CircuitPython была версия 7.2.0. Мы загрузили последнюю версию прошивки, как указано выше.

3. Теперь следует перевести Pico в режим загрузчика. Подключите USB-кабель к компьютеру, нажмите и удерживайте кнопку загрузки (выделена на следующем рисунке) и подключите кабель micro-USB к Pico:

Рисунок 12.2. Расположение кнопки загрузки на Pico

4. Pico должен обнаружиться на вашем компьютере как запоминающее устройство. Содержимое Pico должно выглядеть следующим образом:

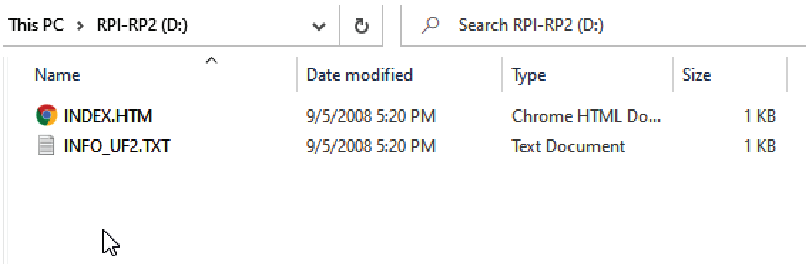


Рисунок 12.3. Содержимое Pico в режиме загрузчика

5. Скопируйте загруженную ранее прошивку на указанное запоминающее устройство.
6. Ваш Pico должен сбросить настройки и обнаружиться как устройство CircuitPython (на следующем рисунке Pico показывается как запоминающее устройство с именем *CIRCUITPY*):

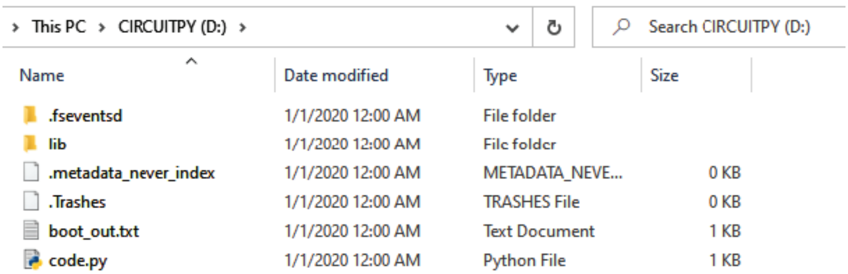


Рисунок 12.4. Pico в списке устройств хранения данных после обновления встроенного ПО

7. Теперь вы можете проверить версию встроенного ПО, запустив интерпретатор CircuitPython в среде Mu IDE.

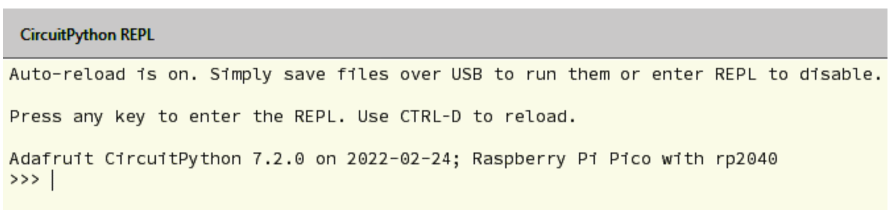


Рисунок 12.5. Последняя версия микропрограммы, установленная на Pico

8. Кроме этого, вы можете найти версию встроенного ПО в файле *boot_out.txt*, который можно найти на Pico, как показано на рис. 12.6.
9. Файл содержит ту же информацию, что отображается в интерпретаторе CircuitPython:

```
Adafruit CircuitPython 7.2.0 on 2022-02-24; Raspberry Pi Pico with rp2040
Board ID:raspberry_pi_pico
```

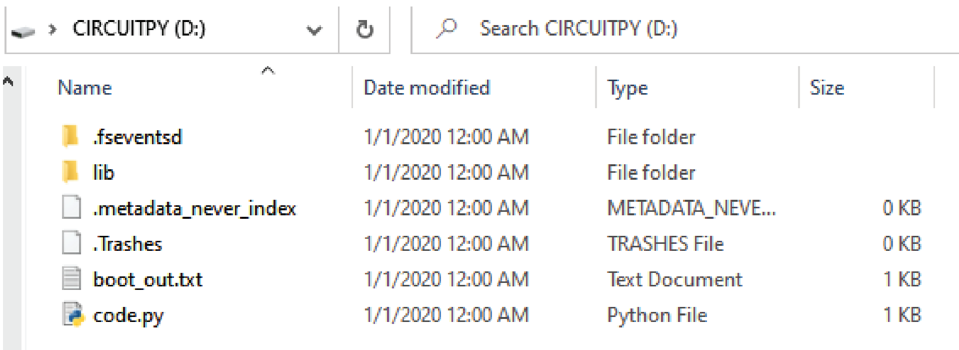


Рисунок 12.6. Файл *boot_out.txt* на Pico, распознанном как запоминающее устройство

Вы можете использовать метод, описанный в данной главе, для обновления встроенного ПО или переключения с CircuitPython на MicroPython и наоборот. Мы также рекомендуем обновлять Pico время от времени и проверять, чтобы убедиться, что у вас установлена последняя версия прошивки.

В следующем разделе мы обсудим программирование Pico с помощью Arduino IDE.

Программирование Pico с помощью Arduino IDE

В проектах, которые мы обсуждали в этой книге, мы программировали на CircuitPython или MicroPython. В этом разделе мы обсудим программирование Pico с использованием платформы Arduino. Платформа Arduino – это аппаратное и программное обеспечение с открытым исходным кодом, предназначенное для облегчения создания макетов и прототипов. Программное и аппаратное обеспечение Arduino простое в использовании и доступно по низкой цене. Это позволило быстро создавать прототипы даже людям с гуманитарным образованием. Существует обширная экосистема дополнительного оборудования, доступного для платформы Arduino. Платформа Arduino позволяет программировать Pico на C++. Она также предоставляет доступ ко всей экосистеме дополнительного оборудования, имеющегося для платформы Arduino. Вы можете узнать больше о платформе Arduino на официальном сайте **arduino.cc**, а также из многочисленной литературы, посвященной этой платформе.

Загрузка и установка Arduino IDE

Вы можете загрузить последнюю версию Arduino IDE по адресу **https://www.arduino.cc/en/software**. Она доступна для операционных систем Windows, Linux и Mac. При загрузке программного обеспечения вам будет предложено внести пожертвование. Мы настоятельно рекомендуем делать пожертвования на дальнейшую разработку аппаратного и программного обеспечения с открытым исходным кодом.

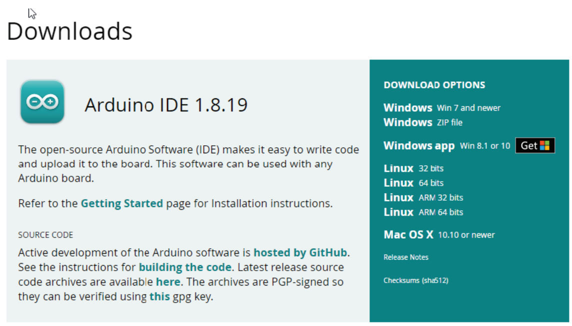


Рисунок 12.7. Загрузка среды разработки Arduino IDE

Процесс установки очень прост[107]. В следующем разделе обсудим установку пакета поддержки платы Pico.

Установка пакета для платы Pico

После установки Arduino IDE необходимо установить пакеты, предназначенные для для поддержки Raspberry Pi Pico на платформе Arduino.

1. После запуска Arduino IDE должно появиться окно, показанное на следующем рисунке:

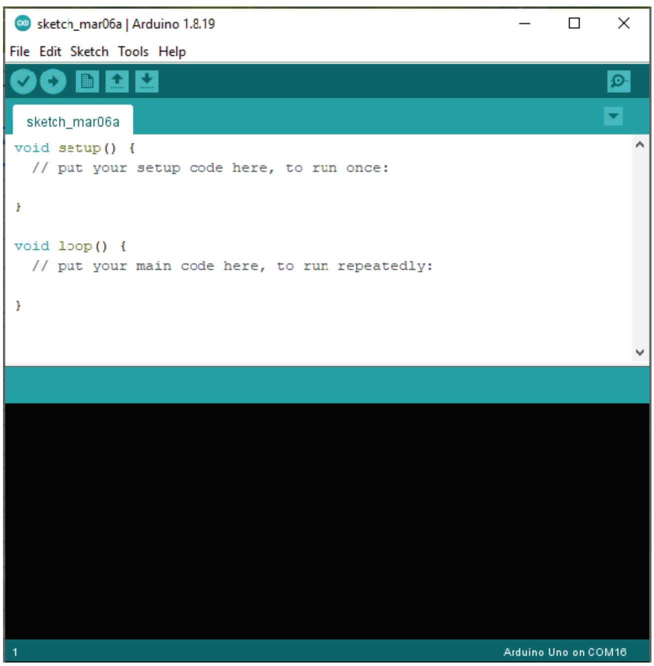


Рисунок 12.8. Новое окно Arduino IDE

107 В частности, не требуется скачивать и устанавливать дополнительные языковые пакеты – язык интерфейса Arduino IDE можно просто указать в настройках (пункт **File** | **Preferences**) после установки. В связи с этим далее все пункты меню, относящиеся к Arduino IDE, будут указываться в двойном виде: по-английски, как у авторов, и по-русски в расчете на русскоязычную версию у читателя. – *Прим. перев.*

2. Теперь перейдите в раздел **Tools | Board | Boards Manager** (*Инструменты | Плата | Менеджер плат*). Должно появиться новое окно, содержащее все пакеты поддержки плат, которые могут быть установлены. В строке поиска найдите пакеты, поддерживающие Pico, как показано на рис.12.9.

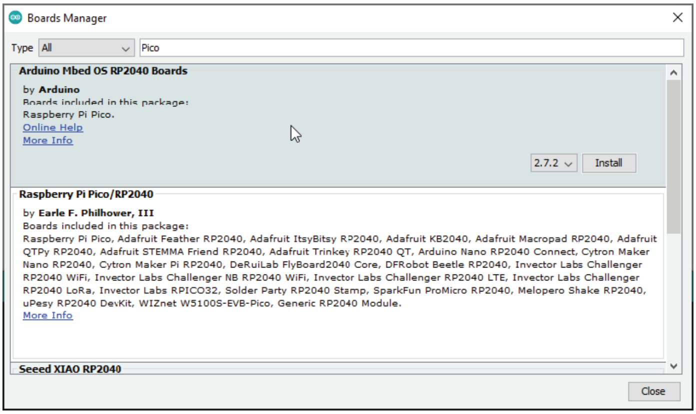


Рисунок 12.9. Менеджер плат Arduino

3. Установите пакет **Arduino Mbed OS RP2040**. Этот пакет включает в себя поддержку Raspberry Pi Pico.
4. Чтобы программировать Pico, нам нужно выбрать необходимую плату. Вы можете выбрать плату в разделе **Tools | Board** (*Инструменты | Плата*), указав пункт **Arduino Mbed OS RP2040 | Raspberry Pi Pico**, как показано на рисунке:

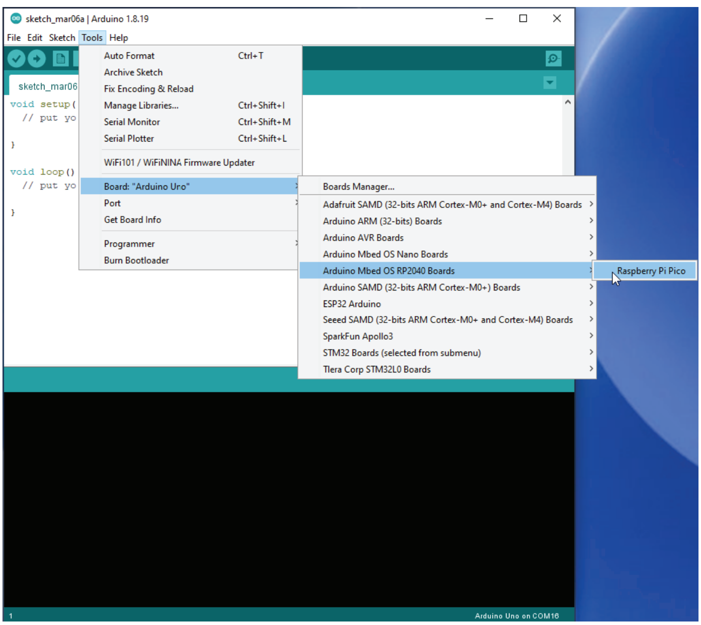


Рисунок 12.10. Выбор Raspberry Pi Pico

5. Загрузим пример мигания встроенным светодиодом Raspberry Pi Pico, теперь доступный в Arduino IDE. Перейдите в раздел **File | Examples** (*Файл | Примеры*), в подраздел **Basics | Blink**, как показано на рисунке:

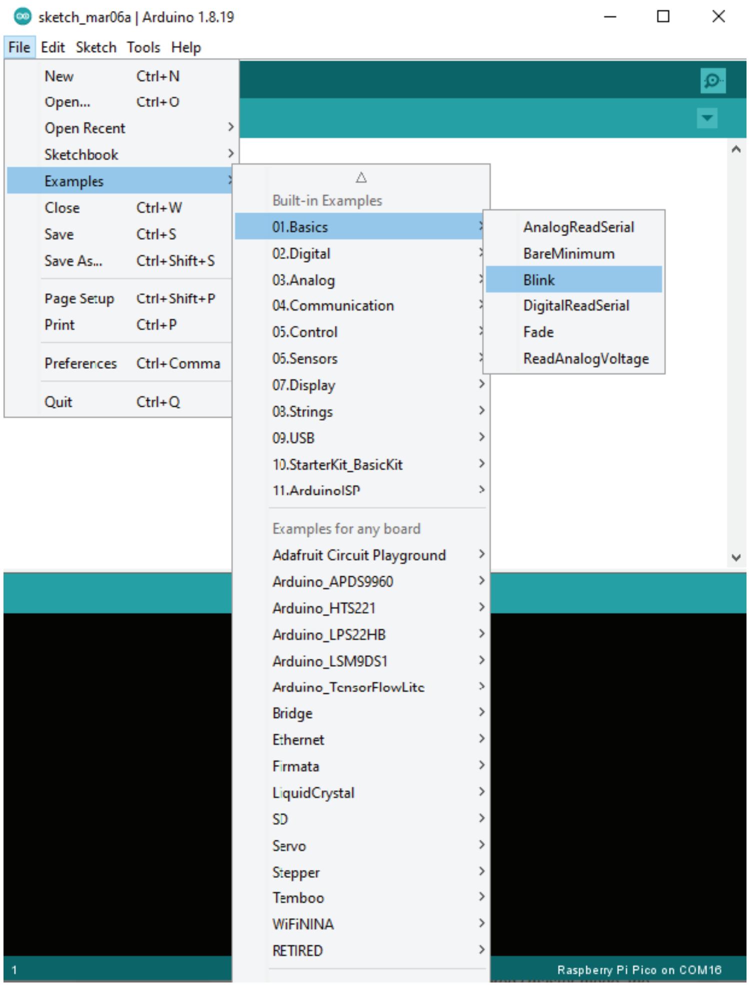


Рисунок 12.11. Загрузка примера мигания светодиодом

6. Должен загрузиться код примера, как показано на следующем скриншоте. Теперь код необходимо скомпилировать, чтобы проверить на отсутствие каких-либо ошибок. В данном случае мы пытаемся проверить, правильно ли работает пакет поддержки платы после установки. Кнопка компиляции представляет собой галочку, расположенную в верхнем левом углу[108] (выделена красным прямоугольником на рисунке):

[108] Компиляцию можно запустить также через меню **Sketch** (*Скетч*) или сочетанием клавиш <**Ctrl**+**R**>. Предварительно необходимо убедиться, что в правом нижнем углу окна указывается именно плата Raspberry Pi Pico (см. рис. 12.11). Если это не так, выберите нужную плату в пункте **Tools** | **Board** (*Инструменты* | *Плата*). – *Прим. перев.*

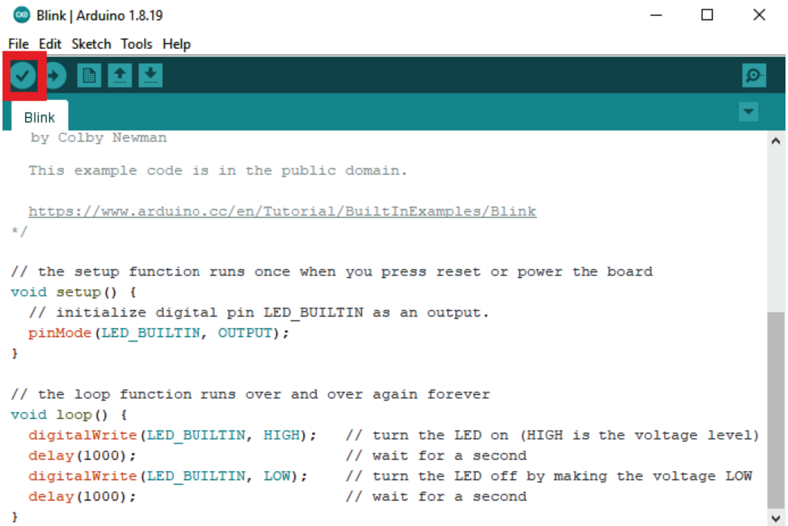


Рисунок 12.12. Компиляция примера Blink

7. После успешной компиляции внизу будет показано сообщение о завершении процесса вместе с результатом работы компилятора, как показано на рис. 12.13.

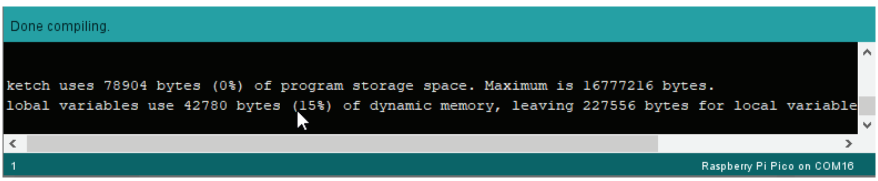


Рисунок 12.13. Результат компиляции

8. Теперь загрузим код в Pico. Чтобы это сделать, необходимо сначала войти в режим загрузчика. Мы обсуждали перевод Pico в режим загрузчика в разделе «*Обновление прошивки Pico*»: при подключении кабеля micro-USB следует нажать кнопку загрузки на плате. Затем в Arduino IDE нажмите на значок загрузки[109] (кнопка со стрелкой рядом с кнопкой компиляции, выделена на рисунке красным прямоугольником):

Рисунок 12.14. Кнопка загрузки

9. Сначала файл будет заново скомпилирован (для проверки возможных изменений перед загрузкой), и только потом начнется загрузка. Как только скомпилированный двоичный файл будет загружен, внизу долж-

109 Или сочетание клавиш <**Ctrl**+**U**>. Предварительно следует убедиться, что в правом нижнем углу окна указывается COM-порт, соответствующий подключенному Pico (см. рис. 12.11). Нужный порт можно указать в пункте **Tools** | **Port** (*Инструменты* | *Порт*); он запомнится и на следующие сеансы работы с Arduino IDE, пока вы не укажете новый. – *Прим. перев.*

но появиться сообщение с указанием **Done uploading** (*Загрузка завершена*), как показано на следующем рисунке:

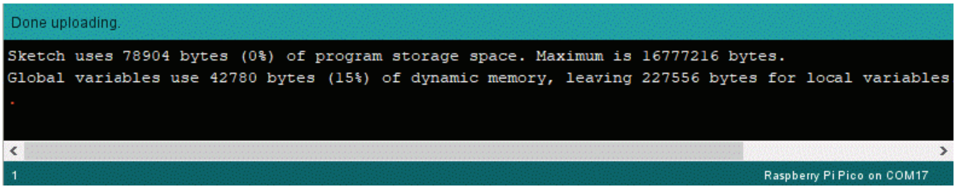


Рисунок 12.15. Скомпилированный двоичный файл загружен в Pico

10. Как только код будет загружен, встроенный зеленый светодиод на плате Pico начнет мигать с интервалом 1 секунда.
11. Давайте рассмотрим пример кода, который мы только что скомпилировали и загрузили в Pico. Файл кода Arduino называется скетчем. Типичный скетч Arduino состоит из двух функций, а именно `setup()` и `loop()`.
12. Функция `setup()` – место, где происходит инициализация программы. Для примера мигания светодиода мы устанавливаем вывод GPIO 25, к которому подключен встроенный светодиод, в качестве выходного. В следующем фрагменте кода `LED_BUILTIN` означает вывод GPIO 25 (это определено в пакете поддержки платы для Pico):

```
void setup() {
pinMode(LED_BUILTIN, OUTPUT);
}
```

13. Функция `loop()` содержит код, который должен выполняться внутри бесконечного цикла. В этом примере нам нужно мигать светодиодом с интервалом в 1 секунду. Для этого нам нужно включать и выключать светодиод в течение 1 секунды между вызовами функции. В следующем фрагменте кода функция `delay()` принимает аргументы в миллисекундах. Поэтому мы использовали число (1000), чтобы ввести 1-секундную задержку:

```
void loop() {
digitalWrite(LED_BUILTIN, HIGH);
delay(1000);
digitalWrite(LED_BUILTIN, LOW);
delay(1000);
}
```

Попробуйте установить задержку в 100 мс, чтобы увидеть эффект быстрого мигания светодиода. Поиграйте с интервалом, чтобы определить наименьший интервал, при котором мигание еще заметно невооруженным глазом.

Теперь, когда мы получили удовольствие от Arduino IDE, обсудим разработку приложений для Raspberry Pi Pico с использованием официального SDK от Raspberry Pi foundation.

Программирование на C/C++ с использованием Pico SDK

В этом разделе мы обсудим разработку приложений с использованием официального пакета разработки программного обеспечения (**software development kit, SDK**) от Raspberry Pi foundation. SDK позволяет разрабатывать приложения на C/C++, а Raspberry Pi foundation выпустила подробную документацию по API, которая поможет вам разрабатывать приложения с применением Pico. Документация доступна по адресу **https://bit.ly/3IpBFwW**.

Документация содержит информацию об установке SDK, компиляции первого примера и загрузке его на Pico. Мы рекомендуем работать по этой документации, если вы заинтересованы в программировании на C/C++.

> **Примеры кода на C/C++**
>
> Мы включили примеры кода C/C++ для Raspberry Pi Pico в репозиторий этой книги на GitHub.

Отладчики для Raspberry Pi Pico

Если вы учитесь программировать на C/C++, полезно иметь средство отладки для устранения ошибок в коде. Raspberry Pi Pico имеет порт последовательной отладки (**Serial Wire Debug**, **SWD**), который позволяет устранять неполадки в коде. Его наличие позволяет вам просматривать код построчно и наблюдать за поведением контроллера, а также за состоянием переменных. Мы рекомендуем следующие статьи по настройке отладчика при разработке приложений:

- если вы используете Raspberry Pi для разработки приложений, в следующей статье описывается сопряжение SWD-порта Raspberry Pi Pico с Raspberry Pi: **https://bit.ly/3JwX8oT**;
- в следующей статье описывается настройка Pico в качестве отладчика (также известного как *picoprobe*) для другого Pico: **https://bit.ly/3qkHt4i**.

В следующем разделе мы обсудим некоторые инструменты, которые могут помочь в создании прототипов и разработке продукта.

Инструменты для создания прототипов и разработки продукта

Мы начнем с обсуждения макета от Саймона Монка.

Макетная плата с указанием разводки выводов Pico

В главе 1 «*Начало работы с Raspberry Pi Pico*» мы рассматривали разводку выводов Pico (см. рис. 1.2). Может быть, неудобно постоянно заглядывать в этот рисунок во время работы. Мы наткнулись на макетную плату от Саймона Монка (под лейблом MonkMakes), на которой обозначены выводы Pico (см. следующий рисунок). Это позволяет быстрее и без ошибок выполнять соединения во время разработки.

На рис. 12.16 синие буквы обозначают номера выводов GPIO, вывод заземления обозначен буквой G, а основные выводы питания обозначены красными буквами. Схема маркировки предполагает, что Pico монтируется на макетной плате следующим образом:

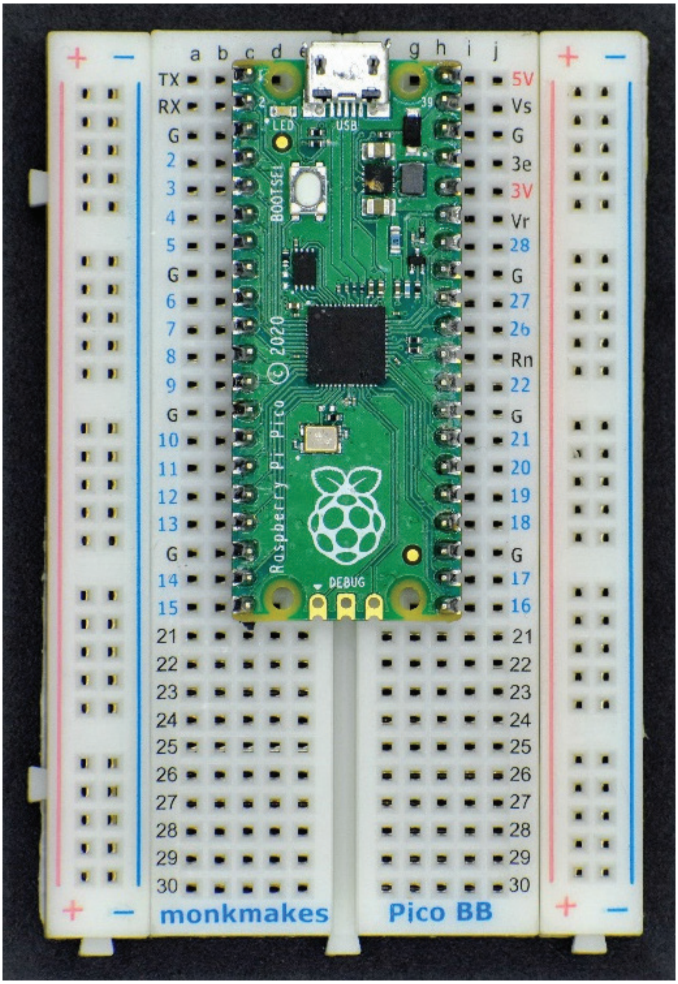


Рисунок 12.16. Макетная плата для Pico

В следующем разделе мы рассмотрим получение профиля потребления для создания продуктов с батарейным питанием.

Получение профиля потребления вашего продукта

Измерители профиля потребления тока вашим изделием служат двум основным целям:

- чтобы помочь вам понять общую структуру энергопотребления вашего устройства. Это поможет вам подобрать батарею или источник питания для вашего приложения;
- в устройствах с батарейным питанием построитель профиля потребления может помочь измерить ток ожидания и выявить утечки тока в вашей конструкции.

Мы разыскали два построителя профиля потребления – подешевле и подороже.

Nordic Power Profiler Kit

Комплект Nordic Power Profiler Kit от Nordic Semiconductors позволяет измерять профили питания устройств с входным напряжением от 1,8 В до 3,3 В и током до 70 мА. Этот построитель профиля потребления идеально подходит для приложений, где устройство питается от одноэлементной батареи[110].

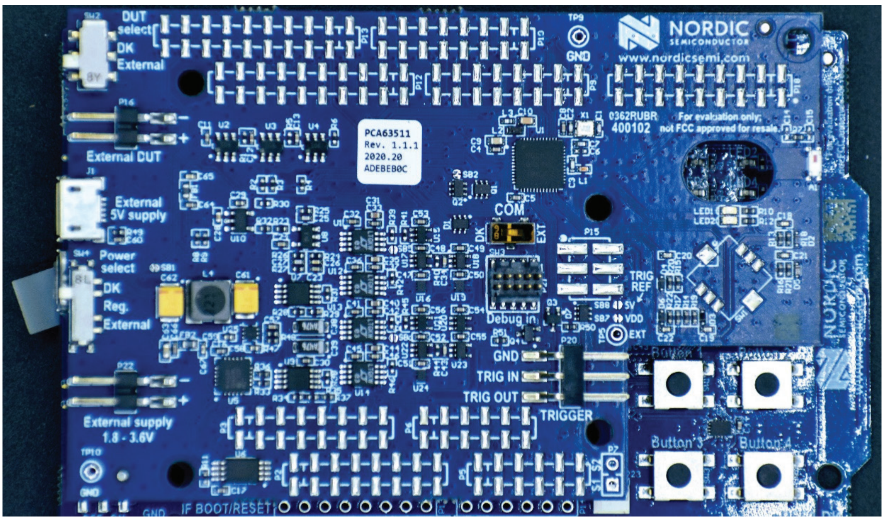


Рисунок 12.17. Комплект Nordic Power Profiler Kit

Комплект можно найти по адресу **https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit**[111].

В следующем разделе мы обсудим Joulescope.

Joulescope

Joulescope – это надежный и высокоточный анализатор потребления энергии, который может выдерживать напряжение от 1 до 15 В и ток до 3 А с разре-

---

110 Очевидно, авторы имеют в виду литиевые элементы («монетки»), так как другие типы (обычные пальчиковые щелочные, серебряно-цинковые «таблетки») обладают значительно меньшим начальным напряжением в расчете на один элемент. – *Прим. перев.*

111 В России комплект Nordic Power Profiler можно найти в интернет-магазинах, например **https://www.chipdip.ru/product/nrf6707-1**. Надо учитывать, что для работы с этим комплектом необходимо дополнительное оборудование (см. описание по указанным ссылкам). – *Прим. перев.*

шением 1,5 нА. Joulescope стоит дорого, но он полезен, если вы разрабатываете профессиональные продукты с широким диапазоном потребления тока.

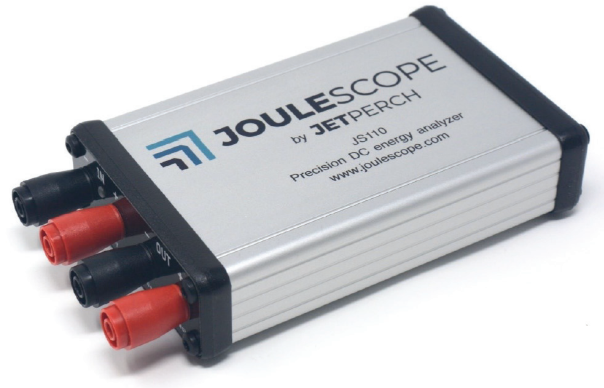


Рисунок 12.18. Joulescope для построения профиля потребления мощности

В следующем разделе мы обсудим программирование PIO на микроконтроллере RP2040.

Программирование PIO

PIO означает **Programmable Input/Output**, программируемый ввод/вывод. Это позволяет вам писать ассемблерные программы для реализации функций и интерфейсов, которые обычно недоступны на микроконтроллере RP2040. Например, микроконтроллер RP2040 содержит два интерфейса UART. Если вы использовали оба UART и вам нужен третий, то можете реализовать его с помощью PIO. Согласно документации, существует два блока PIO с четырьмя конечными автоматами, с помощью которых вы можете реализовать следующие интерфейсы:

- параллельные шины 8080 и 6800;
- I2C;
- 3-выводной I2S[112];
- SDIO[113];
- SPI, DSPI и QSPI[114];
- UART;
- DVI или VGA.

PIO позволяют запускать эти интерфейсы независимо от основного процессора. На рис. 12.19 показан один блок PIO в микроконтроллере:

112 Integrated Inter-chip Sound, I2S – последовательной интерфейс, использующийся для передачи ИКМ-аудиоданных при соединении цифровых аудиоустройств. – *Прим. перев.*

113 SDIO – расширенный стандарт интерфейса SD-карт памяти. – *Прим. перев.*

114 Вероятно, авторы имеют в виду сдвоенный (Double SPI) и счетверенный (Quad SPI) SPI-интерфейсы (буквы D и Q в этих сокращениях могут иметь и другие толкования). – *Прим. перев.*

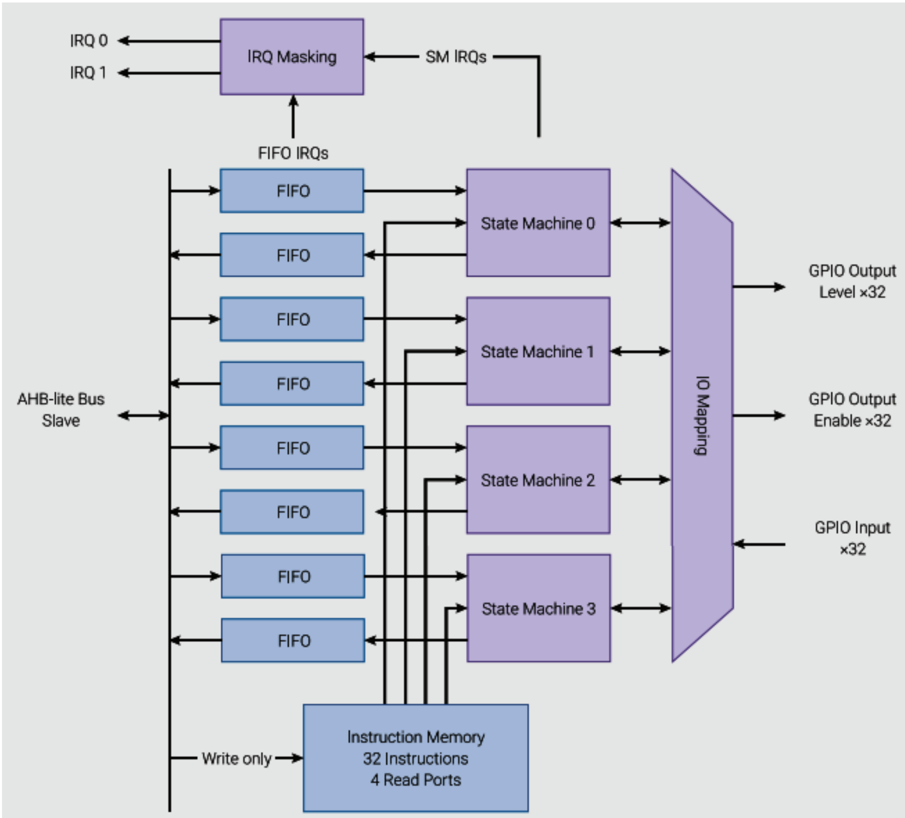


Рисунок 12.19. Одиночный блок вывода PIO
(источник изображения: спецификация RP2040)

Разработчики подключили контроллер к VGA-дисплею с помощью PIO. Мы наткнулись на следующие ресурсы, которые помогут вам использовать PIO на Raspberry Pi Pico:

- глава 3 спецификации микроконтроллера RP2040: **https://bit.ly/3wnG0ht**;
- использование CircuitPython для программирования PIO: **https://bit.ly/3N3Clvp**;
- статья Digikey о программировании PIO на C/C++: **https://bit.ly/3qlcOyr**.

Несколько слов в завершение этой книги.

Итоги

Мы надеемся, что вам понравились проекты, которые мы обсуждали в нашей книге. Мы завершили повествование главой с некоторыми советами и подробностями, которые собрали во время работы над другими главами. Мы будем поддерживать репозиторий GitHub этой книги с последними примерами кода и обновлениями по мере необходимости. Рекомендуем отправить сообщение о проблеме в этот наш репозиторий, если вдруг столкнетесь с какими-либо ошибками во время работы над проектами, рассмотренными в данной книге.

Мы как проектировщики и гражданские ученые рады, что вы присоединились к нам в этом путешествии по разработке приложений с помощью Raspberry Pi Pico. Движение разработчиков-любителей значительно выросло и превратилось в мощную силу по всему миру, затрагивающую людей всех возрастов, рас и происхождения. Гражданское научное движение усваивает некоторые из тех же ключевых принципов сотрудничества и открытости, стремясь решать важные личные или общественные проблемы с минимальными затратами.

Мы также надеемся, что помимо этого учителя, студенты, индивидуальные любители и предприниматели также найдут на этих страницах что-то, что позволит расширить знания, а также заработать на коммерческих решениях. Рассмотренные проекты могут быть распространены на другие архитектуры и устройства. Возможно, вы стремитесь к повышению эффективности затрат или снижению габаритов, и мы надеемся, что эта книга и содержащиеся в ней проекты могут оказаться полезными, каковы бы ни были ваши цели.

Мы верим, что вам будет весело работать с Raspberry Pi Pico. Счастливого созидания!

Предметный указатель

Книги издательства «ДМК ПРЕСС» можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).

Адрес: г. Москва, пр. Андропова, 38;
Тел.: **+7(499) 782-38-89**. Электронная почта: **books@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине:
www.galaktika-dmk.com.

Сай Яманур
Шрихари Яманур

Raspberry Pi Pico в любительских проектах

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Зам. главного редактора *Сенченкова Е. А.*
Перевод *Ревич Ю. В.*
Корректор *Синяева Г. И.*
Верстка *Паранская Н. В.*
Дизайн обложки *Мовчан А. Г.*

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 22,91. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**

Raspberry Pi Pico – последнее дополнение к семейству продуктов Raspberry Pi, представленное компанией Raspberry Pi Foundation. Основанный на чипе RP2040, Pico – миниатюрный и быстрый микроконтроллер, обладающий достаточной мощностью для выполнения широкого спектра приложений.

Эта книга поможет вам использовать возможности Pico для создания инновационных продуктов. Вы познакомитесь с множеством проектов – от домашней автоматизации до создания большого семисегментного дисплея для демонстрации качества воздуха. К концу чтения вы даже создадите собственного робота!

В числе рассматриваемых тем:

- программирование в средах MicroPython и CircuitPython;
- знакомство со встроенным и дополнительным оборудованием;
- периферийные устройства Pico;
- последовательные интерфейсы, доступные на Pico;
- различные варианты плат Raspberry Pi Pico;
- советы и хитрости при разработке с помощью Pico.

Издание адресовано любителям DIY (Do It Yourself – «Сделай сам»), а также будет полезно программистам, инженерам, студентам и школьникам, которые хотят разрабатывать встроенные системы для домашней автоматизации, интернета вещей, робототехники и др.

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
books@alians-kniga.ru

Packt

ДМК пресс
издательство
www.дмк.рф

ISBN 978-5-93700-165-8

9 785937 001658 >