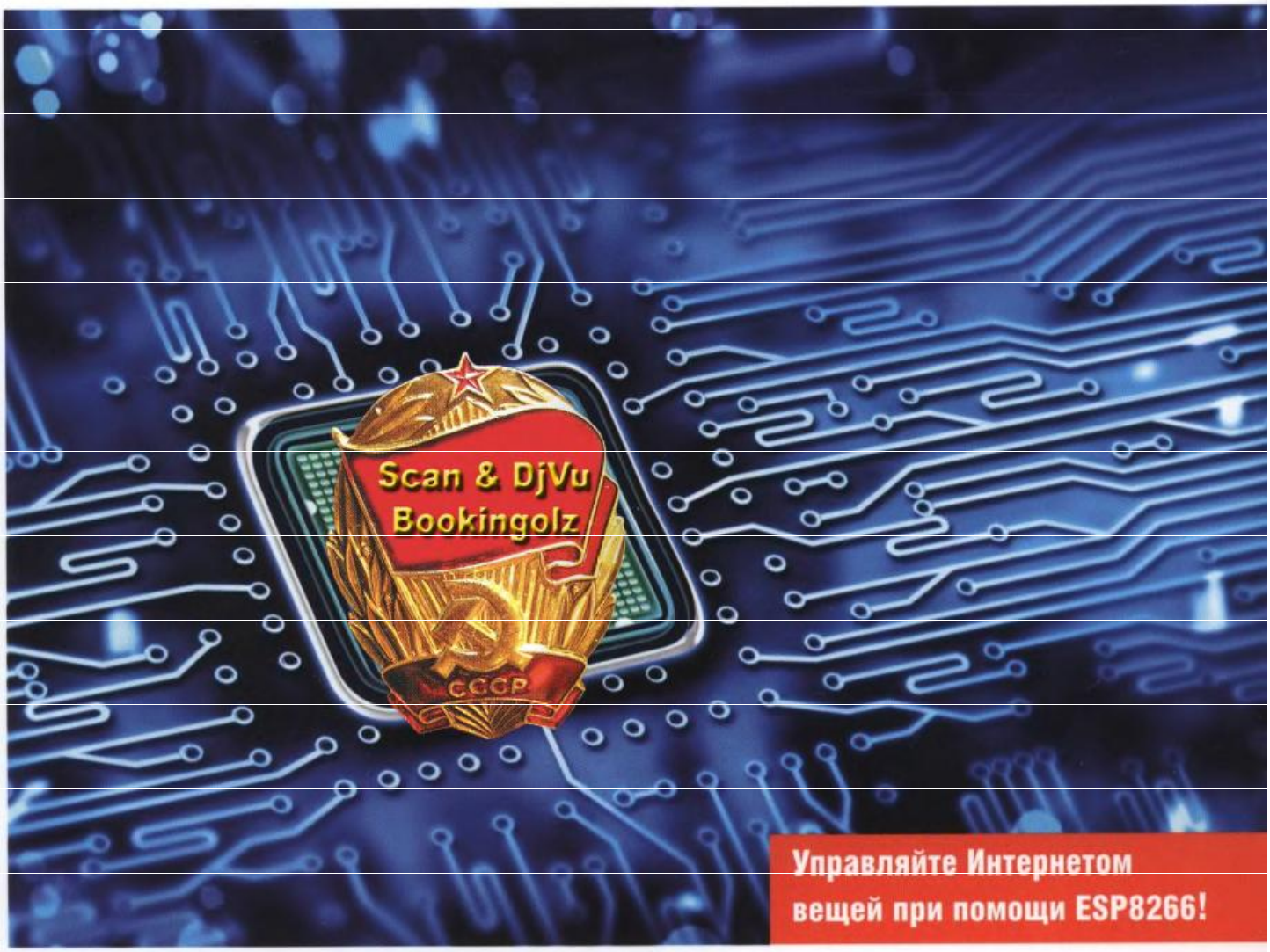
Электроника

Марко Шварц

bhv®



Управляйте Интернетом вещей при помощи ESP8266!

Интернет вещей с ESP8266

М. Шварц
Интернет вещей с ESP8266
bhv®

Internet of Things with ESP8266

Build amazing Internet of Things projects using the ESP8266 Wi-Fi chip

Marco Schwartz

[PACKT] PUBLISHING

open source
community experience distilled

BIRMINGHAM - MUMBAI

Марко Шварц

Интернет вещей с ESP8266

Санкт-Петербург
«БХВ-Петербург»
2018

УДК 004
ББК 32.973.26-018.2
Ш33

Шварц Марко

Ш33 Интернет вещей с ESP8266: Пер. с англ. — СПб.: БХВ-Петербург, 2018. — 192 с.: ил. — (Электроника)

ISBN 978-5-9775-3867-1

Описан процесс разработки недорогих, но эффективных устройств для Интернета вещей на основе популярного микроконтроллера с функцией Wi-Fi ESP8266. Проекты доступны для повторения новичкам в области Интернета вещей, имеющим начальный опыт работы с платформой Arduino. Рассказано, как считывать, отправлять и отслеживать данные через облачные сервисы и дистанционно управлять устройствами откуда угодно, применять ESP8266 для взаимодействия с социальными сетями Twitter и Facebook, отправлять пользователям ESP8266 сообщения по email, SMS и push-каналам, организовывать межмашинное взаимодействие без участия человека, в том числе через облако. На практических примерах показано построение простой системы домашней автоматики с управлением через облако, а также развертывание собственной облачной платформы. Описано, как сделать дверной замок с управлением через облако, физический индикатор курса цифровой валюты, беспроводное садовое оборудование и многие другие устройства на основе ESP8266.

Для читателей, интересующихся электроникой и робототехникой

УДК 004
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Перевод с английского	*Валерия Яценкова*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Оформление обложки	*Марины Дамбиевой*

Подписано в печать 29.09.17.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 15,48.
Тираж 1500 экз. Заказ № 5311.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ООО "Печатное дело",
142300, МО, г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-1-78646-802-4 (англ.)
ISBN 978-5-9775-3867-1 (рус.)

Оглавление

Об авторе

Марко Шварц (Marco Schwartz) — инженер-электротехник, предприниматель и блогер. Он получил диплом магистра электротехники и компьютерных технологий в престижной французской Высшей школе электротехники (École Supérieure d'Électricité, Supélec) и диплом магистра по микроэлектронике в Федеральной политехнической школе Лозанны (Ecole Polytechnique Fédérale de Lausanne), Швейцария.

Более пяти лет Марко занимается разработками в области электротехники. Его технические интересы сфокусированы вокруг электроники, автоматики «умного дома», платформ Arduino и Raspberry Pi, открытых проектов и 3D-печати.

Он создал несколько веб-сайтов про Arduino, включая сайт **openhomeautomation.net**, где рассказывается о том, как построить оборудование для домашней автоматики на основе открытых проектов.

Марко написал еще несколько книг, в том числе про домашнюю автоматику и Arduino: «Home Automation With Arduino: Automate Your Home Using Open-source Hardware», а также о том, как создавать на основе Arduino проекты Интернета вещей: «Internet of Things with the Arduino Yun»[1].

[1] Насколько мне известно, на русском языке они не издавались. — *Прим. пер.*

О рецензенте

Каталин Батрину (Catalin Batrinu) окончил Бухарестский политехнический университет по специальности «Электроника, телекоммуникационные и информационные технологии». Последние 16 лет он занимается разработкой программного обеспечения в области телекоммуникаций.

Каталину довелось работать со всеми версиями сетевых протоколов и технологий — и со старыми, и с новыми современными — так что он стал свидетелем и участником всего процесса развития телекоммуникационной промышленности.

Он знаком с реализацией различных сетевых протоколов — от оконечных устройств и магистральных маршрутизаторов до скоростных провайдерских коммутаторов на различных аппаратных платформах от Wintegra и Broadcom.

Интернет вещей стал для Каталина Батрину очередным этапом развития, и сегодня он сотрудничает с различными компаниями, создавая мир завтрашнего дня, который сделает нашу жизнь более комфортной и безопасной.

Воспользовавшись микроконтроллером ESP8266, Каталин Батрину разработал прототипы таких устройств, как автоматика полива растений, умные розетки, приводы жалюзи окон, цифровые системы управления освещением — все они способны управляться через облачные сервисы с помощью приложений, установленных на смартфоне. Для ESP8266 существует даже MQTT-брокер с реализацией мостового подключения и сокет-сервером[1]. Вскоре эти устройства станут частью повседневной жизни и будут радовать нас своей функциональностью.

Блог Каталина Батрину находится по адресу: **http://myesp8266.blogspot.com**.

---

[1] MQTT (Message Queue Telemetry Transport) — упрощенный сетевой протокол, работающий поверх TCP/IP. Используется для обмена сообщениями между устройствами по принципу «издатель-подписчик». — *Ред.*

Издательство «Packt»

Электронные книги, скидки и многое другое

Знаете ли вы, что издательство «Packt» для каждой опубликованной книги предлагает электронные версии в форматах PDF и ePub? Купив печатную книгу, вы сможете на сайте **www.packtpub.com** получить ее электронное издание со скидкой для владельцев печатной версии[1]. С вопросами обращайтесь по адресу: **customercare@packtpub.com**.

На сайте **www.packtpub.com** вы также можете найти множество бесплатных технических публикаций, подписаться на бесплатные рассылки и получать уникальные скидки и предложения.

Вам срочно нужны ответы на вопросы в области информационных технологий? У нас есть онлайн-библиотека PacktLib:
https://www2.packtpub.com/books/subscription/packtlib.

Подпишитесь, и вы сможете получить доступ к любой из книг нашей библиотеки.

Что дает подписка?

- Полнотекстовый поиск по всем книгам издательства «Packt».
- Возможность ставить закладки, копировать и распечатывать фрагменты текста из книг.
- Печать книг по запросу и доступ к ним при помощи браузера.

---

[1] Не исключено, что это предложение относится только к владельцам исходного, английского издания книги. — *Ред.*

Предисловие

Интернет вещей (IoT, Internet of Things) — это захватывающая идея, согласно которой все устройства вокруг нас подключены к Интернету и общаются не только с нами, но и друг с другом. Ожидается, что к 2020 году в Сеть выйдут около 50 миллиардов устройств.

С другой стороны, существует микросхема ESP8266 — маленький и дешевый (стоимостью менее 5 долларов), но мощный чип со встроенным модулем Wi-Fi, который весьма легко программировать. Очевидно, что это прекрасный инструмент для разработки качественных и недорогих проектов для Интернета вещей. В этой книге мы и займемся изучением всего, что понадобится для создания проектов Интернета вещей на основе ESP8266.

О чем эта книга?

- *Глава 1 «Первые шаги с ESP8266»* — расскажет вам, как правильно выбрать отладочную плату на основе ESP8266 и загрузить в память микросхемы первую программу.
- *Глава 2 «Первые проекты на ESP8266»* — пояснит основные принципы работы ESP8266 на примере простейших проектов.
- *Глава 3 «Сохраняем данные в облако»* — окунет вас в глубины темы, которой посвящена книга, и расскажет о проекте, способном сохранять данные измерений в облачном хранилище.
- *Глава 4 «Управляем устройствами отовсюду»* — покажет, как при помощи ESP8266 управлять устройствами из любой точки мира.
- *Глава 5 «Взаимодействие с веб-сервисами»* — расскажет об использовании ESP8266 для общения с такими веб-сервисами, как Twitter.

- *Глава 6 «Общение между устройствами»* — пояснит, как заставить устройства на основе ESP8266 общаться между собой без участия человека.
- *Глава 7 «Отправка уведомлений»* — продемонстрирует отправку через ESP8266 автоматических уведомлений с помощью SMS, электронной почты и push-каналов.
- *Глава 8 «Управляем дверным замком через облако»* — используя полученные ранее знания, построим наше первое физическое устройство: дверной замок, управляемый дистанционно.
- *Глава 9 «Монитор курса биткоина»* — используем ESP8266 для забавного проекта: дисплея, отображающего курс валюты «биткоин» в реальном времени.
- *Глава 10 «Сетевое облачное садоводство»* — усложним задачу и разберемся, как автоматизировать удаленный уход за растениями.
- *Глава 11 «Домашняя автоматика и облачные сервисы»* — покажем, как на основе ESP8266 построить элементы домашней автоматики.
- *Глава 12 «Робот, управляемый через облако»* — используем ESP8266 для управления мобильным роботом из любой точки мира.
- *Глава 13 «Строим собственную облачную платформу для устройств на ESP8266»* — расскажем, как развернуть собственную облачную платформу для своих проектов на ESP8266.

Что понадобится в дополнение к этой книге?

Для всех проектов из этой книги вам понадобится установить среду разработки Arduino IDE. В *главе 1* мы подробно расскажем, как это сделать.

Главы книги написаны с нарастанием сложности. Если ваши знания о платформе Arduino или ESP8266 невелики — не беда. Вы будете учиться по мере чтения книги. Тем не менее, желательно иметь хоть какие-то навыки программирования — особенно на языке C/C++ или JavaScript.

Для кого эта книга?

Эта книга написана для тех, кто хочет создавать функциональные и недорогие проекты Интернета вещей на основе микросхемы ESP8266. Она будет интересна и полезна как новичкам, так и тем, у кого уже есть опыт работы с Arduino и другими подобными платформами.

Обозначения

В этой книге вы найдете несколько вариантов оформления текста, которые соответствуют различным типам информации. Продемонстрируем несколько стилей текста и поясним, что они обозначают.

- Отдельные директивы программного кода, имена таблиц баз данных, вводимые пользователем данные и строки Twitter обозначены шрифтом `Courier`. Например:

  «Для подключения внешних библиотек применяется директива `include`.»

- Имена файлов, а также и расширения имен файлов обозначены шрифтом Arial. Например:

  «Вставьте текст программы в Arduino IDE или откройте файл ch3_1.ino.»

- Фрагменты кода выглядят так:

```
void loop() {
Serial.print(«Connecting to «);
Serial.println(host);
// Use WiFiClient class to create TCP connections
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
Serial.println(«connection failed»);
return;
}
```

- Текст, вводимый и выводимый в командной строке терминальных программ, обозначен **полужирным шрифтом Courier**. Например:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
    /etc/asterisk/cdr_mysql.conf
```

- Новые термины и важные понятия обозначены *курсивным* шрифтом. Например:

  «В проекте мы используем *фреймворк* (набор готовых программных решений)...»

- Надписи на кнопках и элементах меню программ в тексте книги выделены **полужирным** шрифтом. Например:

  «Откройте из меню **Инструменты** | **Платы** окно Менеджера программ и установите поддержку платформы ESP8266.»

-  Этим значком обозначены важные примечания и комментарии.

-  Этим значком обозначены полезные советы и подсказки.

Обратная связь

Мы всегда благодарны читателям за отзывы. Расскажите нам, что вы думаете об этой книге, что вам понравилось или не понравилось. Отзывы читателей помогают нам готовить издания, которые действительно будут для вас полезны.

Для отправки отзыва общего плана достаточно написать нам по адресу электронной почты **feedback@packtpub.com**, указав название книги в теме письма.

Если вы хорошо разбираетесь в какой-либо теме и хотели бы написать книгу или стать соавтором, прочтите руководство для авторов: **www.packtpub.com/authors**.

Поддержка потребителей

Поскольку вы стали правомочным обладателем книги издательства «Packt», мы поможем вам извлечь максимальную пользу из ее покупки.

Скачивание исходных кодов программ

Вы можете скачать исходные коды программ после регистрации на сайте **www.packtpub.com**. Независимо от места приобретения книги, вы можете зарегистрироваться по адресу **www.packtpub.com/support** и получить файлы непосредственно на свою электронную почту.

Для скачивания исходных кодов с сайта издательства «Packt» выполните следующие шаги[1]:

1. Войдите под своим именем или зарегистрируйтесь на сайте.
2. Наведите указатель мыши на вкладку **SUPPORT** в верхней части сайта.
3. Щелкните на пункте **Code Download & Errata**.
4. Введите название книги или часть названия в поле **Search**.
5. Выберите нужную книгу в результатах поиска.
6. Выберите в раскрывающемся поле место покупки книги.
7. Щелкните на ссылке **Code Download**, которая появится ниже этого поля.

После скачивания воспользуйтесь одним из архиваторов для извлечения файлов из архива:

- WinRAR или 7-ZIP — для ОС Windows;
- Zipeg, iZip или UnRarX — для Mac OS;
- 7-Zip или PeaZip — для ОС Linux.

Электронный архив файлов для русского издания

Электронный архив с материалами к этой книге можно скачать с FTP-сервера издательства «БХВ-Петербург» по ссылке **ftp://ftp.bhv.ru/9785977538671.zip** или со страницы книги на сайте **www.bhv.ru** (см. *приложение*).

---

[1] Напомним, что предложения издательства «Packt» могут относиться только к владельцам исходного, английского издания книги. — *Ред.*

Исправления

Несмотря на все усилия и аккуратную работу над книгами, ошибки все-таки иногда проникают в текст. Если вы нашли ошибку в одной из наших книг — в тексте или в программе — мы будем признательны вам за сообщение о ней. Сделав это, вы убережете других читателей от огорчения и поможете нам улучшить следующее издание книги. Если вы нашли ошибку, пожалуйста, сообщите нам, зайдя на сайт по адресу: **http://www.packtpub.com/submit-errata**. Выберите вашу книгу, щелкните по ссылке: **Errata Submission Form** и введите описание вашей поправки. После проверки ваша поправка будет одобрена и размещена на сайте в разделе для соответствующей книги.

Для просмотра ранее внесенных поправок перейдите по адресу: **www.packtpub.com/books/content/support** и введите название книги в поле поиска. Информация о правках размещена под заголовком **Errata**.

Пиратство

Хищение авторских материалов в Интернете стало общей проблемой для всех средств массовой информации. В издательстве «Packt» очень серьезно относятся к защите своих авторских прав и лицензий. Если вы обнаружили незаконную копию одного из наших изданий в любой форме, пожалуйста, незамедлительно свяжитесь с нами по адресу электронной почты: **copyright@packtpub.com** и сообщите нам физический адрес этого места или адрес веб-сайта, чтобы мы смогли принять меры.

Мы благодарны вам за помощь в защите наших авторов и наших усилий по разработке полезных материалов, которые мы создаем для вас.

Вопросы

Если у вас возникли затруднения с любыми аспектами использования этой книги, обращайтесь по адресу **questions@packtpub.com**, и мы постараемся переадресовать ваш вопрос специалисту для наилучшего решения проблемы.

Читатели русского перевода книги могут обращаться с вопросами и пожеланиями по адресу издательства «БХВ-Петербург»: **mail@bhv.ru**.

Предисловие к русскому изданию

Уважаемый читатель!

Автор этой книги инженер и предприниматель Марко Шварц — известный разработчик проектов и приложений для Интернета вещей. Многие из его ранних проектов «ушли в народ» и часто используются в других разработках и публикациях без указания авторства. Марко разработал несколько библиотек, которые включены в состав Arduino IDE, а также несколько сайтов и собственный облачный сервис для Интернета вещей.

Книга хороша тем, что ее можно начинать читать с любой главы и с любым уровнем технической подготовки. Каждая глава содержит детальное описание законченного проекта. Опытные любители электроники также найдут в книге полезные для себя советы и интересные решения.

Обратите внимание на адаптацию проектов книги к российской действительности. Проекты, вошедшие в книгу, разрабатывались в 2013–2015 годах, и выбор компонентов опирается на ассортимент западных интернет-магазинов. Сегодня российскому читателю доступны более современные и функциональные компоненты по низкой цене.

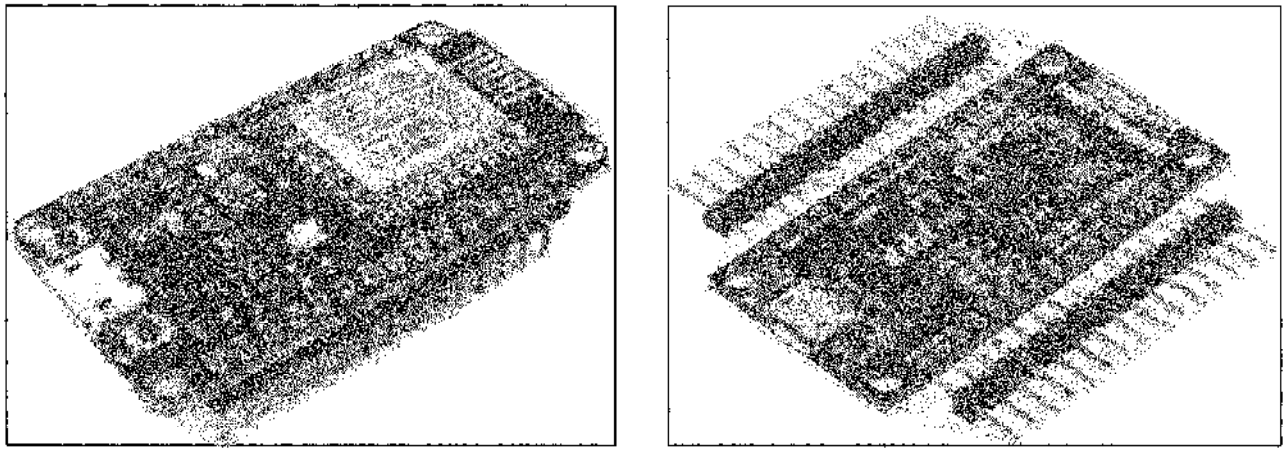
Издательство «БХВ-Петербург» планирует выпустить набор компонентов, необходимых для реализации проектов книги, и предоставляет для скачивания по ссылке **ftp://ftp.bhv.ru/9785977538671.zip** или со страницы книги на сайте **www.bhv.ru** электронный файловый архив с исходными кодами всех программ, снабженными переведенными комментариями (см. *приложение*).

Разумеется, вы можете и самостоятельно подобрать доступные аналоги компонентов, упомянутых в книге. Пожалуйста, внимательно ознакомьтесь с приведенными далее рекомендациями перед тем, как сделать заказы.

Рекомендации по замене компонентов

Все перечисленные в книге модули на основе микросхемы ESP8266 мы настоятельно рекомендуем заменить отладочной платой NodeMCU (рис. Пр.1, *а*) или RobotDyn WiFi (рис. Пр.1, *б*). Эти платы уже содержат все необходимое:

- стабилизатор питания +3,3 В;
- конвертер USB-UART для подключения к персональному компьютеру;
- схему автоматического сброса в режим загрузки прошивки;
- модуль ESP-12.



а *б*

Рис. Пр.1. Отладочные платы NodeMCU (*а*) и RobotDyn WiFi (*б*)

Все рабочие выводы ESP8266 на этих платах разведены на боковые разъемы. Сами платы рассчитаны на установку в стандартную беспаечную макетную плату. При использовании этих плат вам не понадобятся дополнительный источник питания +3,3 В и конвертер USB FTDI, упомянутые в книге, а благодаря схеме автоматического сброса загрузка прошивки будет происходить автоматически.

В меню среды разработки Arduino IDE — чтобы правильно работала схема автоматического сброса — следует выбирать плату NodeMCU 1.0.

Маркировка выводов отладочных плат исторически не совпадает с нумерацией портов GPIO[1] микросхемы ESP8266. Это досадная проблема, которая постоянно вызывает путаницу не только у начинающих радиолюбителей. Даже автор этой книги не избежал ошибок с нумерацией. Собирая устройство или разрабатывая программу, постоянно сверяйтесь со схемой, представленной на рис. Пр.2. Например, выводу GPIO5 соответствует вывод с маркировкой D1 на плате NodeMCU и многих других платах. Но! — если вы работаете с облачным сервисом aREST и его библиотекой, то нумерация выводов полностью соответствует разметке отладочной платы. Например, если в проекте для сервиса aREST идет речь о выводе 5, то это вывод D5 отладочной платы. При подготовке перевода этой книги мы постарались

[1] GPIO — General Ports of Input/Output, порты ввода/вывода общего назначения. — *Ред.*

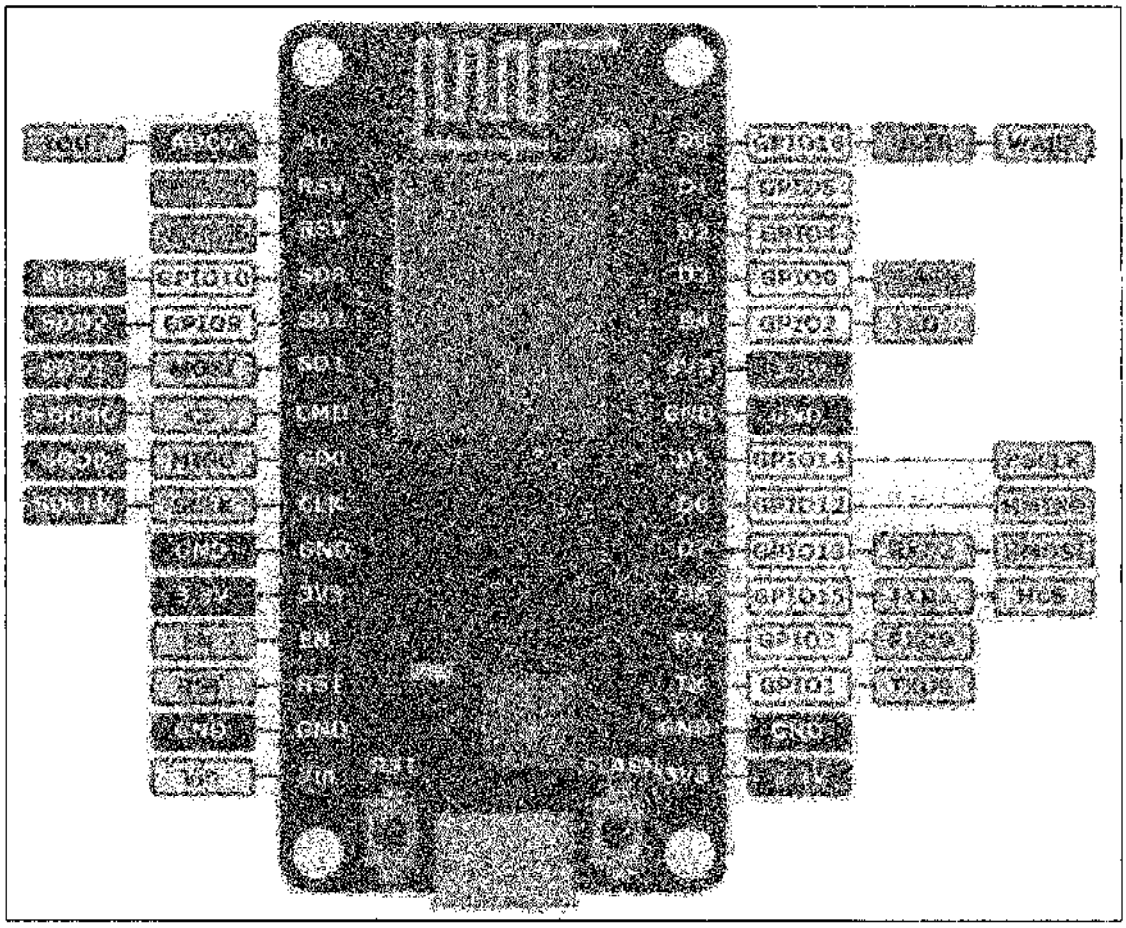


Рис. Пр.2. Схема обозначений выводов отладочных плат NodeMCU и RobotDyn

учесть оба варианта нумерации и снабдить программы соответствующими комментариями.

Вместо упомянутого в *главе 4* устройства PowerSwitch Tail Kit, которое представляет собой реле, размещенное в пластиковом корпусе, и стоит около $30, можно использовать обычное недорогое реле с оптической развязкой для платформы Arduino (рис. Пр.3). Это реле рассчитано на рабочее напряжение 5 В, поэтому его вывод

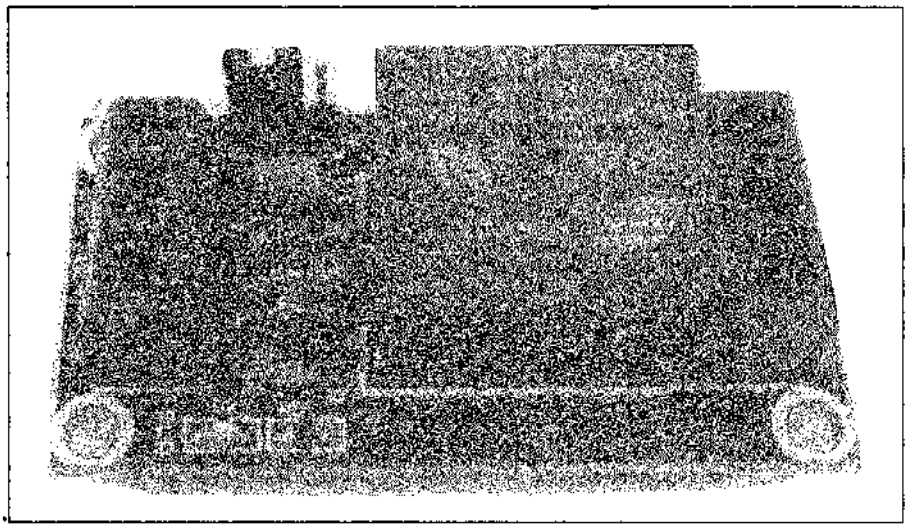


Рис. Пр.3. Реле с оптической развязкой

питания D+ необходимо соединить с выводом Vin (+5 В) платы NodeMCU (от напряжения 3,3 В реле может срабатывать неустойчиво), соответственно, с выводом GND платы должен быть соединен вывод D– реле. На вход IN реле подаем управляющий сигнал с того выхода платы, который указан в описании соответствующего проекта и в тексте программы. Нагрузку следует подключать с противоположной стороны реле — к выводам COM и NO.

Внимание!

При коммутации напряжения 220 вольт в цепи нагрузки обязательно используйте качественную изоляцию модуля реле и всех соединительных проводов! Например, можно затянуть модуль в толстую термоусадочную пленку или поместить в самодельный пластиковый корпус.

Вместо упомянутого в *главе 9* OLED-дисплея (см. рис. 9.4) можно использовать более распространенный и недорогой дисплей, изображенный на рис. Пр.4. Программы из сопровождающего книгу электронного архива (см. *приложение*) протестированы на совместимость с этим дисплеем. Какие-либо изменения в конструкции дисплея не требуются.

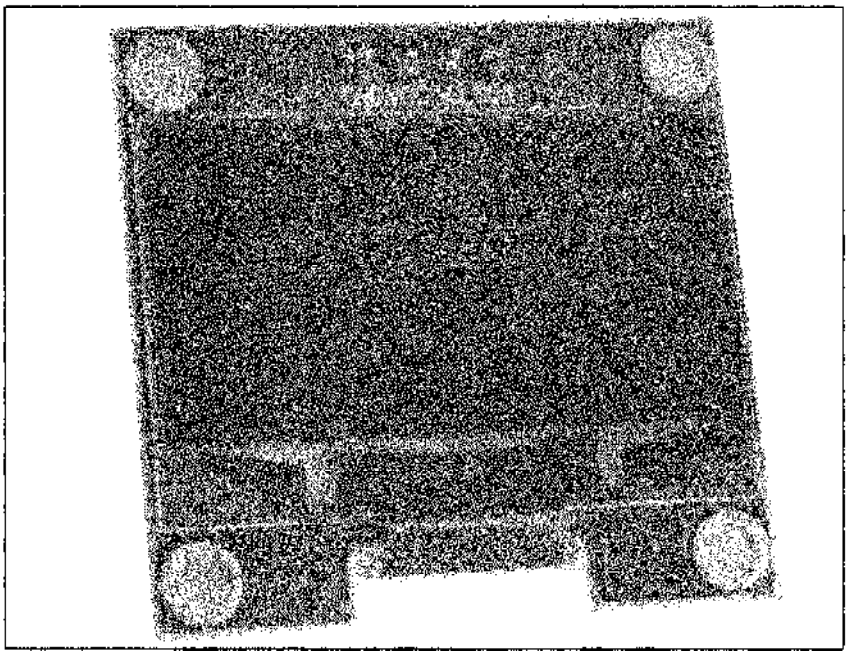


Рис. Пр.4. OLED-дисплей для проектов *главы 9*

Датчик температуры и влажности почвы SHT10 (см. рис. 10.1), к сожалению, не имеет равноценных по качеству аналогов. Особенность этого датчика заключается в полупроницаемой оболочке, изготовленной из спеченного металлического порошка. Такая оболочка пропускает отдельные молекулы воды, но не позволяет датчику целиком намокнуть. Для домашних экспериментов можно, конечно, приобрести недорогой датчик китайского производства в пластиковом корпусе (рис. Пр.5), но такие датчики во влажной почве часто запотевают изнутри и перестают корректно работать.

Для мобильного робота, о котором рассказано в *главе 12*, я рекомендую приобрести плату управления электромоторами (рис. Пр.6), которая полностью совместима

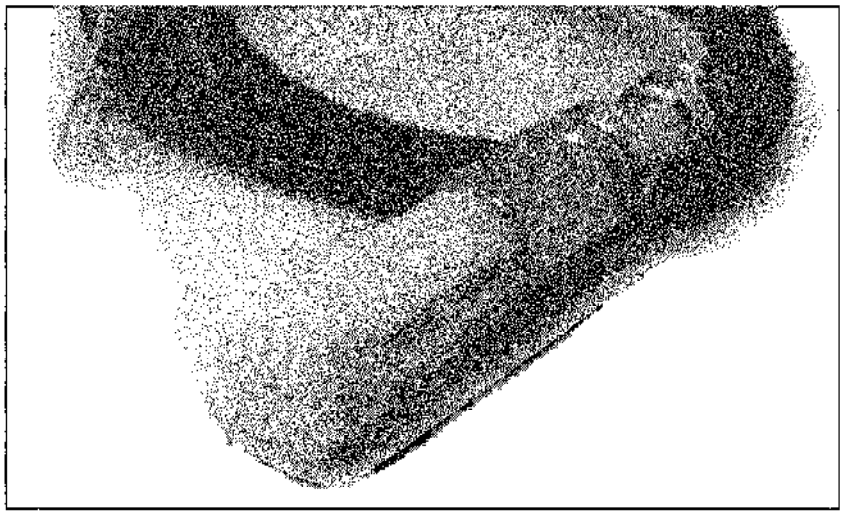


Рис. Пр.5. Аналог датчика SHT10 в пластиковом корпусе

с отладочными платами NodeMCU или RobotDyn. Использование готовой платы значительно облегчит сборку проекта. Эту плату можно установить на любую двухмоторную колесную или гусеничную платформу, аналогичную упомянутым в описании проекта.

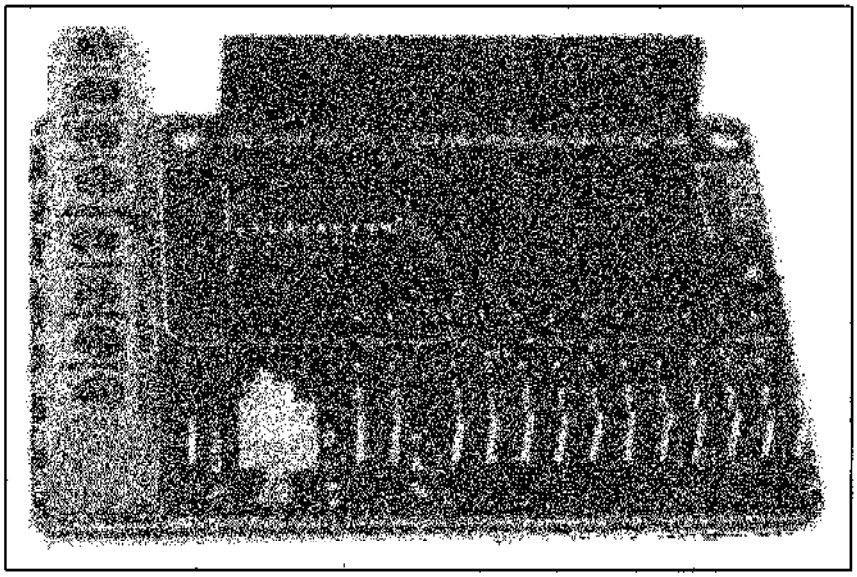


Рис. Пр.6. Плата управления электромоторами для NodeMCU

Прочие мелкие компоненты: датчик DHT11, резисторы, светодиоды, транзистор, датчик движения — не требуют подбора аналогов и широко представлены в продаже.

При подготовке перевода этой книги мы тщательно протестировали все программы, исправили обнаруженные ошибки и недочеты и адаптировали код программ для работы с аналогами упомянутых в книге компонентов. Мы приложили максимум усилий для того, чтобы избежать ошибок и обеспечить совместимость проектов с зарубежными онлайн-сервисами. Но мы не можем гарантировать, что к моменту покупки книги эти сервисы не изменят интерфейс или протоколы взаимодействия.

1

Первые шаги с ESP8266

Эта глава начинается с подготовки микросхемы ESP8266 к работе. Мы разберемся, как выбрать для вашего проекта правильный модуль на ее основе и установить программное обеспечение, необходимое для его использования. Кроме того, мы рассмотрим подключение ESP8266 к компьютеру и программирование ее через кабель USB.

Затем мы узнаем, как подготовить и загрузить в ESP8266 код прошивки. Для этого мы воспользуемся интегрированной средой Arduino IDE. Такой подход — благодаря хорошо знакомым многим интерфейсу этой среды и языку программирования — значительно упрощает использование ESP8266. При этом мы сможем задействовать большинство ранее установленных библиотек Arduino. Итак, начинаем!

Как выбрать модуль ESP8266?

Прежде всего, нужно выбрать правильный модуль с микросхемой ESP8266. На рынке представлено множество модулей — и так легко допустить ошибку, имея возможность выбора! Вы наверняка уже слышали о популярном модуле беспроводного приемопередатчика ESP-01 для последовательного канала (рис. 1.1).

Этот модуль — один из наиболее популярных, он действительно небольшой и стоит сейчас порядка 500 рублей[1]. Однако количество доступных выводов GPIO (General Ports of Input/Output, порты ввода/вывода общего назначения) у этого модуля очень невелико, и его трудно подключить к обычной макетной плате.

Если вы все же выбрали для себя модуль ESP-01, имейте в виду, что он не будет работать в некоторых проектах из этой книги. Например, вы не сможете реализо-

[1] Цены в рублях приведены на момент подготовки русского издания книги: июнь-июль 2017 года. — *Ред.*

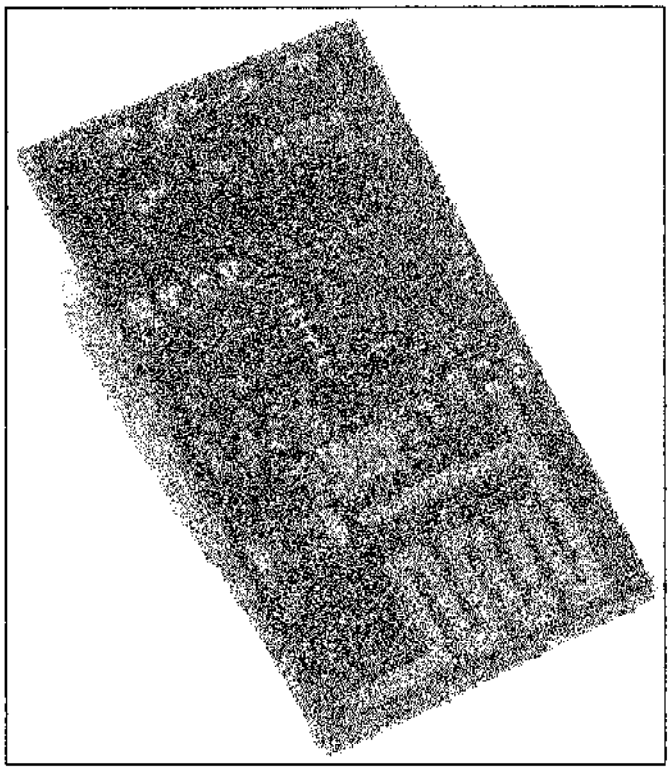


Рис. 1.1. Модуль беспроводного приемопередатчика ESP-01

вать проекты, использующие аналоговые датчики, потому на плате этого модуля вывод аналогового входа отсутствует.

Более подробную техническую информацию о микросхеме ESP8266 вы можете найти по адресу: **https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf**.

Дополнительная техническая информация о модуле ESP-01 доступна по адресу: **http://codeclub.cornwall.ac.uk/edenofthings/files/ESP-01FactSheet.pdf**.

На рынке встречаются и другие модули, которые предоставляют доступ ко всем выводам ESP8266. Например, мне очень нравится модуль ESP8266 Olimex, который стоит сейчас порядка 700 рублей (рис. 1.2).

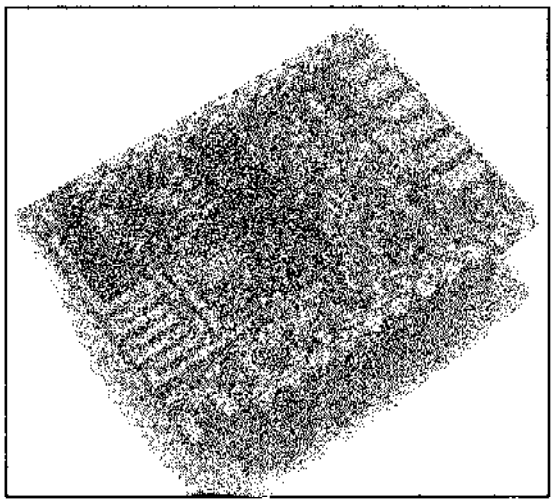


Рис. 1.2. Модуль Olimex

Этот модуль можно легко установить в беспаечную макетную плату, и столь же легко с его помощью получить доступ ко всем портам ESP8266. Такой модуль я использовал для разработки большинства проектов этой книги.

Дополнительную информацию об этом модуле можно получить по адресу: **www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/open-source-hardware**.

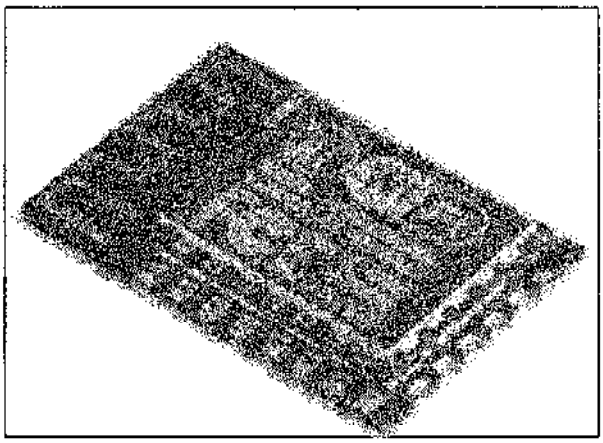


Рис. 1.3. Плата на основе модуля ESP-12

Еще один вариант — использовать плату на основе модуля ESP-12 (рис. 1.3), установленного на плате расширения. Эта версия также предоставляет вам доступ ко всем выводам ESP8266. Найти плату на основе модуля ESP-12 довольно несложно. Для примера, так выглядит плата, которую я купил в интернет-магазине Tindie (рис. 1.4)[2].

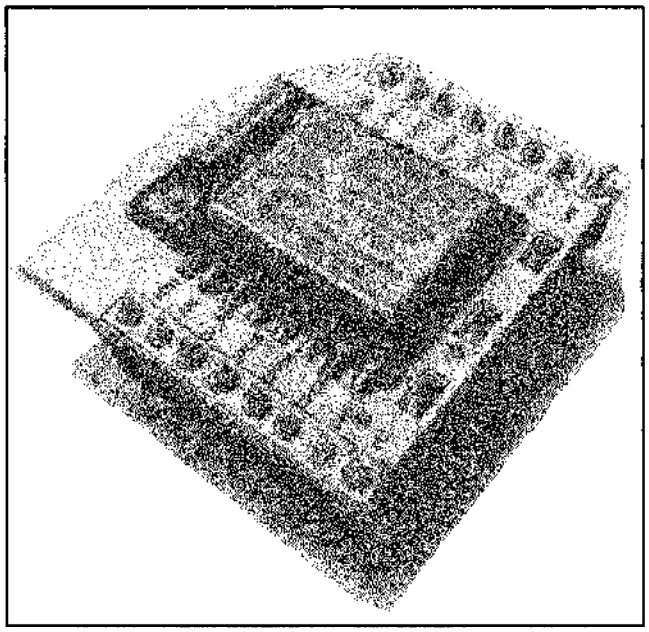


Рис. 1.4. Отладочная плата на основе микросхемы ESP8266

[2] Автор ошибается. На рис. 1.4 изображен модуль ESP-07. — *Прим. пер.*

Дополнительную информацию о модуле ESP-12 можно получить по адресу: **http://www.seeedstudio.com/wiki/images/7/7d/ESP-12E_brief_spec.pdf**.

Вы также можете приобрести плату Adafruit ESP8266, в которую модуль ESP-12 уже интегрирован: **http://www.adafruit.com/product/2471**.

Требования к оборудованию

Давайте теперь разберемся в том, что нам нужно, чтобы микросхема ESP8266 заработала. Обычно, но неправильно, предполагается, что вполне достаточно обзавестись этим маленьким чипом, и более ничего, чтобы заставить его работать, не потребуется, но далее мы увидим, что это не так.

Прежде всего, нам понадобится какой-то способ программирования ESP8266. Для этого вы можете подключить ее к плате Arduino, но для меня особое преимущество микросхемы ESP8266 заключается в том, что она может функционировать полностью автономно, опираясь на встроенный процессор. Поэтому для загрузки программ в микросхему ESP8266 я использую адаптер USB FTDI.

Имейте в виду, что адаптер USB должен быть совместим с логическими уровнями 3,3 В микросхемы ESP8266.

С учетом этого примечания я задействовал модуль, в котором можно переключать напряжение логических уровней 3,3 и 5 В (рис. 1.5).

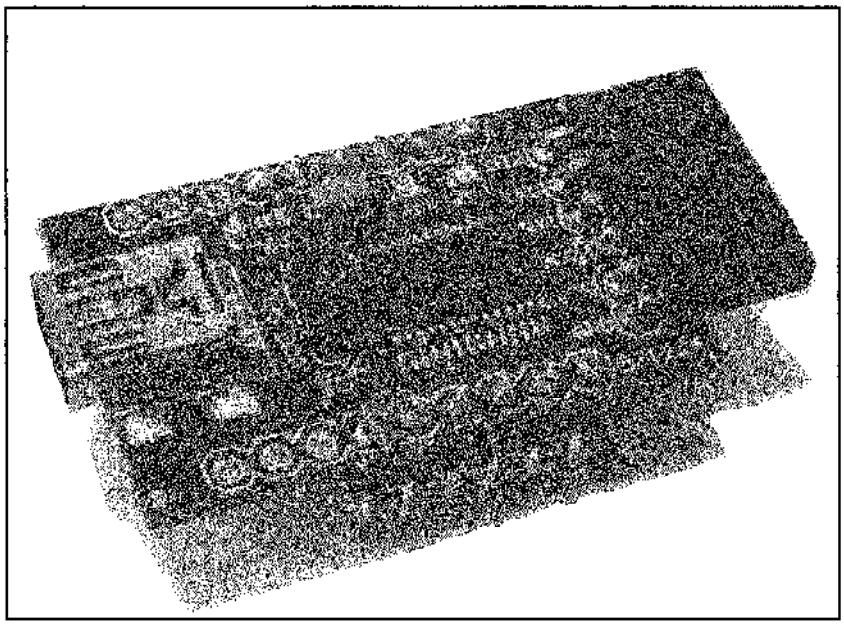


Рис. 1.5. Модуль адаптера USB FTDI

Нам также понадобится подходящий источник питания. Об этом часто забывают, что влечет за собой множество проблем. Если вы попробуете, например, запитать микросхему ESP8266 от источника питания напряжением +3,3 В, встроенного в плату адаптера USB FTDI или плату Arduino, то устройство просто не будет нор-

мально работать[3]. Поэтому для надежной работы большинства модулей ESP8266 требуется источник питания, который обеспечивает ток не менее 300 мА. Некоторые платы имеют разъем microUSB и встроенный источник питания, но это не относится к платам, о которых идет речь в этой главе. Так что я воспользовался источником питания для макетных плат, который обеспечивает ток 500 мА по линии +3,3 В (рис. 1.6).

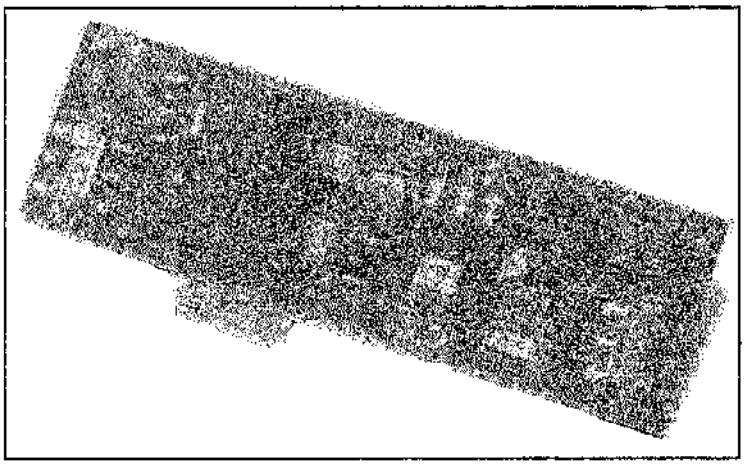


Рис. 1.6. Источник питания для макетной платы

Далее приведен список всех компонентов, которые понадобятся вам здесь для использования ESP8266:

- модуль ESP8266 Olimex: **www.olimex.com/Products/IoT/MODWIFI-ESP8266-DEV/open-source-hardware**;
- источник питания для макетной платы: **www.sparkfun.com/products/13032**;
- адаптер USB FTDI: **www.sparkfun.com/products/9873**;
- макетная плата: **www.sparkfun.com/products/12002**;
- соединительные провода: **www.sparkfun.com/products/12795**.

Аппаратная конфигурация

Теперь пора уделить внимание схеме соединений компонентов для первого включения платы ESP8266 — мы должны соединить их так, как показано на схеме (рис. 1.7).

В зависимости от того, какую плату вы используете, выводы могут иметь различные обозначения. Поэтому я приготовил иллюстрации для каждого из своих модулей.

- Назначение выводов модуля ESP-01 (рис. 1.8).
- Так обозначаются выводы отладочной платы расширения (рис. 1.9).
- И наконец, так выглядит назначение выводов модуля Olimex (рис. 1.10).

[3] Причина в том, что встроенный источник микросхемы адаптера USB предназначен только для формирования напряжений логических уровней и не поддерживает нагрузку более 100 мА. — *Прим. пер.*

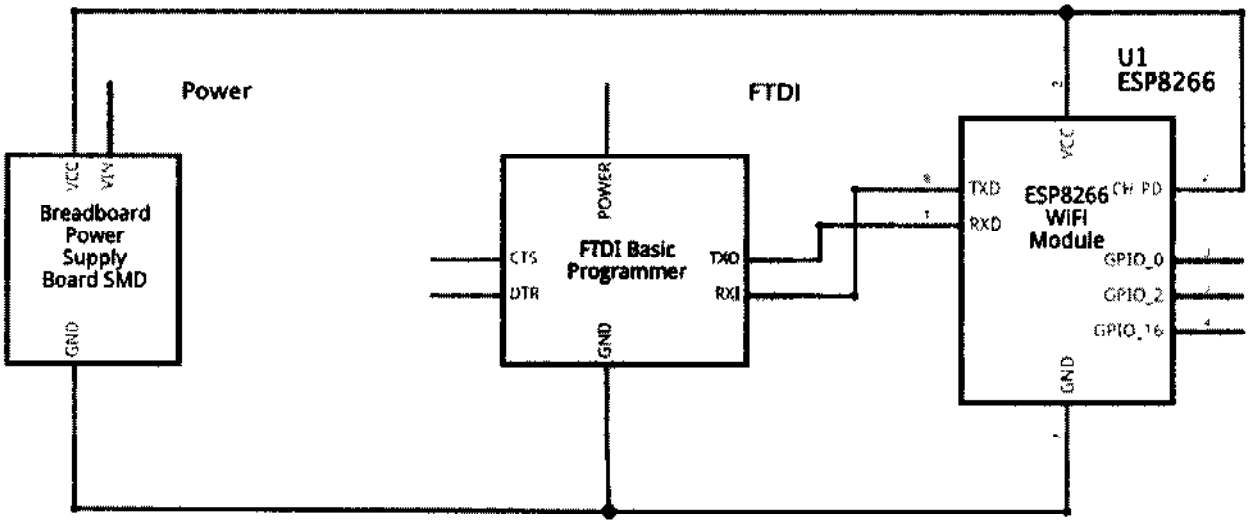


Рис. 1.7. Схема соединения компонентов

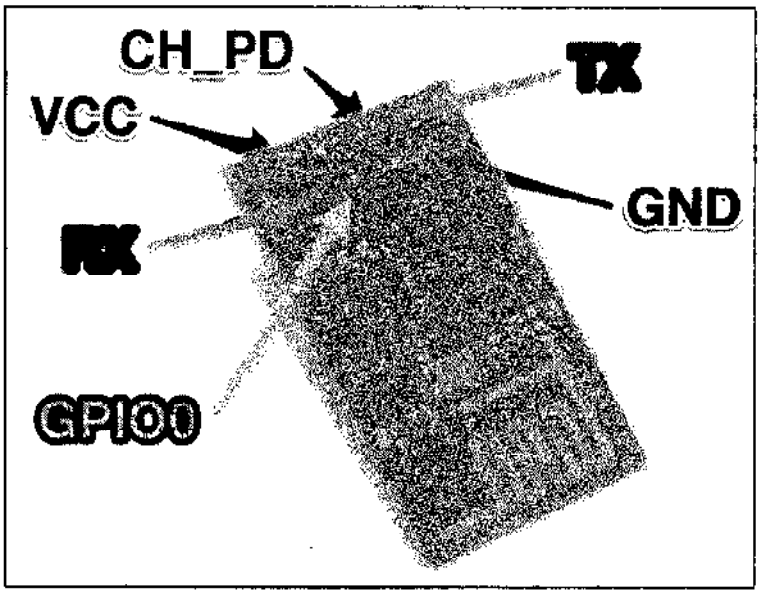


Рис. 1.8. Назначение выводов модуля ESP-01

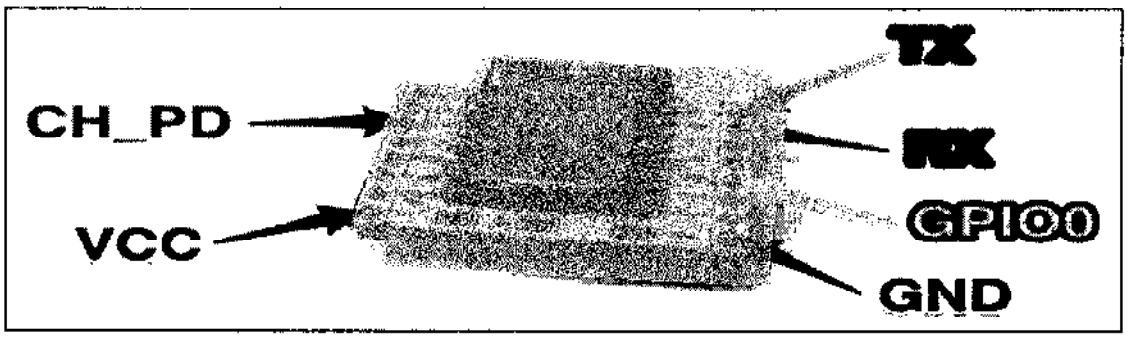


Рис. 1.9. Обозначение выводов отладочной платы расширения

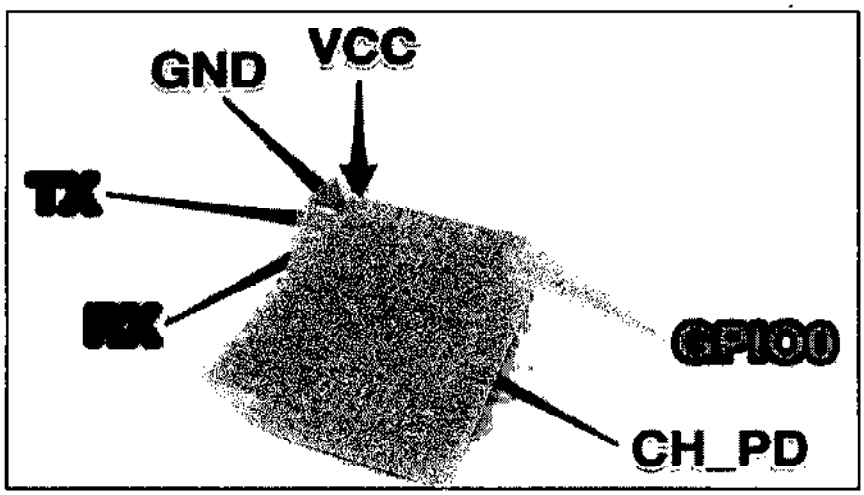


Рис. 1.10. Назначение выводов модуля Olimex

Плата Olimex в сборе с остальными компонентами выглядит так, как показано на рис. 1.11.

Убедитесь, что все соединения выполнены правильно, потому что иначе вы не сможете двигаться дальше.

Тщательно удостоверьтесь, что переключатели напряжения на источнике питания и модуле USB FTDI установлены в положение 3,3 В, иначе вы испортите микросхему!

Подключите отдельный проводник к выводу GPIO0 микросхемы ESP8266, но пока не соединяйте его больше ни с чем, — он понадобится позже для перевода микросхемы в режим программирования[4].

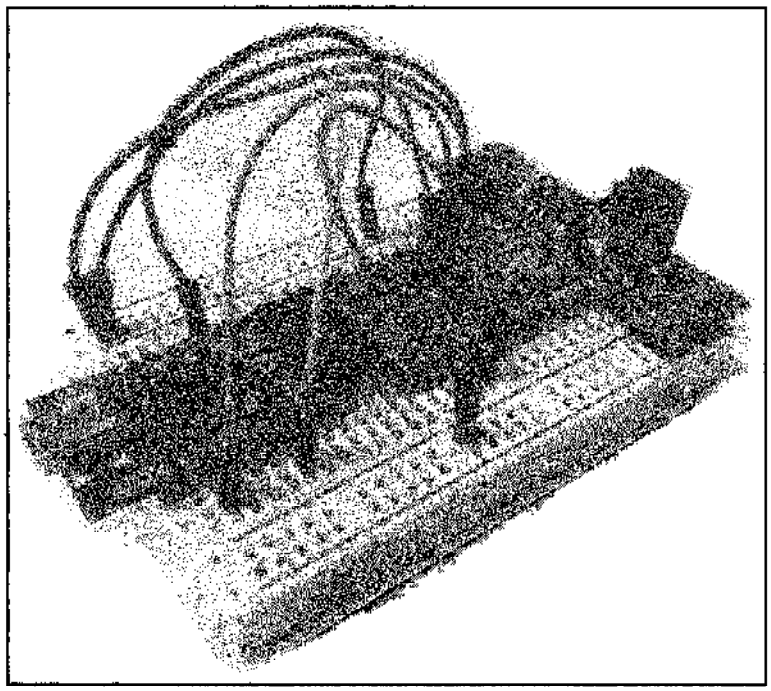


Рис. 1.11. Плата Olimex в сборе с другими компонентами на макетной плате

[4] При использовании платы NodeMCU этот проводник не нужен. — *Прим. пер.*

Установка Arduino IDE для работы с ESP8266

Итак, мы закончили подготовку оборудования на ESP8266 и готовы программировать его при помощи Arduino IDE.

Самый простой способ работы с ESP8266 состоит в отправке текстовых команд через последовательный порт, потому что микросхема изначально задумана как элементарный приемопередатчик данных через Wi-Fi. Однако этот способ неудобен, и я не советую его применять.

А что я на самом деле рекомендую — так это использовать среду Arduino IDE, которую вам придется установить на своем компьютере. Она сделает работу с ESP8266 очень комфортной, потому что среда Arduino IDE хорошо знакома всем, кто когда-либо имел дело с проектами на основе Arduino. Этот метод будет задействован на протяжении всей книги.

И сейчас наступило время установить самую свежую версию Arduino IDE[5]. Вы можете скачать ее по адресу: **http://www.arduino.cc/en/main/software**.

Теперь нам нужно настроить Arduino IDE для работы с ESP8266:

1. Запустите Arduino IDE и откройте окно **Файл | Настройки**.
2. Введите в поле **Дополнительные ссылки для Менеджера плат** адрес:

   `http://arduino.esp8266.com/stable/package_esp8266com_index.json`[6]

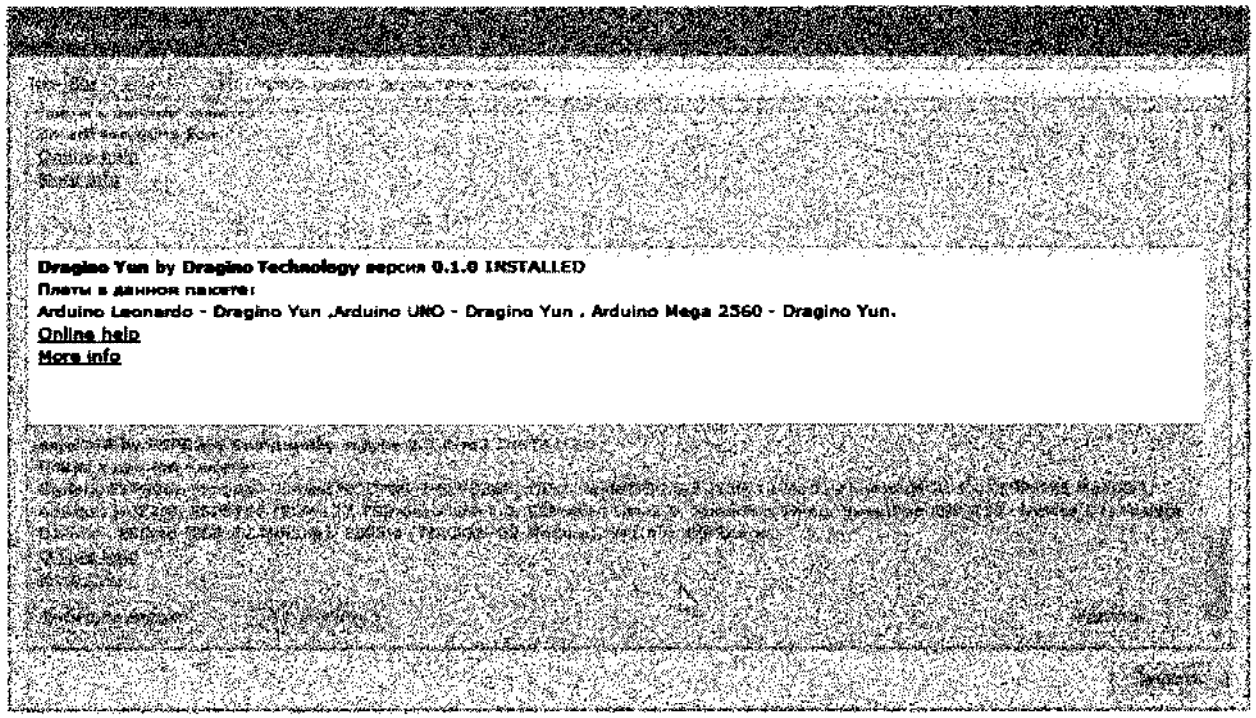


Рис. 1.12. Установка библиотеки **esp8266** в Менеджере плат Arduino IDE

[5] На момент работы над переводом это была версия 1.8.2, на совместимость с которой проверены все примеры программ из книги. — *Прим. пер.*

[6] Или `http://arduino.esp8266.com/staging/package_esp8266com_index.json` — если хотите иметь доступ к версиям, находящимся в разработке. — *Прим. пер.*

3. Откройте окно для установки расширений **Инструменты | Плата | Менеджер плат** и установите платформу **esp8266**, как показано на рис. 1.12.

Подключение модуля к сети Wi-Fi

Сейчас мы проверим правильность работы Arduino IDE и ESP8266, подключив ваш модуль к домашней сети Wi-Fi. Для этого выполним пошагово такие действия:

1. Напишем программу[7] и загрузим ее в память модуля. Программа очень простая — мы всего лишь хотим установить соединение с домашней сетью Wi-Fi и вывести в окно терминала IP-адрес, который получила наша плата. Вот исходный код программы (листинг 1.1).

Листинг 1.1 (Программа для проверки соединения с сетью Wi-Fi)

```
// Импортируем библиотеку поддержки ESP8266
#include <ESP8266WiFi.h>

// Параметры вашей сети Wi-Fi
const char* ssid = "your_wifi_name";
const char* password = "your_wifi_password";

void setup(void)
{
  // Инициализация последовательного порта
  Serial.begin(115200);
  // Инициализация соединения Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  // Вывод IP-адреса платы в терминал
  Serial.println(WiFi.localIP());
}
void loop() {
}
```

Вы можете открыть готовый файл программы из папки ch1_1 сопровождающего книгу электронного архива (см. *приложение*) или ввести текст программы непосредственно в окне редактора Arduino IDE. Разумеется, не забудьте подставить

[7] Программы для Arduino часто называют *скетчами* — иногда это название будет встречаться в тексте книги. — *Ред.*

имя и пароль *вашей* точки доступа Wi-Fi в исходный код программы. Сохраните программу, указав любое имя файла по своему усмотрению.

2. Перейдите в меню **Инструменты | Плата** и выберите **Generic ESP8266 Module**[8]. В пункте меню **Инструменты | Порт** укажите правильный номер виртуального последовательного порта, к которому подключен ваш адаптер USB.
3. Теперь нужно перевести плату в режим загрузки прошивки[9]. Для этого соедините вывод GPIO0 с общим проводом при помощи того самого проводника, который подключили ранее. Затем перезагрузите плату, отключив и заново подключив питание.
4. Откройте окно терминала **Инструменты | Монитор порта** и установите скорость 115200. Нажмите в меню Arduino IDE кнопку загрузки прошивки в плату. Отключите провод между выводами GPIO0 и GND. Повторите перезагрузку платы, отключив и снова подключив питание. Когда соединение установлено, и плата получила IP-адрес, вы увидите сообщение наподобие такого:

```
WiFi connected
192.168.1.103
```

Это сообщение означает, что ваша плата подключена к сети Wi-Fi.

Теперь вы готовы к созданию первого проекта для ESP8266.

Заключение

В этой главе мы рассмотрели основы работы с ESP8266. Познакомились с различными платами, которые подходят для ваших проектов. Научились подключать модули ESP8266. И наконец, узнали, как установить среду Arduino IDE и настроить ее для работы с ESP8266. А завершили главу загрузкой очень простой программы в память ESP8266.

В следующей главе мы применим усвоенные нами знания для создания нескольких проектов, основанных на микросхеме ESP8266, обеспечивающей работу в сети Wi-Fi.

---

[8] Для платы NodeMCU выберите опцию **NodeMCU 1.0**. — *Прим. пер.*

[9] Плата NodeMCU автоматически переходит в режим программирования, можете пропустить этот шаг. — *Прим. пер.*

2

Первые проекты на ESP8266

Теперь, когда ваша плата ESP8266 готова к использованию, и вы умеете подключать ее к сети Wi-Fi, можно создать несколько базовых проектов. Они помогут вам понять основы функционирования ESP8266.

Мы рассмотрим в этой главе три проекта: управление светодиодом, чтение данных с вывода GPIO и скачивание содержимого веб-страницы, а также разберемся, как считывать данные с цифрового датчика.

Управление светодиодом

Прежде всего научимся управлять одиночным светодиодом. Выводы портов GPIO микросхемы ESP8266 могут быть настроены для выполнения различных функций: цифровой вход, цифровой выход, *широтно-импульсная модуляция* (ШИМ), а также шины SPI и I^2C[1]. И первый проект научит вас использовать вывод GPIO в качестве выхода.

1. Добавим в проект светодиод. Для этого вам потребуются два дополнительных компонента:
   - светодиод с диаметром линзы 5 мм: **https://www.sparkfun.com/products/9590**;
   - Резистор 330 Ом для ограничения тока через светодиод: **https://www.sparkfun.com/products/8377**.
2. Установите на макетную плату резистор.
3. Затем установите на плату светодиод, соединив более длинный вывод светодиода (анод) с одним из выводов резистора.

---

[1] Микросхема также имеет один аналоговый вход. — *Прим. пер.*

4. Соедините второй вывод резистора с выводом GPIO5 платы ESP8266[2]. Короткий вывод светодиода соедините с общим проводом («земля»).

   Готовая схема может выглядеть примерно так, как показано на рис. 2.1.

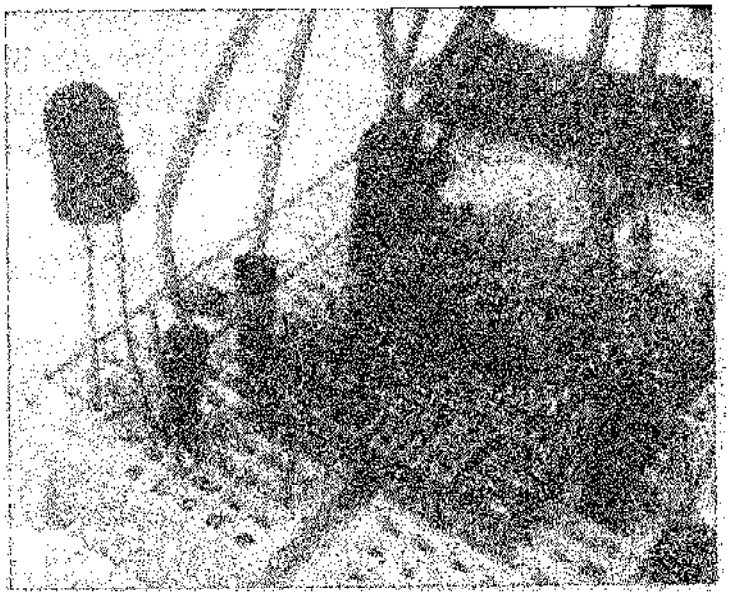


Рис. 2.1. Вариант монтажа схемы для проекта управления светодиодом

5. Теперь мы готовы зажечь светодиод, запрограммировав для этого микросхему ESP8266, — аналогично тому, как мы это сделали в *главе 1* для подключения к сети Wi-Fi. Полный текст программы управления светодиодом приведен в листинге 2.1.

Листинг 2.1. Простейшая программа управления светодиодом

```
// Импорт необходимой библиотеки
#include <ESP8266WiFi.h>
void setup() {
// Настраиваем GPIO5 (D1) как выход
pinMode(5, OUTPUT);
// Устанавливаем на выходе высокий уровень
digitalWrite(5, HIGH);
}
void loop() {
}
```

Эта простейшая программа переводит вывод GPIO5 в режим выхода и устанавливает на нем *высокий* логический уровень. Термин «высокий уровень» означает, что на выходе присутствует напряжение, близкое к напряжению питания +3,3 В. Соответственно *низкий* логический уровень означает выходное напряжение, близкое к нулю.

[2] На большинстве отладочных плат вывод GPIO5 имеет обозначение D1. — *Прим. пер.*

6. Вы можете скопировать код программы и ввести его в окне редактора Arduino IDE или открыть готовый файл программы из папки ch2_1 сопровождающего книгу электронного архива (см. *приложение*).
7. Загрузите прошивку в плату аналогично тому, как это делалось в *главе 1*. Сразу после перезагрузки платы светодиод начнет светиться. Вы можете погасить его при помощи строки кода `digitalWrite(5, LOW)`. При желании код можно изменить так, чтобы микросхема ESP8266 зажигала и гасила светодиод каждую секунду.

Чтение данных с вывода GPIO

1. Во втором проекте этой главы мы научимся считывать логический уровень на выводе GPIO. Воспользуемся для этого тем же выводом, что и в предыдущем проекте. Резистор и светодиод с платы можно убрать.
2. Теперь просто соедините вывод GPIO5 с линией питания +3,3 В при помощи отдельного проводника. Код программы второго проекта очень прост (листинг 2.2).

Листинг 2.2. Программа для чтения логического уровня на пине

```
// Импорт необходимой библиотеки
#include <ESP8266WiFi.h>
void setup(void)
{
// Старт последовательного порта
// для передачи текста в монитор
Serial.begin(115200);
// Настраиваем GPIO5 (D1) как вход
pinMode(5, INPUT);}
void loop() {
// Читаем уровень на GPIO5 и выводим в монитор
Serial.print("State of GPIO 5: ");
Serial.println(digitalRead(5));
// Пауза 1 секунда
delay(1000);
}
```

В этой программе мы устанавливаем контакт GPIO5 в качестве входа, считываем значение с этого контакта и выводим его в окно монитора порта каждую секунду.

Скопируйте и вставьте код из листинга 2.2 в окно IDE Arduino или откройте готовый файл программы из папки ch2_2 сопровождающего книгу электронного архива (см. *приложение*), а затем загрузите его на плату, используя инструкции из предыдущей главы.

В результате выполнения программы вы должны увидеть в окне монитора порта следующую ежесекундно выводимую строку:

```
State of GPIO 5: 1
```

Как мы и ожидали, возвращаемое значение равно 1 (т. е. цифровое состояние вывода — высокое) — потому что мы подключили вывод к положительному источнику питания. В качестве эксперимента вы можете подключить этот вывод к «земле», и тогда его состояние сменится на 0 (т. е. на низкое).

Скачивание содержимого веб-страницы

В следующем проекте мы наконец воспользуемся соединением Wi-Fi, чтобы скачать содержимое веб-страницы, и обратимся для этого к странице по адресу: **www.example.com**. Это реально существующий адрес, широко используемый для тестирования приложений. Полный код программы скачивания содержимого веб-страницы приведен в листинге 2.3.

Листинг 2.3. Программа скачивания содержимого веб-страницы

```
// Импорт необходимой библиотеки
#include <ESP8266WiFi.h>
// Параметры вашей точки доступа
const char* ssid = "your_wifi_network";
const char* password = "your_wifi_password";
// Адрес сайта
const char* host = "www.example.com";
void setup() {
// Старт последовательного порта
// для передачи текста в монитор
Serial.begin(115200);
// Подключаемся к сети WiFi
Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}
```

```
int value = 0;
void loop() {
Serial.print("Connecting to ");
Serial.println(host);
// Используем класс WiFiClient для создания подключения TCP
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
// Если сайт недоступен
Serial.println("connection failed");
return;
}
// Отправляем HTTP-запрос на сервер
client.print(String("GET /") + " HTTP/1.1\r\n" +
"Host: " + host + "\r\n" + "Connection: close\r\n\r\n");
delay(10);
// Читаем все строки ответа сервера и выводим их в монитор
while(client.available()){
String line = client.readStringUntil('\r');
Serial.print(line);
}
Serial.println();
Serial.println("closing connection");
delay(5000);
}
```

Этот код тоже весьма прост: сначала мы открываем соединение с веб-сайтом **www.example.com**, а потом посылаем ему GET-запрос, чтобы получить содержимое страницы. Используя строку программы `while(client.available())`, мы принимаем все входящие данные с сайта, пока они не закончатся, и выводим их в последовательный порт для просмотра в мониторе.

Скопируйте код программы и введите его в окне редактора Arduino IDE или откройте готовый файл программы из папки ch2_3 сопровождающего книгу электронного архива (см. *приложение*). Затем загрузите программу в плату согласно инструкциям *разд. «Подключение модуля к сети Wi-Fi» главы 1*.

После загрузки программы плата соединится с сетью Wi-Fi и отправит к сайту запрос[3]. Запрос этот будет повторяться каждые пять секунд.

Текст, который должен быть выведен в окно монитора, показан на рис. 2.2, — это HTML-код страницы в исходном виде.

---

[3] Рекомендуется после загрузки прошивки «сбросить» ESP8266 кнопкой сброса или отключением питания — иначе она может не соединиться с сайтом. Это связано с особенностями работы стека TCP в микросхеме. — *Прим. пер.*

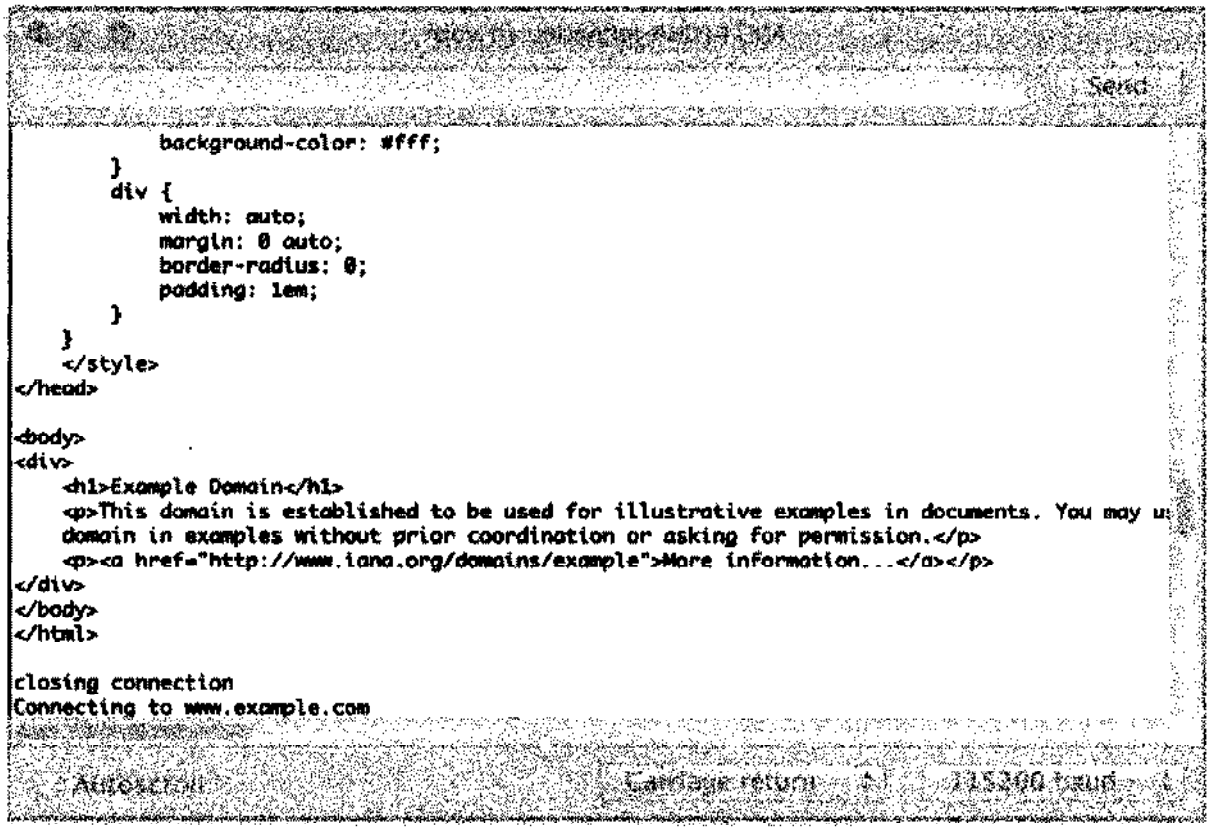


Рис. 2.2. Содержимое веб-страницы в окне монитора порта Arduino IDE

Чтение данных с цифрового датчика

В завершающем разделе этой главы мы подключим к ESP8266 цифровой датчик и будем считывать с него данные. Воспользуемся, например, для этого популярным датчиком DHT11, измеряющим температуру и влажность окружающего воздуха. Соответственно, в комплект компонентов этого проекта следует добавить такой датчик (**https://www.adafruit.com/products/386**).

Подключите датчик DHT11 к ESP8266:

1. Установите датчик на макетную плату. Затем соедините его первый вывод с линией питания +3,3 В, второй его вывод — с выводом GPIO5 (D1) микросхемы ESP8266, а четвертый — с общим проводом («земля»). Собранная схема будет выглядеть приблизительно так, как показано на рис. 2.3.

Обратите внимание, что здесь я применил другую плату ESP8266 — а именно Adafruit ESP8266. Я буду использовать эту плату еще в ряде проектов в других главах книги[4].

В этом проекте мы также используем *фреймворк* (набор готовых программных решений) под названием aREST. Это законченный пакет инструментов для уда-

[4] Для таких простых проектов версия платы не имеет особого значения. Плата NodeMCU остается наиболее удобным вариантом, в том числе и для этого проекта. — *Прим. пер.*

ленного управления платой ESP8266 (в том числе и через облако), и мы к нему будем обращаться в этой книге и в дальнейшем. Более подробную информацию об этом фреймворке вы можете найти на сайте: **http://arest.io**.

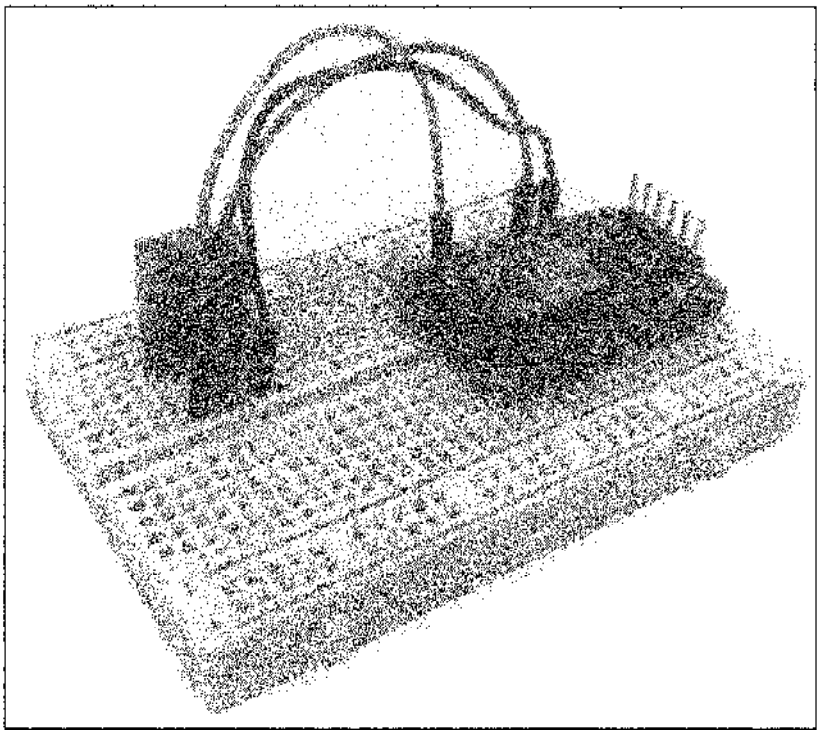


Рис. 2.3. Вариант монтажа схемы с датчиком DHT11

2. Подготовим код программы для записи прошивки в ESP8266. Программа эта слишком велика для того, чтобы вставить ее в книгу полностью, но я детально поясню здесь самые важные фрагменты.

Вы можете скачать исходный код программы в репозитории GitHub по адресу: **https://github.com/openhomeautomation/iot-esp8266-packt**. Готовый к использованию файл программы находится также в папке ch2_DHT11 сопровождающего книгу электронного архива (см. *приложение*).

3. Программа начинается с подключения необходимых библиотек:

```
#include "ESP8266WiFi.h"
#include <aREST.h>
#include "DHT.h"
```

4. Для установки недостающих библиотек перейдите в Менеджер библиотек Arduino IDE: **Скетч | Подключить библиотеку | Управлять библиотеками**. Найдите эти библиотеки в списке, введя их названия в поле поиска, и установите их. Далее указываем, к какому выводу подключен датчик и тип этого датчика:

```
#define DHTPIN 5
#define DHTTYPE DHT11
```

5. Объявляем объект `DHT`:

```
DHT dht(DHTPIN, DHTTYPE);
```

6. Как и раньше, вы должны указать данные домашней точки доступа Wi-Fi:

```
const char* ssid = "your_wifi_name";
const char* password = "your_wifi_pass";
```

7. Определяем две переменных, в которых будут храниться результаты измерений:

```
float temperature;
float humidity;
```

8. В стандартной функции `setup()` инициализируем датчик:

```
dht.begin();
```

9. Оставаясь в функции `setup()`, подключаем переменные к aREST API, чтобы иметь возможность удаленного доступа к ним:

```
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
```

10. И наконец, в стандартной функции `loop()` циклически считываем данные из датчика:

```
humidity = dht.readHumidity();
temperature = dht.readTemperature();
```

11. Теперь пора тестировать проект! Скопируйте код программы и введите его в окне редактора Arduino IDE или откройте готовый файл программы из папки ch2_DHT11 сопровождающего книгу электронного архива (см. *приложение*). Убедитесь, что вы установили библиотеки aREST и DHT при помощи Менеджера библиотек Arduino IDE.

12. Загрузите прошивку в плату ESP8266. Перезагрузите плату и откройте монитор порта — вы должны увидеть IP-адрес, присвоенный плате (рис. 2.4).

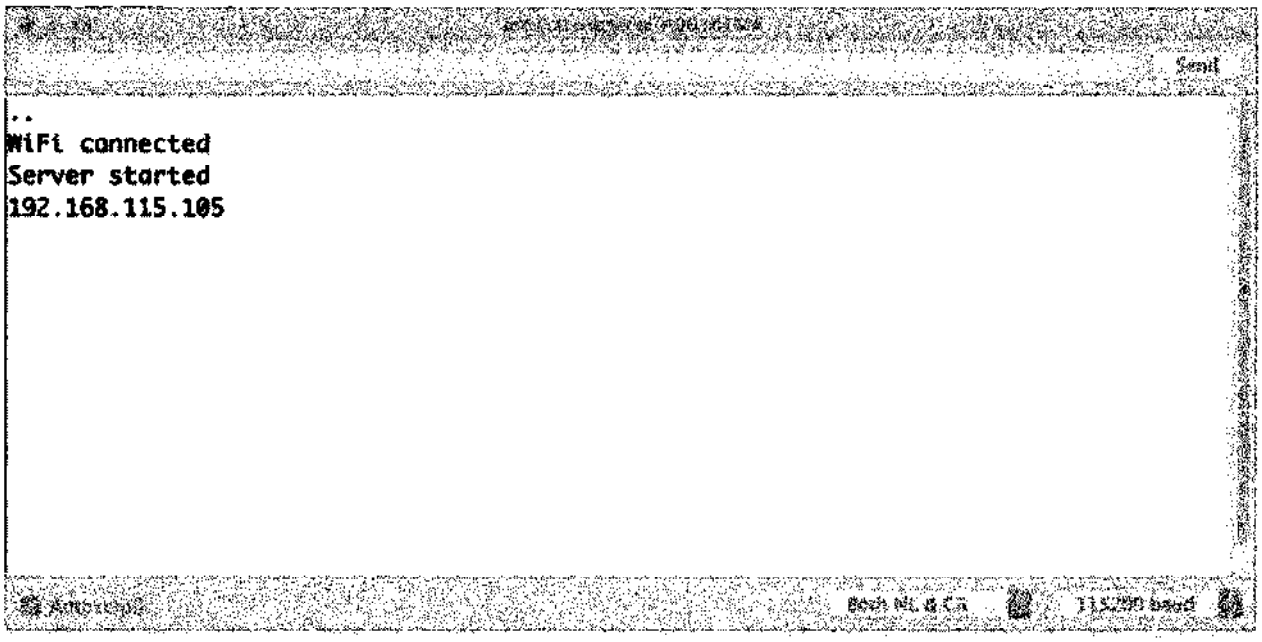


Рис. 2.4. Адрес платы ESP8266 в домашней сети

13. Теперь мы можем получить удаленный доступ к измерениям от нашего датчика, поскольку по IP-адресу, выведенному в монитор, доступен встроенный сервер aREST. Введите этот IP-адрес в адресную строку браузера на компьютере или планшете, подключенном к этой же локальной сети, например:

    ```
    192.168.115.105/temperature
    ```

    Почти сразу вы получите в браузере ответ от ESP8266, в котором содержится значение температуры:

    ```
    {
    "temperature": 25.00,
    "id": "1",
    "name": "esp8266",
    "connected": true
    }
    ```

Разумеется, вы можете точно так же получить значение влажности (`humidity`).

Напомню, что в этом проекте мы использовали API aREST, к которому будем обращаться в этой книге и в дальнейшем. Более подробную информацию об этом API вы можете найти на сайте: http://arest.io.

Поздравляю, вы только что успешно реализовали свои первые проекты на ESP8266! Не бойтесь экспериментировать со всем, что узнали из этой главы, и переходите затем к дальнейшему изучению микросхемы ESP8266.

Заключение

В этой главе мы воплотили в реальность наши первые проекты для микросхемы ESP8266, обеспечивающей работу в сети Wi-Fi. Сначала мы научились управлять состоянием одиночного выхода, зажигая и гася светодиод. Затем мы узнали, как читать логический уровень на цифровом входе микросхемы. Мы также скачали содержимое веб-страницы и вывели его в монитор последовательного порта. Наконец, мы смогли прочитать данные с цифрового датчика и отобразить их при помощи фреймворка aREST, который будем использовать в этой книге и в дальнейшем.

В следующей главе мы обратимся непосредственно к главной теме книги и построим на основе ESP8266 наш первый проект Интернета вещей.

3

Сохраняем данные в облако

В этой главе мы воспользуемся микросхемой ESP8266 для автоматического сохранения замеров температуры и влажности в облако и отображения сохраненных данных на онлайновой приборной панели.

На основе этих проектов вы сможете построить компактную и недорогую измерительную платформу, способную хранить данные в облаке. Разумеется, сказанное здесь относится ко всевозможным датчикам, в том числе и таким, как датчик движения. Начнем!

Оборудование и программное обеспечение

Для проектов этой главы потребуется следующее оборудование:

- плата с микросхемой ESP8266 — можно, например, использовать модуль Olimex или NodeMCU;
- датчик температуры — я взял датчик DHT11, который очень прост в использовании и позволяет измерять температуру и влажность окружающего воздуха;
- модуль USB FTDI для программирования ESP8266[1];
- беспаечная макетная плата и соединительные провода.

Вот конкретный перечень использованных в проектах компонентов с указанием места приобретения каждого из них:

- плата ESP8266 Olimex: **https://www.olimex.com/Products/IoT/MODWIFI-ESP8266-DEV/open-source-hardware**;
- стабилизатор питания 3,3 вольт: **https://www.sparkfun.com/products/13032**;

---

[1] При использовании платы NodeMCU этот модуль не понадобится. — *Прим. пер.*

- модуль FTDI USB: **https://www.sparkfun.com/products/9873**;
- датчик DHT11: **https://www.adafruit.com/products/386**;
- макетная плата: **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

Вам также потребуется новая версия Arduino IDE, которую можно скачать по адресу: **http://www.arduino.cc/en/Main/Software**.

Теперь нам нужно настроить Arduino IDE для работы с ESP8266:

1. Запустите Arduino IDE и откройте окно **Файл | Настройки**.
2. Введите в поле **Дополнительные ссылки для Менеджера плат** адрес: `http://arduino.esp8266.com/stable/package_esp8266com_index.json`[2].
3. Откройте окно для установки расширений **Инструменты | Плата | Менеджер плат** и установите платформу **esp8266**, как показано на рис. 1.12.
4. Вам также понадобится библиотека DHT, которую вы можете получить по адресу: **https://github.com/adafruit/DHT-sensor-library**.

Для установки библиотеки Arduino:

1. Скачайте архив библиотеки из репозитория GitHub.
2. Перейдите в меню **Скетч | Подключить библиотеку | Добавить ZIP.библиотеку**.
3. Выберите скачанный файл.

Подключение компонентов

Прежде всего, соединим между собой все компоненты. Схема подключений показана на рис. 3.1.

1. Сначала[3] вы должны соединить выводы VCC и GND источника питания макетной платы с выводами VCC и GND платы ESP8266. Также соедините вывод GND платы конвертера USB FTDI с выводом GND платы ESP8266.
2. Затем соедините вывод TX платы FTDI с выводом RX платы ESP8266, а вывод RX — с выводом TX.
3. Наконец, соедините вывод CH_PD (или CHIP_EN) платы ESP8266 с линией VCC.
4. Завершив эти действия, установите на макетную плату датчик DHT11.
5. Соедините вывод 1 датчика с линией питания (красная шина макетной платы), вывод 4 — с общим проводом (синяя шина макетной платы). Вывод 2 подклю-

---

[2] Или `http://arduino.esp8266.com/staging/package_esp8266com_index.json` — если хотите иметь доступ к версиям, находящимся в разработке. — *Прим. пер.*

[3] При использовании платы NodeMCU переходите сразу к подключению датчика DHT11 (пункты 4 и 5 этого списка). — *Прим. пер.*

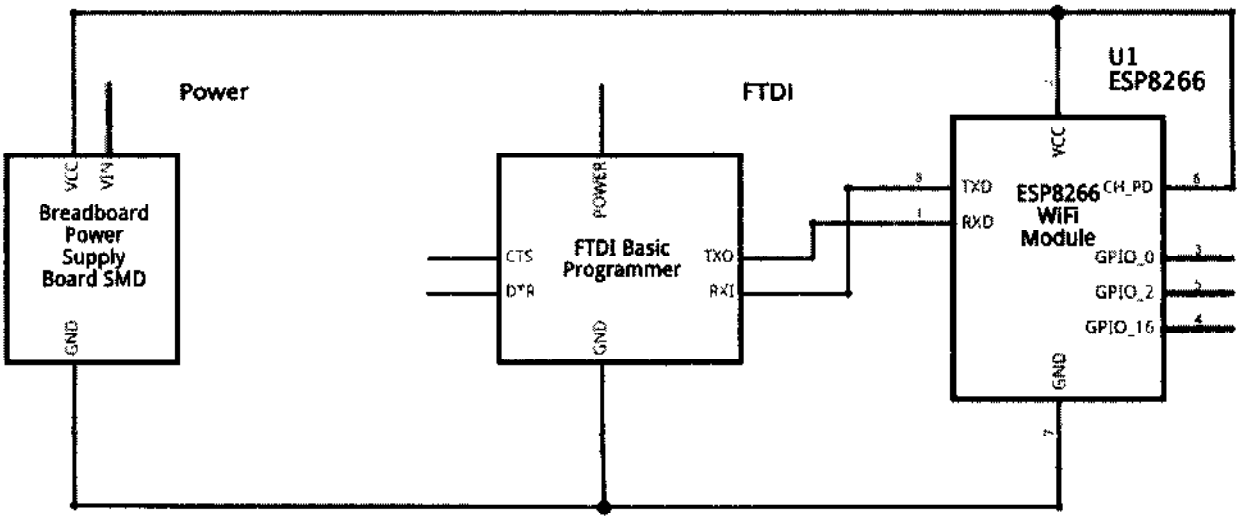


Рис. 3.1. Схема подключения компонентов

чите к выводу GPIO5 (D1) модуля ESP8266. Результат будет выглядеть примерно так, как показано на рис. 3.2.

Убедитесь, что все соединения выполнены правильно, иначе вы не сможете продолжить работу. Также удостоверьтесь, что переключатели USB FTDI и стабилизатора питания установлены в положение 3,3 В, — чтобы не вывести из строя микросхему ESP8266.

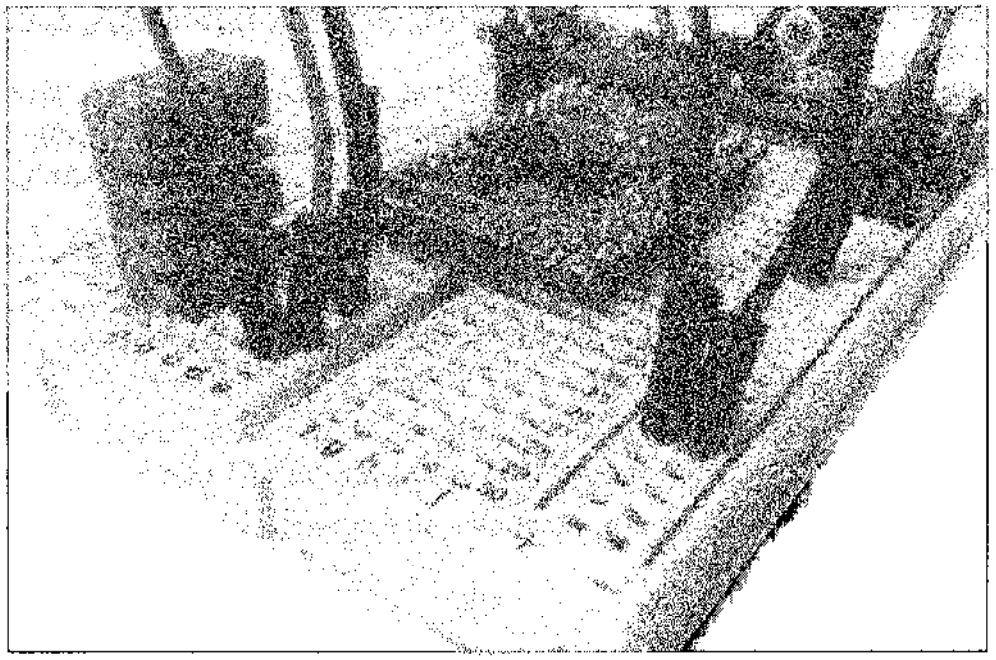


Рис. 3.2. Подключение датчика DHT11

Подключите отдельный проводник к выводу GPIO0 микросхемы ESP8266, но пока не соединяйте его больше ни с чем, — он понадобится позже для перевода микросхемы в режим программирования.

Проверка датчика

Что ж, мы готовы к использованию датчика. Еще раз напоминаю, что мы работаем в среде Arduino IDE, поэтому наш код и выглядит так, будто мы программируем для платы Arduino. В данном же случае мы просто выводим значение температуры на печать в окно монитора порта Arduino IDE. Если этого не происходит, установите библиотеку датчика Adafruit DHT при помощи Менеджера библиотек Arduino IDE.

Полный текст программы приведен в листинге 3.1.

Листинг 3.1. Программа для проверки датчика DHT11

```
// Подключаем библиотеку DHT
#include "DHT.h"
// Объявляем номер вывода для датчика
#define DHTPIN 5
// Объявляем датчик DHT11
#define DHTTYPE DHT11
// Создаем объект датчика
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  // Инициализируем последовательный порт
  Serial.begin(115200);
  // Инициализируем датчик
  dht.begin();
}
void loop() {
// Пауза 2 секунды перед очередным измерением
  delay(2000);
  // Считываем влажность
  float h = dht.readHumidity();
  // Считываем температуру в градусах Цельсия
  float t = dht.readTemperature();
  // Выводим данные на печать
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.println(" *C ");
}
```

Уточним некоторые детали этой программы. Как можно видеть, все действия, непосредственно относящиеся к измерению, выполняются внутри функции `loop()`, которая повторяет включенный в нее код каждые две секунды.

Далее мы считываем данные с датчика DHT11 и выводим значения температуры и влажности в последовательный порт.

Вы можете скачать исходный код программы в репозитории GitHub по адресу: **https://github.com/openhomeautomation/iot-esp8266**. Готовый к использованию файл программы находится также в папке ch3_1 сопровождающего книгу электронного архива (см. *приложение*).

Приступим теперь непосредственно к проверке работы датчика[4].

1. Скопируйте код программы и введите его в окне редактора Arduino IDE или откройте готовый файл программы из папки ch3_1 сопровождающего книгу электронного архива (см. *приложение*). Откройте окно для установки расширений **Инструменты | Плата** и выберите плату **Generic ESP8266 module** и последовательный порт, к которому она подключена.
2. Переведите плату ESP8266 в режим загрузки, для чего соедините вывод GPIO0 с общим проводом.
3. Отключите и вновь подключите питание платы.
4. Загрузите прошивку в плату и запустите монитор последовательного порта Arduino IDE, установите скорость порта 115200.
5. Сразу после этого в окне монитора вы должны увидеть значения температуры и влажности. Мой датчик во время проверки показал 24 градуса Цельсия. Это вполне реалистичное значение.

Загрузка данных в сервис *dweet.io*

Теперь разберемся, как сохранять в облако измеренные значения температуры и влажности. Для этого мы воспользуемся облачным сервисом **http://dweet.io/**, весьма удобным для онлайнового сохранения данных:

Исходный код программы слишком велик для размещения в книге. Вы можете скачать его в репозитории GitHub по адресу: **https://github.com/openhomeautomation/iot-esp8266**. Готовый к использованию файл программы находится также в папке ch3_Cloud сопровождающего книгу электронного архива (см. *приложение*).

Все измерения в программе вновь происходят внутри функции `loop()`, которая повторяет замеры каждые 10 секунд, используя функцию `delay()`.

Внутри цикла мы соединяемся с сервером **dweet.io** при помощи кода:

```
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
Serial.println("connection failed");
return;
}
```

[4] Шаги 2 и 3 не требуются, если вы используете плату NodeMCU или аналогичную. — *Прим. пер.*

Затем считываем данные с датчика:

```
int h = dht.readHumidity();
int t = dht.readTemperature();
```

Потом передаем эти данные на сервер **dweet.io**:

```
client.print(String("GET /dweet/for/myesp8266?temperature=") +
String(t) + "&humidity=" + String(h) + " HTTP/1.1\r\n" +
"Host: " + host + "\r\n" +
"Connection: close\r\n\r\n");
```

Вам следует заменить в коде `myesp8266` на имя вашего устройства. Оно должно быть сложным, как пароль, чтобы вы были уверены, что создаете на **dweet.io** уникальное устройство.

Кроме этого, мы выводим все полученные данные в последовательный порт:

```
while(client.available()){
String line = client.readStringUntil('\r');
Serial.print(line);
}
```

Не забудьте подставить в код имя и пароль вашей сети Wi-Fi. Теперь вы можете загрузить прошивку в плату ESP8266, согласно инструкциям *разд. «Подключение модуля к сети Wi-Fi» главы 1*, и открыть монитор порта Arduino IDE[5].

Вы должны увидеть, что каждые 10 секунд на сервер **dweet.io** отправляется запрос и приходит ответ:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
Content-Length: 147
Date: Mon, 16 Mar 2015 10:03:37 GMT
Connection: keep-alive

{"this":"succeeded","by":"dweeting","the":"dweet",
"with":{"thing":"myesp8266","created":"2015-03-16T10:03:37.053Z",
"content":{"temperature":24, "humidity":39}
   }
}
```

Если вы видите в ответе слово `succeeded` (успешно) — поздравляю, вы только что загрузили данные в облако при помощи ESP8266!

[5] Рекомендуется после загрузки прошивки «сбросить» ESP8266 кнопкой сброса или отключением питания — иначе она может не соединиться с сайтом. Это связано с особенностями работы стека TCP в микросхеме. — *Прим. пер.*

Отображение данных при помощи сервиса *freeboard.io*

Теперь мы хотели бы получить возможность отображать записанные в облако данные на экране, находясь в любой точке мира. Для этого мы воспользуемся сервисом **freeboard.io**, который нравится мне не меньше, чем уже известный нам **dweet.io**.

Для использования сервиса **freeboard.io**:

1. Создадим аккаунт на сайте **https://www.freeboard.io.**
2. Создадим новую приборную панель, а внутри этой панели создадим новый источник данных `datasource`. Свяжем этот источник с нашим устройством на **dweet.io** — тем, имя которого вы задали в коде для ESP8266 (рис. 3.3).

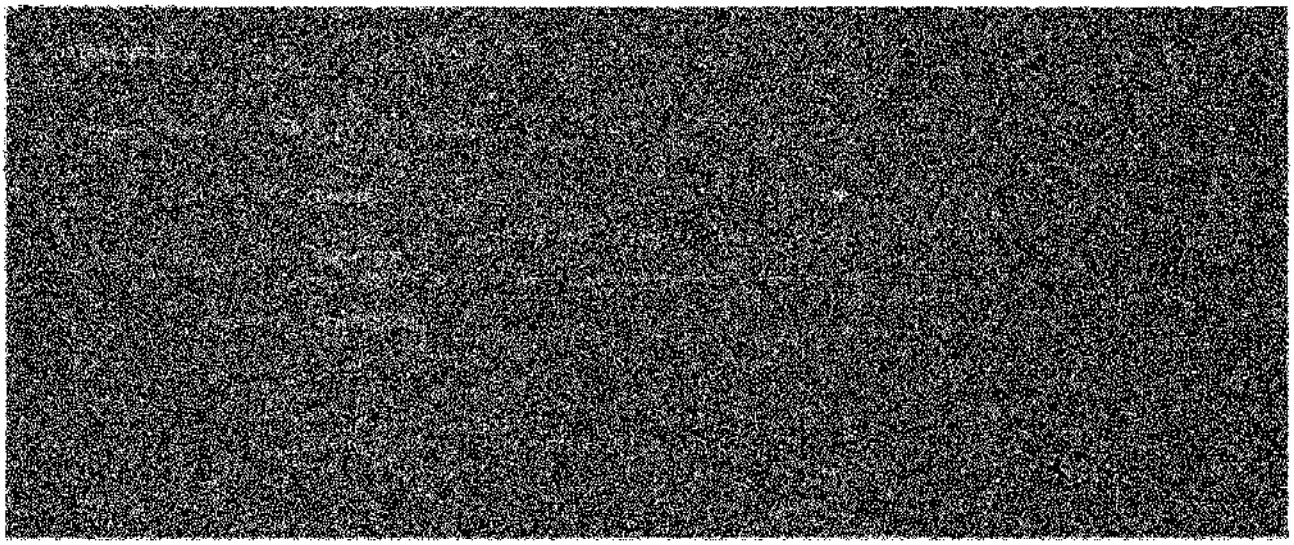


Рис. 3.3. Создаем приборную панель и источник данных

3. Теперь создадим новый виджет измерительного прибора **Gauge**, который будет отображать температуру (рис. 3.4).

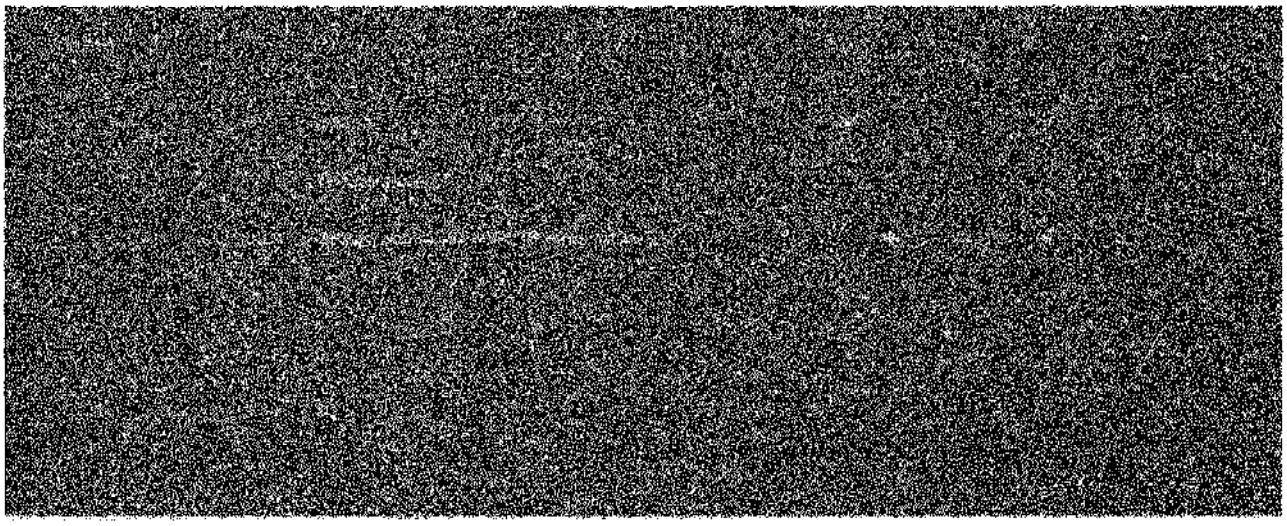


Рис. 3.4. Создаем виджет измерительного прибора

Готовый прибор показан на рис. 3.5.

Вы должны увидеть, что данные температуры поступают с устройства ESP8266 каждые 10 секунд и немедленно отображаются на панели **freeboard.io**.

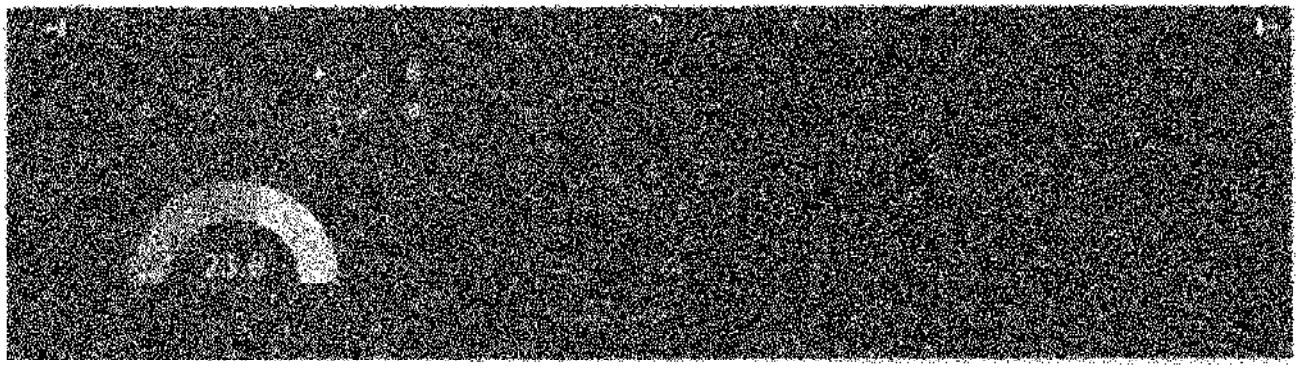


Рис. 3.5. Готовый виджет измерительного прибора

Поздравляю, вы создали компактный и недорогой датчик температуры, который хранит и отображает данные через облако!

Вы можете проделать то же самое и с виджетом влажности и поместить его на приборной панели, как показано на рис. 3.6.

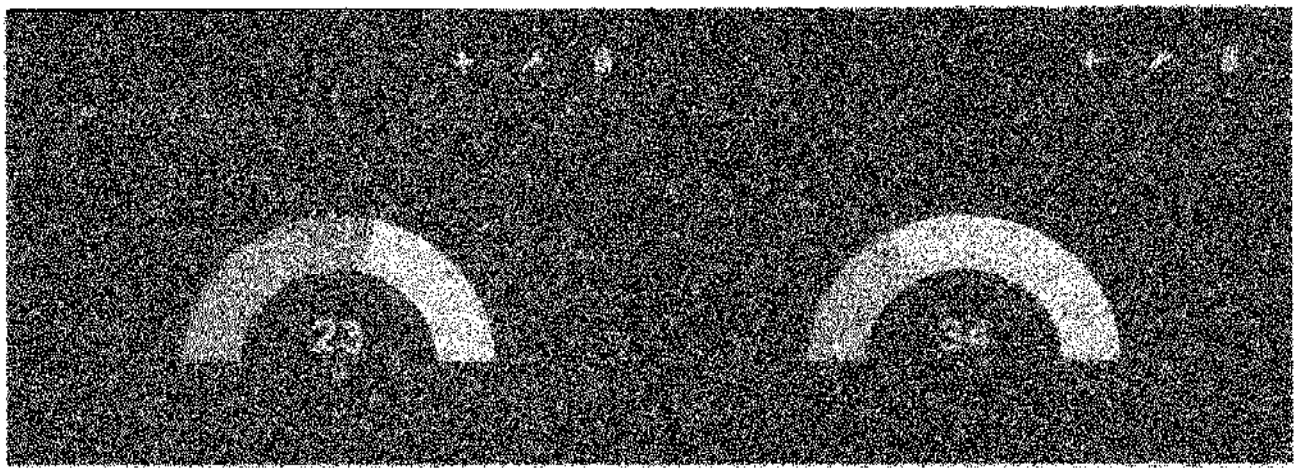


Рис. 3.6. Виджеты температуры и влажности на приборной панели

На приборную панель сервиса **freeboard.io** можно с легкостью добавить и другие виджеты. Например, графики, отображающие историю измерений температуры и влажности. Воспользуемся для этого виджетом **Sparkline**. Он создается точно так же, как и измерительный прибор **Gauge**. Начнем с виджета графика температуры (рис. 3.7).

Теперь можно повторить те же действия и для графика влажности (рис. 3.8).

Окончательный результат вывода графиков показан на рис. 3.9.

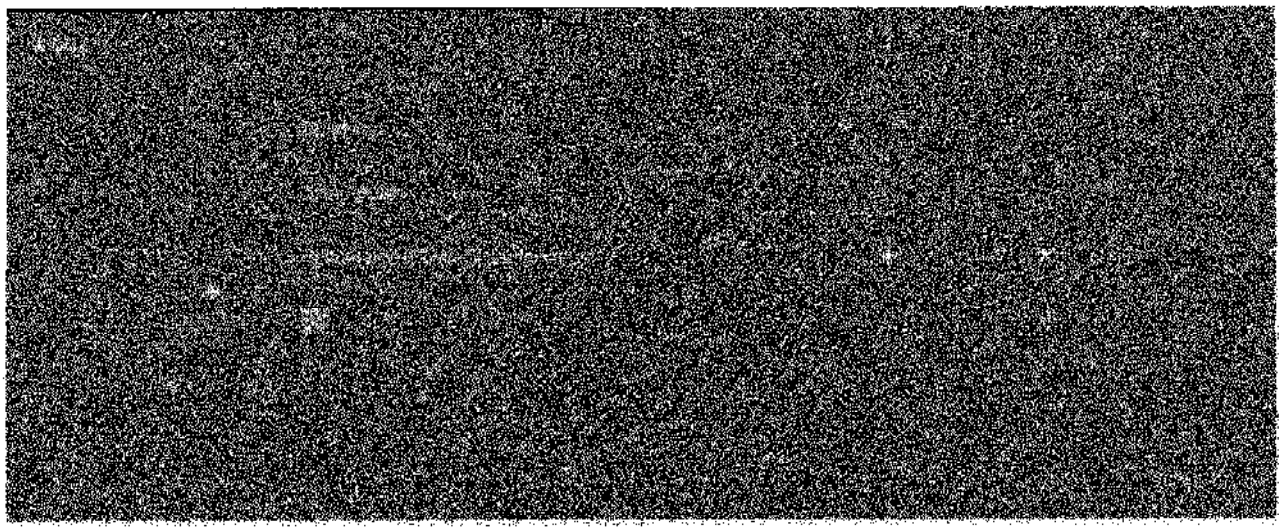


Рис. 3.7. Создаем виджет графика температуры

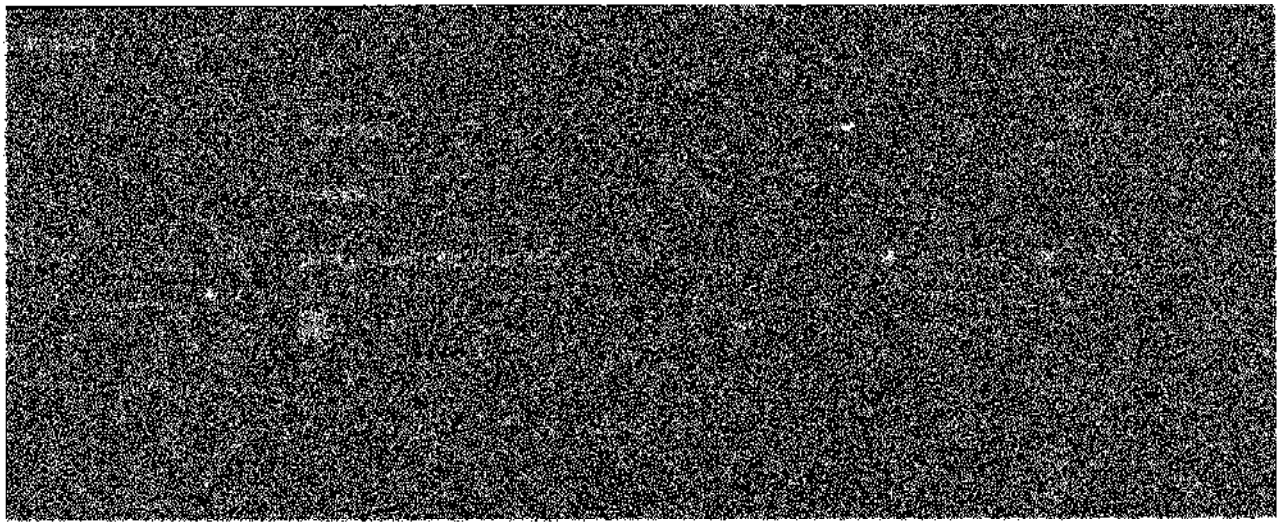


Рис. 3.8. Создаем виджет графика влажности

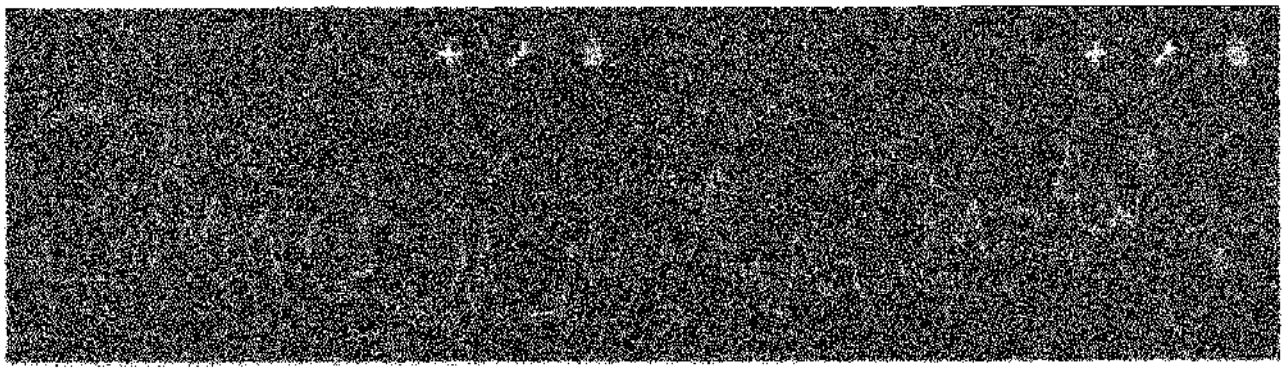


Рис. 3.9. Окончательный вид приборной панели с графиками температуры и влажности

Заключение

Подведем итог сделанного в этом проекте. Мы создали простой и недорогой измеритель температуры и влажности на основе микросхемы ESP8266. Мы запрограммировали его на автоматическую запись данных в сервис **dweet.io** и отобразили результаты измерений на онлайновой приборной панели сервиса **freeboard.io**.

Есть много способов улучшить этот проект — например, просто добавить больше датчиков и отображать их данные на приборной панели **freeboard.io**.

Вы можете также полностью изменить проект, подключив к плате ESP8266 датчик движения и настроив устройство так, чтобы оно отправляло вам сигнал тревоги каждый раз, когда в зоне действия этого датчика обнаружено движение. Такой сигнал может быть отправлен вам в виде сообщения по электронной почте или через Твиттер. Возможности безграничны!

В следующей главе мы научимся использовать еще одну важную возможность Интернета вещей — управление устройствами через облако.

4

Управляем устройствами отовсюду

Из предыдущей главы мы узнали, как с помощью ESP8266 выполнять измерения и отправлять данные на сервер, а также как сохранять данные в облаке и отображать их на приборной панели, доступной, где бы мы ни находились.

В этой главе мы рассмотрим обратную ситуацию — управление устройством из любой точки мира. Обычно для этого требуется протокол MQTT, который отличается от классического HTTP. Однако протокол MQTT достаточно сложен для применения, поэтому мы воспользуемся разработанной мной библиотекой aREST, существенно упрощающей все процессы. Библиотека aREST включает в себя лишь самые важные функции MQTT, но этого более чем достаточно для управления устройствами из любого места!

Мы рассмотрим в этой главе два примера удаленного управления при помощи приборной панели облачного сервиса: изменение яркости светодиода и включение/выключение настольной лампы из любой точки мира.

Завершив этот проект, вы сможете управлять лампой, подключенной к ESP8266, простым щелчком на приборной панели, доступной отовсюду. Начинаем!

Оборудование и программное обеспечение

Для проектов этой главы потребуется следующее оборудование:

- прежде всего — это плата с микросхемой ESP8266. В основном я использую в этой книге модуль Adafruit Huzzah ESP8266, но любой другой модуль ESP8266 будет работать здесь ничуть не хуже;
- устройство, способное управлять высоким (из розетки домашней электросети) напряжением питания лампы или иного аналогичного устройства. Сначала я использовал для тестирования проекта обычное реле, но быстро перешел на уст-

ройство PowerSwitch Tail Kit, которое позволяет просто и безопасно включить высоковольтные устройства в ваши проекты[1];

- светодиод — я взял обычный красный светодиод диаметром 5 мм, включенный последовательно с резистором 330 Ом;
- модуль USB FTDI 3,3/5 В для загрузки программ в ESP8266[2];
- беспаечная макетная плата и соединительные провода.

Вот конкретный перечень использованных в проектах компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266: **https://www.adafruit.com/products/2471**;
- модуль FTDI USB: **https://www.adafruit.com/products/284**;
- светодиод: **https://www.sparkfun.com/products/9590**;
- резистор 330 Ом: **https://www.sparkfun.com/products/8377**;
- реле PowerSwitch Tail Kit: **https://www.sparkfun.com/products/10747**;
- макетная плата: **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

Вам также понадобится устройство, которым вы будете управлять. В качестве подобного устройства я использовал настольную лампу мощностью 30 ватт, но вы можете подключить любой другой бытовой прибор, потребляемый ток которого не превышает максимально допустимое значение для реле PowerSwitch Tail Kit. Можно управлять лампой, кофеваркой, стиральной машиной, электроплитой и т. п. Для отладки же можно использовать простое реле или светодиод.

Если вы до сих пор не установили среду разработки Arduino IDE, сделайте это сейчас. Самую свежую версию среды можно скачать по адресу: **http://www.arduino.cc/en/Main/Software**.

Затем установите библиотеки aREST и PubSubClient, воспользовавшись для этого Менеджером библиотек Arduino IDE.

Создайте аккаунт на сайте приборной панели aREST: **http://dashboard.arest.io/**.

Программирование модуля ESP8266 и управление светодиодом

Теперь мы приступаем к подготовке модуля ESP8266, которая заключается в сборке схемы и загрузке прошивки в плату — чтобы она могла принимать команды из облака.

---

[1] Вместо него можно использовать обычный модуль реле с оптической развязкой, упомянутый в предисловии к русскому изданию. — *Прим. пер.*

[2] При использовании платы NodeMCU этот модуль не понадобится. — *Прим. пер.*

Итак, установите модуль ESP8266 на макетную плату и подключите к нему адаптер USB FTDI (рис. 4.1). Длинный вывод светодиода соедините с резистором. Оставшийся вывод резистора соедините с выводом 5 (D1) платы ESP8266, а второй вывод светодиода — с общим проводом (вывод GND)[3].

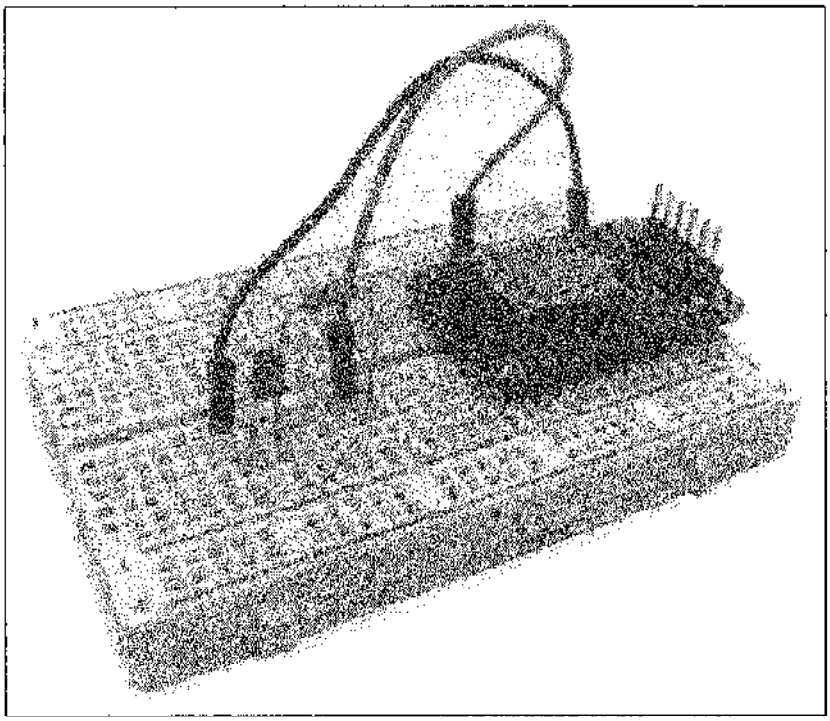


Рис. 4.1. Монтажная схема подключения светодиода

Чтобы плата могла получать команды из облака, ее надо соответствующим образом запрограммировать (листинг 4.1).

Листинг 4.1. Управление светодиодом (часть 1)

```
// Подключаем нужные библиотеки
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
// Создаем объекты клиентов
WiFiClient espClient;
PubSubClient client(espClient);
// Создаем объект aREST
aREST rest = aREST(client);
// Уникальный идентификатор для cloud.arest.io
char* device_id = "9u2co4";
```

[3] Обратите внимание, что в сервисе aREST используется нумерация не портов GPIOx, а цифровых выводов платы. То есть, когда мы говорим про вывод номер 5 в сервисе **aREST.io**, это соответствует выводу D5, а не GPIO5. Таблица взаимного соответствия вводов показана на рис. Пр.2 *предисловия к русскому изданию. — Прим. пер.*

```
// Параметры вашей сети Wi-Fi
const char* ssid = "your_wifi_name";
const char* password = "your_wifi_password";
// Функции
void callback(char* topic, byte* payload, unsigned int length);
void setup(void)
{
  // Инициализируем последовательный порт
  Serial.begin(115200);
  // Задаем функцию обратного вызова
  client.setCallback(callback);
  // Задаем имя и ID устройства
  rest.set_id(device_id);
  rest.set_name("devices_control");
  // Подключение к сети Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  // Задаем канал сообщений
  char* out_topic = rest.get_topic();
}
void loop() {
  // Подключаемся к облаку
  rest.handle(client);
}
// Обработка сообщений, поступающих из канала (каналов)
void callback(char* topic, byte* payload, unsigned int length) {
  rest.handle_callback(client, topic, payload, length);
}
```

Рассмотрим этот скетч поподробнее. Он начинается с подключения необходимых библиотек:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
```

Затем мы объявляем клиентов Wi-Fi и PubSub:

```
WiFiClient espClient;
PubSubClient client(espClient);
```

После этого создаем объект aREST, который даст плате возможность обрабатывать команды, поступающие из облака:

```
aREST rest = aREST(client);
```

Обратите внимание на следующие два важных пункта в скетче, которые вы должны изменить для своего проекта:

- прежде всего, это ID (идентификатор) устройства:

```
char* device_id = "9u2co4";
```

  Здесь вы должны указать идентификатор именно вашего устройства. Существует вероятность, что другой пользователь облачного сервиса использует аналогичный идентификатор, поэтому вы должны быть уверены, что задаете нечто совершенно уникальное;

- затем вы должны задать имя и пароль вашей сети Wi-Fi:

```
const char* ssid = "your_wifi_name";
const char* password = "your_wifi_password";
```

Далее, внутри функции `setup()` мы указываем функцию обратного вызова. Буквально через минуту мы узнаем, что это такое, но сейчас просто отметим, что это относится к клиенту `PubSubClient`:

```
client.setCallback(callback);
```

Оставаясь внутри функции `setup()`, задаем ID и имя устройства:

```
rest.set_id(device_id);
rest.set_name("devices_control");
```

В функции `loop()` скетча мы гарантируем, что плата всегда пробует соединиться с облачным сервисом:

```
rest.handle(client);
```

И наконец, мы определяем функцию обратного вызова. Она просто обслуживает входящие запросы из облака и отвечает нужным образом:

```
void callback(char* topic, byte* payload, unsigned int length) {
  rest.handle_callback(client, topic, payload, length);
}
```

Что ж, настало время испытать наш проект! Для начала мы лишь убедимся, что он и правда подключается к облачному серверу.

Загрузите в плату прошивку. Не забудьте предварительно исправить в ее коде ID устройства и настройки вашей сети Wi-Fi. О том, как загружать прошивку в плату, рассказано в предыдущих главах книги.

Готовый к использованию файл программы находится в папке ch4_1 сопровождающего книгу электронного архива (см. *приложение*).

Все запросы к плате будут выполняться посредством облачного сервера aREST, расположенного по адресу: **http://cloud.arest.io**.

Чтобы протестировать его, откройте свой любимый браузер и введите в адресную строку запрос:

```
http://cloud.arest.io/9u2co4/id
```

Разумеется, вам следует заменить в этом запросе ID устройства на тот, который вы указали в скетче. Запрос проверяет, подключена ли ваша плата к облачному серверу. Если это так, в своем браузере вы увидите ответ наподобие такого:

```
{
"id": "9u2co4",
"name": "devices_control",
"connected": true
}
```

Это означает, что ваше устройство находится в Сети и отвечает на команды.

На этом месте мы пока остановимся и попробуем включать светодиод через облако. Но прежде всего надо настроить вывод GPIO5 как выход. Для этого введите в браузере:

```
https://cloud.arest.io/9u2co4/mode/5/o
```

Вы должны получить подтверждающее сообщение:

```
{
"message": "Pin D5 set to output",
"id": "9u2co4",
"name": "devices_control",
"connected": true
}
```

Теперь включим светодиод:

```
https://cloud.arest.io/9u2co4/digital/5/1
```

Вы тут же увидите, что светодиод зажегся, а в браузере появилось подтверждение:

```
{
"message": "Pin D5 set to 1",
"id": "9u2co4",
"name": "devices_control",
"connected": true
}
```

Поздравляю, теперь вы можете управлять светодиодом из любой точки мира! Чтобы узнать больше про aREST и команды, посетите сайт **http://arest.io/**.

Управление светодиодом через облачную приборную панель

Иметь возможность управлять светодиодом командами через адресную строку браузера — уже хорошо, но все же хотелось бы делать это из любой точки мира при помощи удобного графического интерфейса.

Именно этим мы сейчас займемся при помощи сервиса, который называется **aREST dashboard** (приборная панель aREST), — мы сможем не только включать светодиод, но и менять его яркость движком прямо в окне браузера.

Если вы еще не создали аккаунт на сайте **http://dashboard.arest.io/**, сделайте это сейчас.

Создайте новую приборную панель в главном окне сервиса (рис. 4.2).

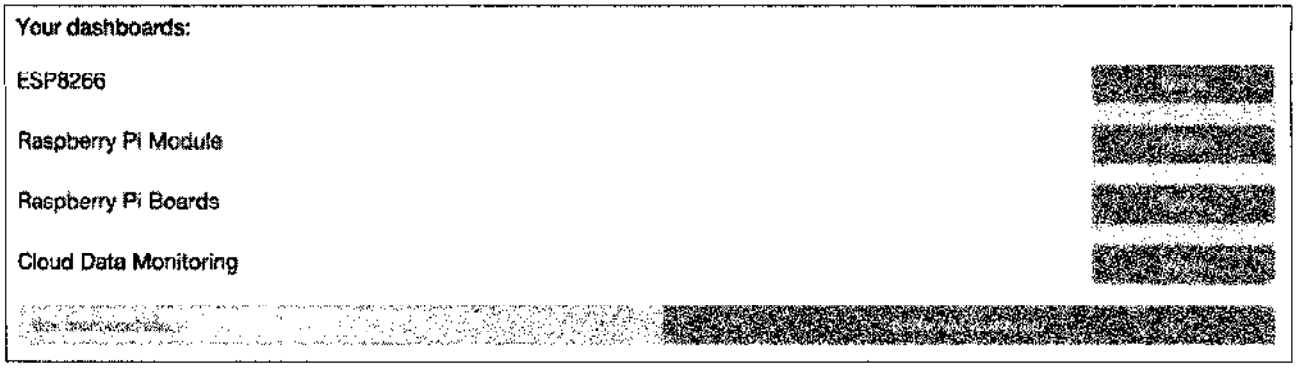


Рис. 4.2. Интерфейс управления приборными панелями сервиса aREST

Затем внутри приборной панели (рис. 4.3) создайте для управления яркостью светодиода новый элемент типа **Analog** и подставьте ID устройства, которым хотите управлять. Не забудьте также указать номер вывода GPIO5. Вы сразу увидите на приборной панели свой новый элемент (рис. 4.4).

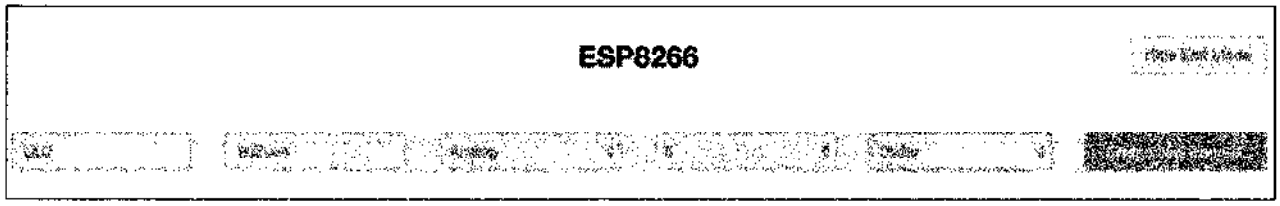


Рис. 4.3. Создание нового элемента управления

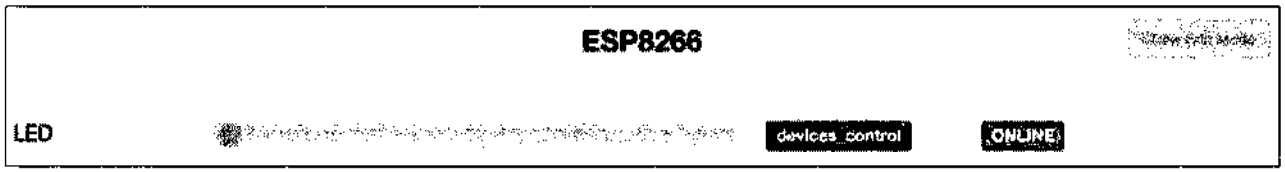


Рис. 4.4. Новый элемент на приборной панели aREST

Теперь испытаем движок, который только что создали. Как вы можете заметить, если переместить движок и отпустить кнопку мыши, то яркость светодиода изменится пропорционально смещению (рис. 4.5). Это весьма удобный способ управления, например, яркостью светодиодного освещения у вас дома.

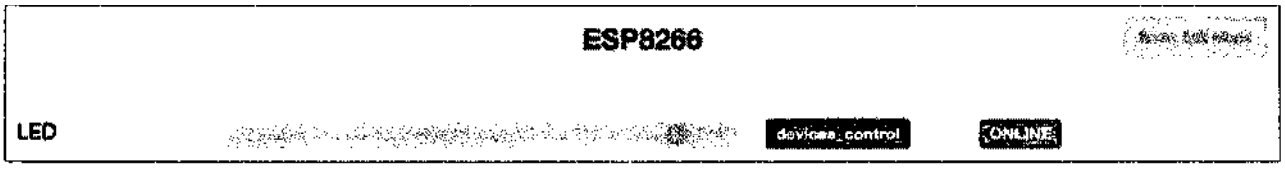


Рис. 4.5. Движок управления в действии

Управление лампой из любой точки мира

Распространим рассмотренные принципы работы и код предыдущих проектов на удаленное управление лампой (или любым другим потребителем).

Схема устройства для этого проекта очень проста. Поместите модуль ESP8266 на макетную плату. Соедините вывод Vin+ модуля PowerSwitch Tail с выводом GPIO5 (D1) ESP8266. Два оставшихся вывода PowerSwitch Tail соедините с общим проводом (GND) ESP8266 (рис. 4.6).

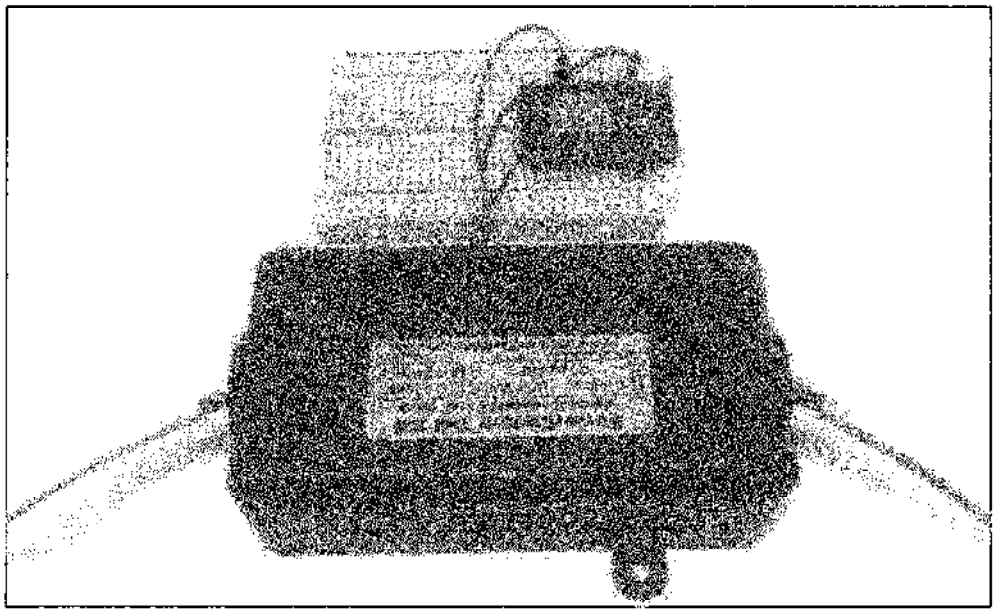


Рис. 4.6. Монтажная схема управления лампой

Затем подключите к реле PowerSwitch Tail лампу (или другую нагрузку по своему выбору), а силовые его контакты подключите к розетке электросети.

Мы используем здесь тот же код, что и в предыдущем проекте — изменения коснутся лишь содержимого приборной панели.

Внутри приборной панели удалите элемент движка и создайте новый элемент типа **Digital**, используя прежние параметры (рис. 4.7), — вновь созданный элемент сразу появится на приборной панели (рис. 4.8).

Рис. 4.7. Настройка элемента типа **Digital**

Рис. 4.8. Готовый элемент **Digital** на приборной панели aREST

Проверим действие кнопок **On** и **Off** — как можно убедиться, они без промедления включают и выключают лампу или другое устройство, подключенное к реле PowerSwitch Tail.

Примером использования этого проекта в реальной жизни может служить наше устройство, подключенное к настольной лампе (рис. 4.9).

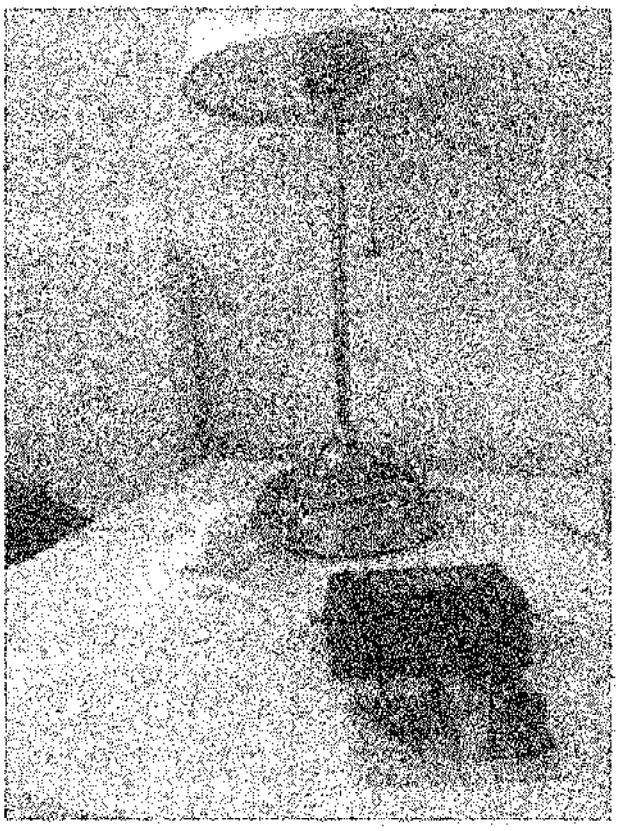


Рис. 4.9. Дистанционное управление настольной лампой

Теперь вы можете управлять любым электрическим прибором откуда угодно, используя облачную приборную панель, которую создали для этого проекта.

Заключение

Подведем итог наших достижений в этом проекте. Мы использовали протокол MQTT для удаленного управления светодиодом, лампой или любым электрическим устройством при помощи микросхемы ESP8266. Мы настроили ESP8266 как MQTT-клиент, а затем подключили приборную панель aREST, благодаря которой получили возможность удобно управлять нагрузкой из любой точки мира.

Вы можете усовершенствовать этот проект разными способами. Например, добавить несколько одинаковых модулей в панель управления aREST, чтобы удаленно управлять всеми лампами в доме. Можно также подключить к приборной панели несколько других датчиков, как мы и покажем далее в этой книге.

В следующей главе мы рассмотрим еще одну важную тему Интернета вещей — взаимодействие через ESP8266 с такими веб-сервисами, как Фейсбук и Твиттер.

5

Взаимодействие с веб-сервисами

К этому моменту мы разобрались с тем, как через Интернет управлять модулем ESP8266 Wi-Fi и подключенными к нему устройствами. Тем не менее это лишь небольшая часть того, что мы можем сделать с его помощью в рамках инфраструктуры Интернета вещей.

Чем мы займемся в этой главе? Будем использовать ESP8266 для взаимодействия с существующими веб-сервисами. Проще говоря, станем общаться через него с этими сервисами на физическом уровне. В качестве примера мы подключим наш модуль сначала к сервису погоды Yahoo Weather, а затем к Твиттеру и Фейсбуку. Начнем!

Оборудование и программное обеспечение

Для проектов этой главы потребуется следующее оборудование:

- прежде всего — это плата с микросхемой ESP8266. В основном я использую в этой книге модуль Adafruit Huzzah ESP8266, но любой другой модуль ESP8266 будет работать здесь ничуть не хуже;
- простой датчик влажности и температуры DHT11 — данные замеров с него мы будем отправлять через ESP8266 в Твиттер. Впрочем, вы можете использовать любой другой датчик;
- может понадобиться также модуль USB FTDI 3,3/5 В для программирования чипа ESP8266[1];
- беспаечная макетная плата и соединительные провода.

---

[1] При использовании платы NodeMCU этот модуль не понадобится. — *Прим. пер.*

Вот конкретный перечень использованных в проектах компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266: **https://www.adafruit.com/products/2471**;
- датчик DHT11: **https://www.adafruit.com/products/386**;
- модуль FTDI USB: **https://www.adafruit.com/products/284**;
- макетная плата: **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

Если вы до сих пор не установили среду разработки Arduino IDE, сделайте это сейчас. Самую свежую версию среды можно скачать по адресу: **http://www.arduino.cc/en/Main/Software**.

Для подключения ESP8266 к таким веб-сервисам, как Yahoo или Twitter, мы воспользуемся сервисом Temboo (**https://temboo.com/**), поэтому, прежде всего, создадим на этом сервисе свой аккаунт (рис. 5.1).

Temboo — это сервис-посредник и интегратор, который обеспечивает удобное взаимодействие с десятками различных сетевых сервисов в одном месте.

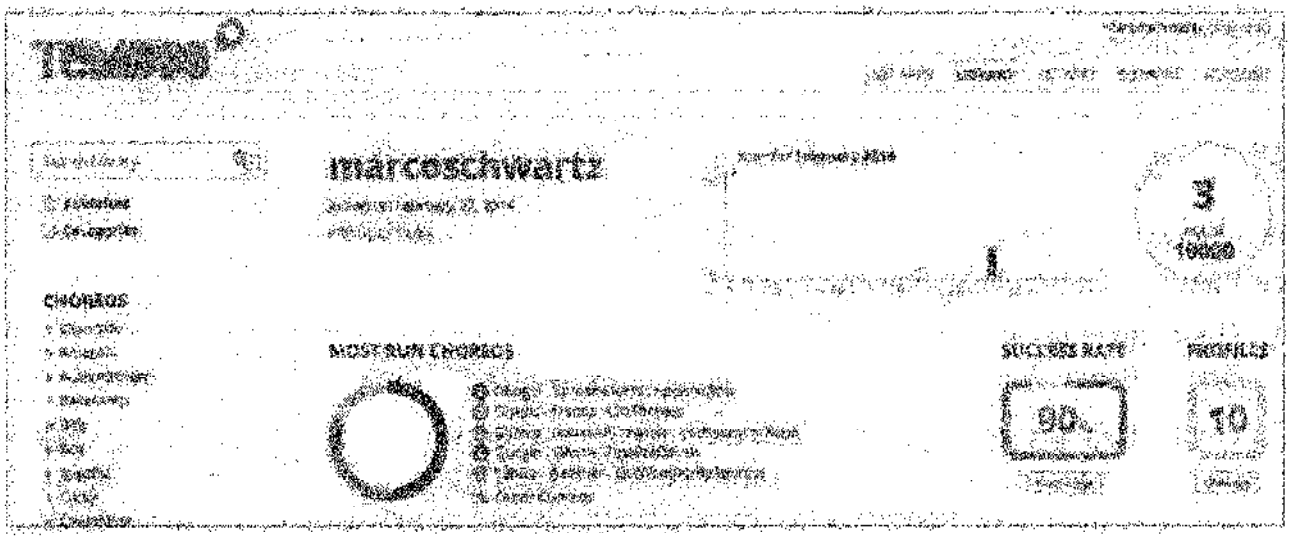


Рис. 5.1. Создаем аккаунт Temboo

Затем перейдем на вкладку **Account** и создадим первое приложение (рис. 5.2). Имя приложения (**Application**) и ключ (**Key**) понадобятся вам позже.

Для работы с сервисом Temboo вам потребуется доработанная версия его библиотеки Temboo, которую я модифицировал для работы с ESP8266[2]. Версию библиотеки, которая включена в код программы для этой главы, можно скачать по адресу: **https://github.com/openhomeautomation/iot-esp8266-packt**.

[2] Temboo периодически меняет функции API и встроенную библиотеку Arduino. Возможна ситуация, когда предложенная автором библиотека не сработает. В этом случае надо подождать, пока он ее обновит, и скачать повторно. — *Прим. пер.*

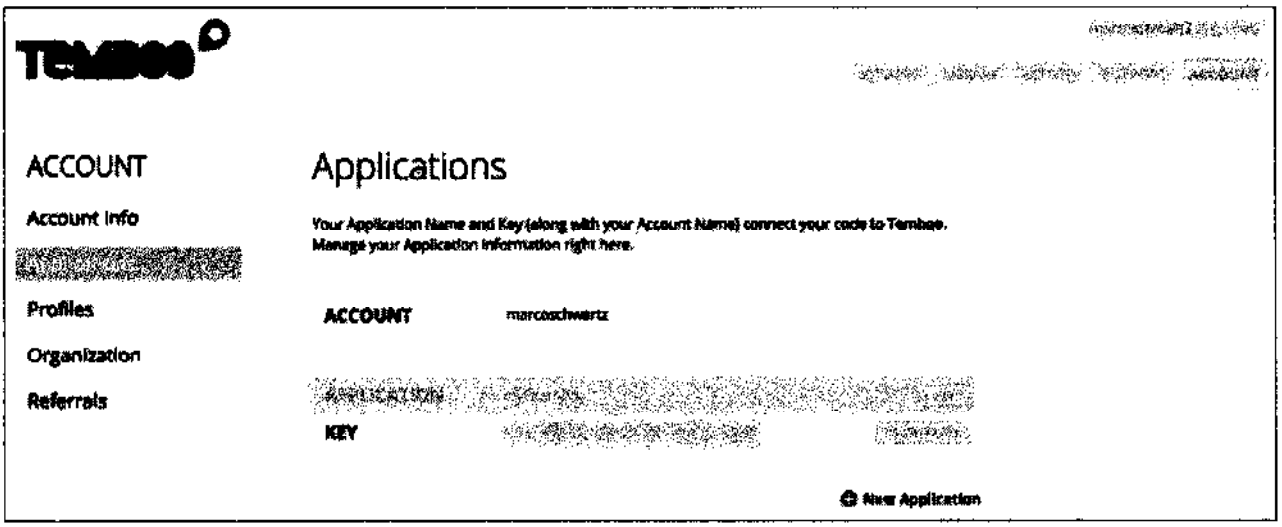


Рис. 5.2. Задаем имя приложения и генерируем его ключ

Найдите на своем компьютере главную папку библиотек Arduino. В случае ОС Windows путь к ней выглядит так: C:\Program Files (x86)\Arduino\libraries. В OS X щелкните на пункте меню **Show package content** ярлыка приложения Arduino. Предварительно сохраните в другом месте старую версию библиотеки Temboo — она может пригодиться при работе с проектами, отличными от проектов на ESP8266. Сохранив старую библиотеку Temboo, удалите ее из папки библиотек Arduino и запишите вместо нее новую.

Информация о погоде из сервиса Yahoo

В первом проекте этой главы мы научимся получать информацию о погоде с сервиса Yahoo Weather. Вы убедитесь, что с Temboo это действительно просто:

1. Перейдите по адресу:
   https://temboo.com/library/Library/Yahoo/Weather/GetWeatherByAddress/.
2. Поскольку вы пользуетесь Temboo впервые, вам нужно прописать в этом сервисе свой «шилд». Не волнуйтесь, я знаю, что мы здесь не используем реальный шилд Arduino, — просто в то время, когда я писал эту книгу, Temboo не поддерживал платы ESP8266.

   Лучше всего выбрать шилд Wi-Fi для Arduino (**Arduino WiFi**), потому что он наиболее близок в смысле программирования. Внутри формы настройки шилда введите имя и пароль вашей домашней сети Wi-Fi (рис. 5.3). Благодаря этому, вам не придется каждый раз вводить параметры доступа к Сети.

   На той же самой странице вы можете увидеть вновь созданный шилд (рис. 5.4).

   Затем узнаем, как автоматически сгенерировать код для проекта, — вам потом потребуется лишь незначительно изменить его для наших целей.
3. Введите в форме Temboo название своего города (рис. 5.5).
4. Теперь вы можете проверить или сгенерировать код нажатием кнопки **Run** (рис. 5.6).

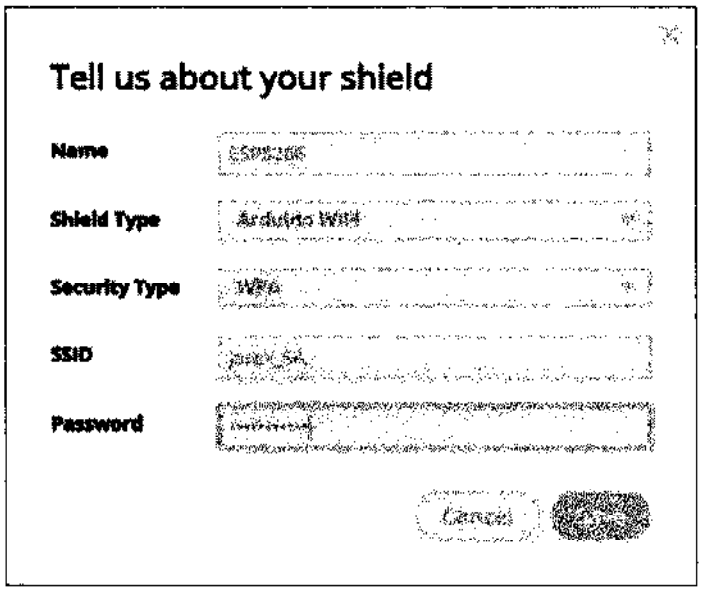


Рис. 5.3. Форма настройки шилда Wi-Fi в Temboo

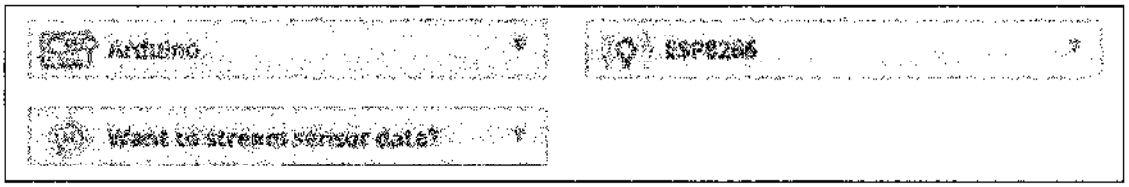


Рис. 5.4. Вновь созданный шилд в Temboo

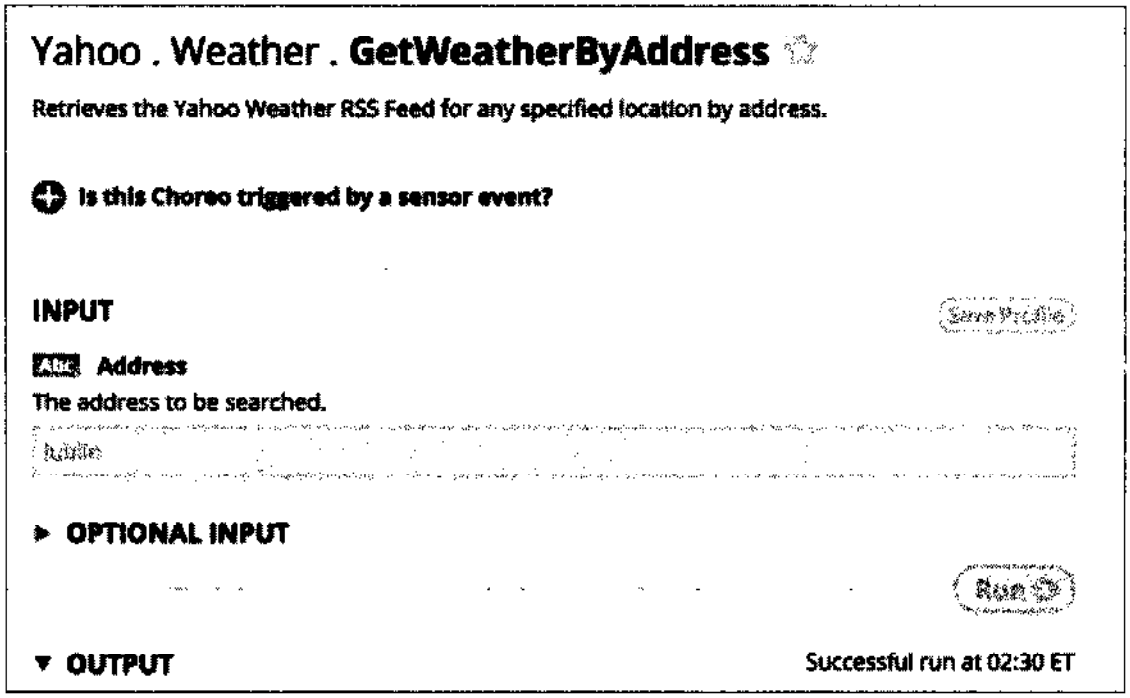


Рис. 5.5. Вводим в Temboo название своего города

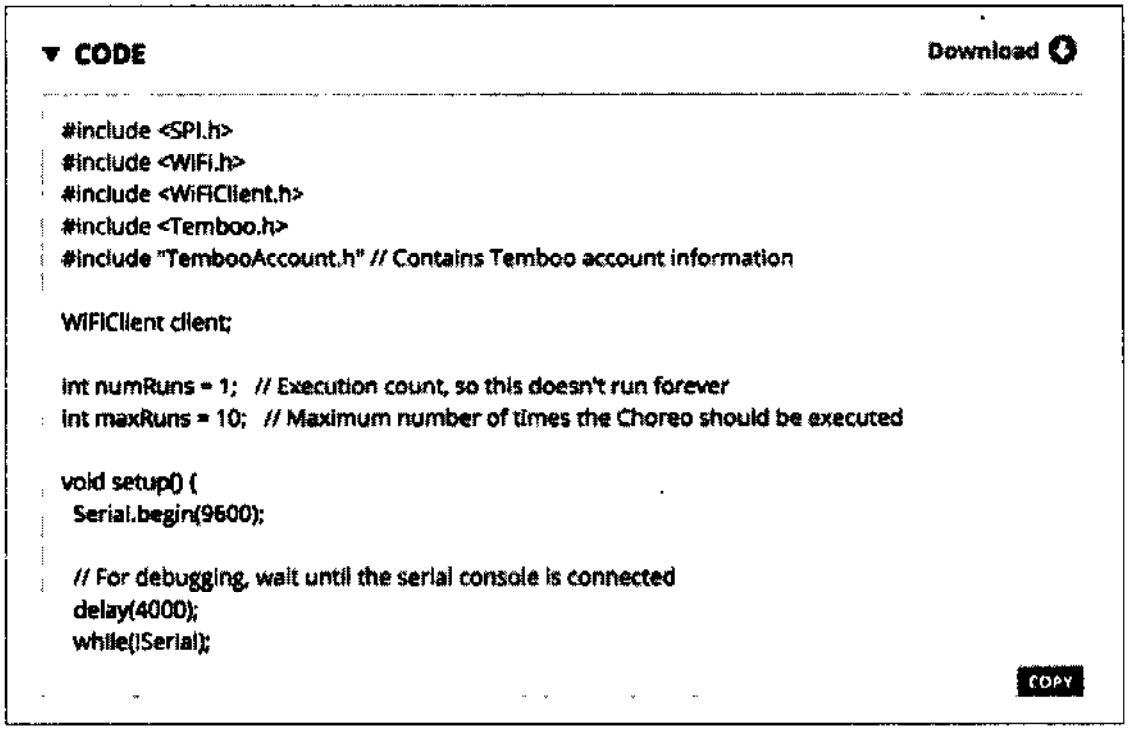


Рис. 5.6. Автоматически сгенерированный код программы

5. Скачайте сгенерированный код и сохраните его на своем компьютере. Вы можете далее работать с этим кодом или взять готовый код из папки ch5_Yahoo сопровождающего книгу электронного архива (см. *приложение*)[3].

6. Рассмотрим этот код детально. Чтобы использовать его с ESP8266, вам достаточно исправить в нем только две строки. Прежде всего надо изменить библиотеку Wi-Fi для ESP8266:

```
#include <ESP8266WiFi.h>
#include <Temboo.h>
#include "TembooAccount.h" // Информация об аккаунте Temboo
WiFiClient client;
```

   А далее указать, сколько раз мы хотим выполнить эту программу (Temboo имеет ограничение на количество бесплатных запросов):

```
int calls = 1;   // Счетчик запросов
int maxCalls = 2;   // Максимальное количество запросов
#define WIFI_SSID "your_wifi_ssid"
#define WPA_PASSWORD "your_wifi_password"
```

7. Внутри функции `setup()` мы подключаем плату к сети Wi-Fi:

```
int wifiStatus = WL_IDLE_STATUS;
// Determine if the WiFi Shield is present
Serial.print("\n\nShield:");
```

[3] Используя готовый код из файлового архива, не забудьте подставить в него свои значения **Application, Key** и парамстры доступа к сети Wi-Fi. — *Прим. пер.*

```
if (WiFi.status() == WL_NO_SHIELD) {
  Serial.println("FAIL");
  // If there's no WiFi shield, stop here
  while (true);
}
Serial.println("OK");
// Try to connect to the local WiFi network
while (wifiStatus != WL_CONNECTED) {
  Serial.print("WiFi:");
  wifiStatus = WiFi.begin(WIFI_SSID, WPA_PASSWORD);
  if (wifiStatus == WL_CONNECTED) {
    Serial.println("OK");
  } else {
    Serial.println("FAIL");
  }
  delay(5000);
}
Serial.println("Setup complete.\n");
}
```

8. **Внутри функции** `loop()` **отправляем запрос в Temboo и выводим результат в последовательный порт:**

```
if (calls <= maxCalls) {
Serial.println("Running GetWeatherByAddress - Run #" + String(calls++));
   TembooChoreo GetWeatherByAddressChoreo(client);

    // Инициализируем клиента Temboo
    GetWeatherByAddressChoreo.begin();

    // Данные из аккаунта Temboo
    GetWeatherByAddressChoreo.setAccountName(TEMBOO_ACCOUNT);
    GetWeatherByAddressChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
    GetWeatherByAddressChoreo.setAppKey(TEMBOO_APP_KEY);

    // Входные параметры Choreo
    GetWeatherByAddressChoreo.addInput("Address", "Moscow");

    // Определяем Choreo для запуска

GetWeatherByAddressChoreo.setChoreo("/Library/Yahoo/Weather/GetWeatherByAddr
ess");

   // Запускаем запрос и выводим ответ в терминал
    GetWeatherByAddressChoreo.run();
    Serial.println("Run");
    while (GetWeatherByAddressChoreo.available()) {
      char c = GetWeatherByAddressChoreo.read();
```

```
        ESP.wdtFeed();
        Serial.print(c);
      }
      GetWeatherByAddressChoreo.close();
    }
    Serial.println("\nWaiting...\n");

    delay(30000); // ждем 30 секунд перед следующим запросом
  }
```

9. Настало время испытать первый проект этой главы! Просто убедитесь, что данные внутри файла Temboo.h указаны правильно, переведите плату в режим программирования и загрузите прошивку.
10. Откройте монитор последовательного порта — вы должны видеть, что плата получает данные из сервиса Yahoo Weather и отправляет их в последовательный порт (рис. 5.7).

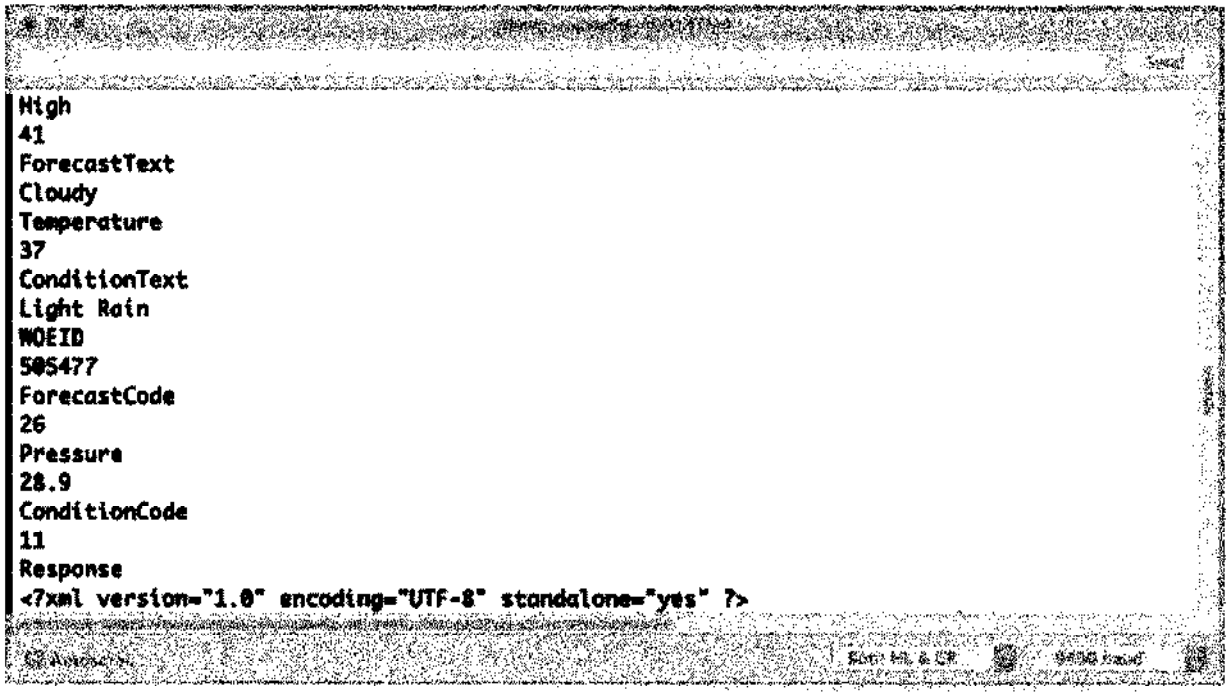


Рис. 5.7. Ответ сервиса Yahoo Weather в окне монитора последовательного порта

Теперь вы можете получать информацию о температуре, влажности и другие параметры погоды через ESP8266 прямо от сервиса Yahoo Weather!

Отправка значений температуры и влажности в Твиттер

Второй проект этой главы посвящен отправке данных измерений в Твиттер.

Здесь вам снова понадобится модуль ESP8266 с подключенным к нему датчиком DHT11. Установите модуль ESP8266 в макетную плату, расположите рядом с ним

датчик DHT11. Соедините вывод 1 датчика с линией питания, его вывод 2 подключите к выводу GPIO5 (D1) модуля ESP8266, а последний вывод датчика соедините с общим проводом (GND). Результат сборки показан на рис. 5.8.

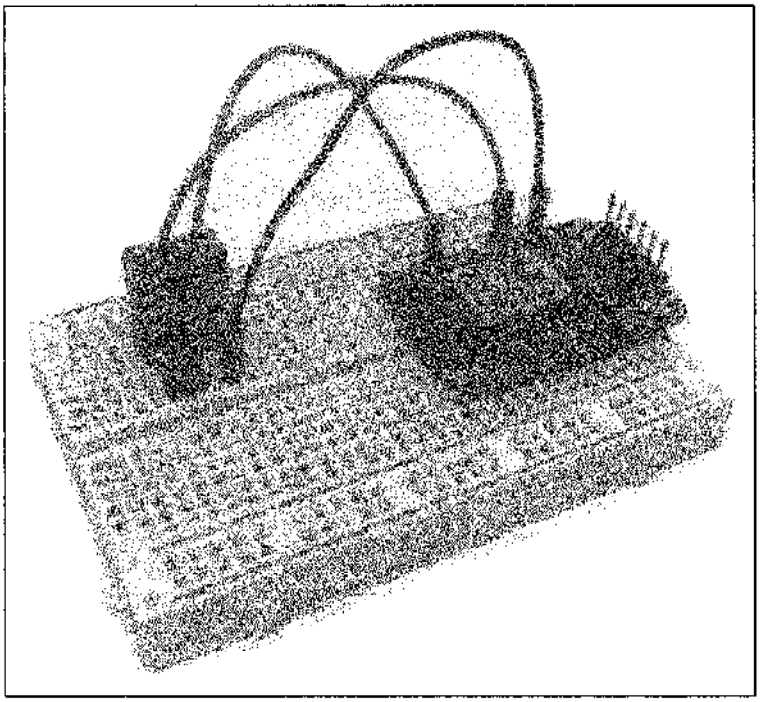


Рис. 5.8. Монтажная схема подключения компонентов для второго проекта главы

1. Чтобы создавать твиты с данными, вам надо зарегистрировать свое приложение в Твиттере, — пройдите для этого по адресу: **https://apps.twitter.com/**.

   Вас попросят авторизоваться в аккаунте Твиттера, после чего вы сможете увидеть свои подключенные приложения, если они у вас есть (рис. 5.9).

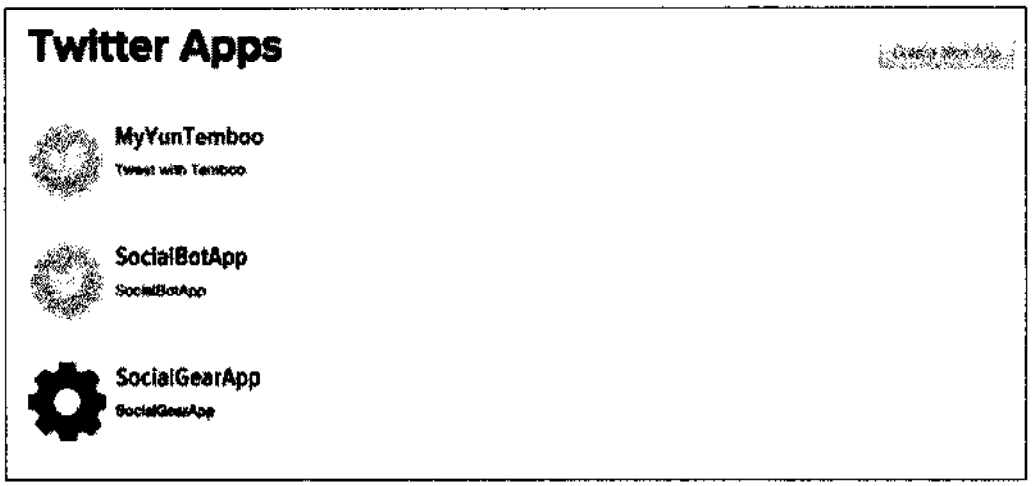


Рис. 5.9. Список подключенных приложений в аккаунте Твиттера

2. Нажмите кнопку **Create New App** (Создать новое приложение) и в открывшейся панели (рис. 5.10) введите имя и адрес сайта, назначив их по своему усмотрению.

Рис. 5.10. Создаем новое приложение для своего аккаунта

Создав приложение, вы получите доступ к такому важному параметру, как **Consumer Key** (Ключ пользователя) (рис. 5.11).

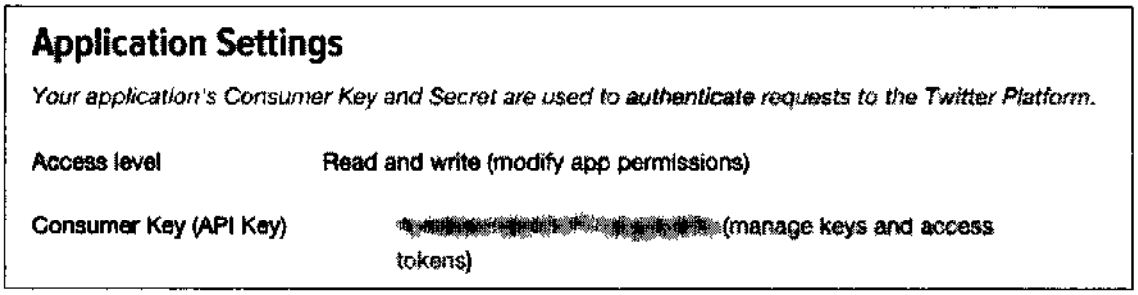


Рис. 5.11. Ключ пользователя для доступа к приложению

На вкладке **Application Settings** также представлен и **Consumer Secret** — секретный ключ API (рис. 5.12).

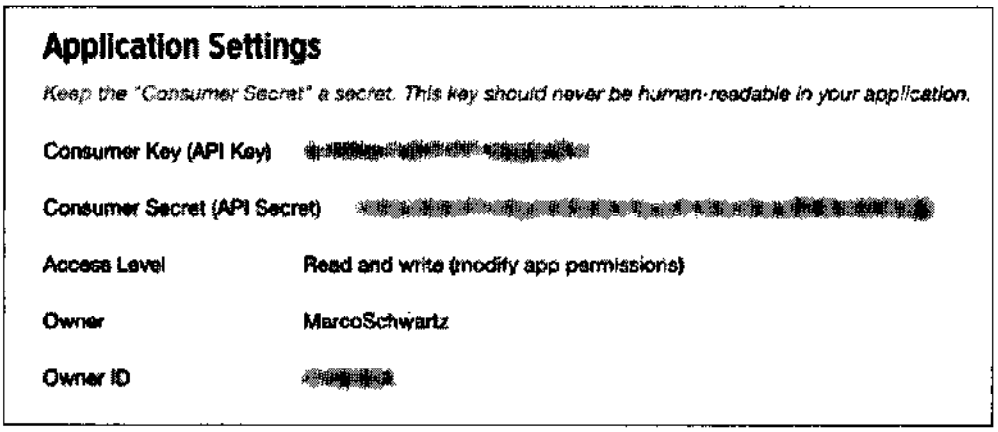


Рис. 5.12. Секретный ключ API

3. Затем на той же странице создайте токен для приложения (рис. 5.13) — вы получите токен доступа (**Access Token**) и секретный токен доступа (**Access Token Secret**). Все четыре только что полученных параметра понадобятся вам для настройки проекта в Temboo.

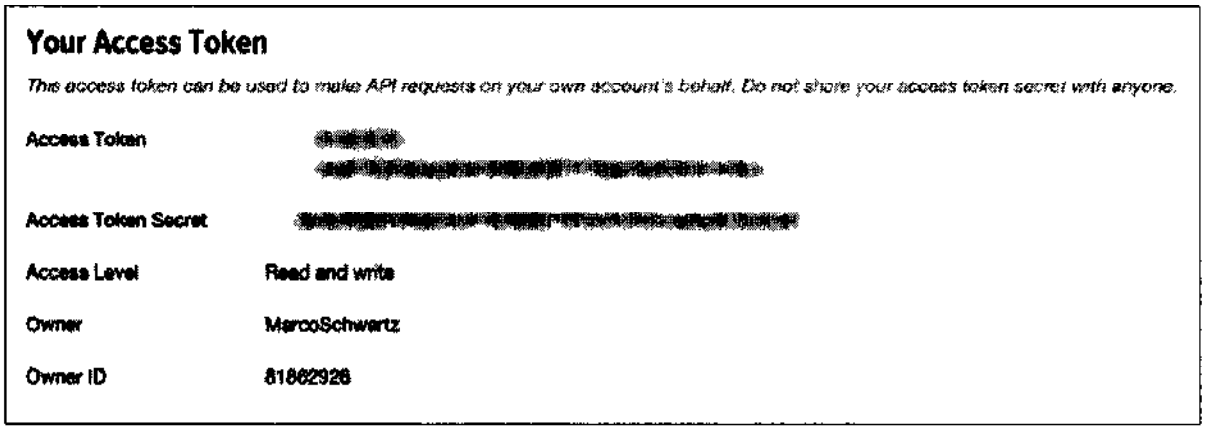


Рис. 5.13. Создаем токен приложения

4. Теперь пройдите по адресу: **https://temboo.com/library/Library/Twitter/Tweets/StatusesUpdate/** и внесите в четыре имеющихся там поля соответствующие значения параметров из приложения Твиттер (рис. 5.14).

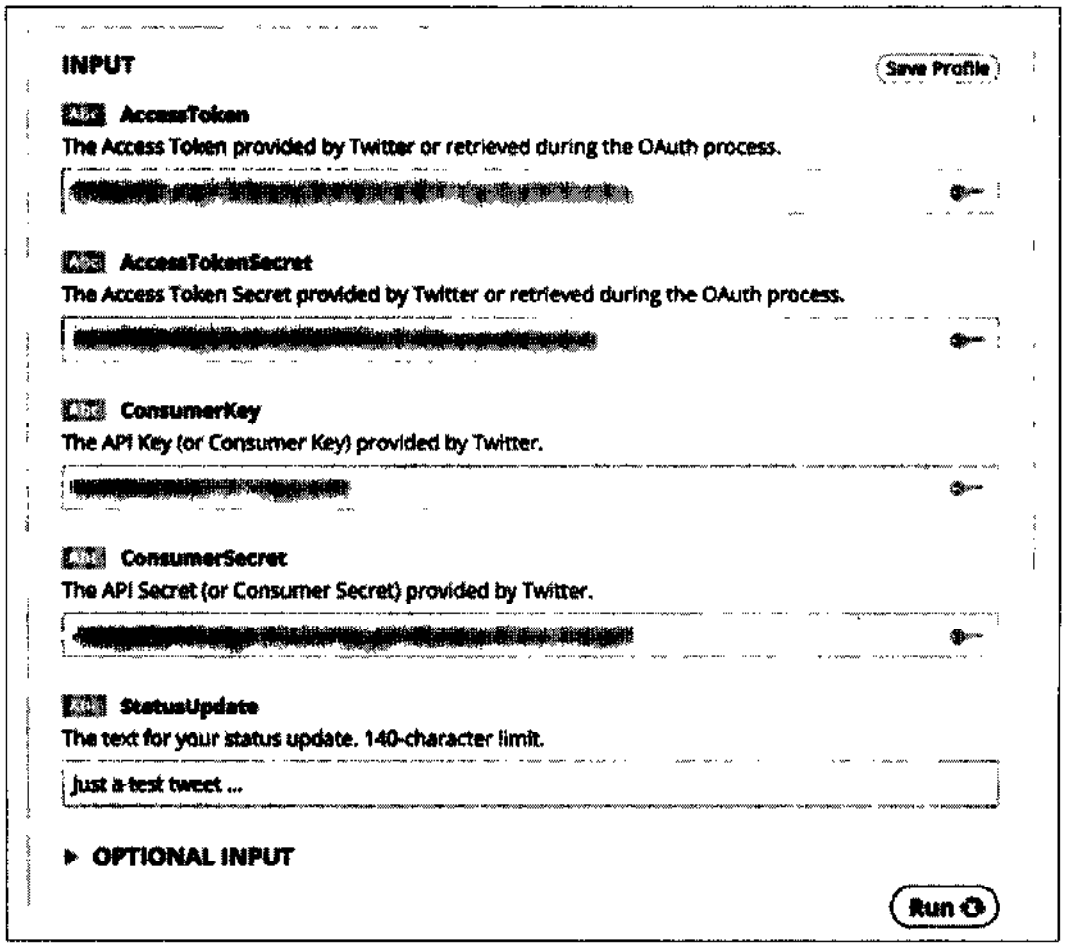


Рис. 5.14. Форма Temboo для ввода параметров доступа к приложению

5. Далее заново сгенерируйте код для скетча. Вы можете работать с этим кодом или взять готовый код из папки ch5_Twitter сопровождающего книгу электронного архива (см. *приложение*). Не беспокойтесь о содержимом поля **Status Update** — мы изменим его позже.

6. Как и в первом проекте, нужно лишь немного изменить скетч. Добавим библиотеки ESP8266WiFi и DHT:

```
#include <ESP8266WiFi.h>
#include <Temboo.h>
#include "TembooAccount.h" // Contains Temboo account information
#include "DHT.h"
WiFiClient client;
```

7. Вставим код для датчика DHT11:

```
// Вывод GPIO для датчика DHT11
#define DHTPIN 5
#define DHTTYPE DHT11
// Создаем объект датчика DHT11
DHT dht(DHTPIN, DHTTYPE);
```

8. Внутри функции `setup()` инициализируем датчик DHT11:

```
// Инициализация DHT11
dht.begin();
```

9. Внутри функции `loop()` добавим код для считывания данных датчика:

```
// Считывание данных
float h = dht.readHumidity();
float t = dht.readTemperature();
```

10. Теперь вам надо изменить значение строки **Status Update**, добавив в нее данные измерений:

```
String StatusUpdateValue = "The temperature is " + String(t) + " and the
humidity is " + String(h) + ".";
```

11. Ну вот и пришло время испытать скетч! Проверьте параметры внутри файла Temboo.h и загрузите прошивку в плату ESP8266.

    Через некоторое время вы должны увидеть новый твит с данными измерений, опубликованный в вашем аккаунте (рис. 5.15).

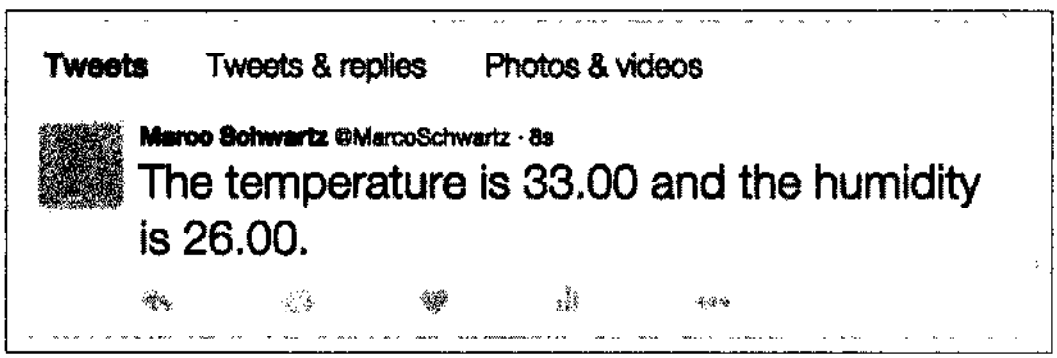


Рис. 5.15. Твит, опубликованный при помощи ESP8266

Поздравляем, теперь вы можете отправлять данные из своего ESP8266 в аккаунт Twitter!

Новый пост в Фейсбуке при помощи ESP8266

В завершающем проекте этой главы мы увидим, как взаимодействовать через сервис Temboo с сервисом Фейсбук с помощью ESP8266. Вы научитесь отправлять обновление статуса, но этот метод также можно использовать для размещения чего-либо на стене у друзей, публикации данных на странице и многого другого!

1. На первом этапе создадим в Фейсбуке приложение. Для этого перейдите по адресу: **https://developers.facebook.com/**.
2. Щелкните на опции **Add a New App** (Добавить новое приложение) (рис. 5.16).
3. Когда вас спросят о типе приложения, выберите **Website** (рис. 5.17).

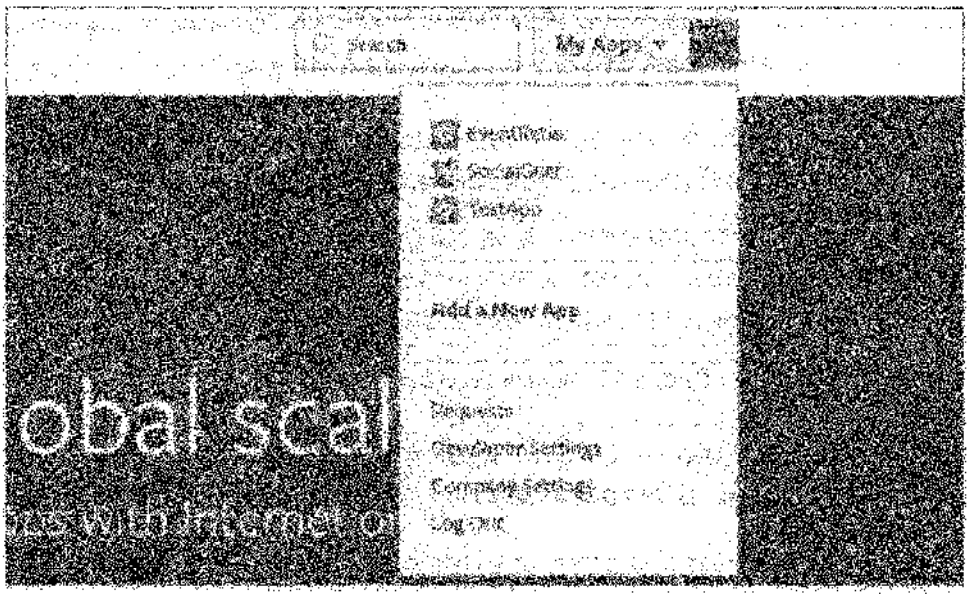


Рис. 5.16. Создаем новое приложение в Фейсбуке

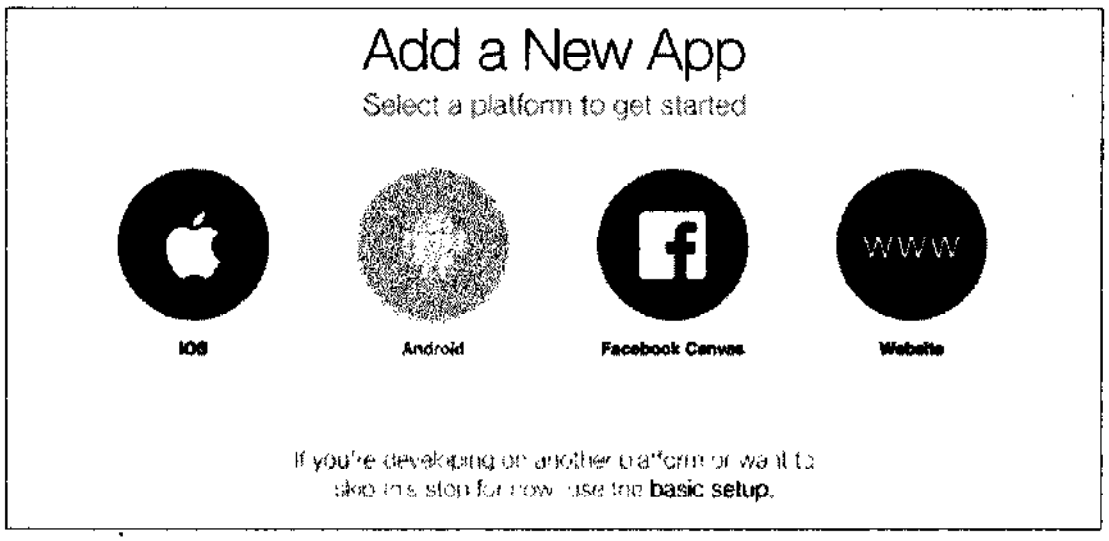


Рис. 5.17. Выбираем тип приложения

4. Затем присвойте приложению имя (рис. 5.18).
5. У вас спросят **Site URL** (адрес сайта). Этот адрес не имеет значения, поскольку не используется в проекте, — можете ввести все, что угодно (рис. 5.19).

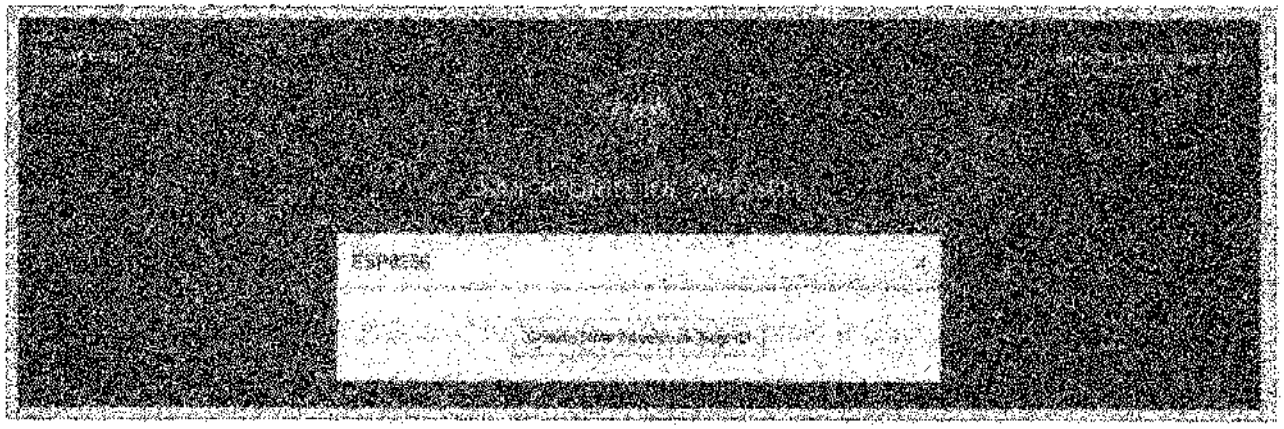


Рис. 5.18. Присваиваем приложению имя

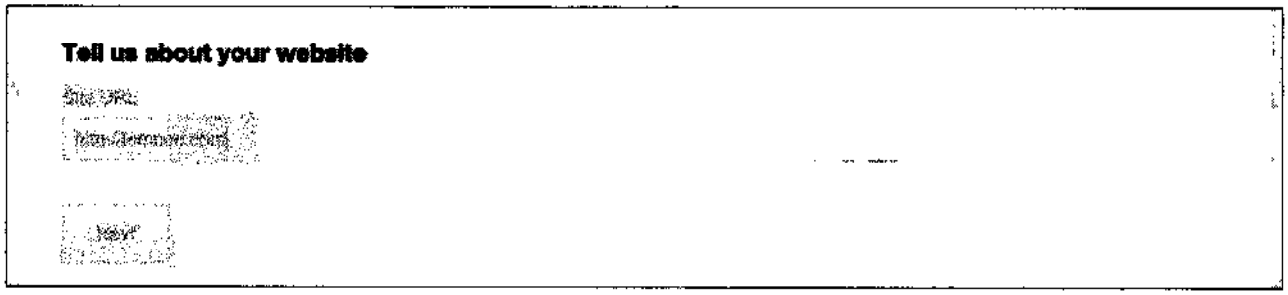


Рис. 5.19. Форма ввода имени сайта

6. Ваше приложение создано — на открывшейся панели (рис. 5.20) вы найдете его идентификатор (**App ID**) и секретный код (**App Secret**).

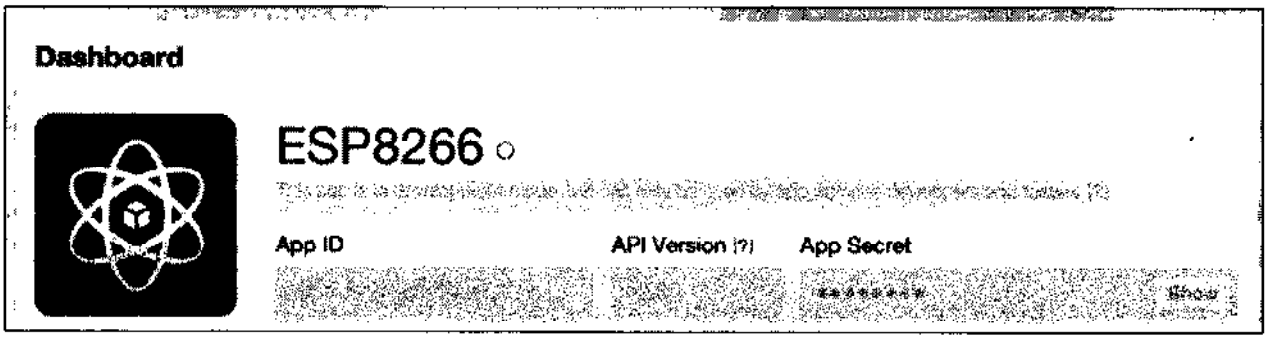


Рис. 5.20. Идентификатор и секретный код приложения

7. Перейдите на вкладку **Setting** и выберите пункт **Client OAuth Settings**. Введите в поле **Valid OAuth redirect URLs** адрес обратного вызова, показанный на рис. 5.21, но укажите в нем свое имя пользователя Temboo[4].

[4] На рис. 5.21 изображен адрес: **https://marcoschwartz.temboolive.com/callback/**. Замените в нем **marcoschwartz** на свое имя пользователя Temboo. — *Прим. пер.*

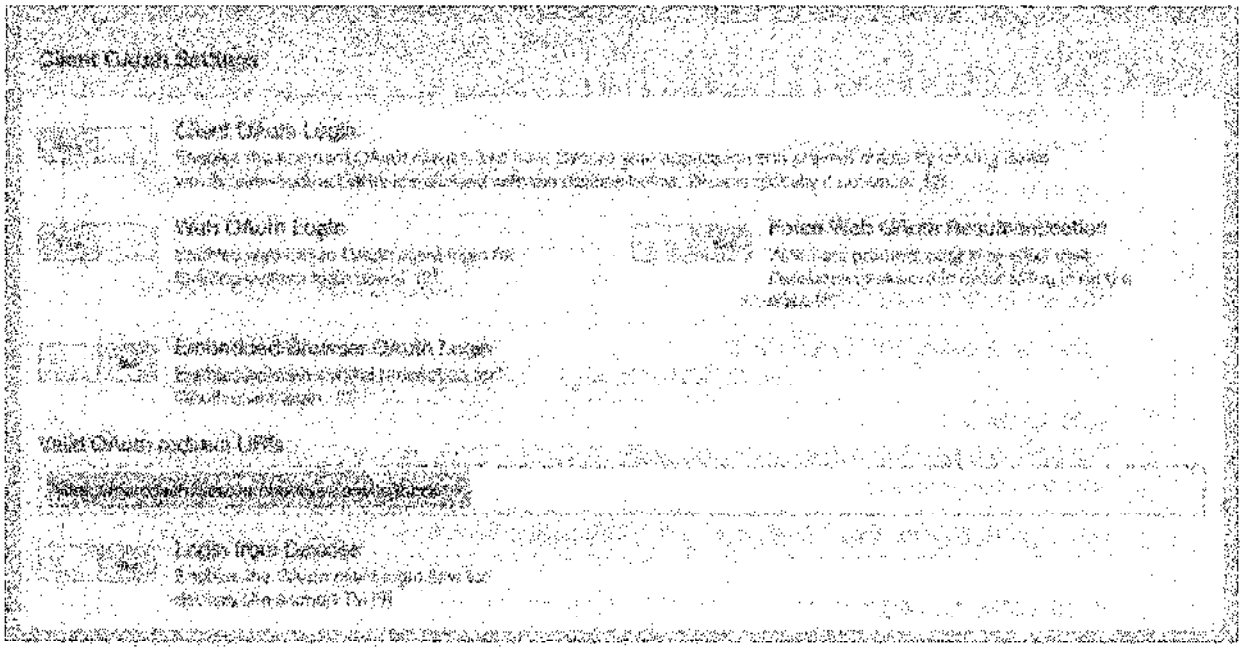


Рис. 5.21. Указываем адрес обратного вызова

8. На этом вы завершили настройки на стороне Фейсбука! Теперь вернемся в Temboo: **https://www.temboo.com/library/Library/Facebook/OAuth/**.
9. Эта страница (рис. 5.22) просто дает вам возможность получить токен доступа к Фейсбуку. Введите в соответствующие поля параметры App ID (который вы получили ранее) и Scope (область применения). Используйте значение: `publish_actions`.

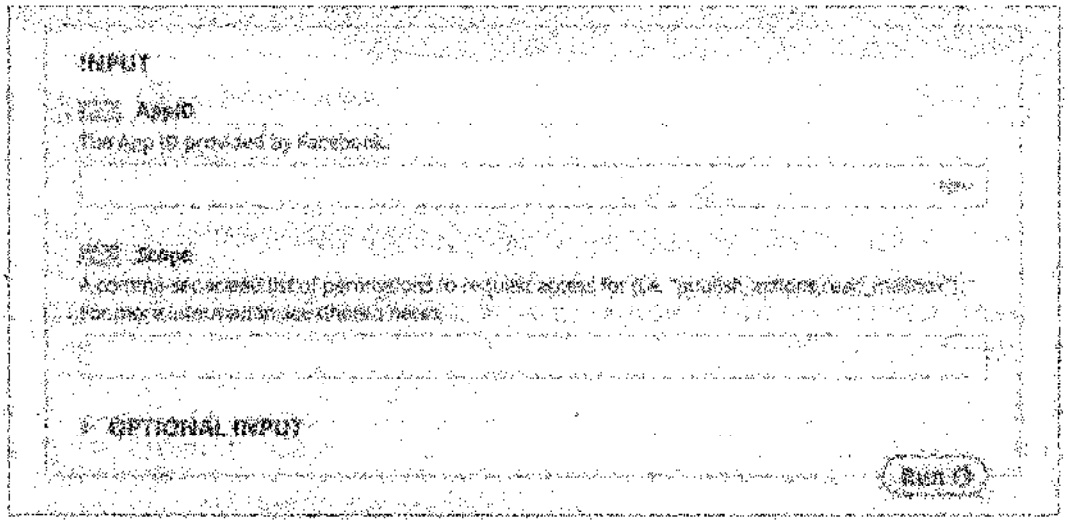


Рис. 5.22. Форма для получения токена доступа

10. Теперь вас попросят пройти по ссылке — сделайте это, авторизуйте приложение и вернитесь обратно, к странице окончательной авторизации. Введите запрошенные данные (рис. 5.23).
11. После щелчка по кнопке **Run** вы, наконец, получите свой токен доступа (рис. 5.24).

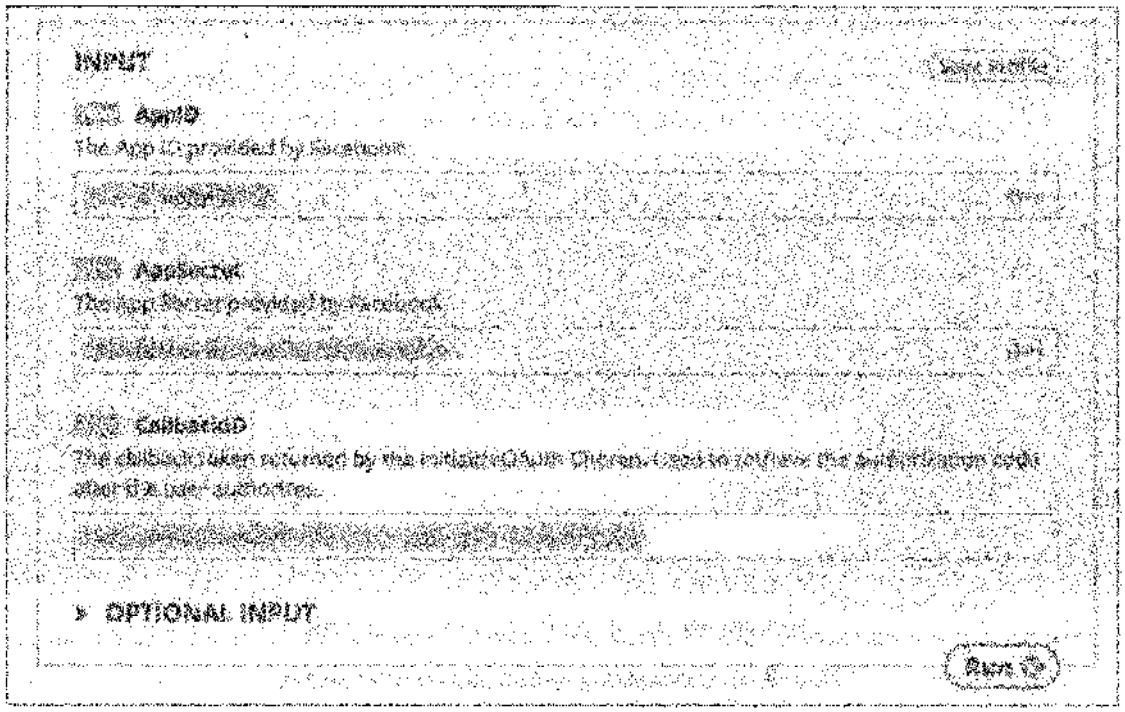


Рис. 5.23. Страница окончательной авторизации

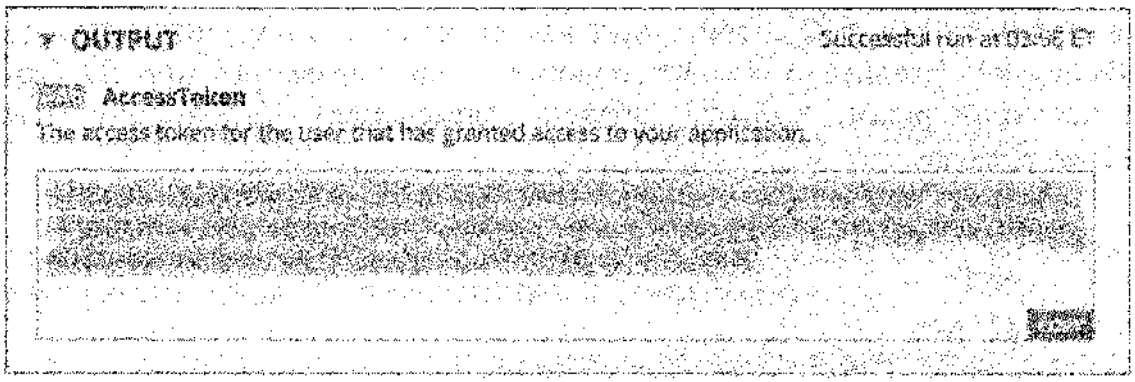


Рис. 5.24. Ваш токен доступа к приложению

12. Затем перейдите в библиотеку Temboo, которую собираетесь использовать. В данном случае — это библиотека для публикации записей в Фейсбуке: **https://temboo.com/library/Library/Facebook/Publishing/Post/**.
13. Введите там свой токен доступа и сообщение, которое хотите опубликовать на стене при помощи ESP8266 (рис. 5.25). В итоге все должно выглядеть так, как показано на рис. 5.26.
14. Сгенерируйте код для нового скетча. Вы можете работать с этим кодом или взять готовый код из папки ch5_Facebook сопровождающего книгу электронного архива (см. *приложение*). Как и в прошлый раз, изменим пару строк кода. Необходимо добавить библиотеку ESP8266WiFi:

```
#include <ESP8266WiFi.h>
#include <Temboo.h>
#include "TembooAccount.h"
```

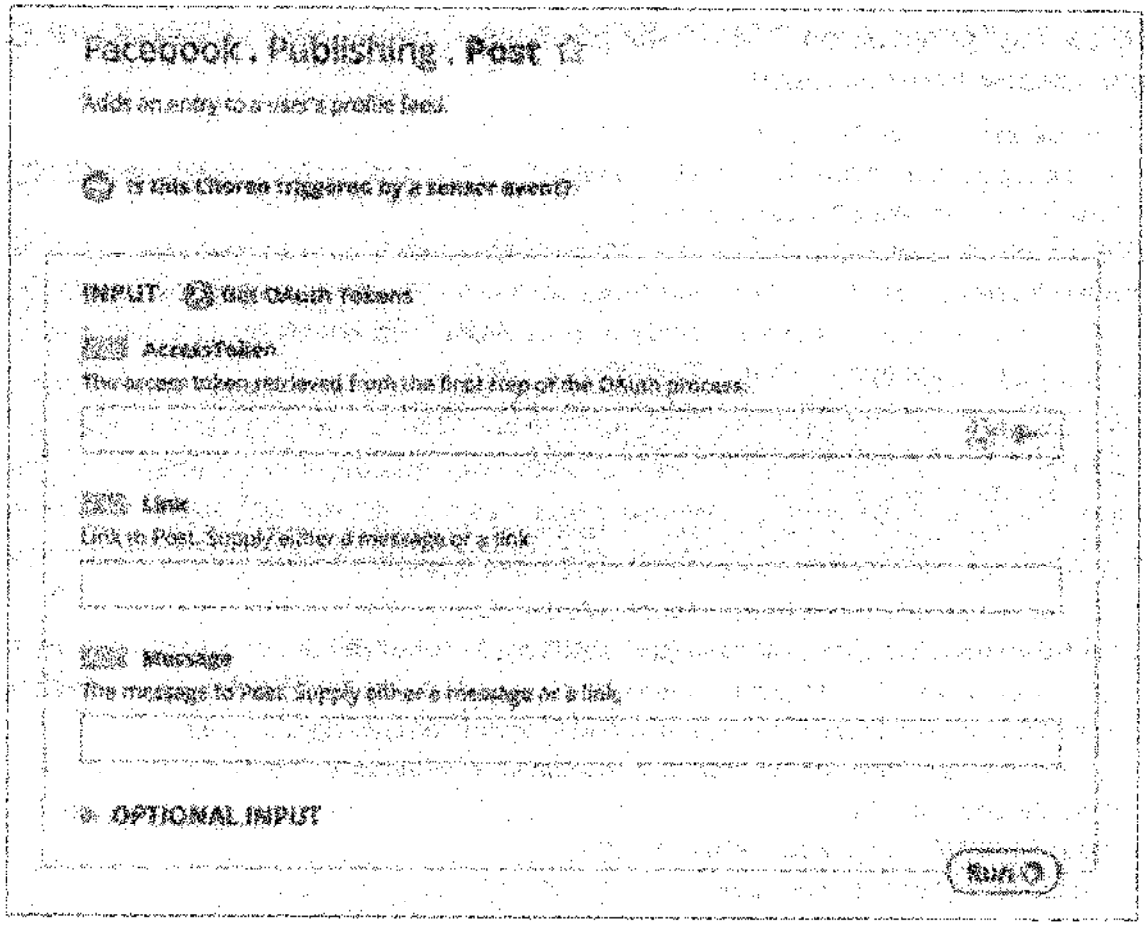


Рис. 5.25. Вводим токен доступа и текст сообщения

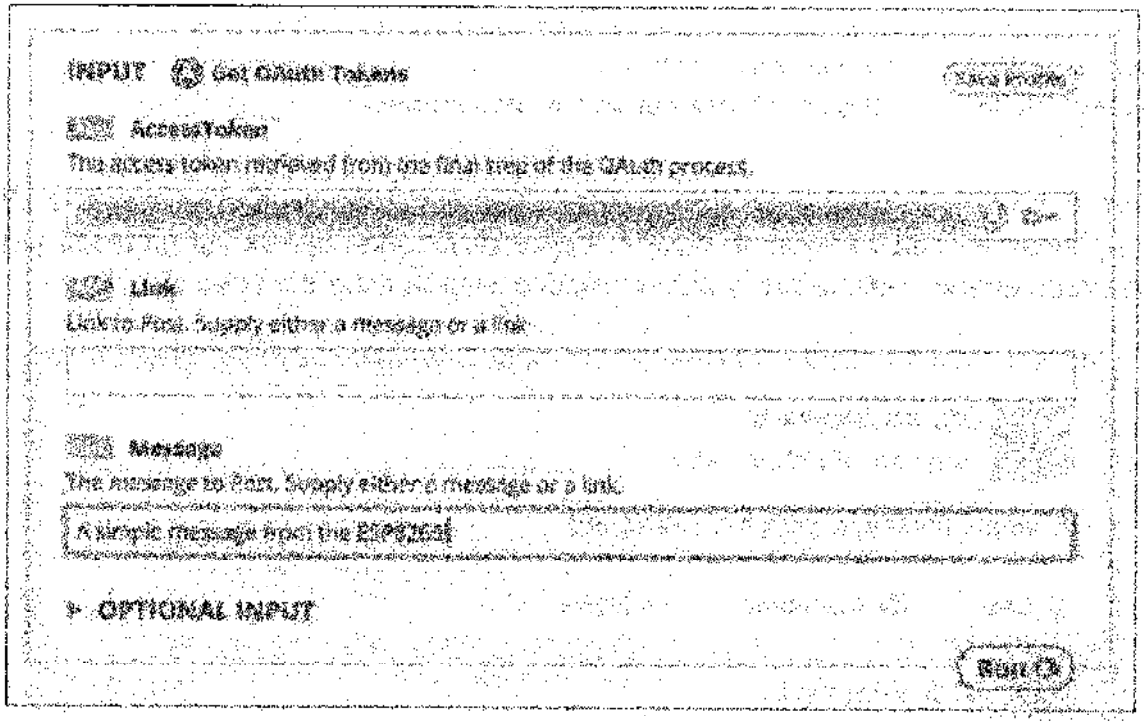


Рис. 5.26. Заполненная форма на странице библиотеки Temboo

15. Далее, внутри функции `loop()` мы размещаем сообщение на стене в Фейсбуке при каждом проходе цикла:

```
if (numRuns <= maxRuns) {
  Serial.println("Running Post - Run #" + String(numRuns++));
  TembooChoreo PostChoreo(client);
  // Вызов клиента Temboo
  PostChoreo.begin();
  // Права доступа аккаунта Temboo (см. файл Temboo.h)
  PostChoreo.setAccountName(TEMBOO_ACCOUNT);
  PostChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
  PostChoreo.setAppKey(TEMBOO_APP_KEY);
  // Входные данные Choreo
  String MessageValue = "A simple message from the ESP8266!";
  PostChoreo.addInput("Message", MessageValue);
  String AccessTokenValue = "accessToken";
  PostChoreo.addInput("AccessToken", AccessTokenValue);
  // Определяем Choreo для запуска
  PostChoreo.setChoreo("/Library/Facebook/Publishing/Post");
  // Запускаем Choreo; результат выводим в терминал
  PostChoreo.run();
  while (PostChoreo.available()) {
    char c = PostChoreo.read();
    Serial.print(c);
  }
  PostChoreo.close();
}

Serial.println("\nWaiting...\n");
delay(30000); // Пауза 30 секунд между повторами
}
```

16. Вот и пришло время испытать скетч! Загрузите прошивку в плату ESP8266 и проверьте свой аккаунт Фейсбук. Через некоторое время вы должны увидеть на стене новое сообщение, текст которого задали в скетче (рис. 5.27).

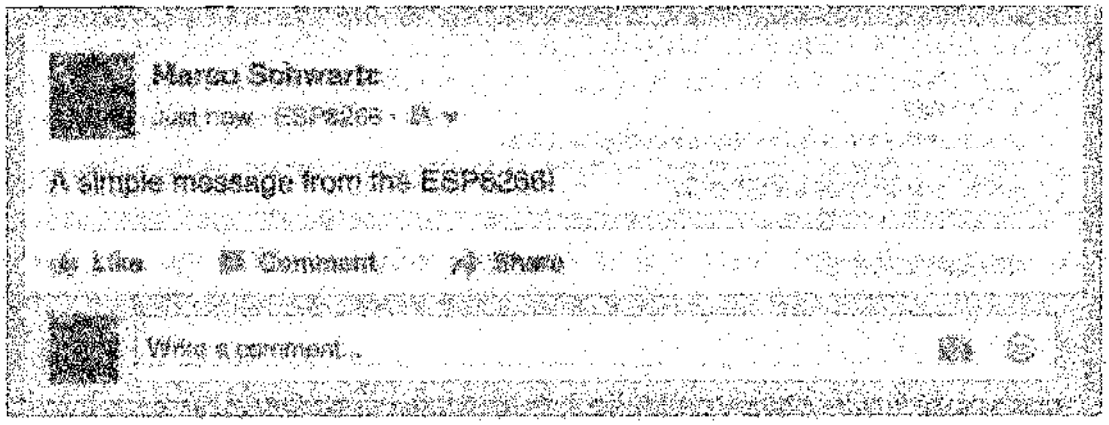


Рис. 5.27. Новое сообщение, отправленное через ESP8266

Теперь вы можете использовать Temboo для публикаций в Фейсбуке при помощи ESP8266!

Заключение

В этой главе мы научились взаимодействовать с веб-сервисами через ESP8266 — узнали, как отправить данные в Твиттер, как получить погодные данные из Yahoo и как взаимодействовать с Фейсбуком.

Я настоятельно рекомендую вам опробовать все библиотеки, предлагаемые Temboo. Их возможности почти безграничны. Например, вы можете завести в Фейсбуке страницу специально для своего дома и автоматически публиковать на ней данные о нем! Вы также можете использовать дополнительные данные, предоставляемые этими сервисами (например, о погоде) для развития собственных проектов.

В следующей главе мы займемся изучением другой важной составляющей Интернета вещей — общением устройств между собой.

6

Общение между устройствами

Из предыдущих глав мы узнали, как с помощью ESP8266 загружать в Сеть различные данные, а также как управлять устройствами удаленно из любой точки мира. Впрочем, эти проекты Интернета вещей имеют нечто общее — в какой-то момент им требуется ваше вмешательство: открытие окон для просмотра данных или нажатие на кнопки дистанционного управления.

В этой главе мы рассмотрим другую область Интернета вещей — *межмашинное взаимодействие* (M2M, Machine-to-Machine communications), то есть те случаи, когда вмешательство человека не требуется, и устройства для решения тех или иных задач общаются друг с другом напрямую.

Именно это мы и продемонстрируем здесь с помощью модулей ESP8266. Сначала мы выполним весьма простой проект, иллюстрирующий общие принципы M2M-взаимодействия, а затем используем полученные знания для разработки фотореле на основе двух плат ESP8266. Начинаем!

Оборудование и программное обеспечение

Для проектов этой главы потребуется следующее оборудование:

- прежде всего — это плата с микросхемой ESP8266. В основном я использую в этой книге модуль Adafruit Huzzah ESP8266, но любой другой модуль ESP8266 будет работать здесь ничуть не хуже;
- для первого проекта вам понадобится светодиод и резистор 330 Ом, кнопка и резистор 1 кОм;
- а для второго — реле, резистор 10 кОм и миниатюрный фотоэлемент;
- может понадобиться также модуль USB FTDI 3,3/5 В для программирования чипа ESP8266[1];
- беспаечная макетная плата и соединительные провода.

---

[1] При использовании платы NodeMCU этот модуль не понадобится. — *Прим. пер.*

Вот конкретный перечень использованных в проектах компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266 (2 шт.): **https://www.adafruit.com/products/2471**;
- модуль FTDI USB: **https://www.adafruit.com/products/284**;
- светодиод: **https://www.sparkfun.com/products/9590**;
- резистор 330 Ом: **https://www.sparkfun.com/products/8377**;
- реле: **https://www.sparkfun.com/products/10747**;
- фотоэлемент: **https://www.sparkfun.com/products/9088**;
- резистор 10 кОм: **https://www.sparkfun.com/products/8374**;
- миниатюрная кнопка: **https://www.sparkfun.com/products/97**;
- резистор 1 кОм: **https://www.sparkfun.com/products/8980**;
- макетная плата (2 шт.): **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

Если вы до сих пор не установили среду разработки Arduino IDE, сделайте это сейчас. Самую свежую версию среды можно скачать по адресу: **http://www.arduino.cc/en/Main/Software**.

Затем установите библиотеки aREST и PubSubClient, воспользовавшись для этого Менеджером библиотек Arduino IDE.

Чтобы микросхемы ESP8266 смогли общаться друг с другом, мы воспользуемся вебсервисом IFTTT. В этом сервисе вы можете создать наборы правил, согласно которым будут выполняться заданные действия при получении определенного *триггера* (сигнала о событии), что идеально подходит для межмашинного взаимодействия.

Если вы еще не создали учетную запись IFTTT, сделайте это сейчас на сайте **https://ifttt.com**. Добавьте также в свою учетную запись канал[2] **Maker**. Это гарантирует, что она будет доступна для наборов правил. Вы также получите ключ, который понадобится вам в этой главе позже.

Совет от переводчика

Чтобы найти ключ триггера, после создания канала перейдите в свой профиль, выберите опцию **Settings | Maker Webhooks settings** и нажмите на кнопку редактирования апплета (она имеет вид шестеренки). Вы увидите образец ссылки вида: **https://maker.ifttt.com/use/4nevuFo6sRig5bPidw1123589u7QAeDoLTgjMrGTqsf**. Длинный набор символов после слеша и есть ключ. Скопируйте и сохраните этот ключ — его нужно будет подставить в скетч, обрабатывающий нажатие кнопки.

---

[2] Теперь это называется не *каналами*, а *апплетами* (**Applets**). Найдите и подключите к своей учетной записи приложение **Maker Webhooks**. Его значок отличается от показанного в авторских иллюстрациях книги. — *Прим. пер.*

Простое межмашинное взаимодействие

В первом проекте этой главы мы рассмотрим простейший случай M2M-взаимодействия, когда один чип ESP8266 отправляет триггер события другому (разумеется, через облако), а тот в ответ переключает состояние светодиода. В качестве источника события мы используем кнопку, подключенную к первой плате ESP8266.

Подготовим сначала аппаратную часть проекта. У платы для управления светодиодом схема действительно очень проста: соедините светодиод последовательно с резистором, как мы это делали в предыдущих главах. Резистор подключите к выводу GPIO5 (D1) платы ESP8266, а второй вывод светодиода — к общему проводу (рис. 6.1).

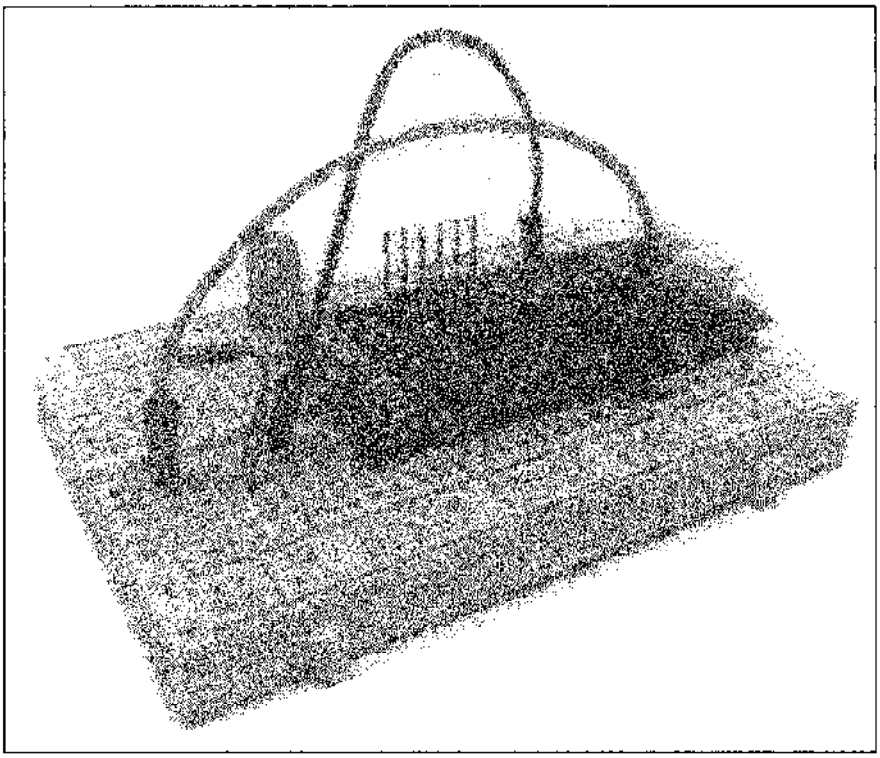


Рис. 6.1. Монтажная схема платы управления светодиодом

На макетную плату, которая будет обслуживать кнопку, сначала поместите модуль ESP8266 и саму кнопку. Затем подключите к одному контакту кнопки резистор 1 кОм и соедините второй вывод резистора с общим проводом. С этим же контактом кнопки соедините вывод GPIO5 (D1) платы ESP8266. Второй контакт кнопки соедините с линией питания VCC 3,3 В (рис. 6.2).

Займемся теперь подготовкой плат с тем, чтобы они смогли установить связь между собой. Как и отмечалось ранее, поможет платам общаться друг с другом сервис IFTTT.

Запрограммируем сначала плату со светодиодом, чтобы она могла принимать команды из облака. Для этого мы снова воспользуемся фреймворком aREST:

1. Скетч начинает работу с импорта необходимых библиотек:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
```

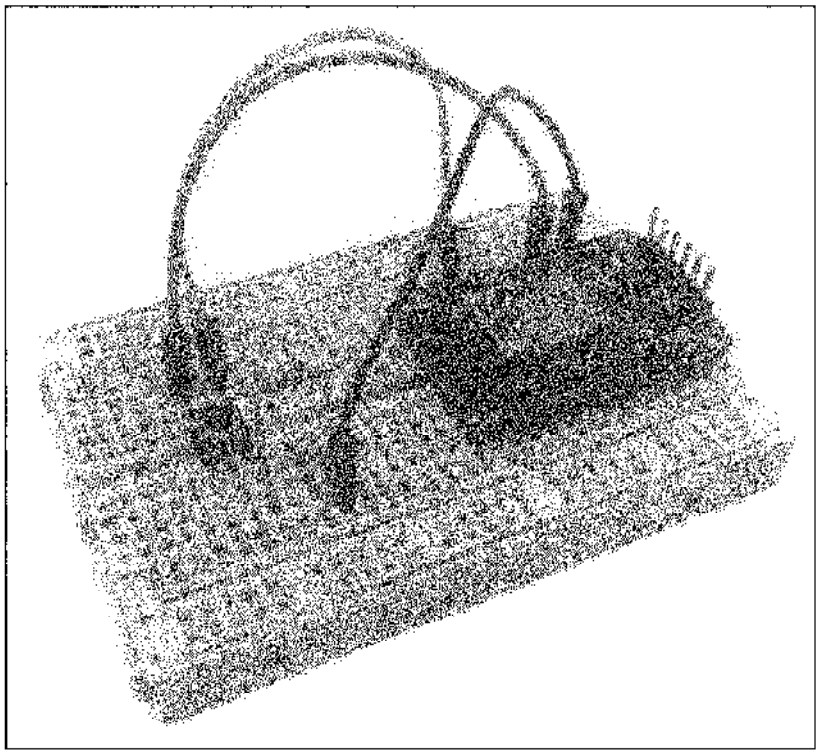


Рис. 6.2. Монтажная схема платы для генерации события «нажатие кнопки»

2. Затем мы создаем программного клиента для подключения к облаку:

```
WiFiClient espClient;
PubSubClient client(espClient);
```

3. Инициализируем библиотеку aREST:

```
aREST rest = aREST(client);
```

4. Присваиваем устройству уникальный идентификатор:

```
char* device_id = "01e47f";
```

Следует также ввести данные для подключения к своей сети Wi-Fi:

```
const char* ssid = "wifi-name";
const char* password = "wifi-password";
```

5. Объявим логическую переменную для хранения текущего состояния светодиода:

```
bool ledState;
```

Объявим также функцию, переключающую светодиод (она будет реализована позже), и функцию обратного вызова:

```
int toggle(String command);
// Функция обратного вызова
void callback(char* topic, byte* payload, unsigned int length);
```

6. В функции `setup()` скетча мы подключаем функцию переключения светодиода[3] к событию aREST и подключаем чип к сети Wi-Fi:

[3] В качестве функции обратного вызова по событию. — *Прим. пер.*

```
void setup(void)
{
  // Инициализация последовательного порта
  Serial.begin(115200);
  // Подключение функции обратного вызова
  client.setCallback(callback);
  // Задаем имя и идентификатор устройства
  rest.set_id(device_id);
  rest.set_name("led");
  // Задаем имя функции обратного вызова
  rest.function("toggle", toggle);
  // Текущее состояние светодиода
  ledState = false;
  // Подключение к Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  // Pin 5 as output
  pinMode(5, OUTPUT);
}
```

7. В функции `loop()` скетча мы просто поддерживаем соединение с облачной платформой aREST:

```
void loop() {
  // Подключение к облаку
  rest.handle(client);
}
```

8. Наконец, вам нужно реализовать функцию переключения светодиода. Мы инвертируем состояние переменной светодиода при каждом вызове функции и применяем новое состояние к выводу светодиода:

```
// Переключение состояния светодиода
int toggle(String command) {
  ledState = !ledState;
  digitalWrite(5, ledState);
  return 1;
}
```

Исправьте в коде параметры доступа и загрузите прошивку в плату, к которой подключен светодиод.

Готовый к использованию файл программы находится в папке ch6_LED_board сопровождающего книгу электронного архива (см. *приложение*).

Теперь разберемся, как запрограммировать плату с кнопкой, при нажатии которой будет генерироваться событие IFTTT:

1. Начинаем с подключения библиотеки ESP8266:

```
#include <ESP8266WiFi.h>
```

2. Определяем параметры доступа к сети Wi-Fi:

```
const char* ssid = "wifi-name";
const char* password = "wifi-pass";
```

3. В этой части кода надо вставить ключ IFTTT:

```
const char* host = "maker.ifttt.com";
const char* eventName = "button_pressed";
const char* key = "ifttt-key";
```

4. В функции `setup()` этого скетча мы подключаем ESP8266 к сети Wi-Fi:

```
void setup() {
  Serial.begin(115200);
  delay(10);
  // Подключаемся к сети Wi-Fi
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  // Вывод GPIO5 как вход
  pinMode(5, INPUT);
}
```

5. В функции `loop()` проверяем, нажата ли кнопка:

```
if (digitalRead(5)) {
```

6. В случае нажатия посылаем сообщение в IFTTT. Сначала соединяемся с сервером и создаем запрос:

```
// Используем WiFiClient для TCP-подключения
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
  Serial.println("connection failed");
```

```
  return;
}
// Создаем URI запроса
String url = "/trigger/";
url += eventName;
url += "/with/key/";
url += key;
Serial.print("Requesting URL: ");
Serial.println(url);
```

7. Когда подключение установлено, отправляем запрос в IFTTT:

```
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
             "Host: " + host + "\r\n" +
             "Connection: close\r\n\r\n");
int timeout = millis() + 5000;
while (client.available() == 0) {
  if (timeout - millis() < 0) {
    Serial.println(">>> Client Timeout !");
    client.stop();
    return;
  }
}
```

8. В завершение запроса читаем данные, поступившие от IFTTT, и выводим их в терминал:

```
while (client.available()) {
  String line = client.readStringUntil('\r');
  Serial.print(line);
}
Serial.println();
Serial.println("closing connection");
}
```

Вы можете скачать исходный код скетча в репозитории GitHub по адресу: **https://github.com/openhomeautomation/iot-esp8266-packt**. Готовый к использованию файл программы находится также в папке ch6_BUTTON_board сопровождающего книгу электронного архива (см. *приложение*).

9. Исправьте в коде необходимые параметры и загрузите прошивку в плату, к которой подключена кнопка.

   Сейчас обе наших платы еще не общаются друг с другом. Поэтому мы отправимся на сайт IFTTT, чтобы создать *правило взаимодействия* (recipe) и соединить эти платы.

10. На сайте IFTTT создайте новое правило и выберите канал **Maker** в качестве триггера (рис. 6.3).

11. Выберите тип триггера **Receive a web request** (рис. 6.4).

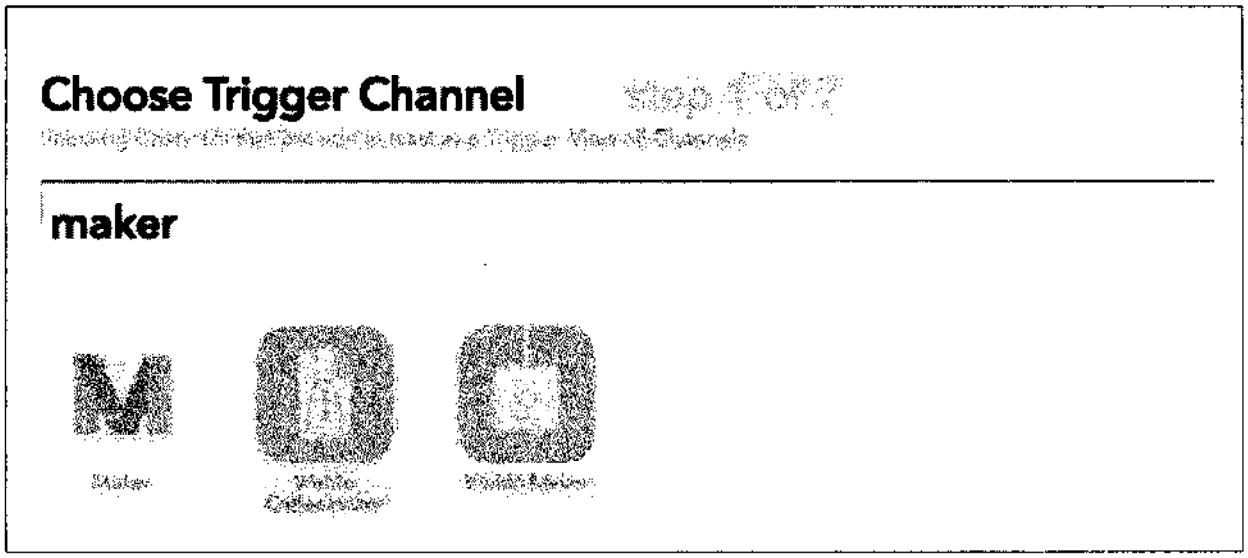


Рис. 6.3. Создаем канал и правило на сайте IFTTT

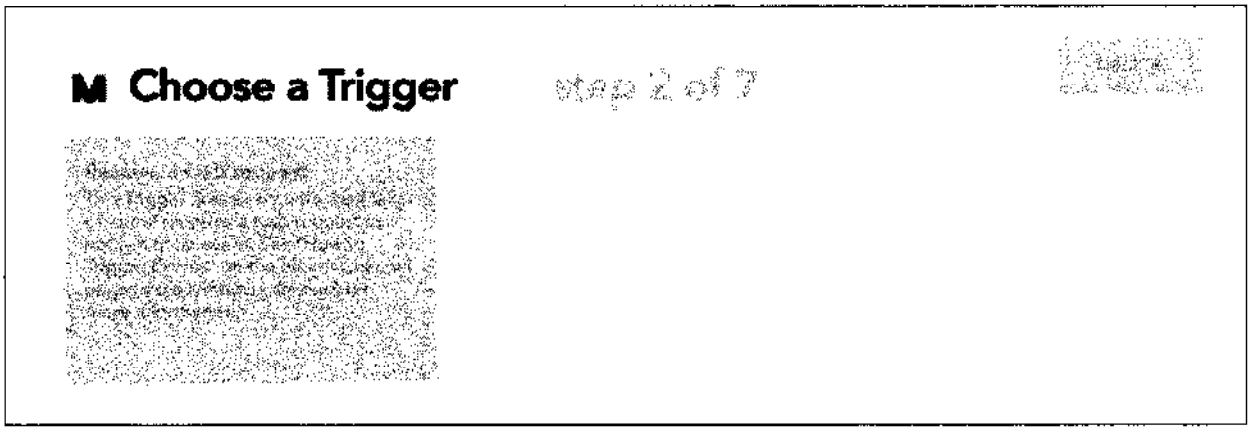


Рис. 6.4. Выбираем триггер IFTTT

12. В поле события введите: `button_pressed` (кнопка нажата) — это также указано в п. 3 скетча (рис. 6.5).
13. В качестве канала действия также выберите канал **Maker** (рис. 6.6).

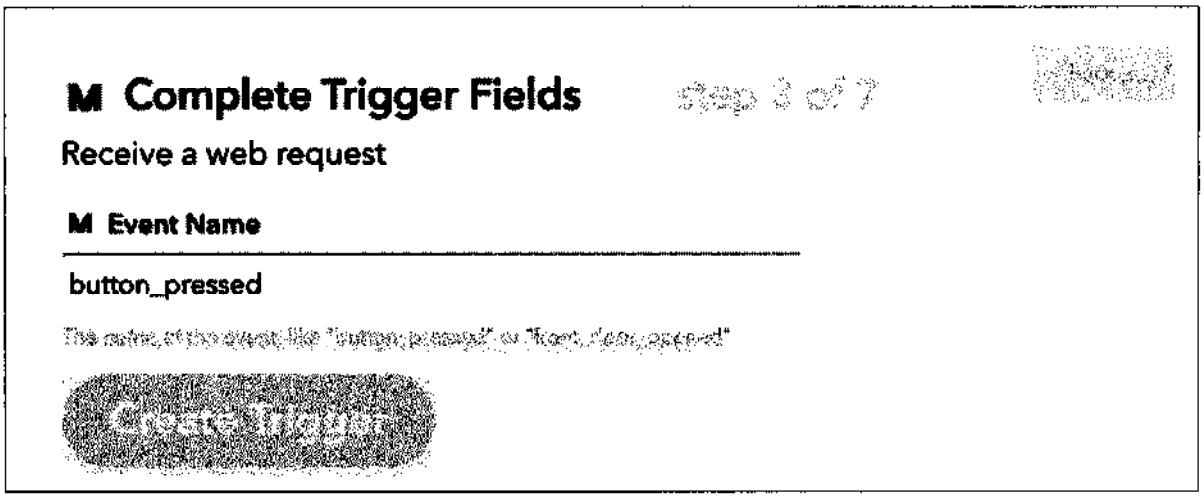


Рис. 6.5. Вводим название события

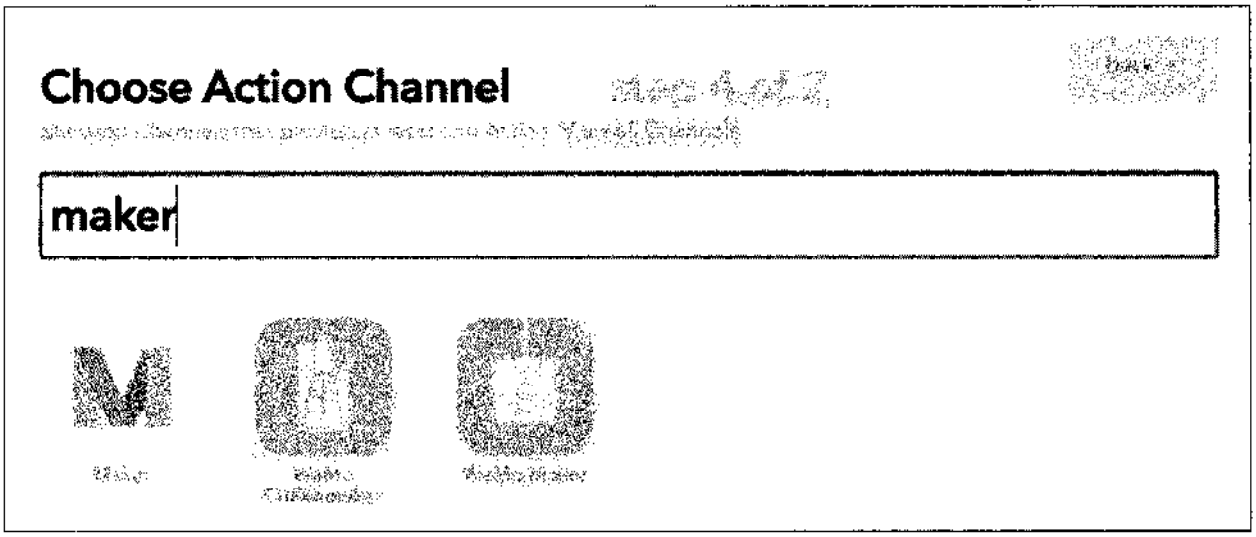


Рис. 6.6. Выбираем канал действия

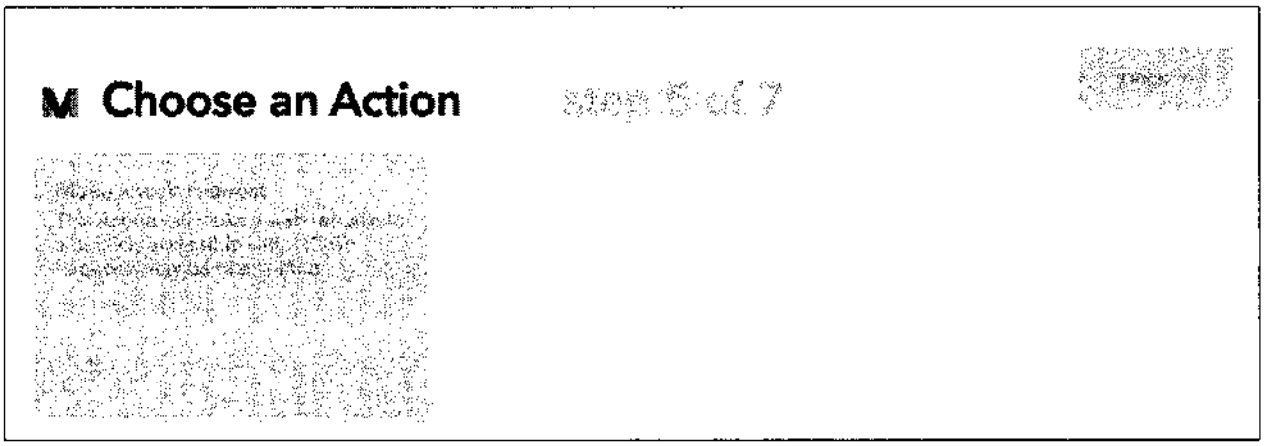


Рис. 6.7. Выбор действия по событию

14. Выберите действие **Make a web request** (рис. 6.7).
15. Для передачи параметров действия мы, как правило, должны вызвать функцию на созданном нами устройстве. Поэтому просто введите параметры, как показано на рис. 6.8.
16. Разумеется, вы должны изменить ID устройства в поле **URL** и указать там идентификатор устройства, который задали в коде прошивки. После этого создайте правило события, которое будет храниться внутри вашей учетной записи IFTTT.
17. Теперь можно приступить к окончательному тестированию проекта, потому что IFTTT уже поддерживает связь между двумя нашими платами. Стоит нажать на кнопку, и вскоре вы должны увидеть, что светодиод зажегся. Учтите, что задержка включения может достигать 2–3 секунд, поскольку информация сначала должна пройти через сервер IFTTT. Можете нажать кнопку снова для выключения светодиода. Вот вы и создали свой первый проект межмашинного взаимодействия!

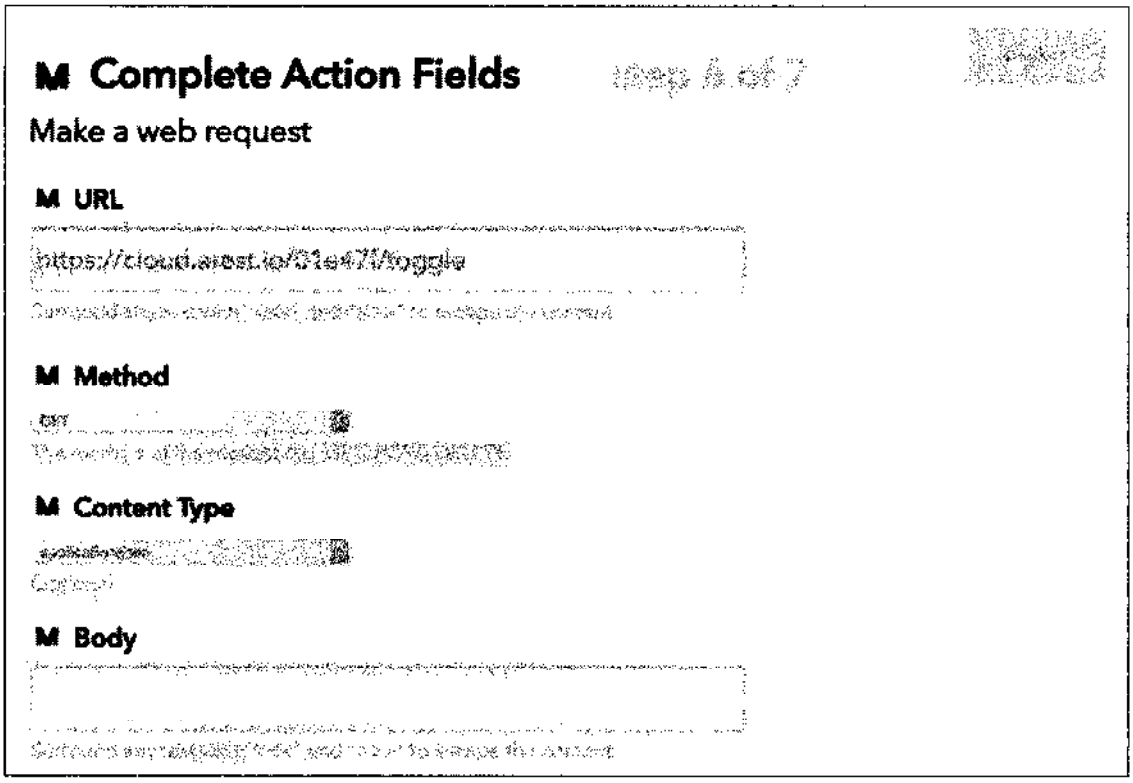


Рис. 6.8. Форма для ввода параметров действия в ответ на событие

Создаем беспроводное фотореле

Все, что мы узнали о создании систем межмашинного взаимодействия на основе ESP8266, мы используем в простом проекте, где вам даже не придется нажимать на кнопку. Мы собираемся разработать фотореле — устройство, включающее реле, подключенное к одной плате ESP8266, в зависимости от уровня освещенности, измеренного другой платой ESP8266. Таким образом, мы смоделируем часть системы домашней автоматики, которая включает освещение, когда снаружи стемнело.

Прежде всего, соберем обе платы. И начнем с платы, которая управляет реле, потому что это наиболее простая часть сборки (рис. 6.9):

1. Вывод VCC реле соедините с выводом VCC платы ESP8266.
2. Соедините выводы GND.
3. Вывод реле SIG или IN соедините с выводом GPIO5 (D1) ESP8266.

Соберем затем плату, которая содержит фотоэлемент. Установите на плату модуль ESP8266 и фотоэлемент, последовательно соединенный с резистором[4] 10 кОм. Общую точку подключения резистора и фотоэлемента соедините с аналоговым входом платы ESP8266 (он обозначен, как A или ADC). Второй вывод фотоэлемента соедините с линией VCC, а второй вывод резистора — с GND (рис. 6.10)[5].

[4] Выводы этого фотоэлемента равнозначны, поэтому полярность не имеет значения. — *Прим. пер.*

[5] Автор показал на рис. 6.10 два последовательно соединенных резистора, но это не соответствует реальной схеме и тексту книги. — *Прим. пер.*

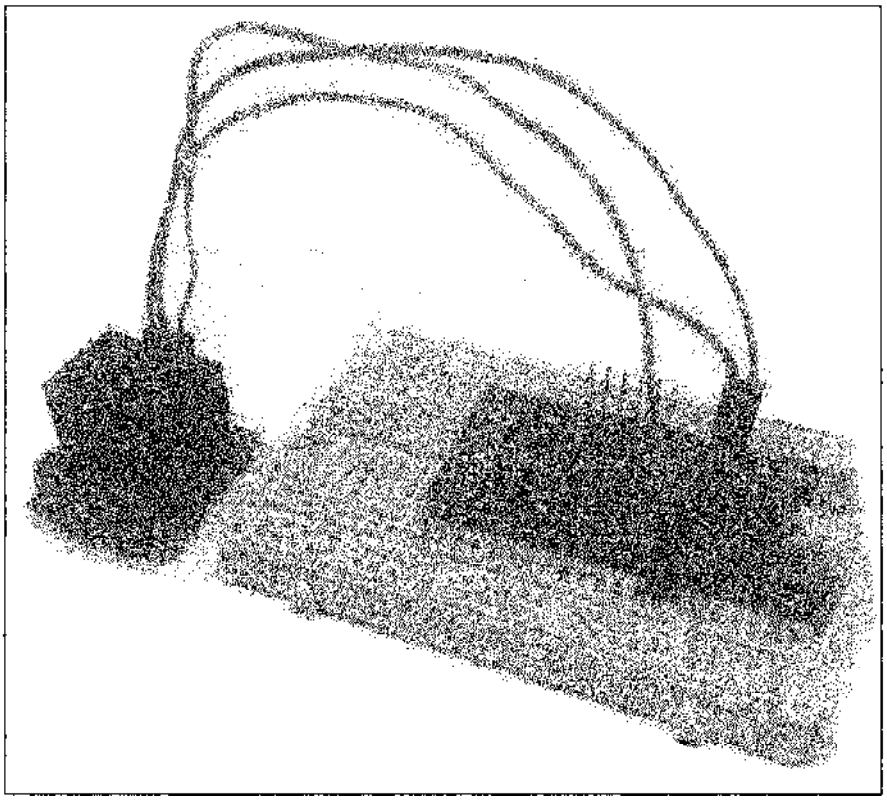


Рис. 6.9. Монтажная схема исполнительной части

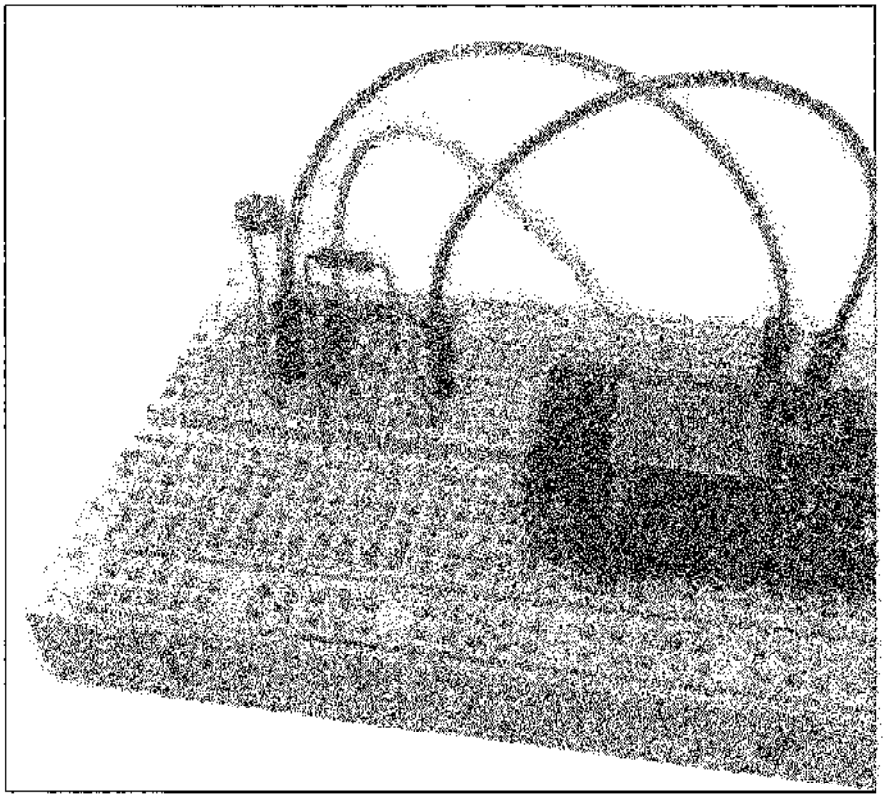


Рис. 6.10. Монтажная схема управляющей части

Разберемся теперь, как запрограммировать эти платы:

- для платы с реле вы можете просто взять скетч управления светодиодом из предыдущего проекта, т. к. мы снова воспользуемся облачным сервером aREST;
- плата с фотоэлементом очень похожа на плату с кнопкой из первого проекта, поэтому далее мы обсудим только основные различия.

Итак, сначала мы задаем переменную, значение которой соответствует *высокому* или *низкому* уровню освещенности в данный момент. По умолчанию мы считаем, что освещенность *низкая*, присвоив переменной логическое значение `false`:

```
bool lightLevel = false;
```

В функции `loop()` скетча сначала измеряем состояние аналогового вывода и выводим результат в монитор последовательного порта:

```
Serial.print("Light level: ");
Serial.println(analogRead(A0));
```

Затем проверяем, находится ли уровень освещенности ниже заданного порога, а также был ли предыдущий уровень освещенности *высоким*. Учтите, что порог здесь задан произвольно — лишь для демонстрации M2M-взаимодействия между двумя платами. Если мы действительно попадаем в указанные условия, то генерируем событие `light_lewel_low`:

```
if (analogRead(A0) < 700 && lightLevel) {
  lightLevel = false;
  makeRequest("light_level_low");
}
```

Наоборот, если текущий уровень освещенности *высокий*, а предыдущий уровень был *низким*, мы отправляем в IFTTT событие `light_level_high`:

```
if (analogRead(A0) > 800 && !lightLevel) {
  lightLevel = true;
  makeRequest("light_level_high");
}
```

Теперь вы можете запрограммировать плату с фотоэлементом этим кодом.

Исходный код скетча вы можете скачать в репозитории GitHub по адресу: **https://github.com/openhomeautomation/iot-esp8266-packt**. Готовый к использованию файл программы находится также в папке ch6_FOTO_board сопровождающего книгу электронного архива (см. *приложение*).

Далее мы вновь должны создать наборы правил IFTTT для связи двух плат — войдите на сайт IFTTT и создайте новый набор правил. Используйте канал **Maker** как триггер и установите имя `light_level_low`, определенное в скетче (рис. 6.11).

Снова используйте канал **Maker** и создайте веб-запрос с параметрами, как показано на рис. 6.12.

Здесь мы устанавливаем вывод исполнительной платы GPIO5 в *высокий* уровень, потому что хотим включить реле, когда освещенность становится *низкой*. Разумеется,

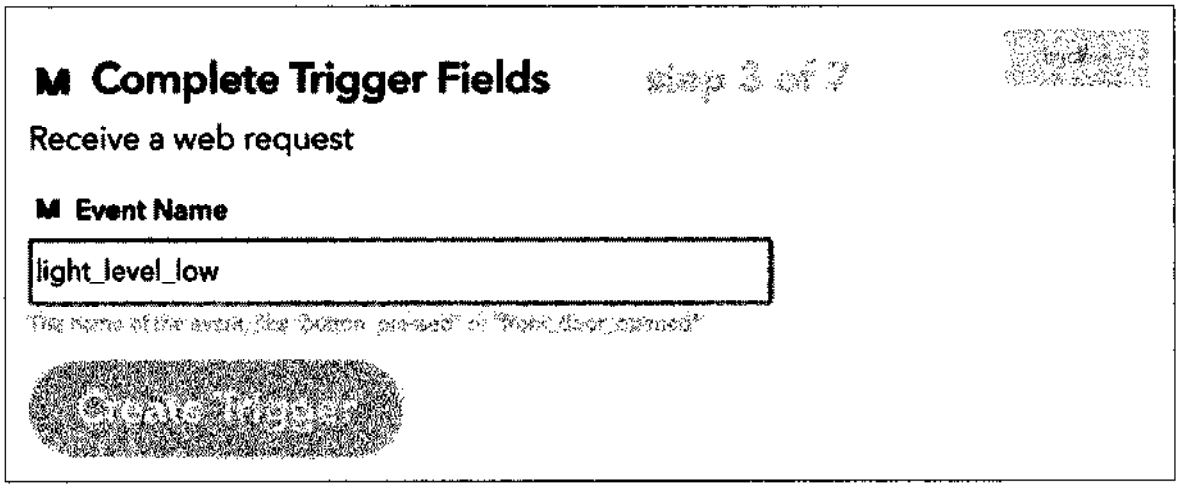


Рис. 6.11. Создаем новый триггер IFTTT для события низкой освещенности

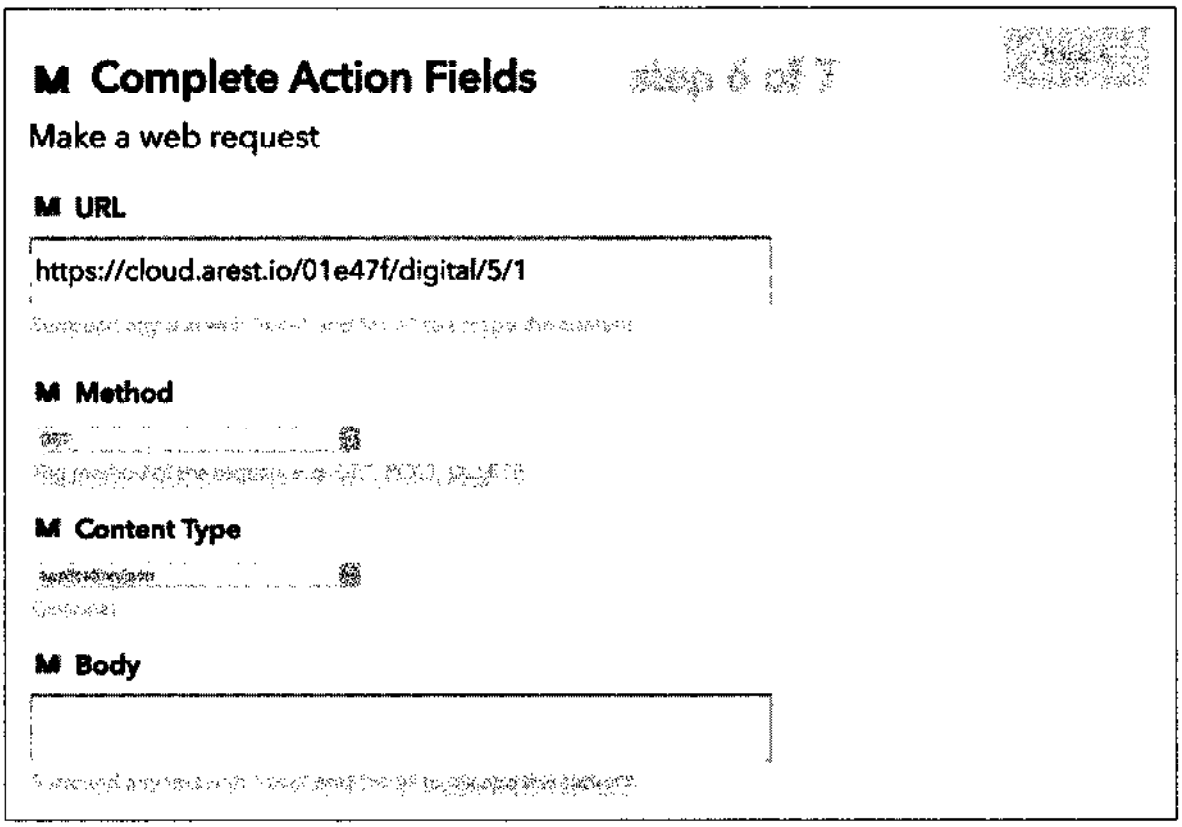


Рис. 6.12. Форма для ввода параметров действия в ответ на событие `light_level_low`

в поле **URL** вы должны указать идентификатор устройства, который задали в коде прошивки.

Сохраните правило и проделайте приблизительно то же самое для другого сценария, создав следующее правило. На этот раз в качестве события триггера укажите `light_level_high` (рис. 6.13).

Для создания действия заполните аналогичную форму, но в этот раз введите `0` в конце строки, — поскольку мы хотим, чтобы лампы погасли, если уровень освещенности вновь *высокий* (рис. 6.14).

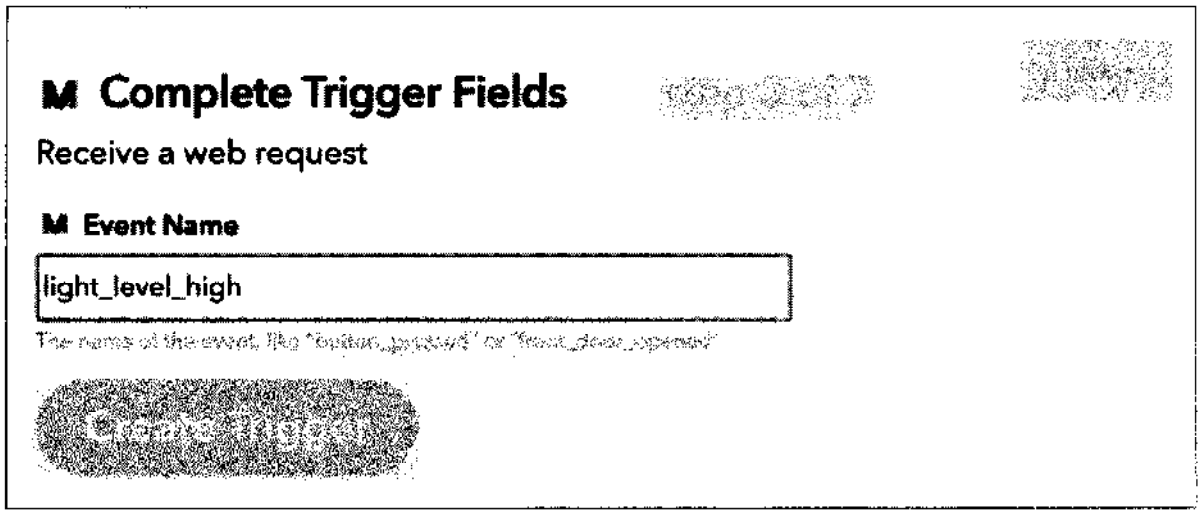


Рис. 6.13. Создаем новый триггер IFTTT для события высокой освещенности

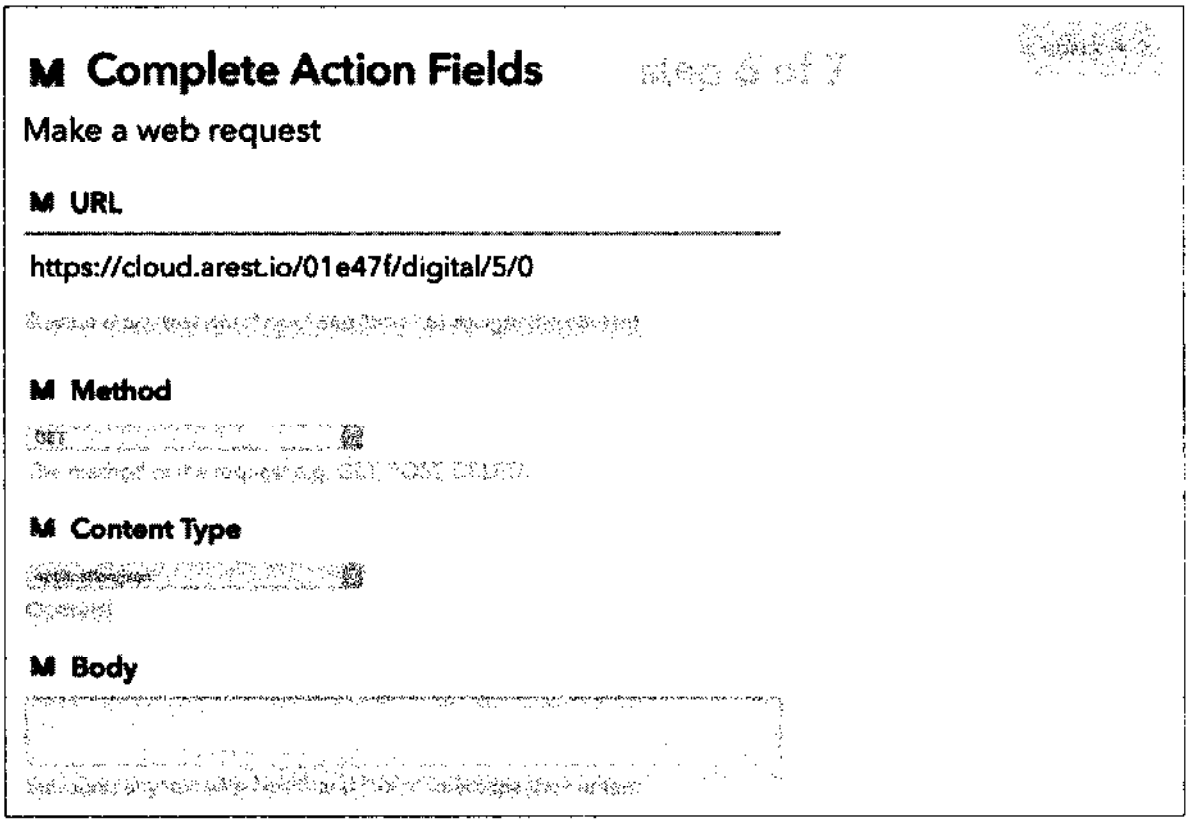


Рис. 6.14. Форма для ввода параметров действия в ответ на событие `light_level_high`

На панели IFTTT вы должны видеть два активных блока правил (рис. 6.15).

Теперь вы можете протестировать проект! Чтобы симулировать темноту, достаточно накрыть фотоэлемент ладонью, — спустя 2–3 секунды на второй плате должно включиться реле. Если вы уберете руку, реле снова отключится.

Отметим, что обе платы могут находиться в совершенно разных местах, потому что M2M-взаимодействие происходит через облачный сервис. Например, можно разместить одну плату где-то снаружи дома, а вторую — внутри, даже если они подключены к разным сетям Wi-Fi.

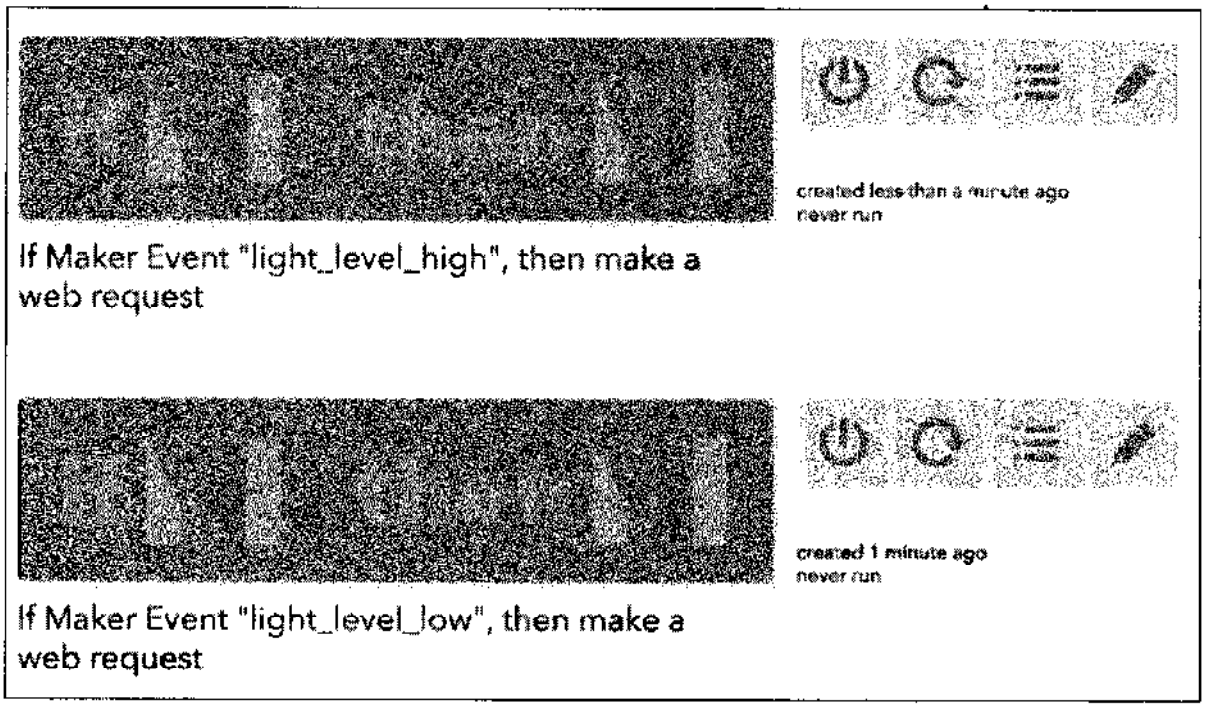


Рис. 6.15. Панель сервиса IFTTT с двумя наборами правил

Заключение

Давайте подведем итоги достигнутого в этой главе. Мы узнали, как реализовать M2M-взаимодействие при помощи ESP8266, и проиллюстрировали это, создав два простых проекта, использующих сервис IFTTT для организации связи между двумя платами ESP8266.

Теперь вы, несомненно, сможете применить полученные знания для разработки собственных проектов межмашинного взаимодействия. M2M — это очень обширная тема. На ее основе можно реализовать множество интересных разработок. Например, вы могли бы построить законченную систему безопасности, основанную на облачном сервисе, где каждый модуль ESP8266 взаимодействует со всеми остальными, и каждый оборудован датчиком движения. Кое-что из этого мы рассмотрим далее в книге.

В следующей главе мы снова обратимся к сервису IFTTT, но для другой цели — отправки автоматических уведомлений из ваших проектов на ESP8266.

7

Отправка уведомлений

В этой главе мы узнаем, как реализовать на ESP8266 очень важный компонент Интернета вещей — *уведомления*. Устройства Интернета вещей отправляют пользователю уведомления, если происходит что-то достойное внимания, либо с заданным интервалом, — например, сообщают ему какие-либо данные.

Мы рассмотрим в этой главе три сценария. Сначала мы научимся отправлять из ESP8266 простые уведомления на электронную почту. Затем разберемся, как заставить ESP8266 отправлять текстовые сообщения на ваш смартфон. И наконец, узнаем, как отправлять предупреждения с помощью push-уведомлений.

Оборудование и программное обеспечение

Для проектов этой главы потребуется следующее оборудование:

- прежде всего — это плата с микросхемой ESP8266. В основном я использую в этой книге модуль Adafruit Huzzah ESP8266, но любой другой модуль ESP8266 будет работать здесь ничуть не хуже;
- простой датчик влажности и температуры DHT11 — он задействован здесь в двух из трех сценариев. Впрочем, вы можете применить любой другой датчик или использовать произвольные данные. Наша цель — научиться отправлять уведомления;
- может понадобиться также модуль USB FTDI 3,3/5 В для программирования чипа ESP8266[1];
- беспаечная макетная плата и соединительные провода.

---

[1] При использовании платы NodeMCU этот модуль не понадобится. — *Прим. пер.*

Вот конкретный перечень использованных в проектах компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266: **https://www.adafruit.com/products/2471**;
- датчик DHT11: **https://www.adafruit.com/products/386**;
- модуль FTDI USB: **https://www.adafruit.com/products/284**;
- макетная плата: **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

Если вы до сих пор не установили среду разработки Arduino IDE, сделайте это сейчас. Самую свежую версию среды можно скачать по адресу: **http://www.arduino.cc/en/Main/Software**.

Вам также понадобится учетная запись сервиса IFTTT, который мы задействуем во всех проектах этой главы. IFTTT — это замечательный веб-сервис, способный связать между собой два других веб-сервиса при помощи наборов правил, которые активируются *триггерами* (событиями) *ЕСЛИ* и запускают ответные действия (события *ТО*).

Если вы еще не создали учетную запись IFTTT, сделайте это сейчас на сайте **https://ifttt.com**. Как можно видеть, учетная запись создается бесплатно (рис. 7.1).

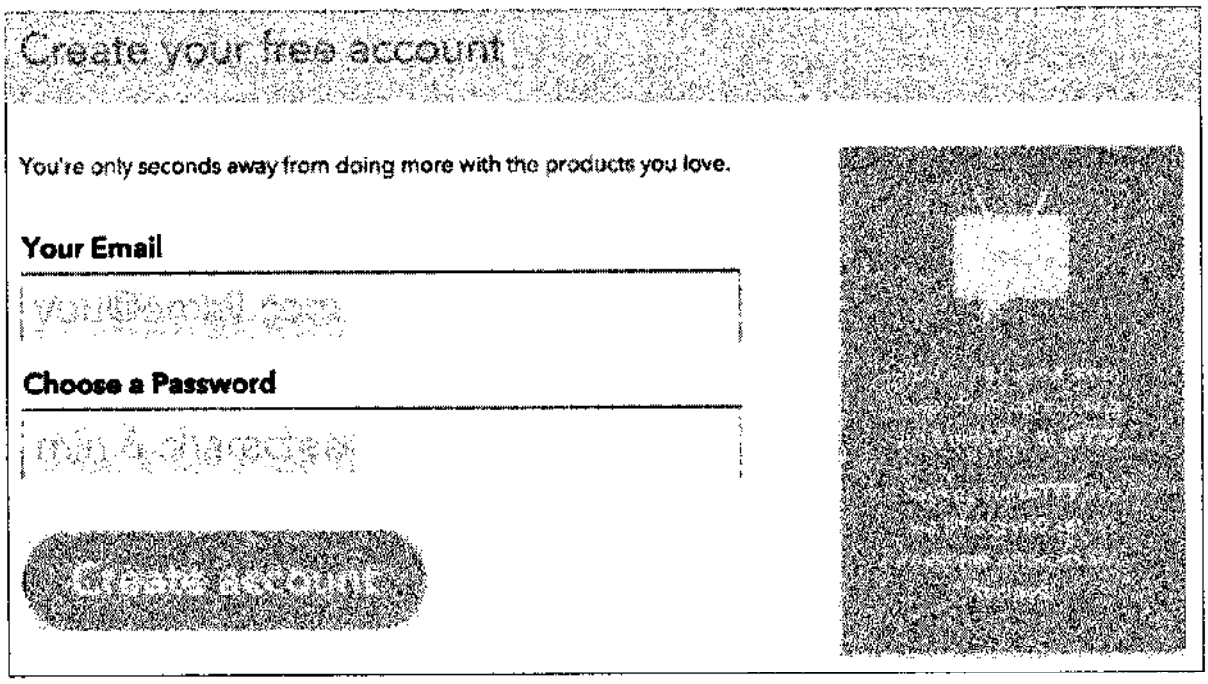


Рис. 7.1. Создание бесплатной учетной записи IFTTT

Схема соединений

Схему, используемую для проектов этой главы, вы уже собирали ранее, поэтому за описанием процесса ее сборки я отсылаю вас к *главе 5*. Результат сборки должен выглядеть так, как показано на рис. 7.2.

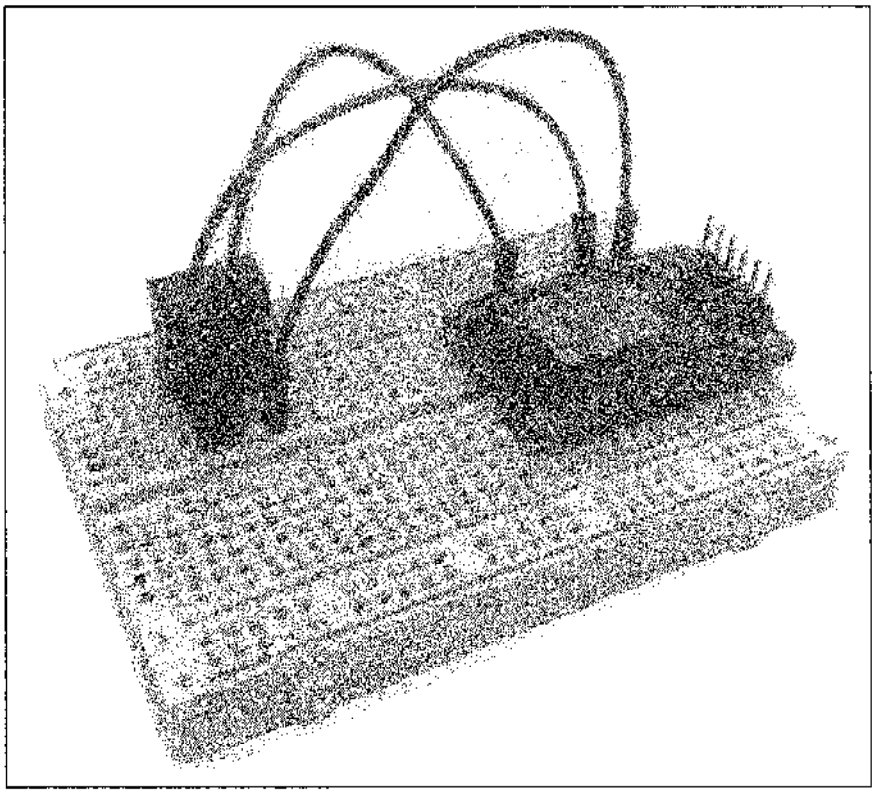


Рис. 7.2. Монтажная схема для проектов этой главы

Отправка уведомлений по электронной почте

Настало время создать первый проект — отправку уведомлений по электронной почте. Наш первый подход к уведомлениям заключается в отправке сообщений по электронной почте на указанный вами адрес с заданным интервалом.

Чтобы подключить ESP8266 к сервису IFTTT, требуется доступный в IFTTT канал[2], который называется **Maker**. Для этого сначала перейдите на вкладку **Channels** и найдите этот канал (рис. 7.3). Открыв канал в окне браузера, соедините его с вашей учетной записью IFTTT (рис. 7.4). Теперь вы можете получить секретный ключ канала **Maker** (рис. 7.5).

Совет от переводчика

Чтобы найти ключ триггера, после создания канала перейдите в свой профиль, выберите опцию **Settings | Maker Webhooks settings** и нажмите кнопку редактирования апплета (она имеет вид шестеренки). Вы увидите ссылку вида: **https://maker.ifttt.com/use/4nsvuFo6sRig5bPldw1123589u7QAeDoL TgjMrGTqsf**. Длинный набор символов после слеша и есть ключ.

Скопируйте и сохраните этот ключ — он вам понадобится чуть позже. Причем для всех каналов достаточно получить такой ключ только один раз.

---

[2] Сейчас это называется не *каналами*, а *апплетами* (**Applets**). Найдите и подключите к своей учетной записи приложение **Maker Webhooks**. Его значок отличается от показанного в авторских иллюстрациях книги. — *Прим. пер.*

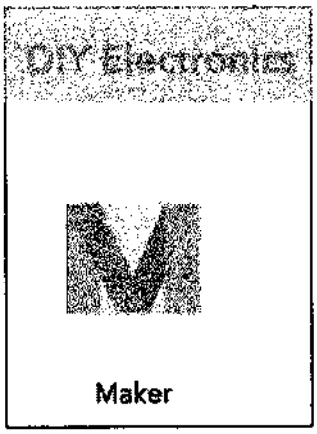


Рис. 7.3. Логотип канала **Maker**

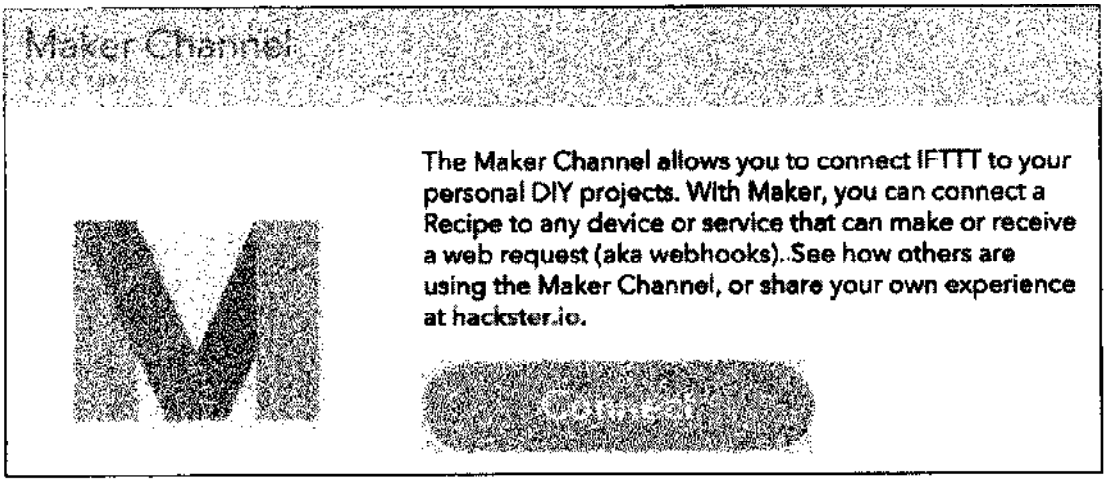


Рис. 7.4. Подключаем канал **Maker** к своей учетной записи IFTTT

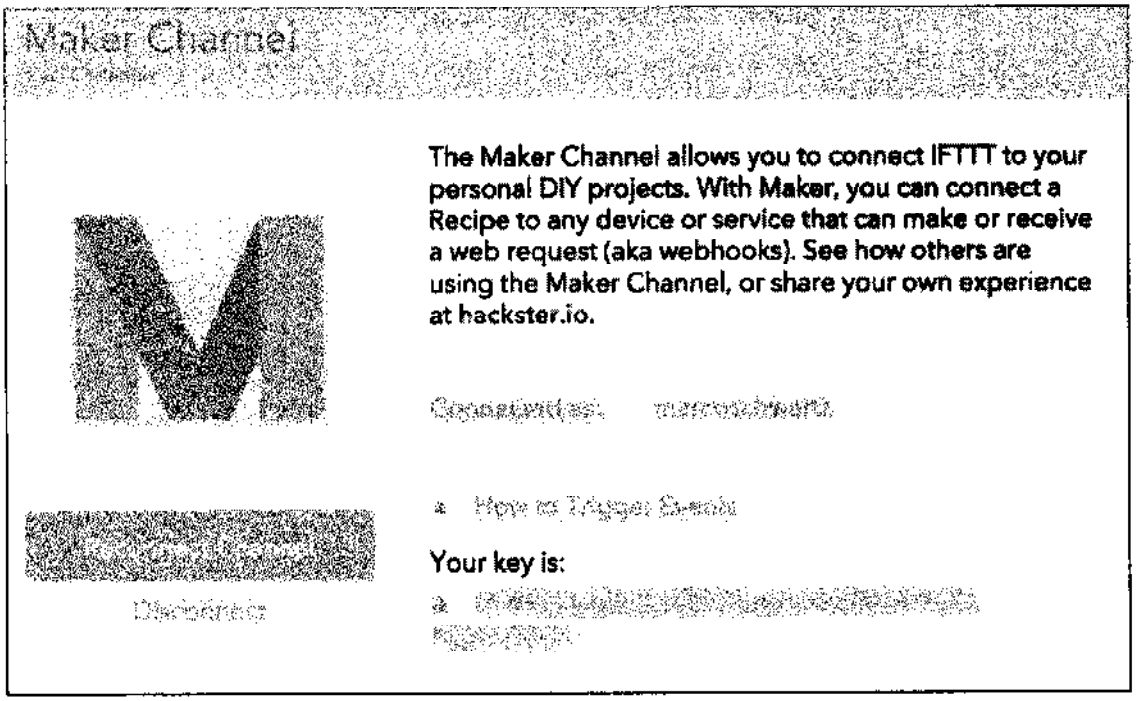


Рис. 7.5. Секретный ключ канала для подключения извне

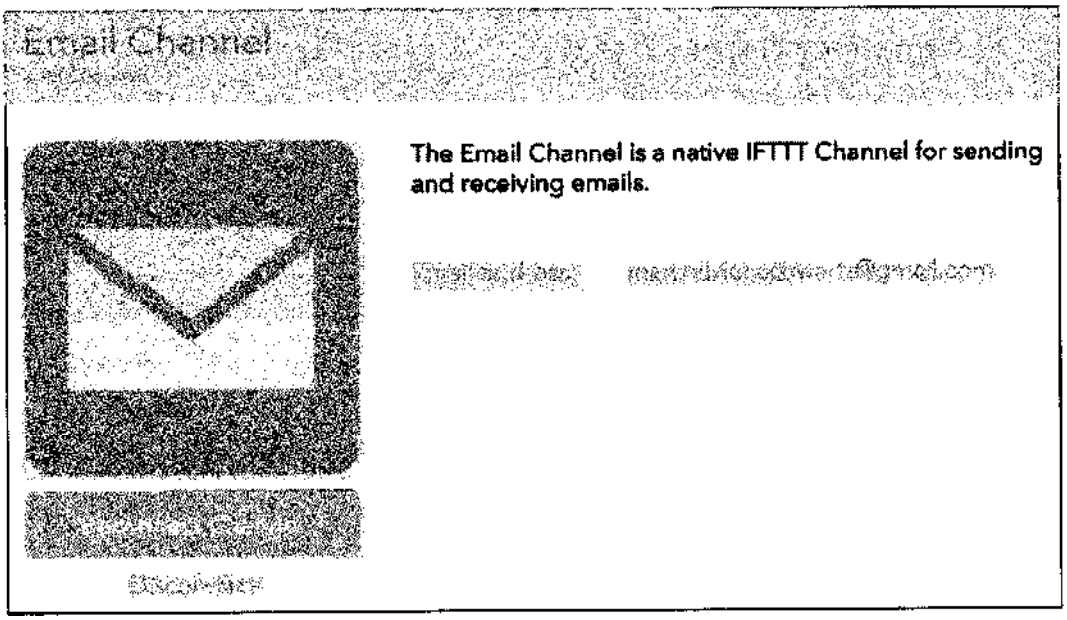


Рис. 7.6. Канал для работы с сообщениями электронной почты

Создадим далее следующий канал — **Email** (рис. 7.6).

Введите в этом окне адрес электронной почты, на который хотите получать тестовые сообщения от сервера IFTTT. Рекомендуется использовать только собственный адрес — в противном случае это будет расценено как спам.

Создайте далее правило, которое свяжет канал **Maker** с отправкой почты. Для этого просто нажмите кнопку **Create a Recipe**[3] на главной странице (рис. 7.7).

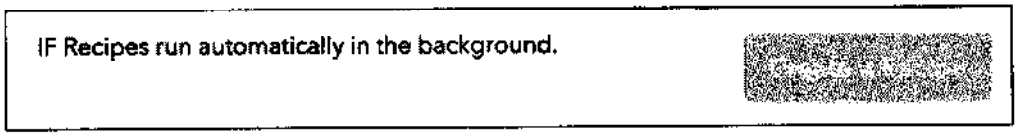


Рис. 7.7. Создание правила для электронной почты

В окне для создания нового правила выберите подключенный ранее канал **Maker** (рис. 7.8).

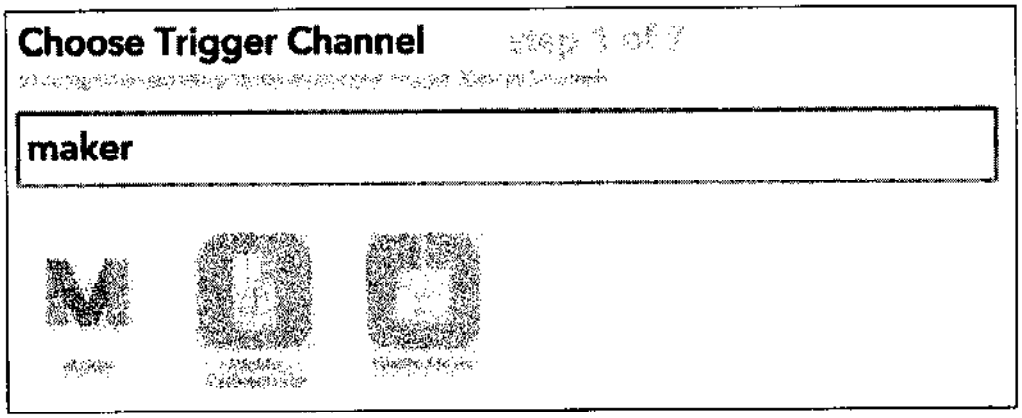


Рис. 7.8. Выбор канала триггера для отправки сообщений

[3] Сейчас эта кнопка называется **New Applet**. — *Прим. пер.*

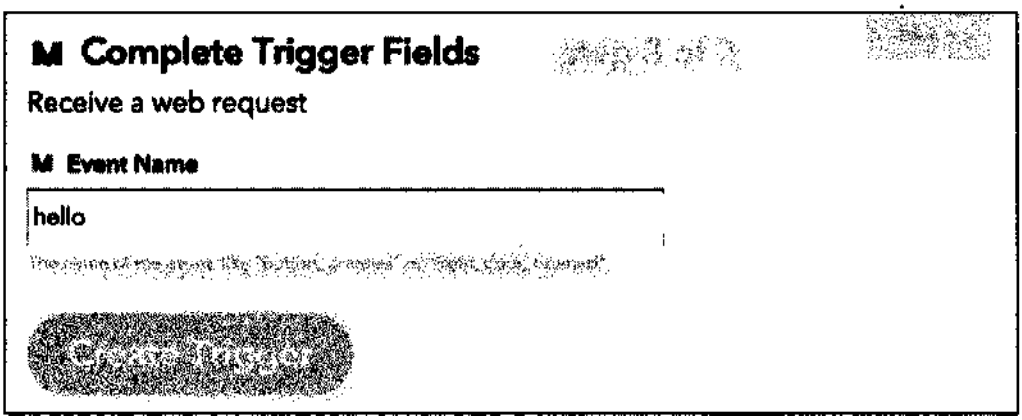


Рис. 7.9. Имя события для отправки сообщения

В качестве *события* (триггера) просто напишем `hello` (рис. 7.9).

В качестве *канала действия* выберите добавленный ранее канал электронной почты (рис. 7.10).

Вас попросят заполнить поля канала действия содержимым электронного письма (рис. 7.11).

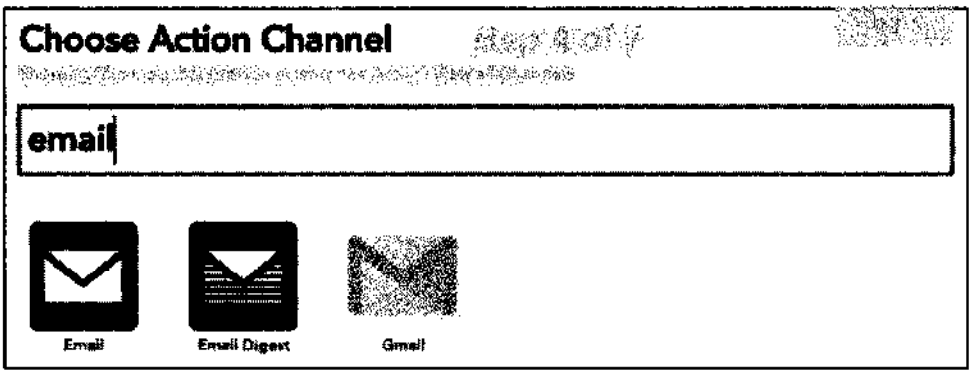


Рис. 7.10. Выбираем канал действия

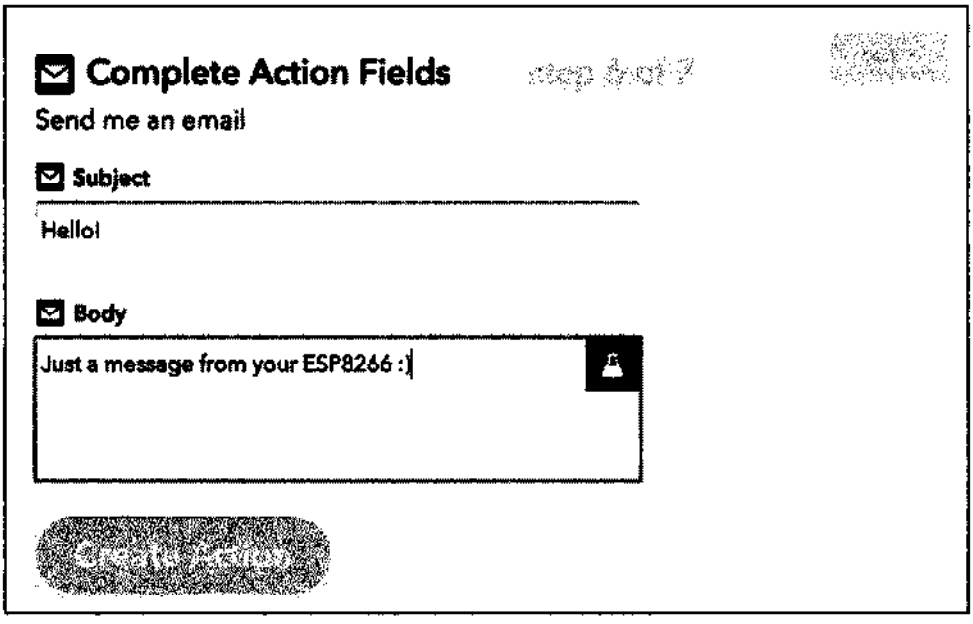


Рис. 7.11. Вводим тему и тело сообщения

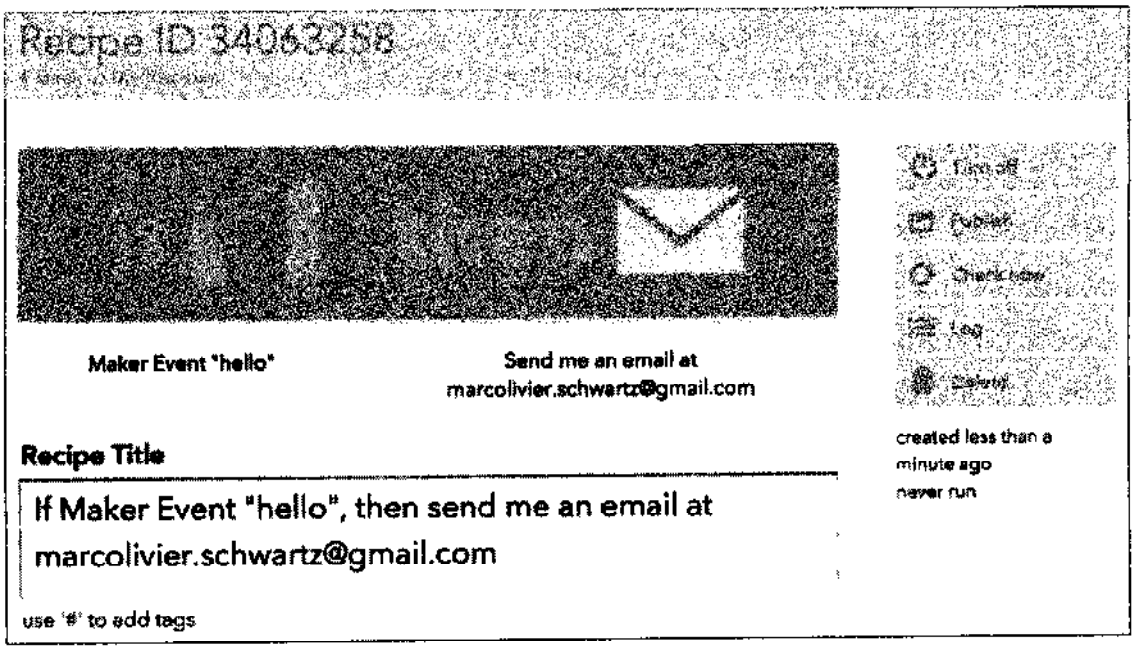


Рис. 7.12. Готовое правило для отправки электронного письма

Сохраните правило, и оно появится на вашей панели IFTTT (рис. 7.12).

Теперь, когда правило создано, настало время разработать скетч Arduino, который его активирует. Как обычно, здесь мы детально рассмотрим лишь наиболее важные части скетча:

1. Он начинается с подключения библиотеки ESP8266WiFi:

```
#include <ESP8266WiFi.h>
```

2. Сюда вам надо подставить имя и пароль вашей сети Wi-Fi:

```
const char* ssid = "your_wifi_ssid";
const char* password = "your_wifi_password";
```

3. После этого поместить некоторую информацию о IFTTT — имя события и секретный ключ вашего канала **Maker**:

```
// Параметры доступа к IFTTT
const char* host = "maker.ifttt.com";
const char* eventName   = "hello";
// Введите здесь ваш ключ сервиса Maker Webhooks
const char* key = "**************************************";
```

4. В функции `setup()` мы подключаем ESP8266 к сети Wi-Fi:

```
Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
```

```
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
```

5. В функции `loop()` первым делом подключаемся к серверу IFTTT:

```
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
  Serial.println("connection failed");
  return;
}
```

6. Затем готовим запрос, который содержит имя события:

```
String url = "/trigger/";
url += eventName;
url += "/with/key/";
url += key;
```

7. Мы можем отправить запрос на сервер:

```
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
             "Host: " + host + "\r\n" +
             "Connection: close\r\n\r\n");
int timeout = millis() + 5000;
// Проверяем продолжительность ожидания ответа
while (client.available() == 0) {
  if (timeout - millis() < 0) {
    Serial.println(">>> Client Timeout !");
    client.stop();
    return;
  }
}
```

8. И читаем ответ сервера:

```
while(client.available()){
  String line = client.readStringUntil('\r');
  Serial.print(line);
}
```

9. Наконец, мы ждем минуту перед каждой отправкой запроса:

```
delay(60 * 1000);
```

Теперь самое время проверить проект!

Скачайте сначала код программы из репозитория GitHub для этой книги: **https://github.com/openhomeautomation/iot-esp8266-packt**.

Готовый к использованию файл программы находится также в папке ch7_EMAIL сопровождающего книгу электронного архива (см. *приложение*).

Затем измените код, подставив в него ваши параметры Wi-Fi и ключ IFTTT. Загрузите прошивку в плату и откройте окно монитора последовательного порта. Вы должны увидеть, что плата соединилась с сервером IFTTT и получила подтверждение запуска события (рис. 7.13).

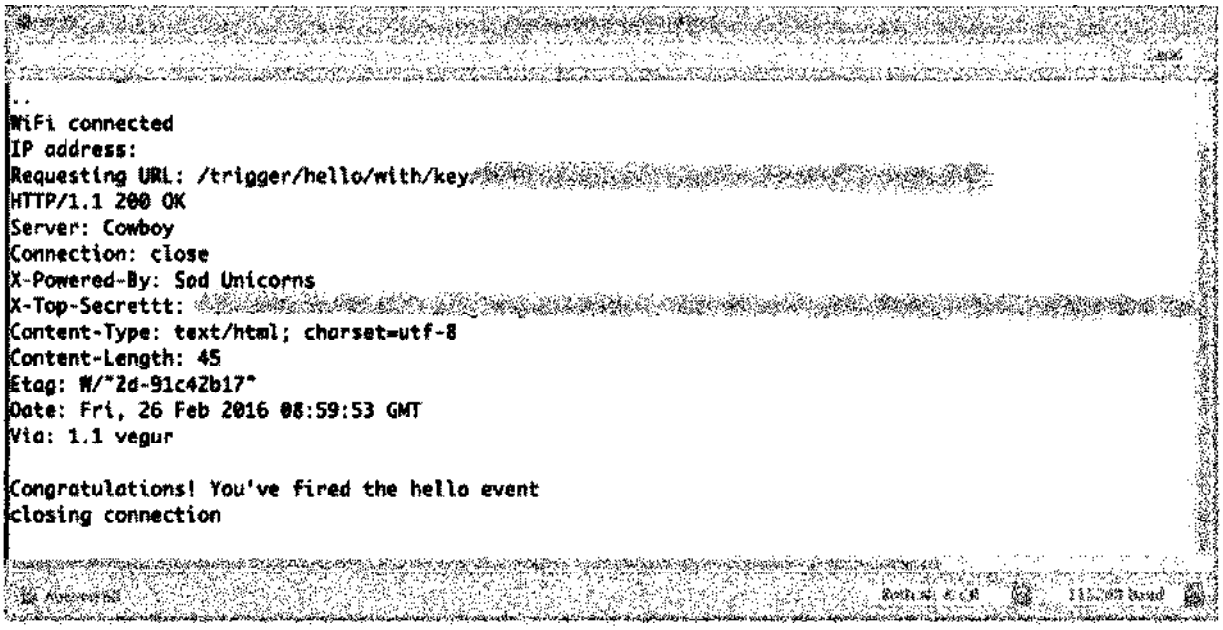


Рис. 7.13. Журнал подключения к серверу IFTTT и подтверждение срабатывания события

Проверьте также свой почтовый ящик — вы должны получить сообщение от вашей платы ESP8266 (рис. 7.14).

Теперь скорее отключите питание ESP8266 или деактивируйте правило, иначе вас завалит спамом собственный проект!

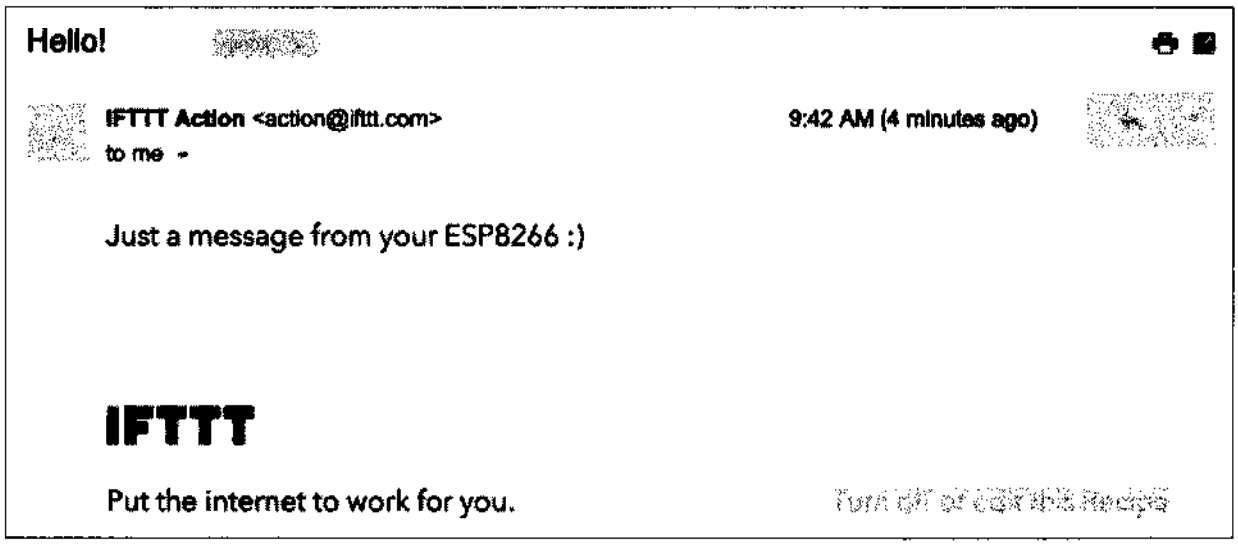


Рис. 7.14. Письмо от ESP8266

Отправка данных в SMS

Воспользуемся тем же самым оборудованием для совершенно иного проекта — при помощи сервиса IFTTT мы будем отправлять данные измерений из ESP8266 на ваш мобильный телефон (смартфон):

1. Первый шаг заключается в подключении сервиса SMS к вашей учетной записи IFTTT. Это очень просто — нужно лишь ввести ваш телефонный номер[4] (рис. 7.15).

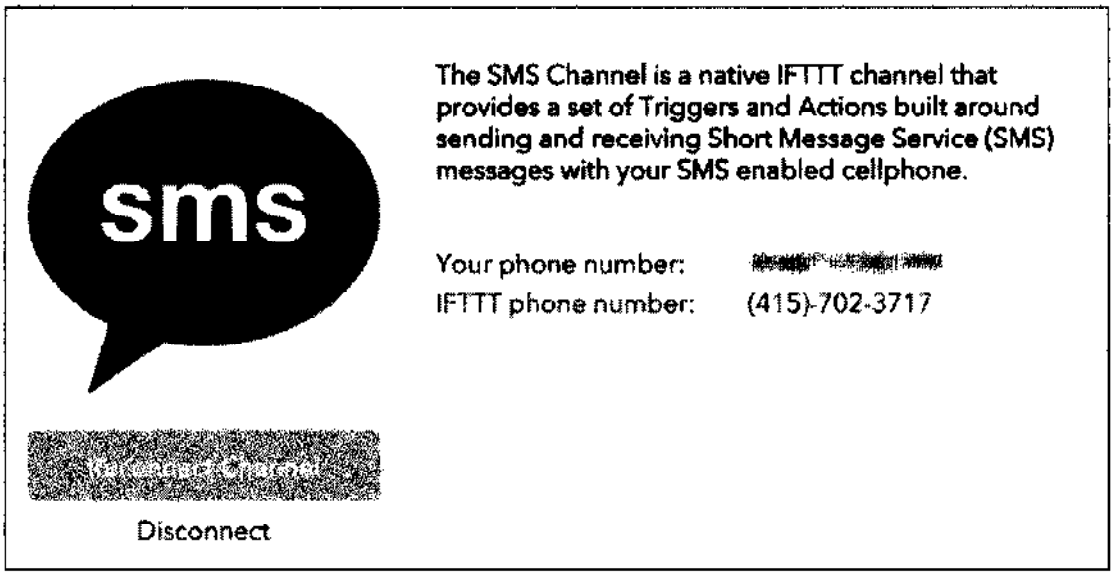


Рис. 7.15. Подключение сервиса **SMS** к учетной записи IFTTT

2. Создадим новое правило. В этот раз в качестве имени события используйте слово `data` (рис. 7.16).

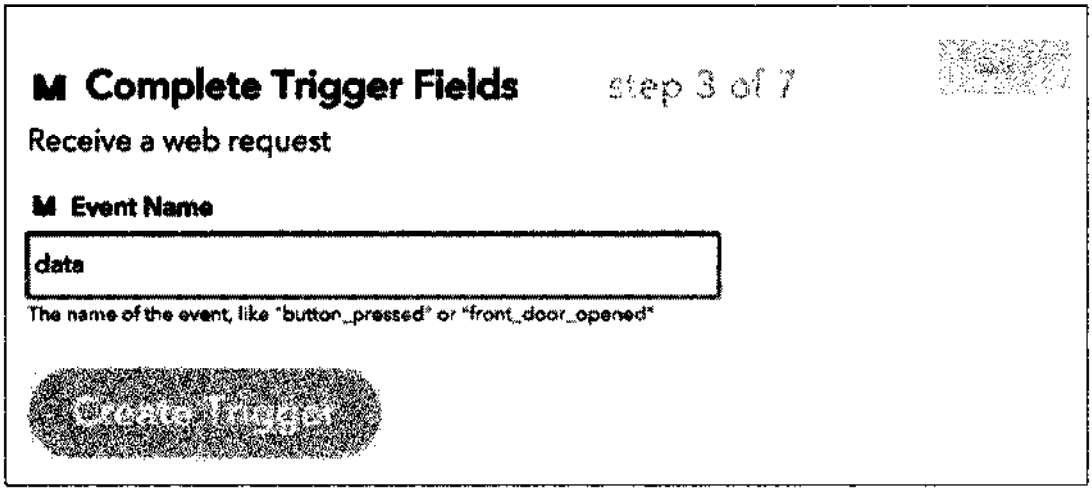


Рис. 7.16. Имя события для активации отправки сообщения SMS

[4] Номер следует вводить с двумя ведущими нолями перед кодом страны, например: 007913xxxxxx. Этот сервис работает с номерами «большой тройки», поддержка номеров других российских операторов не гарантируется. — *Прим. пер.*

3. Затем выберите недавно подключенный сервис **SMS** в качестве действия для правила (рис. 7.17).

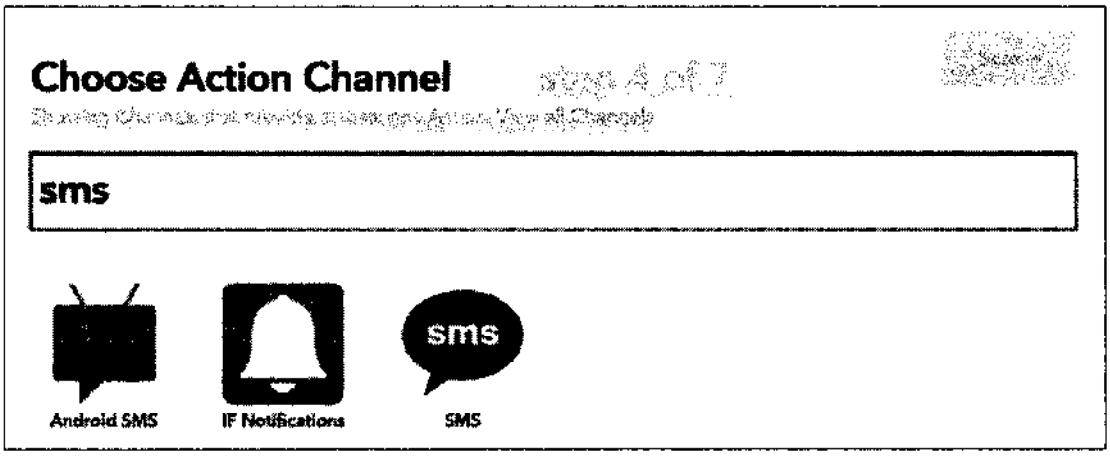


Рис. 7.17. Выбираем сервис, который выполнит действие по событию

4. Создадим теперь более сложное сообщение, чем в первом проекте этой главы, потому что мы хотим получать информацию об измерениях, выполненных ESP8266 с помощью включенного в нашу схему датчика DHT11 (см. рис. 7.2). Благодаря IFTTT это очень просто сделать, используя инструмент помощника для добавления значений (кнопка **Add Ingredient**)[5], поступивших из триггерного канала (рис. 7.18).

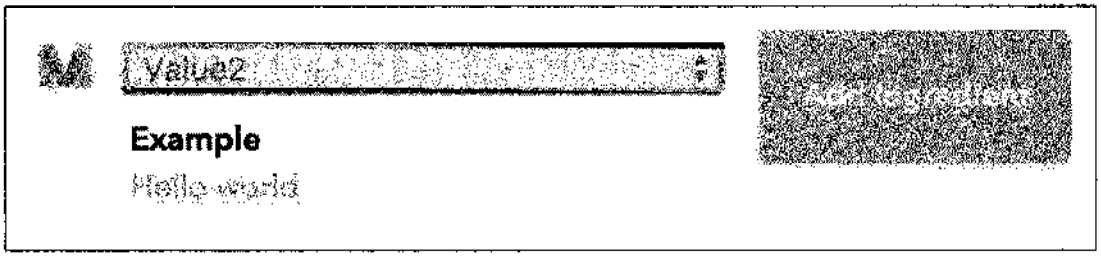


Рис. 7.18. Добавление переменных в триггерный канал

5. Ваше сообщение в итоге должно выглядеть так, как показано на рис. 7.19.
6. Наконец, создаем правило (рис. 7.20).

Теперь, когда правило активно, мы можем перейти к разработке скетча Arduino. Поскольку многое в нем совпадает с предыдущим скетчем, здесь я отмечу только наиболее важные изменения:

1. Вы должны подключить библиотеку DHT:

```
#include "DHT.h"
```

2. Затем мы определяем тип датчика и вывод для подключения:

```
#define DHTPIN 5
#define DHTTYPE DHT11
```

[5] В новом интерфейсе это кнопка **+Ingredients**. — *Прим. пер.*

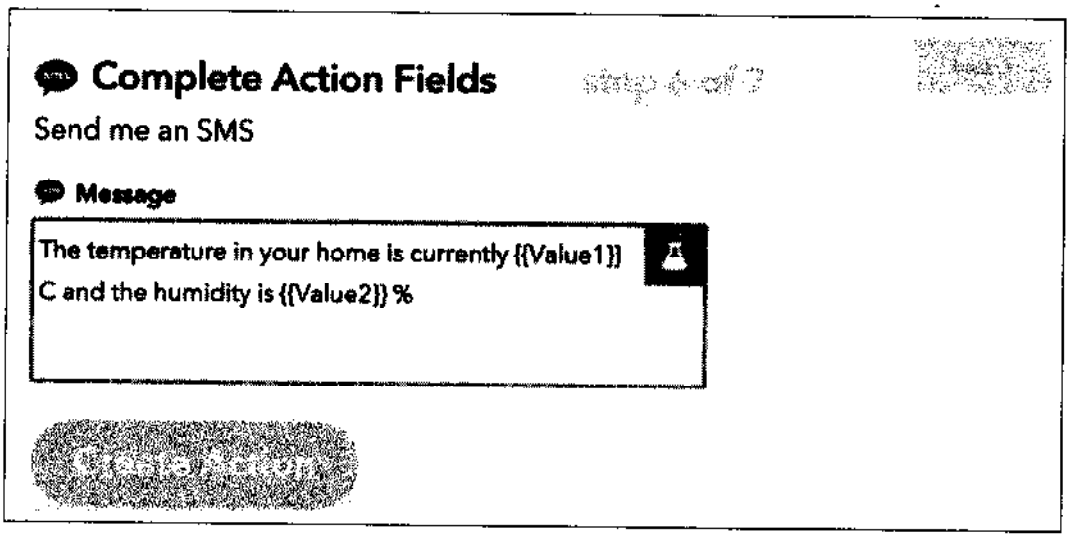


Рис. 7.19. Шаблон сообщения с указанием переменных для подстановки

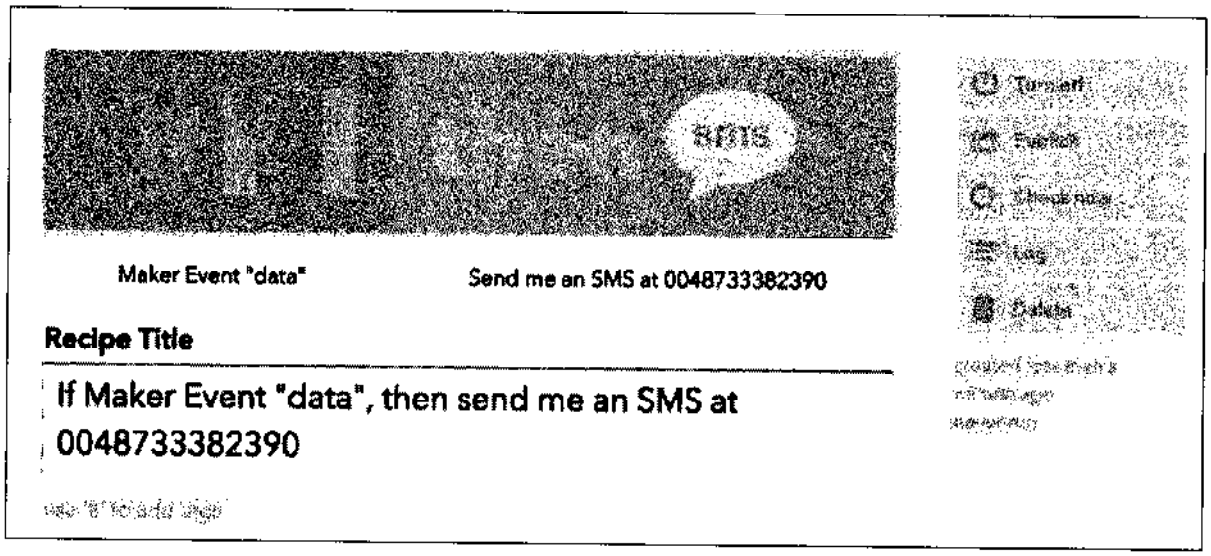


Рис. 7.20. Законченное правило для отправки сообщений

3. Создаем объект датчика:

```
DHT dht(DHTPIN, DHTTYPE);
```

4. В этот раз мы назовем событие словом `data`:

```
const char* eventName = "data";
```

5. В функции `setup()` нашего скетча инициализируем датчик DHT:

```
dht.begin();
```

6. Затем внутри функции `loop()` получаем данные от сенсора:

```
float h = dht.readHumidity();
float t = dht.readTemperature();
```

7. После этого вставляем данные в обращение к серверу:

```
String url = "/trigger/";
url += eventName;
```

```
url += "/with/key/";
url += key;
url += "?value1=";
url += String(t);
url += "&value2=";
url += String(h);
```

Настало время испытать скетч! Скачайте его из репозитория GitHub для книги и не забудьте модифицировать, добавив свои параметры Wi-Fi и ключ IFTTT.

Готовый к использованию файл программы находится в папке ch7_SMS сопровождающего книгу электронного архива (см. *приложение*).

Загрузите прошивку в плату. Через несколько секунд вы должны получить на свой смартфон сообщение, которое содержит данные измерений (рис. 7.21).

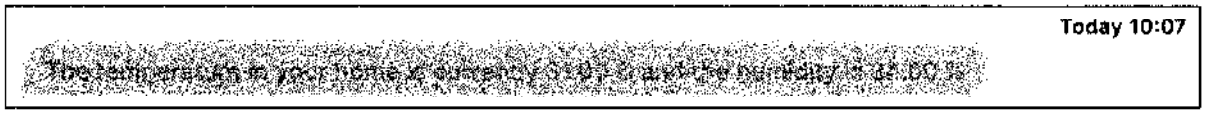


Рис. 7.21. Входящее сообщение с результатами измерений

Получение push-уведомлений

В завершающем проекте главы мы узнаем, как отправлять третий тип сообщений — push-уведомления. Это идеально для сигналов тревоги, потому что сообщение отображается на экране смартфона немедленно после срабатывания триггера события.

Для организации такого рода сообщений мы воспользуемся приложением под названием **Pushover**, доступным для iOS и Android.

Прежде всего создайте учетную запись на сайте **https://pushover.net/**. В панели настроек приложения вам следует скопировать ключ API — он потребуется для привязки Pushover к учетной записи IFTTT.

Теперь, войдя на IFTTT, вы можете подключить канал Pushover (рис. 7.22).

Создадим последнее правило в этой главе. В качестве имени события используем слово `alert` (рис. 7.23).

Затем в качестве обработчика события выбираем канал **Pushover** (рис. 7.24).

Текст push-уведомления может быть любым — каким захотите. Например, это может быть сообщение о том, что в доме слишком большая влажность (рис. 7.25).

Сохраните правило. Оно должно выглядеть так, как показано на рис. 7.26.

Теперь запрограммируем плату ESP8266. Этот скетч очень похож на предыдущие скетчи этой главы, поэтому я и здесь отмечу только изменения.

Прежде всего, мы должны изменить имя события, которое отправляем в IFTTT:

```
const char* eventName = "alert";
```

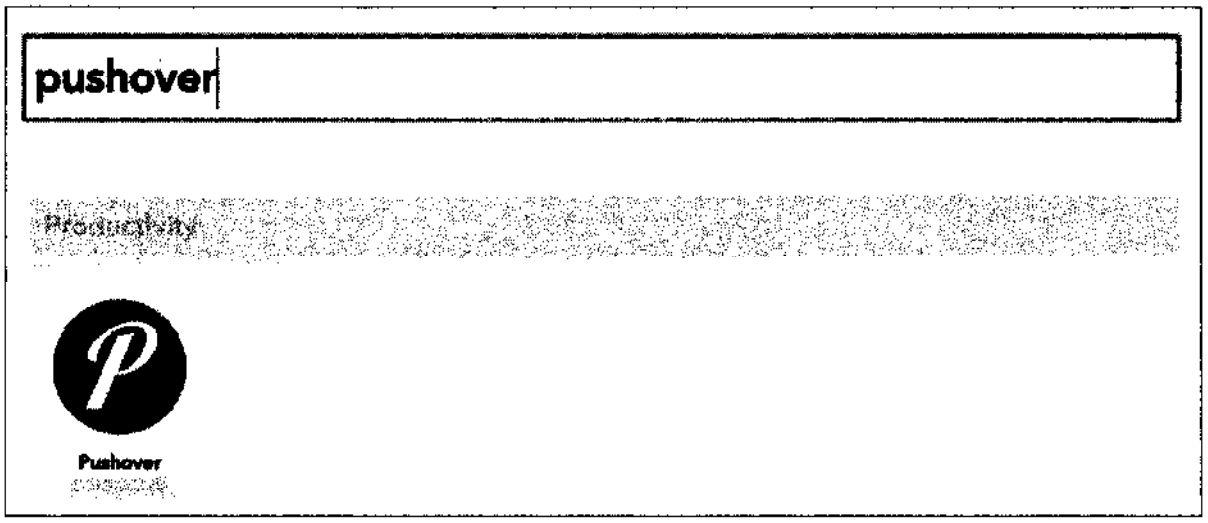


Рис. 7.22. Подключаем канал **Pushover** на сайте IFTTT

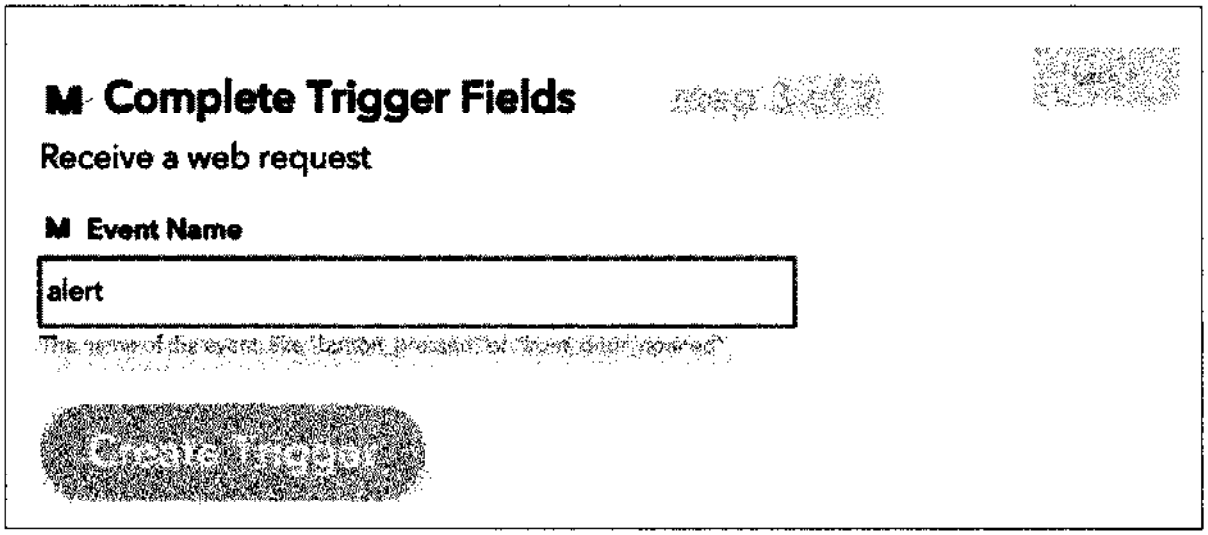


Рис. 7.23. Создаем триггер для события **alert**

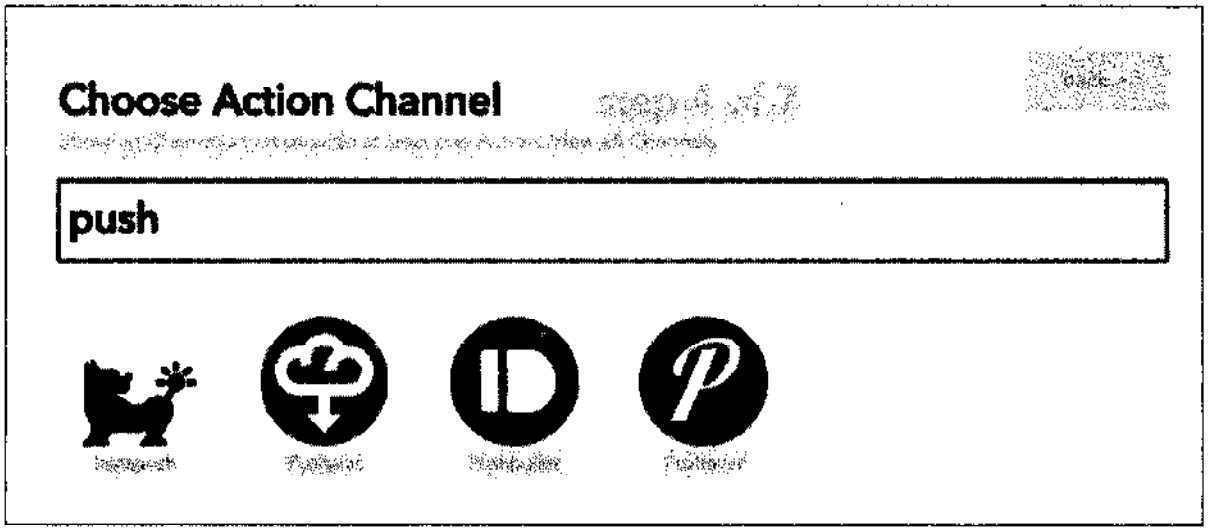


Рис. 7.24. Выбираем канал обработчика событий

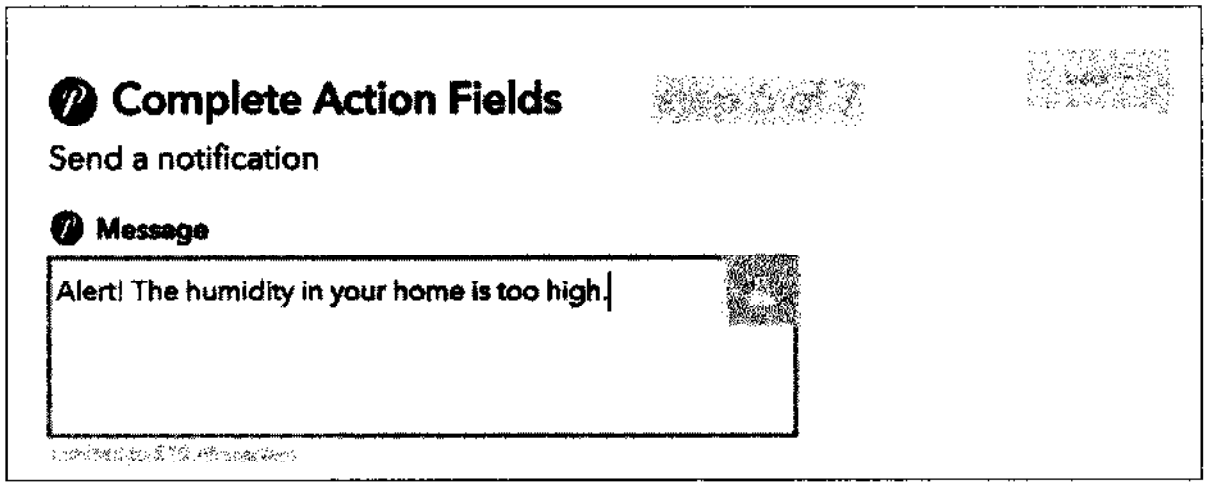


Рис. 7.25. Сообщение, которое отобразится на экране смартфона

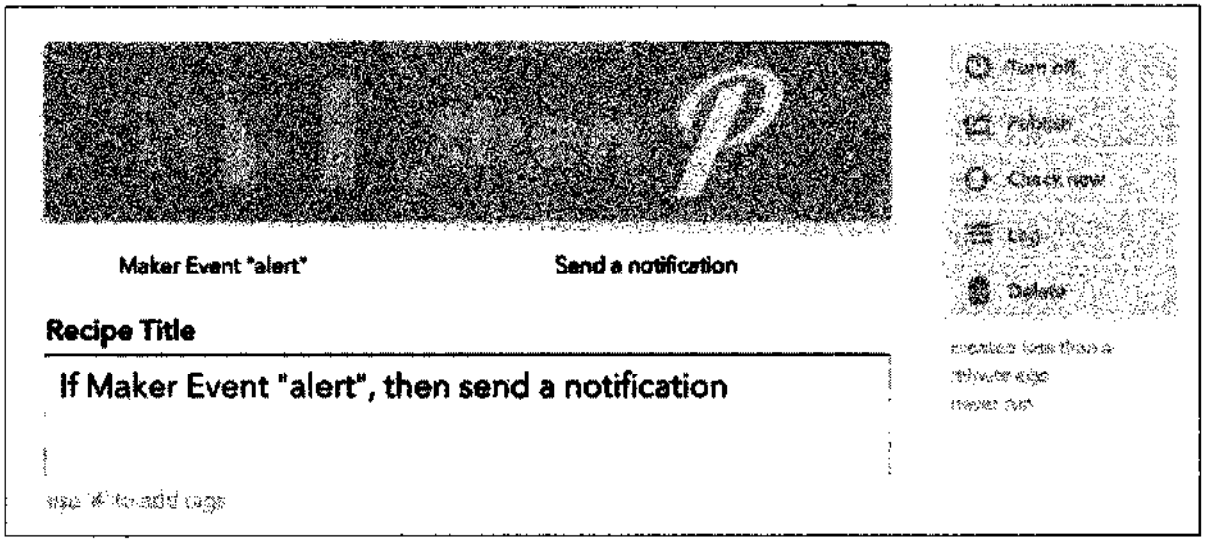


Рис. 7.26. Правило для push-уведомлений на панели IFTTT

Затем мы программируем плату на отправку предупреждения, если влажность поднимется выше 30%:

```
if (h > 30.00) {
```

Также надо добавить увеличенную задержку после срабатывания триггера, чтобы не получать сигнал тревоги каждые 5 секунд:

```
delay(10 * 60 * 1000);
```

Настало время проверить проект! Скачайте код программы из репозитория GitHub, измените параметры доступа к сети Wi-Fi и сервису IFTTT. Загрузите прошивку в плату.

Готовый к использованию файл программы находится в папке ch7_PUSH сопровождающего книгу электронного архива (см. *приложение*).

Если влажность превысит пороговый уровень, вы должны очень скоро получить на свой смартфон соответствующее уведомление (рис. 7.27).

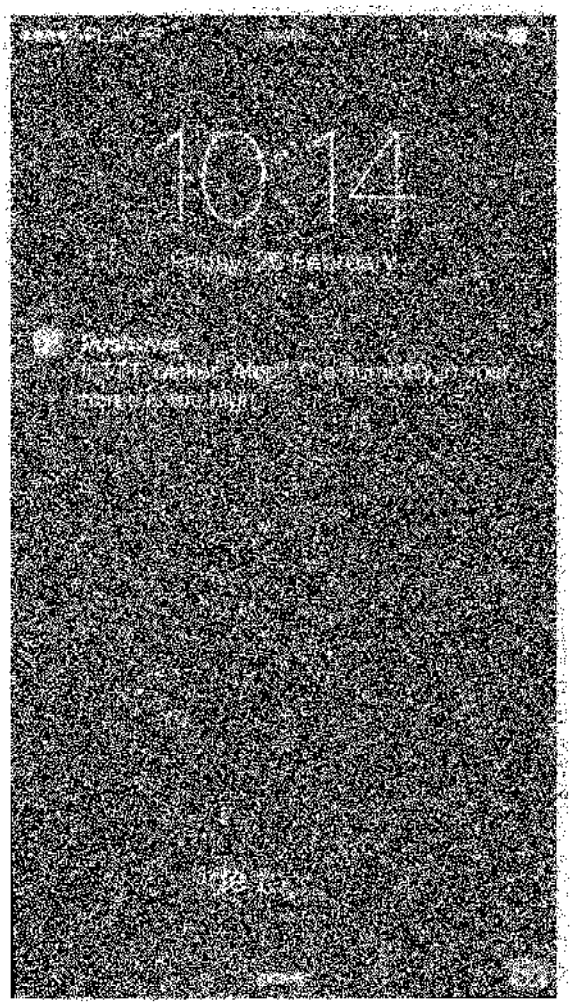


Рис. 7.27. Push-уведомление на экране смартфона

Такой тип сообщения намного более удобен, чем сообщение по электронной почте или SMS, поскольку вы видите сообщение, даже если в этот момент не проверяете почту или SMS. Следовательно, это прекрасный инструмент для срочных сигналов тревоги, и вы теперь знаете, как отправлять такие сигналы прямо с вашей платы ESP8266.

Заключение

В этой главе мы узнали, как с помощью платы ESP8266 автоматически отправлять сообщения по электронной почте и на смартфон в виде SMS и push-уведомлений. Это позволяет нам получать на смартфоне данные замеров с различных датчиков или тревожные сообщения, связанные с результатами таких замеров.

Вы можете применить полученные знания к своим потребностям и организовать получение от одного или нескольких модулей ESP8266 предупреждений, которые немедленно дадут вам знать об изменении отслеживаемого вами параметра.

В следующей главе мы объединим все полученные нами к настоящему моменту знания и создадим новый проект с ESP8266, — дверной замок, которым можно управлять через облако.

8

Управляем дверным замком через облако

В этой главе мы объединим все ранее полученные знания, чтобы построить законченный проект Интернета вещей, пригодный в реальной жизни.

А займемся мы созданием дверного замка, которым вы сможете управлять из любой точки планеты при помощи микросхемы ESP8266 Wi-Fi. Мы разберемся также, как интегрировать в нашу программу уведомления, чтобы получить push-сообщение, если кто-то другой открыл ваш замок. Начинаем!

Оборудование и программное обеспечение

Для проекта этой главы потребуется следующее оборудование:

- прежде всего — это плата с микросхемой ESP8266. В основном я использую в этой книге модуль Adafruit Huzzah ESP8266, но любой другой модуль ESP8266 будет работать здесь ничуть не хуже;
- в качестве замка я использовал защелку с соленоидным приводом, работающую от постоянного напряжения 12 В (рис. 8.1). Впрочем, вы можете взять любой аналогичный замок, имеющийся в продаже;
- для подключения замка вам понадобятся *n-p-n*-транзистор, защитный диод и резистор 1 кОм. Далее я привожу ссылки на эти компоненты;
- источник питания 12 В постоянного тока, который вы сможете подключить к макетной плате;
- может понадобиться также модуль USB FTDI 3,3/5 В для программирования чипа ESP8266[1];
- беспаечная макетная плата и соединительные провода.

[1] При использовании платы NodeMCU этот модуль не понадобится. — *Прим. пер.*

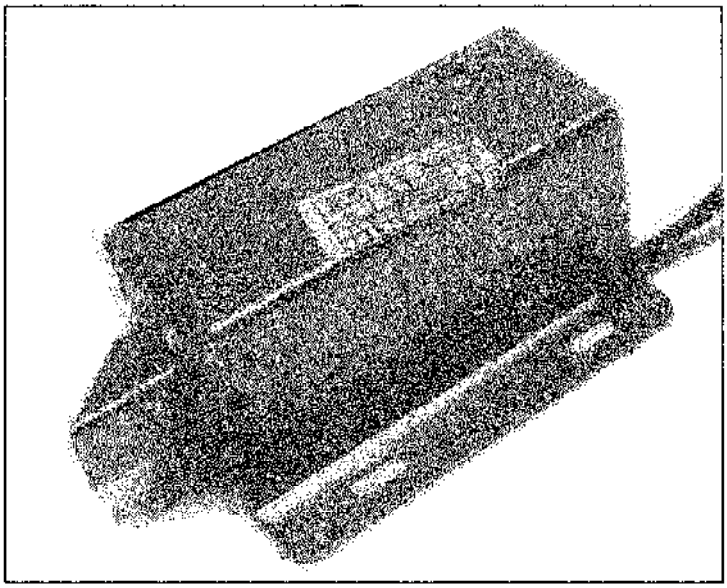


Рис. 8.1. Дверная защелка с приводом соленоидного типа

Вот конкретный перечень использованных в проекте компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266: **https://www.adafruit.com/products/2471**;
- модуль FTDI USB: **https://www.adafruit.com/products/284**;
- дверная защелка соленоидного типа: **https://www.adafruit.com/products/1512**;
- *n-p-n*-транзистор: **https://www.sparkfun.com/products/13689**;
- резистор: **https://www.sparkfun.com/products/8980**;
- защитный диод: **https://www.sparkfun.com/products/8589**;
- макетная плата: **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

Если вы до сих пор не установили среду разработки Arduino IDE, сделайте это сейчас. Самую свежую версию среды можно скачать по адресу: **http://www.arduino.cc/en/Main/Software**.

Для последнего раздела этой главы вам понадобится учетная запись сервиса IFTTT. Если вы еще ее не завели, обратитесь к *главе 6*, чтобы узнать, как создать аккаунт IFTTT.

Сборка схемы

Займемся аппаратной частью проекта. Проект этот достаточно сложен, поэтому на рис. 8.2 я привел его принципиальную схему.

Сначала установите на макетную плату транзистор и соедините его базу с выводом D5 модуля ESP8266 (платы NodeMCU или аналогичной) через резистор 1 кОм. Соедините эмиттер транзистора с общим проводом (GND). После этого соедините один вывод защелки с линией питания (VCC), а второй — с оставшимся выводом

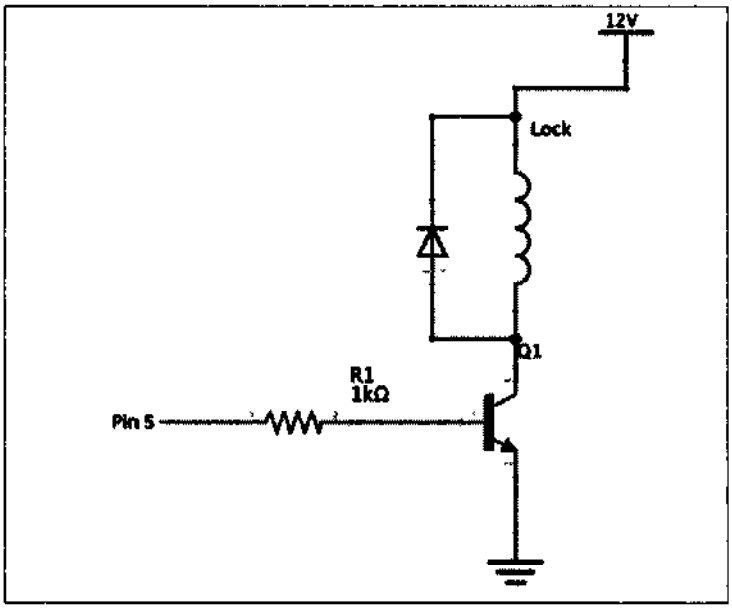


Рис. 8.2. Принципиальная схема управления соленоидной защелкой

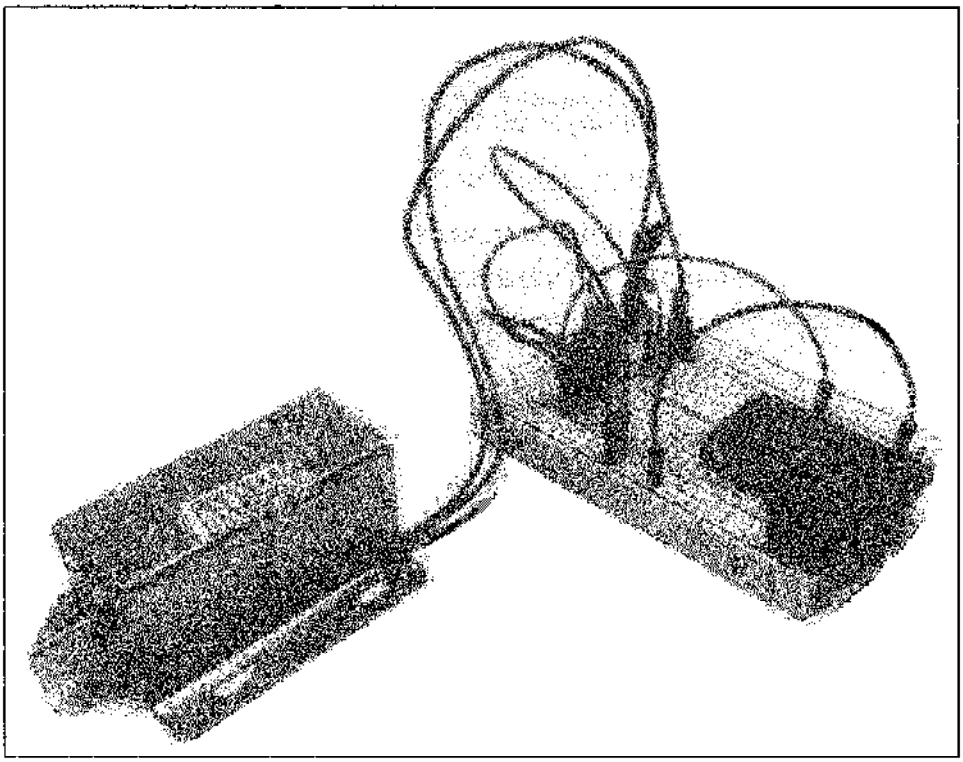


Рис. 8.3. Монтажная схема проекта

транзистора. Наконец, подключите диод параллельно защелке (замечу, что у диода катод обычно обозначен серой полоской). Собранная схема показана на рис. 8.3.

Программируем плату ESP8266

Запрограммируем теперь микросхему ESP8266, чтобы она могла принимать команды из облака. В общем-то, мы уже видели, как это делается, но всегда полезно освежить в памяти основы, чтобы потом разобраться с более сложным скетчем.

1. Сначала подключаем необходимые библиотеки:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
```

2. Затем объявляем клиентов Wi-Fi и PubSub (MQTT):

```
WiFiClient espClient;
PubSubClient client(espClient);
```

3. После этого создаем объект библиотеки aREST:

```
aREST rest = aREST(client);
```

4. Вы также должны внести в скетч имя и пароль вашей точки доступа Wi-Fi:

```
const char* ssid = "wifi-ssid";
const char* password = "wifi-pass";
```

5. Задаем имя устройства:

```
rest.set_id(device_id);
rest.set_name("door_lock");
```

6. Наконец, в функции `loop()` обслуживаем входящее подключение из облака:

```
rest.handle(client);
```

Просто скачайте готовый код из репозитория GitHub для этой книги, убедитесь, что изменили в скетче имя и пароль точки доступа Wi-Fi, загрузите прошивку в плату и отложите ее в сторону — для управления замком мы воспользуемся приборной панелью.

Готовый к использованию файл программы находится в папке ch8_DOOR_LOCK сопровождающего книгу электронного архива (см. *приложение*).

Управление замком из облачного сервиса

Для управления дверным замком из облака мы снова воспользуемся преимуществами библиотеки aREST и быстро создадим с ее помощью облачную панель управления:

1. Перейдите по адресу **http://dashboard.arest.io/** и создайте там себе учетную запись.
2. После создания учетной записи добавьте первую панель управления (рис. 8.4).

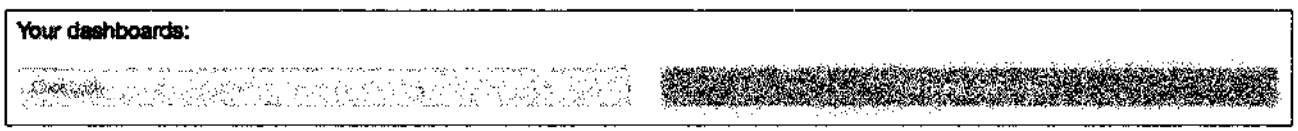


Рис. 8.4. Добавление новой панели управления

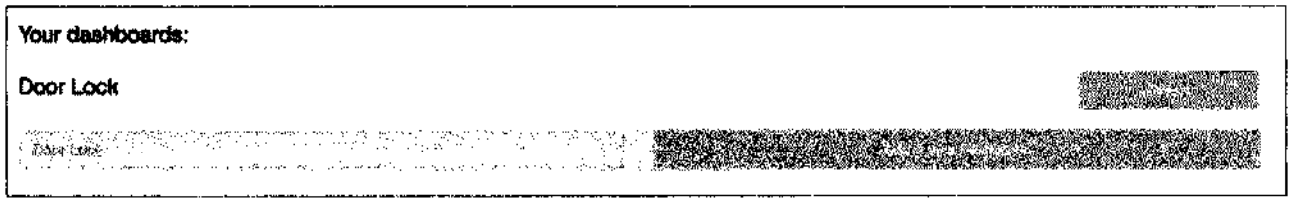


Рис. 8.5. Щелкните на имени новой панели

3. Создав панель, просто щелкните на ее имени (рис. 8.5).
4. В открывшейся панели создайте новый элемент (рис. 8.6). Поскольку вам нужна всего лишь кнопка включения/выключения, задайте следующие настройки: имя элемента `lock`, идентификатор устройства (разумеется, обязательно замените предложенный здесь идентификатор на идентификатор, которым вы обозначили свое устройство), а также укажите корректный номер вывода (5).

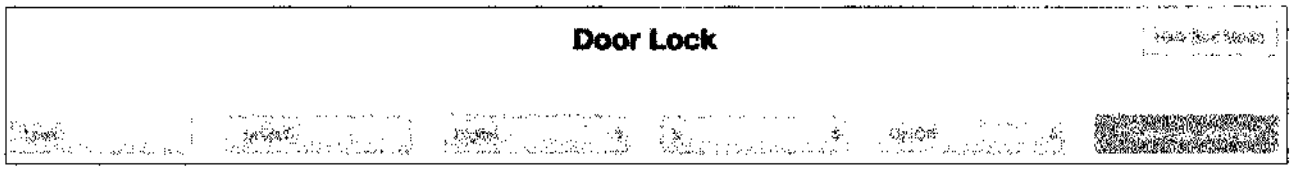


Рис. 8.6. Настройки элемента управления дверным замком

5. Завершив настройку, вы должны увидеть на своей облачной панели новый орган управления (рис. 8.7).

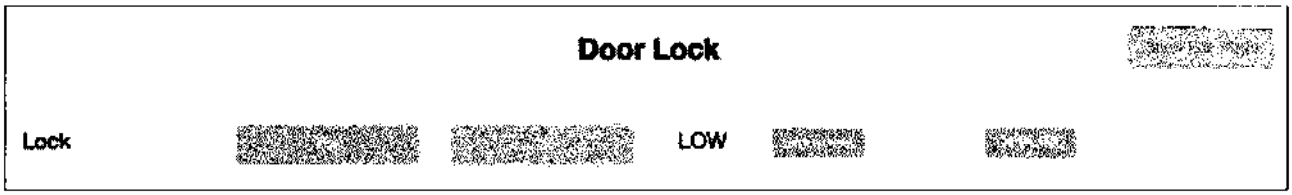


Рис. 8.7. Элемент управления готов к использованию

Чтобы испытать проект, просто щелкните на кнопке **On** — и замок должен немедленно сработать. Теперь вы можете запирать и отпирать свою дверь из любой точки мира! Только не нажимайте на кнопки слишком часто — в отличие от светодиода, замку требуется 1–2 секунды для срабатывания.

Получение уведомления об открытии замка

Управление замком через облако — это прекрасно, но текущее состояние замка вы можете увидеть, только открыв панель управления на компьютере или мобильном телефоне[2]. Но что, если вы мчитесь по шоссе, а замок подключен к важной двери

[2] Поскольку обратной связи с механизмом замка не предусмотрено, вы не можете дистанционно проверить его реальное состояние. Программный индикатор открытия на панели aREST не гарантирует, что механизм сработал на самом деле. Вероятно, автор сделал это допущение в учебных целях, чтобы упростить понимание проекта. — *Прим. пер.*

в вашем доме? Вы можете захотеть получить уведомление о том, что дверь открыта.

Именно это мы сейчас реализуем при помощи сервиса IFTTT — запрограммируем плату на отправку уведомлений на ваш смартфон, когда дверной замок открыт:

1. Сначала войдите на IFTTT и добавьте два канала, если еще не сделали этого: канал **Maker** и канал **Pushover**. Установите также на свой смартфон приложение Pushover. Чтобы узнать об этом больше, вернитесь к *главе 7*.
2. Затем создайте новое правило и выберите канал **Maker** в качестве обработчика триггера (рис. 8.8). Мы используем канал **Maker**, потому что он позволяет задействовать сервис IFTTT в пользовательских проектах, аналогичных нашим.
3. В качестве триггерного события введите `lock_opened` (рис. 8.9).

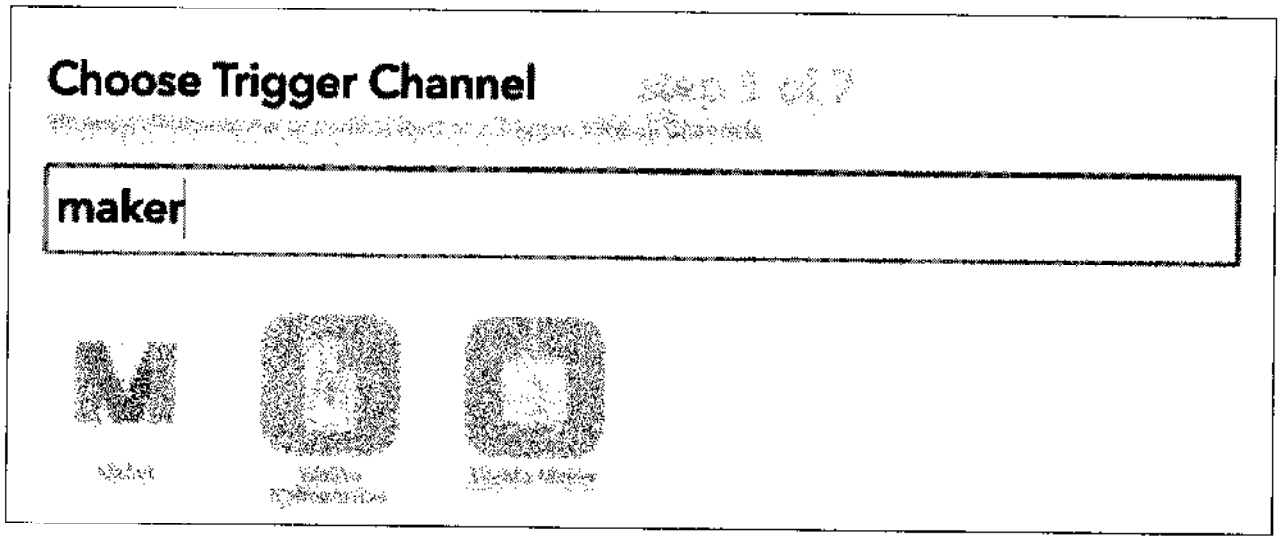


Рис. 8.8. Выбираем канал обработчика триггера

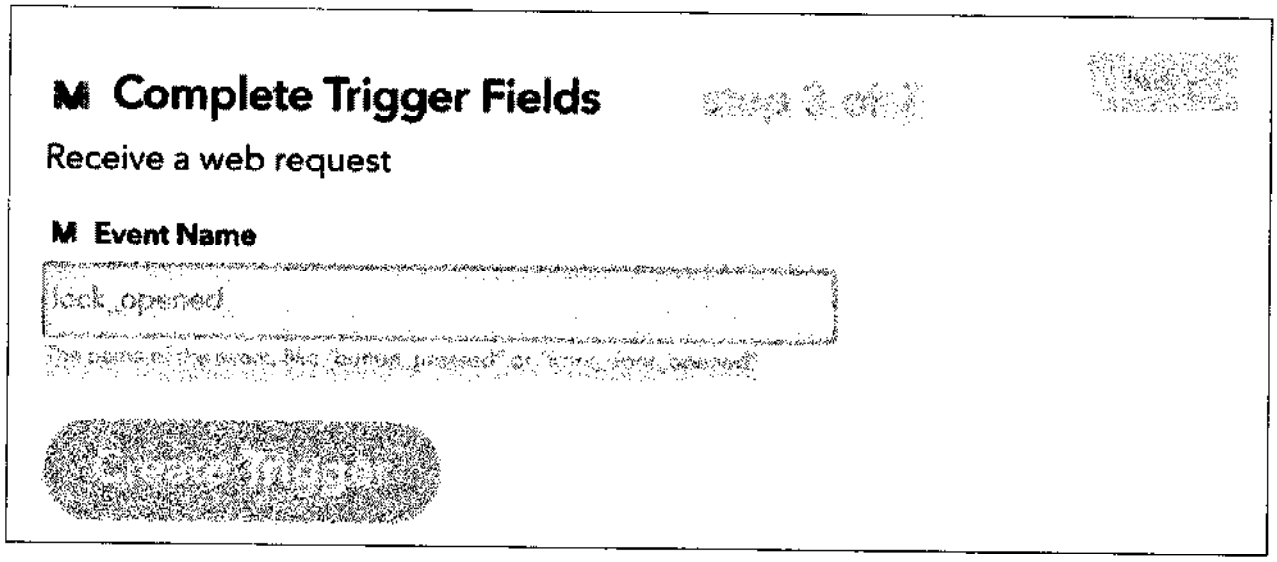


Рис. 8.9. Задаем имя триггерного события

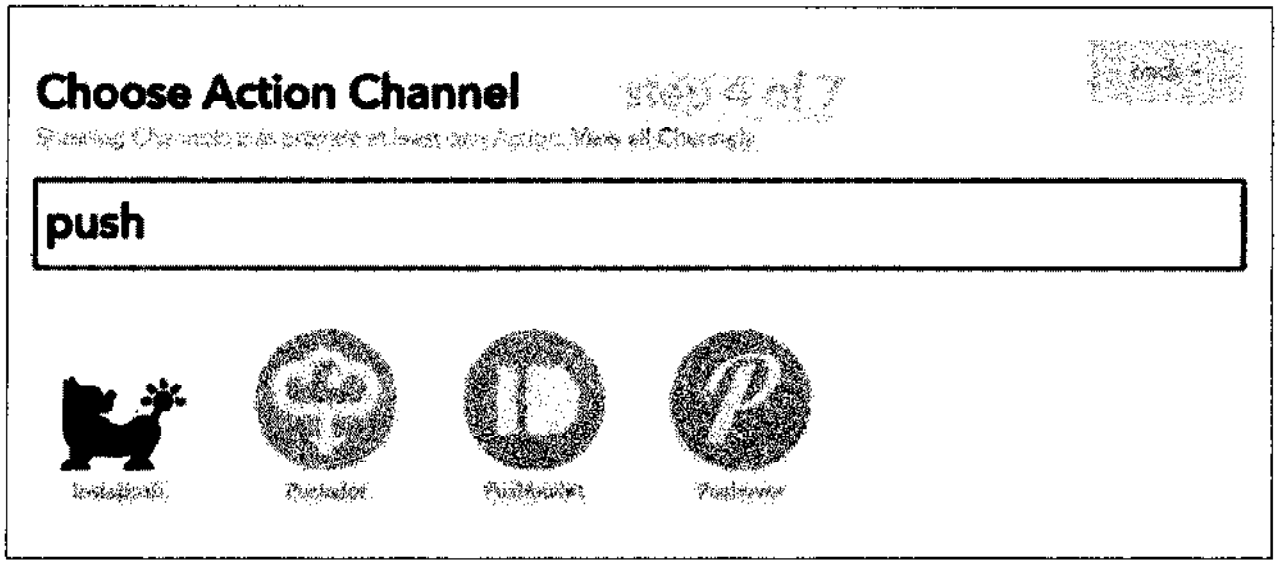


Рис. 8.10. Канал **Pushover** в качестве исполнителя действия

4. Исполнителем заданного действия будет канал **Pushover**, которым мы здесь воспользуемся, чтобы получать уведомления через IFTTT (рис. 8.10).
5. Выберите тип уведомления по своему усмотрению (рис. 8.11). Тем самым мы укажем сервису IFTTT, какой тип уведомления отправить в приложение: простой или с высоким приоритетом.

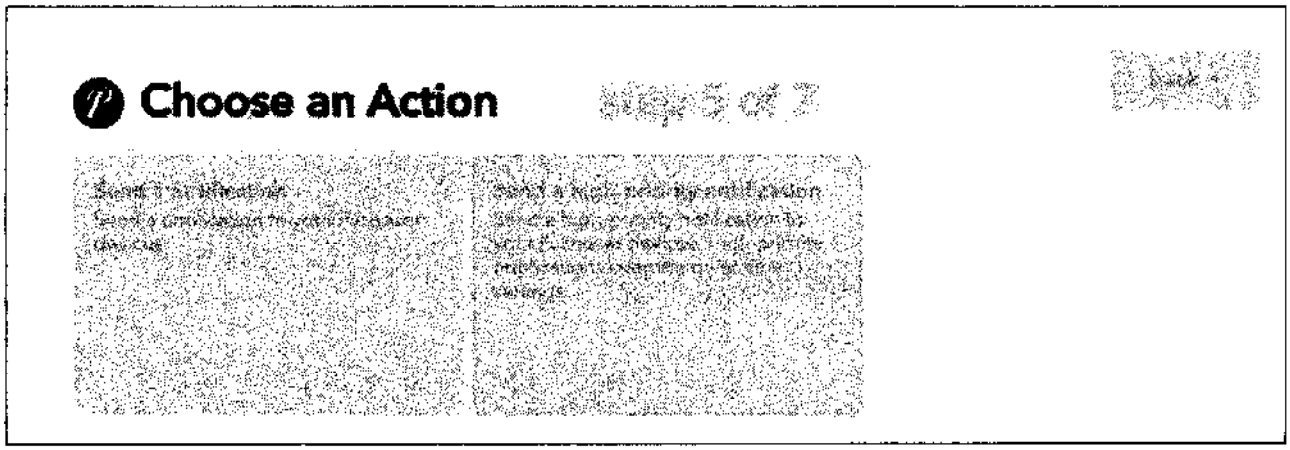


Рис. 8.11. Выбираем тип уведомления

6. Введите сообщение, которое вы хотите получать каждый раз, когда зафиксировано событие (рис. 8.12).
7. В завершение подтверждаем создание правила (рис. 8.13).
8. Теперь давайте взглянем, что нужно добавить в предыдущий код, чтобы подключить уведомления. Прежде всего, мы должны объявить две переменных для хранения статуса замка:

```
bool lockStatus;
bool previousLockStatus;
```

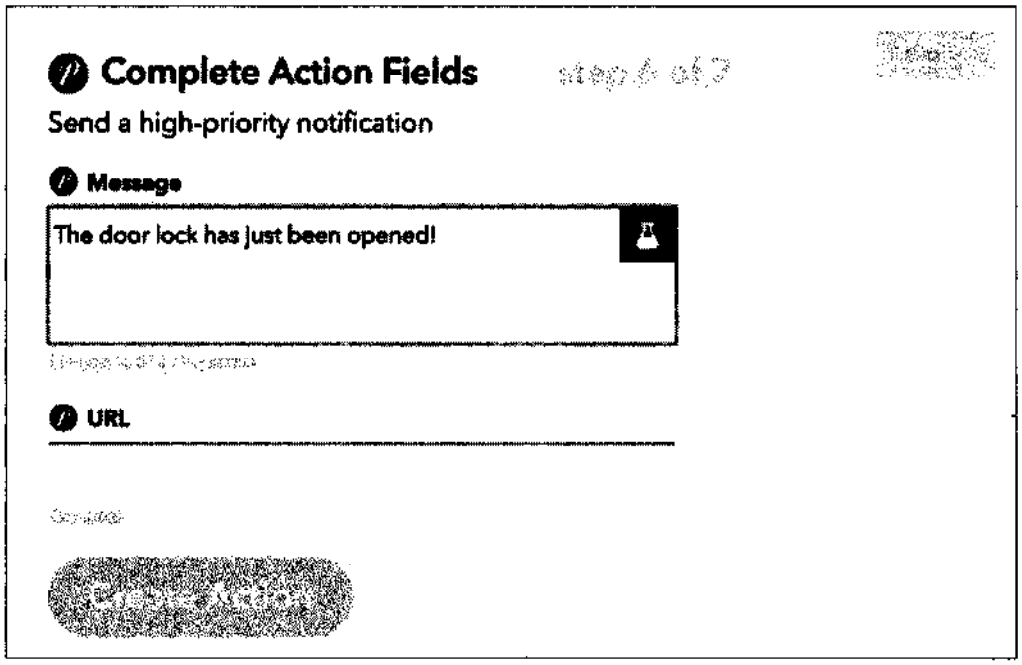


Рис. 8.12. Вводим текст уведомления о событии

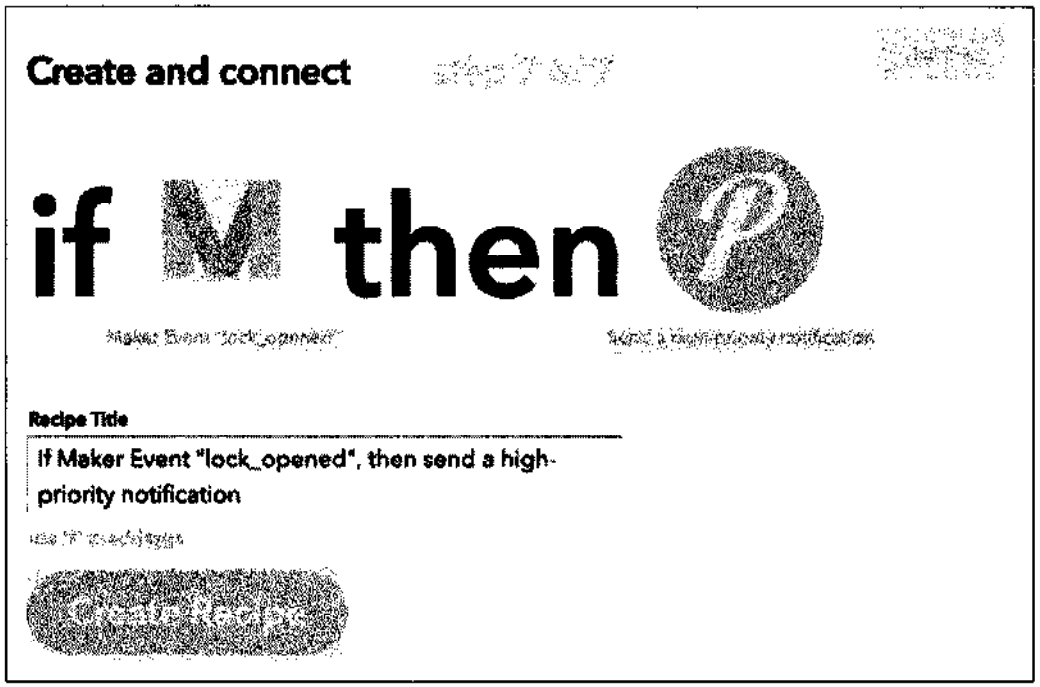


Рис. 8.13. Подтверждаем создание правила

9. Затем внутри функции `setup()` мы считываем уровень вывода номер 5, чтобы узнать, активирован ли замок в начале работы:

```
lockStatus = digitalRead(5);
previousLockStatus = lockStatus;
```

10. Далее, внутри функции `loop()` мы снова считываем уровень на выводе 5:

```
lockStatus = digitalRead(5);
```

11. Если он изменился по сравнению с ранее прочитанным — т. е. если замок открыт (это означает уровень HIGH на выводе), — мы отправляем уведомление через IFTTT:

```
if (lockStatus != previousLockStatus && lockStatus == 1) {
  Serial.print("connecting to ");
  Serial.println(host);
  // Используем WiFiClient для TCP-подключения:
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }
  // Создаем строку URI для запроса:
  String url = "/trigger/";
  url += eventName;
  url += "/with/key/";
  url += key;
  Serial.print("Requesting URL: ");
  Serial.println(url);
  // Отправляем запрос на сервер:
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  int timeout = millis() + 5000;
  while (client.available() == 0) {
    if (timeout - millis() < 0) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }
  }
  // Читаем ответ сервера и выводим в терминал:
  while (client.available()) {
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }
  Serial.println();
  Serial.println("closing connection");
}
// Сохраняем предыдущее состояние замка:
  previousLockStatus = lockStatus;
```

Чтобы узнать больше про отправку уведомлений через IFTTT, пожалуйста, прочтите *главу 7*.

12. Пришло время проверить проект! Загрузите прошивку в плату и вернитесь к онлайновой панели, которую вы раньше использовали для управления замком.

Готовый к использованию файл программы находится в папке ch8_DOOR_LOCK_PUSH сопровождающего книгу электронного архива (см. *приложение*).

Теперь каждый раз, когда вы открываете замок, вы должны немедленно получать уведомление на свой смартфон. При этом, если вы предоставите доступ к онлайновой панели кому-либо еще, о том, что они открыли замок, вы также узнаете.

Заключение

Здесь мы увидели, как все, изученное прежде в этой книге, можно применить для создания проекта, полезного у вас дома, — управляемого через облако дверного замка. Мы узнали, как управлять замком из любой части света, а также научились отправлять автоматические уведомления на смартфон, когда замок открыт.

Есть много вариантов доработки проекта, рассмотренного в этой главе. Один из них — добавить на одну панель несколько замков и управлять ими всеми, откуда угодно. Вы также можете добавить в проект функцию таймера и автоматически отпирать/запирать дверь в заданном временном интервале (например, когда приходит уборщик).

В следующей главе вы узнаете, как использовать свои знания о микросхеме ESP8266, чтобы создать еще один замечательный проект Интернета вещей — монитор (тикер) курса электронной валюты Bitcoin.

9

Монитор курса биткоина

К этому моменту мы уже рассмотрели, как микросхема ESP8266 может быть использована в самых различных ситуациях. Здесь мы применим ее для достаточно экзотического проекта Интернета вещей — получения текущего обменного курса биткоина и отображения его на небольшом OLED-экране.

Сначала мы узнаем, что такое «биткоин», и как можно получить его стоимость из веб-сервиса. Затем запрограммируем наш монитор курса биткоина и выведем стоимость биткоина на OLED-дисплей. Наконец, мы добавим в проект светодиоды, демонстрирующие, растет или падает курс. Начинаем!

Что такое «биткоин»?

Прежде, чем мы приступим к созданию проекта, позвольте сказать несколько слов о том, что такое «биткоин», если вы еще не знаете.

Биткоин — это виртуальная валюта, появившаяся в 2009 году. В отличие от других виртуальных валют, существовавших до того момента, биткоин защищен глобальной сетью компьютеров, работающих с одним и тем же распределенным регистром под названием *блокчейн* (blockchain), который гарантирует каждую выполненную транзакцию.

С тех пор многие люди и компании по всему миру начали принимать биткоин как средство оплаты. Биткоин также принимается в физических (не онлайновых) магазинах (рис. 9.1), которые оповещают об этом покупателей, вывесив объявление «Bitcoin Accepted Here» (Здесь принимают оплату биткоинами).

Разумеется, биткоин также торгуется на биржах и имеет обменный курс, номинированный в долларах и в большинстве основных валют. Например, на момент подготовки этой книги один биткоин стоил 417 долларов США.

Поэтому было бы довольно интересно, да и забавно, сделать проект на ESP8266, чтобы узнавать текущую стоимость биткоина. Именно этим мы сейчас займемся.

Рис. 9.1. Этот магазин принимает оплату виртуальной валютой «биткоин»

Онлайновые сервисы курса биткоина

Прежде, чем начать работать с онлайновыми сервисами, мы должны разобраться, как получить текущую стоимость биткоина в долларах США.

В Интернете довольно легко найти стоимость или исторические данные по курсу биткоина. Например, для этого хорошо подходит сайт Coindesk: **http://www.coindesk.com/price/**, предоставляющий доступ к текущему курсу биткоина и наглядным графикам его ретроспективы (рис. 9.2).

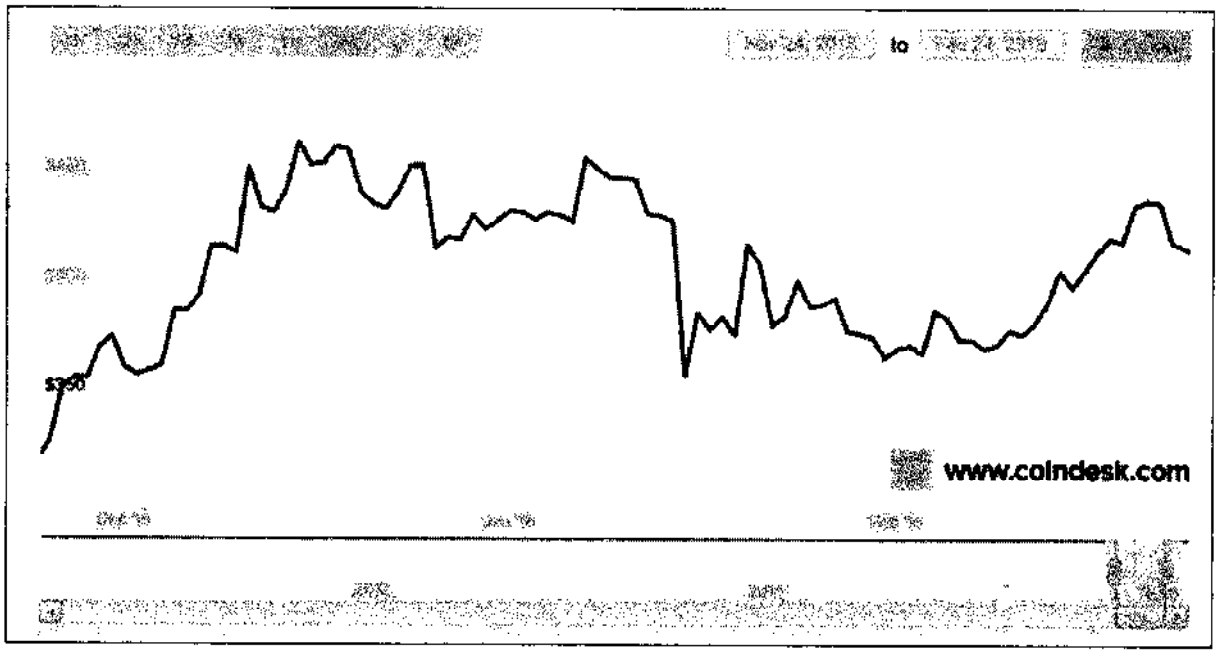


Рис. 9.2. График изменения курса биткоина за заданный период

Это замечательно, но данные доступны только через веб-сайт, и это слишком сложно для нашей маленькой платы ESP8266.

Более подходящее решение заключается в использовании биткоин-тикера — монитора, отображающего курс биткоина в режиме реального времени[1]. Хороший пример такого тикера можно найти на сайте **http://preev.com/btc/usd**. Вы увидите текущую стоимость биткоина, которую можно отобразить в различных валютах (рис. 9.3). Это именно то, чего мы ждем от нашего проекта, — простой биткоин-тикер, показывающий текущую стоимость биткоина в долларах США.

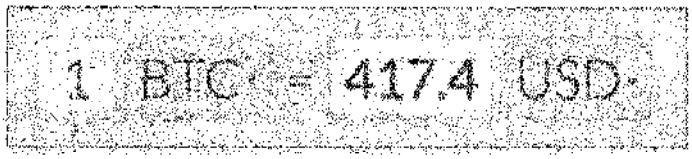


Рис. 9.3. Онлайновый тикер курса

Однако мы не можем обрабатывать эти данные непосредственно в ESP8266. Нам нужен доступ к API (Application Programming Interface, программный интерфейс приложения), который будет возвращать текущий курс биткоина в виде, доступном для обработки нашим чипом.

Лучший из известных мне API для получения курса также доступен на Coindesk по адресу: **http://api.coindesk.com/v1/bpi/currentprice.json**. Вы можете опробовать его прямо из браузера. Ответ будет похож на такой текст:

```
{
"time": {
"updated": "Feb 24, 2016 08:15:00 UTC",
"updatedISO": "2016-02-24T08:15:00+00:00",
"updateduk": "Feb 24, 2016 at 08:15 GMT"
  },
"disclaimer": "This data was produced from the CoinDeskBitcoin Price
  Index (USD). Non-USD currency data converted using hourly conversion
  rate from openexchangerates.org",
"bpi": {
"USD": {
"code": "USD",
"symbol": "&#36;",
"rate": "416.4970",
"description": "United States Dollar",
      "rate_float": 416.497
    },
"GBP": {
"code": "GBP",
"symbol": "&pound;",
```

[1] Бесплатные сервисы вносят задержку в 1–2 минуты, играть с их помощью на бирже не получится. — *Прим. пер.*

```
"rate": "298.0036",
"description": "British Pound Sterling",
      "rate_float": 298.0036
    },
"EUR": {
"code": "EUR",
"symbol": "&euro;",
"rate": "378.1955",
"description": "Euro",
      "rate_float": 378.1955
    }
  }
}
```


Нетрудно заметить, что API возвращает текущий курс биткоина в нескольких валютах. Именно этими данными мы и воспользуемся в нашем проекте для извлечения курса биткоина.

Оборудование и программное обеспечение

Для проекта этой главы потребуется следующее оборудование:

- прежде всего — это плата с микросхемой ESP8266. В основном я использую в этой книге модуль Adafruit Huzzah ESP8266, но любой другой модуль ESP8266 будет работать здесь ничуть не хуже;
- монохромный OLED-дисплей с разрешением 128×64 на драйвере SSD1306 (рис. 9.4). Насколько я знаю, только он совместим с ESP8266[2];

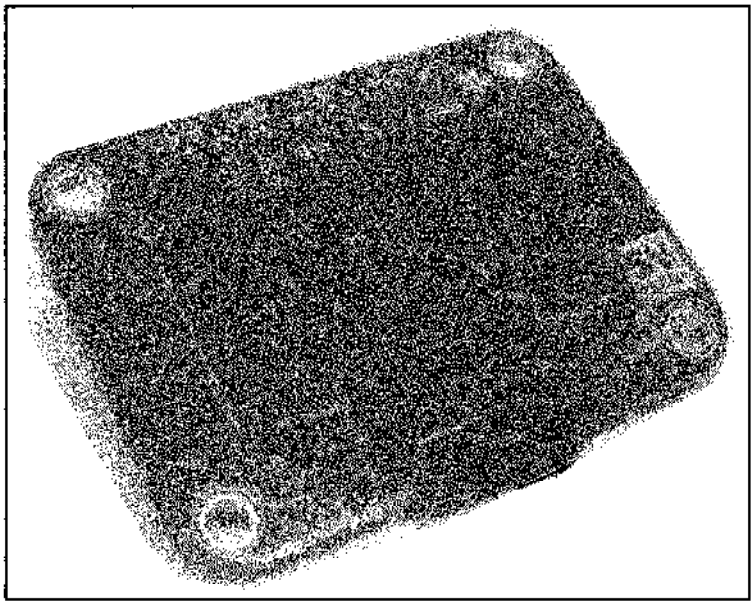


Рис. 9.4. Малогабаритный монохромный OLED-дисплей

[2] Более популярный на российском рынке и недорогой дисплей изображен на рис. Пр.4 в *предисловии к русскому изданию. — Прим. пер.*

- для второй части проекта я использовал два светодиода: красный и зеленый, включенные последовательно с резисторами 330 Ом;
- может понадобиться также модуль USB FTDI 3,3/5 В для программирования чипа ESP8266[3];
- беспаечная макетная плата и соединительные провода.

Вот конкретный перечень использованных в проекте компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266: **https://www.adafruit.com/products/2471**;
- модуль FTDI USB: **https://www.adafruit.com/products/284**;
- светодиод (2 шт): **https://www.sparkfun.com/products/9590**;
- резистор 330 Ом (2 шт): **https://www.sparkfun.com/products/8377**;
- дисплей OLED 128×64 пикселов на драйвере SSD1306: **https://www.adafruit.com/products/326**;
- макетная плата: **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

Если вы до сих пор не установили среду разработки Arduino IDE, сделайте это сейчас. Самую свежую версию среды можно скачать по адресу: **http://www.arduino.cc/en/Main/Software**.

Затем установите библиотеку для управления OLED-дисплеем на SSD1306. Я рекомендую скачать ее при помощи Менеджера библиотек Arduino из репозитория: **https://github.com/squix78/esp8266-oled-ssd1306**.

Сборка схемы

Приступим к сборке проекта (рис. 9.5):

1. Установите модуль ESP8266 на макетную плату рядом с OLED-дисплеем.
2. Подключите к дисплею питание: соедините вывод Vin дисплея с выводом 3,3 В модуля и вывод GND дисплея с выводом GND модуля.
3. Теперь подключите к ESP8266 выводы I^2C дисплея. Для этого соедините вывод Data (SDA) дисплея с выводом 14 ESP8266 и вывод CLK (SCL) с выводом 12 ESP8266.

Учтите, что некоторые OLED-дисплеи могут быть настроены для работы как по протоколу SPI, так и I^2C. Убедитесь, что ваш дисплей настроен для использования протокола I^2C. В зависимости от марки дисплея, для соединений может потребоваться пайка.

[3] При использовании платы NodeMCU этот модуль не понадобится. — *Прим. пер.*

4. Наконец, соедините вывод RST дисплея с выводом 2 ESP8266[4]. Я установил, что это соединение требуется только в случае проблем со сбросом дисплея. Но для большей надежности все-таки подключите этот вывод.

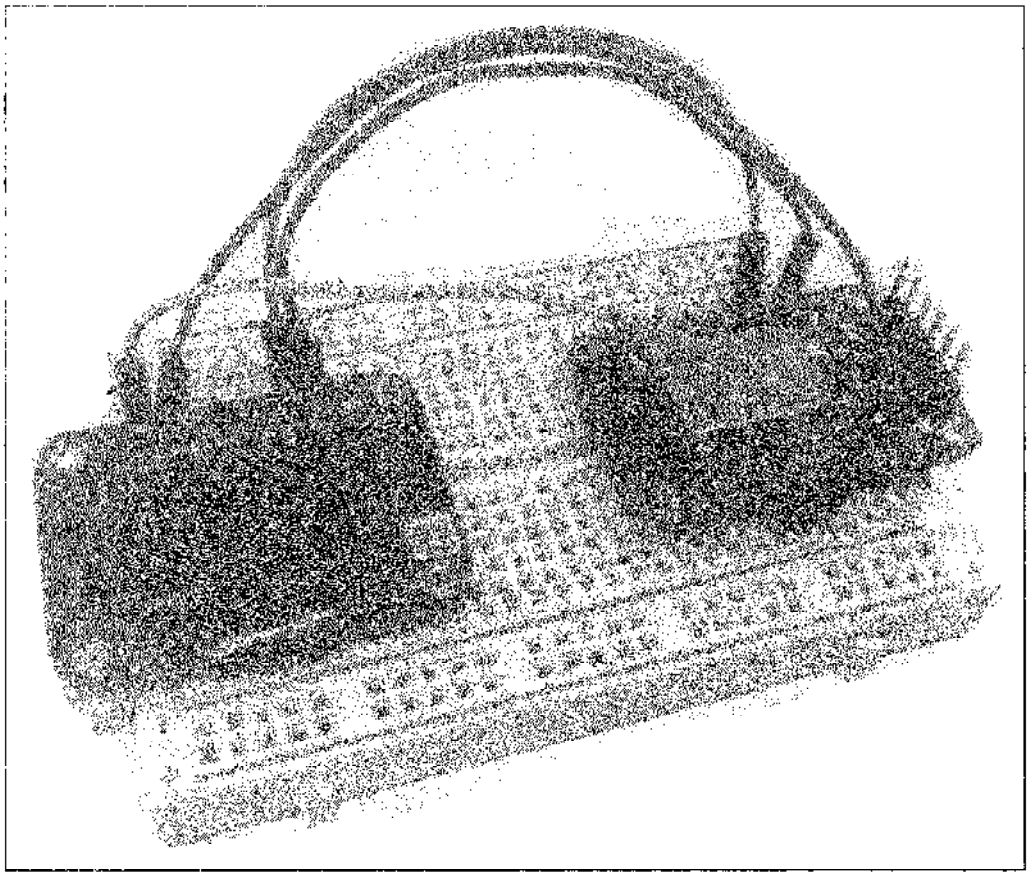


Рис. 9.5. OLED-дисплей, подключенный к модулю ESP8266

Тестирование тикера

Приступим к программированию проекта, но начнем с детального рассмотрения кода программы.

Программа начинается с подключения необходимых библиотек[5]:

```
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include "SSD1306.h"
```

[4] В дисплее нового типа, который мы рекомендуем (см. рис. Пр.4 в *предисловии к русскому изданию*), вывод RST не используется. Сброс дисплея происходит автоматически. — *Прим. пер.*

[5] Менеджер библиотек Arduino предлагает несколько библиотек для SSD1306. Вам следует выбрать библиотеку **ESP8266 and ESP32 Oled Driver for SSD1306 display**. — *Прим. пер.*

Затем мы определяем номера выводов, к которым подключен дисплей:

```
#define SDA 14
#define SCL 12
#define I2C 0x3D
```

Нам нужно создать объект OLED-дисплея:

```
SSD1306 display(I2C, SDA, SCL);
```

Для получения курса биткоина мы должны использовать API сайта Coindesk. Следовательно, нам нужно задать адрес этого API в коде:

```
const char* host = "api.coindesk.com";
```

Кроме того, следует модифицировать код, подставив имя и пароль своей точки доступа Wi-Fi:

```
const char* ssid = "wifi-network";
const char* password = "wif-password";
```

Внутри функции `setup()` мы инициализируем дисплей:

```
display.init();
display.flipScreenVertically();
display.clear();
display.display();
```

Оставаясь в этой функции, подключаем нашу плату к сети Wi-Fi:

```
Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
```

В функции `loop()` мы сначала соединяемся с сервером API:

```
WiFiClient client;
constinthttpPort = 80;
if (!client.connect(host, httpPort)) {
  Serial.println("connection failed");
  return;
}
```

Затем готовим адресную строку с запросом, который будет отправляться на сервер. Как вы уже видели ранее в этой главе, мы обращаемся по такому адресу:

```
String url = "/v1/bpi/currentprice.json";
```

После этого отправляем запрос:

```
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
"Host: " + host + "\r\n" +
"Connection: close\r\n\r\n");
```

Когда запрос отправлен, мы полностью считываем ответ от сервера:

```
String answer;
while(client.available()){
String line = client.readStringUntil('\r');
answer += line;
}
client.stop();
Serial.println();
Serial.println("closing connection");
```

Теперь, когда у нас есть ответ сервера, пришло время обработать этот ответ. Обработка начинается с извлечения данных в формате JSON из исходного ответа:

```
String jsonAnswer;
int jsonIndex;
for (int i = 0; i < answer.length(); i++) {
  if (answer[i] == '{') {
    jsonIndex = i;
    break;
  }
}
```

Затем мы извлекаем объект JSON и сохраняем его в строковой переменной:

```
jsonAnswer = answer.substring(jsonIndex);
Serial.println();
Serial.println("JSON answer: ");
Serial.println(jsonAnswer);
jsonAnswer.trim();
```

Оттуда мы можем извлечь курс биткоина в долларах США, выполнив операции над строкой, содержащей объект JSON:

```
intrateIndex = jsonAnswer.indexOf("rate_float");
String priceString = jsonAnswer.substring(rateIndex + 12, rateIndex +
                     18);
priceString.trim();
float price = priceString.toFloat();
```

Наконец, для отладки программы мы выводим значение курса в терминал:

```
Serial.println();
Serial.println("Bitcoin price: ");
Serial.println(price);
```

А также выводим курс на экран — по центру и крупными символами:

```
display.clear();
display.setFont(ArialMT_Plain_24);
display.drawString(26, 20, priceString);
display.display();
```

Мы не хотим, чтобы запросы следовали непрерывно, один за другим, поэтому задаем задержку между запросами:

```
delay(5000);
```

Теперь пора проверить скетч! Вы можете скачать его полный исходный код из репозитория GitHub: **https://github.com/openhomeautomation/iot-esp8266-packt**.

Готовый к использованию файл этой программы находится также в папке ch9_TICKER сопровождающего книгу электронного архива (см. *приложение*).

После извлечения программы исправьте имя и пароль своей точки доступа Wi-Fi и загрузите прошивку в плату.

Проделав это, откройте монитор последовательного порта. Вы должны сразу увидеть, что чип ESP8266 получает ответ сервера Coindesk (рис. 9.6).

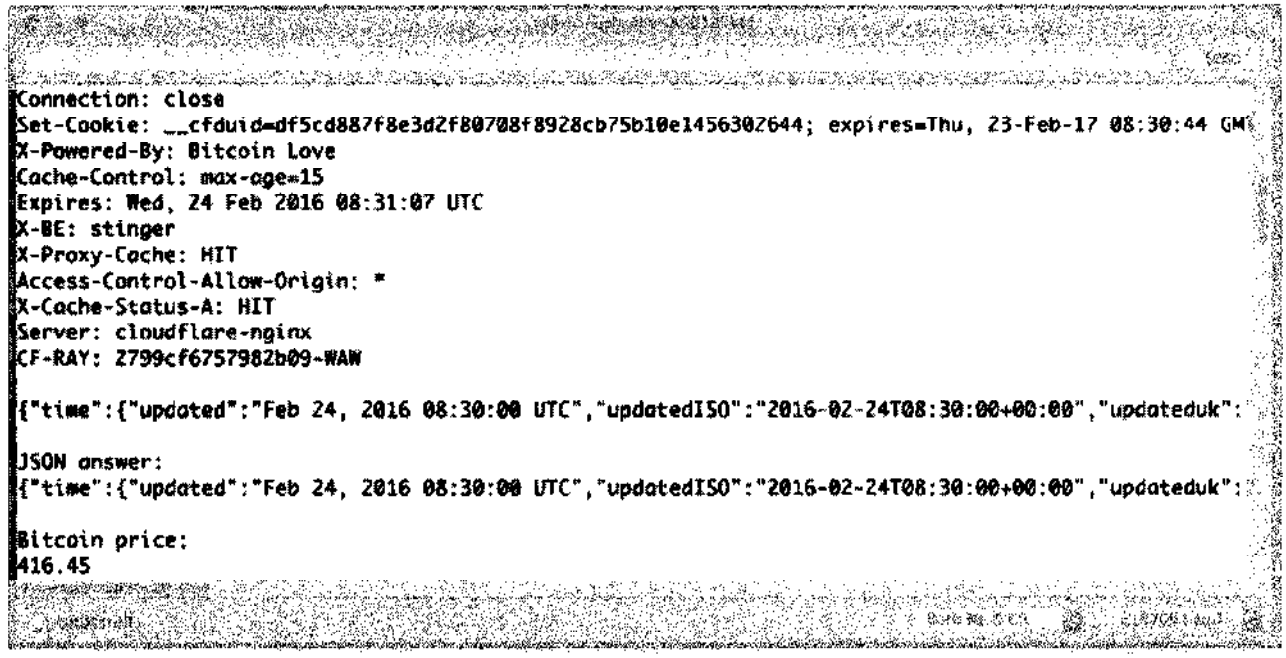


Рис. 9.6. Ответ сервера Coindesk

Как можно видеть, когда я тестировал этот проект, курс биткоина составлял 416,45 доллара США. Конечно же, этот курс должен отобразиться и на экране OLED-дисплея (рис. 9.7).

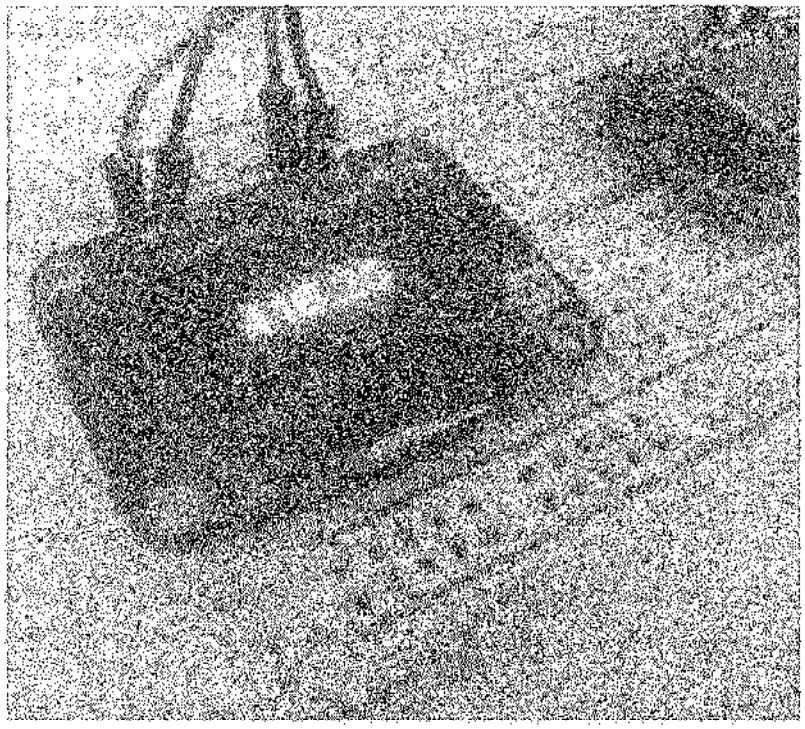


Рис. 9.7. Курс биткоина на OLED-дисплее

Примите мои поздравления, вы только что изготовили собственный физический тикер курса биткоина!

Добавляем в тикер светодиоды

Мы реализовали самый сложный из проектов — построили физический тикер, который показывает курс биткоина в реальном времени.

Теперь мы готовы усовершенствовать этот проект, чтобы сделать его еще лучше. Например, мы добавим в него два светодиода: красный и зеленый — для индикации, растет или падает курс. Красный светодиод будет зажигаться, когда курс падает, и зеленый — когда растет.

Но сначала мы должны добавить в наш проект новые аппаратные компоненты. Установите светодиоды последовательно с резисторами 330 Ом на макетную плату и подключите их, как было показано в предыдущих главах. Удостоверьтесь, что подключили красный светодиод к выводу 5, а зеленый — к выводу 4 платы ESP8266 (рис. 9.8).

Давайте посмотрим, как запрограммировать этот проект (его полный код вы найдете в репозитории GitHub для этой книги). Здесь я отмечу только различия с предыдущим проектом:

1. Сначала мы должны объявить номера выводов, к которым подключены светодиоды:

```
#define LED_PIN_UP 4
#define LED_PIN_DOWN 5
```

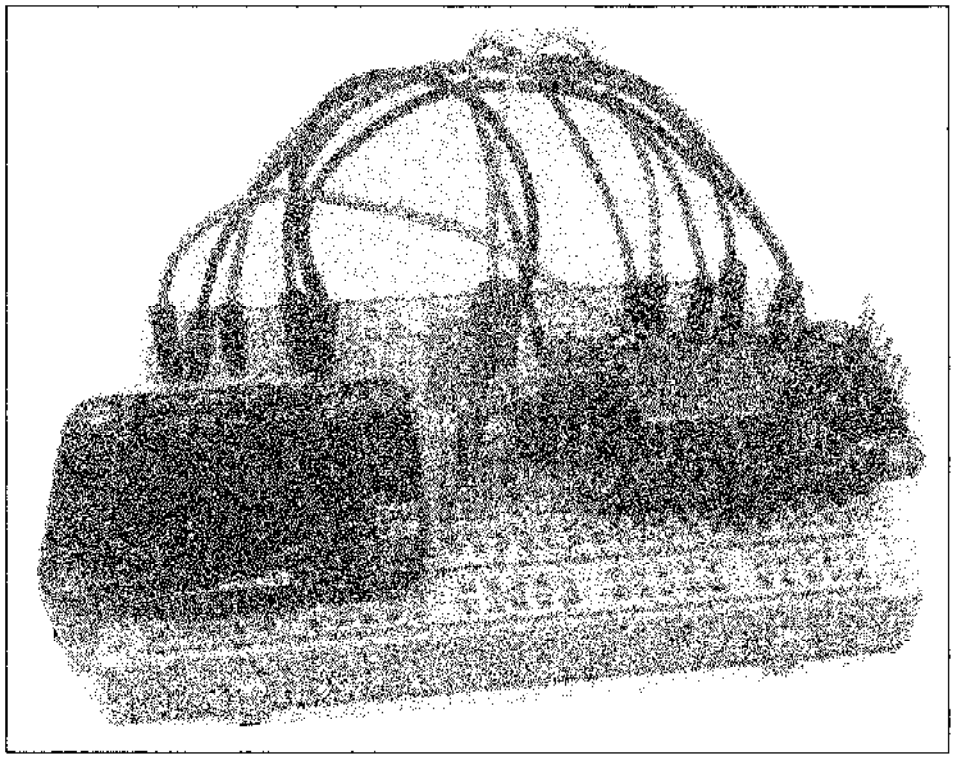


Рис. 9.8. Тикер биткоинов с сигнальными светодиодами

2. Затем мы объявляем переменную для хранения предыдущего значения курса и порог изменения в долларах США, по достижении которого мигает светодиод:

```
float previousValue = 0.0;
float threshold = 0.05;
```

3. В функции `setup()` определяем выводы светодиодов, как выходы:

```
pinMode(LED_PIN_DOWN, OUTPUT);
pinMode(LED_PIN_UP, OUTPUT);
```

4. Внутри функции `loop()` проверяем, запущен ли код впервые. Если да, присваиваем переменной предыдущего курса текущий курс биткоина:

```
if (previousValue == 0.0) {
previousValue = price;
}
```

5. Затем проверяем, не упал ли курс ниже порогового значения. В этом случае мы заставляем вспыхивать красный светодиод:

```
if (price < (previousValue - threshold)) {
digitalWrite(LED_PIN_DOWN, HIGH);
delay(100);
digitalWrite(LED_PIN_DOWN, LOW);
delay(100);
digitalWrite(LED_PIN_DOWN, HIGH);
delay(100);
digitalWrite(LED_PIN_DOWN, LOW);
}
```

6. То же самое мы делаем с зеленым светодиодом, если курс растет:

```
if (price > (previousValue + threshold)) {
digitalWrite(LED_PIN_UP, HIGH);
delay(100);
digitalWrite(LED_PIN_UP, LOW);
delay(100);
digitalWrite(LED_PIN_UP, HIGH);
delay(100);
digitalWrite(LED_PIN_UP, LOW);
}
```

7. В конце цикла сохраняем текущее значение в переменную предыдущего курса:

```
previousValue = price;
```

Готовый к использованию файл этой программы находится в папке ch9_TICKER\ch9_TICKER_LED сопровождающего книгу электронного архива (см. *приложение*).

И вновь протестируйте проект, загрузив прошивку в плату. Сразу после этого вы должны увидеть, что если отклонение курса вверх или вниз превышает пороговое значение, один из светодиодов быстро вспыхивает.

Заключение

Давайте подведем итог достигнутого в этом проекте. Мы использовали чип ESP8266 для получения курса биткоина через веб-API и вывели этот курс на OLED-дисплей, подключенный к ESP8266. Затем мы добавили в проект два светодиода, чтобы видеть, растет или падает курс.

Конечно, в этом проекте можно реализовать множество и других опций. Вы можете, например, выводить курс биткоина в валютах, отличных от доллара США, и добавить кнопку для переключения валют.

В следующей главе мы узнаем, как использовать ESP8266 для слежения за влажностью почвы и управления поливом грядки (или целого сада!) через облако.

10

Сетевое облачное садоводство

В этой главе мы снова собираемся использовать полученные ранее знания в новом проекте — организовать на основе ESP8266 сетевое садоводство, управляемое через облако. Это позволит нам удаленно следить за температурой и влажностью почвы у отдельного растения или целого сада и вовремя поливать его.

Мы намерены сначала настроить проект так, чтобы он отправлял нам автоматическое уведомление, когда почва становится слишком сухой. Затем мы доработаем его таким образом, чтобы не только удаленно контролировать сад, но и включать при необходимости насос полива. Давайте начнем!

Оборудование и программное обеспечение

Кроме модуля ESP8266, ключевым компонентом этого проекта является датчик температуры и влажности. Поскольку он для замеров должен быть погружен в почву, мы не сможем использовать тот обычный датчик, который применяли в проектах этой книги ранее, — вам потребуется датчик, который подходит для погружения в почву. Поэтому я воспользуюсь для нашего проекта датчиком на основе сенсора SHT10[1] от компании Sensirion, который продает компания Adafruit (рис. 10.1).

Для подключения этого датчика к модулю ESP8266 понадобится резистор 10 кОм.

Нам также потребуется каким-то образом поливать растение или сад по мере необходимости. Для этого я просто использовал реле с обмоткой на 5 В, которое раньше встречалось в этой книге, — с его помощью вы сможете управлять большинством поливочных насосов, имеющихся в продаже.

Понадобится нам и беспаечная макетная плата с комплектом соединительных проводов.

---

[1] Особенностью конструкции этого датчика является оболочка из спеченных металлических микрогранул. — *Прим. пер.*

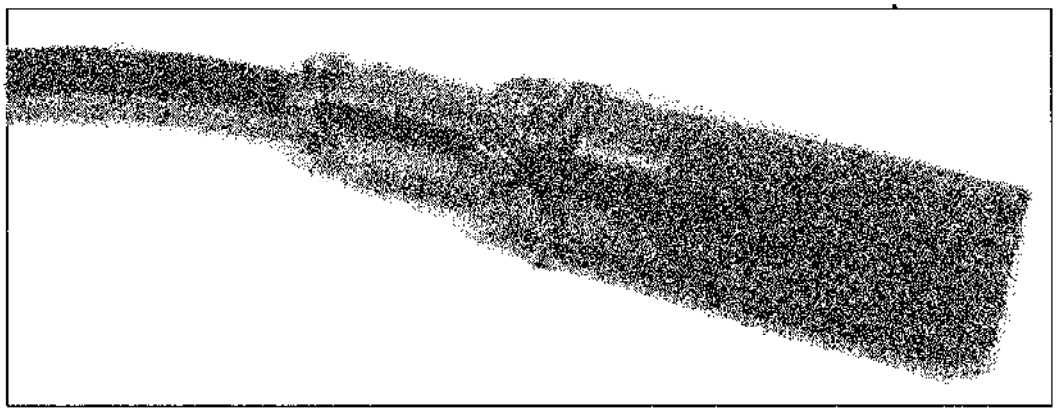


Рис. 10.1. Датчик температуры и влажности почвы

Вот конкретный перечень использованных в проекте компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266: **https://www.adafruit.com/products/2471**;
- модуль FTDI USB: **https://www.adafruit.com/products/284**;
- реле: **https://www.pololu.com/product/2480**;
- резистор 10 кОм: **https://www.sparkfun.com/products/8374**;
- макетная плата: **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

Для программы потребуется библиотека поддержки датчика SHT10, которую можно скачать по адресу: **https://github.com/practicalarduino/SHT1x**.

Сборка схемы

Прежде чем заняться соединением различных частей проекта воедино, разберитесь с назначением выводов датчика SHT10 (рис. 10.2)[2], после чего вставьте разъем датчика в макетную плату так, чтобы выводы датчика VCC и GND были соединены с красной и синей шинами питания макетной платы соответственно.

Подключите также к красной и синей шинам питания макетной платы выводы VCC и GND платы ESP866. Затем соедините вывод Data датчика с выводом GPIO4 (D2) платы ESP8266 и вывод Clock датчика — с выводом GPIO5 (D1) платы. Наконец, добавьте подтягивающий резистор 10 кОм между выводами Data и VCC датчика.

Для подключения реле достаточно соединить вывод GND реле с синей шиной питания платы, вывод VCC реле — с красной шиной питания платы и вывод SIG (IN) реле с выводом GPIO15 (D8) платы ESP8266.

Собранная схема изображена на рис. 10.3.

---

[2] На монохромной книжной иллюстрации не виден цвет проводов: Data — синий, Clock — желтый, VCC — красный, GND — зеленый. — *Прим. пер.*

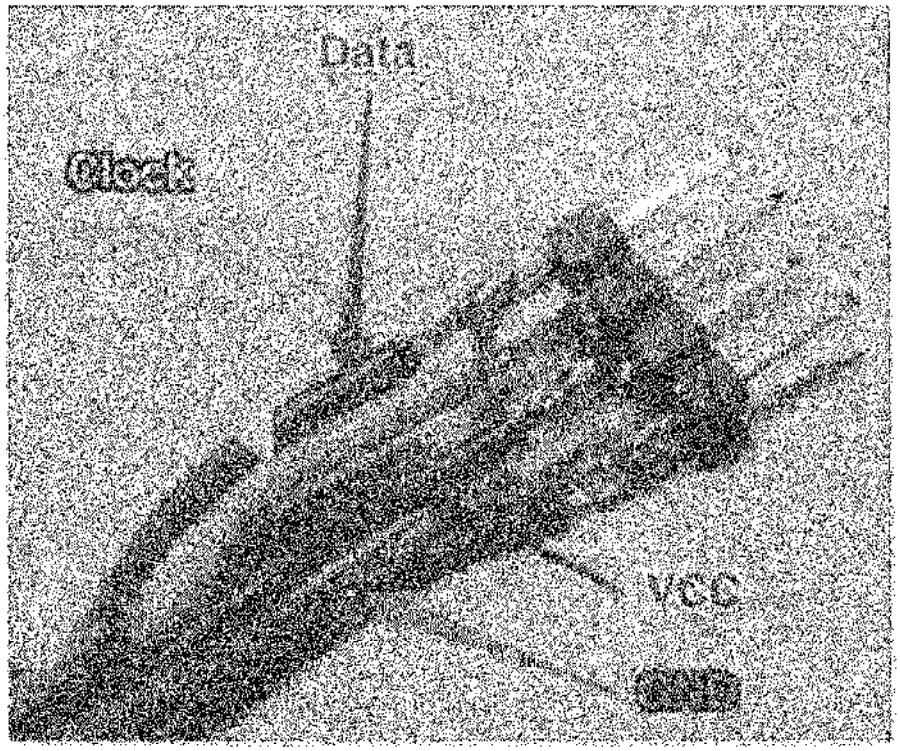


Рис. 10.2. Назначение выводов датчика SHT10

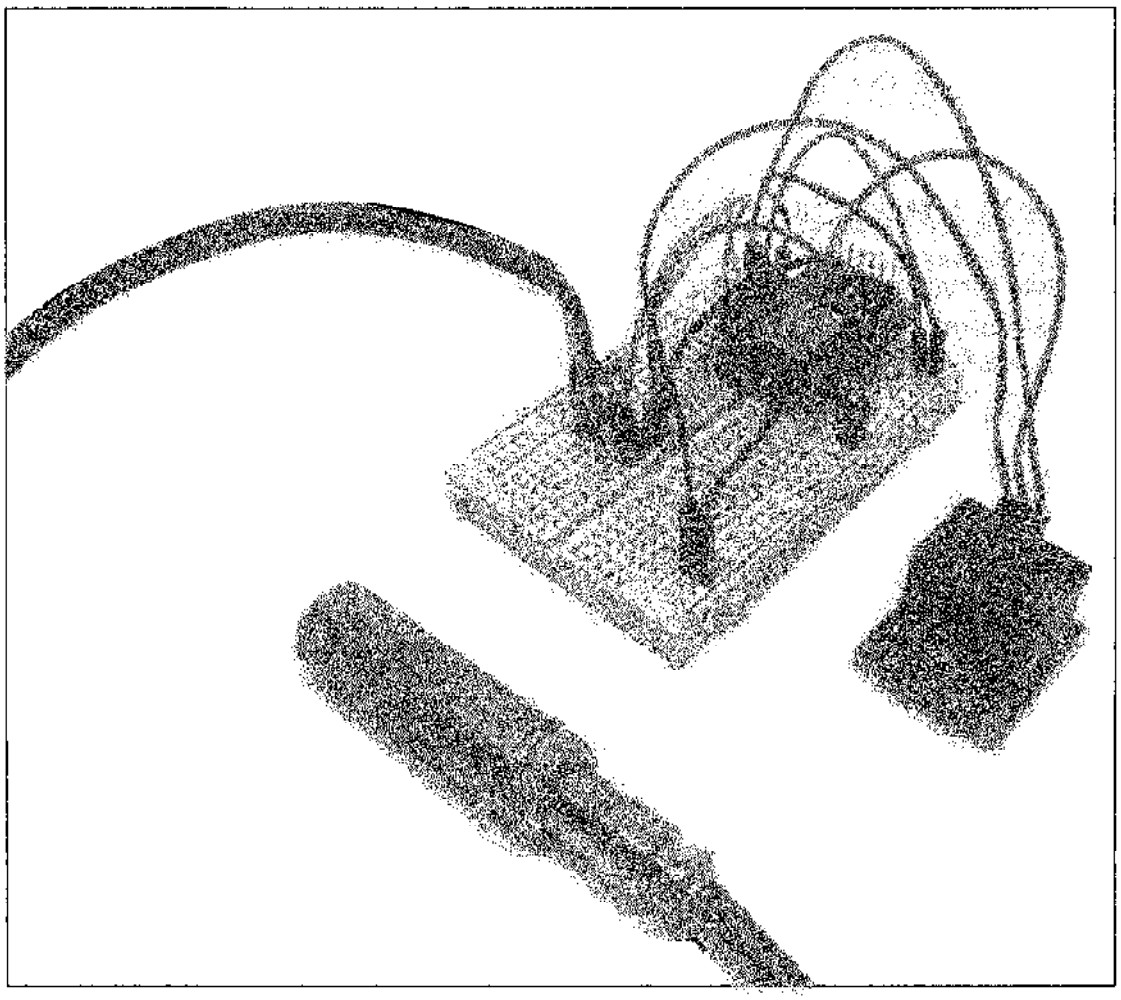


Рис. 10.3. Собранная схема измерителя температуры и влажности почвы

Но мы еще не закончили — нам нужен объект измерений! Это, например, может быть цветочный горшок у вас дома или грядка в саду. Теперь вы можете погрузить датчик в почву. Смотрите, как я установил его для наблюдения за растением у себя дома (рис. 10.4).

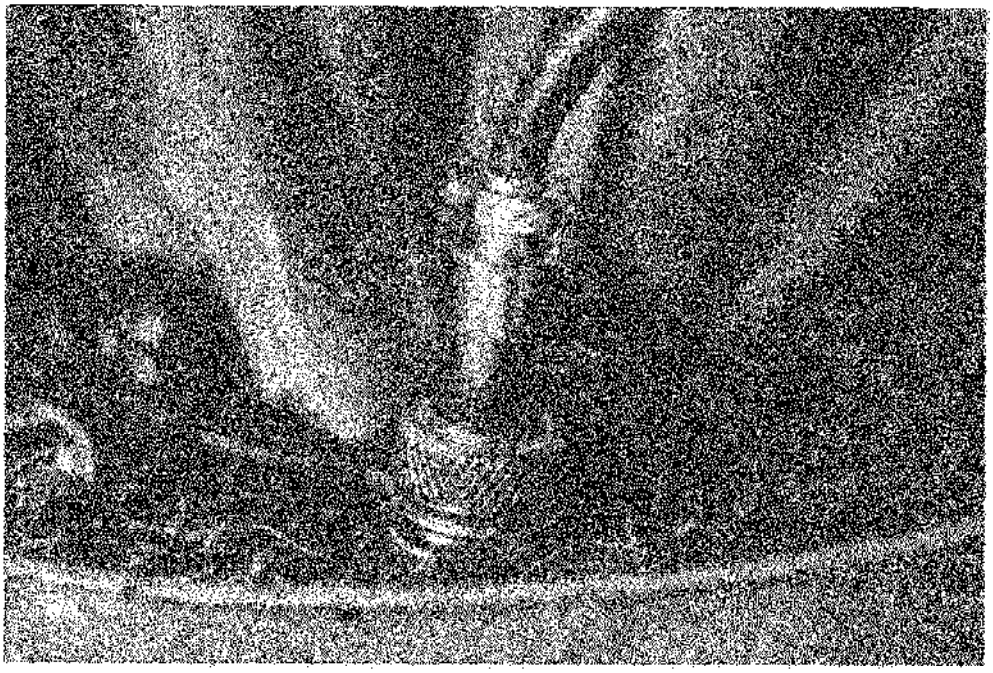


Рис. 10.4. Погрузите датчик SHT10 в почву возле корней растения

Создаем уведомление о поливе растения

Прежде, чем создавать захватывающие проекты в области удаленного садоводства, которые вы найдете в этой главе, следует начать с одного важного момента — проверки правильности работы датчика. В листинге 10.1 приведен скетч, обеспечивающий эту процедуру.

Листинг 10.1. Скетч, проверяющий датчик SHT10

```
// Подключаем библиотеку
#include <SHT1x.h>
// Определяем выводы
#define dataPin 4
#define clockPin 5
// Создаем объект датчика
SHT1xsht1x(dataPin, clockPin);

void setup()
{
  Serial.begin(115200); // Открываем последовательный порт
  Serial.println("Starting up");
}
```

```
void loop()
{
  // Переменные
  float temp_c;
  float temp_f;
  float humidity;
  // Читаем данные из датчика
  temp_c = sht1x.readTemperatureC();
  temp_f = sht1x.readTemperatureF();
  humidity = sht1x.readHumidity();
  // Выводим значения в терминал порта
  Serial.print("Temperature: ");
  Serial.print(temp_c, DEC);
  Serial.print("C / ");
  Serial.print(temp_f, DEC);
  Serial.print("F. Humidity: ");
  Serial.print(humidity);
  Serial.println("%");
  // Пауза 2 секунды
  delay(2000);
}
```

Этот скетч предельно прост — мы создаем объект датчика SHT10, внутри функции `loop()` производим измерения и выводим результаты в окно монитора последовательного порта.

Готовый к использованию файл скетча находится в папке ch10_SENSOR_TEST сопровождающего книгу электронного архива (см. *приложение*).

Просто скопируйте этот скетч в Arduino IDE, переведите ESP8266 в режим программирования и загрузите прошивку в плату. Затем откройте монитор последовательного порта. Вот что вы должны видеть в окне монитора (рис. 10.5).

Если вы видите осмысленные данные (например, температуру воздуха в комнате), это означает, что датчик работает правильно. Если нет, еще раз проверьте все соединения и убедитесь, что подключили резистор 10 кОм между выводами Data и VCC датчика. В моем случае именно это стало причиной того, что датчик не начал работать с первой попытки.

Теперь займемся настройкой автоматического оповещения, поступающего на ваш смартфон, когда почва у растения становится слишком сухой. Первым шагом является создание аккаунта в сервисе IFTTT (**https://ifttt.com/**), который мы уже использовали в этой книге.

Я полагаю, что у вас уже есть этот аккаунт и в него добавлены каналы **Maker** и **Pushover**. Если нет, обратитесь к *главам 6* и *7*, где это подробно описано.

Далее создайте новое правило в канале **Maker** (рис. 10.6).

В качестве имени события введите слово `alert` (рис. 10.7).

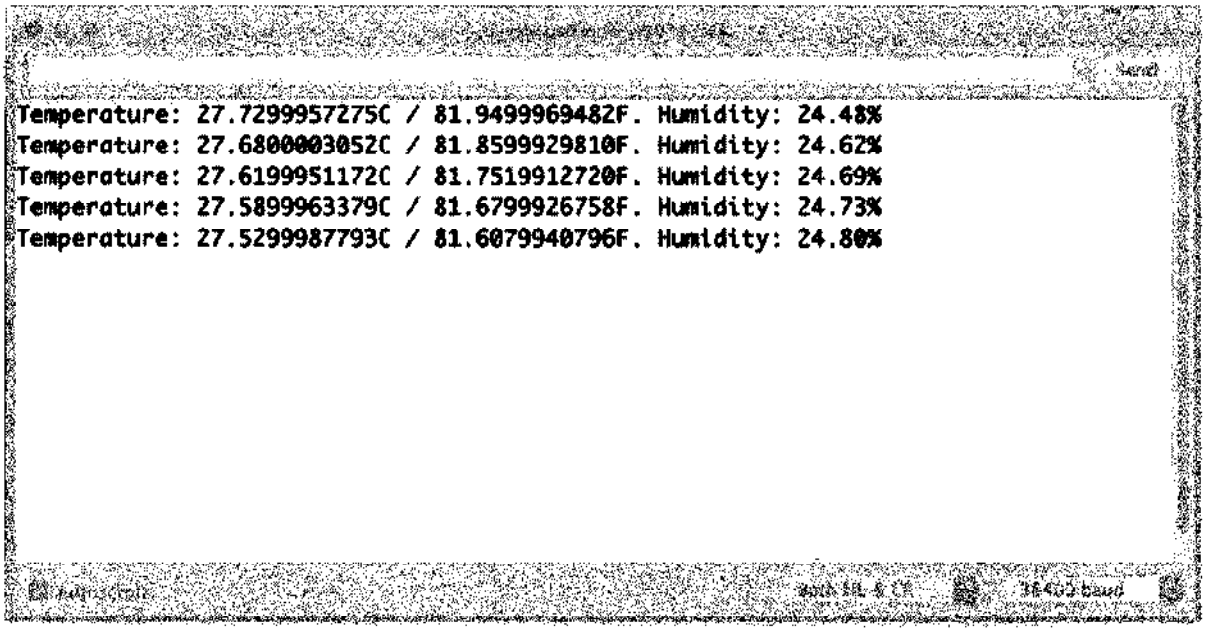


Рис. 10.5. Результаты измерения температуры и влажности

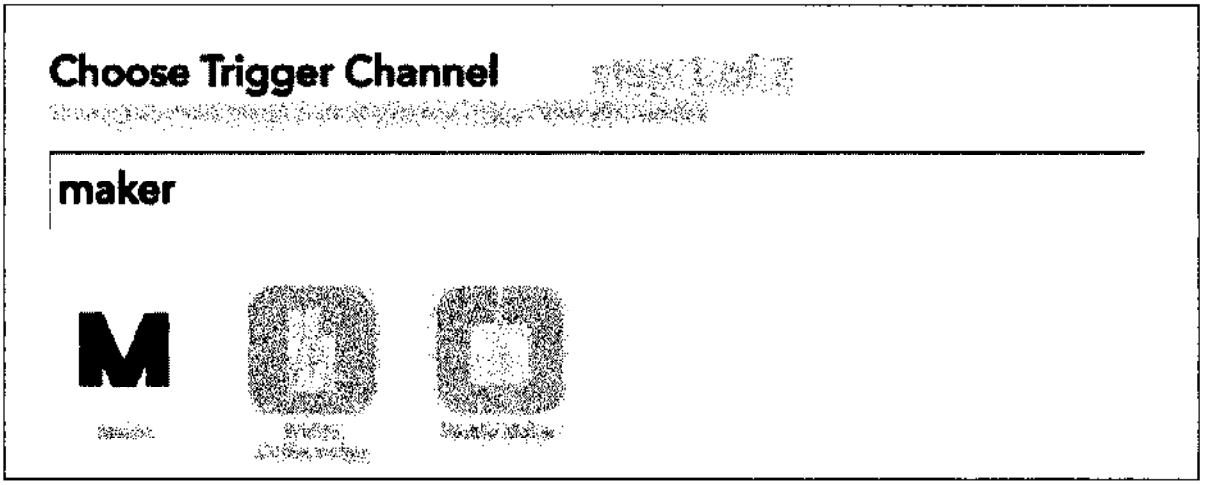


Рис. 10.6. Выбираем канал **Maker**

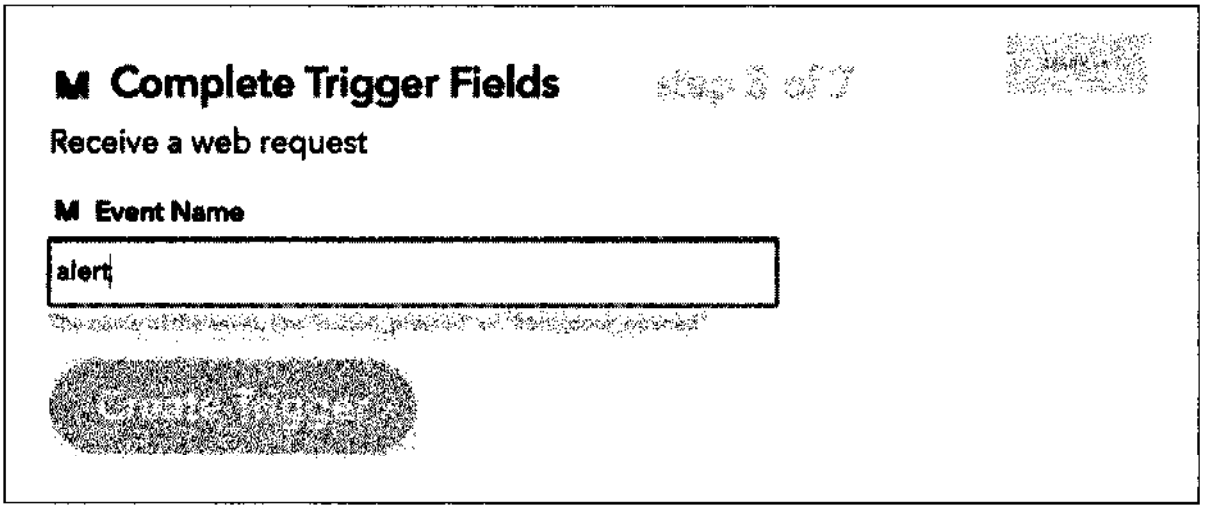


Рис. 10.7. Вводим имя события

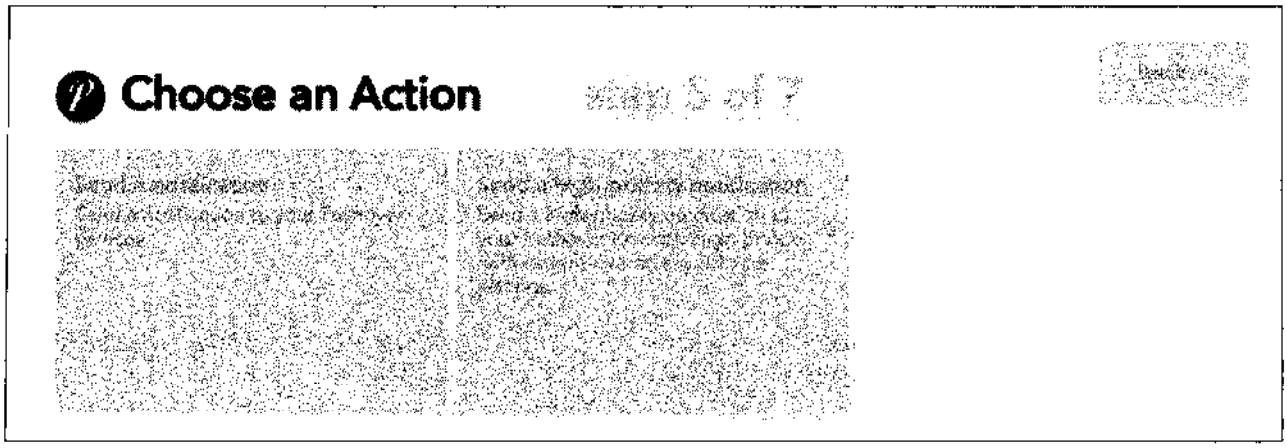


Рис. 10.8. Подключаем сервис push-уведомлений **Pushover**

В качестве исполнительного канала мы воспользуемся сервисом Pushover, который позволит получать уведомления на ваше мобильное устройство (рис. 10.8).

Теперь введите текст уведомления о том, что влажность грядки низка и пора ее полить, в поле для сообщения (рис. 10.9).

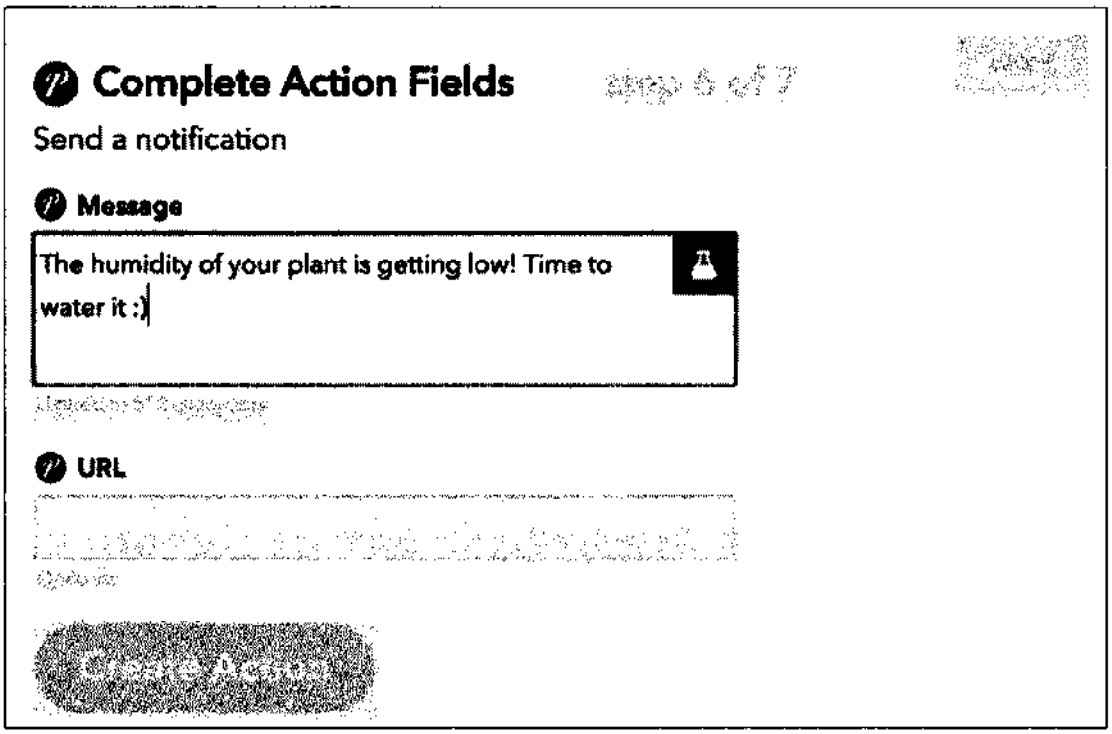


Рис. 10.9. Вводим текст уведомления

Правило создано, теперь мы должны запрограммировать плату на автоматическую отправку предупреждений каждый раз, когда влажность почвы под растением становится слишком низкой. Поскольку в данном случае код достаточно объемен, я поясню лишь его основные моменты.

Программа начинается с подключения библиотек:

```
#include <ESP8266WiFi.h>
#include <SHT1x.h>
```

Затем нам следует задать имя и пароль точки доступа Wi-Fi:

```
const char* ssid = "wifi-name";
const char* password = "wifi-pass";
```

Мы также задаем пороговое значение влажности, ниже которого проект будет автоматически отправлять предупреждение:

```
float threshold = 30.00;
```

Затем задаем параметры доступа к сервису IFTTT. Здесь вы должны подставить свой ключ доступа к каналу IFTTT **Maker**:

```
const char* host = "maker.ifttt.com";
const char* eventName = "alert";
const char* key = "ifttt-key";
```

После получения данных из датчика в функции `loop()` скетча мы проверяем, не упала ли влажность ниже порогового значения:

```
if (humidity < threshold) {
```

В таком случае мы готовим запрос для отправки на сервер IFTTT:

```
String url = "/trigger/";
url += eventName;
url += "/with/key/";
url += key;
```

А затем этот запрос отправляем:

```
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
             "Host: " + host + "\r\n" +
             "Connection: close\r\n\r\n");
int timeout = millis() + 5000;
while (client.available() == 0) {
  if (timeout - millis() < 0) {
    Serial.println(">>> Client Timeout !");
    client.stop();
    return;
  }
}
```

Если запрос отправлен, мы выдерживаем длинную паузу (10 минут) — чтобы не получать уведомления от проекта беспрерывно:

```
delay(10 * 60 * 1000);
```

Настало время проверить первый проект этой главы! Скачайте полный код программы и исправьте данные для доступа, включая настройки IFTTT. Затем загрузите прошивку в плату. Если влажность почвы опустится ниже порогового значения, то вслед за этим вы получите уведомление о том, что растение нуждается в поливе (рис. 10.10).

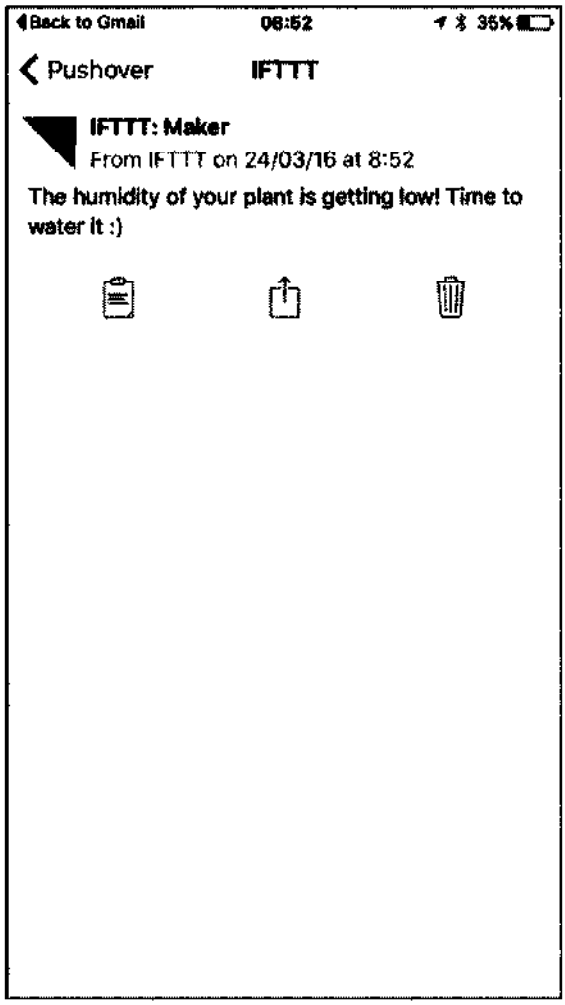


Рис. 10.10. Уведомление о низкой влажности почвы

Готовый к использованию файл программы находится в папке ch10_ALERT сопровождающего книгу электронного архива (см. *приложение*).

Наблюдение за температурой и влажностью

Во втором проекте этой главы мы продвинемся еще дальше и станем наблюдать за температурой и влажностью почвы грядки при помощи облачной приборной панели, которую уже использовали раньше в этой книге. Сначала мы займемся программированием платы, а затем настройкой приборной панели.

Ранее в этой книге уже было показано, как использовать облачную платформу aREST. Поэтому я поясню только наиболее важные части программы.

На первом шаге мы подключаем необходимые библиотеки, включая библиотеку aREST:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
#include <SHT1x.h>
```

Затем мы назначаем идентификатор нашего устройства:

```
char* device_id = "gveie2y5";
```

Как обычно, вам надо ввести имя и пароль вашей точки доступа Wi-Fi:

```
const char* ssid = "your-wifi";
const char* password = "your-password";
```

В функции `loop()` этого скетча мы получаем данные измерений от датчика и обрабатываем подключение к облачной платформе aREST:

```
// Read values from the sensor
temperature = sht1x.readTemperatureC();
humidity = sht1x.readHumidity();
// Connect to the cloud
rest.handle(client);
```

Теперь пора запрограммировать плату. Скачайте код из репозитория GitHub для этой книги и удостоверьтесь, что исправили в нем учетные данные и идентификатор устройства.

Готовый к использованию файл программы находится в папке ch10_MONITORING сопровождающего книгу электронного архива (см. *приложение*).

Затем перейдите по адресу: **http://dashboard.arest.io/**. Если вы еще не создали здесь свою учетную запись, сделайте это сейчас и создайте для нее новую приборную панель (рис. 10.11).

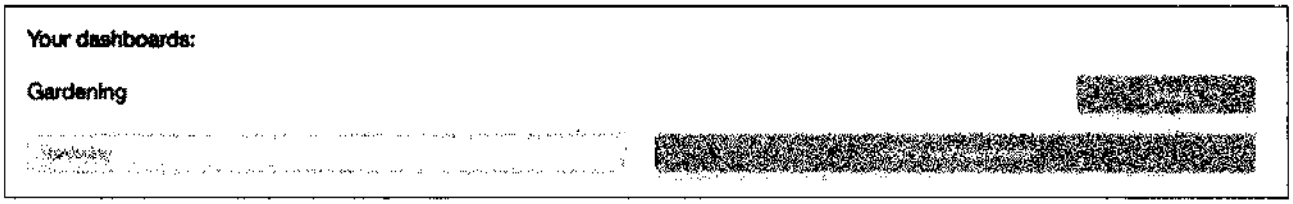


Рис. 10.11. Создаем новую приборную панель

Внутри новой приборной панели создайте индикатор значения переменной величины с параметрами, как показано на рис. 10.12.

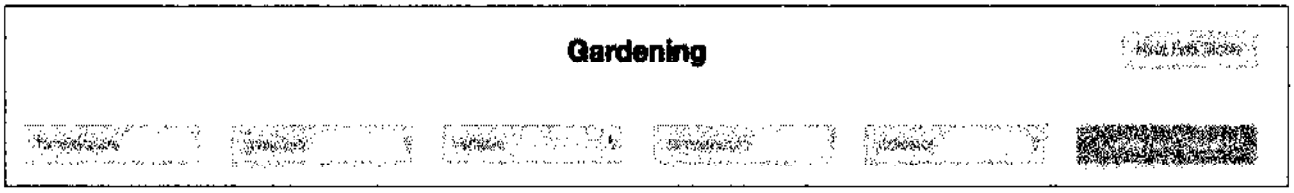


Рис. 10.12. Создаем индикатор значения переменной величины

Вы должны немедленно увидеть текущее значение, поступившее из платы (рис. 10.13).

Повторите этот шаг для измеренного значения влажности (рис. 10.14).

Теперь вы можете следить за температурой и влажностью почвы своего растения, грядки или сада из любого места.

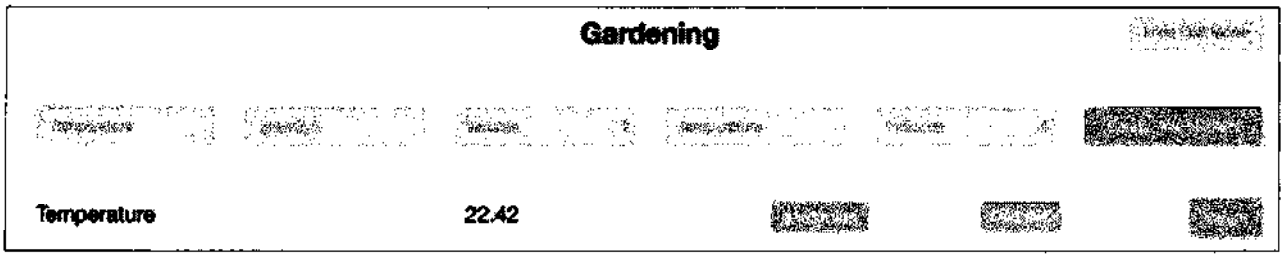


Рис. 10.13. Текущее значение температуры почвы

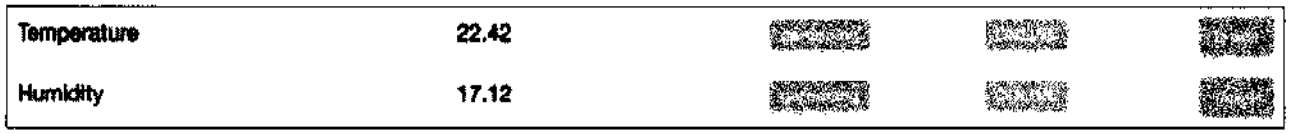


Рис. 10.14. Текущие значения температуры и влажности почвы

Но у нас по-прежнему отсутствует один ключевой элемент — полив. Разве не здорово иметь возможность дистанционно включить водяной насос, когда вы видите, что влажность почвы падает? Или даже автоматически включать его, когда влажность слишком мала? Именно этим мы займемся в следующем разделе главы.

Автоматизация садоводства

Приступим к настройке проекта, который будет автоматически поливать растение, если влажность почвы упадет ниже порогового значения.

На первом шаге мы задаем два пороговых значения:

```
floatlowThreshold = 20.00;
floathighThreshold = 25.00;
```

Эти два порога нужны, т. к. если мы определим только один, помпа будет постоянно включаться и выключаться.

Поэтому мы станем включать помпу, если влажность упадет ниже значения `floatlowThreshold`, и выключать ее, когда влажность превысит значение `floathighThreshold`.

Далее определяем, к какому выводу будет подключено реле:

```
#define relayPin 15
```

В функции `setup()` скетча объявляем вывод реле как выход:

```
pinMode(relayPin, OUTPUT);
```

В функции `loop()` постоянно проверяем, не ушла ли влажность ниже минимального порога или выше максимального:

```
if (humidity < lowThreshold) {
  // Activate pump
  digitalWrite(relayPin, HIGH);
}
```

```
if (humidity > highThreshold) {
  // Deactivate pump
  digitalWrite(relayPin, LOW);
}
```

Здесь я отметил только основные фрагменты программы, но вы, конечно, можете скачать полный исходный код из репозитория GitHub для этой книги. Получите этот код и исправьте в нем параметры доступа, указав свои данные. Затем загрузите прошивку в плату.

Готовый к использованию файл программы находится в папке ch10_AUTOMATED сопровождающего книгу электронного архива (см. *приложение*).

Закончив с этим, вернитесь к приборной панели, которую создали раньше, и добавьте еще один элемент для управления насосом с набором параметров, как показано на рис. 10.15.

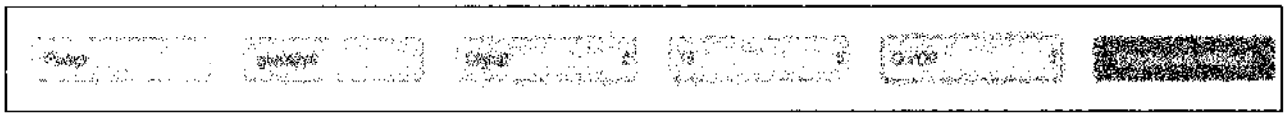


Рис. 10.15. Элемент управления насосом

Вы должны сразу же увидеть кнопки управления и текущий статус реле (рис. 10.16).

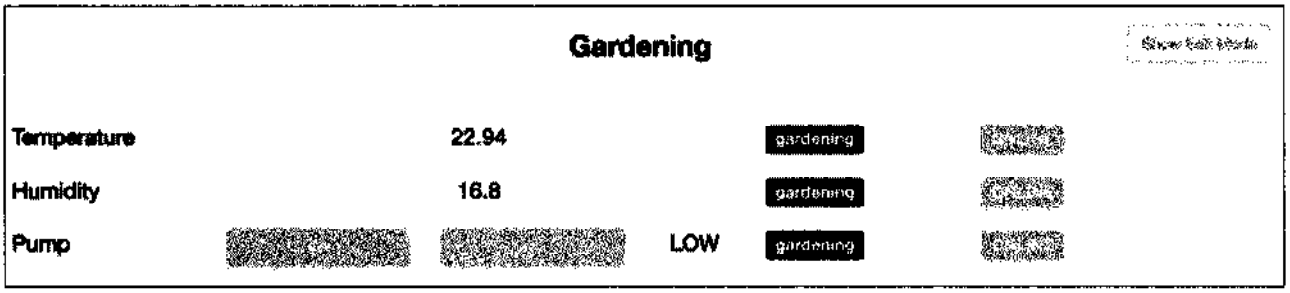


Рис. 10.16. Приборная панель с кнопками управления насосом

Теперь вы можете нажать на кнопку **On**, и насос должен немедленно включиться (рис. 10.17).

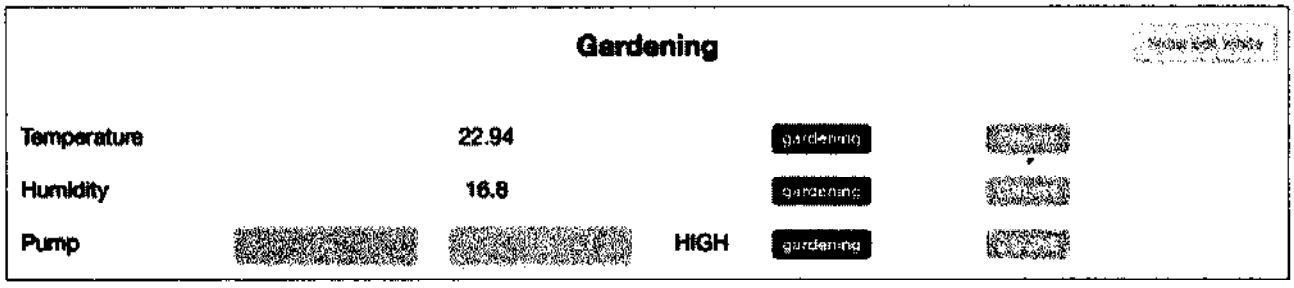


Рис. 10.17. Поливочный насос включен

Кроме того, вы также должны получать уведомления каждый раз, когда влажность почвы опускается ниже минимального порога. При этом насос должен сразу включаться и работать до тех пор, пока не будет пройден верхний порог влажности.

Заключение

В этой главе мы создали проект удаленного облачного садоводства на основе микросхемы ESP8266. Вы узнали, как создавать уведомления о состоянии почвы у своих растений или в саду, как следить за ними откуда угодно, и как удостовериться, что насос автоматически включился, когда влажность стала слишком низкой.

Разумеется, на основе знаний, полученных в этой главе, вы можете пойти намного дальше. Например, добавить в проект больше компонентов и следить за несколькими растениями в разных местах.

В следующей главе мы рассмотрим еще один проект, используя ранее полученные знания о ESP8266, — законченную систему домашней автоматики. Мы будем управлять этой системой и следить за ее состоянием через облако.

11

Домашняя автоматика и облачные сервисы

Здесь мы воспользуемся всем, что изучено нами ранее в этой книге, и на основе полученных знаний создадим простую, но комплектную, систему домашней автоматики, полностью управляемую через облако с помощью модулей ESP8266. Система будет включать в себя датчик движения, датчик температуры и влажности и регулятор яркости светодиода. Таким образом, она сможет имитировать все основные компоненты реальной системы домашней автоматики.

Мы подготовим три проекта, обеспечивающие функционирование этой системы. Сначала мы увидим, как контролировать каждый компонент системы при помощи онлайновой приборной панели. Затем разберемся, как отправлять автоматические оповещения, если в вашем доме обнаружено движение. Наконец, мы узнаем, как можно при помощи сервиса IFTTT и созданной вами системы удаленно управлять всей этой автоматикой. Давайте начнем!

Оборудование и программное обеспечение

Для проектов этой главы в основном потребуются компоненты, которые мы уже не раз в предыдущих проектах использовали: модуль ESP8266, светодиод и датчик DHT11. Единственным новым компонентом будет пассивный[1] инфракрасный датчик движения (PIR, passive infrared sensor), которым мы воспользуемся для обнаружения движения в вашем доме.

Я привел далее список всех компонентов для проектов этой главы, исходя из одного датчика движения, одного светодиода и одного датчика DHT11. Разумеется, если вы хотите использовать большее количество компонентов, это не проблема, — просто добавьте в проект больше модулей ESP8266.

---

[1] Он называется *пассивным*, потому что ничего не излучает сам и реагирует на тепловое излучение человеческого тела. — *Прим. пер.*

Вот конкретный перечень использованных в проекте компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266 (3 шт.): **https://www.adafruit.com/products/2471**;
- модуль FTDI USB[2]: **https://www.adafruit.com/products/284**;
- датчик DHT11: **https://www.adafruit.com/products/386**;
- светодиод: **https://www.sparkfun.com/products/9590**;
- резистор 330 Ом: **https://www.sparkfun.com/products/8377**;
- датчик движения: **https://www.sparkfun.com/products/13285**;
- макетная плата (3 шт.): **https://www.sparkfun.com/products/12002**;
- соединительные провода (3 комплекта): **https://www.sparkfun.com/products/9194**.

В части программного обеспечения вам понадобятся библиотеки aREST, PubSub и DHT. Мы уже установили эти библиотеки в предыдущих главах книги, но если вы еще не сделали этого, просто установите их при помощи Менеджера библиотек Arduino IDE.

Сборка схемы

Прежде всего, подключим модуль датчика движения. Разместите на первой макетной плате модуль ESP8266, соедините вывод VCC датчика движения с выводом VCC модуля, вывод GND датчика — с выводом GND модуля и, наконец, вывод OUT датчика — с выводом 5 (D1) модуля ESP8266 (рис. 11.1).

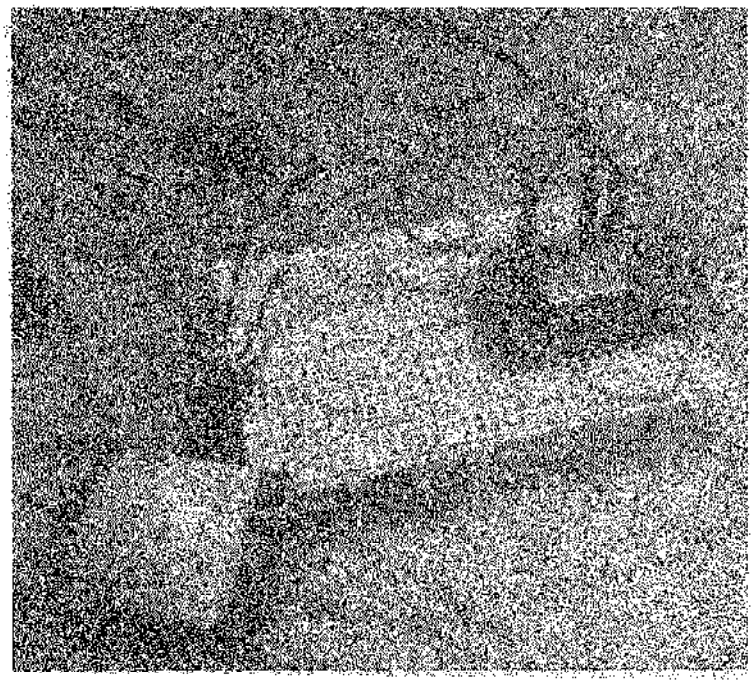


Рис. 11.1. Подключенный датчик движения

[2] При использовании платы NodeMCU этот модуль не понадобится. — *Прим. пер.*

Затем уделим внимание датчику температуры и влажности. Разместите на второй макетной плате модуль ESP8266. Поместите датчик DHT11 на эту макетную плату, потом подключите первый вывод датчика к выводу VCC модуля ESP8266, второй вывод — к выводу 5 (D1) модуля и, наконец, последний вывод датчика — к выводу GND модуля (рис. 11.2).

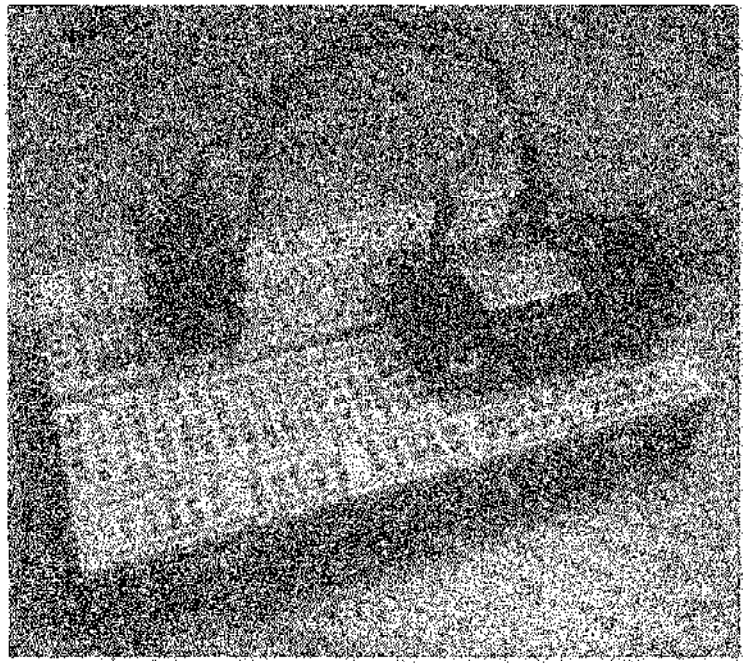


Рис. 11.2. Подключенный датчик DHT11

Соберем теперь блок управления яркостью светодиода. В качестве выходного устройства мы взяли обычный светодиод, но вы, конечно, можете использовать эту схему в качестве отправной точки для создания модуля управления большим количеством светодиодов в вашем доме или даже лампами.

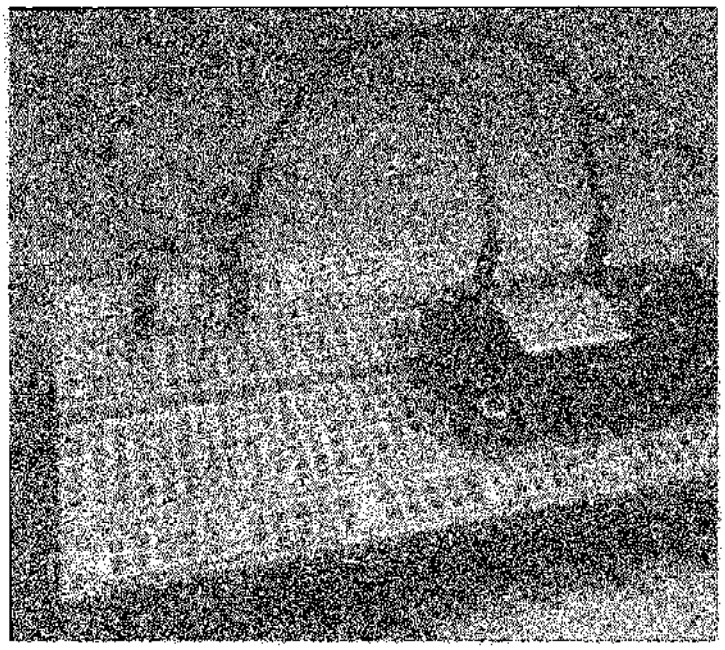


Рис. 11.3. Подключенный светодиод

Разместите третий модуль ESP8266 на третьей макетной плате. Установите на этой макетной плате светодиод последовательно с резистором 330 Ом. Более длинный вывод светодиода соедините с резистором. Затем подключите второй вывод резистора к выводу 5 (D1) модуля ESP8266, а второй вывод светодиода — к выводу GND модуля (рис. 11.3).

Управление домом из приборной панели

В первом проекте этой главы мы научимся управлять всеми ранее собранными модулями из облачной приборной панели при помощи фреймворка aREST, который мы уже использовали в этой книге.

Сначала запрограммируем все модули и начнем с модуля управления светодиодом, который наиболее прост для программирования. Вот полный исходный код программы для этого модуля (листинг 11.1).

Листинг 11.1. Исходный код программы управления светодиодом

```
// Подключаем нужные библиотеки
#include "ESP8266WiFi.h"
#include <PubSubClient.h>
#include <aREST.h>
// Клиенты
WiFiClient espClient;
PubSubClient client(espClient);
// Уникальное имя для идентификации в облаке cloud.arest.io
// придумайте для этой цели свое имя устройства
char* device_id = "6g37g4";
// Создаем объект aREST
aREST rest = aREST(client);
// Параметры доступа WiFi
const char* ssid = "wifi-name";
const char* password = "wifi-pass";
// Порт TCP для входящих подключений
#define LISTEN_PORT 80
// Создаем объект сервера
WiFiServer server(LISTEN_PORT);
void setup(void)
{
  // Активируем последовательный порт
  Serial.begin(115200);
  // Задаем обратный вызов
  client.setCallback(callback);
  // Присваиваем устройству имя и ID
  rest.set_id(device_id);
  rest.set_name("dimmer");
```

```
  // Подключаемся к Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  // Запускаем сервер
  server.begin();
  Serial.println("Server started");
  // Выводим в терминал IP адрес
  Serial.println(WiFi.localIP());
}
void loop() {
  // Обработка обращений aREST
  rest.handle(client);
}
// Обрабатываем обращения от органа управления
void callback(char* topic, byte* payload, unsigned int length) {
  // Обработка
  rest.handle_callback(client, topic, payload, length);
}
```

Готовый к использованию файл программы находится в папке ch11_DASHBOARD_LED сопровождающего книгу электронного архива (см. *приложение*).

В этом коде вам надо изменить несколько значений: указать имя и пароль вашей точки доступа Wi-Fi, а также задать уникальный идентификатор устройства в Сети[3]. Наконец, вы можете изменить название устройства. Например, добавьте какие-либо сведения о том, где именно находится устройство у вас дома.

Когда все это будет сделано, загрузите прошивку в плату и перейдите к следующему устройству — датчику движения.

Программа для этого модуля приблизительно такая же. Надо лишь добавить несколько строк для постоянного наблюдения за состоянием датчика движения и сделать его доступным через облако. Для этого сначала определяем переменную, которая будет хранить состояние датчика движения:

```
int motion;
```

Затем отправляем значение этой переменной в aREST:

```
rest.variable("motion", &motion);
```

[3] Автор имеет в виду уникальное имя устройства для облачного сервиса aREST. — *Прим. пер.*

Далее, внутри функции `loop()` просто считываем состояние датчика движения:

```
motion = digitalRead(5);
```

Готовый к использованию файл программы находится в папке ch11_DASHBOARD_MOTION сопровождающего книгу электронного архива (см. *приложение*).

После изменения тех же параметров доступа, что и для модуля управления светодиодом (логин и пароль Wi-Fi, идентификатор устройства и его имя), загрузите прошивку в плату.

В завершение уделим внимание модулю датчика DHT11. Для этого устройства необходимо импортировать библиотеку DHT:

```
#include "DHT.h"
```

Затем вы должны определить, к какому выводу подключен датчик:

```
#define DHTPIN 5
#define DHTTYPE DHT11
```

После этого создайте объект датчика:

```
DHT dht(DHTPIN, DHTTYPE, 15);
```

А также две переменные, которые будут хранить значение температуры и влажности:

```
float temperature;
float humidity;
```

В функции `setup()` мы инициализируем датчик DHT:

```
dht.begin();
```

А также делаем следующие переменные доступными из aREST API:

```
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
```

Наконец, внутри функции `loop()` считываем из датчика результаты измерения влажности и температуры:

```
humidity = dht.readHumidity();
temperature = dht.readTemperature();
```

Готовый к использованию файл программы находится в папке ch11_DASHBOARD_SENSOR сопровождающего книгу электронного архива (см. *приложение*).

Вновь исправляем в скетче параметры доступа и загружаем прошивку в плату. Имейте в виду, что для питания всех модулей вы можете использовать как внешнюю батарейку, так и блок питания макетной платы. USB-кабель и адаптер FTDI вам для каждого модуля не нужны.

Настало время управлять всеми нашими платами через облако! Первым делом, перейдите на сайт панели приборов aREST: http://dashboard.arest.io.

Создайте новую приборную доску для вашей системы домашней автоматики (рис. 11.4).

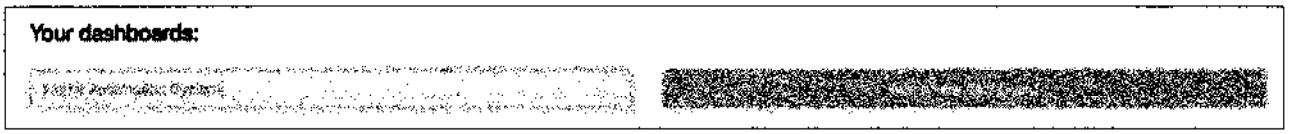


Рис. 11.4. Создаем новую приборную панель

Внутри приборной панели переключитесь в режим редактирования и добавьте первый элемент, который будет поддерживать измерение температуры. Убедитесь, что указали правильный идентификатор модуля измерения влажности и температуры (рис. 11.5).

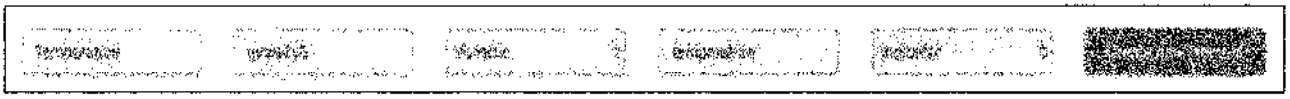


Рис. 11.5. Создаем индикатор значения температуры

Проделайте то же самое для влажности. Вы должны получить результат, подобный показанному на рис. 11.6.

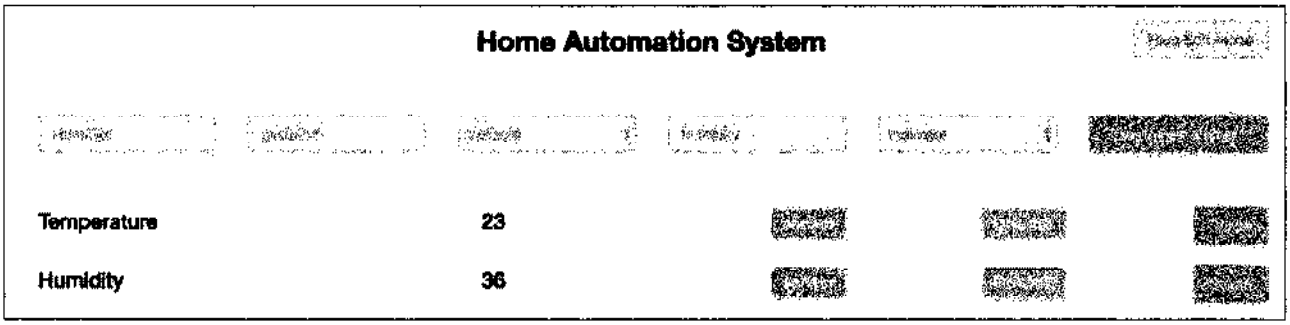


Рис. 11.6. Индикаторы температуры и влажности на панели aREST

Теперь приступим к добавлению модуля регулировки яркости светодиода. Поскольку мы хотим иметь возможность управлять яркостью светодиода, создадим новый элемент с опцией **Analog** (рис. 11.7).

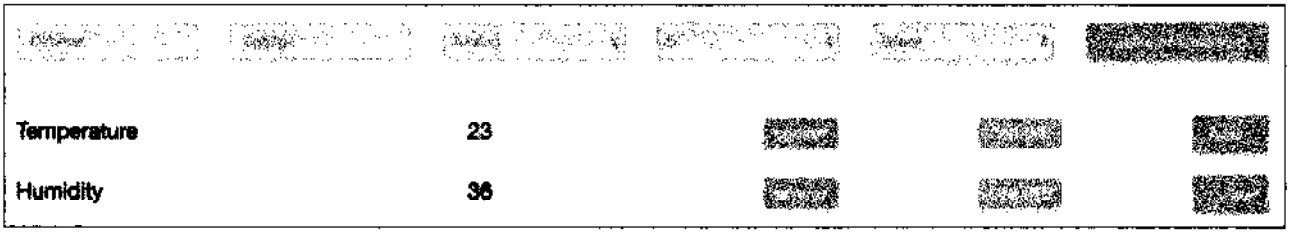


Рис. 11.7. Создаем регулятор яркости светодиода

После этого у вас должна появиться возможность управлять интенсивностью свечения светодиода через приборную панель (рис. 11.8).

В завершение добавим новый элемент для датчика движения (рис. 11.9).

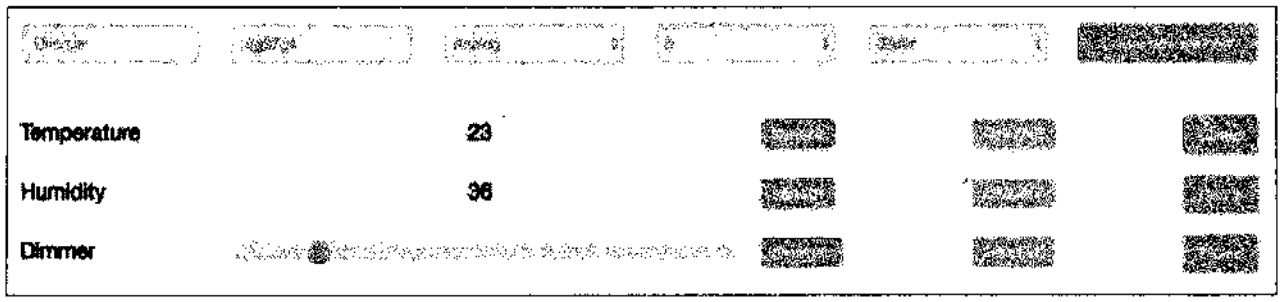


Рис. 11.8. Приборная панель aREST с регулятором яркости светодиода

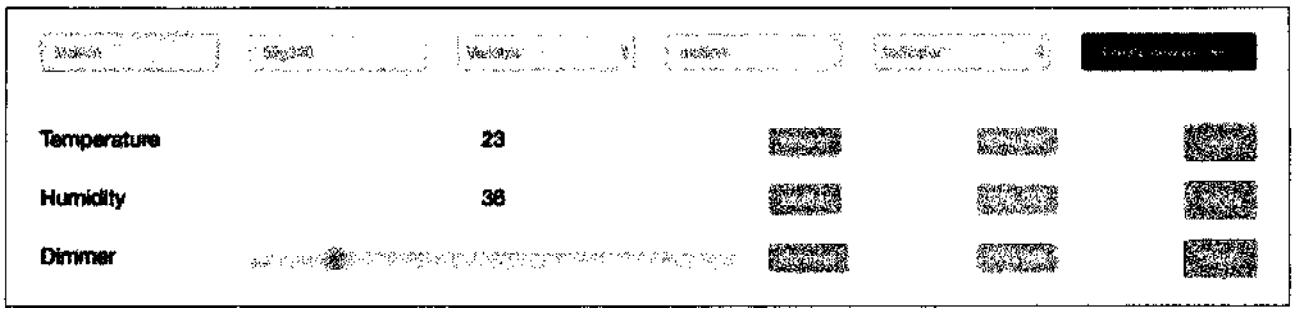


Рис. 11.9. Добавляем отображение состояния датчика движения

В итоге вы должны получить приборную панель, которая содержит все элементы простой системы домашней автоматики (рис. 11.10).

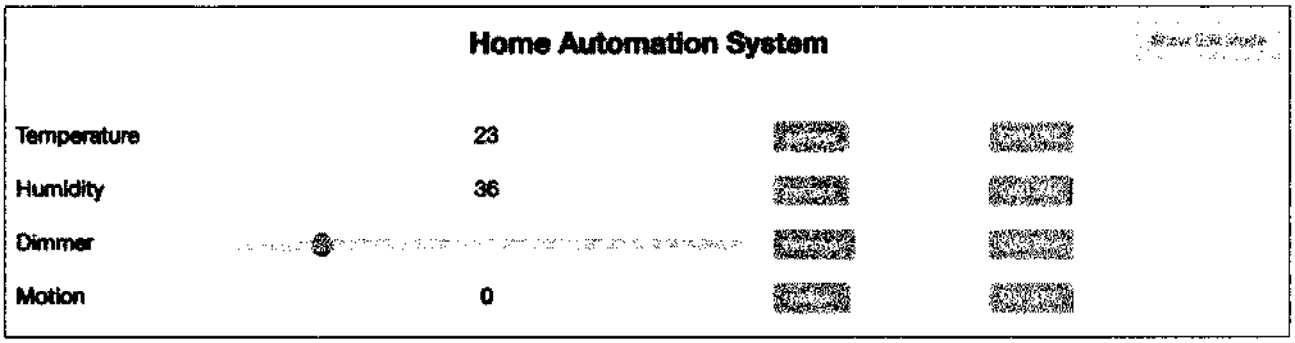


Рис. 11.10. Готовая панель управления системой домашней автоматики

Поздравляю, вы только что на основе модулей ESP8266 создали законченную систему домашней автоматики, которой теперь можете управлять из облака! Конечно, в эту систему можно добавить и дополнительные модули и управлять ими из этой же панели.

Обратите внимание, что панель эта доступна откуда угодно. Вам не нужно обязательно находиться внутри собственного дома для доступа к ней!

Создаем облачную охранную систему

Создадим на базе оборудования, которое уже собрали в этой главе, другой проект, — облачную охранную систему с использованием модуля ESP8266, сопряженного с инфракрасным датчиком движения.

Я покажу вам, как сделать это при помощи всего лишь одного датчика движения, но вы можете добавить к системе дополнительные модули, которые будут размещены по всему дому или любому зданию, за безопасностью которого вы хотите следить. Чтобы добиться этого, мы снова используем сервис IFTTT, который станет отправлять вам текстовое сообщение всякий раз, когда один из модулей обнаружит движение.

Давайте сначала разберемся, как запрограммировать такой модуль. Программа очень похожа на приведенные в этой книге ранее, поэтому сейчас я сосредоточусь только на наиболее важных ее частях.

Вы должны указать ключ, привязанный к вашему каналу **Maker** в сервисе IFTTT:

```
const char* host = "maker.ifttt.com";
const char* eventName = "motion_detected";
const char* key = "key";
```

Затем в функции `loop()` мы считываем состояние датчика движения:

```
bool motion = digitalRead(5);
```

Потом проверяем, обнаружено ли движение:

```
if (motion) {
```

Если это так, создаем запрос, предназначенный для отправки в IFTTT:

```
String url = "/trigger/";
       url += eventName;
       url += "/with/key/";
       url += key;
```

Затем отправляем этот запрос на сервер IFTTT:

```
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
             "Host: " + host + "\r\n" +
             "Connection: close\r\n\r\n");
int timeout = millis() + 5000;
while (client.available() == 0) {
  if (timeout - millis() < 0) {
    Serial.println(">>> Client Timeout !");
    client.stop();
    return;
  }
}
```

После отправки запроса читаем ответ сервера:

```
while (client.available()) {
  String line = client.readStringUntil('\r');
  Serial.print(line);
}
```

Потом достаточно долго ждем, прежде чем отправить новое оповещение, иначе вы просто завалите свой смартфон сообщениями:

```
delay(10 * 60 * 1000);
```

Теперь возьмите исходный код (например, из репозитория GitHub для этой книги), модифицируйте его в соответствии со своими параметрами доступа и загрузите в плату.

Готовый к использованию файл программы находится в папке ch11_SMS_ALARM сопровождающего книгу электронного архива (см. *приложение*).

Затем войдите на сайт IFTTT (**https://ifttt.com/**), чтобы создать новое правило. В качестве обработчика события выберите канал **Maker** (рис. 11.11).

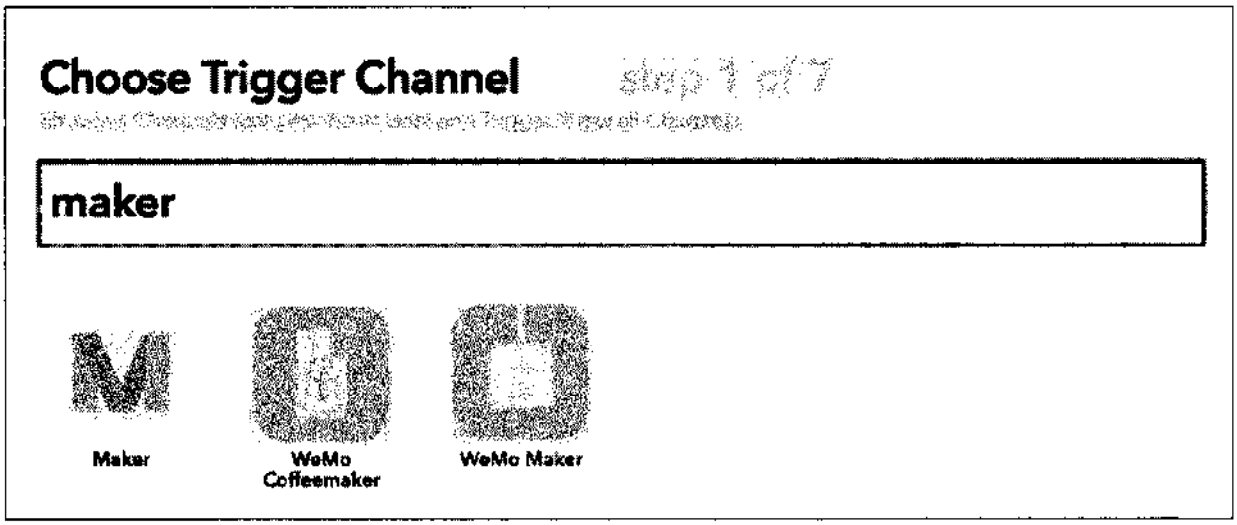


Рис. 11.11. Выбираем канал обработчика события

В качестве имени события введите `motion_detected`. Это же имя мы поместим в программу (рис. 11.12).

Исполнительным каналом в данном случае мы назначим SMS — это наиболее быстрый способ сообщить о тревоге, вызванной срабатыванием датчика движения в вашем доме (рис. 11.13).

Вы можете вставить текст сообщения о срабатывании датчика движения, как, например, показано на рис. 11.14.

Далее создайте правило и активируйте его. Теперь поводите рукой перед датчиком движения — он немедленно отправит тревожное сообщение на ваш смартфон. Если вы хотите отключить охранную систему, просто деактивируйте правило на IFTTT.

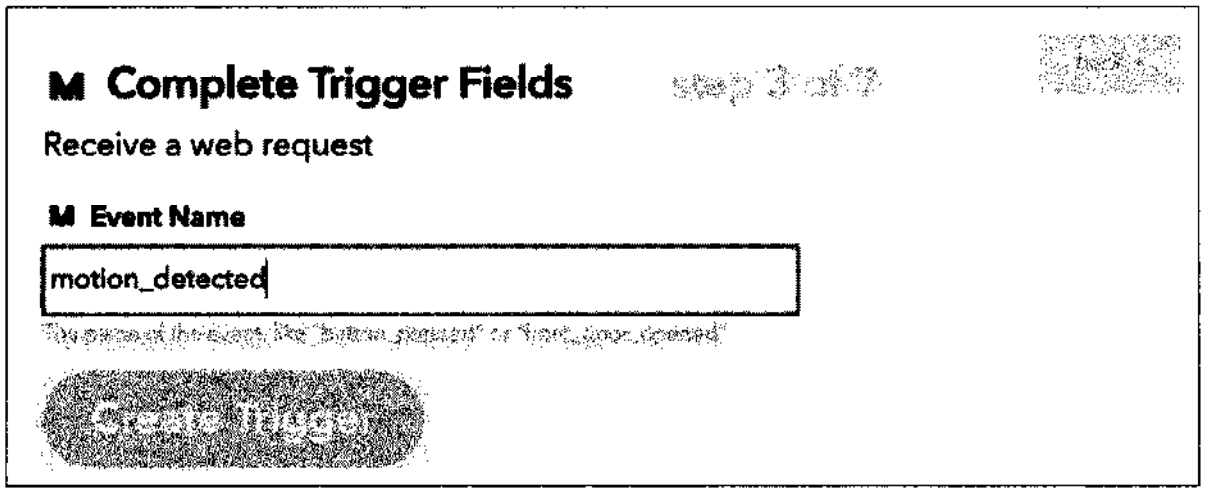


Рис. 11.12. Задаем имя события

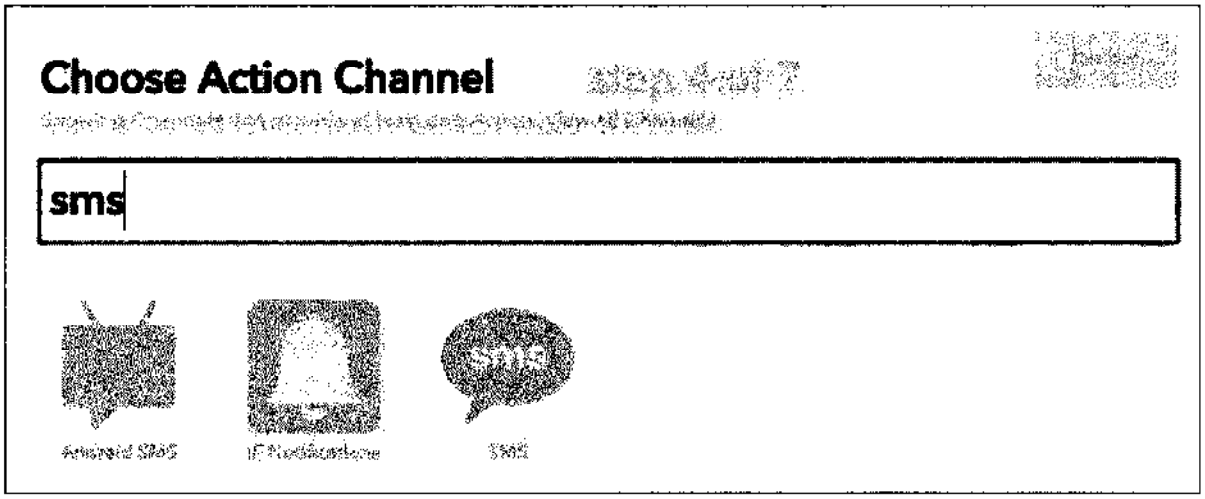


Рис. 11.13. Добавляем канал отправки SMS

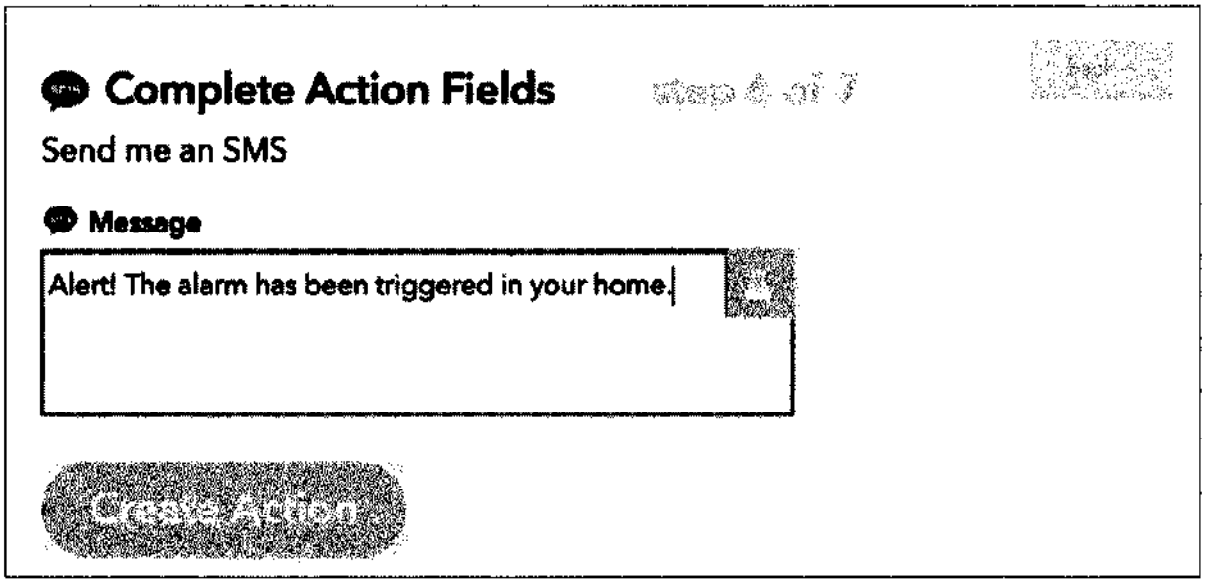


Рис. 11.14. Вводим текст сообщения о тревоге

Вы можете добавить в систему дополнительные датчики, заставив каждый из них отправлять на IFTTT один и тот же сигнал. Тогда при каждом срабатывании любого из датчиков вы станете получать на свой смартфон уведомление о тревоге.

Автоматизация вашего дома

В заключение мы попробуем получить от сервиса IFTTT нечто большее, чем получали до сих пор. На этот раз, в отличие от использования датчика движения, мы разберемся, как задействовать различные триггерные каналы IFTTT для автоматизации управления светодиодным модулем. Само собой, вы сможете заменить светодиод любым другим объектом управления — например, той же лампой.

При этом надо иметь в виду, что канал **Maker** сервиса IFTTT может быть использован и в качестве исполнительного канала[4]. Именно этим мы сейчас займемся — воспользуемся каналом **Maker** для обращения к API сервиса aREST каждый раз, когда выполняется заданное условие.

Мы заново запрограммируем модуль так, чтобы он смог принимать команды из облака. Вот часть кода перед функцией `setup()`:

```
// Импортируем библиотеки
#include "ESP8266WiFi.h"
#include <PubSubClient.h>
#include <aREST.h>
// Инициализируем клиенты Wi-Fi и PubSub
WiFiClient espClient;
PubSubClient client(espClient);
// Здесь введите идентификатор для cloud.arest.io
char* device_id = "********";
// Создаем объект aREST
aREST rest = aREST(client);
// Параметры Wi-Fi
const char* ssid = "wifi-name";
const char* password = "wifi-pass";
// Порт для приема входящих подключений
#define LISTEN_PORT 80
// Создаем объект сервера
WiFiServer server(LISTEN_PORT);
```

А вот функция `setup()`:

```
void setup(void)
{
  // Активируем последовательный порт
  Serial.begin(115200);
  // функция обратного вызова
  client.setCallback(callback);
```

[4] До этого мы его использовали только в качестве получателя события, а исполнителем заданного действия всегда был другой канал. — *Прим. пер.*

```
  // устанавливаем ID и имя устройства
  rest.set_id(device_id);
  rest.set_name("dimmer");
  // Подключаемся к Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  // запускаем сервер
  server.begin();
  Serial.println("Server started");
  // печатаем IP-адрес
  Serial.println(WiFi.localIP());
  // вывод светодиода в режим выхода
  pinMode(5, OUTPUT);
}
```

И, наконец, функция `loop()`:

```
void loop() {
  // обработка вызовов REST
  rest.handle(client);
}
// Обработка вызовов, поступающих из панели или API
void callback(char* topic, byte* payload, unsigned int length) {
  // обработчик
  rest.handle_callback(client, topic, payload, length);
}
```

Готовый к использованию файл программы находится в папке ch11_LED_TIME сопровождающего книгу электронного архива (см. *приложение*).

Модифицируйте параметры доступа (имя и пароль Wi-Fi и идентификатор устройства) и загрузите прошивку в плату.

Потом вернитесь к IFTTT. Первое, чем мы займемся, — это настройка проекта, который зажигает светодиод в заданное время (например, когда за окном стемнело), а затем гасит его в другое время (например, когда вы ложитесь в кровать).

Для этого создайте новое правило с каналом **Date & Time**. Вам придется сначала подключить этот канал, если вы не использовали его раньше (рис. 11.15).

В качестве триггера события выберите **Every day** (каждый день) и введите время, в которое должен включиться светодиод.

Затем в качестве исполнительного канала IFTTT назначьте канал **Maker** и выберите опцию **Make a web request**. Это позволит IFTTT посылать команду в облачный сервис aREST.

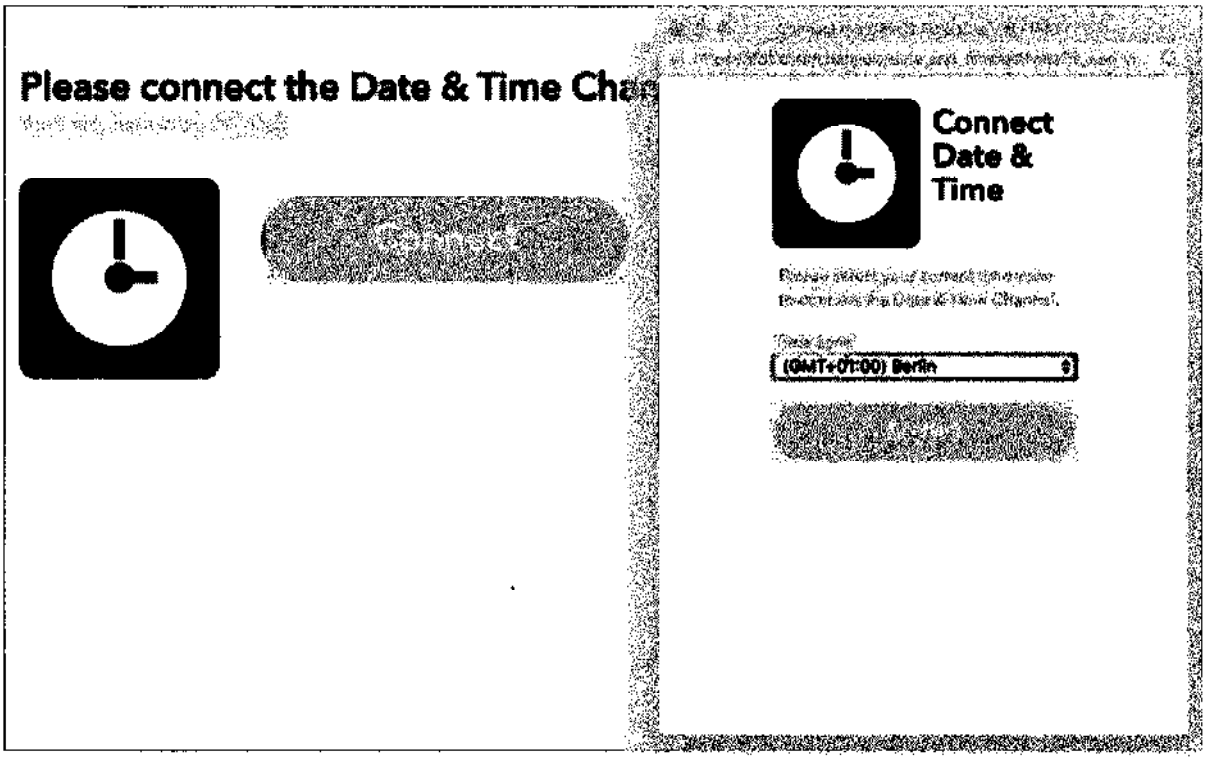


Рис. 11.15. Подключаем канал **Date & Time**

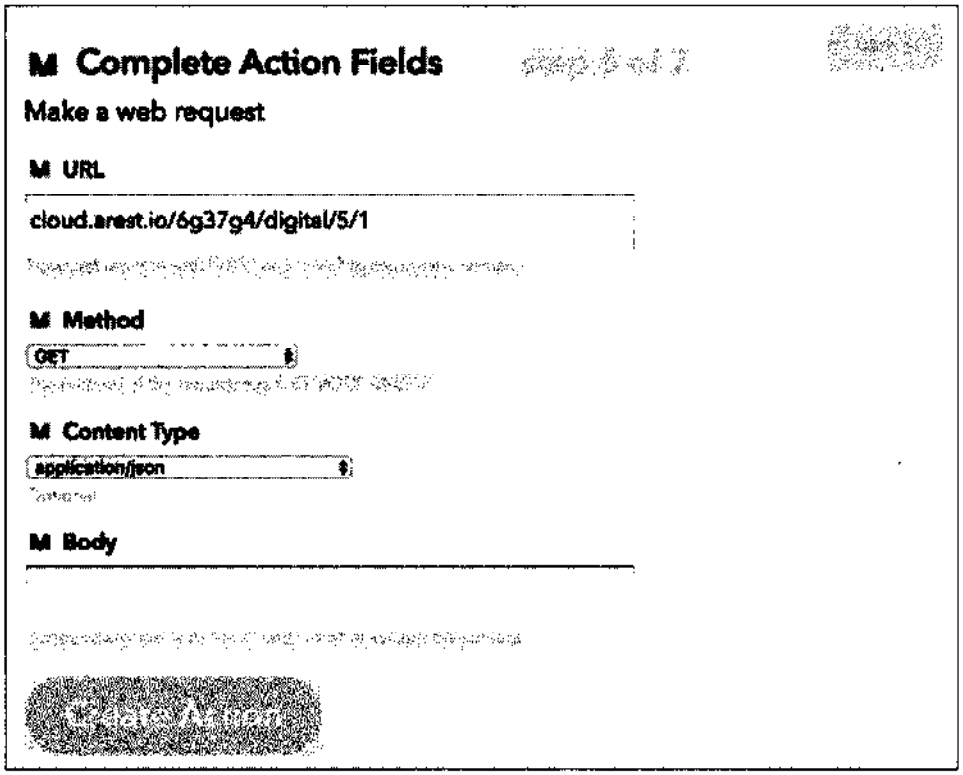


Рис. 11.16. Вводим данные для формирования запроса в облако aREST

Для формирования запроса введите параметры, как показано на рис. 11.16, и не забудьте сменить ID устройства на тот, который указан внутри вашего скетча.

Теперь сделайте то же самое для отключения светодиода — например, в 23.30 (рис. 11.17).

Рис. 11.17. Заданное время отключения

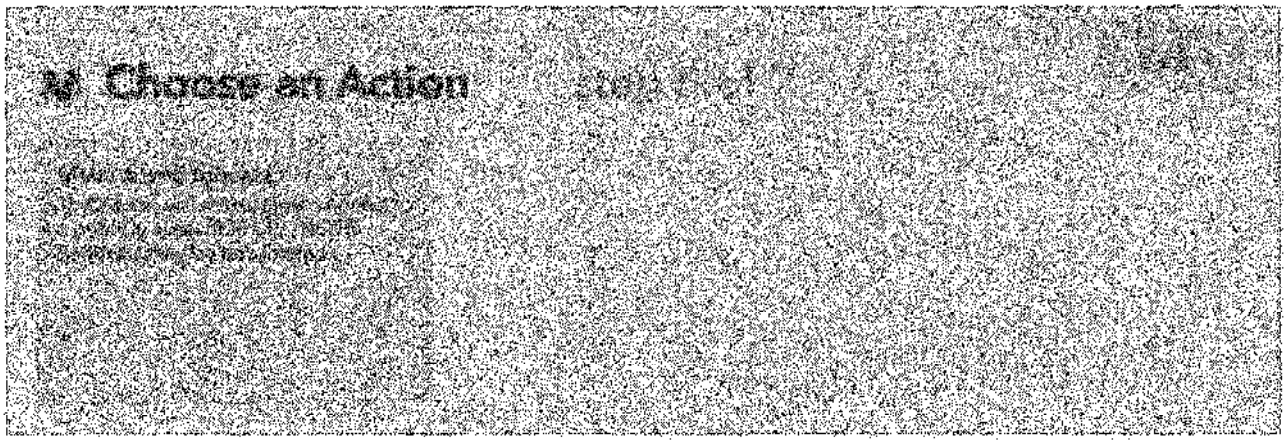


Рис. 11.18. Выбираем исполняющий канал

Выберите тот же исполняющий канал, как и ранее (рис. 11.18).

В запросе используйте те же параметры, что и в прошлый раз. Но теперь дайте команду переключить вывод номер 5 в низкий уровень (рис. 11.19).

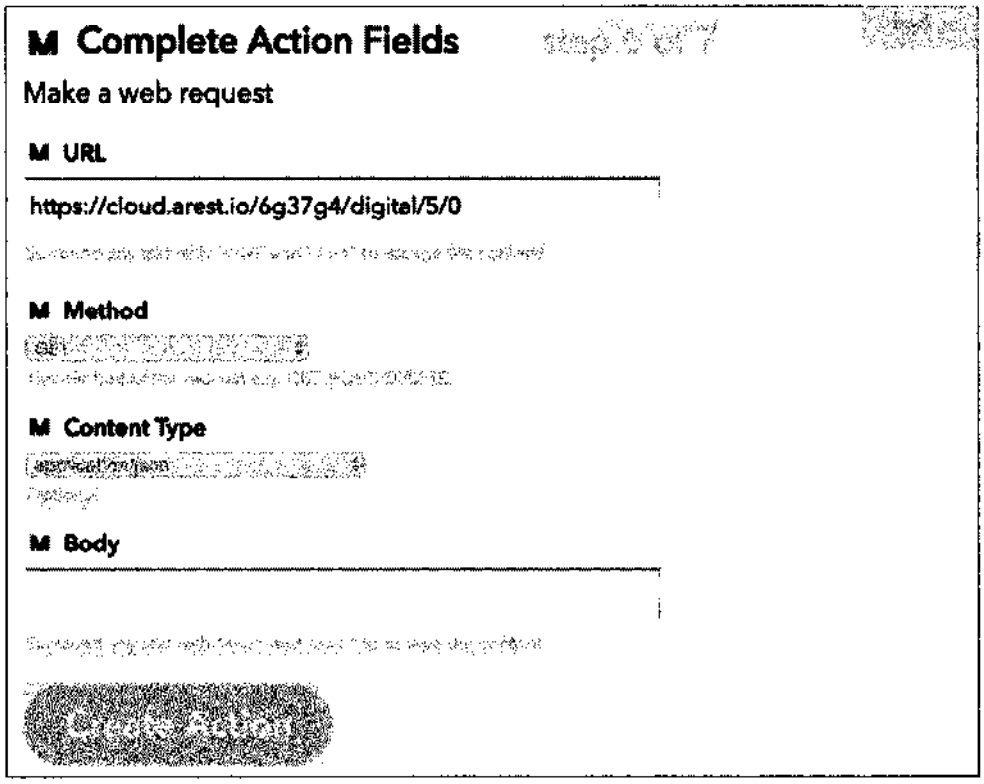


Рис. 11.19. Настраиваем запрос для отключения светодиода

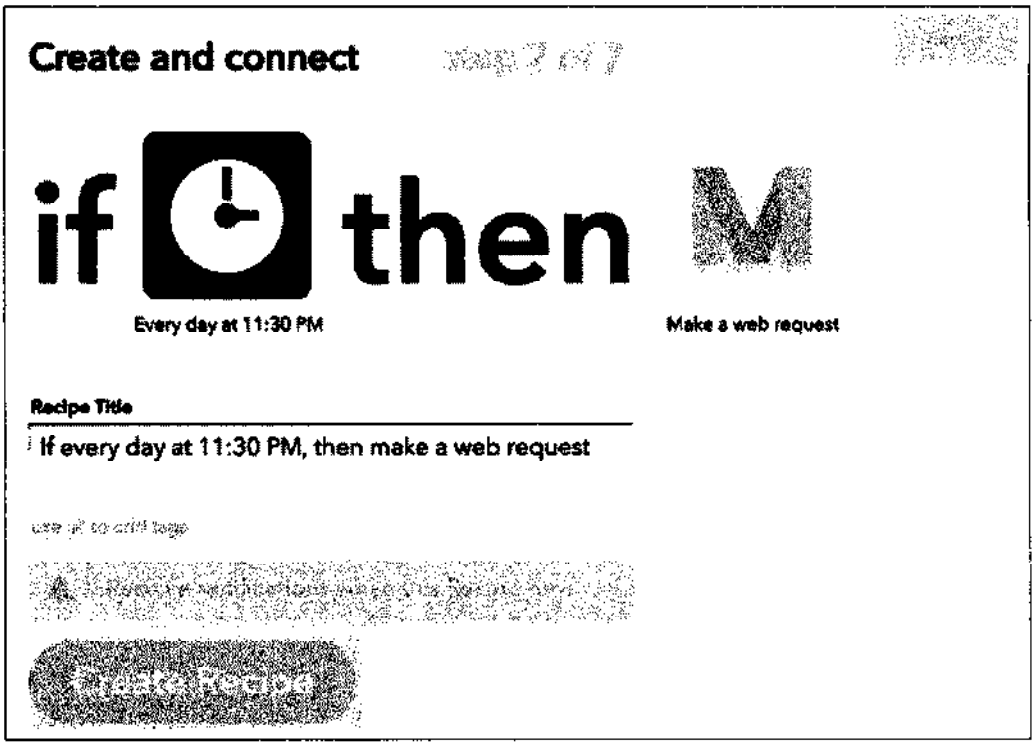


Рис. 11.20. Готовое правило с ежедневным срабатыванием триггера

Посмотрите, как выглядит правило (рис. 11.20).

Настало время проверить это правило в действии! Убедитесь, что модуль ESP8266 правильно запрограммирован, а правила в IFTTT активированы. Теперь каждый раз, когда наступает заданное время, светодиод должен незамедлительно включиться или выключиться.

Давайте воспользуемся другим триггерным каналом, чтобы оценить всю мощь сервиса IFTTT. Например, вы можете подключить канал **Weather** (погода), чтобы узнавать время заката и автоматически включать к этому времени светодиод без необходимости задавать фиксированное время.

Для этого создайте новое правило и выберите для подключения канал **Weather** (рис. 11.21).

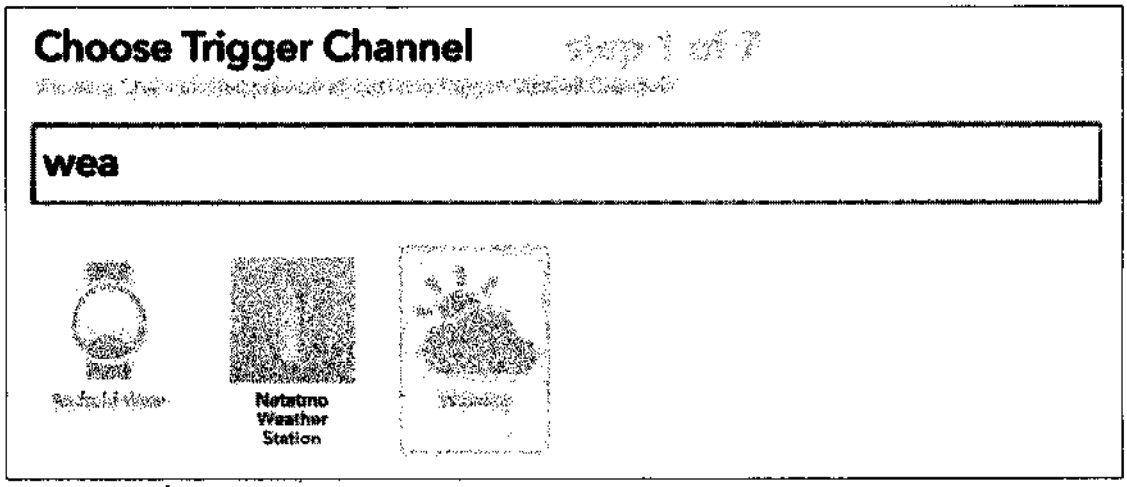


Рис. 11.21. Выбираем канал погоды

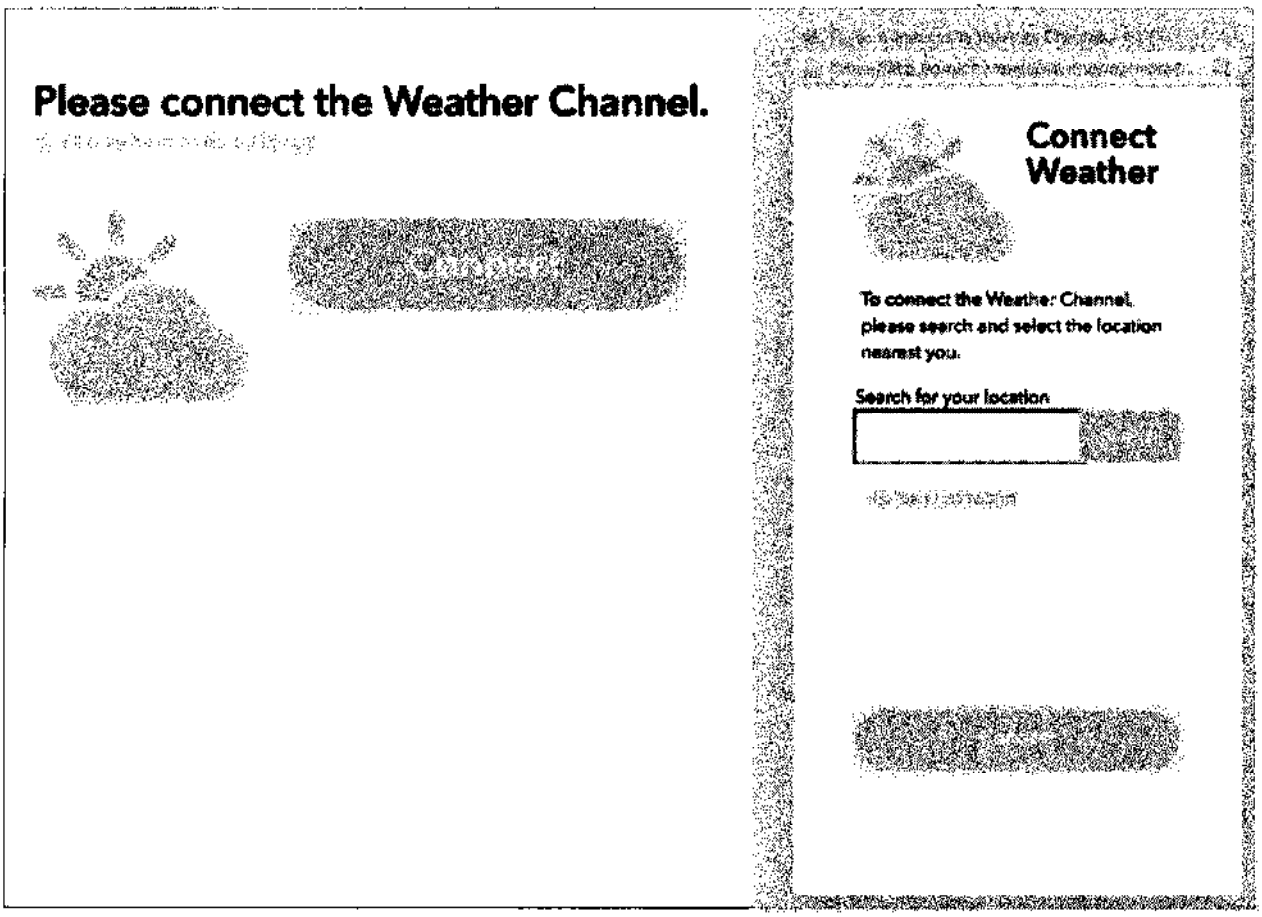


Рис. 11.22. Задаем свое местоположение в настройках канала погоды

Для подключения канала погоды достаточно ввести свое местоположение (рис. 11.22).

В качестве триггера я выбрал событие **Sunset** (заход солнца) (рис. 11.23).

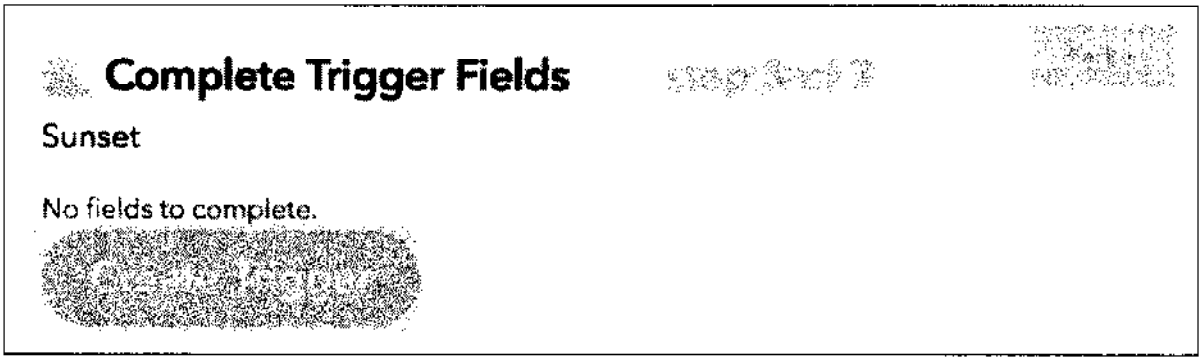


Рис. 11.23. Выбираем событие триггера

А в качестве исполнителя действия — как и ранее, канал **Maker** (рис. 11.24).

Так же, как и в предыдущем правиле, я решил включать светодиод каждый раз, когда срабатывает триггер (рис. 11.25).

Вновь созданное правило выглядит так, как на рис. 11.26.

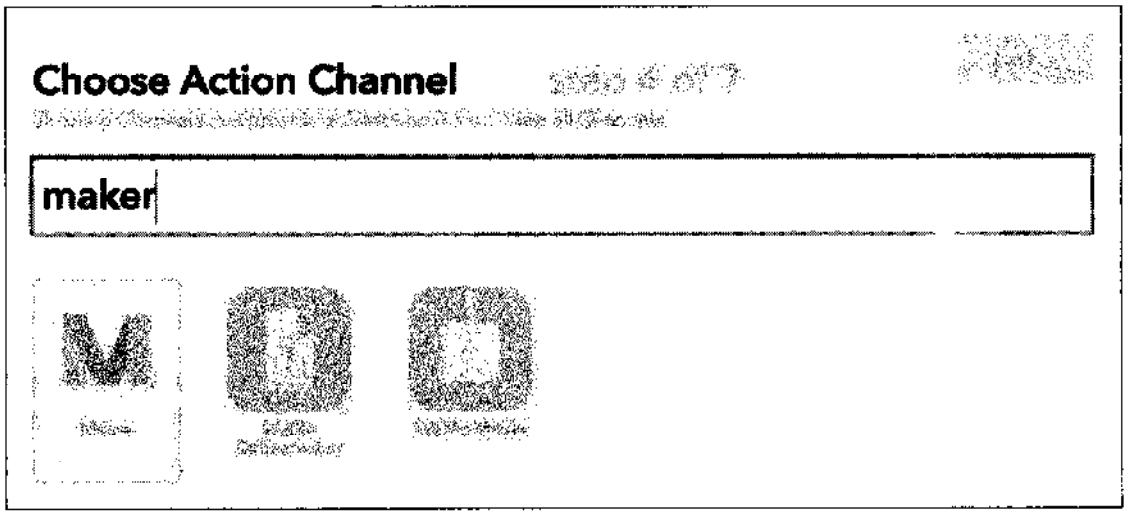


Рис. 11.24. Выбираем канал, выполняющий действие

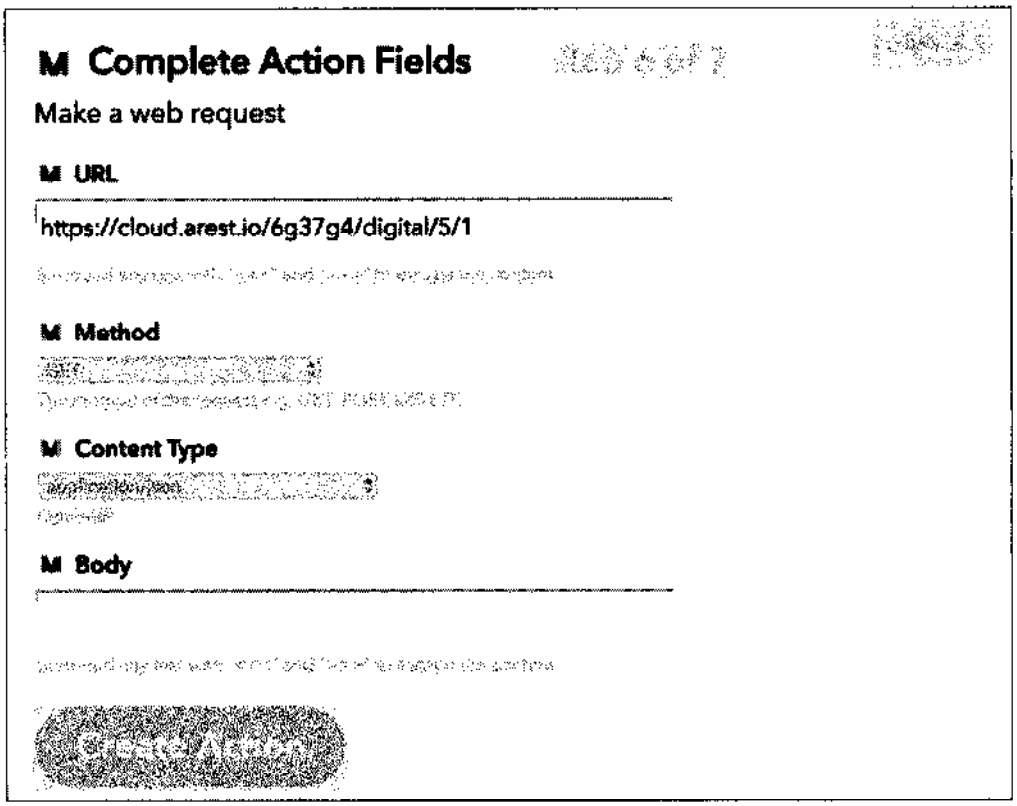


Рис. 11.25. Настраиваем действие, которое выполняет канал

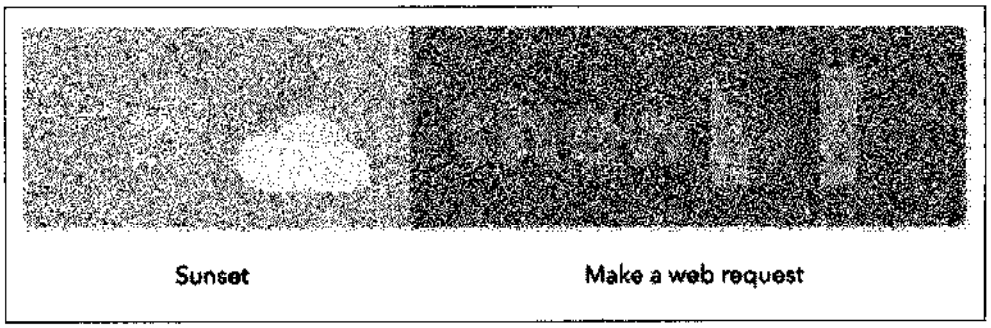


Рис. 11.26. Вновь созданное правило с событием заката солнца

Теперь, когда наступает событие «закат солнца», светодиод должен автоматически включаться. У вас есть возможность варьировать настройки IFTTT и даже использовать канал **Maker** одновременно в качестве как инициатора, так и исполнителя, чтобы через IFTTT связать между собой разные модули вашей системы домашней автоматики.

Заключение

В этой главе мы создали ряд основанных на ESP8266 компонентов системы домашней автоматики и узнали, как можно управлять всеми ими через облако. Сначала мы создали облачную приборную панель, чтобы иметь доступ ко всем устройствам через один интерфейс. Затем мы снова обратились к сервису IFTTT для создания охранной системы, которая автоматически отправляет на ваш смартфон предупреждение, как только в зоне действия соответствующего датчика будет обнаружено какое-либо движение. Простейший способ развития вашей собственной системы на основе этих проектов — просто добавлять больше модулей в соответствии с потребностями у вас дома. Например, совершенно не сложно добавить в нее несколько датчиков движения, благодаря чему вы получите развернутую по всему дому законченную охранную систему, которой вы сможете управлять из обычного браузера.

В следующей главе мы применим ESP8266 для совершенно иного проекта — построим мобильного робота, которым станем управлять через облако.

12

Робот, управляемый через облако

На протяжении всей книги мы в основном использовали модули ESP8266 для построения проектов Интернета вещей, относящихся к автоматизации управления теми или иными системами умного дома и его безопасности, — таких, например, как дистанционно управляемый дверной замок (см. *главу 8*) или комплектная система домашней автоматики, о которой мы говорили в *главе 11*.

Однако ESP8266 — микросхема весьма универсальная, и может использоваться в других областях, кроме двух упомянутых. Например, она может стать «мозгом» мобильного робота. Именно этим мы и собираемся сейчас заняться.

Мы соберем маленького мобильного робота, подключим моторы робота к модулю ESP8266 и свяжем этот модуль через беспроводное соединение с облачной платформой, благодаря чему роботом можно будет управлять откуда угодно с помощью уже знакомой нами облачной панели. Давайте начнем!

Оборудование и программное обеспечение

Для проекта этой главы нам, естественно, потребуется плата с микросхемой ESP8266. В основном я использую в этой книге модуль Adafruit Huzzah ESP8266, но любой другой модуль ESP8266 будет работать здесь ничуть не хуже.

Еще одним важным для нас компонентом является роботизированная платформа. На рынке представлено множество таких платформ, но лишь немногие из них способны работать совместно с ESP8266. Так что нам следует выбрать наиболее универсальную платформу, на которую мы сможем установить компоненты по своему усмотрению.

Роботизированная платформа для этого проекта должна обладать определенным набором базовых особенностей:

- иметь, как минимум, два колеса;
- иметь два мотора;

- быть достаточно крупной, чтобы уместить макетную плату с модулем ESP8266 и комплект питающих элементов.

Платформ, которые соответствуют этим условиям, на самом деле существует множество. Одна из категорий — это простые двухколесные наборы для изготовления роботов. В качестве примера можно привести набор EmGreat (рис. 12.1).

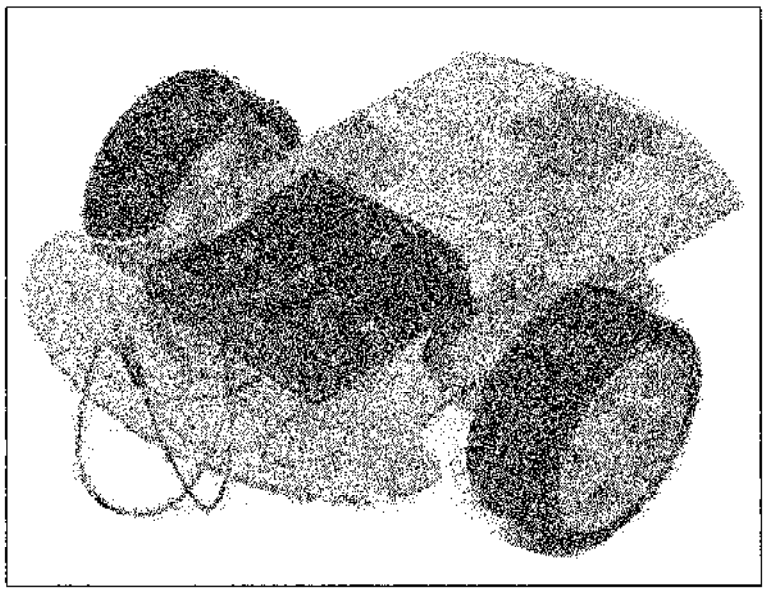


Рис. 12.1. Простая двухколесная универсальная платформа EmGreat

Эта платформа оснащена двумя ведущими колесами с моторами и третьим свободным колесиком впереди, обеспечивающим роботу устойчивость. Все компоненты управления робота можно смонтировать наверху платформы.

Еще один вариант — четырехколесные роботы, своего рода микроавтомобильчики, на которых также можно установить управляющее оборудование (рис. 12.2).

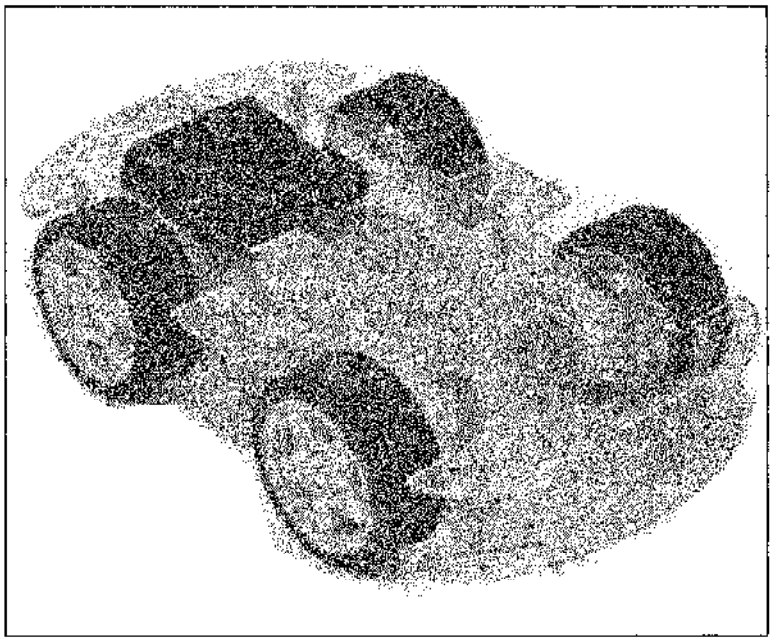


Рис. 12.2. Четырехколесная универсальная платформа EmGreat

Выбирая механическую платформу, убедитесь, что у нее только два мотора, потому что в этой главе я описываю проект с двумя моторами. Разумеется, вы можете без труда адаптировать программу и к четырехколесным платформам.

Я выбрал для этой главы похожую платформу, потому что у меня было гусеничное шасси, которое мне давно хотелось использовать (рис. 12.3).

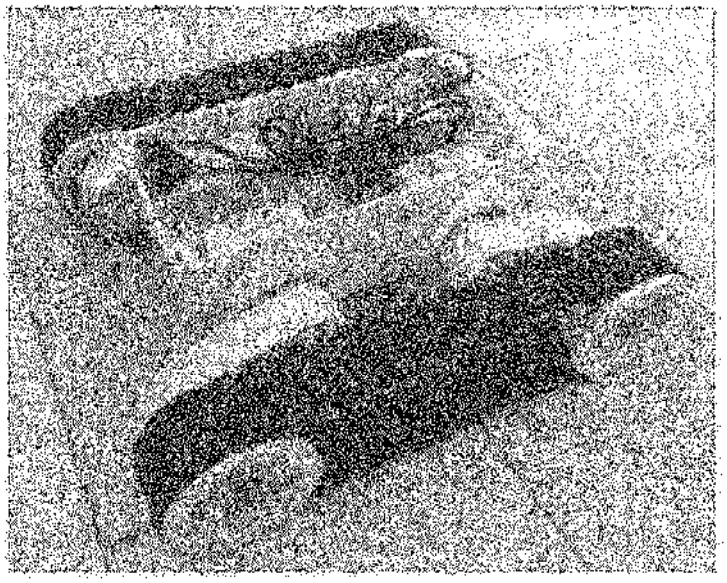


Рис. 12.3. Гусеничное шасси

Это шасси не слишком отличается от упомянутых ранее — оно также имеет два мотора, вращающих его ведущие ролики, разве что на эти ролики надеты резиновые гусеницы, делающие шасси похожим на танк. Благодаря этому можно быть уверенным, что роботу не слишком помешают препятствия и неровности поверхности, по которой он будет перемещаться.

Для управления роботом нам понадобятся также следующие компоненты:

- специализированная микросхема (драйвер) для управления моторами L293D — поскольку невозможно управлять моторами напрямую от ESP8266;
- подходящая батарея для питания моторов роботизированной платформы. Моторы моего гусеничного шасси рассчитаны на рабочее напряжение 7 вольт, поэтому я использовал батарею на 7,4 вольта (рис. 12.4);
- беспаечная макетная плата и соединительные провода.

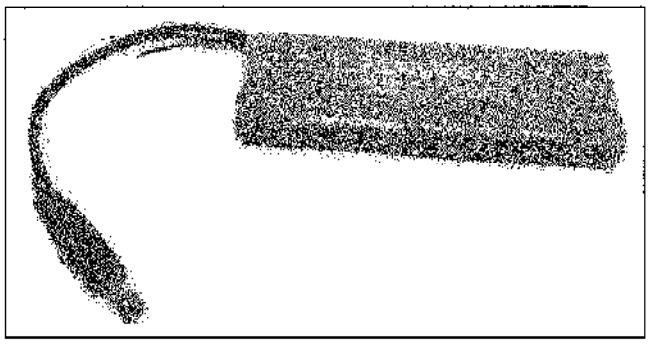


Рис. 12.4. Батарея для питания роботизированной платформы

Вот конкретный перечень использованных в проекте компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266: **https://www.adafruit.com/products/2471**;
- модуль FTDI USB: **https://www.adafruit.com/products/284**;
- гусеничное шасси с двумя моторами: **http://www.dfrobot.com/index.php?route=product/product&product_id=390**;
- драйвер моторов L293D: **https://www.adafruit.com/products/807**;
- батарея 7.4 В с разъемом: **http://www.robotshop.com/en/dfrobot-7-4v-lipo-2200mah-battery.html**;
- макетная плата: **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

В части программного обеспечения вам понадобятся Arduino IDE и библиотека aREST, которые мы уже использовали в этой книге ранее.

Сборка схемы

Поскольку наш проект весьма сложен, я сделал для вас детальную схему соединений, призванную облегчить вам ее понимание (рис. 12.5).

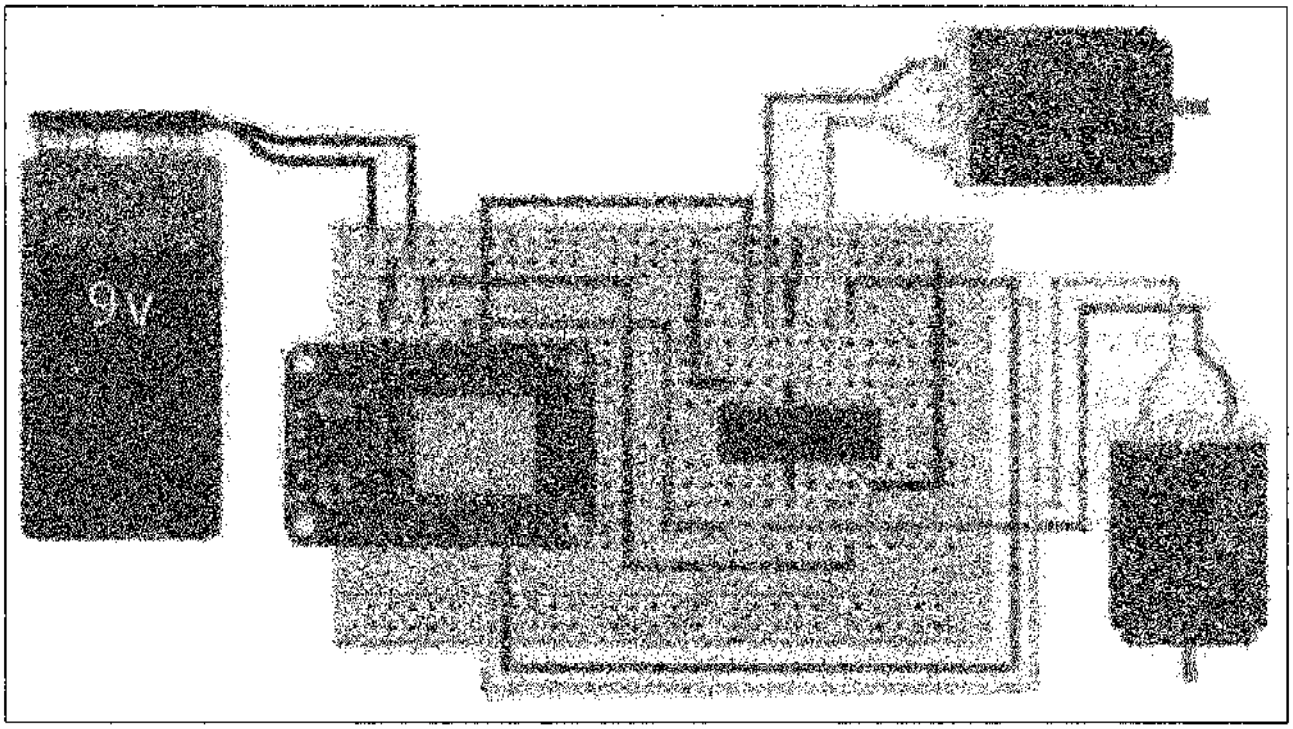


Рис. 12.5. Монтажная схема соединений

Подготовил я вам в помощь и подробную электрическую схему (рис. 12.6).

Сначала установите все компоненты на макетной плате и соедините их между собой согласно схемам, приведенным на рис. 12.5 и 12.6 (рис. 12.7), и только после

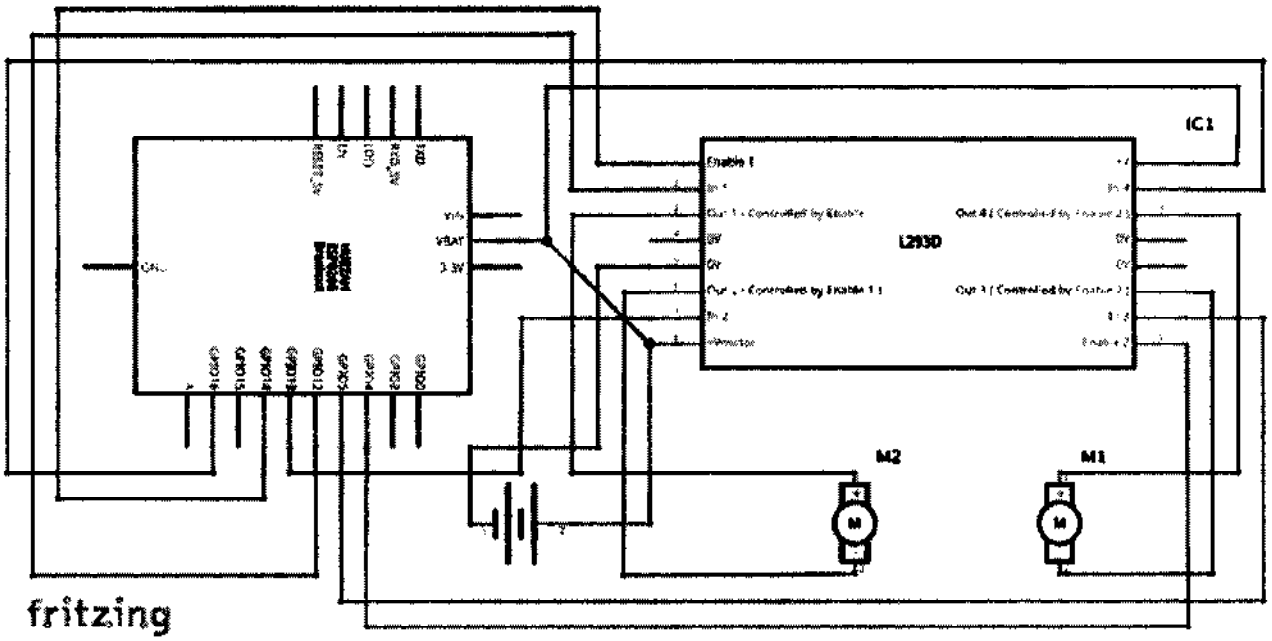


Рис. 12.6. Электрическая схема проекта

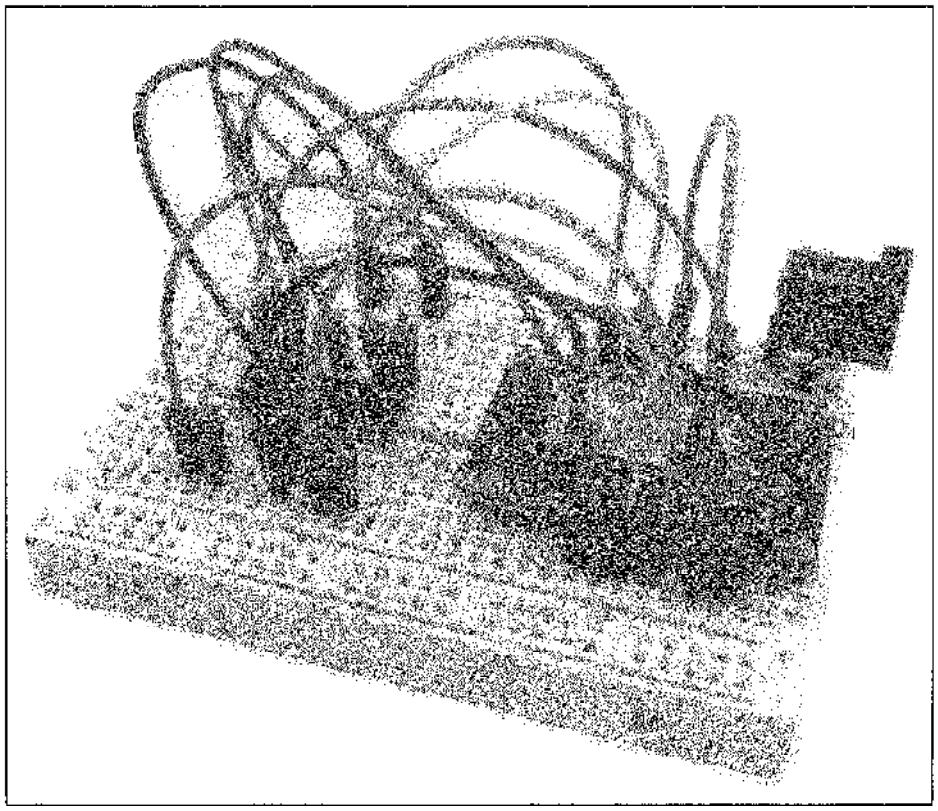


Рис. 12.7. Макетная плата собранного проекта

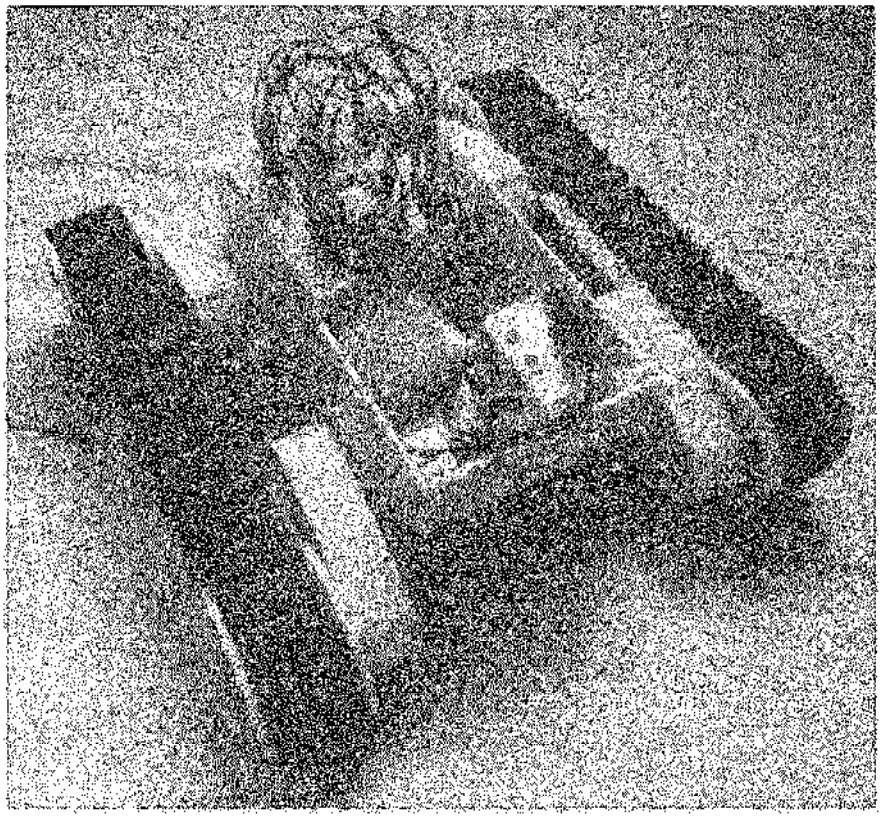


Рис. 12.8. Полностью собранная роботизированная платформа

этого установите макетную плату на шасси и подключите ее к моторам (рис. 12.8). В последнюю очередь подключите батарею.

Платформа продается с крышкой, которая скрывает компоненты, но я решил оставить ее открытой, чтобы вы могли видеть внутренности проекта. Но вы обязательно закройте крышку или надежно закрепите все компоненты на шасси при помощи винтов или клея!

Проверка моторов

Прежде чем управлять роботом через облако, мы выполним простой тест — чтобы удостовериться в правильной работе моторов. Вдобавок, вы сможете подробно познакомиться с кодом для управления моторами (листинг 12.1).

Листинг 12.1. Проверка работоспособности робота

```
// объявляем выводы для управления моторами
int motorOnePlus = 12;
int motorOneMinus = 13;
int motorOneEnable = 14;
int motorTwoPlus = 5;
int motorTwoMinus = 16;
int motorTwoEnable = 4;
```

```
void setup()
{
  // настраиваем выводы, как выходы
  pinMode(motorOnePlus, OUTPUT);
  pinMode(motorOneMinus, OUTPUT);
  pinMode(motorOneEnable, OUTPUT);
  pinMode(motorTwoPlus, OUTPUT);
  pinMode(motorTwoMinus, OUTPUT);
  pinMode(motorTwoEnable, OUTPUT);
}

void loop()
{
  // ускорение вперед
  setMotorOne(true, 500);
  setMotorTwo(true, 500);
  // пауза
  delay(5000);
  // стоп
  setMotorOne(true, 0);
  setMotorTwo(true, 0);
  // пауза
  delay(5000);
}

// функция управления первым мотором
void setMotorOne(boolean forward, int motor_speed) {
  digitalWrite(motorOnePlus, forward);
  digitalWrite(motorOneMinus, !forward);
  analogWrite(motorOneEnable, motor_speed);
}

// функция управления вторым мотором
void setMotorTwo(boolean forward, int motor_speed) {
  digitalWrite(motorTwoPlus, forward);
  digitalWrite(motorTwoMinus, !forward);
  analogWrite(motorTwoEnable, motor_speed);
}
```

Рассмотрим подробнее наиболее важные части этого кода.

Сначала мы определяем все выводы ESP8266, которые задействованы для подключения драйвера моторов L293D:

```
int motorOnePlus = 12;
int motorOneMinus = 13;
int motorOneEnable = 14;
```

```
int motorTwoPlus = 5;
int motorTwoMinus = 16;
int motorTwoEnable = 4;
```

В функции `setup()` мы настраиваем все упомянутые ранее выводы, как выходы:

```
pinMode(motorOnePlus, OUTPUT);
pinMode(motorOneMinus, OUTPUT);
pinMode(motorOneEnable, OUTPUT);
pinMode(motorTwoPlus, OUTPUT);
pinMode(motorTwoMinus, OUTPUT);
pinMode(motorTwoEnable, OUTPUT);
```

Внутри функции `loop()` сначала заставляем моторы вращаться вперед с половиной от максимальной скорости:

```
setMotorOne(true, 500);
setMotorTwo(true, 500);
```

Затем останавливаем моторы и включаем их снова. Взгляните на функцию, задающую режим первого мотора:

```
void setMotorOne(boolean forward, int motor_speed) {
  digitalWrite(motorOnePlus, forward);
  digitalWrite(motorOneMinus, !forward);
  analogWrite(motorOneEnable, motor_speed);
}
```

Как видите, функция состоит из двух частей: одна часть отвечает за направление вращения, другая — за скорость. Направление задается приложением двух противоположных логических уровней на выводы + и – микросхемы L293D для соответствующего мотора. Чтобы задать скорость, мы просто подаем сигнал ШИМ (широтно-импульсная модуляция) на соответствующий вывод микросхемы при помощи функции `analogWrite()`. Учтите, что аргумент этой функции может принимать значения от 0 до 1023.

Функция для управления вторым мотором точно такая же:

```
void setMotorTwo(boolean forward, int motor_speed) {
  digitalWrite(motorTwoPlus, forward);
  digitalWrite(motorTwoMinus, !forward);
  analogWrite(motorTwoEnable, motor_speed);
}
```

Текст этого скетча находится в папке ch12_MOTOR_TEST_B сопровождающего книгу электронного архива (см. *приложение*). В папке ch12_MOTOR_TEST_A архива приведен текст этого же скетча, адаптированный для работы с модулем NodeMCU и шилдом для управления моторами DoIt Smart Car.

Настало время запустить скетч и заставить моторы работать! Но прежде чем что-либо сделать, убедитесь, что колеса робота не касаются земли (установите его, например, на небольшой подставке). В противном случае — когда колеса начнут

вращаться — вас может ждать неприятный сюрприз! Проверьте, подключена ли батарея[1].

Затем загрузите прошивку в плату робота. Вы должны увидеть, что колеса быстро начнут вращаться в одном направлении, остановятся и затем вновь начнут вращаться.

Если колеса вращаются в разных направлениях, проверьте правильность всех соединений. Это важно — ведь мы хотим, чтобы робот двигался вперед, когда мы отдаем обоим моторам одну и ту же команду.

Подключение робота к облаку

Теперь, удостоверившись, что колеса робота работают правильно, займемся подключением его к облачной платформе aREST, чтобы иметь возможность управлять им из любой точки мира.

Поскольку скетч весьма велик и в значительной мере совпадает со скетчем тестирования моторов, который мы рассмотрели ранее, я отмечу здесь только самые важные моменты. Полный текст скетча вы можете найти в репозитории GitHub для этой книги.

Скетч начинается с импортирования нужных библиотек:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
```

Затем мы создаем клиента для соединения с облачным сервером aREST:

```
WiFiClient espClient;
PubSubClient client(espClient);
```

А также создаем объект для клиента aREST:

```
aREST rest = aREST(client);
```

Затем, как мы это делали и в предыдущих главах, необходимо ввести уникальный идентификатор устройства:

```
char* device_id = "40ep12";
```

Далее следует ввести имя и пароль своей точки доступа:

```
const char* ssid = "wifi-name";
const char* password = "wifi-password";
```

Теперь нам следует определить набор функций, которые будут непосредственно осуществлять дистанционное управление роботом. Здесь я буду использовать функции, соответствующие основным командам для робота: стоп, вперед, назад, направо и налево.

---

[1] Рекомендуется подключать силовую батарею только после загрузки прошивки и отключения ESP8266 от компьютера. — *Прим. пер.*

Сначала мы должны определить все эти функции:

```
int stop(String command);
int forward(String command);
int left(String command);
int right(String command);
int backward(String command);
```

В функции `setup()` мы связываем плату с идентификатором устройства и даем ей имя:

```
rest.set_id(device_id);
rest.set_name("robot");
```

Надо также привязать функции к библиотеке aREST, чтобы иметь возможность вызывать их дистанционно:

```
rest.function("forward", forward);
rest.function("stop", stop);
rest.function("right", right);
rest.function("left", left);
rest.function("backward", backward);
```

Затем мы соединяем плату с сетью Wi-Fi:

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
```

В функции `loop()` соединяемся с облаком при помощи этой строки:

```
rest.handle(client);
```

Теперь давайте рассмотрим одну из упомянутых функций, которые будем использовать для того, чтобы робот двигался. Так выглядит функция, которая заставляет робота ехать вперед:

```
int forward(String command) {
  setMotorOne(true, 1000);
  setMotorTwo(true, 1000);
}
```

Если вы вспомните функции из предыдущего раздела этой главы, то увидите, что этот код просто заставляет оба мотора вращаться в одном направлении со скоростью, близкой к максимальной.

Настало время испытать робота! Скачайте код программы из репозитория GitHub, не забудьте присвоить устройству свой уникальный идентификатор и ввести имя и пароль Wi-Fi. Удостоверьтесь, что батарея подключена к роботу.

Текст этого скетча находится в папке ch12_ROBOT_CONTROL_B сопровождающего книгу электронного архива (см. *приложение*). В папке ch12_ROBOT_CONTROL_A архива приведен текст этого же скетча, адаптированный для работы с модулем NodeMCU и шилдом для управления моторами Dolt Smart Car.

Загрузите прошивку в плату робота. На этот раз после загрузки ничего не должно произойти. Чтобы сделать робота автономным, отключите от него кабель USB FTDI и подключите вывод платы Vbat к батарейному источнику питания (модуль Adafruit ESP8266 без проблем выдерживает напряжение 7 вольт). Если вы используете другую плату ESP8266, то вам может потребоваться питание от внешнего источника.

Теперь запустите ваш любимый браузер и перейдите по адресу:
https://cloud.arest.io/40ep12/id.

Разумеется, вам следует заменить идентификатор устройства на тот, который указали в программе. Вы должны получить такой ответ:

```
{
"id": "40ep12",
"name": "robot",
"connected": true
}
```

Далее убедитесь, что вокруг робота достаточно свободного места, и введите в адресную строку браузера следующую строку: `https://cloud.arest.io/40ep12/forward`. Вы должны получить в окне браузера подтверждение того, что функция была выполнена, а также увидеть, что робот поехал вперед!

Чтобы остановить его, просто наберите: `https://cloud.arest.io/40ep12/stop`. Эта команда должна немедленно остановить робота.

Теперь можно поиграть с другими функциями, заставляя робота повернуть направо или налево либо поехать назад. Поздравляю, теперь вы можете управлять роботом из любой точки планеты!

Управление роботом из приборной панели

Иметь возможность управлять роботом командами через адресную строку браузера — уже хорошо, но все же хотелось бы делать это из любой точки мира при помощи удобного графического интерфейса, особенно когда надо выполнить короткое действие, — например, повернуть робота на заданный угол.

Чтобы упростить процесс управления роботом, мы вновь воспользуемся функциями панели управления aREST. Благодаря им мы сможем управлять роботом, нажимая соответствующие кнопки.

Если вы еще не создали учетную запись aREST, сделайте это сейчас на сайте: **http://dashboard.arest.io/**.

Создайте там новую панель управления для вашего робота (рис. 12.9).

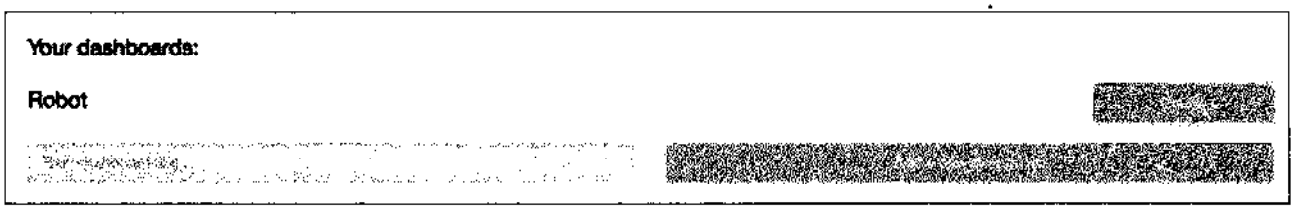


Рис. 12.9. Создаем приборную панель для управления роботом

Теперь внутри новой панели создайте кнопку для первой функции — движения вперед. Для этого создайте новый элемент типа `function`, вызывающий функцию `forward` (рис. 12.10).

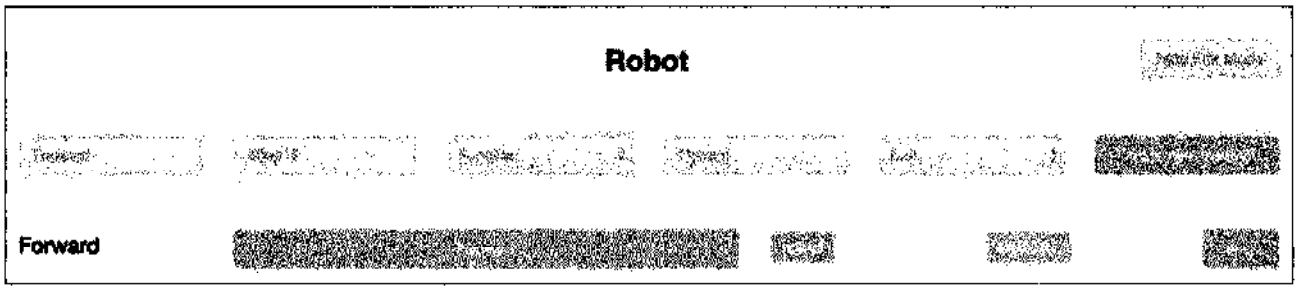


Рис. 12.10. Создаем кнопку для команды **forward** (вперед)

Затем повторим те же действия для функции `stop` (рис. 12.11).

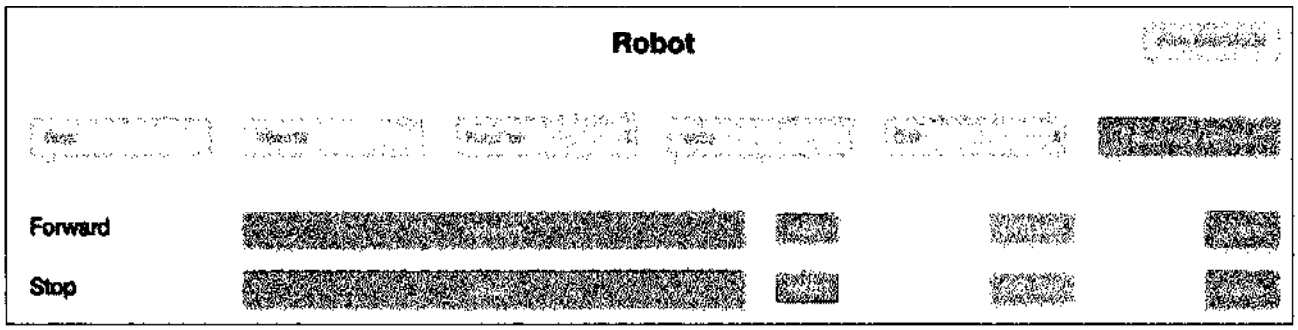


Рис. 12.11. Создаем кнопку для команды **stop** (стоп)

Аналогичным образом создаем кнопки для оставшихся функций (рис. 12.12).

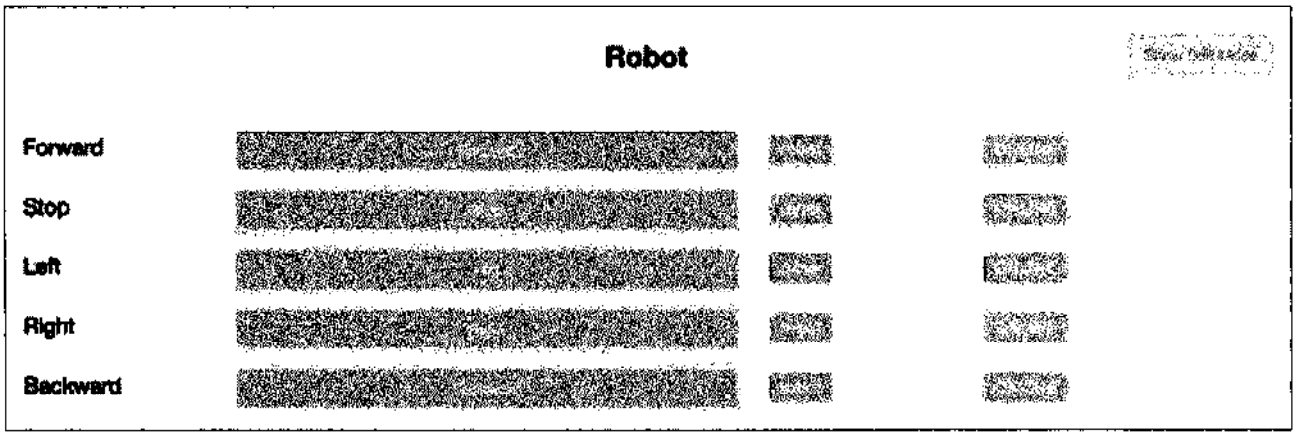


Рис. 12.12. Полный набор кнопок для управления роботом

Вы можете немедленно испытать свои кнопки — всякий раз, когда вы нажимаете кнопку, это должно приводить к незамедлительному выполнению платой ESP8266 предписанного действия — соответствующего маневра робота.

Заключение

В этой главе мы использовали модуль ESP8266 в проекте Интернета вещей, который полностью отличается от проектов предшествующих глав, — мы построили мобильного робота, напрямую подключенного к облаку. Мы использовали онлайновую панель управления, чтобы управлять роботом при помощи графического интерфейса. Самое приятное, что благодаря этому роботом можно управлять откуда угодно.

Хороший способ развить этот проект — добавить в него датчики и также управлять ими через облако. Например, вы можете добавить ультразвуковой сонар, который поможет вам узнать, что перед роботом находится препятствие.

В следующей главе мы перейдем к более сложной теме — развернем собственный облачный сервер, чтобы получить полный контроль над своим Интернетом вещей при помощи ESP8266.

13

Строим собственную облачную платформу для устройств на ESP8266

В предыдущих главах этой книги мы всегда использовали для подключения наших проектов на ESP8266 к облаку, а также для управления ими и мониторинга их из облака, внешние сервисы: IFTTT, Adafruit IO и aREST.

Однако все эти сервисы находятся под управлением других людей или компаний. Это может породить для некоторых проектов проблемы с безопасностью. Кроме того, эти сервисы могут в любое время отключиться или, как минимум, не работать правильно. Если вы действительно хотите иметь полный контроль над своими облачными проектами, единственный выход заключается в развертывании в облаке вашего собственного сервера.

Именно этим мы и собираемся сейчас заняться. Сначала мы создадим облачный сервер, чтобы развертывать на нем веб-приложения, к которым можно получить доступ из любого места. Затем мы расскажем, как развернуть собственный облачный сервис aREST на только что созданном сервере. Наконец, в качестве теста мы подключим простой проект на ESP8266 к вашему собственному серверу. Давайте начнем!

Оборудование и программное обеспечение

Посмотрим сначала, что нам для этого проекта понадобится. Поскольку мы просто хотим протестировать связь между платой ESP8266 и вашим собственным облачным сервером, то и обойдемся без лишних сложностей.

В основном я использую в этой книге модуль Adafruit Huzzah ESP8266, но любой другой модуль ESP8266 будет работать здесь ничуть не хуже. Может понадобиться

и модуль USB FTDI 3,3/5 В для загрузки программ в ESP8266[1]. К модулю ESP8266 мы просто подсоединим два компонента: светодиод и датчик DHT11. Разумеется, понадобятся также обычная макетная плата и соединительные провода.

Вот конкретный перечень использованных в проекте компонентов с указанием места приобретения каждого из них:

- модуль Adafruit ES8266: **https://www.adafruit.com/products/2471**;
- модуль FTDI USB: **https://www.adafruit.com/products/284**;
- светодиод: **https://www.sparkfun.com/products/9590**;
- резистор 330 Ом: **https://www.sparkfun.com/products/8377**;
- датчик DHT11: **https://www.adafruit.com/products/11**;
- макетная плата: **https://www.sparkfun.com/products/12002**;
- соединительные провода: **https://www.sparkfun.com/products/9194**.

В части программного обеспечения вам понадобятся Arduino IDE, а также библиотеки aREST, PubSub и Adafruit DHT, которые мы уже использовали в этой книге ранее.

Сборка схемы

Устройство наше и в самом деле очень простое, потому что мы хотим лишь проверить связь между проектом и нашим собственным облачным сервером, который развернем в этой главе далее.

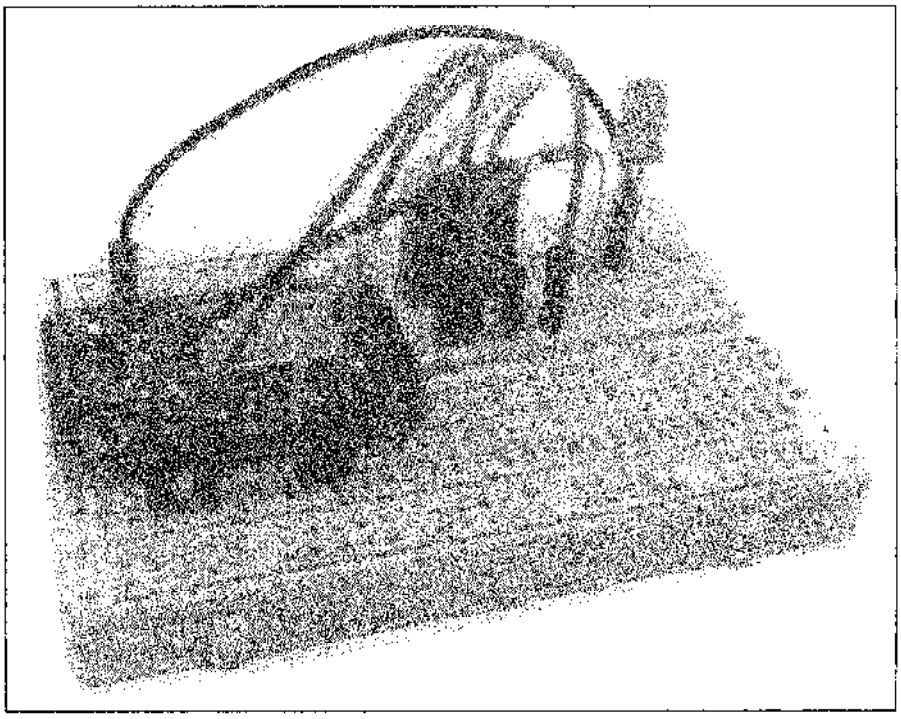


Рис. 13.1. Собранная монтажная схема проекта

[1] При использовании платы NodeMCU этот модуль не понадобится. — *Прим. пер.*

Итак, установите модуль ESP8266 на макетную плату и подсоедините к нему плату расширения FTDI. Соедините светодиод последовательно с резистором так, чтобы более длинный вывод светодиода был подключен к резистору. Затем соедините оставшийся вывод резистора с выводом 5 модуля ESP8266, а свободный вывод светодиода с линией GND модуля.

Проделав это, установите на макетную плату датчик DHT11 и соедините его левый вывод с линией питания VCC модуля ESP8266, его правый вывод — с линией GND модуля, и вывод, следующий за VCC, — к выводу 4 модуля ESP8266.

Собранная монтажная схема проекта показана на рис. 13.1.

Создание облачного сервера

Сейчас мы собираемся сделать первый шаг на пути к тому, чтобы подключить собранное устройство к вашему собственному облачному серверу, — создать сам сервер.

Вы могли бы просто запустить программу, которую мы рассмотрим позже, на собственном компьютере. Но тогда вы не сможете дистанционно подключиться к плате ESP8266, потому что ваш сервер должен быть развернут у облачного провайдера. Если у вас уже имеется сервер, на котором можно запустить приложение Meteor (фреймворк, который мы далее воспользуемся), вы можете пропустить этот раздел.

Существует множество хостинг-провайдеров, но я рекомендую Digital Ocean. Он быстрый, недорогой и оснащен очень простым в использовании интерфейсом. Вы можете найти его по адресу: **https://www.digitalocean.com/**.

Зайдя на их веб-сайт, создайте новый аккаунт. Затем создайте новый *дроплет* (Droplet), который станет именем сервера на Digital Ocean. Вас попросят выбрать географическое расположение сервера для развертывания дроплета — выберите расположение, наиболее близкое к месту нахождения вашего устройства (рис. 13.2).

Затем вам придется выбрать месячный тарифный план, от которого будет зависеть вычислительная мощность вашего сервера. Поскольку мы будем запускать только программу с малой нагрузкой на сервер, самого дешевого плана будет более чем достаточно (рис. 13.3).

Далее выберите операционную систему для вашего сервера — можно выбрать последнюю стабильную версию операционной системы Ubuntu Linux (рис. 13.4).

Наконец, завершите создание своего сервера присвоением ему имени (рис. 13.5).

Теперь вы должны увидеть, что ваш сервер создан и отображается в списке (рис. 13.6). Найдите там IP-адрес этого сервера, который пригодится вам позже.

Вам также следует настроить к своему серверу root-доступ. Поскольку это весьма сложный процесс, я рекомендую обратиться к инструкции по адресу: **https://www.digitalocean.com/community/tutorials/how-to-configuressh-key-based-authentication-on-a-linux-server**.

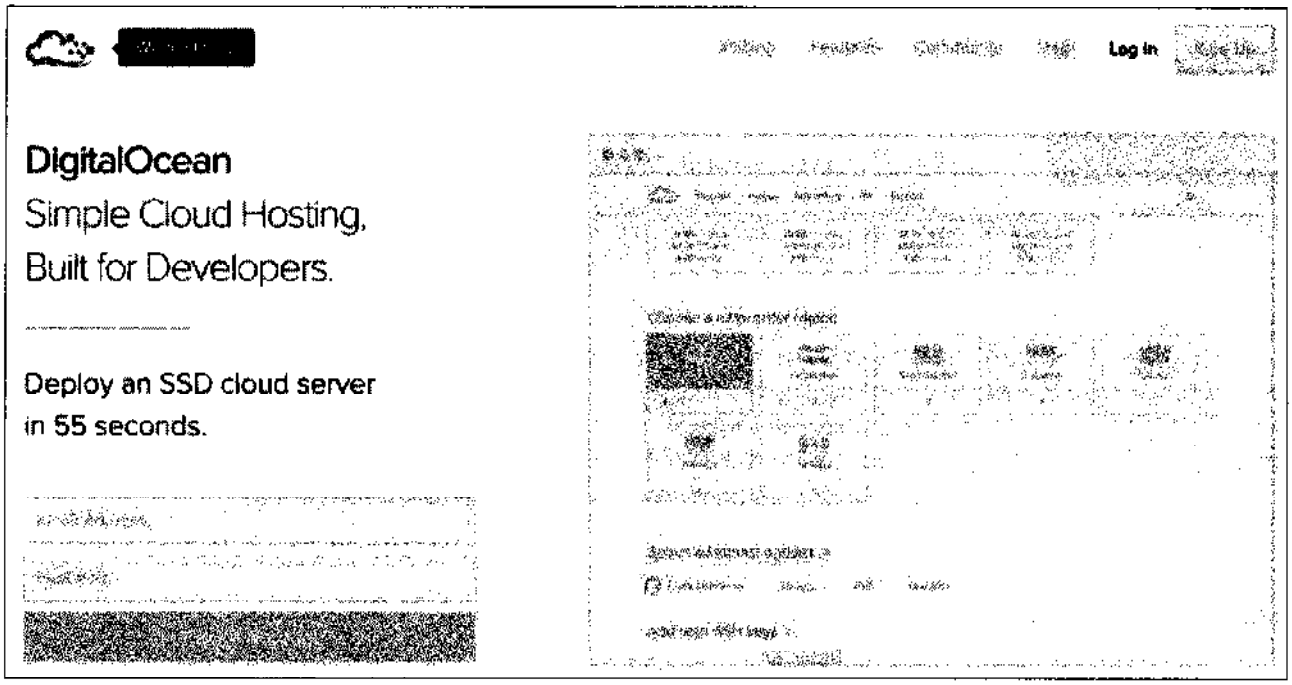


Рис. 13.2. Выберите географическое расположение дроплета

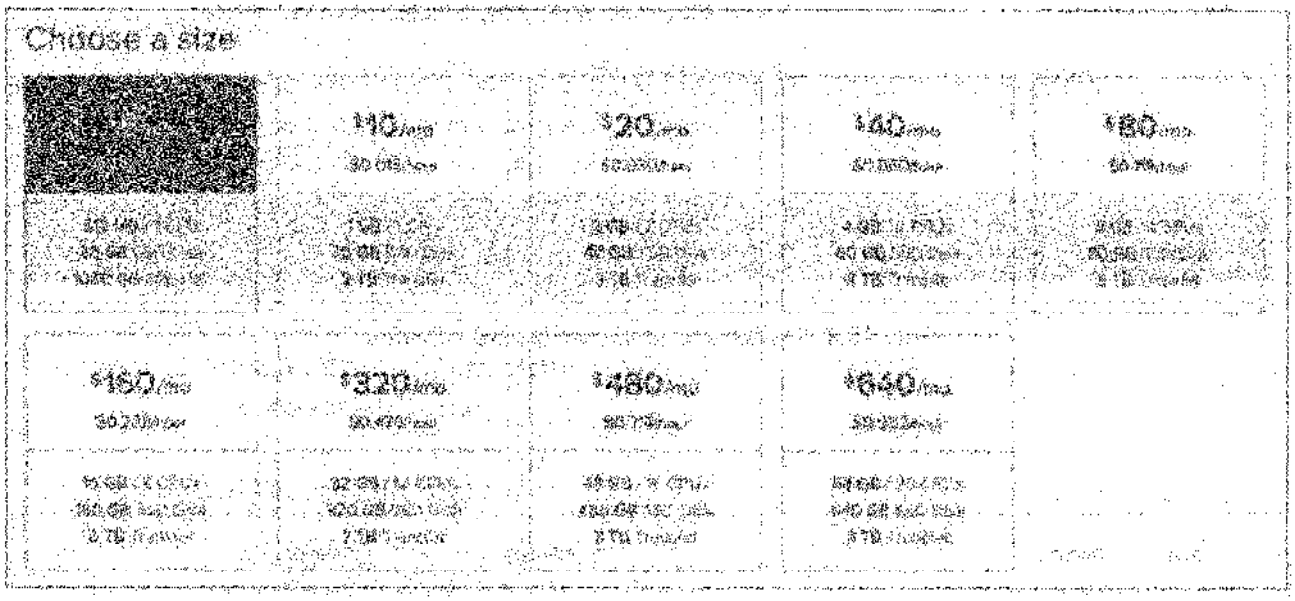


Рис. 13.3. Выберите самый дешевый тарифный план

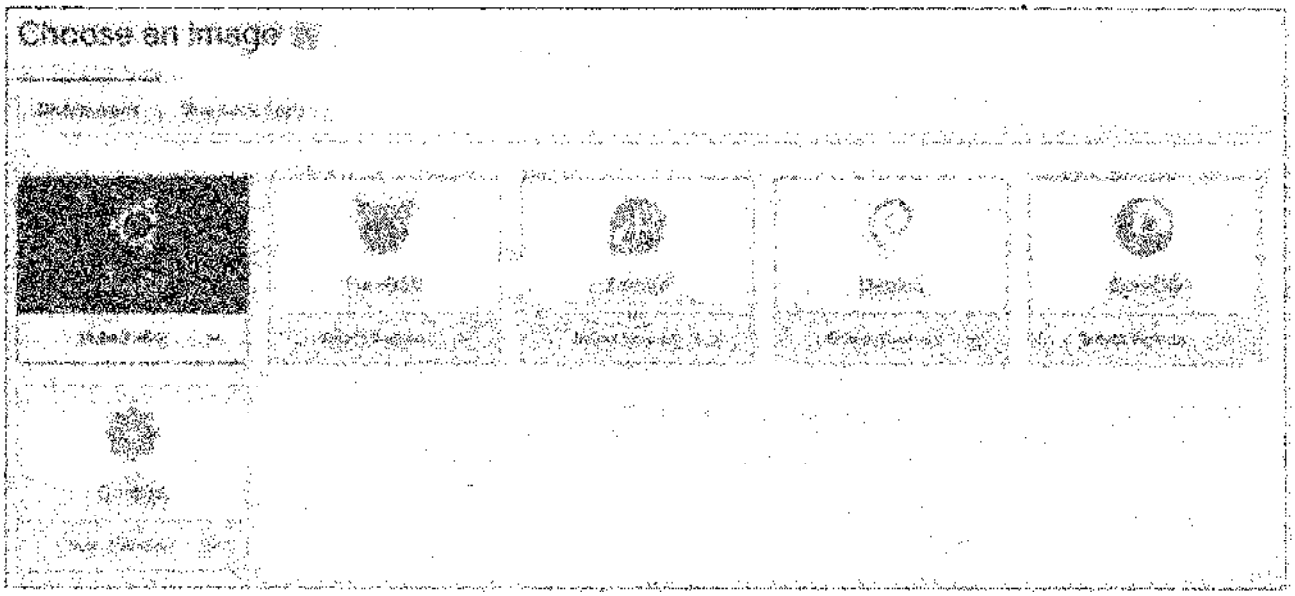


Рис. 13.4. Выберите операционную систему Ubuntu Linux

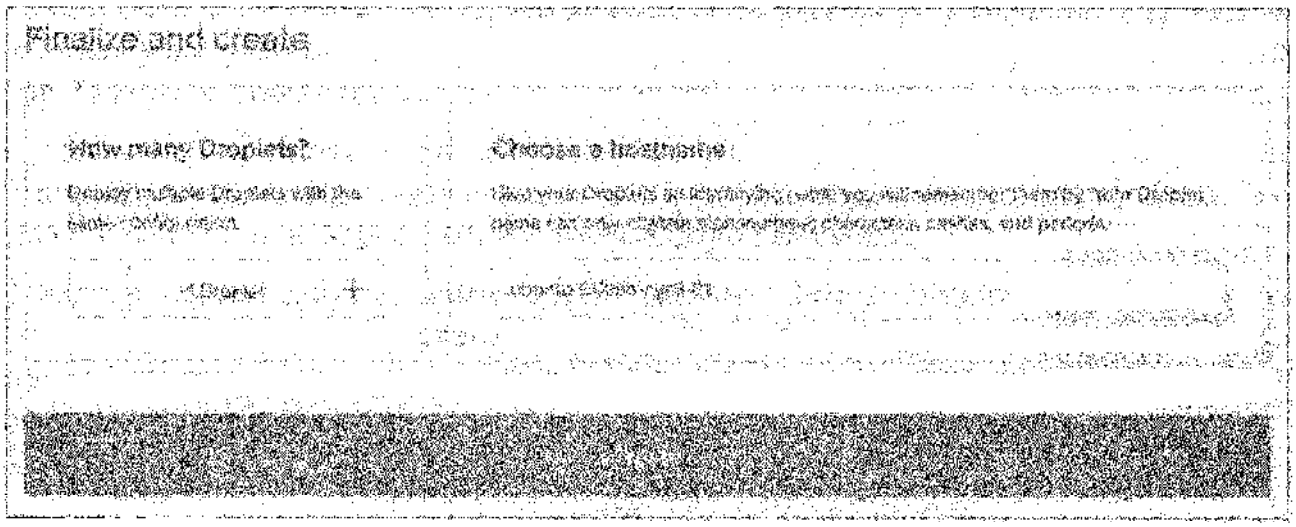


Рис. 13.5. Завершите создание своего сервера

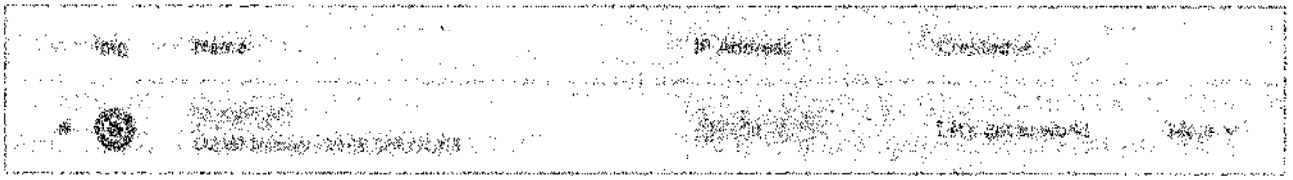


Рис. 13.6. Сервер успешно создан и отображается в списке

Исходный код облачного сервера aREST

Настало время приступить к установке приложения на сервер, который вы только что создали. В качестве программного обеспечения для вашего облачного сервера мы воспользуемся уже знакомым вам сервером aREST — это полностью бесплатный открытый проект, и вы можете скачать последнюю версию программы по адресу: **https://github.com/marcoschwartz/meteor-aREST-mqtt**.

Пока просто скопируйте все файлы из этого репозитория и поместите их в любую папку на своем компьютере.

Это приложение основано на пакете инструментов Meteor, который представляет собой законченный фреймворк на основе JavaScript и Node.js и может быть использован для создания мощных веб-приложений. Если вы еще не установили Meteor, займитесь этим сейчас, для чего перейдите по адресу: **https://www.meteor.com/install**.

Теперь давайте кратко рассмотрим код облачного сервера aREST. Он достаточно сложен, и я хочу выделить здесь лишь некоторые его части, чтобы вы поняли, как это работает.

Всякий раз, когда устройство подключается к серверу, программное обеспечение сначала проверяет, знакомо ли серверу это устройство. Если это так, мы подключаем устройство, и оно становится доступно из облака. Если устройство не опознано, оно добавляется в базу данных сервера.

Вот часть кода, которая отвечает за эту функцию:

```
Server.on('clientConnected', Meteor.bindEnvironment(function(client) {
        console.log('client connected', client.id);
// устройство знакомо?
if (Devices.find({clientId: client.id}).fetch().length == 0) {
        // если нет, добавить в базу данных
        console.log('New device detected');
        device = {
                clientId: client.id,
                connected: true
        }
        Devices.insert(device);
}
else {
        console.log('Existing device detected');
        // обновить статус устройства
        Devices.update({clientId: client.id}, {$set: {"connected":
true}});
        }
}) );
```

Всякий раз, когда пользователь хочет обратиться к устройству, он набирает команду в адресной строке браузера. Затем облачный сервер первым делом сравнивает эту команду со списком доступных команд, например:

```
Router.route('/:device', {
```

Далее сервер проверяет, существует ли такое устройство:

```
var currentDevice = Devices.findOne({clientId: device});
```

Если устройство существует и в данный момент подключено, сервер создает сообщение для отправки на устройство, например:

```
var message = {
   topic: currentDevice.clientId + '_in',
   payload: '',
   qos: 0,
   retain: false
};
```

После того как будет получен ответ от устройства, сервер пересылает его пользователю в формате JSON:

```
answer = sendMessage(message);
this.response.setHeader('Content-Type', 'application/json');
this.response.end(answer);
```

Развертывание сервера

Приступим к развертыванию программного обеспечения на сервере. Прежде всего вам нужно установить Node.js, если вы еще этого не сделали. Просто следуйте инструкциям на сайте **https://nodejs.org/en/**.

Затем мы можем установить программу под названием Meteor Up, которая заметно облегчит процесс развертывания приложения на вашем веб-сервере.

Откройте терминал и введите:

```
sudo npm install -g mup
```

Внутри текущей папки будет создан файл с именем mup.json следующего содержания:

```
{
// Server authentication info
"servers": [
{
"host": "0.0.0.0",
"username": "root",
//"password": "password"
// or pem file (ssh based authentication)
"pem": "~/.ssh/id_rsa"
}
],
```

```
// Install MongoDB in the server, does not destroy local MongoDB on
future setup
"setupMongo": true,
// WARNING: Node.js is required! Only skip if you already have Node.
js installed on server.
"setupNode": true,
// WARNING: If nodeVersion omitted will setup 0.10.36 by default. Do
not use v, only version number.
"nodeVersion": "0.10.41",
// Install PhantomJS in the server
"setupPhantom": true,
// Show a progress bar during the upload of the bundle to the
server.
// Might cause an error in some rare cases if set to true, for
instance in Shippable CI
"enableUploadProgressBar": true,
// Application name (No spaces)
"appName": "arest",
// Location of app (local directory)
"app": ".",
// Configure environment
"env": {
"ROOT_URL": "http://localhost",
"PORT": 3000
},
// Meteor Up checks if the app comes online just after the
deployment
// before mup checks that, it will wait for no. of seconds
configured below
"deployCheckWaitTime": 120
}
```

В этом файле вы должны исправить два момента. Первый — IP-адрес вашего сервера на Digital Ocean:

```
"servers": [
{
"host": "0.0.0.0",
"username": "root",
//"password": "password"
// or pem file (ssh based authentication)
"pem": "~/.ssh/id_rsa"
}
]
```

Второй — номер порта, назначаемого приложению, работающему на вашем сервере. По умолчанию это порт 3000, но вы можете сменить его в случае, если у вас есть несколько приложений на одном сервере:

```
"env": {
"ROOT_URL": "http://localhost",
"PORT": 3000
}
```

Закончив, сохраните файл и введите следующую команду:

```
mup setup
```

Она инициализирует процесс развертывания приложения Meteor на сервере (рис. 13.7).

```
MacBook-Pro:meteor-aREST-mqtt-prod marco$ mup setup

Meteor Up: Production Quality Meteor Deployments
--------------------------------------------------

" Checkout Kadira!
  It's the best way to monitor performance of your app.
  Visit: https://kadira.io/mup "


Started TaskList: Setup (linux)
[45.55.79.41] - Installing Node.js
[45.55.79.41] - Installing Node.js: SUCCESS
[45.55.79.41] - Installing PhantomJS
[45.55.79.41] - Installing PhantomJS: SUCCESS
[45.55.79.41] - Setting up Environment
[45.55.79.41] - Setting up Environment: SUCCESS
[45.55.79.41] - Copying MongoDB configuration
[45.55.79.41] - Copying MongoDB configuration: SUCCESS
[45.55.79.41] - Installing MongoDB
[45.55.79.41] - Installing MongoDB: SUCCESS
[45.55.79.41] - Configuring upstart
[45.55.79.41] - Configuring upstart: SUCCESS
MacBook-Pro:meteor-aREST-mqtt-prod marco$
```

Рис. 13.7. Инициализация сервера перед установкой приложения Meteor

```
MacBook-Pro:meteor-aREST-mqtt-prod marco$ mup deploy

Meteor Up: Production Quality Meteor Deployments
--------------------------------------------------

" Checkout Kadira!
  It's the best way to monitor performance of your app.
  Visit: https://kadira.io/mup "

Building Started: .

Started TaskList: Deploy app 'arest' (linux)
[45.55.79.41] - Uploading bundle
[45.55.79.41] - Uploading bundle: SUCCESS
[45.55.79.41] - Setting up Environment Variables
[45.55.79.41] - Setting up Environment Variables: SUCCESS
[45.55.79.41] - Invoking deployment process
[45.55.79.41] - Invoking deployment process: SUCCESS
MacBook-Pro:meteor-aREST-mqtt-prod marco$
```

Рис. 13.8. Развертывание облачного сервера

Как только вы увидите сообщение success, можете запустить сам процесс развертывания командой:

```
mup deploy
```

Эта команда запускает процесс установки (рис. 13.8).

Если вы увидели сообщение success зеленого цвета — поздравляю, теперь у вас развернут собственный облачный сервер, готовый к управлению вашими устройствами Интернета вещей!

Подключение ESP8266 к вашему облачному серверу

Протестируем сервер, который только что развернули в облаке. Для этого рассмотрим, как сконфигурировать ранее собранное устройство, чтобы оно могло соединиться с вашим собственным облачным сервером.

1. Прежде всего надо подключить необходимые библиотеки:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <aREST.h>
#include "DHT.h"
```

2. Затем определить вывод, к которому подключен датчик DHT11:

```
#define DHTPIN 4
#define DHTTYPE DHT11
```

3. Создаем экземпляр объекта датчика:

```
DHT dht(DHTPIN, DHTTYPE, 15);
```

4. Создаем клиента Wi-Fi для подключения к облачному серверу:

```
WiFiClient espClient;
PubSubClient client(espClient);
```

5. Создаем экземпляр объекта aREST. В этом месте вы должны подставить IP-адрес своего облачного сервера в качестве аргумента:

```
aREST rest = aREST(client, "19.434.34.23");
```

6. Присвойте своему устройству собственный уникальный идентификатор:

```
char* device_id = "01e47f";
```

7. Введите имя и пароль своей точки доступа Wi-Fi:

```
const char* ssid = "wifi-name";
const char* password = "wifi-pass";
```

8. Создайте две переменные, которые хранят данные измерений:

```
float temperature;
float humidity;
```

9. В функции `setup()` мы инициализируем датчик DHT11:

```
dht.begin();
```

10. Затем привязываем две переменные с данными измерений к API aREST:

```
rest.variable("temperature", &temperature);
rest.variable("humidity", &humidity);
```

11. В функции `loop()` получаем данные с датчика и поддерживаем открытым соединение с вашим облачным сервером:

```
// Измерение влажности
humidity = dht.readHumidity();
// Измерение температуры
temperature = dht.readTemperature();
// Соединение с облаком
rest.handle(client);
```

Настало время протестировать скетч, чтобы проверить, может ли он соединиться с облачным сервером, который мы только что развернули! Для этого скопируйте код из репозитория GitHub для этой книги, измените в коде настройки доступа Wi-Fi[2] и загрузите прошивку в плату[3].

После чего откройте окно монитора последовательного порта, чтобы увидеть, установлено ли соединение с вашим облачным сервером (рис. 13.9).

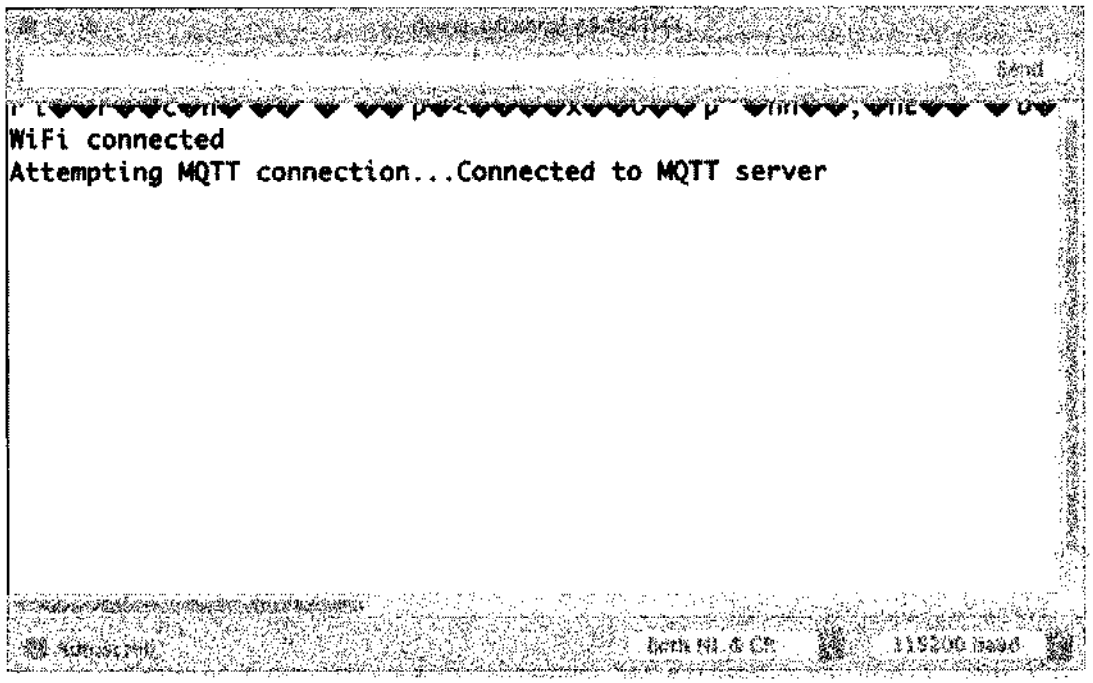


Рис. 13.9. Проверка соединения с облачным сервером

[2] Не забудьте также подставить IP-адрес своего облачного сервера и идентификатор устройства. — *Прим. пер.*

[3] Код прошивки для подключения к облачному сервису используется тот же самый, что приводился в предыдущих главах. — *Прим. пер.*

Если вы видите надпись **Connected to MQTT server**, это означает, что ваше устройство сейчас подключено к развернутому раньше серверу, и оно общается с сервером aREST через протокол MQTT. Если это не так, вернитесь к процессам, который мы рассмотрели в этой главе ранее, и убедитесь, что приложение aREST корректно установлено на облачном сервере.

Теперь вы можете проверить связь между вашим устройством и облачным сервером, как вы это делали раньше с сервером **arest.io**. Например, чтобы получить значение температуры, введите в адресной строке браузера команду: `http://19.434.34.23/01e47f/temperature`.

Разумеется, вам необходимо заменить в этой команде мой IP-адрес на адрес своего сервера. В окне браузера вы должны немедленно увидеть ответ:

```
{
"temperature": 26.00,
"id": "01e47f",
"name": "own_cloud",
"connected": true
}
```

Вы также можете настроить вывод 5 как выход командой: `http://19.434.34.23/01e47f/mode/5/o`.

Затем зажгите светодиод командой: `http://19.434.34.23/01e47f/digital/5/1`.

Поздравляю, теперь вы можете подключать устройства к собственному облачному серверу!

Заключение

В этой главе мы узнали, как развернуть собственный облачный сервер, чтобы получить полный контроль над своим Интернетом вещей на основе ESP8266. Для этого, использовав сервис Digital Ocean, мы создали свой собственный сервер, затем развернули на нем программное обеспечение aREST и, наконец, подключили модуль ESP8266 ко вновь развернутому серверу.

Разумеется, теперь вы можете подключить к этому серверу любой из своих проектов на ESP8266. Вы можете просто взять использующие aREST проекты, с которыми встретились в этой книге, и подключить их к собственному серверу. Вы можете немного повозиться с кодом и создать новые функции для вашего серверного приложения. В таком случае не стесняйтесь делиться своими достижениями с сообществом!

Приложение
Содержание электронного архива

Электронный архив с материалами к этой книге можно скачать с FTP-сервера издательства «БХВ-Петербург» по ссылке **ftp://ftp.bhv.ru/9785977538671.zip** или со страницы книги на сайте **www.bhv.ru**.

В архиве (табл. П.1) содержатся исходные коды (скетчи) программ для всех проектов, рассмотренных в книге. Программы адаптированы для работы с современными компонентами и новыми версиями API интернет-сервисов, которые действовали на момент подготовки перевода книги на русский язык[1]. Название папки, в которой размещен скетч Arduino, должно совпадать с названием скетча.

Таблица П.1. *Содержание электронного архива, сопровождающего книгу*

Файл	Описание
ch1_1	Проверка соединения с точкой доступа Wi-Fi
ch2_1	Программа управления светодиодом
ch2_2	Чтение логического уровня на входе ESP8266
ch2_3	Скачивание содержимого веб-страницы
ch2_DHT11	Чтение данных с датчика DHT11
ch3_1	Проверка датчика DHT11
ch3_Cloud	Загрузка данных в облачный сервис **Dweet.io**
ch4_1	Управление светодиодом через облачный сервис

[1] При воспроизведении проектов из книги следует иметь в виду, что в работе облачных сервисов aREST, Temboo и IFTTT в любой момент могут произойти непредсказуемые изменения. В этом случае некоторые программы из файлового архива могут перестать работать или будут работать некорректно. — *Прим. пер.*

***Таблица П.1** (окончание)*

Файл	Описание
ch5_Facebook	Публикация записи в сервисе Facebook
ch5_Twitter	Публикация записи в сервисе Twitter
ch5_Yahoo	Получение информации о погоде из сервиса Yahoo
ch6_BUTTON_board	Программа для модуля с кнопкой из *главы 6*
ch6_FOTO_board	Программа для модуля с фоторезистором из *главы 6*
ch6_LED_board	Программа для модуля со светодиодом из *главы 6*
ch7_EMAIL	Отправка email через сервис IFTTT
ch7_PUSH	Отправка push-уведомлений через сервис IFTTT
ch7_SMS	Отправка SMS через сервис IFTTT
ch8_DOOR_LOCK	Дистанционное управление дверным замком
ch8_DOOR_LOCK_PUSH	Дистанционное управление дверным замком с отправкой уведомления
ch9_TICKER	Монитор курса электронной валюты «биткоин»
ch10_ALERT	Отправка push-уведомления о пересохшей почве
ch10_AUTOMATED	Автоматическое включение поливочного насоса
ch10_MONITORING	Дистанционное наблюдение за влажностью и температурой почвы
ch10_SENSOR_TEST	Проверка датчика SHT10
ch11_DASHBOARD_LED	Программа для модуля со светодиодом из *главы 11*
ch11_DASHBOARD_MOTION	Программа для модуля с датчиком движения из *главы 11*
ch11_DASHBOARD_SENSOR	Программа для модуля с датчиком DHT11 из *главы 11*
ch11_LED_TIME	Программа включения и выключения светодиода в заданное время
ch11_SMS_ALARM	Отправка SMS о срабатывании сигнализации
ch12_MOTOR_TEST_A	Проверка схемы управления моторами. Программа адаптирована для работы с шилдом управления моторами (см. рис. Пр.6 в *предисловии к русскому изданию*)
ch12_ROBOT_CONTROL_A	Программа дистанционного управления моторами, адаптированная для работы с шилдом управления моторами (см. рис. Пр.6в *предисловии к русскому изданию*)
ch12_MOTOR_TEST_B	Исходная программа проверки схемы управления моторами полностью соответствует схеме в книге
ch12_ROBOT_CONTROL_B	Исходная программа дистанционного управления моторами полностью соответствует схеме в книге

Предметный указатель

М

О

С

У

Ф

Э