МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждениевысшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по выполнению практических работ
по дисциплине «Технологии Интернет-вещей»
Для студентов направления подготовки 09.03.02 Информационные системы и технологии, направленность (профиль) Информационные системы и технологии в бизнесе

(ЭЛЕКТРОННЫЙ ДОКУМЕНТ)

Содержание

Рецензия 5

Введение 8

Часть 1. Практика на базе комплекта интернет-вещей Arduino Uno R3 Starter Learning Kit с RFID-модулем 11

Введение 12
Состав комплекта 14
Немного о макетной плате, резисторах и безопасности 23
Практическое занятие 1. Hello, world! 28
Практическое занятие 2. Эксперимент с мигающим светодиодом 30
Практическое занятие 3. Эксперимент с контролируемой потенциометром яркостью свечения светодиода через порт PWM 31
Практическое занятие 4. Эксперимент с внешним мигающим светодиодом 34
Практическое занятие 5. Эксперимент с рекламной расцветкой 36
Практическое занятие 6. Светофорный эксперимент 38
Практическое занятие 7. Эксперимент с пищалкой 40
Практическое занятие 8. Эксперимент с датчиком наклона 42
Практическое занятие 9. Эксперимент с чистым входным сигналом 44
Практическое занятие 10. Расширенный эксперимент с чистым сигналом 47
Практическое занятие 11. Эксперимент по чтению аналогового значения 49
Практическое занятие 12. Эксперимент по управлению звуком и светом 52
Практическое занятие 13. Эксперимент с датчиком огня 54
Практическое занятие 14. Эксперимент с вольтметром 57
Практическое занятие 15. Эксперимент с распознаванием голоса 59
Практическое занятие 16. Эксперимент с температурным сенсором 62
Практическое занятие 17. Разноцветный термостат 64
Практическое занятие 18. Эксперимент с одноразрядным цифровым светодиодным индикатором 66
Практическое занятие 19. Эксперимент с четырёхразрядным цифровым светодиодным индикатором 71
Практическое занятие 20. Эксперимент со светодиодной матрицей 77
Практическое занятие 21. Эксперимент с трёхцветным светодиодом 83
Практическое занятие 22. Эксперимент с модулем 74HC595 86
Практическое занятие 23. Кнопочный модуль 4×4 и библиотеки 89
Практическое занятие 24. Часы реального времени DS1307 93

Практическое занятие 25. Эксперимент с датчиком уровня воды 97
Практическое занятие 26. Эксперимент с сенсором температуры и влажности DHT11 100
Практическое занятие 27. Эксперимент с релейным модулем 102
Практическое занятие 28. Эксперимент с жидкокристаллическим монитором LCD1602A....104
Практическое занятие 29. Эксперимент с шаговым двигателем 107
Практическое занятие 30. Эксперимент с серводвигателем 110
Практическое занятие 31. Эксперимент с игровым джойстиком 113
Практическое занятие 32. Эксперимент с инфракрасным пультом дистанционного управления 115
Практическое занятие 33. Эксперимент с RFID-модулем RC522 120
Практическое занятие 34. Эксперимент с системой контроля доступа 123

Часть 2. Практика на Raspberry Pi 3 (модель B) 126

Введение 127
Установка ОС Android Things 129
Первый проект в ОС Android Things – трёхцветный светодиод....131
Второй проект в Android Things – система сигнализации....135
Третий проект в Android Things – система мониторинга окружающей среды 147
Четвёртый проект в Android Things – объединение Android Things с облачной платформой интернета вещей 163
Пятый проект в Android Things – шпионский глаз....188

Заключение 200

Список использованных источников 202

Введение

Интернет вещей – активно развивающееся направление, несмотря на отсутствие единых стандартов и разнообразие платформ и самих интернет-вещей, предлагающихся различными компаниями в виде готовых решений со своим дизайном. Существует огромное количество определений этого термина, однако в большинстве случаев под интернетом вещей понимается сбор и обмен данными между различными физическими устройствами (также называемыми умными устройствами, подключёнными устройствами, интернет-вещами и т. д.) на основе определённой сети (здесь иногда добавляют фразу «там, где раньше это было невозможно»). Физическими устройствами могут быть автомобили, здания, камеры, бытовая техника, компьютерная техника и любые другие устройства, оснащённые электроникой, программным обеспечением, сенсорами, двигателями и модулями для подключения к сети интернет, даже города. Данные, получаемые с физических устройств, как правило, собираются в определённом сетевом хранилище – дата-центре, облаке, сервере, репозитории и т. д., которые затем могут быть подвержены анализу методами data mining, machine learning, cloud computing и другими с целью решения тех или иных задач, стоящих перед разработчиками. Для хранения, обработки и визуализации этих данных, а также для предоставления различных сервисов для управления интернет-вещами и анализа данных существуют различные облачные платформы для интернета вещей.

Если взглянуть на кривую компании Гартнер по появляющимся технологиям (emerging technologies), то можно увидеть интернет вещей на пике в 2014–2015 годах, исчезновение IoT в 2016–2017 годах и затем появление на кривой в 2018 году на пути к области спада и разочарования. Однако реальные надёжные технологии (второй версии) появляются только после прохождения через эту впадину избавления от иллюзий. Кроме того, если мы возьмём 2016–2017 годы, на кривой Гартнер область интернета вещей представляет другое направление – платформы для интернета вещей, – активно развивающееся в настоящее время и находящееся на пике этой кривой в 2018 году. В одном из практических заданий второй части данного учебного пособия применяется одна из облачных платформ для интернета вещей и две небольшие вспомогательные платформы со своими сервисами.

С интернетом вещей вплотную я познакомился около полутора лет назад. До этого я, конечно, слышал об этом направлении, или области, – ведь один из студентов, руководителем которого был Восков Л. С., ещё в МИЭМе показывал мне свой проект по умной розетке, которая контролировалась удалённо через вебинтерфейс и зажигала включённую в неё настольную лампу. На слуху были разнообразные умные вещи – умный дом, умный автомобиль, умный холодильник, умная подставка для яиц в холодильнике и т. д. Безусловно, тогда это всё было непросто – датчики и сенсоры, платы и прочее оборудование стоили немало.

Но потом появилась потребность подготовить курс «Экосистемы интернета вещей», и… началось. Прежде всего было не понятно, что делать с практическими занятиями по этому курсу – что давать студентам? «Железок» никаких не было и в помине, в то же время очень хотелось, чтобы студенты работали на реальном оборудовании. Но на каком? Как построить практику? Времени оставалось всё меньше, и тогда я решил попробовать купить комплект с Raspberry Pi 3 – тогда, в 2017 году, несмотря на то что операционная система Android Things была ещё без официального релиза первой версии, по ней были довольно интересные примеры и проекты на официальном сайте разработчиков под Android Things, и не только там. Однако, начав разбираться с системой, я понял, что начинать надо не с этого, а с самых основ, с физики, схемотехники, которых у студентов-программистов, для которых предназначался курс, конечно же, не было. Поэтому вторым шагом была покупка комплекта с платой Arduino Uno – такого, в котором было бы максимально возможное количество «железок». Этот комплект заставил вспомнить основы физики, научиться терпению при определении ошибки в проекте – в схеме она, или в компонентах, или в коде, или в методичке, или где? – и многому другому, в том числе пришлось пропустить через себя методичку с огромным количеством ошибок на ломаном английском языке, которая была на сайте магазина, продававшего комплект. Так появилась первая часть этой книги. Тогда же, когда в моём распоряжении оказались Raspberry Pi 3, HDMI-мини-монитор и Android Things, на книжной полке внезапно появилась и книга Android-разработчика Франческо Эззолы «Android Things Projects», которую пришлось печатать на заказ – недешёвое удовольствие. И, несмотря на устаревший на 1 год код, английский язык и некоторые ошибки в заданиях, или просто иногда пропуск некоторых важных деталей кода в книге, в этом году выросла вторая часть этой книги, посвящённая нескольким проектам под Android Things на плате Raspberry Pi 3. Они приведены к реалиям существующей в настоящее время 1-й версии данной операционной системы и переработаны с целью добавления пользовательского интерфейса, которого в оригинале в некоторых заданиях просто нет. Осваивая книгу и пропуская задания через себя, я понимал, что вот оно – как это просто сделать, оказывается: контроль и полив растения удалённо через смартфон, контроль за удалённой квартирой с помощью ИК-датчика и камеры, пока ты находишься в отпуске, или – получение всей информации о погоде за окном на твоём мониторе в комнате в виде приложения Android Things, включая прогноз погоды.

Данное учебное пособие предлагает повторить этот путь. Первая часть этой книги предназначена для того, чтобы читатели (студенты) освоили основы схемотехники. Да-да, именно в этом и заключается предназначение комплекта модулей, датчиков, проводов, экранов, двигателей, джойстиков, индикаторов, карт и ключей, резисторов и светодиодов с кнопками, вместе с макетной платой и платой Arduino Uno – для того чтобы читатели освоили прежде всего схемы подключения, схемотехническую базу и вспомнили основы физики в части тока и напряжения, а также номиналы резисторов, как их посчитать, – на реальных устройствах, которые могут выйти из строя или работать неправильно при неправильном подключении. Задача же второй части данного учебного

пособия – перейти от схемотехники к реальным проектам для интернета вещей, с их типовой архитектурой, передачей данных от сенсоров к плате, от платы в облако, из облака в мобильное приложение-компаньон, позволяющее осуществить управление платой удалённо, или визуализировать и проанализировать полученные данные, и главное – воплотить в жизнь реальную архитектуру приложений для интернета вещей.

Практические эксперименты первой части учебного пособия построены по определённой схеме. Сначала перечисляются компоненты, необходимые для выполнения задания. Затем приводится описание задания – та часть, которую все обычно пропускают (особенно студенты). После этого дается сначала принципиальная электрическая схема соединения компонентов, а затем – более интересная цветная схема с макетной платой, на которую обычно и ориентируется большинство читателей (она красивее). Затем идёт код программы (иногда их несколько), который необходимо скопировать в Arduino IDE и запустить. Для того чтобы развлечь читателя, комментарии в исправленном коде оставлены без изменений, как они есть в оригинале [9], – на ломаном английском.

Эксперименты из второй части построены немного по-другому – сначала идёт небольшое введение, потом – перечень компонентов и их свойства и изображения. После этого приводится схема подключения компонентов, причём только цветная, с макетной платой. А затем идут этапы выполнения задания, в которых присутствует как исходный код для копирования в Android Studio, так и пояснения к этому коду, а также, при необходимости, скриншоты различных ресурсов и платформ интернета вещей, используемых для выполнения задания.

Отдельно хочется выразить слова благодарности студентам 3-го курса 2017–2018 и 2018–2019 учебных годов образовательной программы «Программная инженерия» департамента программной инженерии факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики», на которых проходила апробация всех заданий из этого пособия. Они не только находили и исправляли ошибки и неточности в некоторых заданиях, но и предлагали разумные вещи для улучшения методических указаний, послуживших основой для этой книги, и даже предоставили материал для научной статьи. И конечно, нельзя не сказать эти слова руководителю упомянутого департамента – Авдошину С. М., который в своё время ошарашил меня курсом «Экосистемы интернета вещей», без которого не было бы никакого учебного пособия.

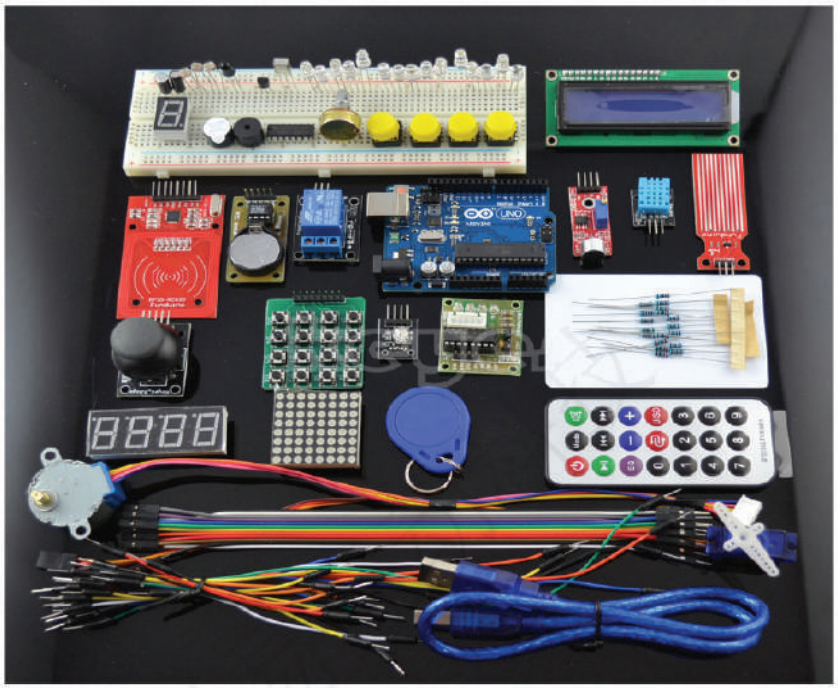
Для выполнения заданий из этого учебного пособия необходимо приобрести, как минимум, 2 комплекта: [1] и [21]. Также отдельно могут понадобиться дополнительные модули и датчики для выполнения заданий из 2-й части учебного пособия, а именно: модуль с Raspberry Pi камерой v 2.1, PIR сенсор и датчик давления, влажности и температуры BME280.

Данная книга выложена в электронном виде на сайте [22], поэтому код из любого рассмотренного практического задания можно скопировать непосредственно из электронной версии.

При подготовке этого учебного пособия не только ни одна плата, но и ни один светодиод, модуль или датчик не пострадал.

Часть 1

Практика на базе комплекта интернет-вещей Arduino Uno R3 Starter Learning Kit с RFID-модулем



Основано на 34 практических заданиях на английском языке [9]

Введение

Arduino – наиболее популярная платформа для разработки как простых, так и достаточно сложных проектов для интернета вещей. Популярность этой платформы обусловлена не только её низкой ценой, но и огромным количеством обучающих материалов, примеров проектов в сети, в том числе и на таких ресурсах, как YouTube, различных форумов разработчиков, а также хорошим официальным сайтом [2, 3], на котором Arduino Team постоянно предлагает ознакомиться с новыми проектами на базе одной из плат Arduino. Существует множество разновидностей плат Arduino, например: Uno, Leonardo, 101, Mega, Zero, Ethernet, Gemma, MKR FOX 1200 и т. д. Все платы на официальном сайте разделены на категории: начальный уровень (для обучения, к нему относится Uno), платы с расширенными функциями, платы для интернета вещей, платы для носимых устройств (в основном в тандеме с Lillypad, например для умной одежды). Несмотря на простоту среды Arduino IDE и её недостатки, в ней можно разрабатывать интересные и сложные проекты; кроме того, существует официальный онлайн-аналог среды и многочисленные библиотеки, способные нарастить её небольшой функционал. Именно поэтому для данного учебного пособия выбрана эта платформа, а конкретно Arduino Uno версии R3, и наиболее богатый и разнообразный в отношении количества различных модулей, сенсоров, датчиков и других компонентов комплект с этой платой – комплект интернет-вещей для начинающих на базе Arduino Uno версии R3 (Starter Learning Kit) с RFID-модулем [1].

Arduino Uno Rev. 3 – лучшая плата для начинающих. Согласно официальному сайту, эта плата является наиболее используемой и обладает наибольшим количеством документации из всех плат семейства Arduino. Arduino Uno – это плата на основе 8-битного микроконтроллера ATmega328P, см. рис. 1 – самая большая чёрная деталь на плате. Таким образом, по сути, плата Arduino Uno является платой расширения, или платой разработчика (developer board), сердце которой – ATmega328P.

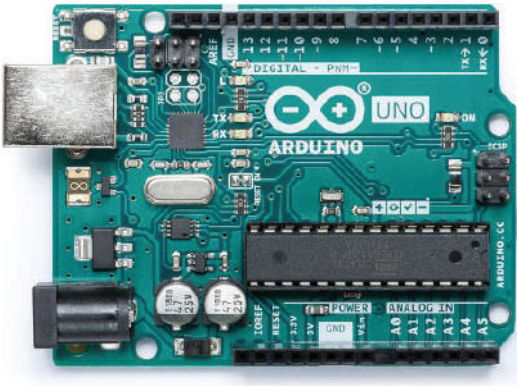


Рис. 1 ❖ Лицевая сторона платы Arduino Uno

На плате есть и другой микроконтроллер – ATmega16U2, служащий для связи микроконтроллера ATmega328P и USB порта платы (на рис. 1 – чёрный квадрат слева от TX и RX). У Arduino Uno есть 14 цифровых выходов (пинов) ввода/вывода, обозначенных на плате цифрами (из них 6 ШИМ (PWM, Pulse-Width Modulation) выходных пинов широтно-импульсной модуляции, обозначенных символом ~), 6 аналоговых входов – A0-A6 (с разрешением в 10 бит, т. е. 1024 различных значения), кварцевый кристалл-резонатор на 16 Мгц, выход USB-Bf (на рис. 1 – слева вверху под кнопкой перезагрузки), выход для подключения питания от адаптера питания (7–12 В) или батарейки на 9 В (обычно – в виде прямоугольного параллелепипеда; выход находится слева внизу на рис. 1), ICSP-разъём (In Circuit Serial Programming, программирование по последовательному протоколу чипа, уже подключённого в некоторую схему, или просто – программирование контроллера внутри схемы; на рис. 1 – посередине правого края) и кнопка перезагрузки (слева вверху, см. рис. 1). Кроме этого, плата располагает тремя пинами земли (GND), одним пином на 5 В, одним – на 3.3 В, Vin-пином для подключения внешнего источника питания или для получения напряжения, если плата подключена к внешнему адаптеру питания через разъём питания (через USB-соединение питание ограничивается 5 В), IOREF-пином (Input Output Reference – информация о напряжении микроконтроллера) и встроенным светодиодом L (или 13).

Микроконтроллер ATmega328 на Arduino Uno поставляется уже с загрузчиком (bootloader), что позволяет загружать код без использования внешнего программатора. При желании можно обойти загрузчик и запрограммировать микроконтроллер через ICSP. ATmega328 обладает 32 Кб встроенной памяти (0.5 Кб из которых отведено загрузчику). В дополнение ко всему некоторые пины платы имеют несколько функций, например пины 2 и 3 могут использоваться для вызова внешних прерываний при некоторых событиях, например при событии смены высокого сигнала на пине на низкий (falling edge). Мы не будем углубляться в мельчайшие подробности характеристик платы – приведённой информации вполне достаточно для знакомства с платой и выполнения 34 практических заданий части 1 этого учебного пособия. Также здесь не будет приведена распиновка платы, так как, по сравнению с платой Raspberry Pi 3, с которой мы познакомимся в части 2 этого учебного пособия, для Arduino Uno делать распиновку нет смысла – все пины уже подписаны на плате, см. рис. 1.

Далее рассмотрим, что же входит, помимо самой платы Arduino Uno R3, в состав упомянутого комплекта интернет-вещей для начинающих на базе Arduino Uno версии R3 (Starter Learning Kit) с RFID-модулем. Этот комплект был выбран также и потому, что позволяет сделать очень много практических заданий на основе различных компонентов и устройств, которые в него входят: конечно же, число возможных проектов на его основе далеко не ограничивается 34 экспериментами, приведёнными в этой части учебного пособия, а ограничивается лишь фантазией разработчика. Практические задания предусмотрены для

всех компонентов этого комплекта, чтобы познакомить читателя со всеми возможными составными частями набора и их функциями. В следующем разделе приведены реальные фотографии компонентов комплекта, как они выглядят на самом деле, а не схематичные изображения или фотографии из интернета.

Состав комплекта

Комплект интернет-вещей Arduino Uno Starter Learning Kit с RFID-модулем [1] состоит из следующих компонентов, показанных на рис. 2–38.

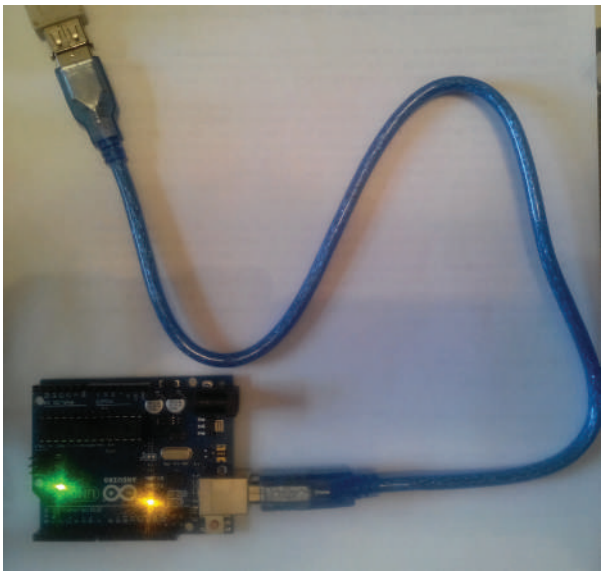


Рис. 2 ❖ Плата Arduino Uno R3 + USB-кабель (Am-Bm)

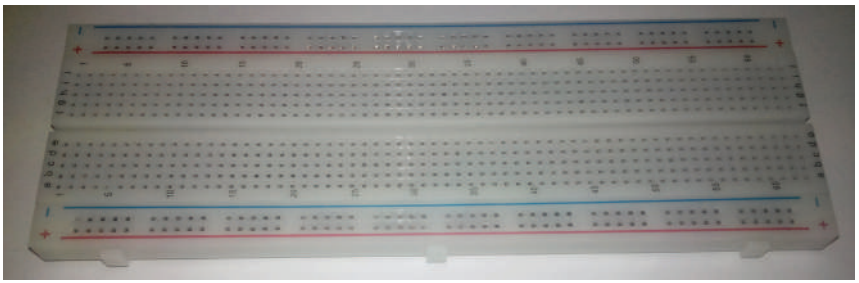


Рис. 3 ❖ Макетная плата (breadboard)

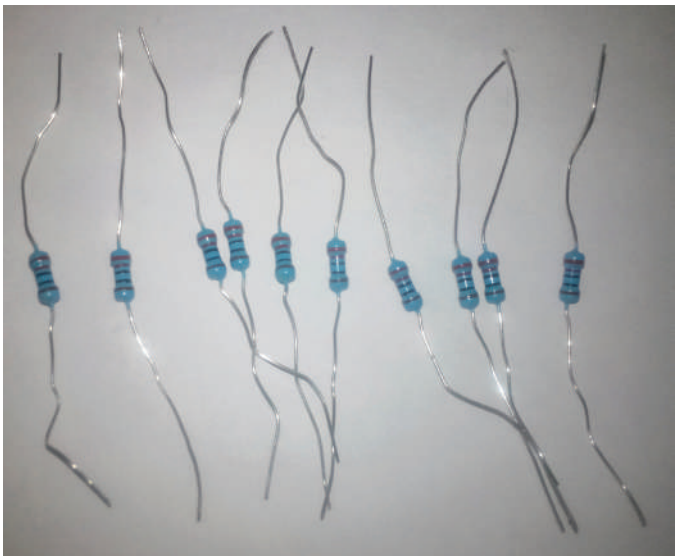


Рис. 4 ❖ Резисторы на 220 Ом, 10 шт.

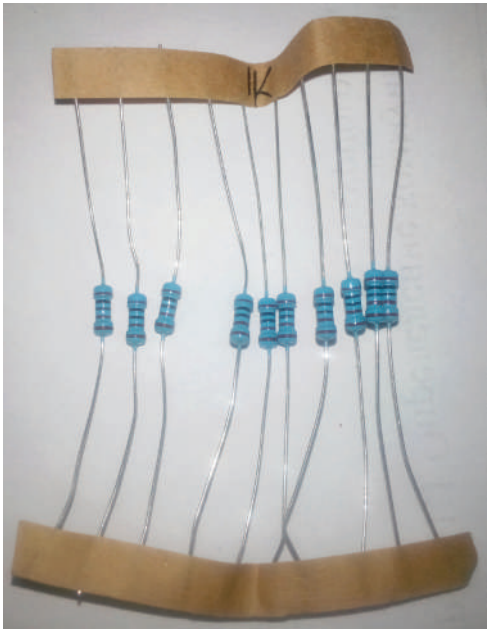


Рис. 5 ❖ Резисторы на 1 кОм, 10 шт.

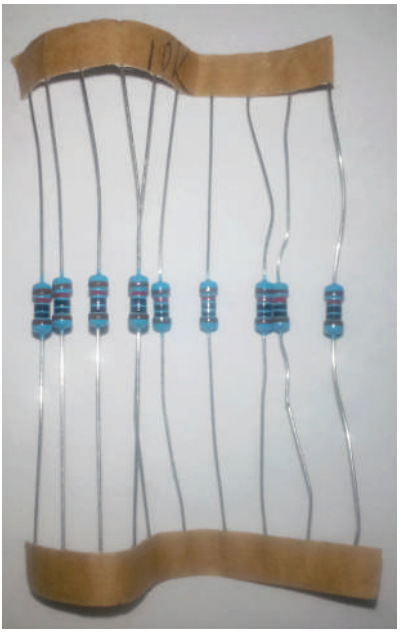


Рис. 6 ❖ Резисторы на 10 кОм, 10 шт.

Рис. 7 ❖ Светодиоды, 15 шт. (5 синих, 5 жёлтых, 5 красных)

Рис. 8 ❖ Кнопки, 4 шт.

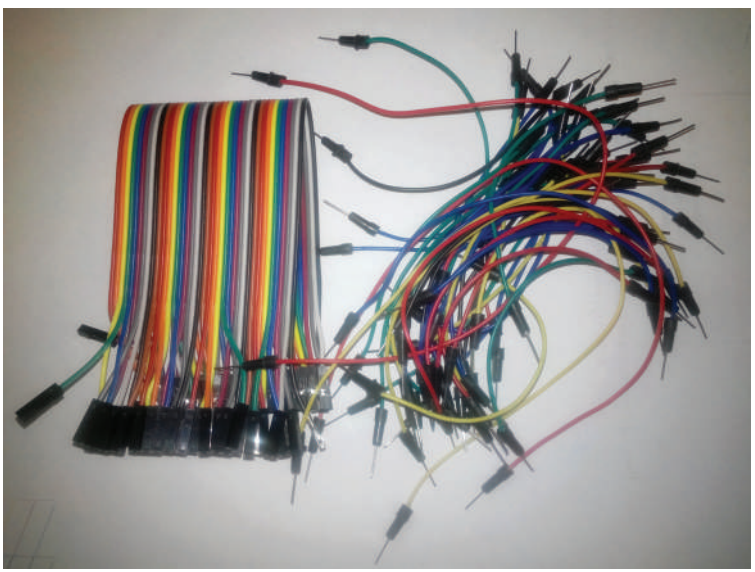


Рис. 9 ❖ Соединительные провода: m-m и f-f

Рис. 10 ❖ Шнур питания от батарейки на 9 В для Arduino Uno

Рис. 11 ❖ Динамик-пищалка, 2 шт.

Рис. 12 ❖ Датчик наклона (tilt switch), 2 шт.

Рис. 13 ❖ Потенциометр (10 КОм)

Рис. 14 ❖ Фоторезистор, 2 шт.

Рис. 15 ❖ Инфракрасный датчик огня

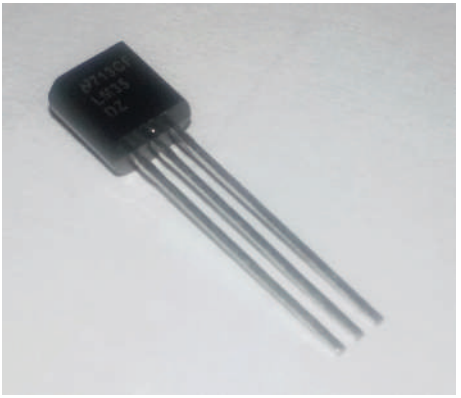


Рис. 16 ❖ Температурный сенсор LM35

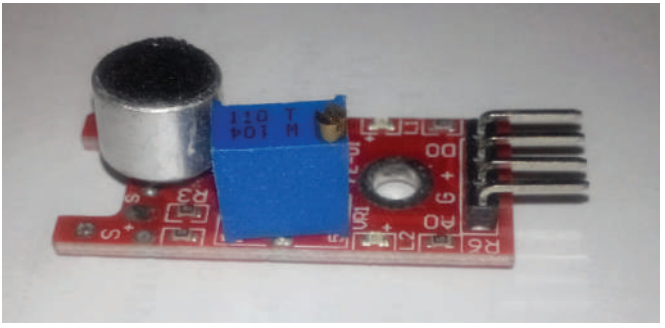


Рис. 17 ❖ Датчик уровня шума

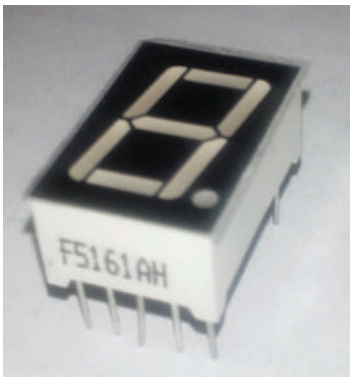


Рис. 18 ❖ Одноразрядный цифровой светодиодный индикатор

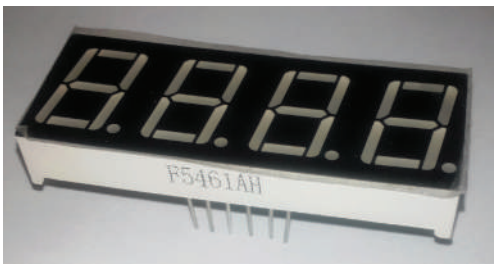


Рис. 19 ❖ Четырёхразрядный цифровой светодиодный индикатор

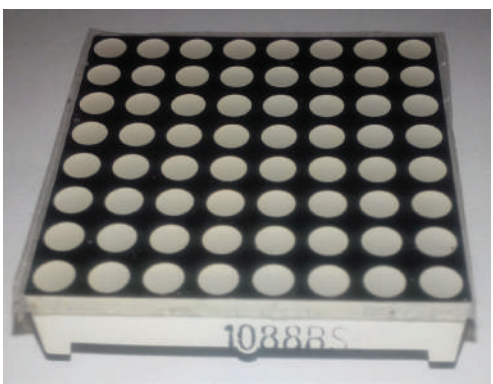


Рис. 20 ❖ Светодиодная матрица 8×8

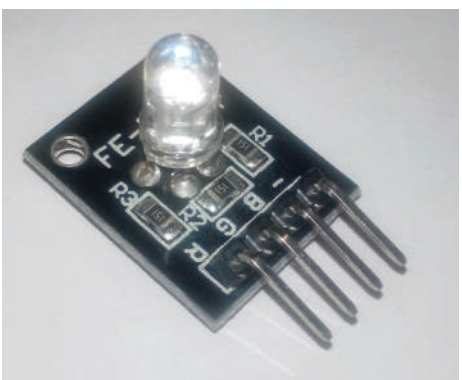


Рис. 21 ❖ Трёхцветный светодиод с общим катодом (на модуле)

Рис. 22 ❖ Модуль 74HC595

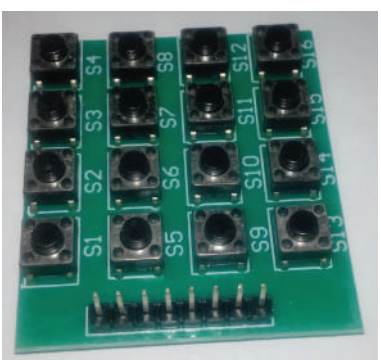


Рис. 23 ❖ Кнопочный модуль 4×4

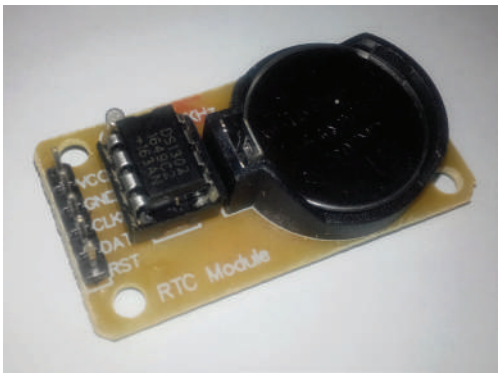


Рис. 24 ❖ Часы реального времени RTC DS1307

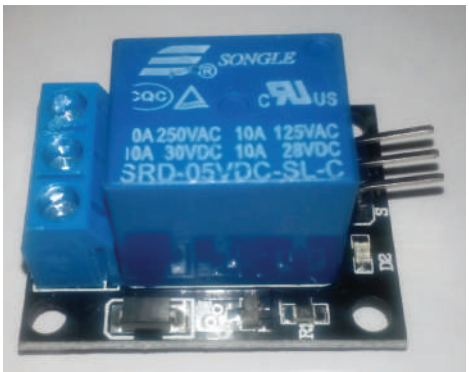


Рис. 25 ❖ Релейный модуль

Рис. 26 ❖ Датчик уровня воды (Water Sensor)

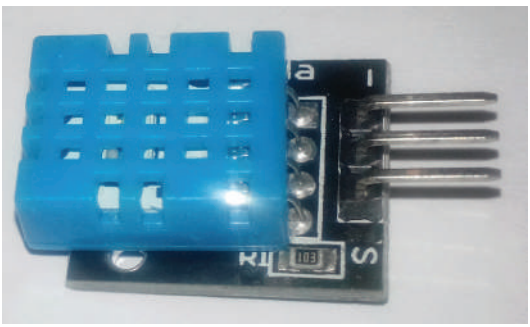


Рис. 27 ❖ Сенсор температуры и влажности DHT11

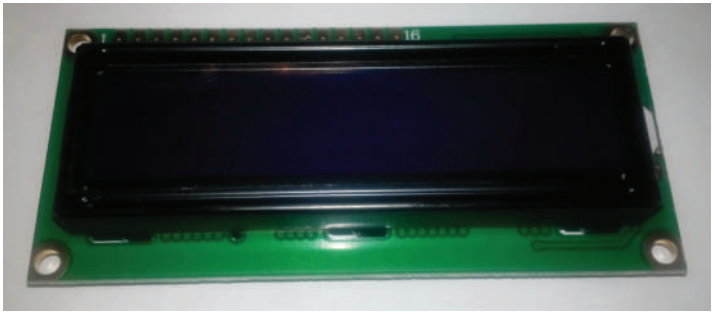


Рис. 28 ❖ Жидкокристаллический монитор (дисплей) LCD1602A

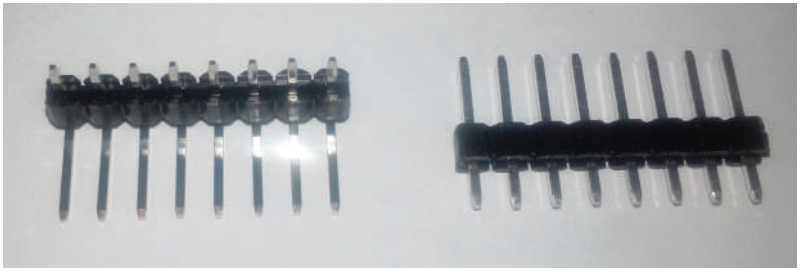


Рис. 29 ❖ Два штырьковых коннектора по 8 пинов каждый

Рис. 30 ❖ Шаговый двигатель

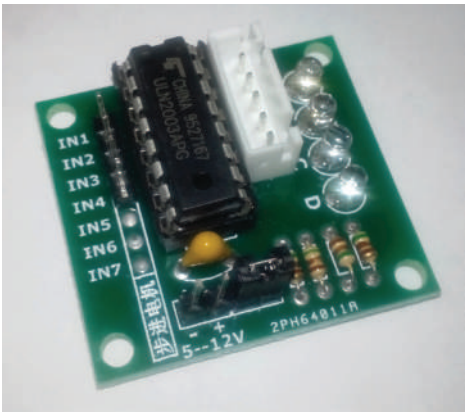


Рис. 31 ❖ Модуль для шагового двигателя

Рис. 32 ❖ Серводвигатель (сервопривод) с комплектом насадок

Рис. 33 ❖ Игровой джойстик

Рис. 34 ❖ Инфракрасный пульт дистанционного управления NEC

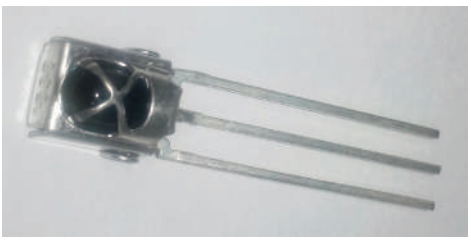


Рис. 35 ❖ Инфракрасный приёмник

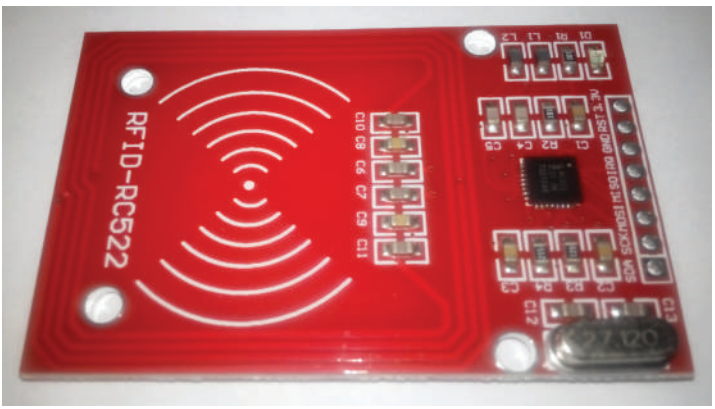


Рис. 36 ❖ RFID-модуль RC522

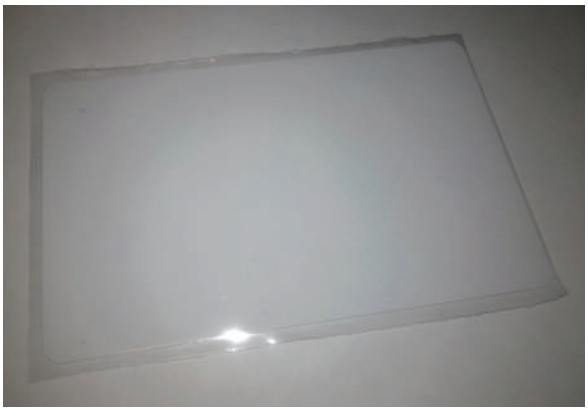


Рис. 37 ❖ RDIF-карта

Рис. 38 ❖ RFID-ключ

Немного о макетной плате, резисторах и безопасности

Как в этой, так и во второй части книги вам понадобятся знания о том, что такое макетная плата, как включать в схему светодиод и как отличить один резистор от другого. Но сначала – о безопасности.

Главное правило обращения с электричеством, компонентами и модулями гласит: помните, что как вы можете повредить технику, так и она может нанести вам вред! Перед тем как выполнять задания, нужно помнить о простых правилах работы с электронными компонентами и тем более системами на модуле (SoM, System on Module), к которым относится Arduino Uno, платами и прочими электронными изделиями:

- собирать и разбирать/менять схему можно только при выключенном питании (отсоединённом USB-кабеле) – имейте терпение;
- светодиоды и другие чувствительные компоненты подключаются строго согласно схеме – через резисторы;
- не стоит путать питание с землёй, плюс с минусом;
- никакого статического электричества! Если на вас свитер из синтетики или шерсти, если вы любите часто поправлять свои волосы, заземляйте свои руки, перед тем как дотрагиваться до электронных изделий (дотроньтесь до корпуса компьютера, железной ножки стола, батареи и т. д.)!

Макетная плата – удобное средство для соединения электрических компонентов в простые схемы и даже в схемы среднего уровня сложности. Макетные платы бывают разных типов, но в основном выделяют два типа: с разрывом горизонтальных линий земли и питания сверху и снизу посередине и без разрыва. В комплекте вам могут попасться оба типа. Макетная плата без разрыва горизонтальных линий земли и питания сверху и снизу показана на рис. 3. Если разрыва нет, это явно показывается синими и красными линиями: на

рис. 3 линии идут непрерывно, значит, разрыва нет. В случае наличия разрыва красные и синие линии прерываются посередине платы.

Соединения макетной платы без разрыва линий земли и питания показаны на рис. 39. Соединения макетной платы с разрывом этих линий, соответственно, проходят снизу и сверху по горизонтали от краёв только до середины макетной платы. Что же касается вертикальных соединений, у макетных плат обоих рассмотренных типов соединения прерываются 3 раза по вертикали: между зелёными разъёмами верхнего и нижнего рядов на рис. 39 нет соединения, так же, как и между зелёными и красными рядами.

Все принципиальные схемы и схемы с макетной платой в этой книге нарисованы с помощью наиболее распространённой и популярной открытой библиотеки + редактора электронных компонентов и схем Fritzing [10].

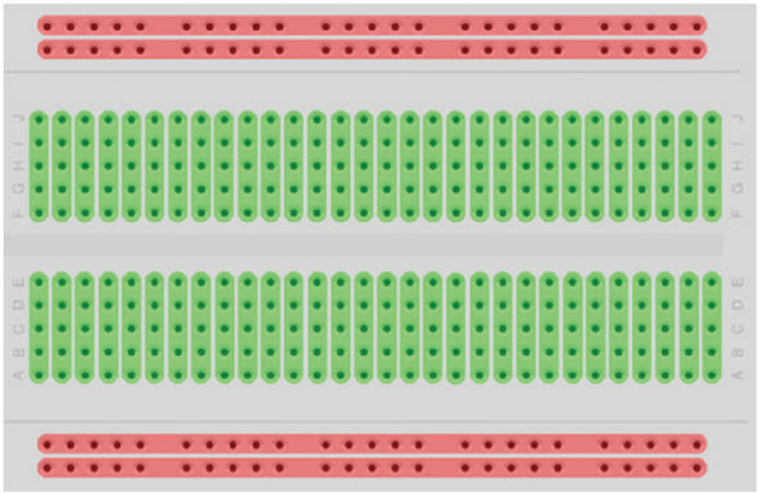


Рис. 39 ❖ Соединения макетной платы без разрывов верхних и нижних горизонтальных линий питания и земли

Теперь – светодиод. У каждого светодиода есть короткий и длинный выходы (пины) – см. рис. 7. Они представляют, соответственно, катод и анод. На схеме у светодиодов эта особенность выражена следующим способом: более длинный пин (анод) изогнут у основания цветной колбы светодиода, более короткий же (катод) входит в колбу прямо, без изгиба (см., например, рисунок с макетной платой к практическому занятию 3 в этой части учебного пособия). При подключении светодиода надо помнить, что ток по нему может протекать только в одном направлении – от анода к катоду (светодиод – вид диода, который работает только в одном направлении), таким образом, анод (длинный пин) всегда подключается к источнику питания или управляющему сигналу, тогда как катод (короткий пин) обычно подключается к земле через сопротив-

ление, ограничивающее ток, протекающий через светодиод. Светодиод нельзя подключать без сопротивления, иначе он может сгореть. Чем больше сопротивление в схеме со светодиодом (от 220 Ом до 10 кОм), тем меньше яркость свечения светодиода (меньший ток проходит через него). Подробнее о светодиодах можно почитать, например, в источнике [12].

И наконец, резисторы. На схемах резисторы обозначаются следующим образом (слева – в англоязычных источниках, справа – в русскоязычных источниках):

Рис. 40 ❖ Обозначение резисторов на принципиальных схемах [11]

У каждого резистора есть номинал: 220 Ом, 1 кОм и т. д. Резисторы, входящие в комплект с Arduino Uno и в другие комплекты с иными платами, обладают цветовыми насечками, см., например, рис. 4–6. Каждый цвет обозначает цифру, от 0 до 9, и цветовых насечек на резисторе несколько: таким образом можно определить номинал резистора, пользуясь правилами, изображёнными на рис. 41 [11]. Подобные резисторы, несмотря на их размер, всё же встречаются в реальных схемах, используемых в промышленности: например, управляющая плата холодильника Whirlpool собрана с помощью таких резисторов, поскольку холодильник большой и делать миниатюрную плату с применением сверхточных технологий, которые используются при производстве материнской платы для настольного компьютера, не имеет смысла.

Обладая знаниями из таблицы, взятой с сайта [11] и изображённой на рис. 41, вы можете подсчитать номиналы и точность резисторов, входящих в комплект с Arduino Uno и показанных на рис. 4, 5 и 6. На рис. 42 даны примеры резисторов и их номиналов.

Резисторы для схем бывают двух типов (здесь мы для простоты не говорим о специфических типах резисторов – потенциометрах, термисторах, варисторах, фоторезисторах и т. д., хотя некоторые из них встретятся нам в этой части книги): включённые последовательно в электрическую цепь (series resistors) по отношению к пинам платы, и – параллельно, которые, в свою очередь, подразделяются на стягивающие и подтягивающие резисторы (pull-down и pull-up resistors). Значения последовательных резисторов обычно варьируются от 100 до 300 Ом, стягивающих и подтягивающих – от 1 кОм до 10 кОм. Последовательные резисторы подключаются в электрическую цепь, чтобы защитить оборудование и, главное, пины платы от больших значений тока: например, так подключается светодиод через резистор номиналом 220 Ом, чтобы светодиод не перегорел. Каждый пин платы обладает ограниченной способностью быть источником (высокое значение, логическая 1) или приёмником (низкое значение, логический 0) электрического тока через электрическую схему, под-

ключённую к нему. Периферия, которая в схеме потребляет большие значения тока – больше, чем может позволить себе пин платы, даже если это происходит очень короткое время, может повредить пин на плате – именно поэтому в схему последовательно включается ограничивающий ток резистор, например как показано на рис. 43 в схеме со светодиодом [19], где OUT – это пины платы, а Vcc – источник питания.

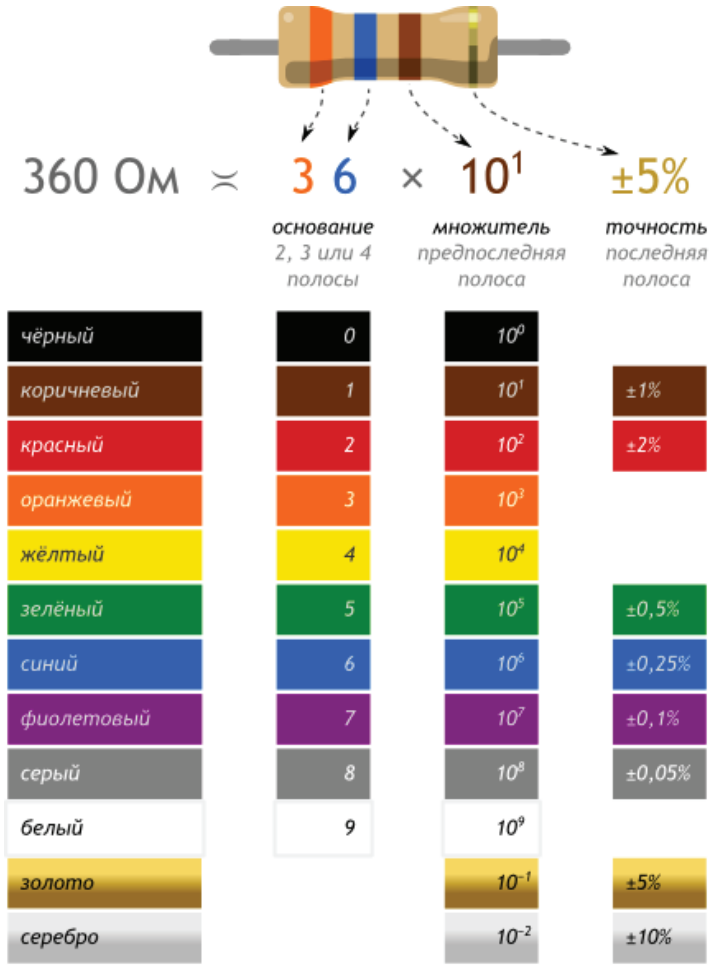


Рис. 41 ❖ Правила подсчёта номинала резистора и его точности [11]

Рис. 42 ❖ Примеры резисторов и их номиналов [11]

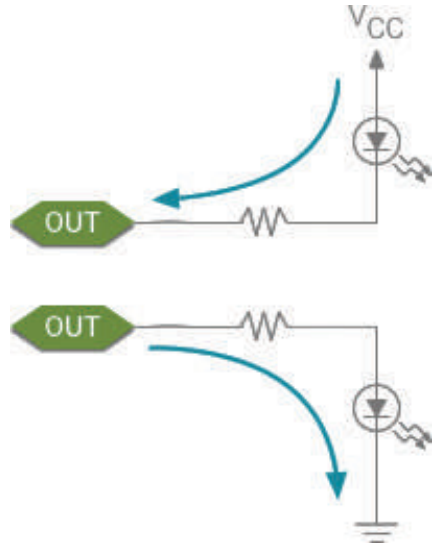


Рис. 43 ❖ Пример подключения последовательных резисторов [19]

Стягивающие и подтягивающие резисторы подключаются в схему параллельно по отношению к пинам и используются для того, чтобы, соответственно, «стягивать» значение напряжения на пине к низкому (обеспечивать стабильный сигнал логического 0) или «подтягивать» значение напряжения на пине к высокому (стабильная логическая 1). Делается это потому, что цифровые входы, не подключённые ни к какой нагрузке, являются «плавающими» (см. рис. 44, левую часть – представьте её без включённых туда резисторов; I/O – пины платы): они подвержены различного рода помехам и искажениям, появляющимся из-за электромагнитных возмущений, которые влияют на значения, читаемые с пинов платы приложениями, и могут способствовать непредсказуемым изменениям этих значений. Стягивающие и подтягивающие резисторы заставляют пины показывать правильные значения 0 или 1, даже если к ним ничего не подключено [19]. В правой части рис. 44 изображён случай с переключателем: если в схеме не будет подтягивающего резистора, значение при открытом переключателе на входном пине IN платы, читающем значения, будет плавающим; если в схеме есть такой резистор, изображённый на рисунке, значение на пине IN будет точно соответствовать логической единице. Для более подробного ознакомления с резисторами рекомендуются источники [11] или [19].

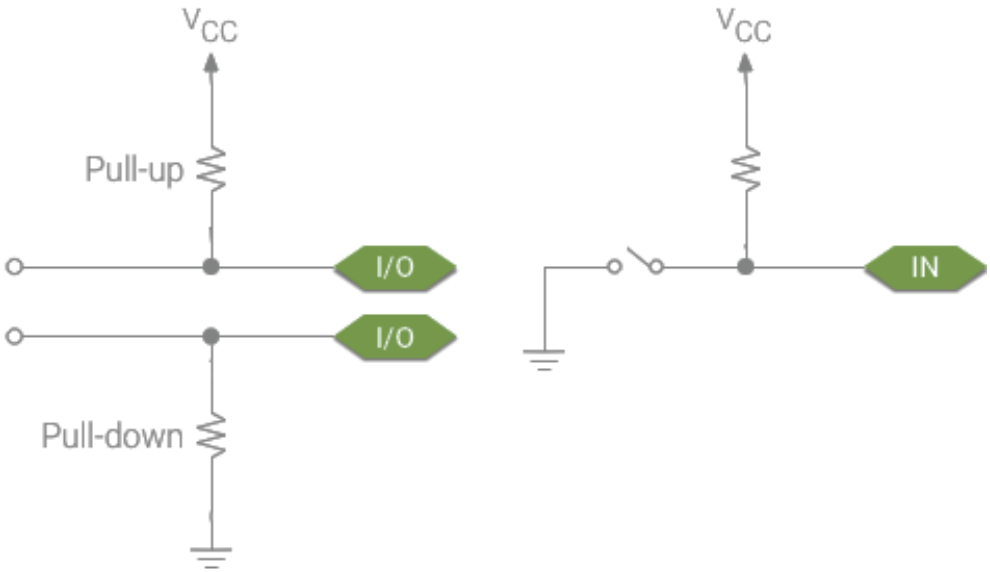


Рис. 44 ❖ Пример подключения стягивающего (pull-down) и подтягивающего (pull-up) резисторов [19]

Ещё один момент, о котором следует упомянуть, – это тот факт, что при подключении различных модулей и сенсоров надо также обращать внимание не только на схему, но и на надписи рядом с выходами (пинами) этих сенсоров. Сенсоры в наборах могут отличаться, хоть эта вероятность и мала, поэтому всегда проверяйте наличие информации/меток рядом с пинами устройств, используемых в практических заданиях, – это первично, а схема вторична. Простой пример – задание 24 из этой части книги, где к конкретным пинам платы подключаются конкретные выходы модуля часов реального времени, обозначения которых можно увидеть на самом модуле – см. рис. 24. Некоторые обозначения: – (минус), GND, G – земля; + (плюс), VCC, VIN, +5V, 3.3V – питание; CLK, SCK – clock (время, частота), DAT, SDA – date/data (дата, данные), RST – reset (сброс настроек), R,G,B – цвета на трёхцветном светодиоде; A0 (аналоговый), D0 (цифровой), SIG, S, VRx,VRy, SW – пин для передачи сигнала (данных).

Теперь, получив базовые знания об основных элементах, использующихся в практических экспериментах, и о технике безопасности, можно приступать к выполнению заданий.

Практическое занятие 1. Hello, world!

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm).

Первое, что нужно сделать, – это выбрать, где вы будете работать: в среде Arduino IDE или с помощью онлайн-системы Arduino Create (Arduino Web Editor) через веб-браузер. В первом случае необходимо скачать Arduino IDE ([2] -> Windows Installer) и установить её, включая установку драйвером для COM- и USB-портов. Во втором случае необходимо зарегистрироваться на сайте ([3] -> sign up).

Далее требуется пройти урок-инструкцию по следующему адресу: [4]. Это нужно для того, чтобы установить драйверы для платы, если они правильно не установились или нет прав администратора на компьютере, а также для того, чтобы правильно настроить среду и выбрать плату Arduino Uno и COM-порт.

После этого можно приступать к занятию. Подсоедините плату Arduino Uno к USB-порту компьютера (если вы ещё этого не сделали). Если драйверы установлены правильно, плата должна определиться, её название появится в панели уведомлений операционной системы. На плате есть встроенный мини-светодиод (miniLED), подключённый к 13-му цифровому порту. В этом занятии мы напишем код, который будет ожидать ввода через консоль буквы R, при её вводе заставлять miniLED 13 загораться на полсекунды, гаснуть на полсекунды и писать в консоль фразу Hello, World! Так как miniLED является встроенным, никаких дополнительных схем создавать не надо. Для того чтобы открыть консоль, надо выбрать в пункте меню **Инструменты** Монитор порта, или нажать комбинацию **Ctrl**+**Shift**+**M** (Arduino IDE), или выбрать пункт меню **Монитор порта** слева (Arduino Web Editor). Когда всё готово, можно скопировать код ниже в среду и загрузить программу на плату с помощью кнопки ->.

В коде используется команда Serial.begin(9600), означающая, что скорость/частота обмена данными платы с компьютером по UBS-соединению составляет 9600 bps (bits per second, бит в секунду). В консоли можно увидеть, что есть и другие частоты, но для выполнения задания в консоли должна быть выставлена такая же частота (справа внизу).

Схема представлена на рис. 45.

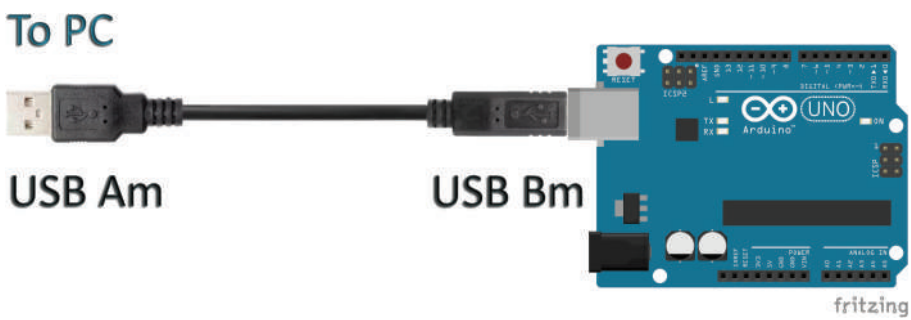


Рис. 45 ❖ Схема подключения для практического занятия 1

Код программы

```
int val ;// define a variable val
int ledpin = 13 ;// define the digital interface 13
void setup ()
{
    Serial.begin (9600) ;// set the baud rate to 9600, where the software settings keep
consistent.
    pinMode (ledpin, OUTPUT) ;// set the digital output interface 13 is, Arduino, we use
the I / O port should be carried out like this definition.
}
void loop ()
{
    val = Serial.read () ;// read the PC sends a command to the Arduino or characters, and
the
    instruction or character assigned val
    if (val == 'R') // determine the received command or character is «R».
    {// If you receive a «R» character
        digitalWrite (ledpin, HIGH) ;// lit Digital 13 LED.
        delay (500);
        digitalWrite (ledpin, LOW) ;// Off Digital 13 LED
        delay (500);
        Serial.println ("Hello World!") ;// Displays «Hello World!» String
    }
}
```

Практическое занятие 2. Эксперимент с мигающим светодиодом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm).

В этом занятии мы всё ещё работаем со встроенным мини-светодиодом на плате (miniLED), подключённым к 13-му цифровому порту. Напишем код, который будет заставлять miniLED 13 загораться на секунду и гаснуть на секунду. Так как miniLED является встроенным, никаких дополнительных схем создавать не надо.

Схема представлена на рис. 46.

Код программы

```
int ledpin = 13 ;// define the digital interface 13
void setup ()
{
        pinMode (ledpin, OUTPUT) ;// set the digital output interface 13 is, Arduino, we
use the I / O port should be carried out like this definition.
}
void loop ()
{
        digitalWrite (ledpin, HIGH) ;// lit Digital 13 LED.
```

```
        delay (1000);
        digitalWrite (ledpin, LOW) ;// Off Digital 13 LED
        delay (1000);
}
```

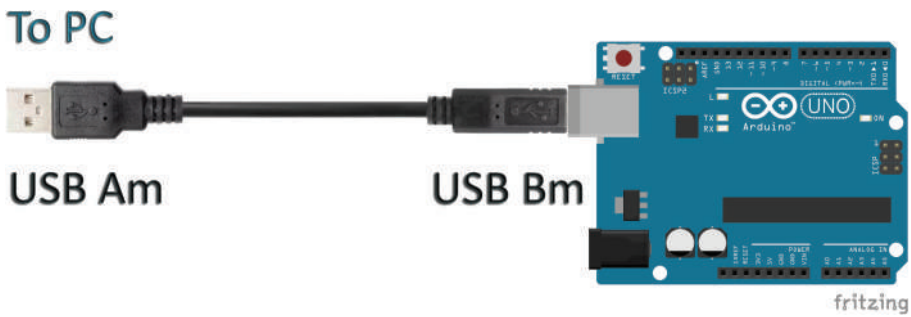


Рис. 46 ❖ Схема подключения для практического занятия 2

Практическое занятие 3. Эксперимент с контролируемой потенциометром яркостью свечения светодиода через порт PWM

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- потенциометр;
- светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

В этом занятии мы познакомимся с потенциометром и портом PWM. PWM (Pulse Width Modulation) – широтно-импульсная модуляция, или процесс управления мощностью, подводимой к нагрузке, путём изменения отношения периода импульса к длительности импульса при неизменной частоте. На плате Arduino Uno можно подавать значения от 0 до 255 на PWM-пин, что заставит плату выдавать PWM-сигнал в определённые моменты времени, соответствующие поданному входному значению. Другими словами, в терминах напряжения, подавая разные значения от 0 до 255 на PWM-пин, мы заставляем плату менять напряжение от 0 до 5 В. В коде используется функция analogWrite(pin, value), с помощью которой можно менять напряжение, подаваемое на PWM-пин, задавая value от 0 до 255. Эта функция используется для регулировки скорости вращения мотора, яркости светодиода и т. д.

Arduino Uno располагает несколькими PWM-пинами, обозначенными символом ~ рядом с номером пина: 3, 5, 6, 9, 10, 11 (см. рис. 1). С помощью потенци-

ометра, подключённого к одному из аналоговых портов платы (A0–A6), и светодиода, катод (короткий пин) которого подключён через резистор на 220 ОМ к земле (всегда! к минусу) и анод (длинный пин) которого подключён к одному из цифровых пинов (0–13) платы (всегда! к плюсу), мы будем регулировать значение переменной val (крутя ручку потенциометра) и, соответственно, напряжение, которое выдаёт плата из этого цифрового пина.

Схема представлена на рис. 47–48.

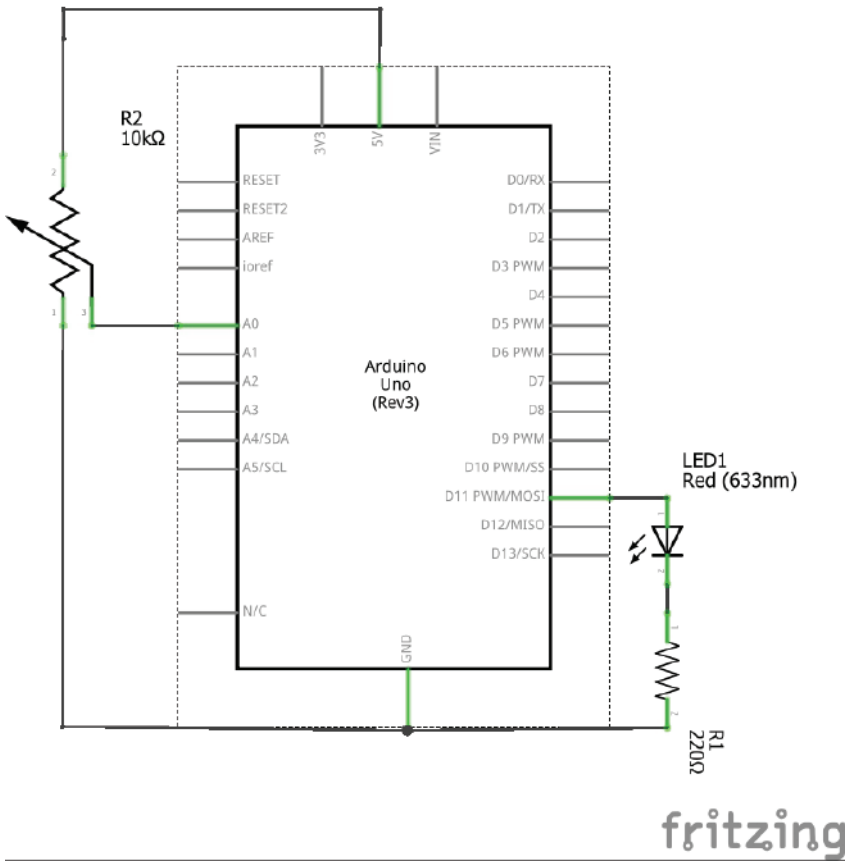


Рис. 47 ❖ Принципиальная схема подключения для практического занятия 3

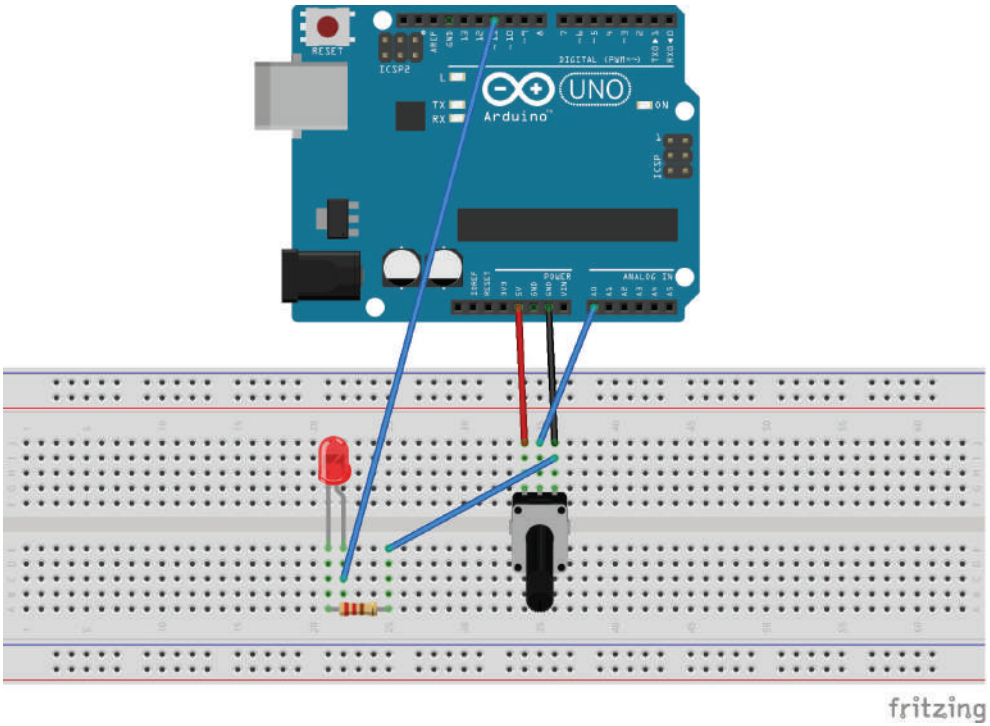


Рис. 48 ❖ Схема подключения с макетной платой для практического занятия 3

Код программы

```
    int potpin = 0 ;// define analog interface 0
    int ledpin = 11 ;// define the digital interface 11 (PWM output)
    int val = 0 ;// temporary values of the variables from the sensor
    void setup ()
    {
        pinMode (ledpin, OUTPUT) ;// define the digital interface 11 as output
        Serial.begin (9600) ;// set the baud rate to 9600
// NOTE: analog interface is automatically set to the input
    }
    void loop ()
    {
        val = analogRead (potpin) ;// read sensor analog values and assigned to val
        Serial.println (val) ;// display val variable
        analogWrite (ledpin, val / 4) ;// turn on the LED and set the brightness (PWM
output max 255)
        delay (10) ;// delay of 0.01 seconds
    }
```

Практическое занятие 4. Эксперимент с внешним мигающим светодиодом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

В этом занятии мы подсоединим внешний светодиод к цифровому порту 10 платы. Напишем код, который будет заставлять светодиод загораться на секунду и гаснуть на секунду. В этот раз, по сравнению с занятием 2, светодиод у нас внешний, поэтому надо собрать схему, показанную ниже. В схеме с внешним светодиодом всегда должен быть резистор с малым сопротивлением (220 Ом), иначе светодиод может перегореть.

Схема представлена на рис. 49–50.

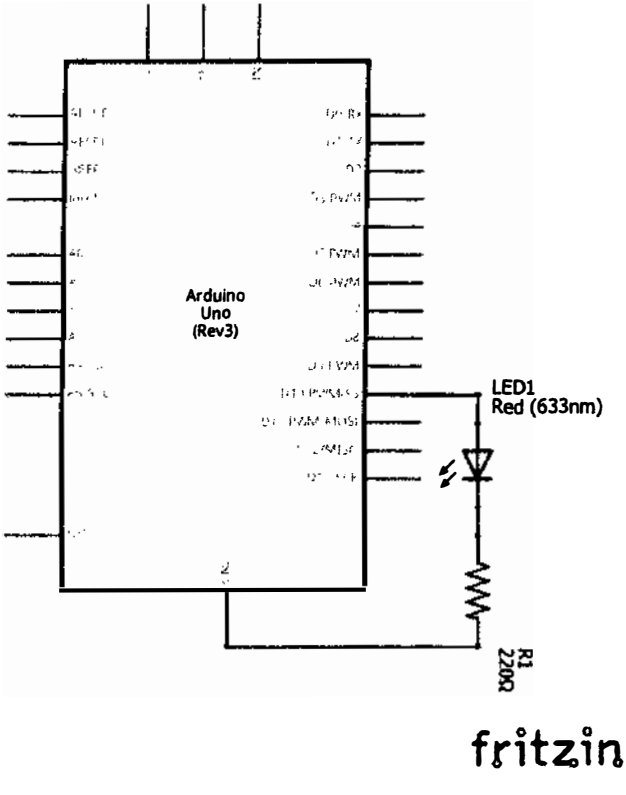


Рис. 49 ❖ Принципиальная схема подключения для практического занятия 4

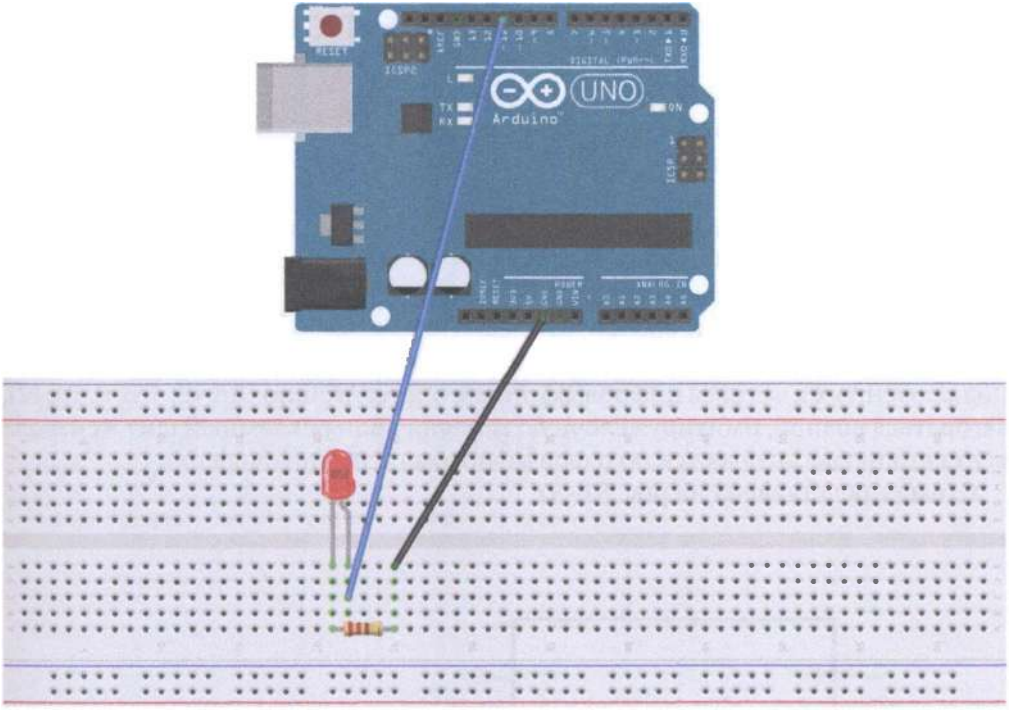


Рис. 50 ❖ Схема подключения с макетной платой для практического занятия 4

Код программы

```
int ledPin = 10; // define the interface number 10
void setup ()
{
  pinMode (ledPin, OUTPUT) ;// define a small lamp interface output interface
}
void loop ()
{
  digitalWrite (ledPin, HIGH); // lit a small lamp
  delay (1000); // Delay 1 second
  digitalWrite (ledPin, LOW); // extinguish small lights
  delay (1000); // Delay 1 second
}
```

Практическое занятие 5. Эксперимент с рекламной расцветкой

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- светодиоды – 6 шт., по 2 каждого цвета;
- резистор на 220 Ом – 6 шт.;
- макетная плата;
- соединительные провода.

В этом занятии мы напишем код, который будет заставлять 6 светодиодов, подключённых к чётным цифровым пинам платы Arduino Uno (2, 4, 6, 8, 10, 12), загораться волной, имитируя ёлочную гирлянду или рекламный щит – сначала все светодиоды загораются по очереди, потом гаснут по очереди.

Схема представлена на рис. 51–52.

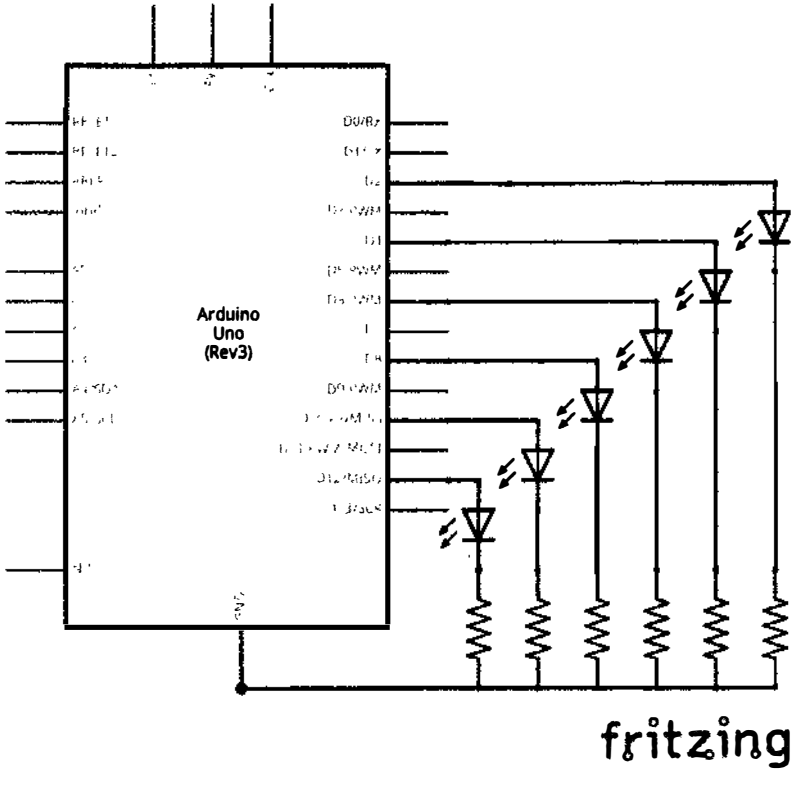


Рис. 51 ❖ Принципиальная схема подключения для практического занятия 5

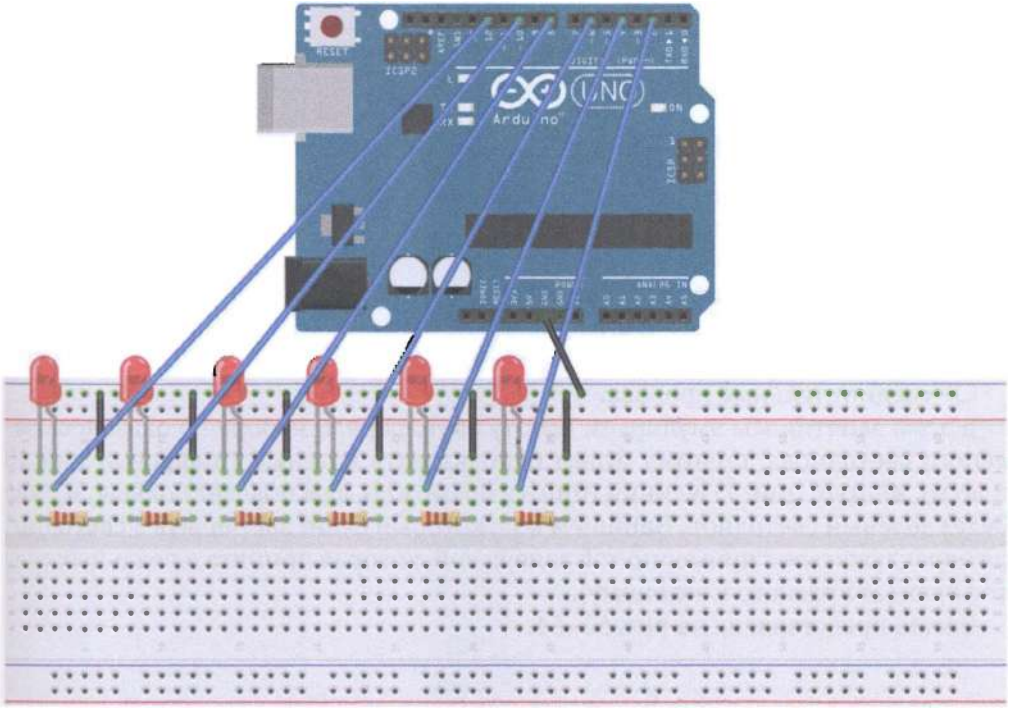


Рис. 52 ❖ Схема подключения с макетной платой для практического занятия 5

Код программы

```
int BASE = 2; // the first one LED connected to the I / O pins
int NUM = 6; // LED's total
void setup ()
{
for (int i = BASE; i <=BASE*NUM; i+=2)
{
    pinMode (i, OUTPUT); // set the digital I / O pin as an output
}
}
void loop ()
{
  for (int i = BASE; i <=BASE*NUM; i+=2)
  {
    digitalWrite (i, LOW); // set the digital I / O pin output is «low», that gradually
turn off the
    lights
    delay (200); // delay
  }
  for (int i = BASE; i <=BASE*NUM; i+=2)
  {
    digitalWrite (i, HIGH); // set the digital I / O pin output is «low», that gradually
lights
```

```
    delay (200); // delay
  }
}
```

Практическое занятие 6. Светофорный эксперимент

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- светодиоды – 3 шт., разных цветов;
- резистор на 220 Ом – 3 шт.;
- макетная плата;
- соединительные провода.

В этом занятии мы эмулируем работу светофора. В наборе не оказалось зелёного светодиода, поэтому будем использовать красный (цифровой пин 10 платы), жёлтый (пин 7) и синий (пин 4). Возможно, вам повезёт больше, и у вас будет зелёный светодиод. Напишем код, который будет заставлять 3 светодиода переключаться с примерно той задержкой, которая принята в реальных светофорах (от красного к зелёному).

Схема представлена на рис. 53–54.

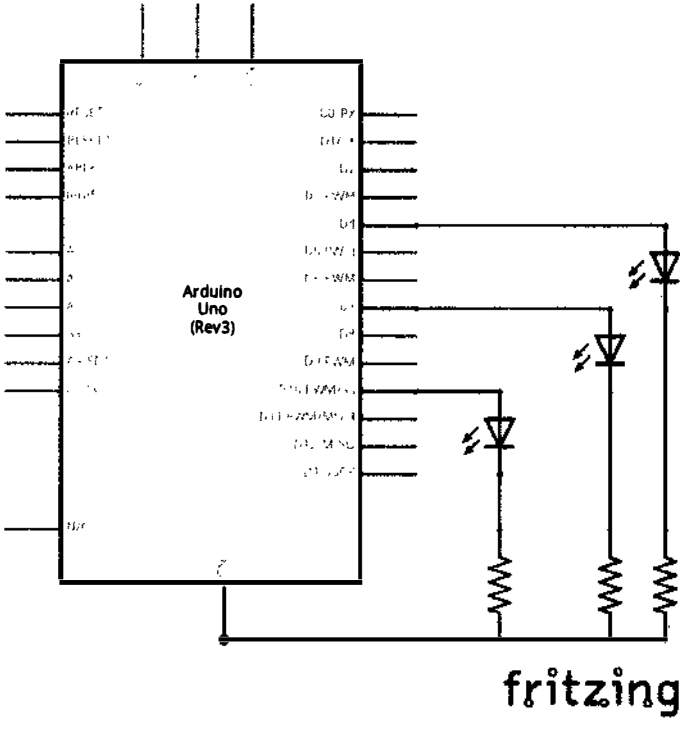


Рис. 53 ❖ Принципиальная схема подключения для практического занятия 6

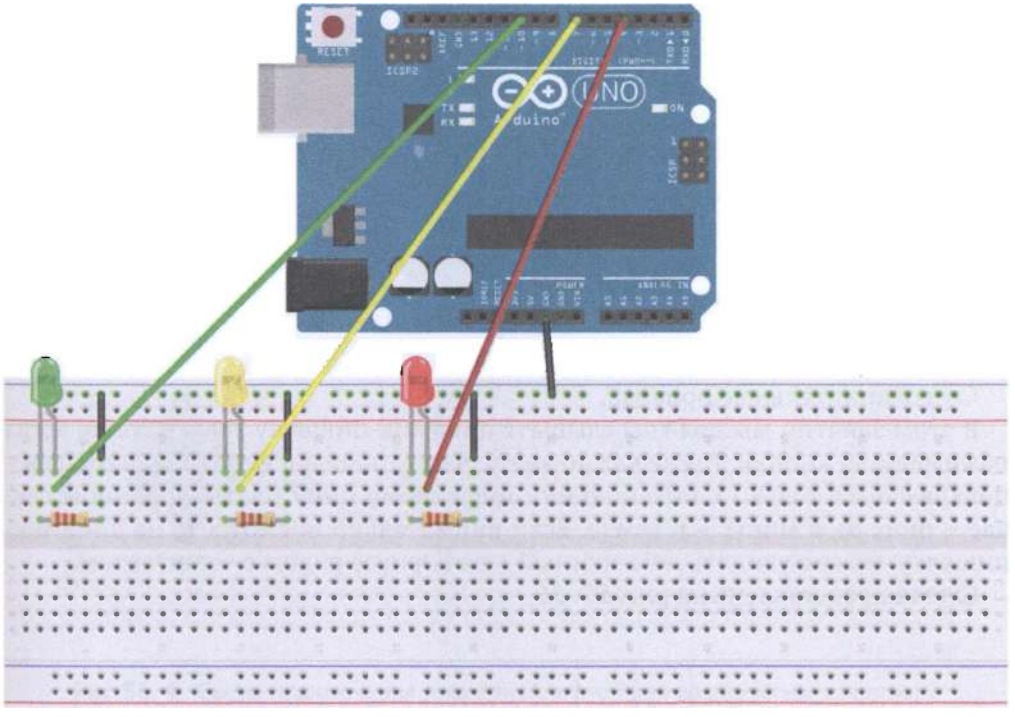


Рис. 54 ❖ Схема подключения с макетной платой для практического занятия 6

Код программы

```
int redled = 10; // define the interface number 10
        int yellowled = 7; // define the number 7 Interface
        int greenled = 4; // define the number 4 Interface
        void setup ()
        {
            pinMode (redled, OUTPUT) ;// define a small red light interface output
interface
            pinMode (yellowled, OUTPUT); // define the yellow light interface output
interface
            pinMode (greenled, OUTPUT); // define the small green light interface output
interface
}
        void loop ()
        {
            digitalWrite (redled, HIGH) ;// lit red lights
            delay (1000) ;// delay of 1 second
            digitalWrite (redled, LOW); // off red light
            digitalWrite (yellowled, HIGH) ;// light yellow light
            delay (200) ;// delay of 0.2 seconds
            digitalWrite (yellowled, LOW) ;// off yellow light
            digitalWrite (greenled, HIGH) ;// flashes the green LED
```

```
    delay (1000) ;// delay of 1 second
    digitalWrite (greenled, LOW) ;// green LED off
}
```

Практическое занятие 7. Эксперимент с пищалкой

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- динамик-пищалка;
- кнопка;
- макетная плата;
- соединительные провода.

В этом занятии мы должны слышать динамик-пищалку только тогда, когда нажимаем на кнопку. Также важно знать, что у пищалки есть + и – выходы и + подключается только к питанию или цифровому пину платы (в данном случае – пину 8), а минус – к земле. Если внимательно посмотреть на динамик-пищалку, можно увидеть помеченный + на корпусе рядом с + контактом.

Схема представлена на рис. 55–56.

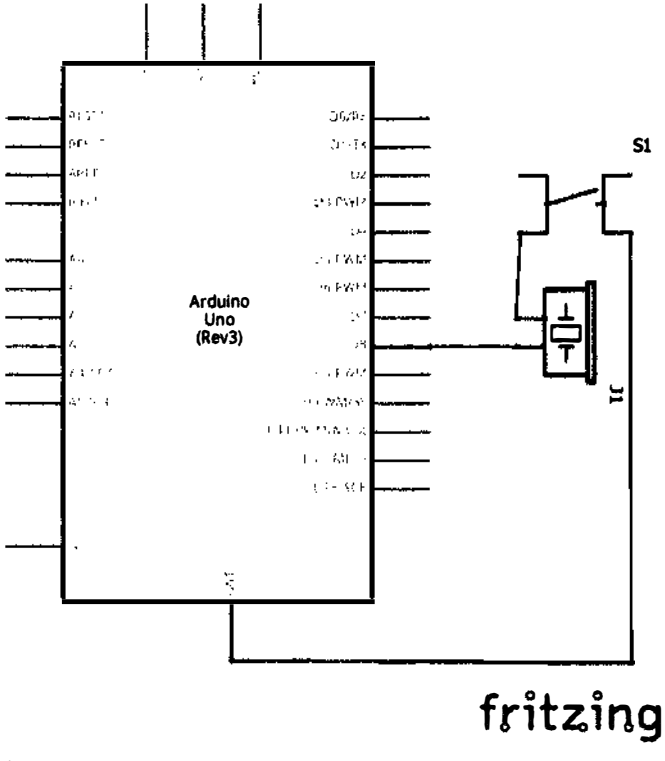


Рис. 55 ❖ Принципиальная схема подключения для практического занятия 7

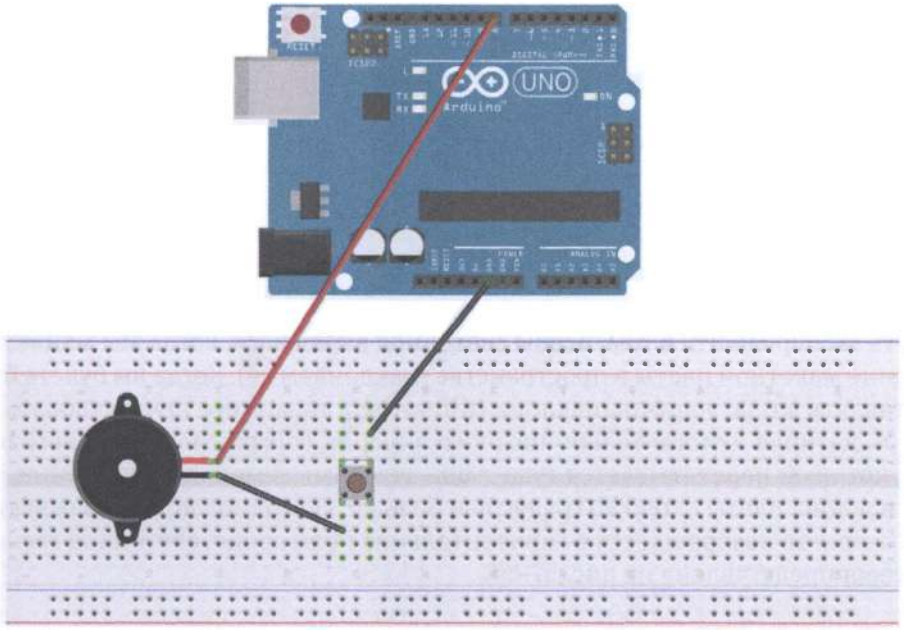


Рис. 56 ❖ Схема подключения с макетной платой для практического занятия 7

Код программы

```
int buzzer = 8 ;// setting controls the digital IO foot buzzer
void setup ()
{
    pinMode (buzzer, OUTPUT) ;// set the digital IO pin mode, OUTPUT out of Wen
}
void loop ()
{
    unsigned char i, j ;// define variables
    while (1)
    {
    for (i = 0; i <80; i++) // Wen a frequency sound
    {
        digitalWrite (buzzer, HIGH) ;// send voice
        delay (1) ;// Delay 1ms
        digitalWrite (buzzer, LOW) ;// do not send voice
        delay (1) ;// delay ms
    }
    for (i = 0; i <100; i++) // Wen Qie out another frequency sound
    {
        digitalWrite (buzzer, HIGH) ;// send voice
        delay (2) ;// delay 2ms
        digitalWrite (buzzer, LOW) ;// do not send voice
        delay (2) ;// delay 2ms
    }
    }
}
```

Практическое занятие 8. Эксперимент с датчиком наклона

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- датчик наклона (tilt switch);
- светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

Суть эксперимента в том, чтобы светодиод включался, когда мы меняем положение макетной платы в пространстве (наклоняем её). Когда вы будете брать и вставлять в плату датчик наклона (tilt switch), то можете почувствовать, что внутри датчика действительно находится шарик (tilt mechanism – mechanical ball), который перекатывается туда-сюда, тем самым меняя напряжение. Если угол наклона платы около 90 градусов и больше, светодиод должен зажигаться, если же плата возвращается в горизонтальное положение, светодиод гаснет.

Схема представлена на рис. 57–58.

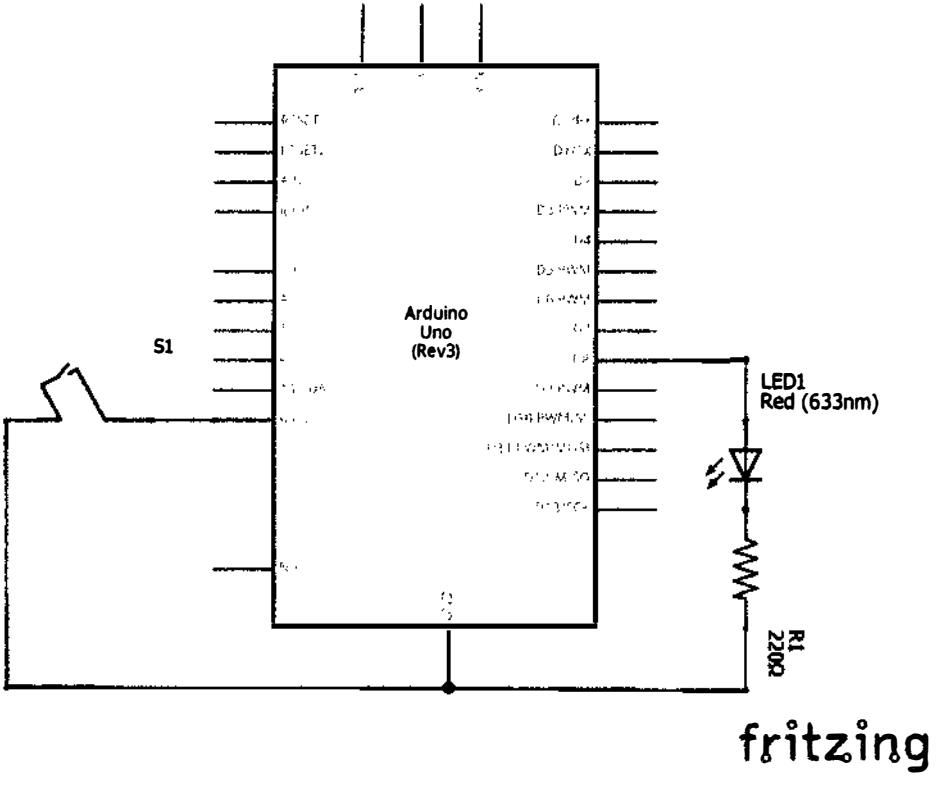


Рис. 57 ❖ Принципиальная схема подключения для практического занятия 8

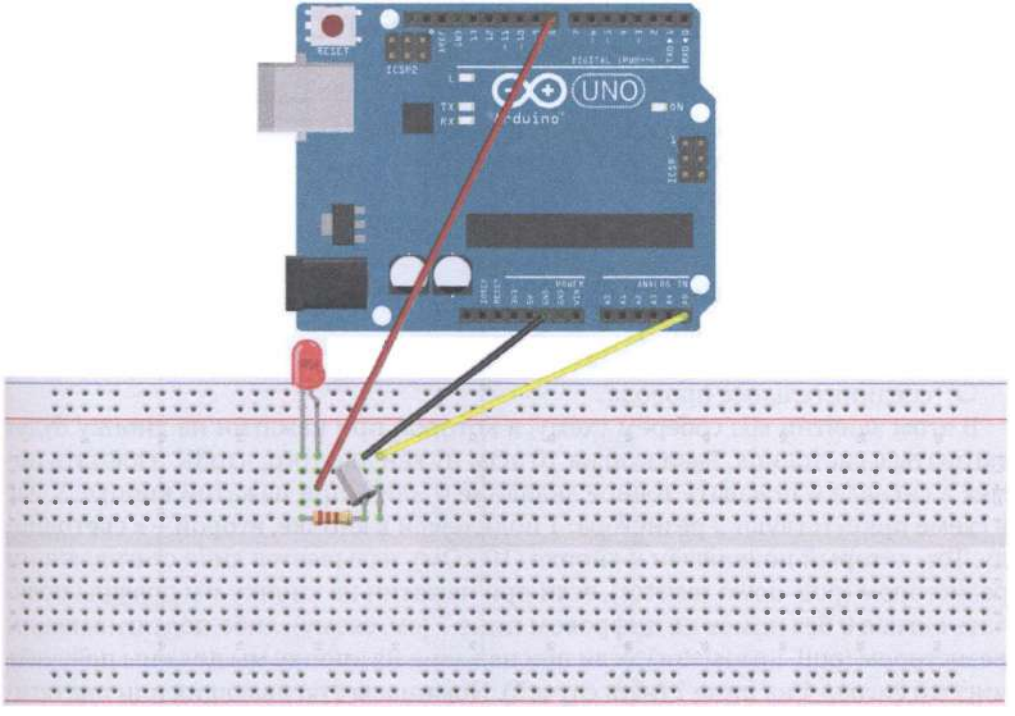


Рис. 58 ❖ Схема подключения с макетной платой для практического занятия 8

Код программы

```
void setup ()
{
    pinMode (8, OUTPUT) ;// set the digital 8 pin to out mode
}
void loop ()
{
    int i ;// define the amount of i
    while (1)
    {
    i = analogRead (5) ;// read the analog voltage value 5
    if (i> 200) // if greater than 512 (2.5V)
    {
        digitalWrite (8, HIGH) ;// led light is lit
    }
    else // Otherwise
    {
        digitalWrite (8, LOW) ;// off led light
 }
 }
}
```

Практическое занятие 9. Эксперимент с чистым входным сигналом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- кнопка;
- светодиод;
- резистор на 220 Ом;
- резистор на 10 кОм;
- макетная плата;
- соединительные провода.

В этом занятии мы соберём схему, в которой при нажатии на кнопку будет загораться светодиод. Резистор на 10 кОм нужен для того, чтобы избежать помех в сигнале и подавать точное значение 0 или 1 при нажатой кнопке. У нас в схеме присутствуют 2 резистора: 1 подключён к кнопке, второй – к светодиоду. Тот, который подключён к кнопке (10 кОм), называется либо стягивающим резистором (pull-down resistor), если при нажатии на кнопку мы должны получать сигнал 0 без помех на цифровом пине 7 (см. схему), либо подтягивающим резистором (pull-up resistor), если при нажатии на кнопку мы должны получить чистый сигнал 1 на пине 7 (наш случай). Номиналы стягивающих или подтягивающих резисторов обычно варьируются от 1 кОм до 10 кОм. Тот же резистор, который подключён к катоду светодиода, называется серийным, или последовательным (series resistor), и нужен для того, чтобы брать на себя часть тока во избежание перегорания светодиода. Значения таких резисторов обычно варьируются от 100 Ом до 300 Ом. Почему нам важен чистый сигнал в этом занятии? Потому что мы читаем его с пина 7 с помощью функции digitalRead(7).

Схема представлена на рис. 59–60.

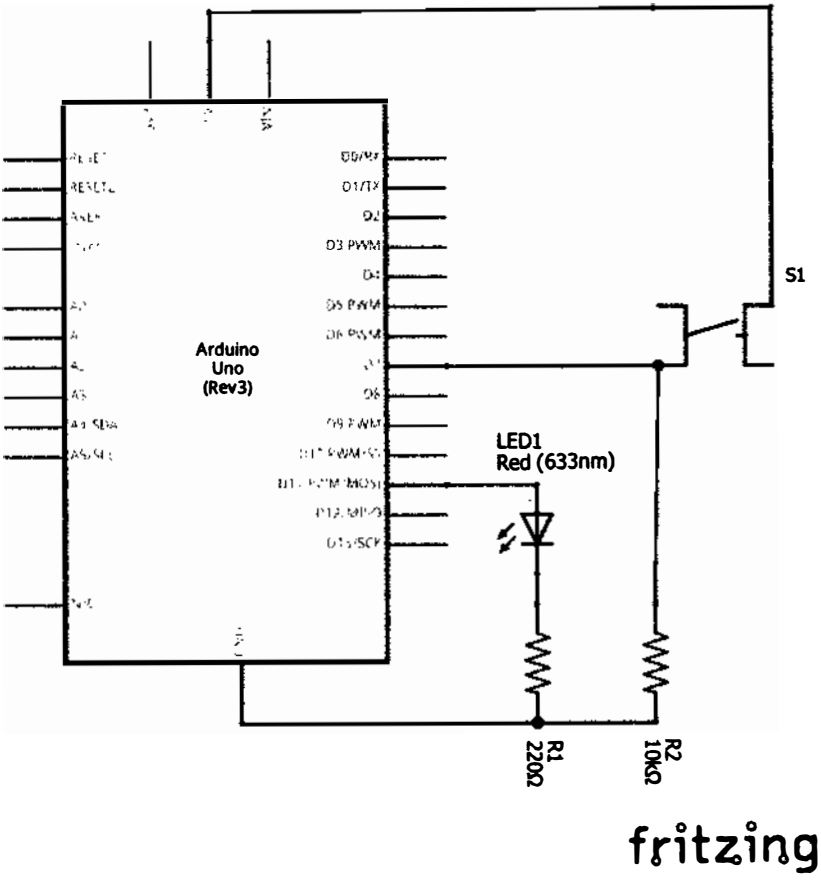


Рис. 59 ❖ Принципиальная схема подключения для практического занятия 9

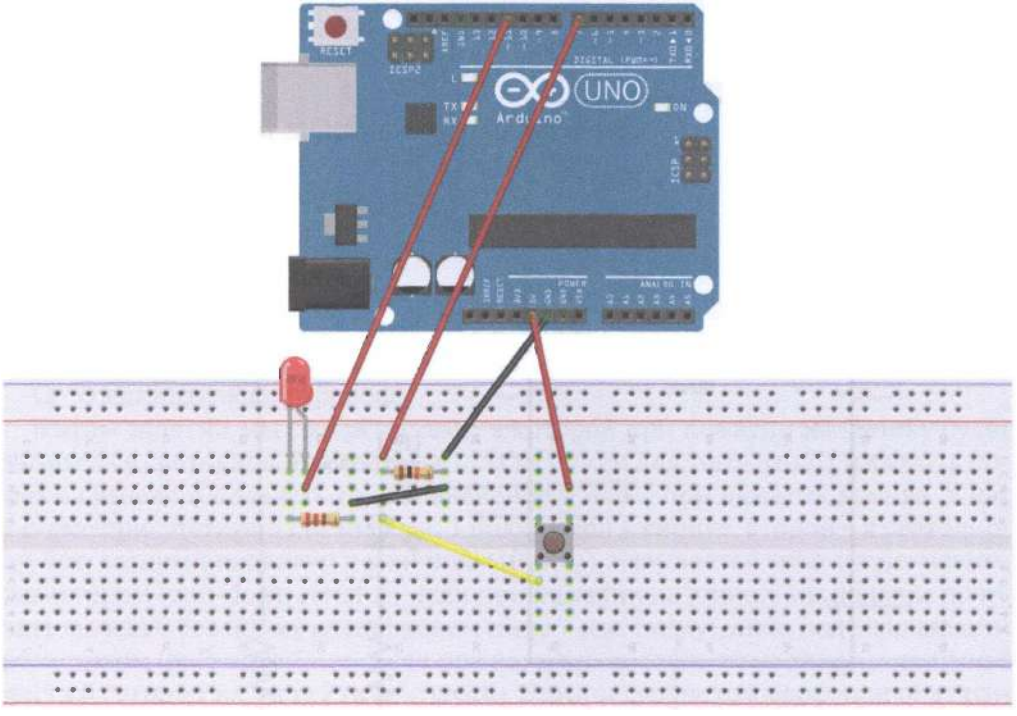


Рис. 60 ❖ Схема подключения с макетной платой для практического занятия 9

Код программы

```
int ledpin = 11 ;// define the interface number 11
int inpin = 7 ;// define the number 7 Interface
int val ;// define a variable val
void setup ()
{
    pinMode (ledpin, OUTPUT) ;// define a small lamp interface output interface
    pinMode (inpin, INPUT) ;// define the key interface for the input interface
}
void loop ()
{
    val = digitalRead (inpin) ;// read digital 7-level value assigned to val
    if (val == LOW) // test button is pressed, the button lights up when pressed small
    {digitalWrite (ledpin, LOW);}
    else
    {digitalWrite (ledpin, HIGH);}
}
```

Практическое занятие 10. Расширенный эксперимент с чистым сигналом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- кнопка – 3 шт.;
- светодиоды разных цветов – 3 шт.;
- резистор на 220 Ом – 3 шт.;
- резистор на 10 кОм – 3 шт.;
- макетная плата;
- соединительные провода.

В этом занятии мы соберём схему, в которой при нажатии на каждую из кнопок будет загораться соответствующий светодиод. В сложных схемах, подобных этой, лучше всего выводить питание и землю на отдельные рельсы (+ и –, или красный и синий горизонтальный рельс) макетной платы.

Схема представлена на рис. 61–62.

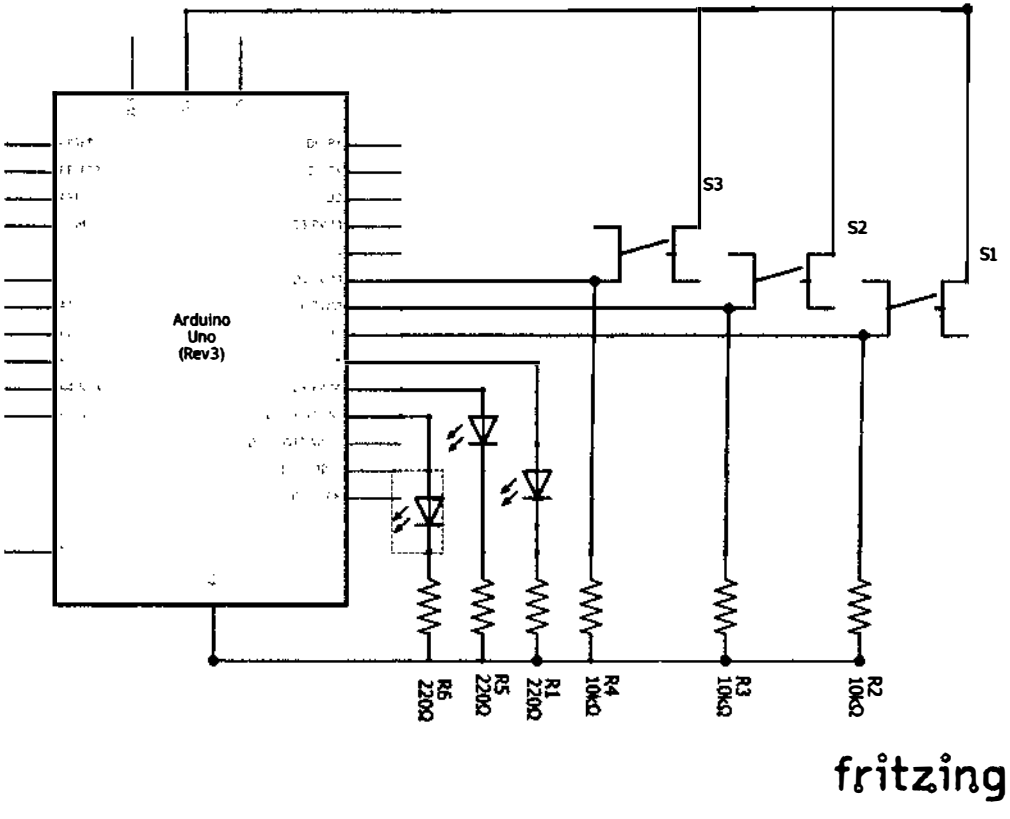


Рис. 61 ❖ Принципиальная схема подключения для практического занятия 10

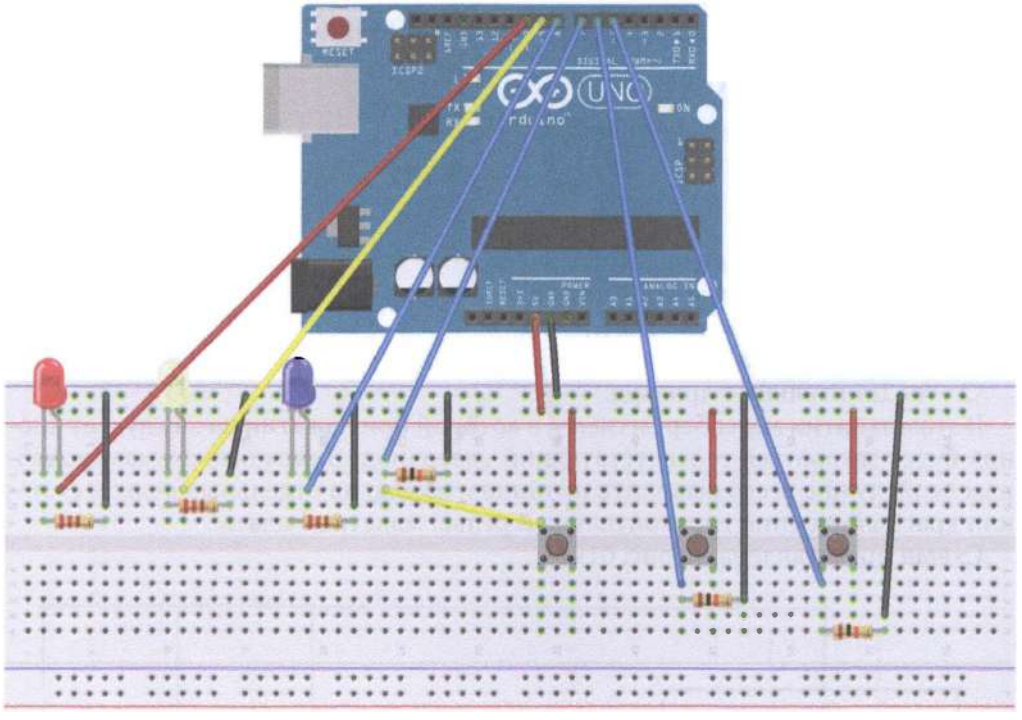


Рис. 62 ❖ Схема подключения с макетной платой для практического занятия 10

Код программы

```
int redled = 10;
int yellowled = 9;
int greenled = 8;
int redpin = 7;
int yellowpin = 6;
int greenpin = 5;
int red;
int yellow;
int green;
void setup ()
{
    pinMode (redled, OUTPUT);
    pinMode (yellowled, OUTPUT);
    pinMode (greenled, OUTPUT);
    pinMode (redpin, INPUT);
```

```
    pinMode (yellowpin, INPUT);
    pinMode (greenpin, INPUT);
}
void loop ()
{
    red = digitalRead (redpin);
    if (red == LOW)
    {digitalWrite (redled, LOW);}
    else
    {digitalWrite (redled, HIGH);}
    yellow = digitalRead (yellowpin);
    if (yellow == LOW)
    {digitalWrite (yellowled, LOW);}
    else
    {digitalWrite (yellowled, HIGH);}
    green = digitalRead (greenpin);
    if (green == LOW)
    {digitalWrite (greenled, LOW);}
    else
    {digitalWrite (greenled, HIGH);}
}
```

Практическое занятие 11. Эксперимент по чтению аналогового значения

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- потенциометр;
- макетная плата;
- соединительные провода.

У Arduino Uno есть аналоговые входы A0–A5. Воспользуемся входом A0 и функцией analogRead(), чтобы считывать данные о текущем сопротивлении потенциометра (значения от 0 до 1023, т. к. Arduino Uno располагает 10-битными аналоговыми или цифровыми портами, 2^10 = 1024 значения) и выводить их на экран. В коде используется команда Serial.begin(9600), означающая, что скорость/частота обмена данными платы с компьютером по UBS-соединению составляет 9600 bps (bits per second, битов в секунду). В консоли должна быть выставлена такая же частота (справа внизу). Не забудьте открыть консоль, чтобы видеть сообщения (**Ctrl**+**Shift**+**M**)!

Схема представлена на рис. 63–64.

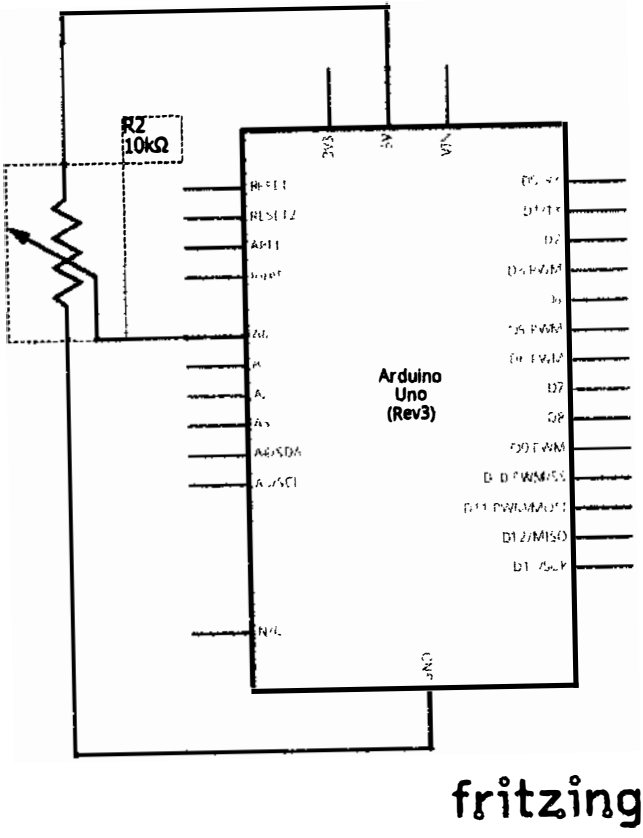


Рис. 63 ❖ Принципиальная схема подключения для практического занятия 11

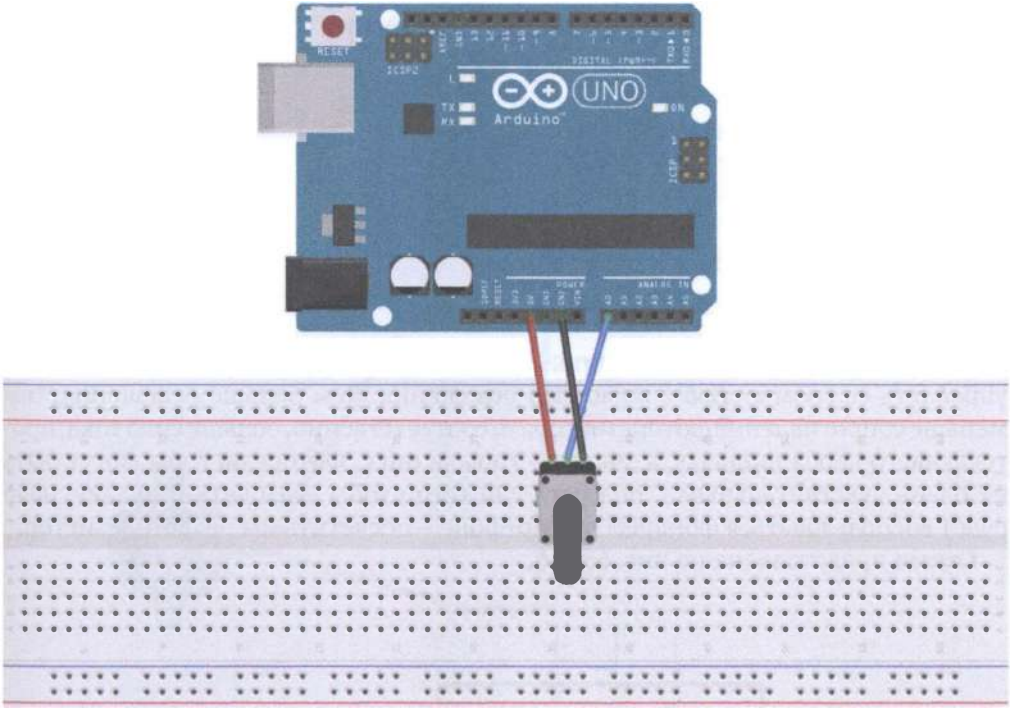


Рис. 64 ❖ Схема подключения с макетной платой для практического занятия 11

Код программы

```
int potpin = 0 ;// define analog interface 0
int ledpin = 13 ;// define the digital interface 13
int val = 0 ;// will define the variable val, and the initial value 0
void setup ()
{
    pinMode (ledpin, OUTPUT) ;// output interface defines the digital interface
    Serial.begin (9600) ;// set the baud rate to 9600
}
void loop ()
{
    digitalWrite (ledpin, HIGH) ;// digital interface 13 of the LED lights
    delay (50) ;// delay of 0.05 seconds
    digitalWrite (ledpin, LOW) ;// off LED digital interface 13
    delay (50) ;// delay of 0.05 seconds
    val = analogRead (potpin) ;// read the value of analog interface 0, and assign val
    Serial.println (val) ; // shows the value of val
}
```

Практическое занятие 12. Эксперимент по управлению звуком и светом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- фоторезистор;
- динамик-пищалка;
- макетная плата;
- соединительные провода.

В данном занятии мы подсоединяем фоторезистор к пищалке, для того чтобы управлять её громкостью с помощью освещения. Чем больше освещения, тем меньше сопротивление фоторезистора и, соответственно, больше сила тока, протекающего через пищалку, а значит – громче писк. Фоторезисторы могут быть не очень чувствительными, поэтому, для того чтобы услышать пищалку, надо будет воспользоваться фонарём в смартфоне – посветить им на фоторезистор.

Схема представлена на рис. 65–66.

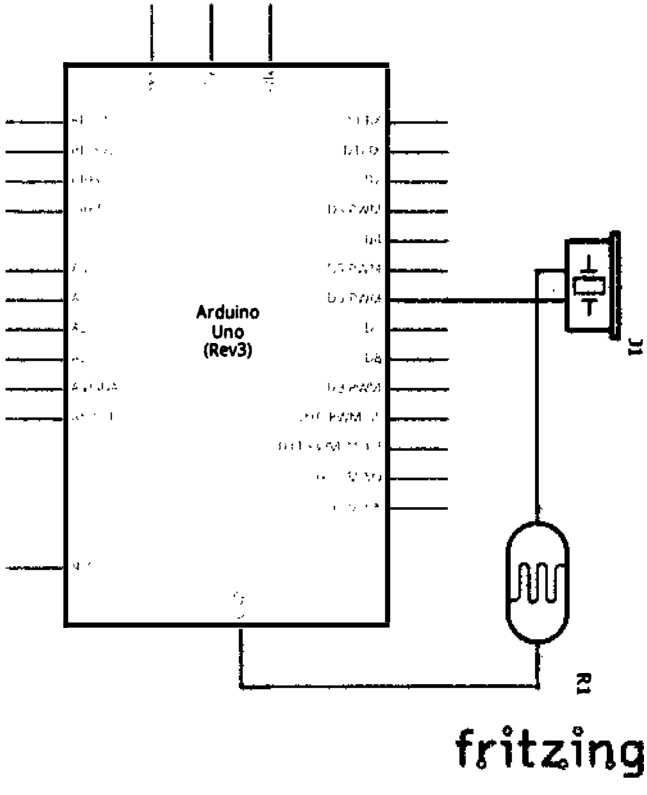


Рис. 65 ❖ Принципиальная схема подключения для практического занятия 12

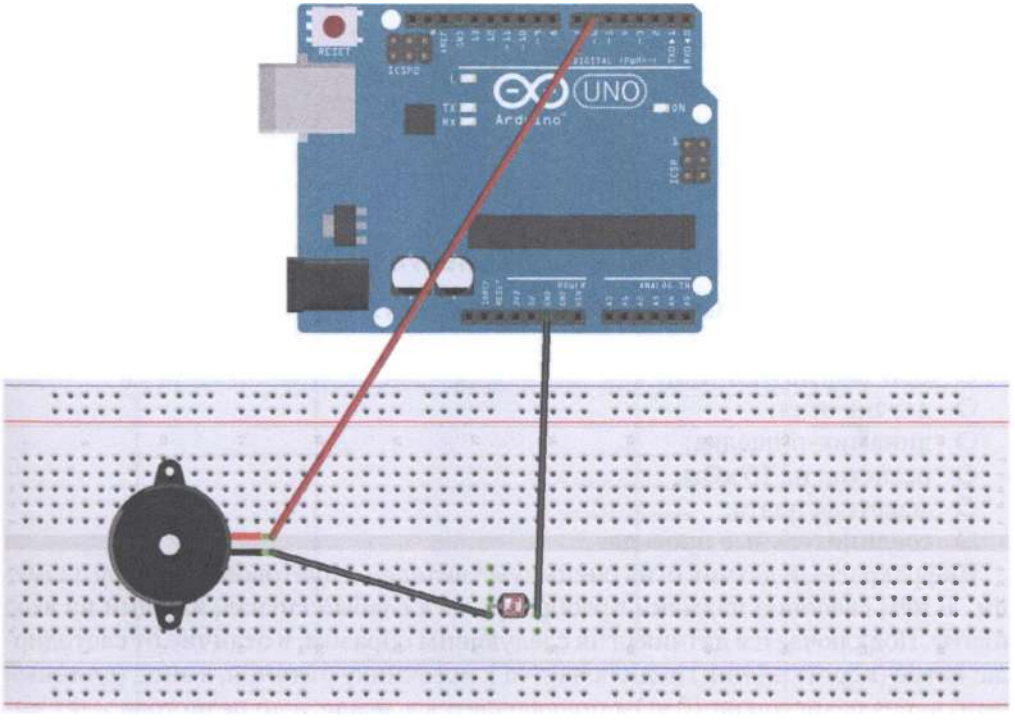


Рис. 66 ❖ Схема подключения с макетной платой для практического занятия 12

Код программы

```
void setup ()
{
    pinMode (6, OUTPUT);
}
void loop ()
{
    while (1)
    {
        char i, j;
        while (1)
        {
            for (i = 0; i <80; i++) // Wen Qie one frequency sound
            {
                digitalWrite (6, HIGH);
                delay (1);
                digitalWrite (6, LOW);
                delay (1);
            }
            for (i = 0; i <100; i++) // Wen Qie out another frequency sound
            {
                digitalWrite (6, HIGH);
```

```
                delay (2);
                digitalWrite (6, LOW);
                delay (2);
            }
        }
    }
}
```

Практическое занятие 13. Эксперимент с датчиком огня

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- датчик огня;
- динамик-пищалка;
- резистор на 10 кОм;
- макетная плата;
- соединительные провода.

Инфракрасный датчик огня очень чувствителен к цветовому спектру пламени. В нём свечение пламени преобразуется в слабый сигнал, идущий на вход платы. Подключается датчик огня следующим образом, в отличие от светодиода: катод (короткий пин) подключается к источнику питания, а анод (длинный пин) через резистор на 10 кОм подключается к земле, а до резистора идёт выход на аналоговый пин платы Arduino Uno. Датчик огня работает и без огня, выдавая на аналоговый пин платы около 0.3 В. Когда пламя находится слишком близко, датчик выдаёт уже 1.0 В и больше. Поэтому в коде было бы целесообразно использовать пороговое значение 0.6 В. Однако практика показала, что на самом деле датчик очень чувствительный, и даже в комнате с верхним освещением из трёх обыкновенных ламп накаливания выдаёт значение 984–985, что соответствует 4.8 В, хотя никакого огня в комнате нет. Поэтому в коде пороговое значение выставлено в 1000. В этом задании нам также понадобится консоль (**Ctrl+Shift+M**) и зажигалка/спички (зажигалка, конечно, получше спичек).

Схема представлена на рис. 67–68.

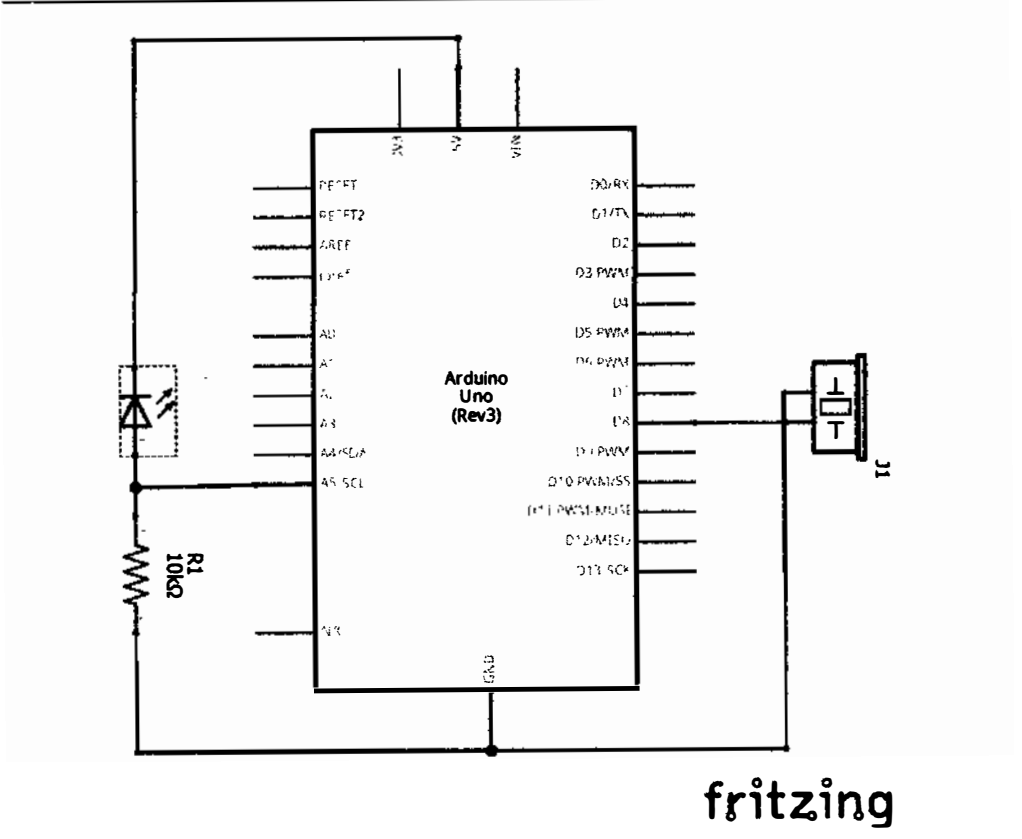


Рис. 67 ❖ Принципиальная схема подключения для практического занятия 13

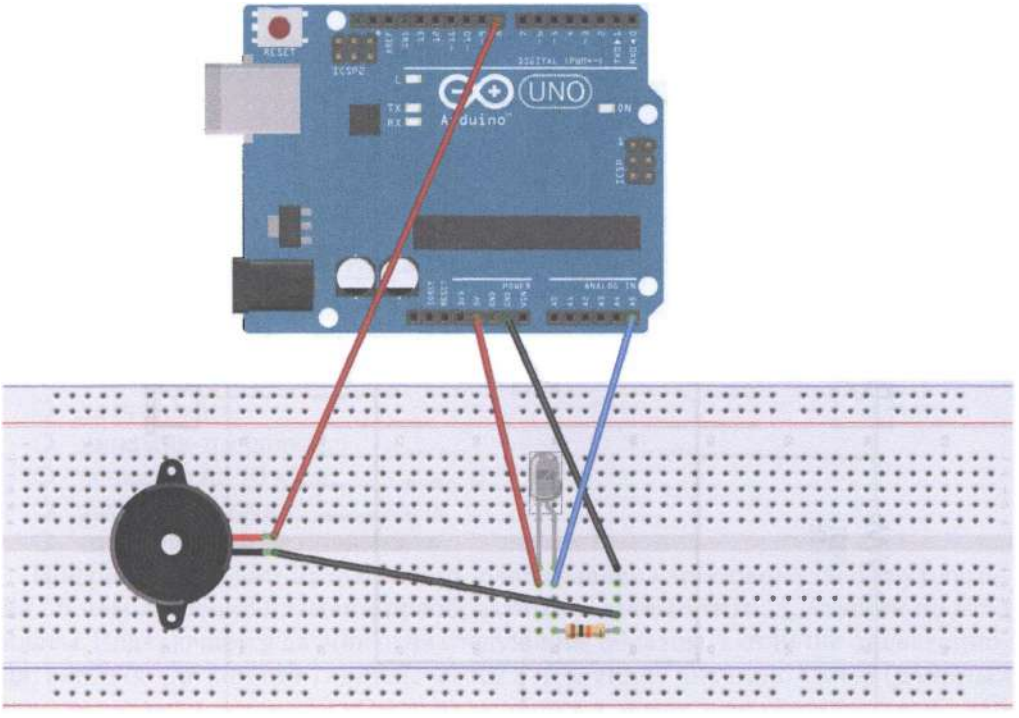


Рис. 68 ❖ Схема подключения с макетной платой для практического занятия 13

Код программы

```
int flame = A5;// define the flame interface analog 0 interface
int Beep = 8;// buzzer interface defines the interface number 7
int val = 0;// define numeric variables val
void setup ()
    {
        pinMode (Beep, OUTPUT) ;// define LED as output interface
        pinMode (flame, INPUT) ;// define the buzzer as the input interface
        Serial.begin (9600) ;// set the baud rate to 9600
    }
void loop () {
        val = analogRead (flame) ;// read the analog value flame sensor
        Serial.println (val) ;// output analog values, and print them out
        if (val >= 1000) // When the analog value is greater than 600 when the buzzer
sounds
            {
            digitalWrite (Beep, HIGH);
            } else {
            digitalWrite (Beep, LOW);
            }
    }
```

Практическое занятие 14. Эксперимент с вольтметром

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- резистор на 1 кОм;
- макетная плата;
- батарейка/аккумулятор на 1.5 В (**не больше**);
- соединительные провода.

В этом занятии будем строить схему с вольтметром с диапазоном 0–5 В. Нам также понадобится консоль (**Ctrl+Shift+M**), чтобы смотреть, какие значения мы считываем с аналогового порта 0 платы. При подключении схемы в консоли мы будем видеть, какой заряд у батарейки/аккумулятора.

Схема представлена на рис. 69–70.

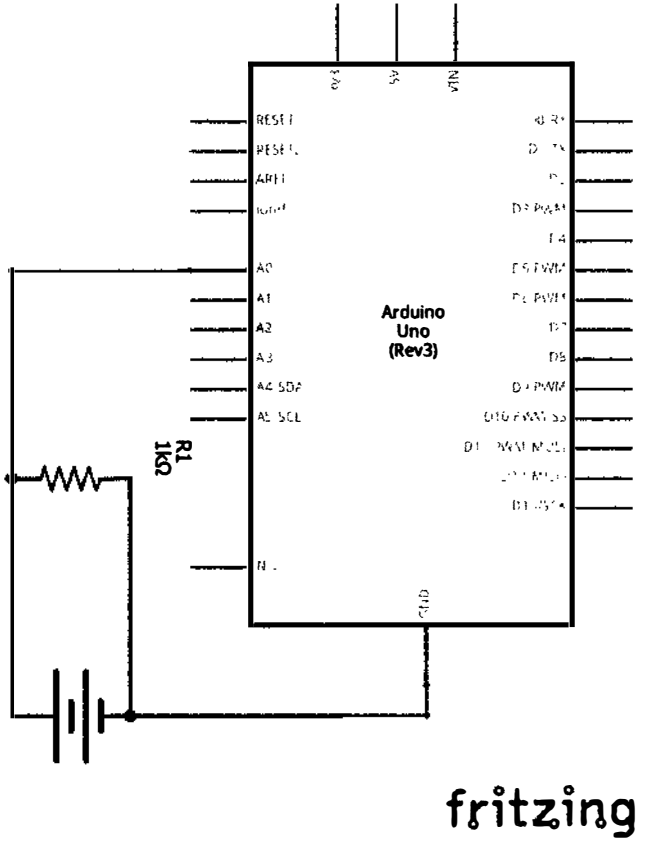


Рис. 69 ❖ Принципиальная схема подключения для практического занятия 14

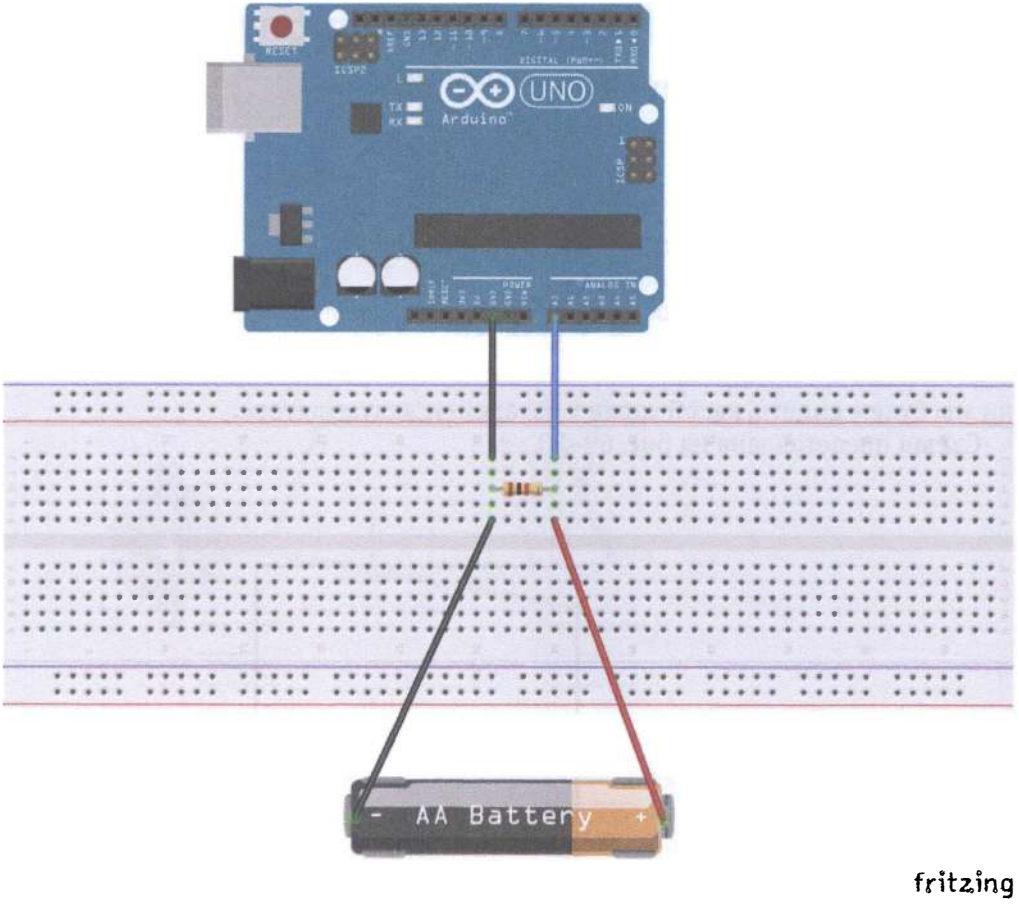


Рис. 70 ❖ Схема подключения с макетной платой для практического занятия 14

Код программы

```
float temp; // create a float variable temp as a storage space to store data preparation
void setup ()
{
Serial.begin (9600); // use 9600 baud rate serial communication
}
void loop ()
{
int V1 = analogRead (A0);
// Read voltage from A0 mouth integer type data into the newly created variable V1, analog
port voltage measurement range of 0-5V returns a value of 0-1024
float vol = V1 * (5.0 / 1023.0);
// We will be converted into the actual value of V1 voltage value into a float variable vol
if (vol == temp)
// This part of the judgment is used to filter duplicate data, only the second voltage
value when the output and the last mixed
```

```
{
temp = vol; // After the completion of the comparison, the ratio of this value into a
variable temp for comparison
}
else
{
Serial.print (vol); // serial output voltage value, and do not wrap
Serial.println ("V"); // serial output character V, and line breaks
temp = vol;
delay (1000); // Wait a second output is completed, the data used to control the refresh
rate.
}
}
```

Практическое занятие 15. Эксперимент с распознаванием голоса

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- модуль распознавания шумов;
- светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

У модуля распознавания шумов есть 4 выхода. A0 – аналоговый выход, который мы подключаем к аналоговому входу A0 платы. G – земля, а + выход надо подключить к 5 В на плате. Далее собираем схему со светодиодом, как обычно, и открываем консоль (**Ctrl+Shift+M**), чтобы смотреть значение шума, создаваемого в аудитории. На схеме ниже представлен другой элемент с 4 выходами (не модуль распознавания шумов), так как в библиотеке Fritzing не оказалось подобного модуля или микрофона с 3 выходами; но главное – суть подключения.

Схема представлена на рис. 71–72.

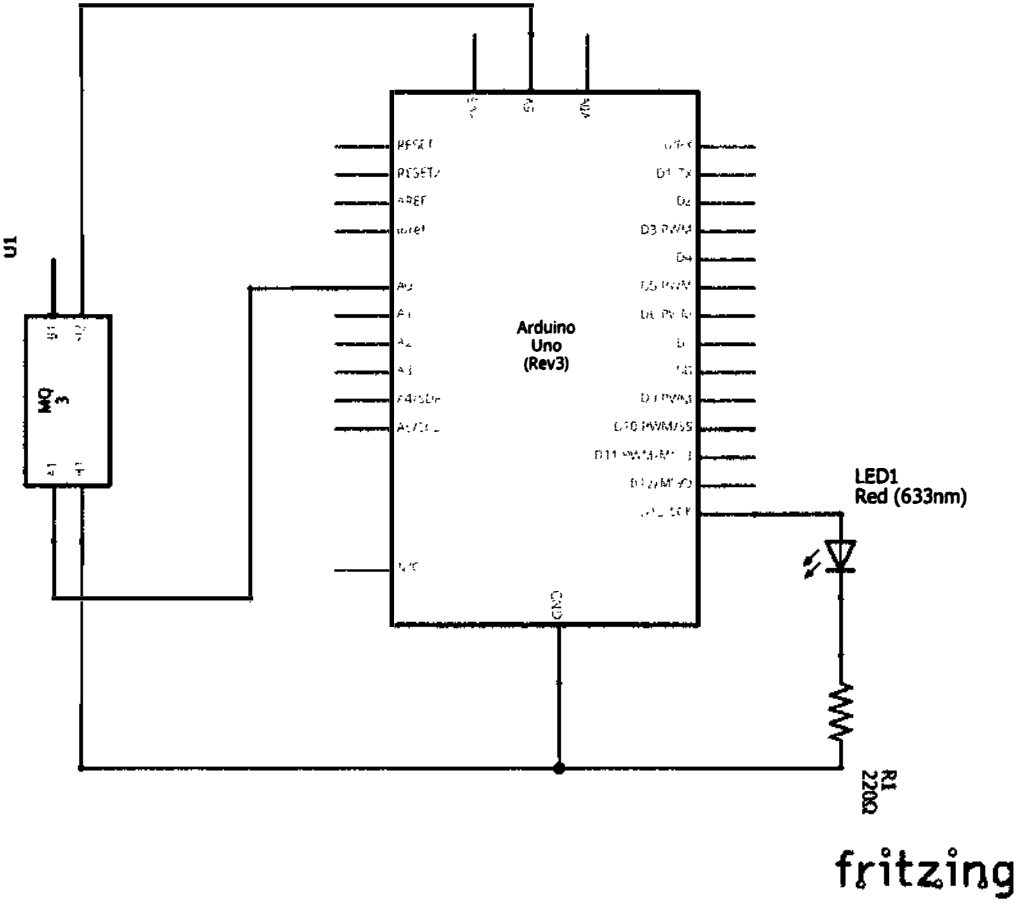


Рис. 71 ❖ Принципиальная схема подключения для практического занятия 15

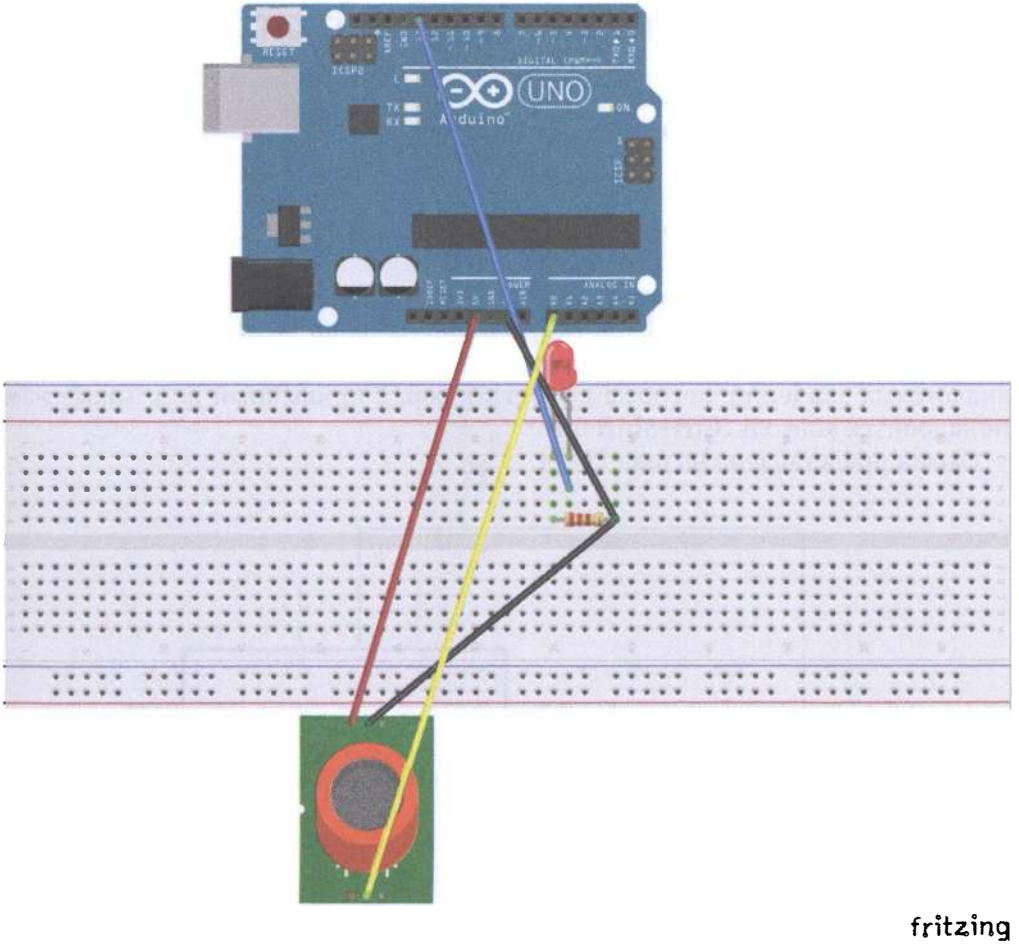


Рис. 72 ❖ Схема подключения с макетной платой для практического занятия 15

Код программы

```
int sensorPin = A0; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor
void setup () {
pinMode (ledPin, OUTPUT);
Serial.begin (9600);
}
void loop () {
sensorValue = analogRead (sensorPin);
digitalWrite (ledPin, HIGH);
delay (sensorValue);
digitalWrite (ledPin, LOW);
delay (sensorValue);
Serial.println (sensorValue, DEC);
}
```

Практическое занятие 16. Эксперимент с температурным сенсором

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- температурный сенсор LM35;
- макетная плата;
- соединительные провода.

Температурный сенсор LM35 достаточно широко используется и прост в эксплуатации. Единственная нетривиальная часть этого задания – перевод аналоговых значений, которые выдаёт сенсор, в привычный вид. Также нам понадобится консоль (**Ctrl+Shift+M**).

Схема представлена на рис. 73–74.

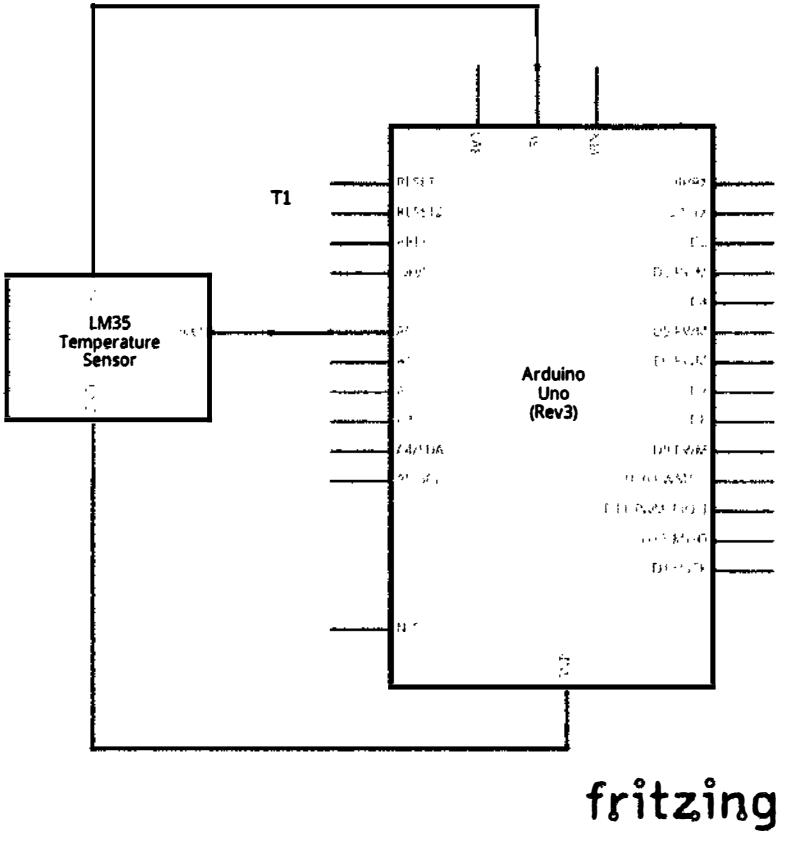


Рис. 73 ❖ Принципиальная схема подключения для практического занятия 16

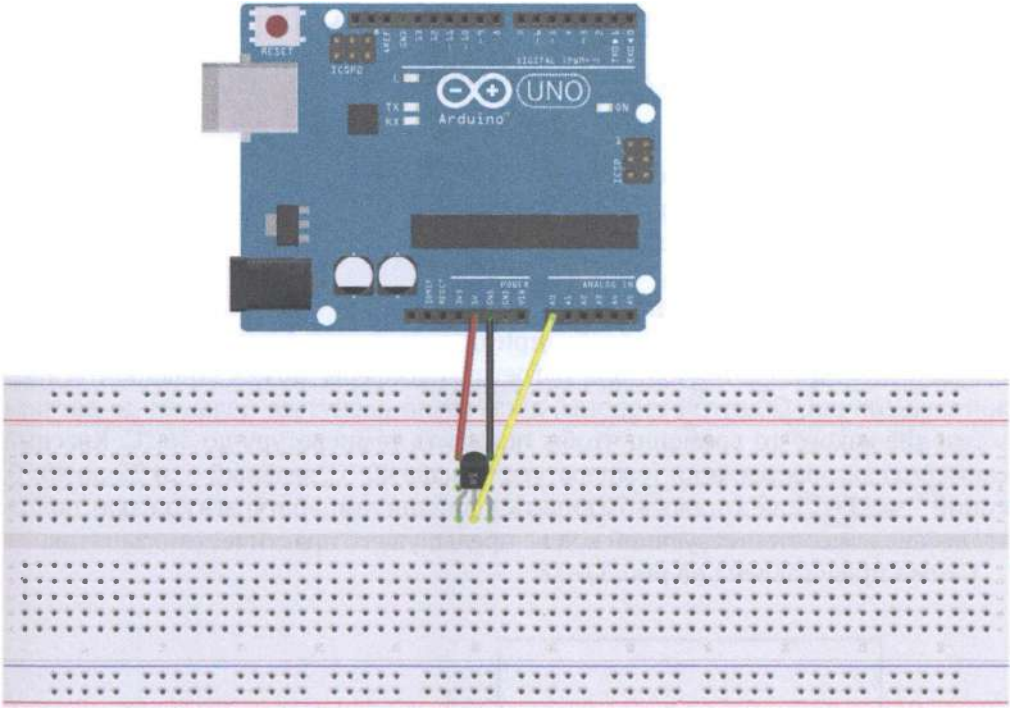


Рис. 74 ❖ Схема подключения с макетной платой для практического занятия 16

Код программы

```
int potPin = A0; // define the analog interface 0 LM35 temperature sensor connection
void setup ()
{
Serial.begin (9600) ;// set the baud rate
}
void loop ()
{
int val ;// define variables
int dat ;// define variables
val = analogRead (potPin) ;// read sensor analog values and assigned to val
dat = (125 * val) >> 8 ;// temperature formula
Serial.print ("Temp:") ;// output as a string represents the temperature display Tep
Serial.print (dat) ;// output shows the value of dat
Serial.println ("C") ;// output display as a C string
delay (500) ;// delay of 0.5 seconds
}
```

Практическое занятие 17. Разноцветный термостат

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- температурный сенсор LM35;
- 3 разноцветных светодиода;
- 3 резистора на 220 Ом;
- макетная плата;
- соединительные провода.

Суть этого задания в том, чтобы при определённой температуре зажигался соответствующий светодиод. Для нагрева атмосферы может понадобиться зажигалка/спички. С другой стороны, достаточно коснуться пальцем до датчика в течение какого-то времени, чтобы повысить температуру до 36 °C. Красный светодиод загорается, если температура больше 40 °C, зелёный – от 32 до 40 °C, синий – до 31 °C. Ещё хотелось бы видеть значения температуры в консоли, поэтому добавляем соответствующий код из предыдущего практического занятия.

Схема представлена на рис. 77–78.

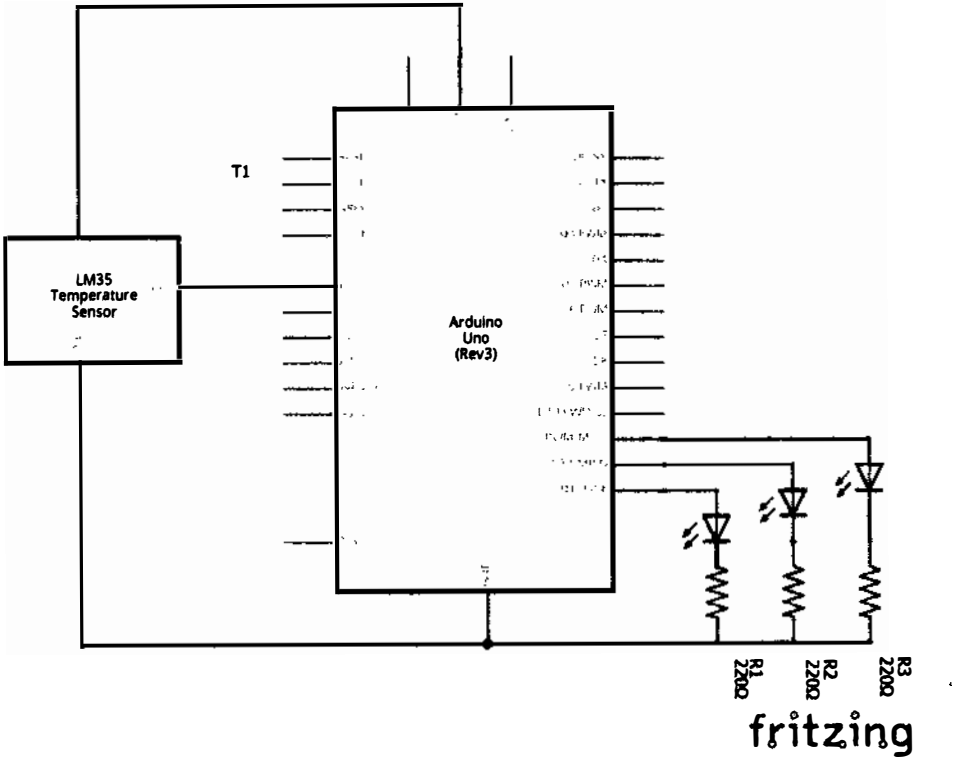


Рис. 75 ❖ Принципиальная схема подключения для практического занятия 17

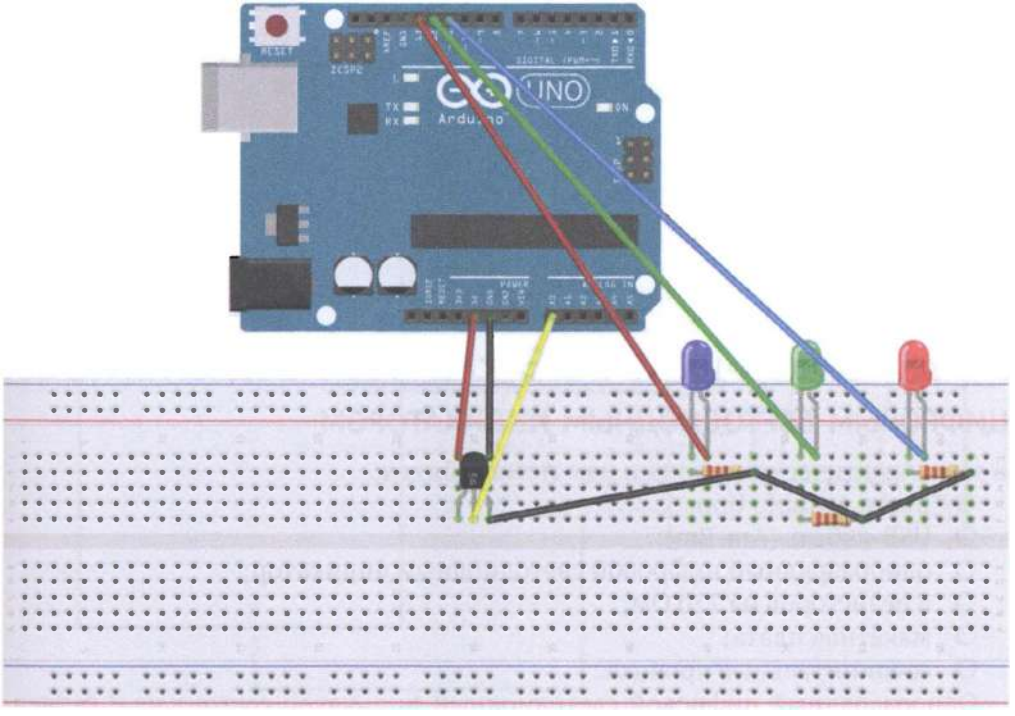


Рис. 76 ❖ Схема подключения с макетной платой для практического занятия 17

Код программы

```
   void setup () {
    pinMode (13, OUTPUT);
    pinMode (12, OUTPUT);
    pinMode (11, OUTPUT);
    Serial.begin (9600);
}
void loop () {
    int vol = analogRead (A0) * (5.0 / 1023.0 * 100); // read the LM35 temperature
    Serial.print ("Temp:") ;// output as a string represents the temperature display Tep
    Serial.print (vol) ;// output shows the value of dat
    Serial.println ("C") ;// output display as a C string
    if (vol <= 31) // set the value of the temperature of the low temperature region, and
led display
    {
        digitalWrite (13, HIGH);
        digitalWrite (12, LOW);
        digitalWrite (11, LOW);
    }
    else if (vol >= 32 && vol <= 40)
    {
        digitalWrite (13, LOW);
```

```
        digitalWrite (12, HIGH);
        digitalWrite (11, LOW);
    }
    else if (vol >= 41) // hot zone temperature setting
    {
        digitalWrite (13, LOW);
        digitalWrite (12, LOW);
        digitalWrite (11, HIGH);
    }
    delay (500);
}
```

Практическое занятие 18. Эксперимент с одноразрядным цифровым светодиодным индикатором

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- одноразрядный цифровой светодиодный индикатор;
- 8 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

Одноразрядный цифровой светодиодный индикатор состоит из 8 сегментов, каждый из которых представляет собой маленький светодиод, отвечающий за соответствующий сегмент цифры на индикаторе (или точки). Бывают два типа разноцветных светодиодов: с общим катодом (когда катодный выход один, а анодных несколько), который подключается к земле, и с общим анодом (анод один, катодов несколько), который подключается к 3.3 или 5 В выходу платы. В этом задании мы построим схему и напишем код так, чтобы индикатор показывал цифры от 1 до 8.

Схема представлена на рис. 77–78.

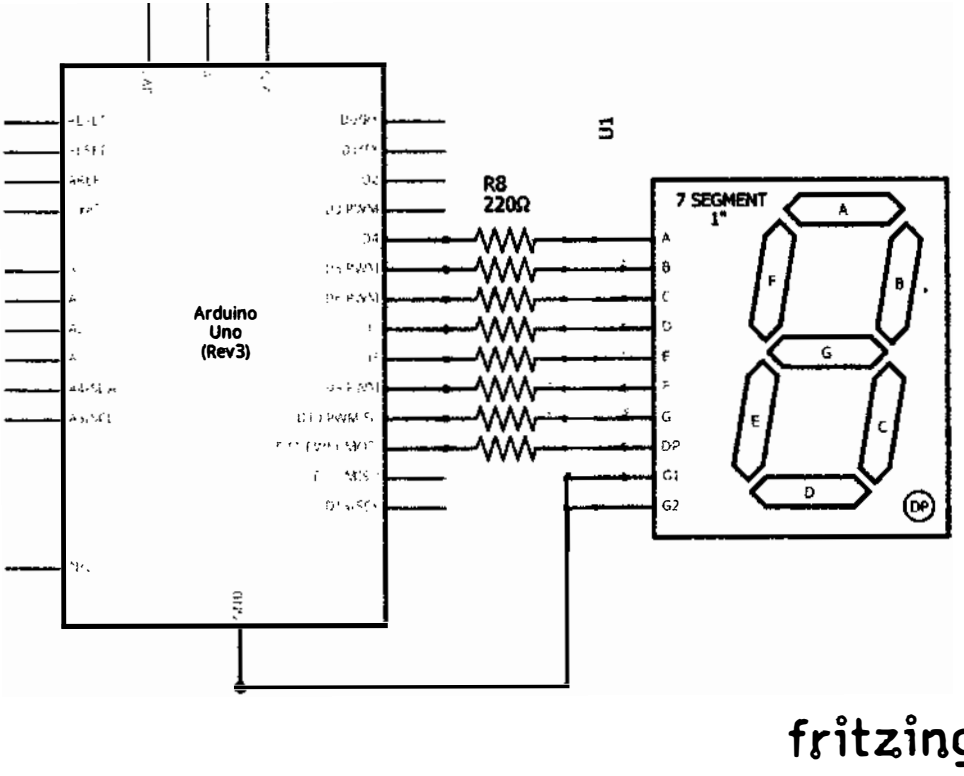


Рис. 77 ❖ Принципиальная схема подключения для практического занятия 18

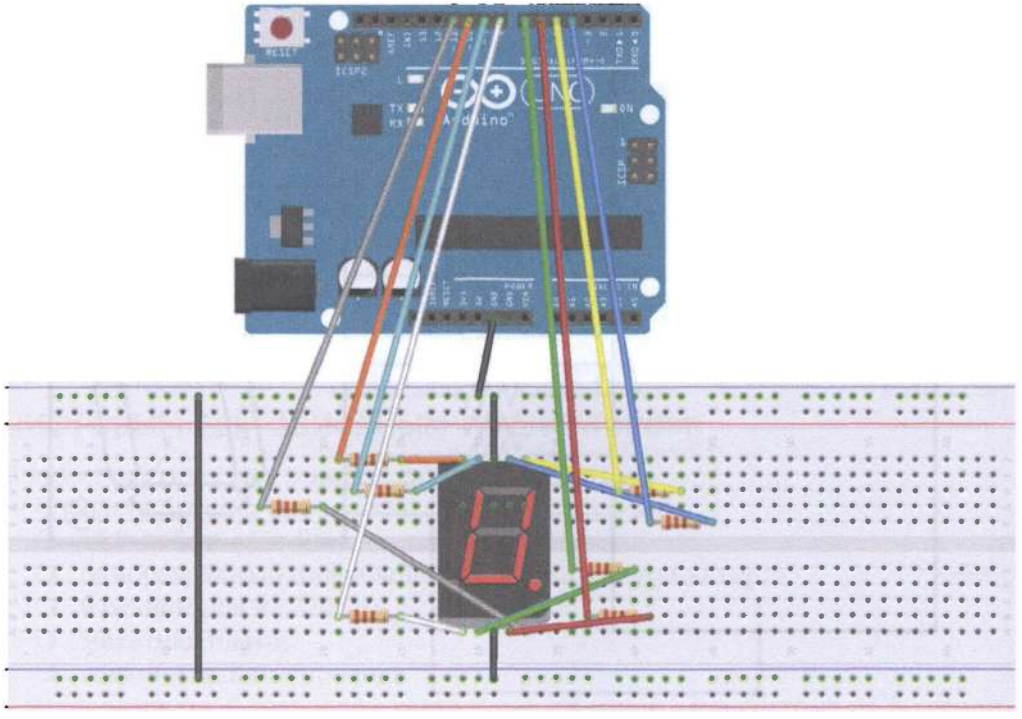


Рис. 78 ❖ Схема подключения с макетной платой для практического занятия 18

Код программы

```
// Set control each segment digital IO pin
    int a = 4 ;// define the digital interface to connect a seven segment LED
    int b = 5 ;// define the connection b Digital Interface 6-segment LED
    int c = 6 ;// define paragraph (c) Digital Interface 5 digital connection
    int d = 7 ;// define the digital interface 11 is connected to d-segment digital tube
    int e = 8 ;// define the digital interface 10 is connected to e-segment digital tube
    int f = 9 ;// define the digital interface 8 digital tube connection f
    int g = 10 ;// define the digital interface 9 g of the digital control connection
    int dp = 11 ;// define the digital interface 4 digital tube connecting dp
    void digital_1 (void) // display the number 1
    {
        digitalWrite (a, LOW);
        digitalWrite (b, HIGH);
        digitalWrite (c, HIGH);
        digitalWrite (d, LOW);
        digitalWrite (e, LOW);
        digitalWrite (f, LOW);
        digitalWrite (g, LOW);
        digitalWrite (dp, LOW);
    }
    void digital_2 (void) // display number 2
```

```
{
    digitalWrite (a, HIGH);
    digitalWrite (b, HIGH);
    digitalWrite (c, LOW);
    digitalWrite (d, HIGH);
    digitalWrite (e, HIGH);
    digitalWrite (f, LOW);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void digital_3 (void) // display the number 3
{
    digitalWrite (a, HIGH);
    digitalWrite (b, HIGH);
    digitalWrite (c, HIGH);
    digitalWrite (d, HIGH);
    digitalWrite (e, LOW);
    digitalWrite (f, LOW);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void digital_4 (void) // show 4
{
    digitalWrite (a, LOW);
    digitalWrite (b, HIGH);
    digitalWrite (c, HIGH);
    digitalWrite (d, LOW);
    digitalWrite (e, LOW);
    digitalWrite (f, HIGH);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void digital_5 (void) // display the number 5
{
    digitalWrite (a, HIGH);
    digitalWrite (b, LOW);
    digitalWrite (c, HIGH);
    digitalWrite (d, HIGH);
    digitalWrite (e, LOW);
    digitalWrite (f, HIGH);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void digital_6 (void) // display the number 6
{
    digitalWrite (a, HIGH);
    digitalWrite (b, LOW);
    digitalWrite (c, HIGH);
    digitalWrite (d, HIGH);
    digitalWrite (e, HIGH);
    digitalWrite (f, HIGH);
    digitalWrite (g, HIGH);
```

```
    digitalWrite (dp, LOW);
}
void digital_7 (void) // display the number 7
{
    digitalWrite (a, HIGH);
    digitalWrite (b, HIGH);
    digitalWrite (c, HIGH);
    digitalWrite (d, LOW);
    digitalWrite (e, LOW);
    digitalWrite (f, LOW);
    digitalWrite (g, LOW);
    digitalWrite (dp, LOW);
}
void digital_8 (void) // display the number 8
{
    digitalWrite (a, HIGH);
    digitalWrite (b, HIGH);
    digitalWrite (c, HIGH);
    digitalWrite (d, HIGH);
    digitalWrite (e, HIGH);
    digitalWrite (f, HIGH);
    digitalWrite (g, HIGH);
    digitalWrite (dp, LOW);
}
void setup ()
{
int i ;// define variables
for (i = 4; i <= 11; i++)
pinMode (i, OUTPUT) ;// set 4 to 11 pin to output mode
}
void loop ()
{
while (1)
{
digital_1 () ;// display numbers 1
delay (1000) ;// delay 2s
digital_2 () ;// display number 2
delay (1000); // delay 1s
digital_3 () ;// display the number 3
delay (1000); // delay 1s
digital_4 () ;// show 4
delay (1000); // delay 1s
digital_5 () ;// display the number 5
delay (1000); // delay 1s
digital_6 () ;// display the number 6
delay (1000); // delay 1s
digital_7 () ;// display the number 7
delay (1000); // delay 1s
digital_8 () ;// display the number 8
delay (1000); // delay 1s
}
}
```

Думаю, вам не составит особого труда **добавить ещё 2 цифры – 0 и 9!**

Практическое занятие 19. Эксперимент с четырёхразрядным цифровым светодиодным индикатором

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- четырёхразрядный цифровой светодиодный индикатор;
- 8 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

У четырёхразрядного цифрового светодиодного индикатора 12 разъёмов, по сравнению с одноразрядным, у которого 10. В принципе, не намного больше. Нумеруются они против часовой стрелки: если положить перед собой индикатор, внизу слева направо будут пины 1–6, а вверху справа налево – 7–12. Соберём схему и напишем программу, которая будет показывать значения от 0:00 до 9:00 в цикле.

Схема представлена на рис. 79–80.

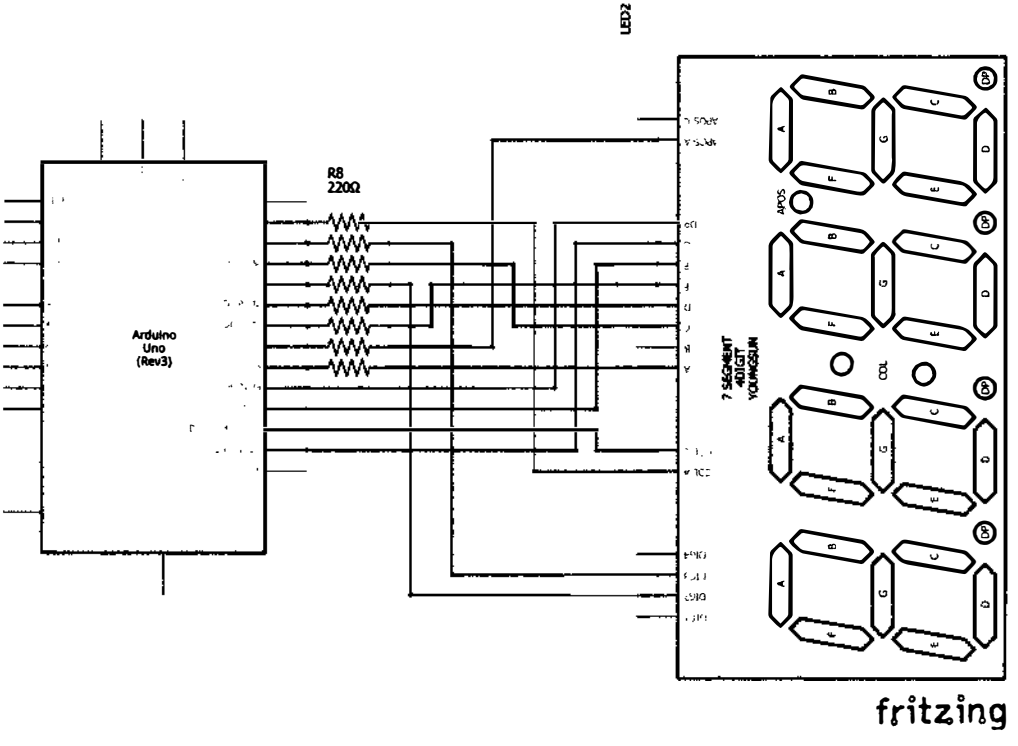


Рис. 79 ❖ Принципиальная схема подключения для практического занятия 19

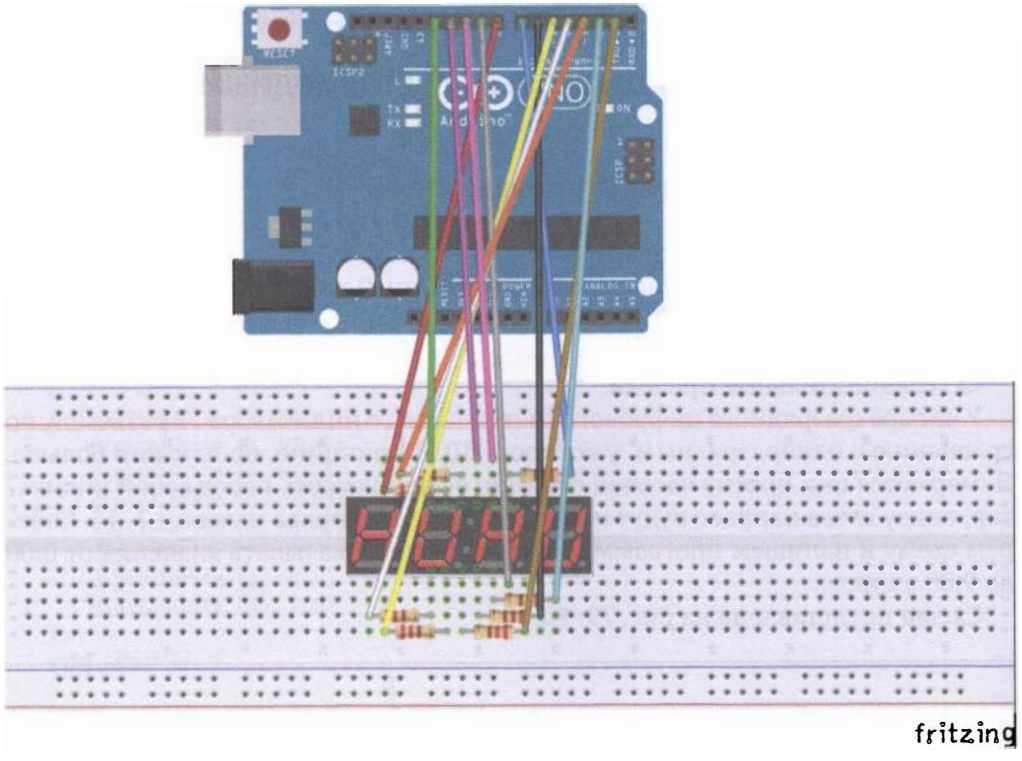


Рис. 80 ❖ Схема подключения с макетной платой для практического занятия 19

Код программы

```
int a = 8;
int b = 7;
int c = 6;
int d = 5;
int e = 4;
int f = 3;
int g = 2;
int p = 1;

int d4 = 9;
int d3 = 10;
int d2 = 11;
int d1 = 12;

long n = 0;
int x = 100;
int del = 55;

void setup()
{
    pinMode(d1, OUTPUT);
    pinMode(d2, OUTPUT);
```

```
    pinMode(d3, OUTPUT);
    pinMode(d4, OUTPUT);
    pinMode(a, OUTPUT);
    pinMode(b, OUTPUT);
    pinMode(c, OUTPUT);
    pinMode(d, OUTPUT);
    pinMode(e, OUTPUT);
    pinMode(f, OUTPUT);
    pinMode(g, OUTPUT);
    pinMode(p, OUTPUT);
}

void loop()
{
    clearLEDs();
    pickDigit(1);
    pickNumber((n/x/1000)%10);
    delayMicroseconds(del);

    clearLEDs();
    pickDigit(2);
    pickNumber((n/x/100)%10);
    delayMicroseconds(del);

    clearLEDs();
    pickDigit(3);
    dispDec(3);
    pickNumber((n/x/10)%10);
    delayMicroseconds(del);

    clearLEDs();
    pickDigit(4);
    pickNumber(n/x%10);
    delayMicroseconds(del);

    n++;

    if (digitalRead(13) == LOW)
    {
        n = 0;
    }
}

void pickDigit(int x)
{
    digitalWrite(d1, HIGH);
    digitalWrite(d2, HIGH);
    digitalWrite(d3, HIGH);
    digitalWrite(d4, HIGH);

    switch(x)
    {
        case 1:
            digitalWrite(d1, LOW);
            break;
```

```
        case 2:
            digitalWrite(d2, LOW);
            break;
        case 3:
            digitalWrite(d3, LOW);
            break;
        default:
            digitalWrite(d4, LOW);
            break;
    }
}

void pickNumber(int x)
{
    switch(x)
    {
        default:
            zero();
            break;
        case 1:
            one();
            break;
        case 2:
            two();
            break;
        case 3:
            three();
            break;
        case 4:
            four();
            break;
        case 5:
            five();
            break;
        case 6:
            six();
            break;
        case 7:
            seven();
            break;
        case 8:
            eight();
            break;
        case 9:
            nine();
            break;
    }
}

void dispDec(int x)
{
    digitalWrite(p, LOW);
}
```

```
void clearLEDs()
{
    digitalWrite(a, LOW);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
    digitalWrite(p, LOW);
}

void zero()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, LOW);
}

void one()
{
    digitalWrite(a, LOW);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
}

void two()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, LOW);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, LOW);
    digitalWrite(g, HIGH);
}

void three()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, HIGH);
```

```
}
void four()
{
    digitalWrite(a, LOW);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
}
void five()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, LOW);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
}
void six()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
}
void seven()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
}
void eight()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
```

```
    digitalWrite(g, HIGH);
}
void nine()
{
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, LOW);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
}
```

Практическое занятие 20. Эксперимент со светодиодной матрицей

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- светодиодная матрица 8×8;
- 8 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

Светодиодная матрица 8×8 содержит 64 светодиода, соединённых так, чтобы можно было зажечь соответствующий светодиод, пользуясь его *X*- и *У*-координатой и подавая, соответственно, на одну координату землю через резистор, а на другую – управляющий сигнал с одного из цифровых пинов-выходов платы Arduino Uno. К сожалению, в parts-библиотеке Fritzing не нашлось соответствующего элемента, поэтому пришлось пририсовывать светодиодной матрице 8×8 с 12 выходами ещё 4 на схеме с макетной платой и рисовать с нуля принципиальную схему. Напишем два кода: первый будет просто зажигать и гасить верхний левый светодиод на матрице, второй – выводить буквы английского алфавита от A до I.

Схема представлена на рис. 81–82.

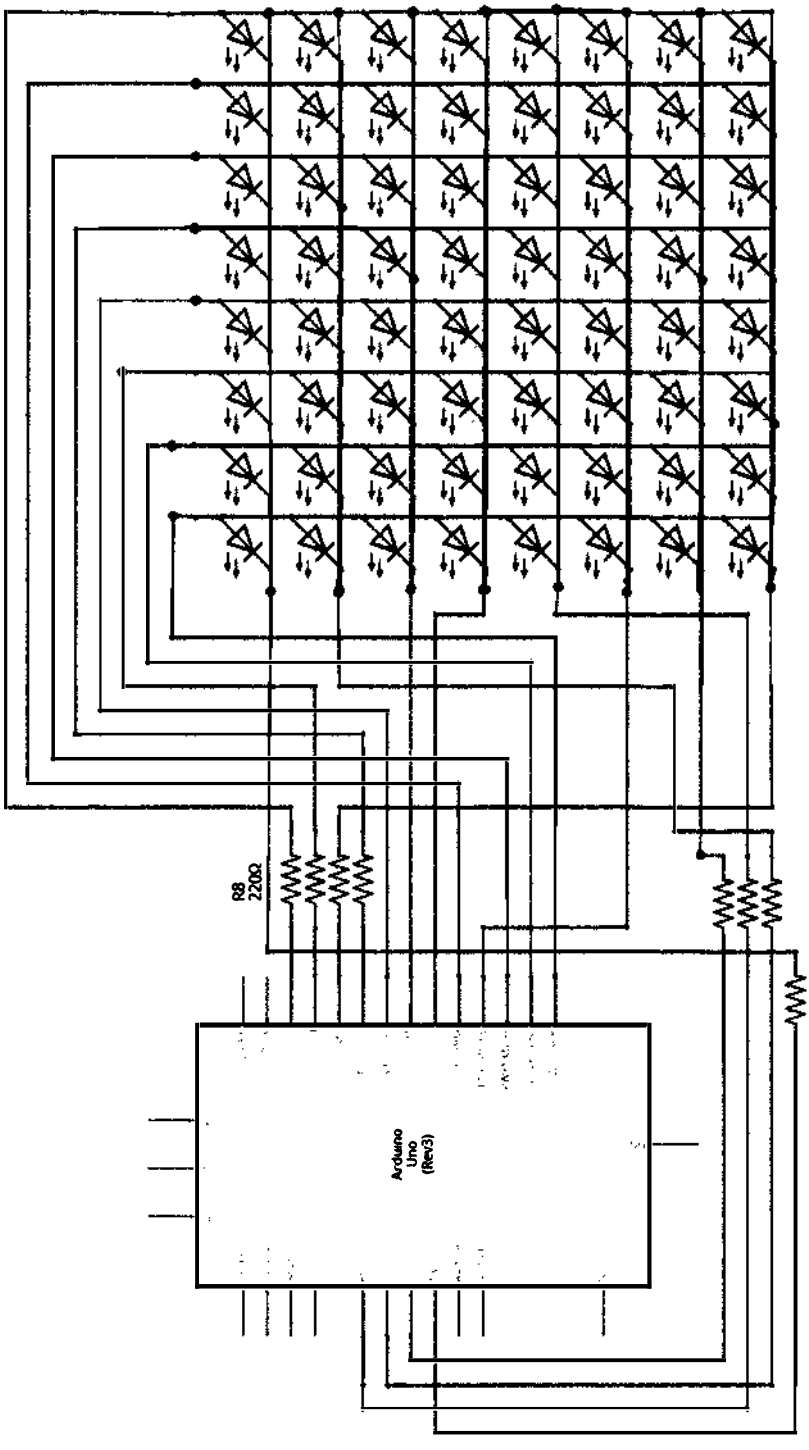


Рис. 81 ❖ Принципиальная схема подключения для практического занятия 20

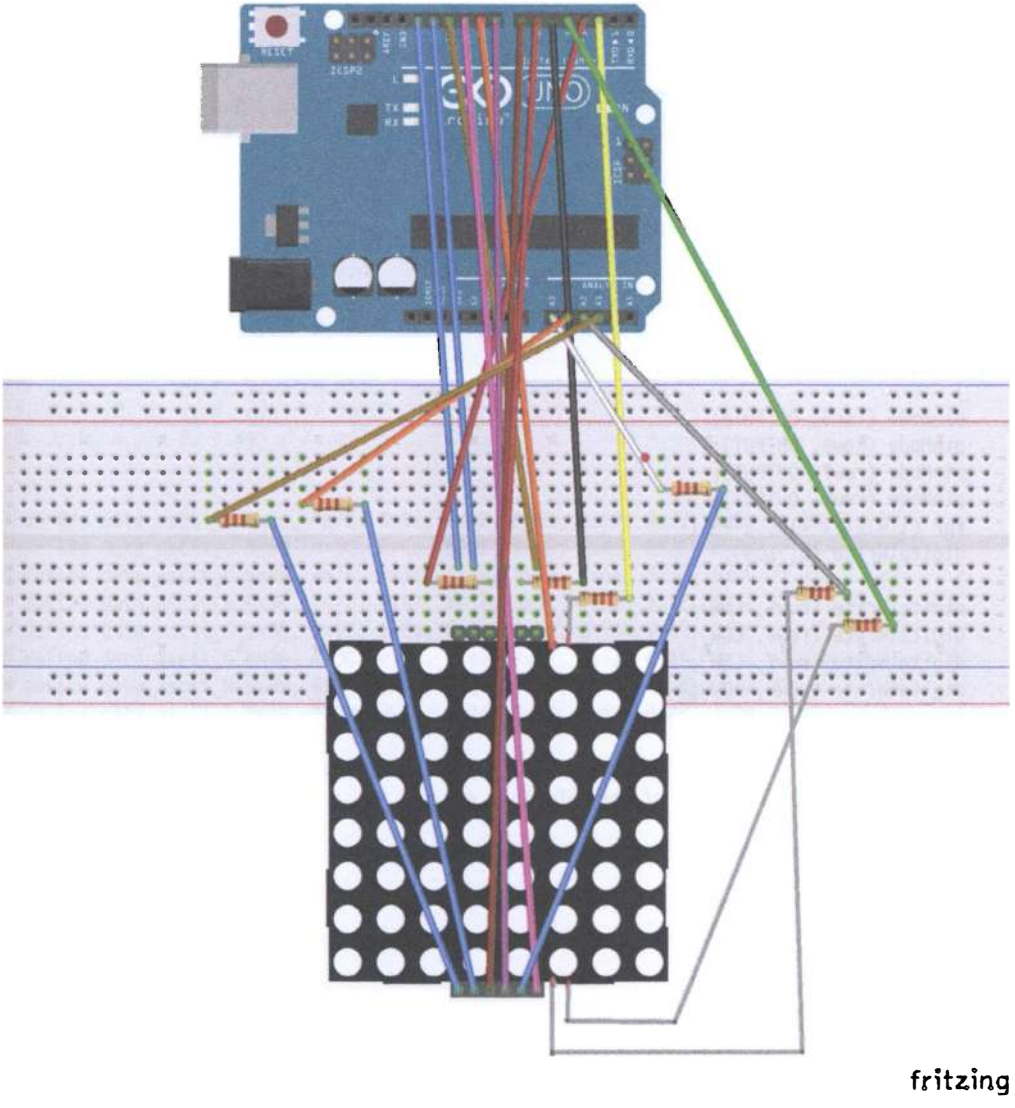


Рис. 82 ❖ Схема подключения с макетной платой для практического занятия 20

Код программы 1

```
const int row1 = 2; // the number of the row pin 9
const int row2 = 3; // the number of the row pin 14
const int row3 = 4; // the number of the row pin
const int row4 = 5; // the number of the row pin 12
const int row5 = 17; // the number of the row pin 1
const int row6 = 16; // the number of the row pin 7
const int row7 = 15; // the number of the row pin 2
const int row8 = 14; // the number of the row pin 5
```

```
// The pin to control COl
const int col1 = 6; // the number of the col pin 13
const int col2 = 7; // the number of the col pin 3
const int col3 = 8; // the number of the col pin 4
const int col4 = 9; // the number of the col pin 10
const int col5 = 10; // the number of the col pin 6
const int col6 = 11; // the number of the col pin 11
const int col7 = 12; // the number of the col pin 15
const int col8 = 13; // the number of the col pin 16
void setup () {
int i = 0;
for (i = 2; i<18; i++)
    {
    pinMode (i, OUTPUT);
    }
    pinMode (row5, OUTPUT);
    pinMode (row6, OUTPUT);
    pinMode (row7, OUTPUT);
    pinMode (row8, OUTPUT);
    for (i = 2; i <18; i++) {
    digitalWrite (i, LOW);
    }
    digitalWrite (row5, LOW);
    digitalWrite (row6, LOW);
    digitalWrite (row7, LOW);
    digitalWrite (row8, LOW);
    }
    void loop () {
    int i;
    // The row # 1 and col # 1 of the LEDs turn on
    digitalWrite (row1, HIGH);
    digitalWrite (row2, LOW);
    digitalWrite (row3, LOW);
    digitalWrite (row4, LOW);
    digitalWrite (row5, LOW);
    digitalWrite (row6, LOW);
    digitalWrite (row7, LOW);
    digitalWrite (row8, LOW);
    digitalWrite (col1, LOW);
    digitalWrite (col2, HIGH);
    digitalWrite (col3, HIGH);
    digitalWrite (col4, HIGH);
    digitalWrite (col5, HIGH);
    digitalWrite (col6, HIGH);
    digitalWrite (col7, HIGH);
    digitalWrite (col8, HIGH);
    delay (1000);
    // Turn off all
    for (i = 2; i <18; i++) {
    digitalWrite (i, LOW);
    }
    delay (1000);
    }
```

Код программы 2

```
# define display_array_size 8
// Ascii 8x8 dot font
# define data_null 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 // null char
# define data_ascii_A 0x02, 0x0C, 0x18, 0x68, 0x68, 0x18, 0x0C, 0x02 /* «A», 0 */
/**
** «A»
# Define A {//
{0, 0, 0, 0, 0, 0, 1, 0}, / / 0x02
{0, 0, 0, 0, 1, 1, 0, 0}, / / 0x0C
{0, 0, 0, 1, 1, 0, 0, 0}, / / 0x18
{0, 1, 1, 0, 1, 0, 0, 0}, / / 0x68
{0, 1, 1, 0, 1, 0, 0, 0}, / / 0x68
{0, 0, 0, 1, 1, 0, 0, 0}, / / 0x18
{0, 0, 0, 0, 1, 1, 0, 0}, / / 0x0C
{0, 0, 0, 0, 0, 0, 1, 0} / / 0x02
}
**/
# define data_ascii_B 0x00, 0x7E, 0x52, 0x52, 0x52, 0x52, 0x2C, 0x00 /* «B», 1 */
# define data_ascii_C 0x00, 0x3C, 0x66, 0x42, 0x42, 0x42, 0x2C, 0x00 /* «C», 2 */
# define data_ascii_D 0x00, 0x7E, 0x42, 0x42, 0x42, 0x66, 0x3C, 0x00 /* «D», 3 */
# define data_ascii_E 0x00, 0x7E, 0x52, 0x52, 0x52, 0x52, 0x52, 0x42 /* «E», 4 */
# define data_ascii_F 0x00, 0x7E, 0x50, 0x50, 0x50, 0x50, 0x50, 0x40 /* «F», 5 */
# define data_ascii_G 0x00, 0x3C, 0x66, 0x42, 0x42, 0x52, 0x16, 0x1E /* «G», 6 */
# define data_ascii_H 0x00, 0x7E, 0x10, 0x10, 0x10, 0x10, 0x7E, 0x00 /* «H», 7 */
# define data_ascii_I 0x00, 0x00, 0x00, 0x7E, 0x00, 0x00, 0x00, 0x00 /* «I», 8 */
// Display array
byte data_ascii [] [display_array_size] = {
data_null,
data_ascii_A, data_ascii_B,
data_ascii_C,
data_ascii_D,
data_ascii_E,
data_ascii_F,
data_ascii_G,
data_ascii_H,
data_ascii_I,
};
// The pin to control ROW
const int row1 = 2; // the number of the row pin 24
const int row2 = 3; // the number of the row pin 23
const int row3 = 4; // the number of the row pin 22
const int row4 = 5; // the number of the row pin 21
const int row5 = 17; // the number of the row pin 4
const int row6 = 16; // the number of the row pin 3
const int row7 = 15; // the number of the row pin 2
const int row8 = 14; // the number of the row pin 1
// The pin to control COl
const int col1 = 6; // the number of the col pin 20
const int col2 = 7; // the number of the col pin 19
const int col3 = 8; // the number of the col pin 18
const int col4 = 9; // the number of the col pin 17
```

```
const int col5 = 10; // the number of the col pin 16
const int col6 = 11; // the number of the col pin 15
const int col7 = 12; // the number of the col pin 14
const int col8 = 13; // the number of the col pin 13
void displayNum (byte rowNum, int colNum)
  {
     int j;
     byte temp = rowNum;
     for (j = 2; j <6; j++)
       {
          digitalWrite (j, LOW);
       }
    digitalWrite (row5, LOW);
    digitalWrite (row6, LOW);
    digitalWrite (row7, LOW);
    digitalWrite (row8, LOW);
    for (j = 6; j <14; j++)
      {
        digitalWrite (j, HIGH);}
        switch (colNum)
      {
    case 1: digitalWrite (col1, LOW); break;
    case 2: digitalWrite (col2, LOW); break;
    case 3: digitalWrite (col3, LOW); break;
    case 4: digitalWrite (col4, LOW); break;
    case 5: digitalWrite (col5, LOW); break;
    case 6: digitalWrite (col6, LOW); break;
    case 7: digitalWrite (col7, LOW); break;
    case 8: digitalWrite (col8, LOW); break;
    default: break;
    }
    for (j = 1; j <9; j++)
      {
       temp = (0x80) & (temp);
       if (temp == 0)
         {
           temp = rowNum << j;
           continue;
         }
        switch (j)
        {
         case 1: digitalWrite (row1, HIGH); break;
         case 2: digitalWrite (row2, HIGH); break;
         case 3: digitalWrite (row3, HIGH); break;
         case 4: digitalWrite (row4, HIGH); break;
         case 5: digitalWrite (row5, HIGH); break;
         case 6: digitalWrite (row6, HIGH); break;
         case 7: digitalWrite (row7, HIGH); break;
         case 8: digitalWrite (row8, HIGH); break;
         default: break;
        }
      temp = rowNum << j;
```

```
  }
}

void setup () {
int i = 0;
for (i = 2; i <18; i++)
{
pinMode (i, OUTPUT);
}
for (i = 2; i <18; i++) {
digitalWrite (i, LOW);
}
}
void loop () {
int t1;
int l;
int arrage;
for (arrage = 0; arrage <10; arrage++)
{
for (l = 0; l <512; l++)
  {
    for (t1 = 0; t1 <8; t1++)
      {
        displayNum (data_ascii [arrage] [t1], (t1 +1));
      }
   }
}
}
```

Практическое занятие 21. Эксперимент с трёхцветным светодиодом

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- трёхцветный светодиод;
- 3 резистора на 220 Ом;
- макетная плата;
- соединительные провода.

В этом занятии мы сначала выведем красный, белый и синий цвета с помощью трёхцветного (RGB) светодиода, а затем выведем смешанные цвета. У этого светодиода 4 выхода: 3 анода (красный, жёлтый, синий цвета) и 1 катод (на землю). Такой светодиод называется светодиодом с общим катодом. Есть также светодиод с общим анодом, когда трём цветам соответствуют 3 катодных выхода. В комплекте Arduino Uno Starter Learning Kit с RFID-модулем нет отдельного трёхцветного светодиода, но есть небольшой модуль со встроенным трёхцветным светодиодом, который мы и будем использовать. Вообще,

на модуле уже есть встроенные сопротивления, но на всякий случай лучше всё-таки использовать дополнительные внешние резисторы на 220 Ом. Обратите внимание на маркировку пинов на модуле с трёхцветным светодиодом – она может отличаться от приведённой ниже схемы подключения (например: R, G, B подключаются, соответственно, к 11, 9, 10 и GND пинам платы)!

Схема представлена на рис. 83–84.

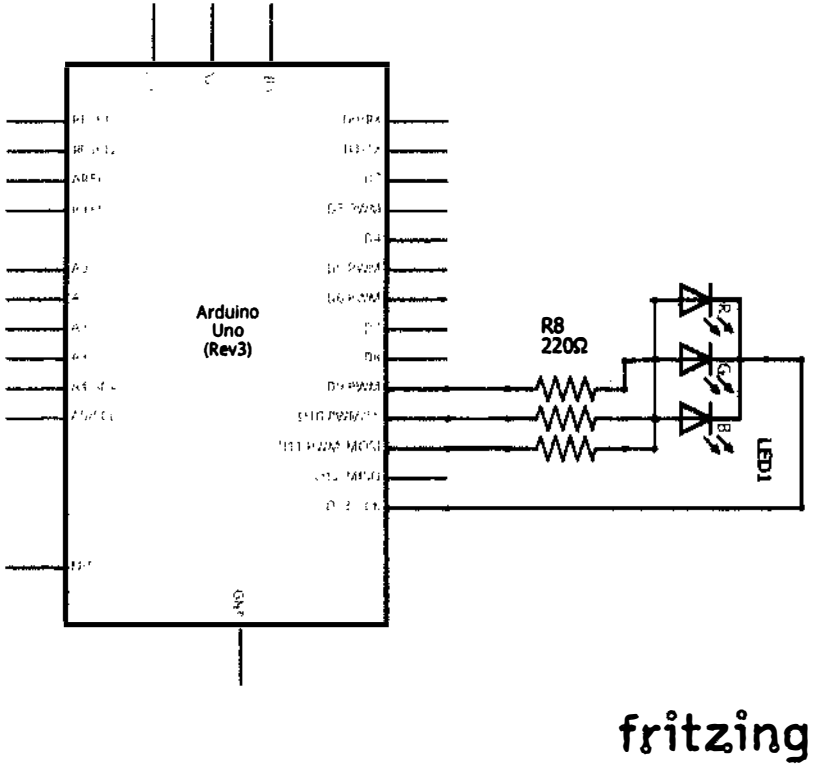


Рис. 83 ❖ Принципиальная схема подключения для практического занятия 21

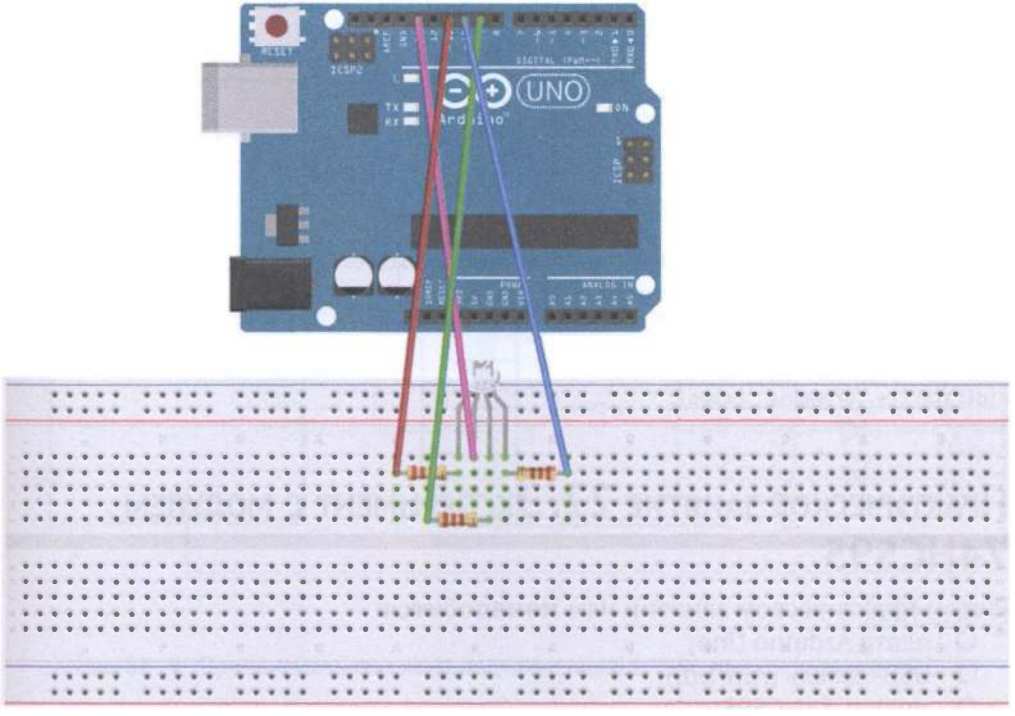


Рис. 84 ❖ Схема подключения с макетной платой для практического занятия 21

Код программы

```
int ledPin = 13; // LED is connected to digital pin 13
int redPin = 11; // R petal on RGB LED module connected to digital pin 11
int greenPin = 9; // G petal on RGB LED module connected to digital pin 9
int bluePin = 10; // B petal on RGB LED module connected to digital pin 10
void setup ()
{
pinMode (ledPin, OUTPUT); // sets the ledPin to be an output
pinMode (redPin, OUTPUT); // sets the redPin to be an output
pinMode (greenPin, OUTPUT); // sets the greenPin to be an output
pinMode (bluePin, OUTPUT); // sets the bluePin to be an output
}
void loop () // run over and over again
{
// Basic colors:
color (255, 0, 0); // turn the RGB LED red
delay (1000); // delay for 1 second
color (0,255, 0); // turn the RGB LED green
delay (1000); // delay for 1 second
color (0, 0, 255); // turn the RGB LED blue
delay (1000); // delay for 1 second
// Example blended colors:
```

```
color (255,255,0); // turn the RGB LED yellow
delay (1000); // delay for 1 second
color (255,255,255); // turn the RGB LED white
delay (1000); // delay for 1 second
color (128,0,255); // turn the RGB LED purple
delay (1000); // delay for 1 second
color (0,0,0); // turn the RGB LED off
delay (1000); // delay for 1 second
}
void color (unsigned char red, unsigned char green, unsigned char blue) // the color
generating function
{
digitalWrite (redPin, red);
digitalWrite (bluePin, blue);
digitalWrite (greenPin, green);
}
```

Практическое занятие 22. Эксперимент с модулем 74HC595

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- модуль 74HC595;
- 8 светодиодов;
- 8 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

Зачем использовать модуль 74HC595, у которого есть сдвиговые регистры и память, для работы 8 светодиодов? Дело в том, что на любой плате есть ограниченное количество входов и выходов (пинов). Для работы 8 светодиодов нам надо занять 9 пинов платы (вместе с землёй). Рассматриваемый модуль сократит количество задействованных пинов до 5. **Будьте внимательны, подключайте модуль правильно, иначе плата будет перегреваться и выключаться.**

Схема представлена на рис. 85–86.

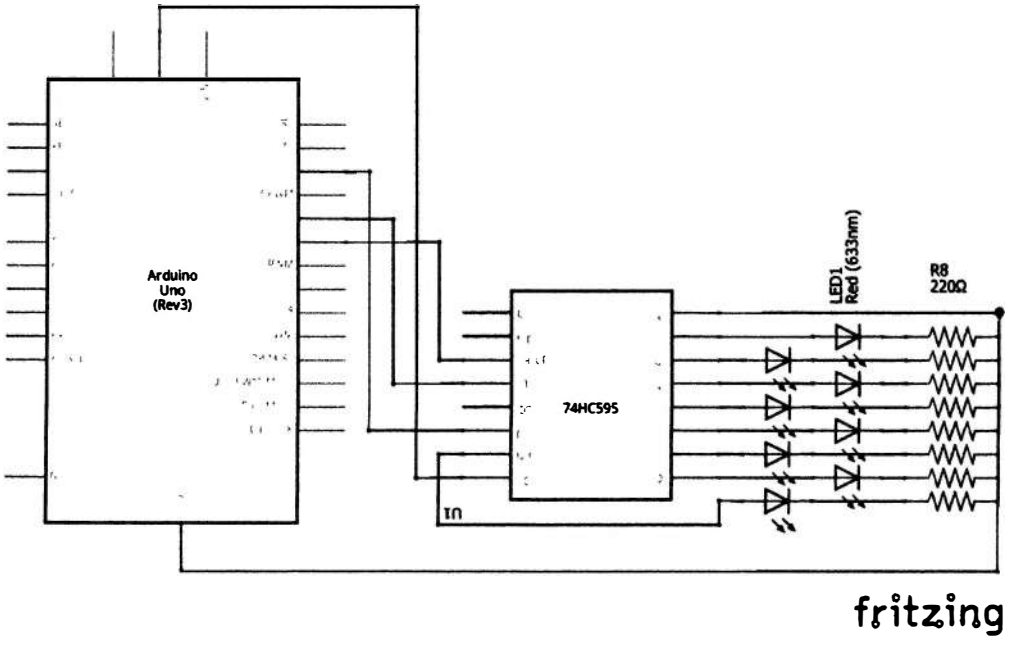


Рис. 85 ❖ Принципиальная схема подключения для практического занятия 22

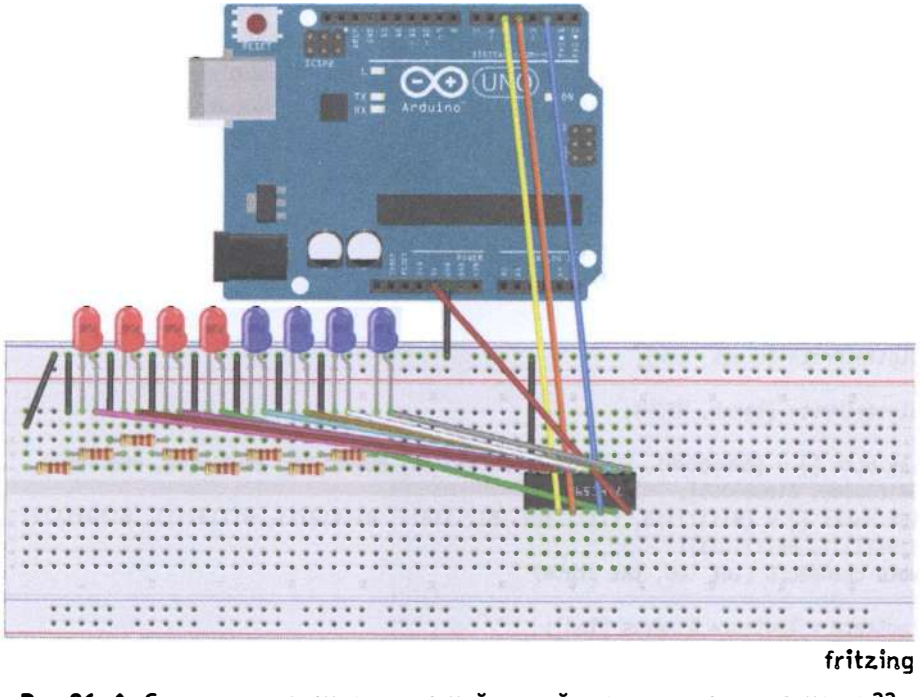


Рис. 86 ❖ Схема подключения с макетной платой для практического занятия 22

Код программы

```
int data = 2;
int clock = 4;
int latch = 5;
int ledState = 0;
const int ON = HIGH;
const int OFF = LOW;
void setup ()
{
    pinMode (data, OUTPUT);
    pinMode (clock, OUTPUT);
    pinMode (latch, OUTPUT);
}
void loop ()
    {
    int delayTime = 100;
    for (int i = 0; i <256; i++)
    {
    updateLEDs (i);
    delay (delayTime);
    }
    }
    void updateLEDs (int value)
    {
    digitalWrite (latch, LOW);
    shiftOut (data, clock, MSBFIRST, value);
    digitalWrite (latch, HIGH);
    }
    void updateLEDsLong (int value)
    {
    digitalWrite (latch, LOW);
    for (int i = 0; i <8; i++)
    {
    int bit = value & B10000000;
    value = value << 1;
    if (bit == 128) {digitalWrite (data, HIGH);}
    else {digitalWrite (data, LOW);}
    digitalWrite (clock, HIGH);
    delay (1);
    digitalWrite (clock, LOW);
    }
    digitalWrite (latch, HIGH);
    }
    int bits [] = {B00000001, B00000010, B00000100, B00001000, B00010000, B00100000,
    B01000000, B10000000};
    int masks [] = {B11111110, B11111101, B11111011, B11110111, B11101111, B11011111,
    B10111111, B01111111};
    void changeLED (int led, int state)
    {
    ledState = ledState & masks [led];
    if (state == ON) {ledState = ledState | bits [led];}
    updateLEDs (ledState);
    }
```

Практическое занятие 23. Кнопочный модуль 4×4 и библиотеки

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- кнопочный модуль 4×4;
- соединительные провода.

В этом занятии мы будем использовать кнопочный модуль 4×4 с 16 кнопками. В библиотеке компонентов Fritzing есть похожий компонент Buttonpad-4×4 в разделе Sparkfun-Electromechanical, однако внутри кнопки никак не соединены между собой и присутствуют светодиоды, которые так же висят в воздухе, как и кнопки, и вообще нам не нужны; поэтому для принципиальной схемы пришлось соединять кнопки между собой и выводить необходимые в занятии 8 выходов компонента. Также нам понадобится консоль, или монитор порта (**Ctrl+Shift+M**), чтобы смотреть результаты нажатия кнопок. Кроме того, нам понадобится библиотека Keypad.h, которую можно скачать с официального сайта Arduino по адресу [5]. Чтобы добавить библиотеку в проект Arduino IDE, нужно выбрать пункт меню **Скетч – Подключить библиотеку – Добавить .ZIP библиотеку...**, выбрать скачанный архив Keypad.zip и затем снова нажать **Скетч – Подключить библиотеку**, после чего выбрать библиотеку Keypad – она будет внизу списка. При этом в начало кода добавляется строка `#include <Keypad.h>`. Первая программа выводит символы в соответствии с нажатой кнопкой, вторая – задействует встроенный мини-светодиод на плате, который горит, пока нажата кнопка, ответственная за знак `'*'`, и загорается или гаснет по нажатии кнопки, ответственной за символ `'#'`.

Схема представлена на рис. 87–88.

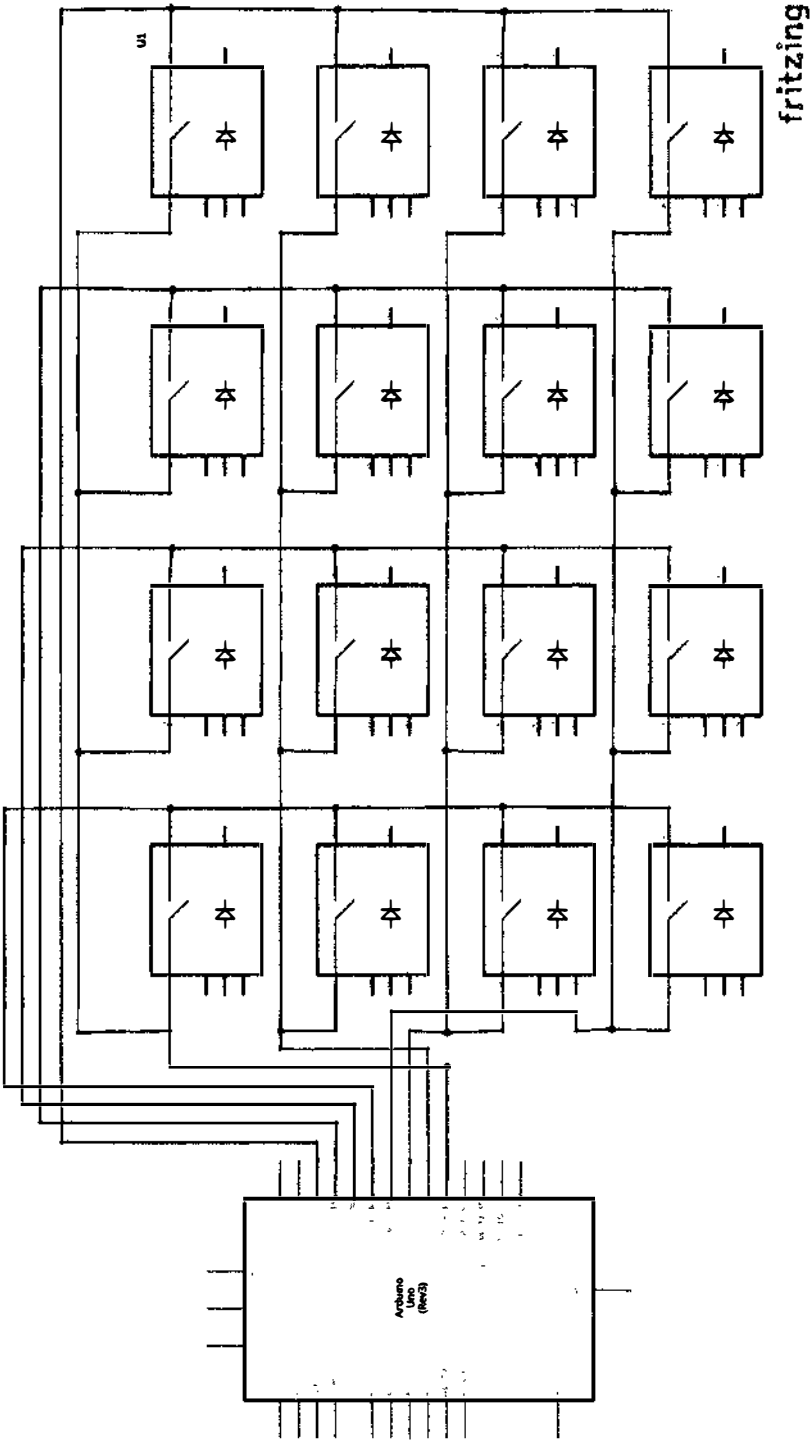


Рис. 87 ❖ Принципиальная схема подключения для практического занятия 23

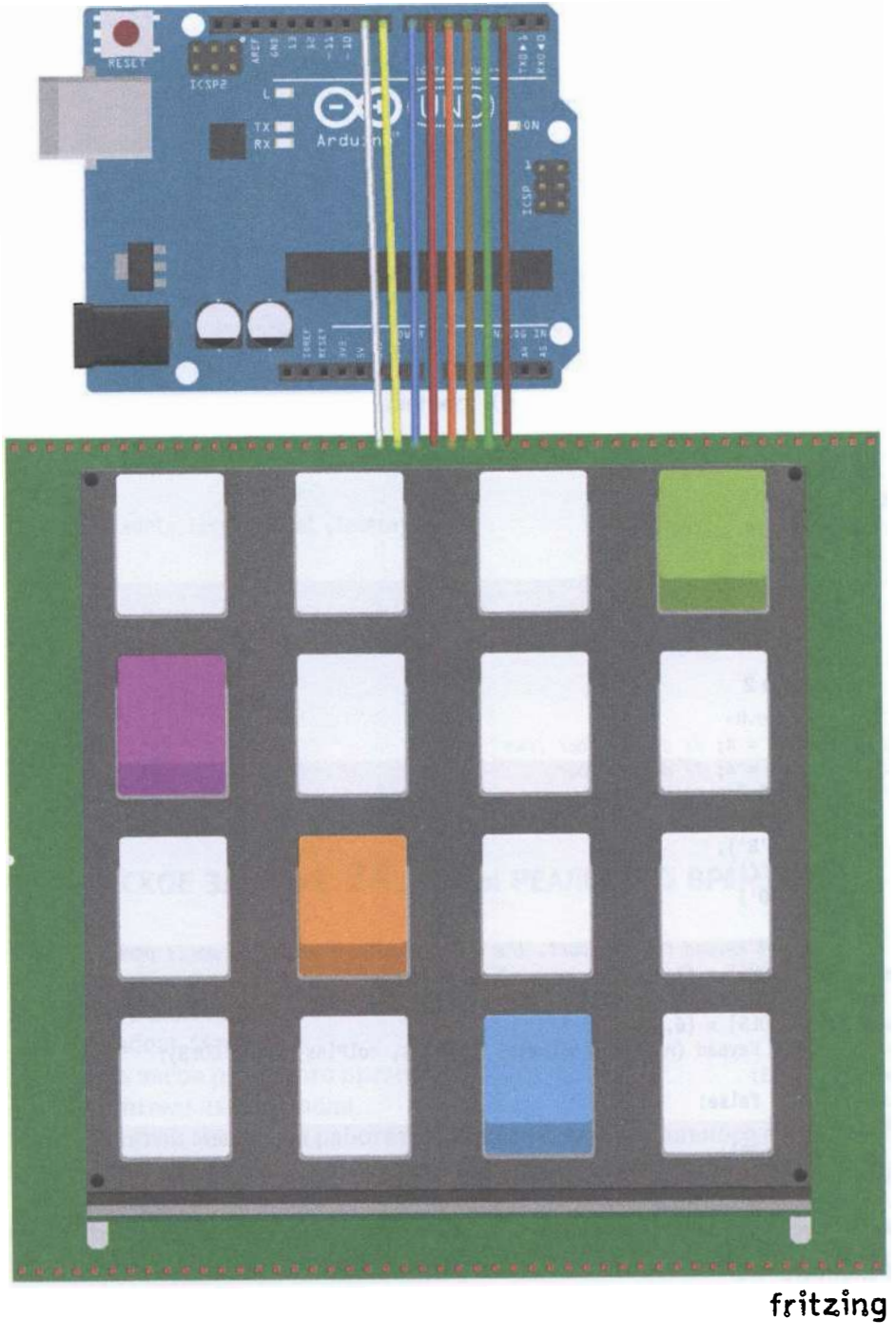


Рис. 88 ❖ Схема подключения с макетной платой для практического занятия 23

Код программы 1

```
#include <Keypad.h>
const byte ROWS = 4; // define four rows
const byte COLS = 4; // define four
char keys [ROWS] [COLS] = {
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}
};
// Connect 4 * 4 keypad row-bit port, the corresponding digital IO ports panel
byte rowPins [ROWS] = {2,3,4,5};
// Connect 4 * 4 buttons faithfully port, the corresponding digital IO ports panel
byte colPins [COLS] = {6,7,8,9};
// Call the function library function Keypad
Keypad keypad = Keypad (makeKeymap (keys), rowPins, colPins, ROWS, COLS);
void setup () {
Serial.begin (9600);
}
void loop () {
char key = keypad.getKey ();
if (key!= NO_KEY) {
Serial.println (key);
}
}
```

Код программы 2

```
#include <Keypad.h>
const byte ROWS = 4; // define four rows
const byte COLS = 4; // define four
char keys [ROWS] [COLS] = {
{'1', '2', '3','A'},
{'4', '5', '6','B'},
{'7', '8', '9','C'},
{'*', '0','#','D'}
};
// Connect 4 * 4 keypad row-bit port, the corresponding digital IO ports panel
byte rowPins [ROWS] = {2,3,4,5};
// Connect 4 * 4 buttons faithfully port, the corresponding digital IO ports panel
byte colPins [COLS] = {6,7,8,9};
Keypad keypad = Keypad (makeKeymap (keys), rowPins, colPins, ROWS, COLS);
byte ledPin = 13;
boolean blink = false;
void setup () {
Serial.begin (9600);
pinMode (ledPin, OUTPUT); // sets the digital pin as output
digitalWrite (ledPin, HIGH); // sets the LED on
keypad.addEventListener (keypadEvent); // add an event listener for this keypad
}
void loop () {
char key = keypad.getKey ();
if (key!= NO_KEY) {
Serial.println (key);
```

```
}
if (blink) {
digitalWrite (ledPin,! digitalRead (ledPin));
delay (100);
}
}
// Take care of some special events
void keypadEvent (KeypadEvent key) {
switch (keypad.getState ()) {
case PRESSED:
switch (key) {
case '#': digitalWrite (ledPin,!digitalRead (ledPin)); break;
case '*':
digitalWrite (ledPin,!digitalRead (ledPin));
break;
}
break;
case RELEASED:
switch (key) {
case '*':
digitalWrite (ledPin,!digitalRead (ledPin));
blink = false;
break;
}
break;
case HOLD:
switch (key) {
case '*': blink = true; break;
}
break;
}
}
```

Практическое занятие 24. Часы реального времени DS1307

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- модуль часов реального времени;
- соединительные провода.

В этом занятии мы будем работать с модулем часов реального времени (RTC module DS1307). Он поддерживает отображение года, месяца, дня, дня недели, часа, минуты и секунды. Нам понадобятся: консоль, или монитор порта (**Ctrl+Shift+M**), чтобы видеть время, выдаваемое модулем, и следующие библиотеки: stdio.h, string.h (они установлены по умолчанию) и DS1302.h, которую можно найти в [6]. Она вполне подходит для работы с модулем DS1307, хотя для него можно отдельно скачать библиотеку через **Скетч – Подключить библиотеку – Управлять библиотеками...** главного меню Adruino IDE и применить

код из примеров именно для него. Когда вы запустите программу, в консоль наверняка будет выводиться что-то типа Sunday 2000-1-1 1:11:17, т. е. часы немного отстали. Однако это можно исправить, отправив в текстовое поле корректные дату, время и день недели в следующем формате: год,месяц,день,час, минута,секунда,день недели, при этом день недели задаётся в виде числа, где воскресенье (Sunday) = 1, понедельник (Monday) = 2 и т. д. Например, можно ввести так: 2017,11,29,0,17,59,4 (откуда вы узнаете, когда я делал это задание). При этом после ввода вы увидите запись в консоли: You inputed:2017,11,29,0,17,59,4, после которой пойдёт отображение правильной даты и времени в соответствии с тем, что было введено: Wednesday 2017-11-29 0:17:59. Вы можете отключить на время модуль, а потом включить его снова и убедиться, что часы работают (потому что в них есть батарейка, рассчитанная на 9 лет работы). Такие же часы установлены на материнской плате любого компьютера. Обратите внимание на маркировку пинов на модуле с часов – она может отличаться от приведённой ниже схемы подключения!

Схема представлена на рис. 89–90.

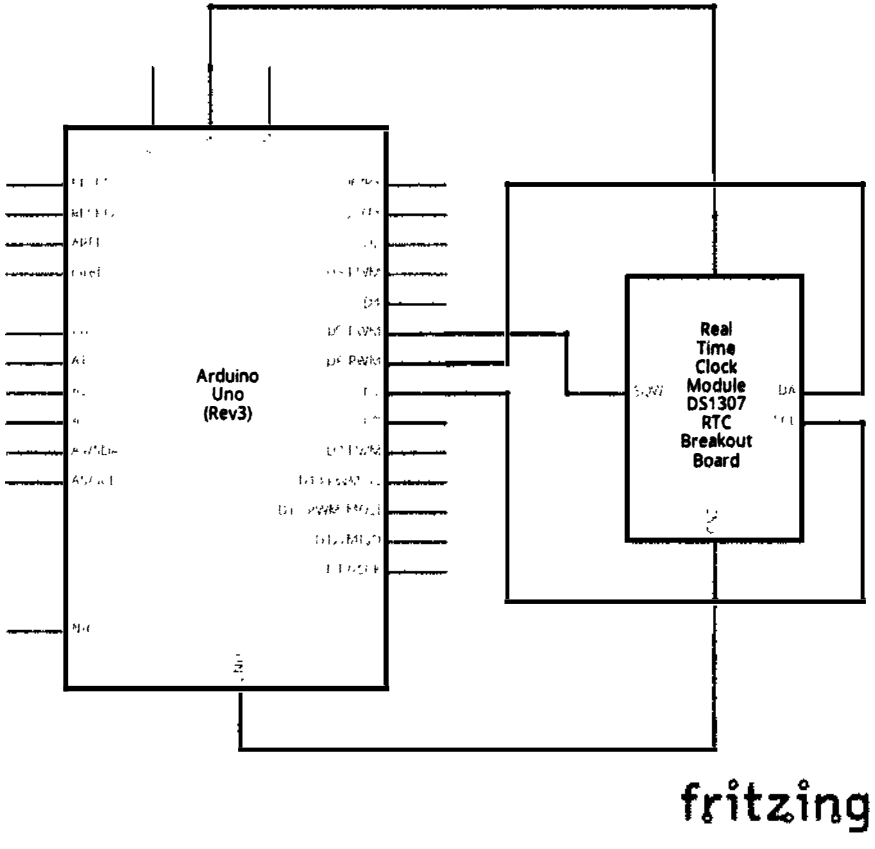


Рис. 89 ❖ Принципиальная схема подключения для практического занятия 24

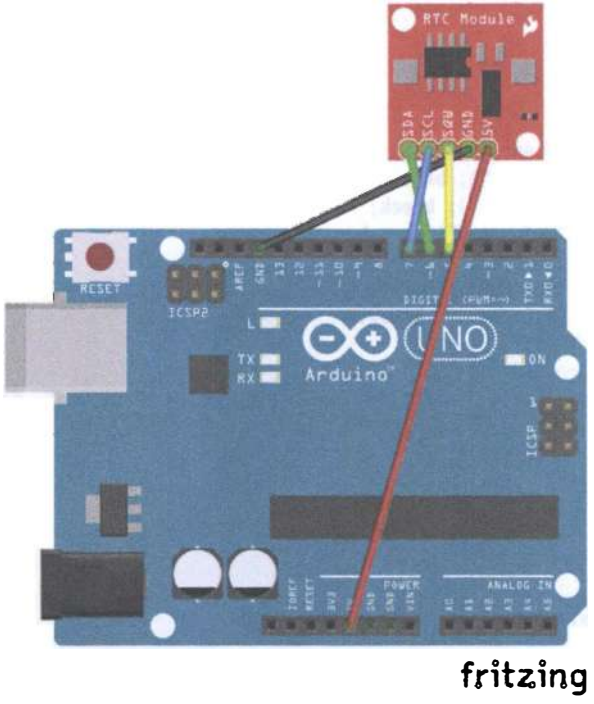


Рис. 90 ❖ Схема подключения для практического занятия 24

Код программы

```
#include <DS1302.h>
#include <stdio.h>
#include <string.h>

/* Interface Definition
CE (DS1302 pin5) -> Arduino D5
IO (DS1302 pin6) -> Arduino D6
SCLK (DS1302 pin7) -> Arduino D7
*/
uint8_t CE_PIN = 5;
uint8_t IO_PIN = 6;
uint8_t SCLK_PIN = 7;
/* Date variable buffer */
char buf [50];
char day [10];
/* Serial data cache */
String comdata = "";
int numdata [7] = {0}, j = 0, mark = 0;
/* Create DS1302 object */
DS1302 rtc (CE_PIN, IO_PIN, SCLK_PIN);
void print_time ()
{
/* Get the current time from the DS1302 */
```

```
Time t = rtc.time ();
/* The week from digital to name */
memset (day, 0, sizeof (day));
switch (t.day)
{
case 1: strcpy (day, "Sunday"); break;
case 2: strcpy (day, "Monday"); break;
case 3: strcpy (day, "Tuesday"); break;
case 4: strcpy (day, "Wednesday"); break;
case 5: strcpy (day, "Thursday"); break;
case 6: strcpy (day, "Friday"); break;
case 7: strcpy (day, "Saturday"); break;
}
/* The date code format to make up for output buf */
snprintf (buf, sizeof (buf), "% s% 04d-% 02d-% 02d% 02d:% 02d:% 02d", day, t.yr, t.mon,
t.date, t.hr, t. min, t.sec);
/* Output the date to the serial port */
Serial.println (buf);
}
void setup ()
{
Serial.begin (9600);
rtc.writeProtect (false);
rtc.halt (false);
}
void loop ()
{
/* When the serial port has data, the data is spliced into a variable comdata */
while(Serial.available()>0)
{
comdata+=char(Serial.read());
delay(2);
mark=1;
}
/* A comma-separated string comdata decomposition, the decomposition results become
converted into digital to numdata [] array */
if(mark==1)
{
Serial.print("You inputed:");
Serial.println(comdata);
for(int i=0;i<comdata.length();i++)
{
if(comdata[i]==','||comdata[i]==0x10||comdata[i]==0x13)
{
j++;
}
else
{
numdata[j]=numdata[j]*10+(comdata[i]-'0');
}
}
/* Make up the converted numdata time format, write DS1302 */
```

```
Time t(numdata[0],numdata[1],numdata[2],numdata[3],numdata[4],numdata[5],
numdata[6]);
rtc.time(t);
mark=0;j=0;
/* Empty comdata variable to wait for the next input */
comdata=String("");
/* Empty numdata */
for(int i=0;i<7;i++)numdata[i]=0;
}
/* Print the current time */
print_time();
delay(1000);
}
```

Практическое занятие 25. Эксперимент с датчиком уровня воды

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- датчик уровня воды;
- красный светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

Датчик уровня воды (Water Sensor) определяет количество (точнее, уровень) воды, гибок в применении, высокочувствителен и потребляет мало энергии, а также поддерживается многими контроллерами и платами расширения/разработчика. Однако в библиотеке Fritzing его почему-то нет, поэтому в схемах пришлось воспользоваться датчиком влажности почвы (Soil Moisture Sensor) из набора элементов (parts) SparkFun-Sensors, который обладает такими же тремя выходами и, в принципе, решает те же задачи. В этом задании мы будем зажигать светодиод, подключённый к цифровому выходу 12 платы Arduino Uno, если уровень воды превышает определённое пороговое значение (550), выдаваемое на аналоговый порт A1 платы. Кроме того, нам понадобится ёмкость с водой, куда мы будем погружать (**только не до конца!**) датчик уровня воды. Не бойтесь погрузить датчик в воду на несколько секунд, бойтесь погрузить его не тем концом, или тем, но – по уровень надписи Water Sensor и глубже, или бойтесь оставить его в воде на долгое время. И ещё нам надо открыть консоль (монитор порта), чтобы видеть показания датчика. Не забудьте вытереть датчик после выполнения задания!

Схема представлена на рис. 91–92.

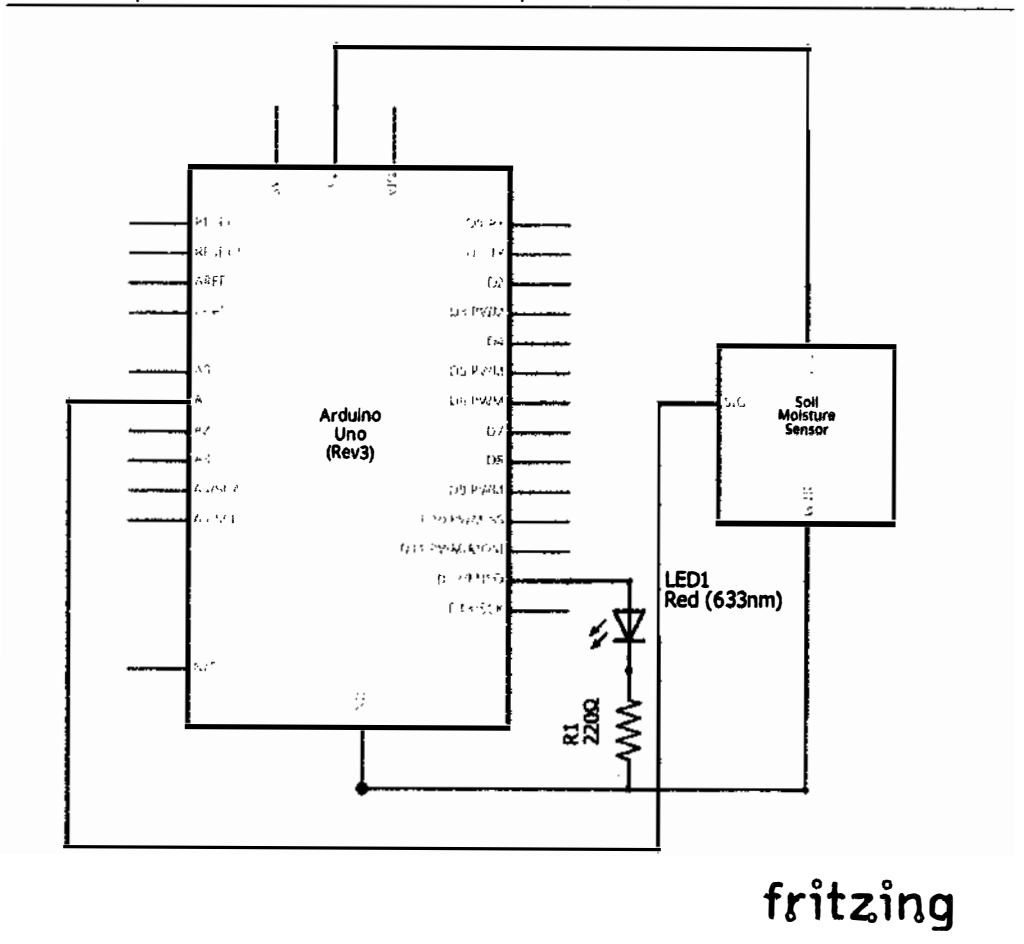


Рис. 91 ❖ Принципиальная схема подключения для практического занятия 25

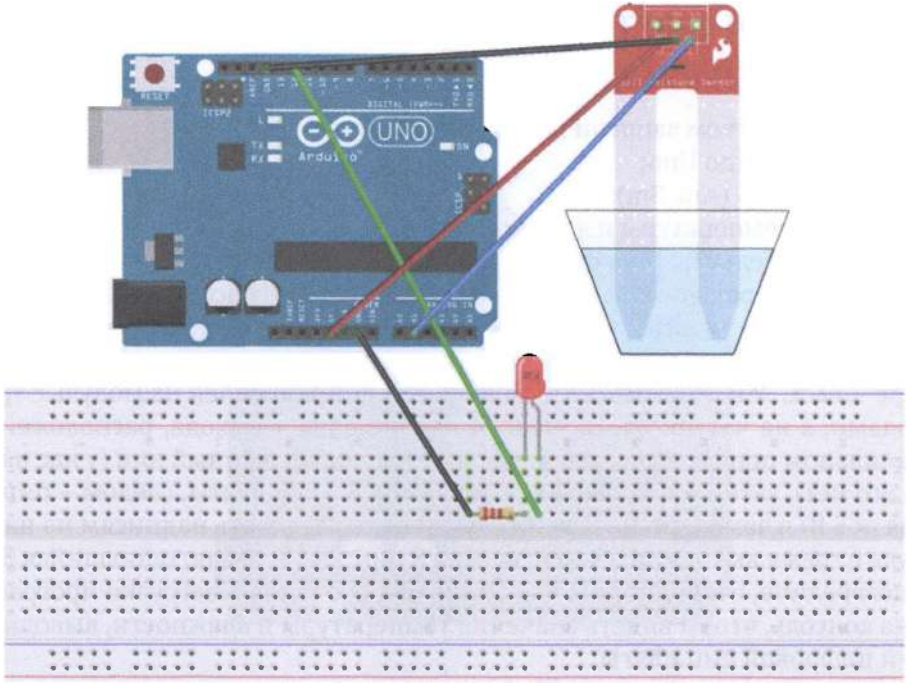


Рис. 92 ❖ Схема подключения с макетной платой для практического занятия 25

Код программы

```
int analogPin = 1; // level sensor connected to an analog port
int led = 12; // LED connected to digital port 12
int val = 0; // define the variable val initial value is 0
int data = 0; // define a variable data initial value is 0
void setup ()
{
pinMode (led, OUTPUT); // define led pin as an output
Serial.begin (9600); // set the baud rate to 9600
}
void loop ()
{
val = analogRead (analogPin); // read the analog value to give the variable val
if (val> 550) {// determine the variable val is greater than 700
digitalWrite (led, HIGH); // variable val is greater than 700, the light LED
}
else {
digitalWrite (led, LOW); // variable val is less than 700, the light goes off
}
data = val; // variable val assigned to the variable data
Serial.println (data); // serial print variable data
delay (100);
}
```

Практическое занятие 26. Эксперимент с сенсором температуры и влажности DHT11

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- сенсор температуры и влажности;
- соединительные провода.

Сенсор температуры и влажности гораздо меньше сенсора уровня воды. Он потребляет мало энергии и измеряет температуру в диапазоне от 0 до 50 °C, влажность – от 20 до 90 %. Точность измерений по влажности – ±5 %, по температуре – ±2 °C. У нас в комплекте сенсор DHT11 прикреплён на модуль с тремя выходами, а не четырьмя. На рис. 94 изображены 4 выхода, расположенных в правильном порядке: слева-направо идёт неподключённый пин (у нас на модуле его нет), затем – подключение сигнала к 8 пину платы, следом – пин питания (к 5 В) и земля. Но на всякий случай надо следовать надписям на нашем модуле с тремя выходами: (S – сигнал, на 8 пин; «–» – земля; оставшийся 3 выход посередине, очевидно, + (5 В)). После подключения и загрузки программы нужна консоль, чтобы видеть значения температуры и влажности, выводимые на 8-й цифровой пин платы.

Схема представлена на рис. 93–94.

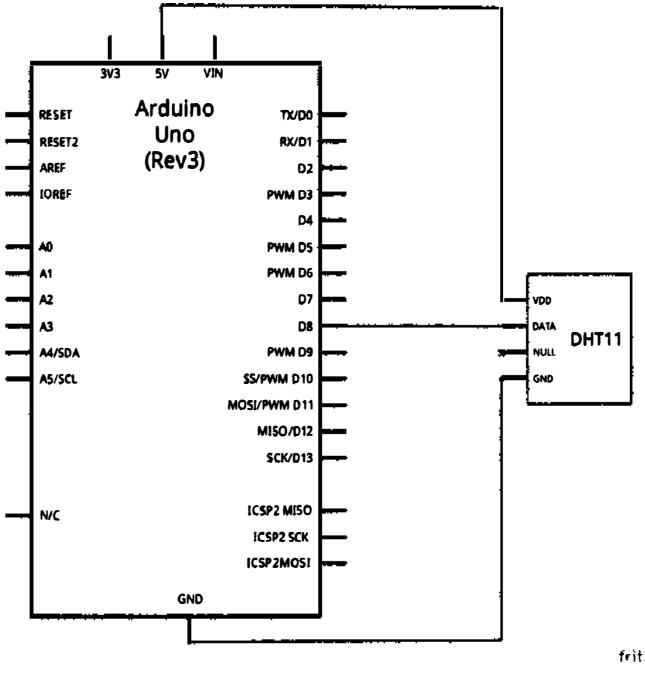


Рис. 93 ❖ Принципиальная схема подключения для практического занятия 26

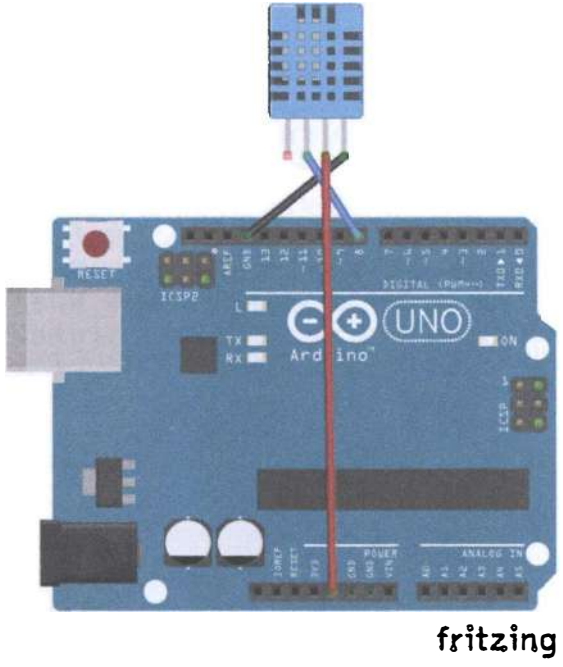


Рис. 94 ❖ Схема подключения для практического занятия 26

Код программы

```
int DHpin = 8;
byte dat [5];
byte read_data ()
{
byte data;
for (int i = 0; i <8; i++)
{
if (digitalRead (DHpin) == LOW)
{
while (digitalRead (DHpin) == LOW); // wait for 50us;
delayMicroseconds (30); // determine the duration of the high level to determine the data
is '0 'or
'1';
if (digitalRead (DHpin) == HIGH)
data |= (1 << (7-i)); // high front and low in the post;
while (digitalRead (DHpin) == HIGH); // data '1 ', wait for the next one receiver;
}
}
return data;
}
void start_test ()
{
```

```
digitalWrite (DHpin, LOW); // bus down, send start signal;
delay (30); // delay greater than 18ms, so DHT11 start signal can be detected;
digitalWrite (DHpin, HIGH);
delayMicroseconds (40); // Wait DHT11 response;
pinMode (DHpin, INPUT);
while (digitalRead (DHpin) == HIGH);
delayMicroseconds (80); // DHT11 a response, pulled the bus 80us;
if (digitalRead (DHpin) == LOW);
delayMicroseconds (80); // DHT11 80us after the bus pulled to start sending data;
for (int i = 0; i <4; i++) // receives temperature and humidity data, the parity bit is not
considered;
dat [i] = read_data ();
pinMode (DHpin, OUTPUT);
digitalWrite (DHpin, HIGH); // sending data once after releasing the bus, wait for the host
to open the next Start signal;
}
void setup ()
{
Serial.begin (9600);
pinMode (DHpin, OUTPUT);
}
void loop ()
{
start_test ();
Serial.print ("Current humdity =");
Serial.print (dat [0], DEC); // display the humidity-bit integer;
Serial.print ('.');
Serial.print (dat [1], DEC); // display the humidity decimal places;
Serial.println ('%');
Serial.print ("Current temperature =");
Serial.print (dat [2], DEC); // display the temperature of integer bits;
Serial.print ('.');
Serial.print (dat [3], DEC); // display the temperature of decimal places;
Serial.println ('C');
delay (700);
}
```

Практическое занятие 27. Эксперимент с релейным модулем

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- релейный модуль;
- синий светодиод;
- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

Релейный модуль используется для переключения электрической схемы при возникновении определённого условия, подающегося на управляющий выход модуля, например определённого сигнала (0 или 1), значения и т. д. Применение такого модуля весьма широкое – от игрушек до систем сигнализации. У релейного модуля всего три выхода, помеченных соответственно +, – и S (управляющий/сигнальный выход). В этом занятии при срабатывании релейного модуля мы будем зажигать светодиод, подключённый не к плате, а к релейному модулю вместе с резистором на 220 Ом. Для этого понадобится отвёртка, чтобы открутить два винта двух контактов релейного модуля. Код программы крошечный: каждую секунду мы то зажигаем светодиод, посылая на 10-й цифровой пин платы 1, то гасим, посылая 0. При этом можно услышать соответствующий звук переключения релейного модуля – не пугайтесь.

Схема представлена на рис. 95–96.

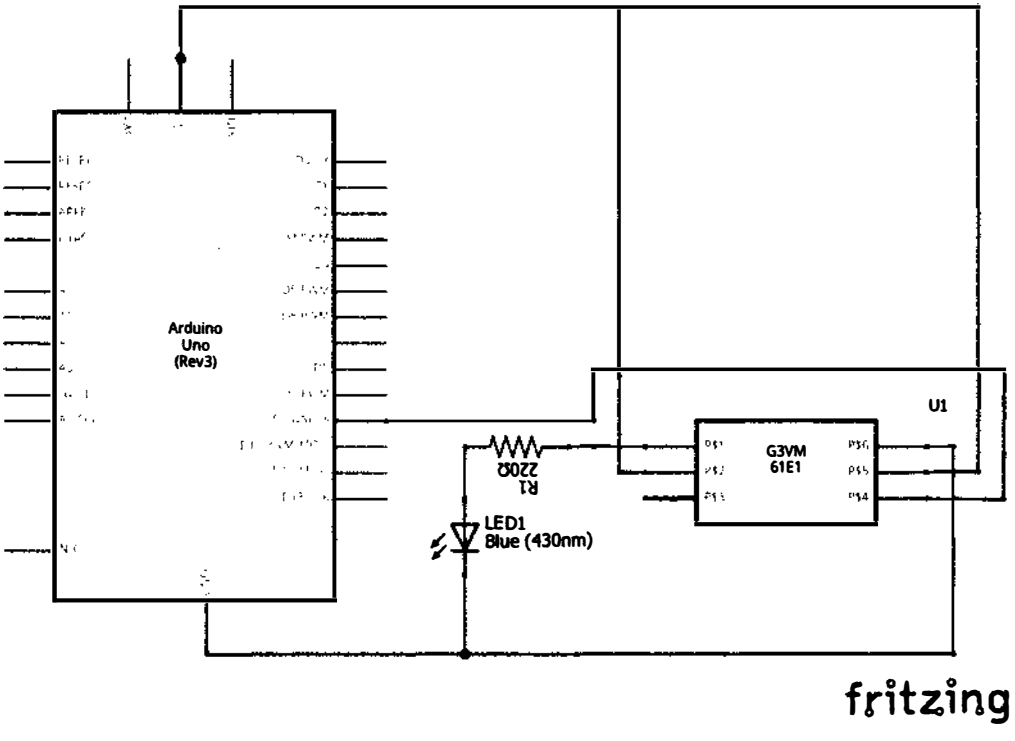


Рис. 95 ❖ Принципиальная схема подключения для практического занятия 27

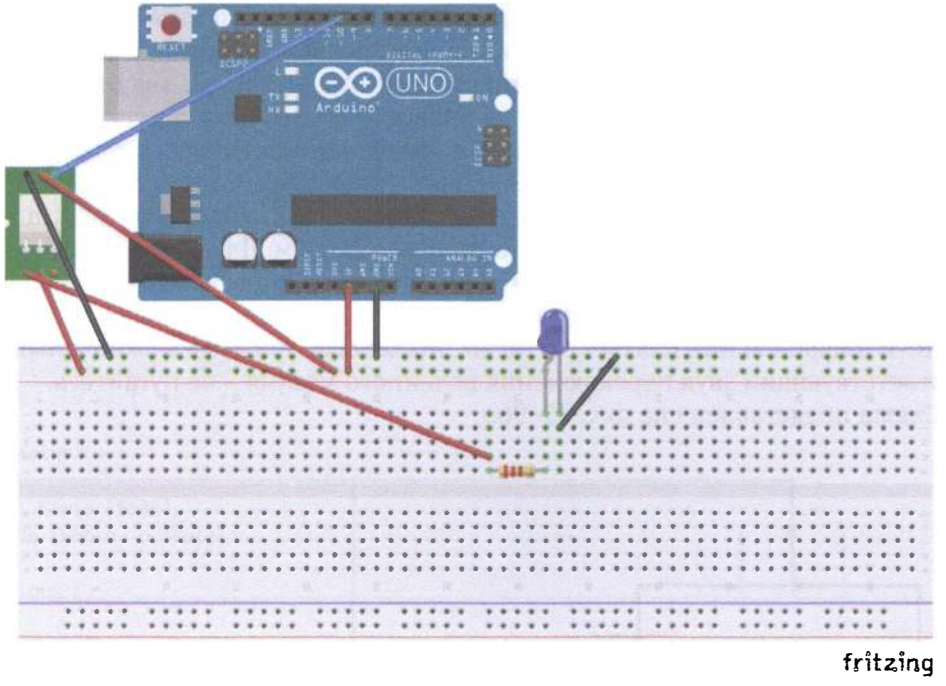


Рис. 96 ❖ Схема подключения с макетной платой для практического занятия 27

Код программы

```
int relay = 10; // relay turns trigger signal - active high;
void setup ()
{
pinMode (relay, OUTPUT); // Define port attribute is output;
}
void loop ()
{
digitalWrite (relay, HIGH); // relay conduction;
delay (1000);
digitalWrite (relay, LOW); // relay switch is turned off;
delay (1000);
}
```

Практическое занятие 28. Эксперимент с жидкокристаллическим монитором LCD1602A

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- жидкокристаллический монитор;
- потенциометр;
- 2 штырьковых коннектора по 8 пинов каждый;

- резистор на 220 Ом;
- макетная плата;
- соединительные провода.

Для работы с жидкокристаллическим монитором LCD1602A, у которого 16 выходов, нам понадобится библиотека LiquidCrystal.h, которая уже встроена в Arduino IDE. Этот мини-монитор может выводить 16 символов в каждой из 2 строк, работает от 5 В; с помощью потенциометра мы сможем регулировать яркость монитора. Фактически мы будем задействовать 12 пинов мини-монитора, поэтому предполагается, что к ним будут припаяны провода. Но можно обойтись и без этого с помощью макетной платы и штырьковых коннекторов двух типов – углового и прямого, каждый из которых содержит 8 пинов-выходов и вставляется через дырки-выходы LCD монитора в макетную плату, или сначала вставляется в макетную плату, а потом на него надеваются дырки вместе с монитором. С помощью представленного ниже кода выведем стандартную для подобных заданий фразу Hello, world! и начнём считать секунды. Обратите внимание, что контакт с монитором должен быть хороший всё время выполнения задания – сначала вы обеспечиваете хороший контакт с пинами монитора, потом подключаете плату к питанию и загружаете скетч. Можно обойтись без штырьковых коннекторов, если воткнуть провода через дырки пинов монитора в макетную плату, тем самым «пригвоздив» его к ней.

Схема представлена на рис. 97–98.

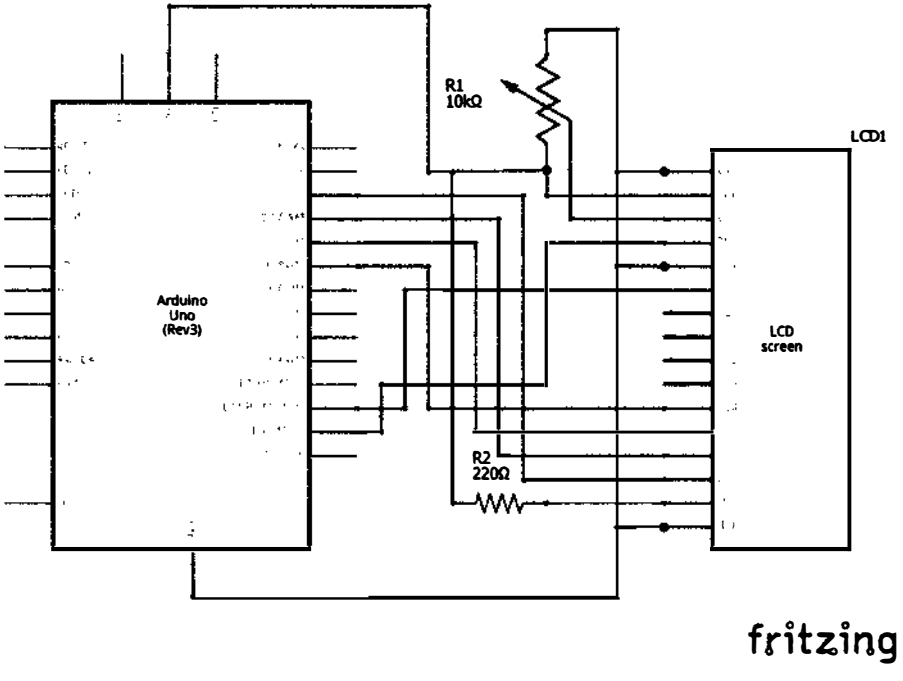


Рис. 97 ❖ Принципиальная схема подключения для практического занятия 28

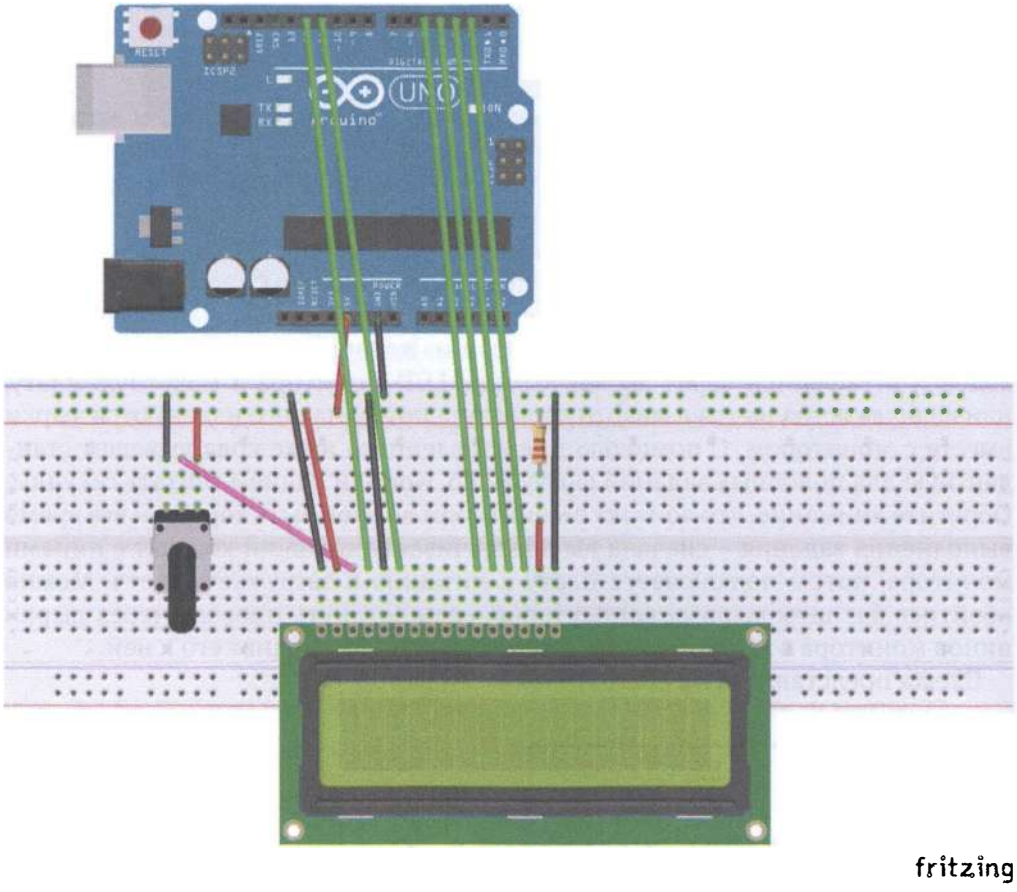


Рис. 98 ❖ Схема подключения с макетной платой для практического занятия 28

Код программы

```
// include the library code:
#include <LiquidCrystal.h>
// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
// set up the LCD's number of columns and rows:
lcd.begin(16, 2);
// Print a message to the LCD.
lcd.print("hello, world!");
}

void loop() {
// set the cursor to column 0, line 1
// (note: line 1 is the second row, since counting begins with 0):
```

```
lcd.setCursor(0, 1);
// print the number of seconds since reset:
lcd.print(millis() / 1000);
}
```

Практическое занятие 29. Эксперимент с шаговым двигателем

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- шаговый двигатель;
- модуль для шагового двигателя;
- потенциометр;
- макетная плата;
- соединительные провода.

Шаговый двигатель при получении импульсного сигнала поворачивает силовой привод (и рукоятку, прикреплённую к нему, или вообще любую деталь – actuator) на определённый угол (шаговый угол). Направление поворота, скорость, ускорение можно контролировать с помощью изменения частоты импульса или количества импульсов. Мы будем использовать четырёхфазный шаговый двигатель диаметром 28 мм с 5 выходами, который работает от 5 В и может поворачивать привод на угол 5,625*(1...64), т. е. на 5,625 градуса, 11,25 градуса и т. д. до 360 (т. е. вращаться до бесконечности). Также нам понадобится модуль ULN2003APG для шагового двигателя (подключается к земле и питанию 5 В соответствующими пинами – и +, а также к выходам 8–11 платы Arduino Uno с помощью пинов IN1–IN4), потенциометр для контроля скорости вращения привода (подключён к пину A0 платы) и библиотека Stepper.h, уже установленная в Arduino IDE. На схемах изображён немного другой шаговый двигатель, т. к. нашего в библиотеке Fritzing не нашлось.

Схема представлена на рис. 99–100.

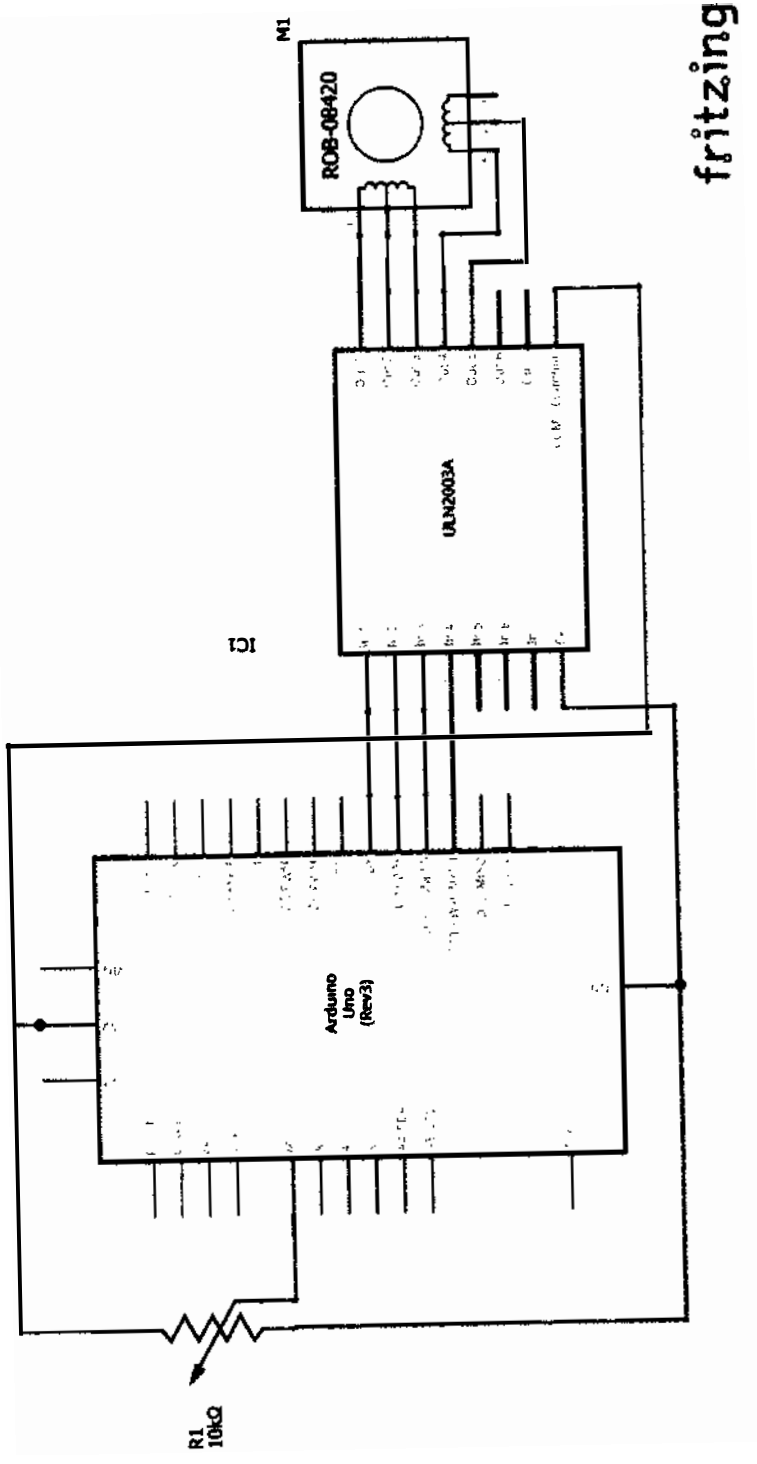


Рис. 99 ❖ Принципиальная схема подключения для практического занятия 29

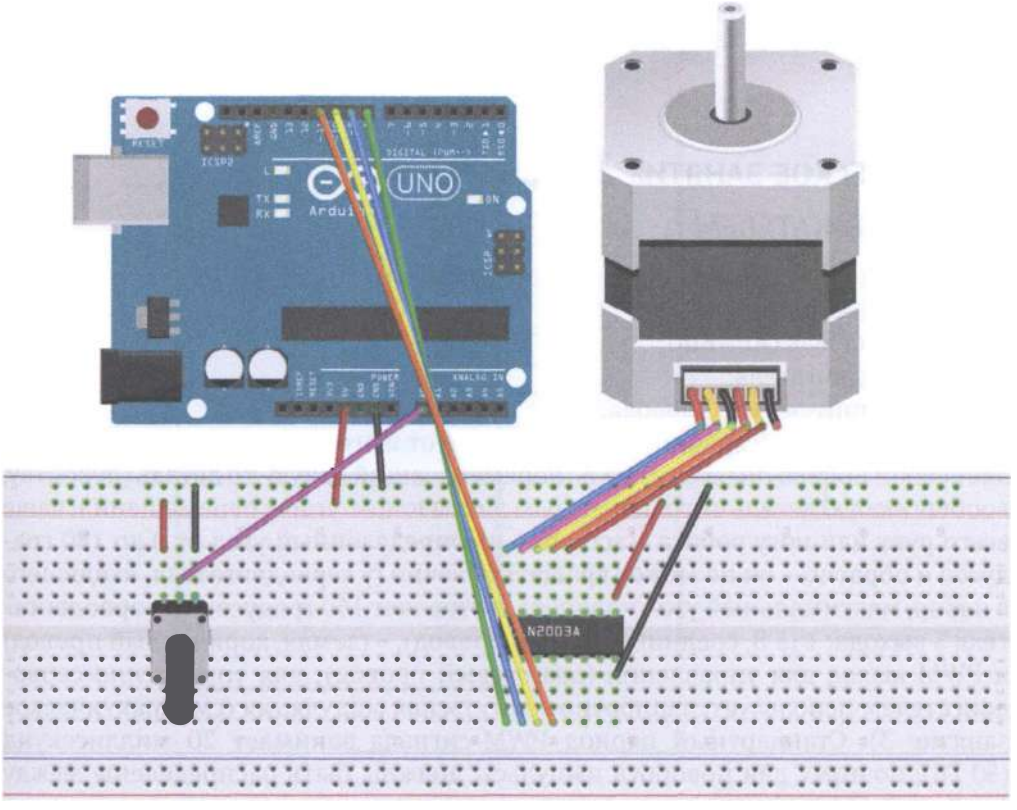


Рис. 100 ❖ Схема подключения с макетной платой для практического занятия 29

Код программы

```
#include <Stepper.h>
// Set here is the number of revolution of the stepper motor step
#define STEPS 100
// Attached to set the number of steps of the stepper motor and pin
Stepper stepper (STEPS, 8, 9, 10, 11);
// Define a variable to store the history of reading
int previous = 0;
void setup ()
{
// Set the motor speed of 90 per minute step
stepper.setSpeed (90);
}
void loop ()
{
// Get the sensor readings
int val = analogRead (0);
// Move to the current reading minus the number of steps historical readings
```

```
stepper.step (val - previous);
// Save history readings
previous = val;
}
```

Практическое занятие 30. Эксперимент с серводвигателем

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- серводвигатель;
- соединительные провода.

Серводвигатель (сервопривод), в отличие от шагового двигателя, представляет собой поворотный двигатель, перемещающий какую-то деталь (рукоятку, вообще любую деталь с ограниченным диапазоном/углом перемещения, чаще всего руку или ногу робота – actuator) на определённый угол от 0 до 180 градусов и обратно – он не может вращаться вечно (у серводвигателя, входящего в набор, максимальный угол поворота составляет 160 градусов). У серводвигателя 3 выхода: + (5 В, средний красный провод), – (земля, коричневый провод) и PWM-выход для управления (оранжевый провод); для управления серводвигателем используется широтно-импульсная модуляция (см. практическое занятие 3). Стандартный период PWM-сигнала занимает 20 миллисекунд (50 Гц), поэтому для поворота импульсы должны быть распределены между 1 и 2 миллисекундами, но на практике эти значения могут варьироваться от 0.5 до 2.5 миллисекунды для поворотов на углы от 0 до 180 градусов. Для поворота на 0 градусов (т. е. никакого поворота) на соответствующий PWM-пин платы подаётся сигнал 1000 (1000 микросекунд = 1 миллисекунда), на 45 градусов – 1250, на 180 градусов – 2000 микросекунд. В реальности эти значения могут варьироваться от 500 до 2480 микросекунд. Чтобы увидеть поворот, можно надеть самую большую насадку из маленького пакетика с 3 насадками и 3 шурупами на белую шестерню двигателя. Напишем программу, в которой через консоль будем вводить значения угла, на который мы хотим повернуть пластмассовую насадку, надетую на двигатель. Если вводить цифры от 1 до 9, то можно поворачивать насадку, соответственно, на 20 градусов, 40, 60 и т. д. (9 = 180 градусов; работать не должно, т. к. у серводвигателя, входящего в набор, максимальный угол поворота составляет 160 градусов). Потом напишем другую программу, использующую встроенную библиотеку Servo.h для управления серводвигателем.

Схема представлена на рис. 101–102.

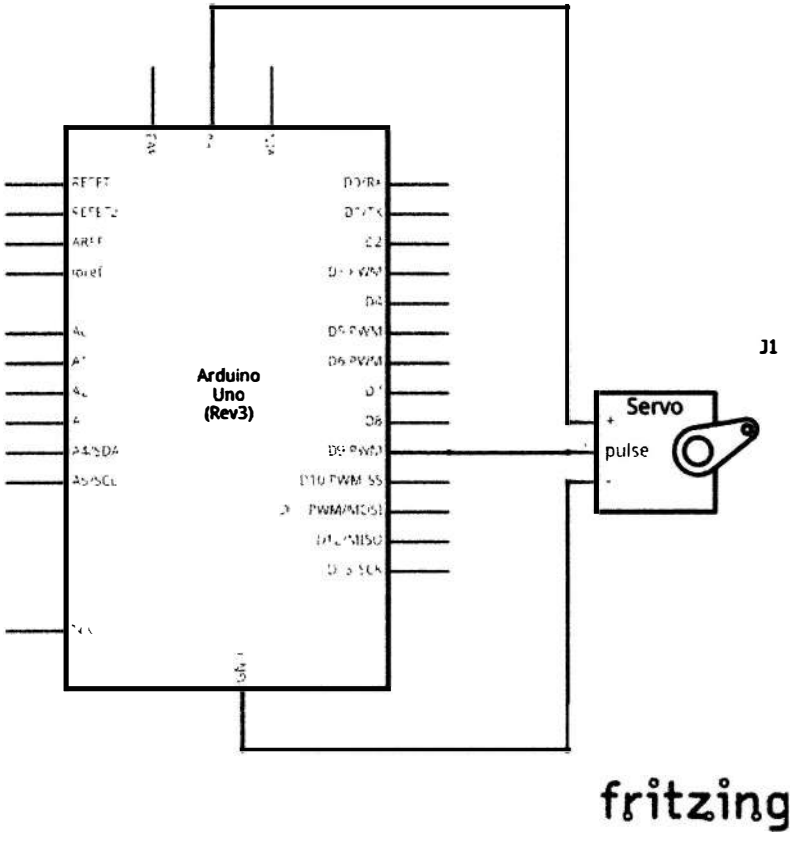


Рис. 101 ❖ Принципиальная схема подключения для практического занятия 30

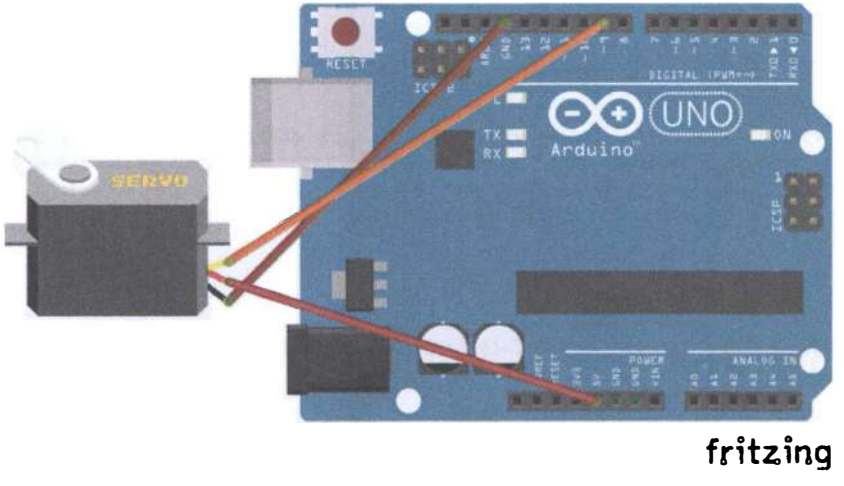


Рис. 102 ❖ Схема подключения для практического занятия 30

Код программы 1

```
int servopin = 9 ;// define the digital interface to connect the servo servo signal line 9
int myangle ;// define the angle variables
int pulsewidth ;// define variable pulse width
int val;
void servopulse (int servopin, int myangle) // define a pulse function
{
    pulsewidth = (myangle * 11) +500 ;// the angle value into a pulse width 500-2480
    digitalWrite (servopin, HIGH) ;// the servo interface level to high
    delayMicroseconds (pulsewidth) ;// number of microseconds delay pulse width value
    digitalWrite (servopin, LOW) ;// the servo interface level to Low
    delay (20-pulsewidth/1000);
}
void setup ()
{
    pinMode (servopin, OUTPUT) ;// set servo interface output interface
    Serial.begin (9600) ;// connect to the serial port, the baud rate is 9600
    Serial.println ("servo = o_seral_simple ready");
}
void loop () // will be 0-9's 0-180 number into perspective, and let the number of times
the  corresponding LED flashes
{
    val = Serial.read () ;// read the value of the serial port
    if (val> '0'&& val <= '9')
    {
        val = val-'0'; // will be converted to numeric variables characteristic quantities
        val = val * (180/9) ;// will figure into perspective
        Serial.print ("moving servo to");
        Serial.print (val, DEC);
        Serial.println ();
        for (int i = 0; i <= 50; i++) // give enough time to let it go to the steering
angle specified
        {
            servopulse (servopin, val) ;// reference pulse function
        }
    }
}
```

Код программы 2

```
#include <Servo.h> // define header files, there is one thing to note, you can directly
click on themenu bar at the Arduino software Sketch> Importlibrary> Servo, Servo function
call, you can also directly enter the # include <Servo.h>, But at the input to the
attention of the # include and <Servo.h> should be a space between, otherwise a compile-
time error.
Servo myservo ;// define a variable name servos
void setup ()
{
myservo.attach (9) ;// define Servo Interface (9,10 all OK, shortcomings can only control
2)
}
void loop ()
{
myservo.write (90) ;// set the steering angle of rotation
}
```

Практическое занятие 31. Эксперимент с игровым джойстиком

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- джойстик;
- соединительные провода.

Игровой джойстик позволяет контролировать значения по осям *x* и *y* и нажатия кнопки; при этом значения по *x* и по *y* отсылаются на аналоговые порты платы (например, A0 и A1 соответственно), а нажатия кнопки – на цифровой порт (например, 7). У джойстика 5 выходов, но они подписаны, так что проблем возникнуть не должно, если учесть, что SW – это определение нажатия кнопки (подаём на 7-й цифровой пин платы). Также нам понадобится консоль (монитор порта, **Ctrl+Shift+M**), чтобы видеть выводимые значения при выполнении программы 1. Вторая программа просто немного по-другому делает то же самое.

Схема представлена на рис. 103–104.

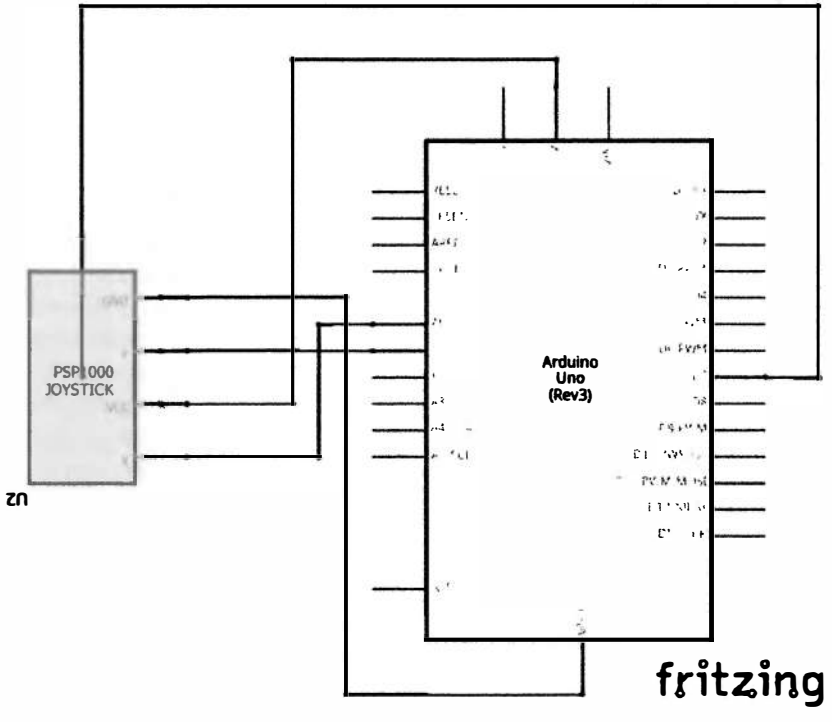


Рис. 103 ❖ Принципиальная схема подключения для практического занятия 31

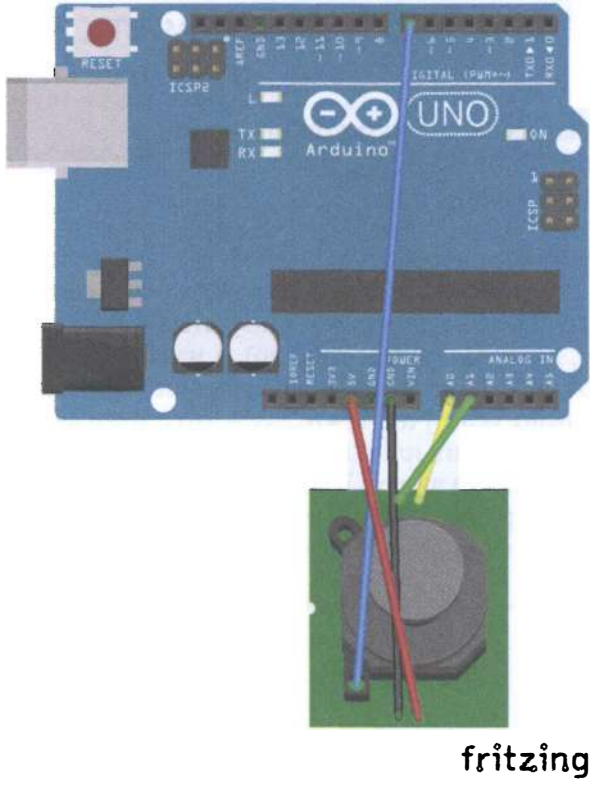


Рис. 104 ❖ Схема подключения для практического занятия 31

Код программы 1

```
int value = 0;
void setup () {
    pinMode (7, INPUT);
    Serial.begin (9600);
}
void loop () {
    value = analogRead (0);
    Serial.print ("X:");
    Serial.print (value, DEC);
    value = analogRead (1);
    Serial.print (" | Y:");
    Serial.print (value, DEC);
    value = digitalRead (7);
    Serial.print (" | Z:");
    Serial.println (value, DEC);
    delay (100);
}
```

Код программы 2

```
int JoyStick_X = 0; // x
int JoyStick_Y = 1; // y
int JoyStick_Z = 7; // key
void setup ()
{
    pinMode (JoyStick_X, INPUT);
    pinMode (JoyStick_Y, INPUT);
    pinMode (JoyStick_Z, INPUT);
    Serial.begin (9600); // 9600 bps
}
void loop ()
{
    int x, y, z;
    x = analogRead (JoyStick_X);
    y = analogRead (JoyStick_Y);
    z = digitalRead (JoyStick_Z);
    Serial.print (x, DEC);
    Serial.print (",");
    Serial.print (y, DEC);
    Serial.print (",");
    Serial.println (z, DEC);
    delay (100);
}
```

Практическое занятие 32. Эксперимент с инфракрасным пультом дистанционного управления

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- инфракрасный пульт дистанционного управления;
- инфракрасный приёмник;
- 6 светодиодов;
- 6 резисторов на 220 Ом;
- макетная плата;
- соединительные провода.

Сигналы, поступающие с инфракрасного пульта дистанционного управления (ИК-пульта ДУ), представляют собой двоичный импульсный код. Чтобы передать без проводов сигнал в ИК-диапазоне, сначала двоичный код проходит модуляцию на определённой частоте, и затем результат высылается ИК-светодиодами в направлении ИК-приёмника, который демодулирует полученный сигнал и превращает его обратно в двоичный код. Оптический сигнал, передаваемый в ИК-диапазоне ИК-светодиодами, преобразуется в слабый электрический сигнал, который затем усиливается, пропускается через фильтр, демодулятор и, таким образом, восстанавливается в исходный сигнал

в двоичном коде. ИК-приёмник сигнала располагает 3 пинами: + (5 В), – (земля) и VOUT для приёма аналогового оптического сигнала. Протокол передачи данных, по которому исходный сигнал кодируется ИК-пультом ДУ и затем декодируется ИК-приёмником сигнала, – это NEC-протокол: он использует 8 адресных битов и 8 битов для команды; адресные и командные биты пересылаются дважды, чтобы убедиться в их правильности; используется импульсно-позиционная модуляция; частота сигнала равна 38 КГц; логическая единица длится 2,25 миллисекунды, в которых первые 0,56 миллисекунды уровень сигнала 1, остальные – 0; логический ноль длится 1,12 миллисекунды, где опять первые 0,56 миллисекунды уровень сигнала равен 1, а остальные – 0. При подаче сигнала первые 9 миллисекунд сигнал равен 1, а затем следует 0 уровень, длящийся 4,5 миллисекунды. Такая комбинация означает, что дальше будет передаваться, собственно, основная информация (адрес + адрес + команда + команда для каждого сигнала). Если нажать на кнопку пульта и не отпускать её, вся подготовка и пересылка команды занимает 110 миллисекунд, которые затем повторяются каждые 110 миллисекунд с помощью повторяющего импульса (9 миллисекунд подаём 1, и 2,25 миллисекунды – 0), если кнопка остаётся нажатой. Для выполнения этого задания нам понадобится библиотека IRemote.h – придётся её установить из [7], т. к. встроенная библиотека немного другая – для роботов; также нам понадобится консоль (монитор порта). Программа, которую мы напишем, будет зажигать светодиоды и выводить в консоль информацию о том, что сигнал распознан (это будет NEC-сигнал, но можно попробовать и другой ИК-пульт ДУ: у меня получилось распознать сигнал с пульта от телевизора SONY). Светодиоды будут зажигаться в соответствии с нажатыми на пульте кнопками: первые 6 кнопок сверху (двух верхних рядов) зажигают светодиоды, вторые 6 кнопок сверху (ряд 3 и 4) гасят светодиоды. Прежде чем нажимать на кнопки пульта, не забудьте вынуть блокирующую контакты батарейки полоску из него! И вставить её обратно после завершения выполнения задания. А теперь самое интересное (для тех, кто дочитал до конца): у ИК-приёмника 3 выхода, и подключать их надо следующим образом: если ИК-приёмник повёрнут к вам «лицом» (на котором «маска» крест-накрест), то справа от вас находится пин питания, слева от вас – пин OUT, который подаётся к 11-му цифровому пину платы, а посередине у ИК-приёмника – земля.

Схема представлена на рис. 105–106.

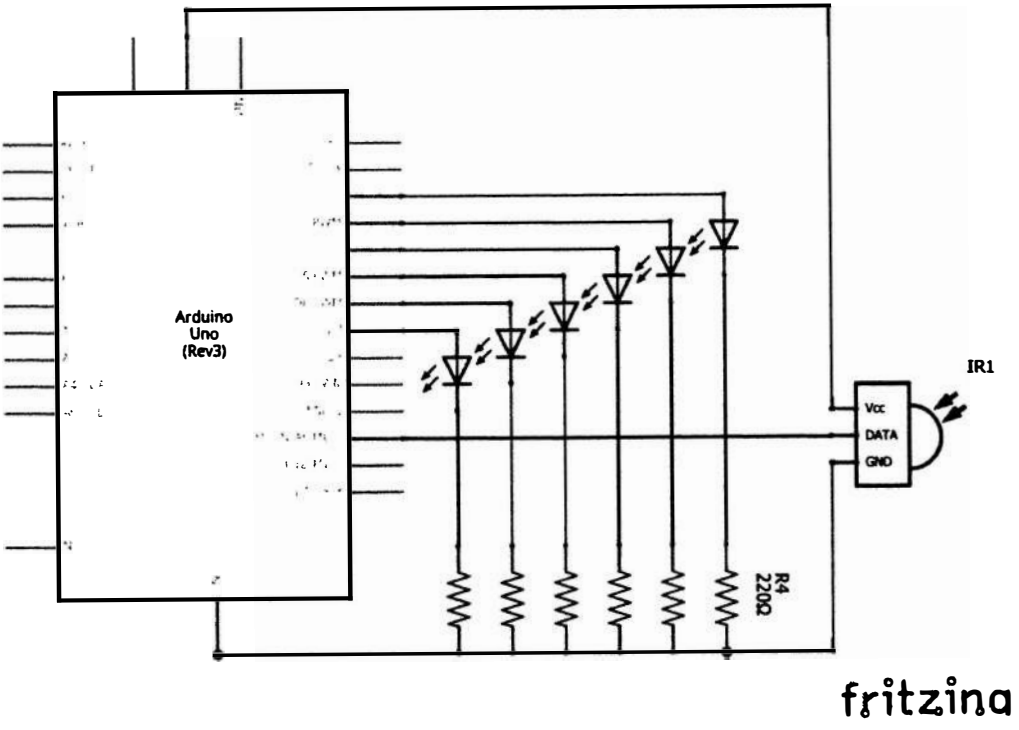


Рис. 105 ❖ Принципиальная схема подключения для практического занятия 32

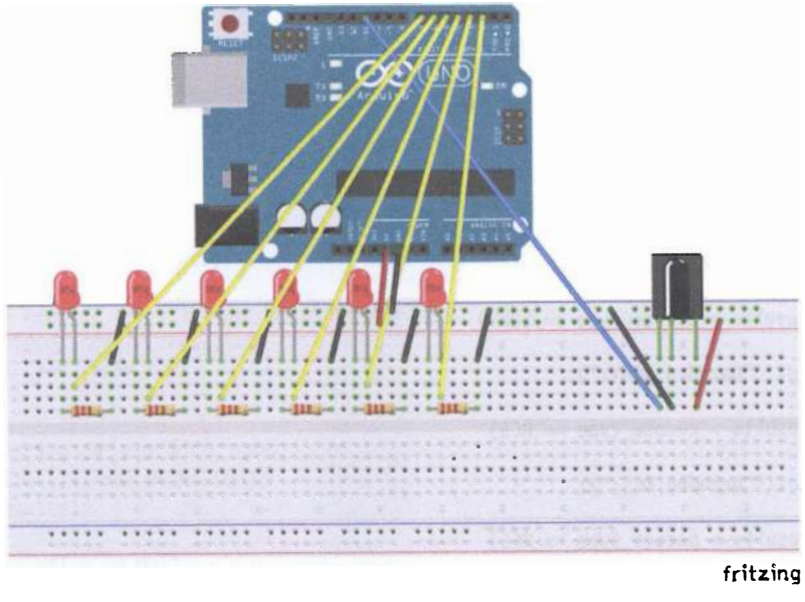


Рис. 106 ❖ Схема подключения с макетной платой для практического занятия 32

Код программы

```
#include <IRremote.h>
int RECV_PIN = 11;
int LED1 = 2;
int LED2 = 3;
int LED3 = 4;
int LED4 = 5;
int LED5 = 6;
int LED6 = 7;
long on1 = 0x00FFA25D;
long off1 = 0x00FFE01F;
long on2 = 0x00FF629D;
long off2 = 0x00FFA857;
long on3 = 0x00FFE21D;
long off3 = 0x00FF906F;
long on4 = 0x00FF22DD;
long off4 = 0x00FF6897;
long on5 = 0x00FF02FD;
long off5 = 0x00FF9867;
long on6 = 0x00FFC23D;
long off6 = 0x00FFB04F;
IRrecv irrecv (RECV_PIN);
decode_results results;
// Dumps out the decode_results structure.
// Call this after IRrecv :: decode ()
// Void * to work around compiler issue
// Void dump (void * v) {
// Decode_results * results = (decode_results *) v
void dump (decode_results * results) {
int count = results-> rawlen;
if (results-> decode_type == UNKNOWN)
{
Serial.println ("Could not decode message");
}
else
{
if (results-> decode_type == NEC)
{
Serial.print ("Decoded NEC:");
}
else if (results-> decode_type == SONY)
{
Serial.print ("Decoded SONY:");
}
else if (results-> decode_type == RC5)
{
Serial.print ("Decoded RC5:");
}
else if (results-> decode_type == RC6)
{
Serial.print ("Decoded RC6:");
}
```

```
Serial.print (results-> value, HEX);
Serial.print ("(");
Serial.print (results-> bits, DEC);
Serial.println ("bits)");
}
Serial.print ("Raw (");
Serial.print (count, DEC);
Serial.print ("):");
for (int i = 0; i <count; i++)
{
if ((i%2) == 1) {
Serial.print (results-> rawbuf [i] * USECPERTICK, DEC);
}
else
{
Serial.print (- (int) results-> rawbuf [i] * USECPERTICK, DEC);
}
Serial.print ("");
}
Serial.println ("");
}
void setup ()
{
pinMode (RECV_PIN, INPUT);
pinMode (LED1, OUTPUT);
pinMode (LED2, OUTPUT);
pinMode (LED3, OUTPUT);
pinMode (LED4, OUTPUT);
pinMode (LED5, OUTPUT);
pinMode (LED6, OUTPUT);
pinMode (13, OUTPUT);
Serial.begin (9600);
irrecv.enableIRIn (); // Start the receiver
}
int on = 0;
unsigned long last = millis ();
void loop ()
{
if (irrecv.decode (& results))
{
// If it's been at least 1/4 second since the last
// IR received, toggle the relay
if (millis () - last> 250)
{
on =! on;
// DigitalWrite (8, on? HIGH: LOW);
digitalWrite (13, on? HIGH: LOW);
dump (& results);
}
if (results.value == on1)
digitalWrite (LED1, HIGH);
if (results.value == off1)
```

```
digitalWrite (LED1, LOW);
if (results.value == on2)
digitalWrite (LED2, HIGH);
if (results.value == off2)
digitalWrite (LED2, LOW);
if (results.value == on3)
digitalWrite (LED3, HIGH);
if (results.value == off3)
digitalWrite (LED3, LOW);
if (results.value == on4)
digitalWrite (LED4, HIGH);
if (results.value == off4)
digitalWrite (LED4, LOW);
if (results.value == on5)
digitalWrite (LED5, HIGH);
if (results.value == off5)
digitalWrite (LED5, LOW);
if (results.value == on6)
digitalWrite (LED6, HIGH);
if (results.value == off6)
digitalWrite (LED6, LOW);
last = millis ();
irrecv.resume (); // Receive the next value
}
}
```

Практическое занятие 33. Эксперимент с RFID-модулем RC522

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- RFID-модуль;
- RFID-ключ;
- RFID-карта;
- штырьковый коннектор на 8 пинов;
- макетная плата;
- соединительные провода.

RFID – Radio Frequency IDentification, т. е. идентификация по радиочастоте. Принцип действия схож с ИК-пультом ДУ и ИК-приёмником, только здесь используется радиосигнал, а не оптический сигнал. Тоже есть источник сигнала (у нас это будет карточка, также он называется тег, метка, носитель сигнала, транспондер) и приёмник – RFID-модуль (считыватель, ридер, сканер). **Для RFID-модуля надо использовать питание 3.3 В, иначе он сгорит.** Также в этом занятии нам понадобятся RFID-ключ и RFID-карта – это 2 отдельных носителя сигнала, или метки, а ещё – библиотека SPI.h, которая уже встроена в Arduino IDE, библиотека MFRC522.h, которую можно скачать и установить из

[8], и консоль, в которой мы будем, поднося к RFID-модулю сначала карточку, потом ключ, считывать данные, записанные на них. Карту надо удерживать у модуля секунд 10, чтобы прочитались все данные! На схемах ниже представлен другой элемент – RFID USB Reader из набора SparkFun библиотеки Fritzing, но главное, что у него 8 выходов, и последовательность подключения сохранена (лучше смотреть цветную схему с макетной платой). У нашего RFID-модуля RC522 тоже 8 выходов, и подключаются они следующим образом: 3,3V – 3,3V, RST – D9 (или просто 9 – здесь имеются в виду только цифровые выходы платы), GND – GND, MISO – 12, MOSI – 11, SCK – 13, SDA – 10. Пропускаем мы только выход прерываний IRQ – он не подключён ни к чему. Обратите внимание, что контакт с RFID-модулем должен быть хороший всё время выполнения задания – сначала вы обеспечиваете хороший контакт с пинами модуля, потом подключаете плату к питанию и загружаете скетч. Можно обойтись без штырьковых коннекторов, если воткнуть провода через дырки пинов RFID-модуля в макетную плату.

Схема представлена на рис. 107–108.

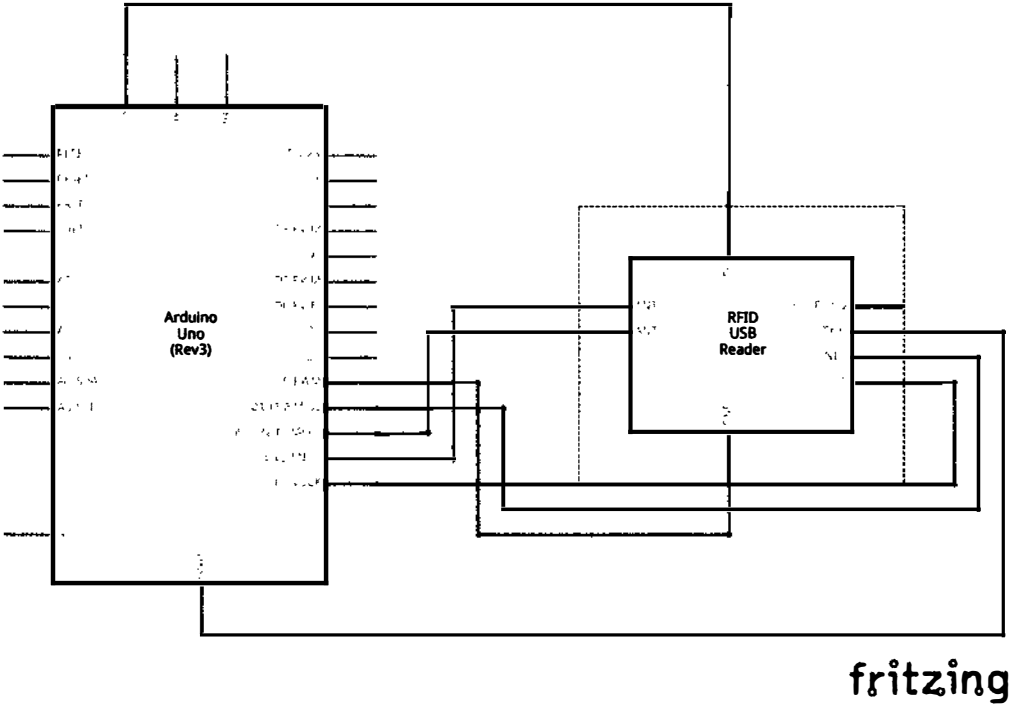


Рис. 107 ❖ Принципиальная схема подключения для практического занятия 33

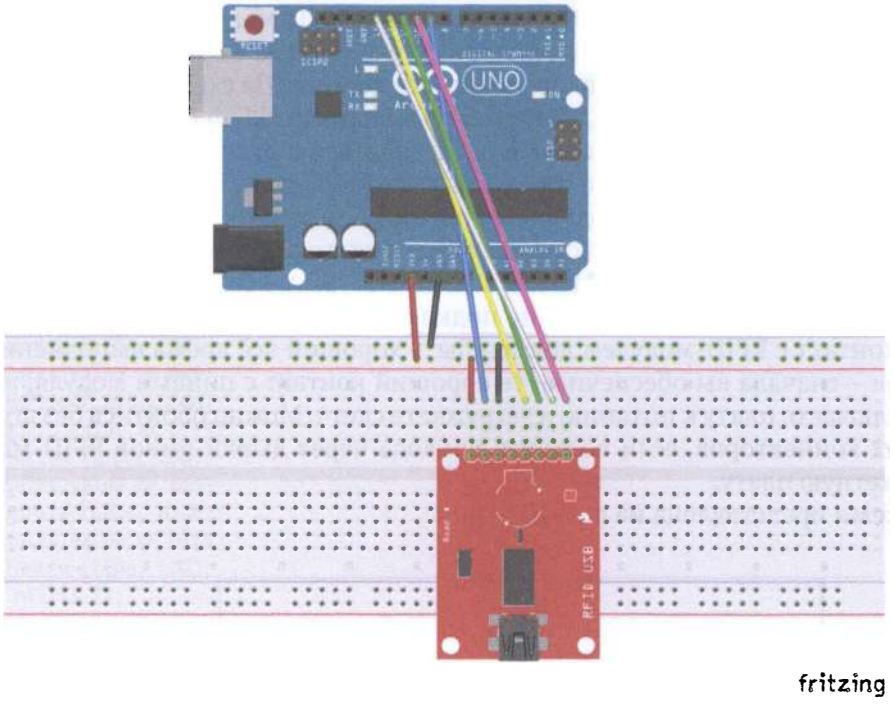


Рис. 108 ❖ Схема подключения с макетной платой для практического занятия 33

Код программы

```
#include <SPI.h>
#include <MFRC522.h>

#define RST_PIN       9     //
#define SS_PIN    10    //

MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance

void setup() {
Serial.begin(9600);       // Initialize serial communications with the PC
while (!Serial);      // Do nothing if no serial port is opened (added for Arduinos based
on ATMEGA32U4)
SPI.begin();          // Init SPI bus
mfrc522.PCD_Init();       // Init MFRC522
ShowReaderDetails();   // Show details of PCD - MFRC522 Card Reader details
Serial.println(F("Scan PICC to see UID, type, and data blocks..."));
}

void loop() {
// Look for new cards
if ( ! mfrc522.PICC_IsNewCardPresent()) {
return;
}

// Select one of the cards
if ( ! mfrc522.PICC_ReadCardSerial()) {
```

```
return;
}

// Dump debug info about the card; PICC_HaltA() is automatically called
mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}

void ShowReaderDetails() {
// Get the MFRC522 software version
byte v = mfrc522.PCD_ReadRegister(mfrc522.VersionReg);
Serial.print(F("MFRC522 Software Version: 0x"));
Serial.print(v, HEX);
if (v == 0x91)
Serial.print(F(" = v1.0"));
else if (v == 0x92)
Serial.print(F(" = v2.0"));
else
Serial.print(F(" (unknown)"));
Serial.println("");
// When 0x00 or 0xFF is returned, communication probably failed
if ((v == 0x00) || (v == 0xFF)) {
Serial.println(F("WARNING: Communication failure, is the MFRC522 properly connected?"));
}
}
```

Практическое занятие 34. Эксперимент с системой контроля доступа

В этом практическом занятии нам **понадобятся**:

- плата Arduino Uno;
- USB-кабель (Am-Bm);
- RFID-модуль;
- RFID-ключ;
- RFID-карта;
- штырьковый коннектор на 8 пинов;
- релейный модуль;
- 2 светодиода – красный и синий (зелёный);
- 2 резистора на 220 Ом;
- макетная плата;
- соединительные провода.

В этом занятии нам понадобится опыт предыдущего занятия и 27-го занятия, где мы работали с релейным модулем. Суть задачи в том, чтобы собрать схему с RFID-модулем и релейным модулем, в которой будут 2 светодиода – красный и синий (или зелёный, но зелёных у нас в наборе нет), и затем написать программу, которая при поднесении RFID-карты с правильным записанным на неё паролем к RFID-модулю будет переключать релейный модуль и включать синий светодиод, сигнализируя о том, что доступ открыт (access granted), в противном случае (неправильный пароль, записанный на RFID-ключ) будет гореть красный светодиод (доступ закрыт, access denied). **Не забудьте, что**

RFID-модуль подключается к 3.3 В, а релейный модуль – к 5 В! Сначала мы должны запустить код в программе 1 (SetPassword), открыть консоль (монитор порта, **Ctrl+Shift+M**) и поднести RFID-карту к RFID-модулю, чтобы изменить пароль; при этом мы увидим сообщение «The password has been changed!» среди бесчисленных сообщений «Error!» в консоли. RFID-ключ мы не подносим, т. е. пароль на нём не меняется на правильный. Таким образом, на карте у нас записан правильный пароль, на ключе – неправильный. Затем мы запускаем код программы 2 (DoorCon) и подносим RFID-карту – релейный модуль переключается, и на 5 секунд должен загореться синий светодиод, сигнализирующий о том, что на карте пароль правильный; при этом красный светодиод гаснет на те же 5 секунд. Затем подносим RFID-ключ и видим, что ничего не происходит.

Схема представлена на рис. 109–110.

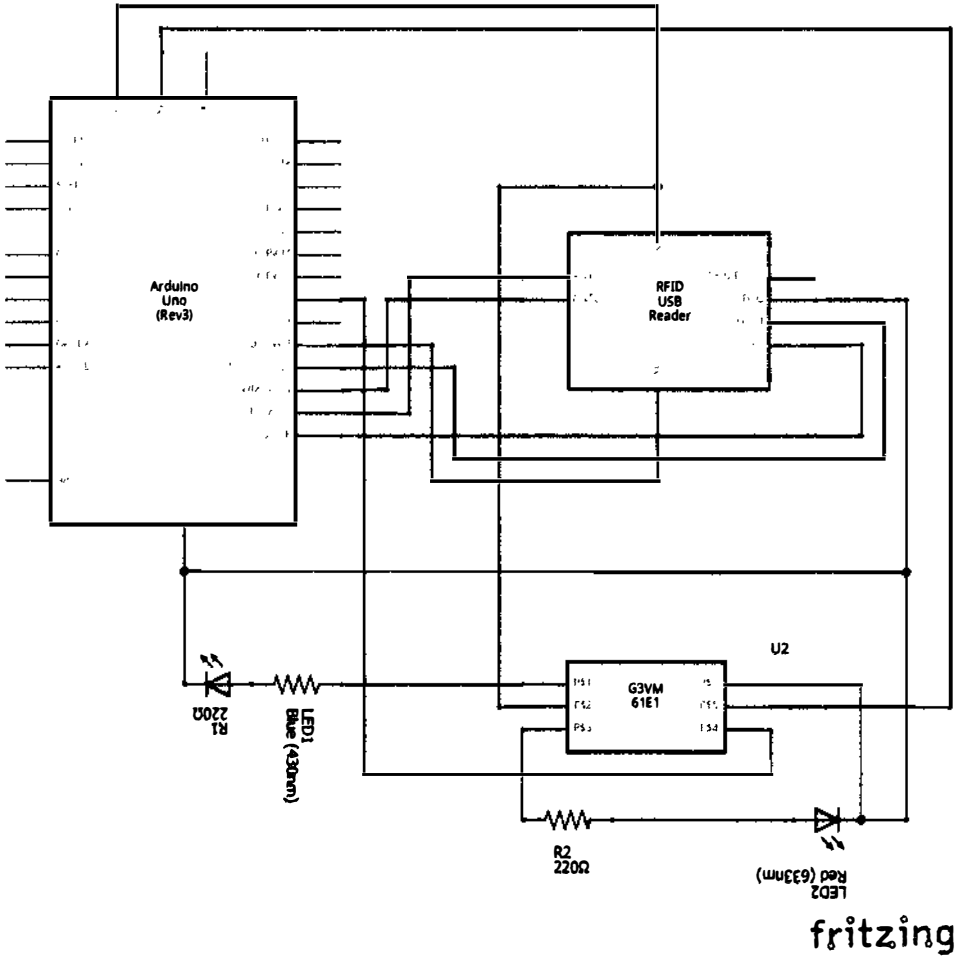


Рис. 109 ❖ Принципиальная схема подключения для практического занятия 34

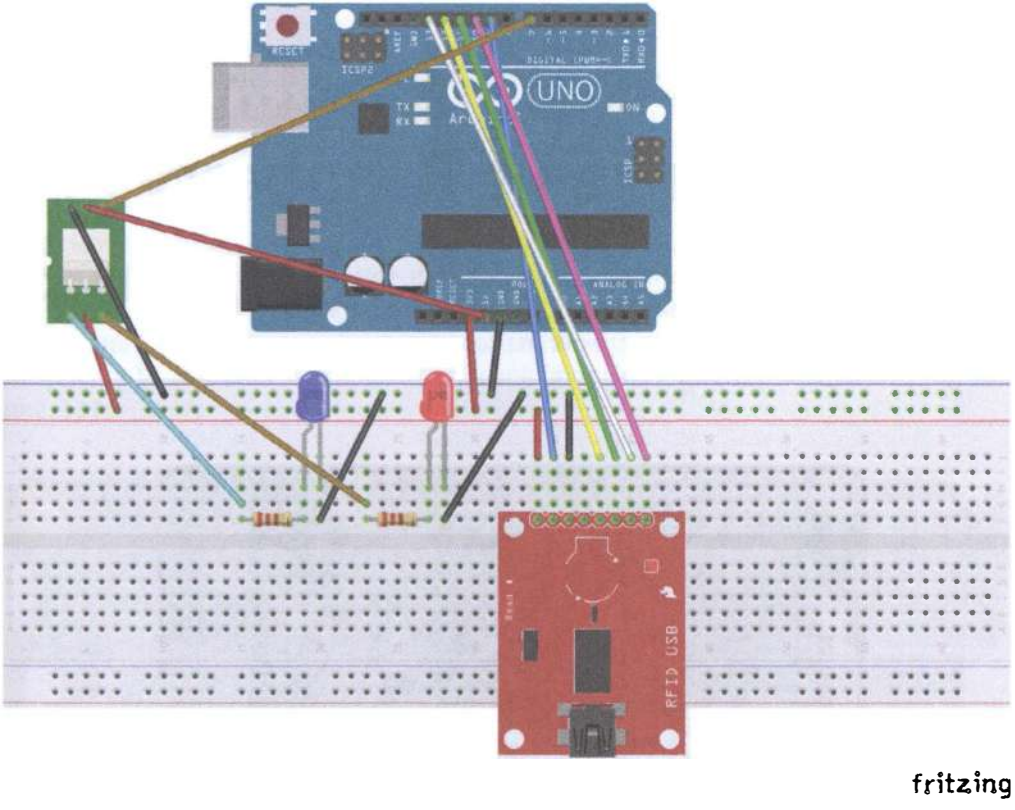
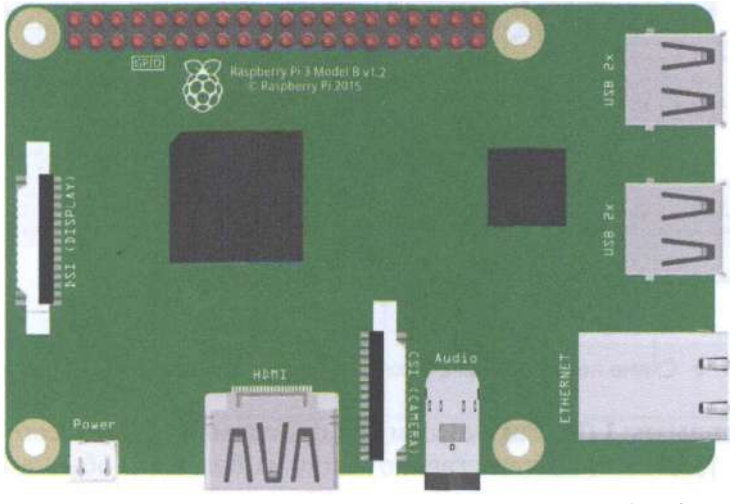


Рис. 110 ❖ Схема подключения с макетной платой для практического занятия 34

Код программы 1 (SetPassword) и **код программы 2 (DoorCon)** загружены по адресам [13] и [14] соответственно ввиду их большого объёма и большого количества ошибок при их копировании в Arduino IDE.

Часть 2

Практика на Raspberry Pi 3 (модель B)



Основано на книге Android Things Projects [15]

Введение

Raspberry Pi 3 Model B продолжает оставаться наиболее популярным в мире компьютером на плате размером с кредитную карту (SBC – Single Board Computer, также SoM – System on Module), даже несмотря на выход в 2017 году новой версии – Model B+. По сути, эта плата может заменить ваш настольный (десктопный) компьютер, если, конечно, вы не играете в требовательные компьютерные игры и не занимаетесь редактированием 3D-графики и прочим майнингом. На плате нет никаких вентиляторов, поэтому она будет самым тихим компьютером, на который для теплоотведения можно прикрепить три радиатора, входящих в официальный комплект поставки этой платы [21]. Raspberry Pi 3 Model B располагает следующими характеристиками [20], см. рис. 111:

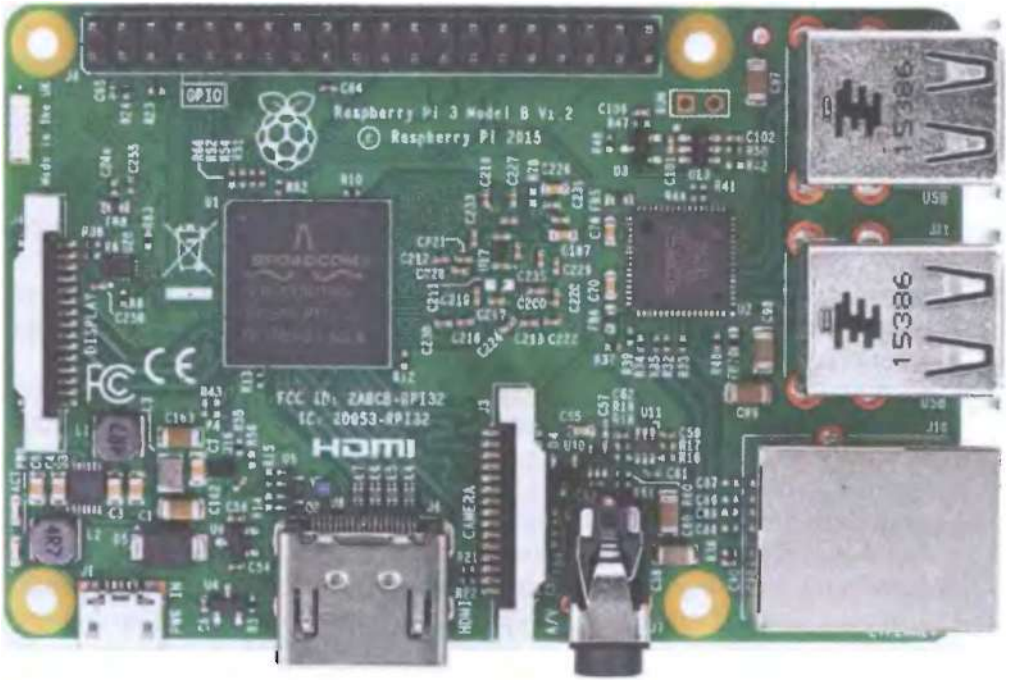


Рис. 111 ❖ Лицевая сторона платы Raspberry Pi 3 модели B [20]

- 64-битным четырёхъядерным процессором ARM Cortex-A53 с тактовой частотой 1,2 ГГц на ядро на однокристальном чипе Broadcom BCM2837 (самый большой чёрный квадрат на рисунке, на него прикрепляется самый большой радиатор из комплекта в первую очередь);
- графическим двухъядерным процессором VideoCore IV®, поддерживающим стандарты OpenGL ES 2.0, OpenVG, MPEG-2, VC-1 и способным ко-

дировать, декодировать и выводить Full HD-видео (1080p, 60 FPS, H.264 High-Profile);

- 1 ГБ оперативной памяти LPDDR2 SDRAM, которая делится с графической подсистемой (чип памяти располагается в виде чёрного квадрата в центре оборотной стороны платы; один из радиаторов (самый маленький) в комплекте [21] предназначен именно для него);
- цифровым видеовыходом HDMI (на рисунке внизу, под надписью HDMI; при подключении к нему разрешение варьируется от 640×350 (EGA) до 1920×1200 (WUXGA) для HDMI. Композитный выход работает в форматах PAL и NTSC);
- композитным выходом (видео/аудио): 3,5 мм (4 pin, на рисунке внизу справа от разъёма CSI-2 для камеры);
- USB/Ethernet Hub'ом – второй по величине чёрный квадрат на рисунке (с буквами SMSC), на который тоже надо поставить радиатор;
- 4 портами USB 2.0 (на рисунке справа вверху и справа по центру), в которые можно подключать клавиатуру, мышь и другие устройства;
- сетевыми модулями: WiFi 802.11n, 10/100 Мб RJ45 Ethernet (на рисунке справа внизу);
- Bluetooth 4.1, Bluetooth Low Energy;
- разъёмом для дисплея: Display Serial Interface (DSI, на рисунке – слева, рядом с надписью DISPLAY);
- разъёмом для видеокамеры: MIPI Camera Serial Interface (CSI-2, на рисунке – внизу по центру, рядом с надписью CAMERA);
- разъёмом для карты памяти microSD (с левой стороны на обратной стороне платы, от 4 Гб), на которую устанавливаются и с которой загружаются образы операционных систем;
- 40 пинами/портами для ввода/вывода – больше, чем на любой другой плате (GPIO, на рисунке – вверху, между двумя дырками), включающими UART (Serial), I2C/TWI, SPI с селектором между двумя устройствами и пины земли и питания на 3,3 В и 5 В;
- разъёмом питания micro-USB (от адаптера на 5 В, входящего в комплект поставки [21], на рисунке разъём снизу слева, рядом с дыркой).

Обычно на microSD-карту уже установлены операционная система Raspbian OS и загрузчик Noobs, с помощью которого можно установить другую систему. Операционные системы, которые поддерживаются платой и устанавливаются с помощью копирования образа на microSD-карту, благодаря архитектуре процессора ARMv53 могут быть совершенно разнообразными: Debian Wheezy, Ubuntu Mate, Fedora Remix, MS Windows 10, Android Things, Windows 10 IoT Core, Android, OSMC, LIBREELEC, PINET, RISC OS, ICHIGOJAM RPI и т. д. Также можно установить Raspberry Pi Desktop для платформ PC и Mac. Полный список поддерживаемых операционных систем, а также многое другое можно найти на официальном сайте https://www.raspberrypi.org.

Мы будем работать с Raspberry Pi 3 Model B в связи с тем, что, кроме всего прочего, она официально поддерживается операционной системой Android

Things (см. [16]), для которой мы будем разрабатывать приложения в этой части учебного пособия.

Установка ОС Android Things

0. Вставьте Raspberry Pi 3 в пластмассовый корпус. Будьте осторожны! Не используйте грубую силу!

1. Ознакомьтесь с начальной информацией об операционной системе Android Things: https://developer.android.com/things/get-started/.

2. Чтобы скачать и установить образ Android Things, следуйте инструкциям на странице https://developer.android.com/things/hardware/raspberrypi.html#flashing_the_image:

1) перейдите на страницу https://partner.android.com/things/console и загрузите Setup Utility (версия 1.0.21 на данный момент; требуется аккаунт Google (gmail)). По желанию вы можете создать новый продукт, см. инструкцию здесь: https://developer.android.com/things/console/create;
2) разархивируйте файл android-things-setup-utility.zip и запустите соответствующий файл. В окне консоли нажмите 1 (install Android Things and optionally set up Wi-Fi). Затем нажмите 1 снова для выбора Raspberry Pi 3 в качестве платы. И затем – ещё раз 1 (Choosing a generic image of Android Things for flashing the board);
3) найдите в наборе microSD-карту на 16 Гб, вставьте её в больший адаптер, а его – в кардридер, затем подключите кардридер к USB, как просит программа установки, и нажмите **Enter** (Plug the SD card into your computer. Press [Enter] when ready). Выберите диск (например, \\.\PHYSICALDRIVE3 (15.9 GB) – Generic-SD/MMC USB Device) и нажмите **Enter**. Вы увидите сообщение: **This will erase the selected drive. Are you sure? (Y/N)** – нажмите **Y**. Затем вы должны увидеть следующее: **Flashing [ ] 1% eta 7m11s**. Подождите указанное время. Далее идёт процесс валидации, и когда он закончится, на вопрос **Would you like to set up Wi-Fi on this device? (Y/N)** выберите **N**. Нажмите **Enter** для выхода из программы установки;
4) когда программа установки закончит запись образа на microSD-карту, извлеките кардридер и выньте из него адаптер с картой, а затем выньте карту из адаптера. Вставьте карту в microSD-слот на обратной стороне платы Raspberry Pi 3;
5) подключите плату Raspberry Pi 3 с помощью кабеля USB Am-micro-USBm к адаптеру питания (miniUSB-слот на плате), затем подключите к плате Ethernet-кабель в слот RJ-45 платы, потом подключите HDMI-кабель к плате и к внешнему дисплею (если у дисплея нет HDMI-выхода, используйте переходник HDMI-VGA и кабель VGA m-m). При этом, если у вас обычный монитор без сенсорного экрана, понадобится USB-мышь, которую надо подключить к одному из USB-портов Raspberry Pi 3. Если у вас монитор с сенсорным экраном, нежно подключить кабель USB-USB

к входу сенсорного управления на мониторе и к плате и воспользоваться сенсорным экраном монитора. Включите адаптер питания и монитор в сеть и убедитесь, что вы видите на мониторе заставку операционной системы Android Things. Если вы её не видите, отключите всё от питания, выньте microSD-карту из слота платы и повторите шаги 2.2–2.5;

6) пропустите установку Android Things Toolkit. После загрузки вы должны увидеть начальный экран Android Things (Overview). В разделе меню menu/system/system updates/check for update вы должны видеть, что установлена система последней версии (System version 8.1.0; Security patch level: Aug. 5, 2018). **Запомните IP-адрес, который отображается в разделе Network!** (например: 192.168.1.106).

Пины Raspberry Pi 3:

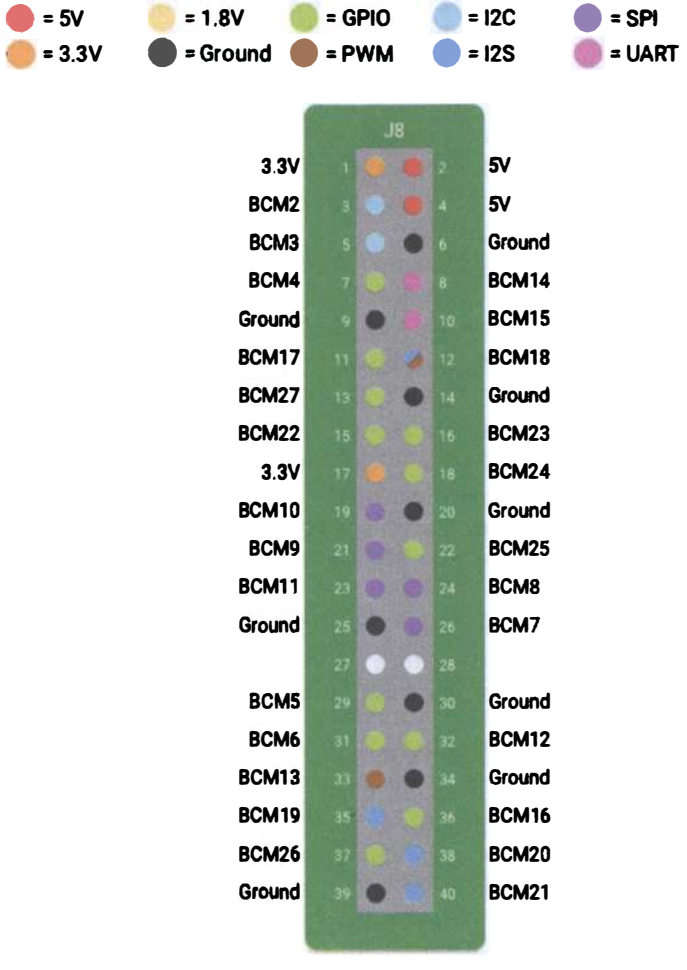


Рис. 112 ❖ Пины платы Raspberry Pi 3 [16]

Первый проект в ОС Android Things – трёхцветный светодиод

0. Создайте новый проект в Android Studio (версии 3.1.4; если у вас не установлена среда Android Studio, перейдите на сайт developer.android.com, загрузите и установите её). Дайте имя проекту, в следующем окне выберите дополнительно с Phone and Tablet платформу Android Things внизу и minimum SDK установите как API 24 (устанавливается по умолчанию). В следующем окне выберите Empty Activity, дайте имя activity, затем в следующем окне выберите Android Things Empty Activity; нажмите **Next** и **Finish**.

Вообще, конкретно в этом проекте нам не нужно выбирать Phone and Tablet-приложение, но для интернета вещей такой выбор естественен: как правило, приложения для интернета вещей используют следующую архитектуру: датчики/сенсоры – плата SoM (System on Module) – приложение, работающее на этой плате – интернет/облако/сервер/хранилище данных/облачная платформа – мобильное или иное приложение с пользовательским интерфейсом. Поэтому здесь и в дальнейшем рекомендуется создавать как Android Things-приложение, так и приложение для Phone и Tablet (смартфона и планшета).

1. Нам понадобятся:
   - соединительные провода,
   - 3 резистора на 220 Ом;
   - 3 резистора на 10 кОм;
   - макетная плата;
   - трёхцветный светодиод (с общим катодом, возьмите его из набора с Arduino Uno [1]);
   - 3 кнопки;
   - Raspberry Pi 3;
   - монитор с HDMI-выходом;
   - кабель HDMI m-m;
   - адаптер питания / кабель питания для HDMI-монитора;
   - адаптер питания для Raspberry Pi 3 (5 В).

Перед подключением всех компонентов к плате убедитесь, что она отключена от питания, иначе вы можете повредить её или компоненты.

Следующая схема с макетной платой показывает, как нужно соединить все компоненты и подключить их к Raspberry Pi 3 (рис. 113).

Стягивающий резистор номиналом 10 кОм соединяет один выход кнопки с землёй. У каждой кнопки есть такой стягивающий резистор. Резистор номиналом 220 Ом соединяет каждый из 3 пинов (анод, отвечающий за определённый цвет) трёхцветного светодиода с пином на плате, ограничивая ток, текущий через светодиод. Четвёртый пин светодиода, или общий катод, соединяется с пином земли на плате Raspberry Pi 3.

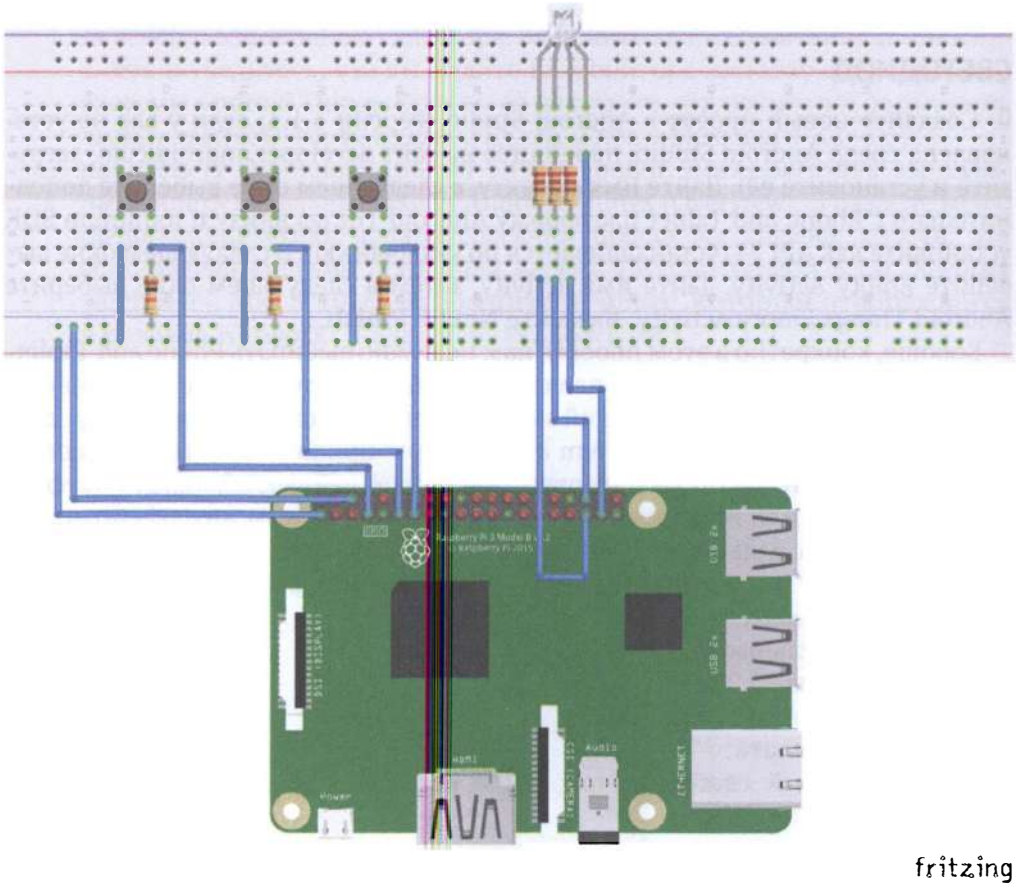


Рис. 113 ❖ Схема подключения для задания 1

2. Откройте файл в папке manifests -> AndroidManifest.xml и добавьте следующую строку перед тегом application (это нужно для того, чтобы получить разрешение обращаться к пинам платы):

```
<uses-permission android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

3. Откройте файл MainActivity.java (в папке things -> название пакета, например things -> com.example.me.IoTbook1) и добавьте следующие строки перед методом onCreate в раздел объявлений переменных класса:

```
private Gpio mREDLedGpio, mBlueLedGpio, mGREENLedGpio, mREDButtonGpio, mBlueButtonGpio,
mGREENButtonGpio;;
boolean redPressed = false;
boolean greenPressed = false;
boolean bluePressed = false;
TextView text1;
```

Эти переменные понадобятся нам далее в коде. У нас в проекте есть интерфейс (things –> res –> main_activity.xml) для нашего приложения, и мы будем менять цвет и текст элемента TextView, когда будет нажата одна из трёх кнопок. Чтобы инициализировать переменную text1, добавьте в метод onCreate следующую строку (при этом убедитесь, что в свойствах элемента TextView атрибут id=text1):

```
text1 = findViewById(R.id.text1);
```

Следующий код в методе onCreate работает только для одной кнопки и одного красного цвета трёхцветного светодиода. Вам нужно самостоятельно выяснить названия пинов и дописать код для остальных цветов светодиода и оставшихся двух кнопок. Чтобы выяснить названия и номера пинов, см. рис. 112 выше.

```
PeripheralManager manager = PeripheralManager.getInstance();
    try {
        mREDLedGpio = manager.openGpio("BCM19"); //red color
        // Step 2. Configure as an output.
        mREDLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW); //is lit
        // Step 1. Create GPIO connection.
        mREDButtonGpio = manager.openGpio("BCM4"); //red color button
        // Step 2. Configure as an input.
        mREDButtonGpio.setDirection(Gpio.DIRECTION_IN);
        // Step 3. Enable edge trigger events.
        mREDButtonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING);
        // Step 4. Register an event callback.
        mREDButtonGpio.registerGpioCallback(mREDCallback);
    } catch (IOException e) {
    Log.e("onCreate", "Error on PeripheralIO API", e);
}
```

Затем нам нужно добавить несколько методов вне метода onCreate для работы с трехцветным светодиодом и для корректного завершения работы нашего приложения:

```
// Step 4. Register an event callback.
private GpioCallback mREDCallback = new GpioCallback() {
    @Override
    public boolean onGpioEdge(Gpio gpio) {
        try {
            redPressed = !redPressed;
            mREDLedGpio.setValue(redPressed);
            text1.setTextColor(Color.RED);
            text1.setText("RED");
            Log.i("GpioCallback", "Red LED button was pressed");

        } catch (IOException e) {
            Log.e("onKeyDown", "Error on PeripheralIO API", e);
        }
        // Step 5. Return true to keep callback active.
        return true;
```

```
    }
};

@Override
protected void onDestroy() {
    super.onDestroy();

    // Step 6. Close the resource
    if (mREDButtonGpio != null) {
        mREDButtonGpio.unregisterGpioCallback(mREDCallback);
        try {
            mREDButtonGpio.close();
        } catch (IOException e) {
            Log.e("onDestroy", "Error on PeripheralIO API", e);
        }
    }
    if (mREDLedGpio != null) {
        try {
            mREDLedGpio.close();
        } catch (IOException e) {
            Log.e("onDestroy", "Error on PeripheralIO API", e);
        }
    }

}
```

4. Теперь добавьте код для остальных двух кнопок и зелёного и синего цветов светодиода. Исправляйте ошибки в коде по мере его добавления с помощью блоков try + catch, импорта классов и т. д.

5. Подключите к питанию HDMI-монитор и плату Raspberry Pi 3. Для запуска своего приложения (нажатием зелёной кнопки **Run** на панели инструментов Android Studio, при этом выделив MainActivity.java из папки things, не из папки mobile!) необходимо подключиться к плате, чтобы она была в списке устройств в ADB (Android Debug Bridge) tool. Для этого сначала вы должны найти Android Studio SDK-папку на вашем компьютере (C:\Users\UserName\AppData\Local\Android\Sdk по умолчанию), найдите папку platform tools и откройте консоль для папки platform-tools (для windows – **Shift**+клик правой кнопкой мыши по папке – открыть окно команд; для windows 10 – открыть окно **PowersShell** здесь); затем в окне команд напечатайте «adb connect IP», где IP – тот адрес, который вы запомнили из раздела «Введение» этой части пособия, он же отображается на начальном экране Android Things, который виден на HDMI-мониторе (например: adb connect 192.168.1.106; для windows 10 PowerShell: .\adb.exe connect 192.168.1.106); затем убедитесь, что плата подключена, с помощью команды «adb devices» (или «.\adb.exe devices») – вы увидите название платы в окне команд. После этого подключения по IP вы получите доступ к плате в среде Android Studio (в окне ADB после запуска приложения вы увидите что-то наподобие «Google Iot_rpi3 Android 7.0, API 24» и то же самое – внизу окна **Android Studio** в окне **Android Monitor**). В окне устройств, открытом

ADB, выберите плату (Google Iot_rpi3 Android 7.0, API 24); откажитесь или примите установку Instant Run, на ваш выбор.

6. После запуска приложение должно работать. Заметьте, что вы можете нажать две или даже три кнопки последовательно или одновременно, чтобы получить различные цвета с помощью трёхцветного светодиода, при этом последний выбранный цвет в виде его названия и цвета отображается в окне приложения на мониторе. Насладитесь своим первым приложением на плате Raspberry Pi 3 с пользовательским интерфейсом, информацией на котором вы управляете с помощью аппаратной составляющей приложения – нажатия кнопок!

Второй проект в Android Things – система сигнализации

0. В этом задании мы создадим проект из реальной жизни, использующий PIR (Passive InfraRed, пассивный инфракрасный) сенсор для обнаружения движения и затем уведомляющий пользователя через его смартфон (или другое мобильное устройство) с помощью дополнительного приложения-компаньона, написанного для смартфона и не зависящего от платы Raspberry Pi 3. Основными шагами будут следущие:

- PIR-сенсор сканирует область обнаружения с целью обнаружения движения;
- как только он обнаруживает движение, он уведомляет нашу плату (или приложение, которое на ней работает) под управлением операционной системы Android Things;
- плата (приложение) обрабатывает событие уведомления и соединяется с Google Firebase, чтобы отправить сообщение на смартфон пользователя.

Создайте новый проект в Android Studio. Дайте имя проекту, в следующем окне выберите дополнительно с Phone and Tablet платформу Android Things внизу и minimum SDK установите как API 24 (устанавливается по умолчанию). В следующем окне выберите Empty Activity, дайте имя activity, затем в следующем окне выберите Android Things Empty Activity; потом нажмите **Next** и **Finish.**

1. Нам понадобятся:

- соединительные провода,
- PIR-сенсор;
- плата Raspberry Pi 3;
- HDMI-монитор;
- кабель HDMI m-m;
- адаптер питания / кабель для HDMI-монитора;
- адаптер питания для платы Raspberry Pi 3 (5 В);
- ваш Google Firebase-аккаунт.

PIR-сенсор выглядит так:

Рис. 114 ❖ PIR-сенсор

Наиболее распространённая модель – та, которая использует линзы Френеля, которые помогают расширить область обнаружения движения. У сенсора есть два потенциометра: один для регулировки чувствительности, другой – для регулировки времени, в течение которого сигнал удерживается высоким (логическая единица) при обнаружении объекта. Это время меняется в диапазоне от 0.3 до 18 секунд. Максимальное расстояние обнаружения составляет 7 метров, угол обзора сенсора (FOV, Field Of View) равен 120 градусам. Рабочая температура может меняться в пределах от –15 до +70 °C.

Перед подключением всех компонентов к плате убедитесь, что она отключена от питания, иначе вы можете повредить её или компоненты.

Следующая схема с макетной платой показывает, как нужно подключить PIR-сенсор к Raspberry Pi 3 (рис. 115).

2. Откройте файл в папке manifests -> AndroidManifest.xml и добавьте следующую строку перед тегом application (это нужно для того, чтобы получить разрешение обращаться к пинам платы):

```
<uses-permission android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

3. Откройте файл MainActivity.java (в папке things -> название пакета, например things -> com.example.me.IoTbook2) и добавьте следующие строки перед методом onCreate в раздел объявлений переменных класса:

```
private Gpio gpioPin;
private boolean status;
```

Эти переменные понадобятся нам далее в коде. Следующие строки кода нужны для установления соединения с сенсором: добавьте их в метод onCreate:

```
PeripheralManager manager = PeripheralManager.getInstance();
gpioPin = null;
try {
    gpioPin = manager.openGpio("BCM4"); // the signal pin of the PIR sensor is connected to
```

```
BCM4
    gpioPin.setDirection(Gpio.DIRECTION_IN); // we want to read values; in we want to write
values - DIRECTION_OUT
    gpioPin.setActiveType(Gpio.ACTIVE_HIGH); // the value is true if the pin is at high
voltage; LOW - true at low voltage
} catch (IOException e) {
    e.printStackTrace();
}
```

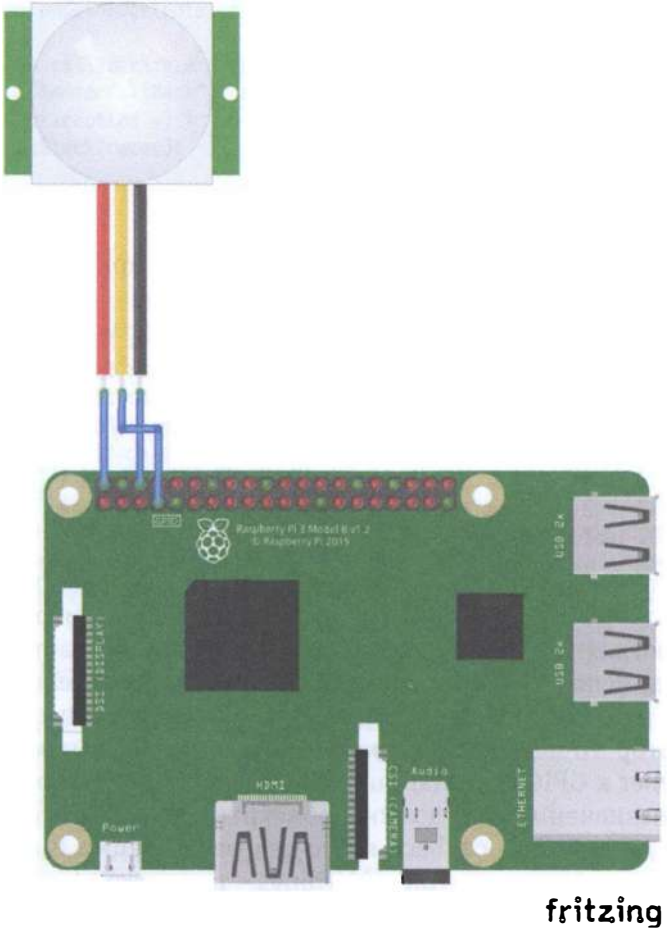


Рис. 115 ❖ Схема подключения для задания 2

Мы могли бы сделать следующее: создать отдельный поток и считывать состояние с пина всё время, как указано в следующем коде, помещённом в конец метода onCreate:

```
(new Thread(new Runnable() {
    @Override
```

```
    public void run() {
        try {
            while (true) {
                status = gpioPin.getValue(); // we read the state of the sensor: true or
false
                Log.d("Runnable","State ["+status+"]");
                if (status) {
                    Log.i("Runnable","Motion detected...");
                }
                Thread.sleep(5000); // every 5 seconds
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
})).start();
```

Если мы так сделаем, подключим плату и запустим приложение, то получим следующие строки в логе Logcat:

```
09-23 12:04:05.231 1862-1879/com.example.me.iotbook2 I/Runnable: Motion detected...
09-23 12:04:10.232 1862-1879/com.example.me.iotbook2 D/Runnable: State [false]
09-23 12:04:15.234 1862-1879/com.example.me.iotbook2 D/Runnable: State [true]
09-23 12:04:15.234 1862-1879/com.example.me.iotbook2 I/Runnable: Motion detected...
09-23 12:04:20.235 1862-1879/com.example.me.iotbook2 D/Runnable: State [false]
09-23 12:04:25.236 1862-1879/com.example.me.iotbook2 D/Runnable: State [false]
09-23 12:04:30.237 1862-1879/com.example.me.iotbook2 D/Runnable: State [true]
09-23 12:04:30.237 1862-1879/com.example.me.iotbook2 I/Runnable: Motion detected...
```

Хотя этот подход работает, он требует много временных ресурсов, потому что нашему Android Things-приложению приходится «мониторить» состояние пина всё время, даже если PIR-сенсор никого не обнаруживает. К счастью, есть другой подход, использующий механизм listeners (слушателей событий). Такой подход потребляет меньше ресурсов по времени и очень похож на подход, с помощью которого мы привыкли разрабатывать Android-приложения. Чтобы добавить listener к GPIO (General Purpose Input/Output, интерфейс ввода-вывода общего назначения), мы должны объявить событие, которое хотим прослушивать, и реализовать callback-класс для регистрации и обработки этого события (см. первый проект в Android Things – там то же самое).

Есть 4 типа изменяющихся событий: **EDGE_NONE** (не произошло никакого события), **EDGE_RISING** (увеличивающийся уровень напряжения, значение напряжения на пине меняется с логического 0 на 1), **EDGE_FALLING** (уменьшающийся уровень напряжения, значение напряжения на пине меняется с логической 1 на 0) и **EDGE_BOTH** (комбинация предыдущих двух событий: мы хотим знать, меняется ли сигнал хоть как-то – от 0 до 1 или от 1 до 0). В этом проекте нам нужно знать, меняется ли сигнал с низкого значения (0) на высокое значение напряжения (1), так как мы обнаруживаем движение; поэтому

в коде мы используем тип триггера EDGE_RISING. Добавьте следующую строку кода в метод onCreate в блок try catch:

```
gpioPin.setEdgeTriggerType(Gpio.EDGE_RISING);
```

В этот раз мы создадим новый callback-класс вместо добавления callback-метода. Добавьте следующий код в конец класса MainActivity.java:

```
public class SensorCallBack implements GpioCallback {
    @Override
    public boolean onGpioEdge(Gpio gpio) {
        try {
            boolean callBackState = gpio.getValue();
            Log.d("SensorCallBack","Callback state ["+callBackState+"]");
        } catch (IOException e) {
            e.printStackTrace();
        }
        return true;
    }

    @Override
    public void onGpioError(Gpio gpio, int error) {
        Log.e("SensorCallBack","GPIO error");
    }
}
```

Есть два важных метода, которые мы должны переопределить, чтобы настроить поведение callback-класса: public Boolean onGpioEdge и public Boolean onGpioError. Первый вызывается тогда, когда срабатывает событие, которое мы зарегистрировали с помощью setEdgeTriggerType. Второй срабатывает при условии возникновения ошибки при чтении данных с пина платы.

Добавьте следующую строку в секцию объявления переменных класса MainActivity:

```
SensorCallBack callback;
```

Наконец, мы должны зарегистрировать наш callback-класс. В методе onCreate добавьте следующий код в конце:

```
callback = new SensorCallBack();
try {
    gpioPin.registerGpioCallback(callback);
} catch (IOException e) {
    e.printStackTrace();
}
```

После того как вы сделали всё это, можно подключить плату к питанию и запустить приложение; как только вы махнёте рукой перед PIR-сенсором, в логе Logcat увидите следующее:

```
09-23 14:11:04.444 2452-2452/com.example.me.iotbook2 D/SensorCallBack: Callback state [true]
09-23 14:11:10.874 2452-2452/com.example.me.iotbook2 D/SensorCallBack: Callback state [true]
09-23 14:11:41.494 2452-2452/com.example.me.iotbook2 D/SensorCallBack: Callback state [true]
```

В завершение мы должны закрыть соединение с GPIO-пином платы. Это делается в методе onDestroy внутри класса MainActivity.java. Мы удаляем все listeners и закрываем соединение с GPIO-пинами (это называется корректное завершение приложения):

```
@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d("MainActivity","onDestroy");
    if (gpioPin != null) {
        gpioPin.unregisterGpioCallback(callback);
        try {
            gpioPin.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        gpioPin = null;
    }
}
```

4. Что, если мы хотим, чтобы наше приложение работало на любой плате, которую поддерживает ОС Android Things, другими словами – что, если мы хотим разработать платонезависимое приложение? Это вполне возможно, так как существует константа Build.BOARD, хранящая автоматически определённый системой тип (название) платы, на которую установлена система. Используя эту информацию, мы можем выбрать название пина платы во время выполнения программы. Создайте новый Java-класс (things -> java -> щёлкните правой кнопкой мыши по названию пакета (package name) и выберите New -> Java Class), дайте ему название «BoardPins» и вставьте следующий код:

```
import android.os.Build;

import com.google.android.things.pio.PeripheralManager;

import java.util.List;

public class BoardPins {
    private static final String EDISON_ARDUINO = "edison_arduino";
    private static final String RASPBERRY  ="rpi3";
    public static String getPirPin() {
        switch (getBoardName()) {
            case RASPBERRY:
                return "BCM4";
            case EDISON_ARDUINO:
                return "IO4";
            default:
                throw new IllegalArgumentException ("Unsupported device");
        }
    }

    public static String getBoardName() {
        String name = Build.BOARD;
```

```
        if (name.equals("edison")) {
            PeripheralManager manager = PeripheralManager.getInstance();
            List<String> pinList = manager.getGpioList();
            if (pinList.size() > 0) {
                String pinName = pinList.get(0);
                if (pinName.startsWith("IO")) return EDISON_ARDUINO;
            }
        }
        return name;
    }
}
```

Название платы, возвращаемое с помощью Android Things SDK (Software Developer Kit), не помогает нам отличить различные варианты платы Intel Edison. Поэтому мы пролистываем список пинов и ищем в нём особое название пина, чтобы определить тип платы и её конкретный вариант.

После создания класса BoardPins можно открыть MainActivity.java и заменить строку

```
gpioPin = manager.openGpio("BCM4");
```

следующей строкой:

```
gpioPin = manager.openGpio(BoardPins.getPirPin());
```

Теперь наше приложение будет работать, по крайней мере, на двух платах, названия которых очевидны из кода. Можно даже изменить значение TextView в интерфейсе приложения: показать там название платы и имя пина, к которому подключён PIR-сенсор вместо стандартной фразы Hello World!:

```
TextView text1 = findViewById(R.id.text1);
text1.setText("This app runs on the "+BoardPins.getBoardName()+" board using "+BoardPins.
getPirPin()+" pin!");
```

5. Теперь мы займёмся системой уведомлений. Мы будем использовать сервис уведомлений Google Firebase, поэтому вы должны создать там аккаунт, если у вас его ещё нет. Есть несколько способов посылать уведомление из Android Things-приложения на смартфон пользователя. Мы будем использовать тему (**topic**). Topic похож на канал. После того как приложение подписывается на определённый topic, оно получит все сообщения, опубликованные в этом канале. В нашем проекте смартфон пользователя (точнее, приложение, которое на нём работает) ведёт себя как подписчик (**subscriber**), получающий сообщения из канала, в то время как Android Things-приложение ведёт себя как издатель/тот, кто публикует сообщения (**publisher**).

5.1. Перейдите на домашнюю страницу Firebase (https://firebase.google.com/) и щёлкните по Get Started, для того чтобы создать ваш аккаунт. Заполните все необходимые поля и создайте аккаунт.

5.2. Перейдите в консоль Firebase (https://console.firebase.google.com). Добавьте новый проект (нажмите на **Add/create a new project**). В появившемся окне введите название проекта и страну/регион по необходимости и выберите все

3 чекбокса (галочки). Нажмите **Create Project**. Затем немного подождите, пока создаётся проект, и нажмите **Continue**. Вы увидите консоль администратора для вашего проекта. Добавьте наше Android-приложение в этот проект с помощью нажатия на кнопку **Add firebase to your Android app** или на круглую иконку **Android**. В следующем окне надо добавить свойства Android Things-приложения. Введите название пакета (package name, его можно найти в файле манифеста AndroidManifext.xml в качестве значения переменной package, например: **package="com.example.me.iotbook2"**) и псевдоним для нашего приложения. Нажмите кнопку **Register application**. Затем загрузите файл google-services.json и поместите его в корень папки things в Android Studio: нажмите **Копировать на этом файле** (или выделите его и нажмите **Ctrl+C**) в папке, затем нажмите правой кнопкой мыши по папке things в Android Studio. Выберите любую папку, нажмите **OK** и затем в поле **To directory** выберите папку things, нажмите **OK**. Если переключиться в отображение Project view в Android Studio, вы увидите файл в папке things. Наконец, нужно нажать **Next** в консоли Firebase и добавить Firebase SDK в ваши файлы build.gradle. Добавьте строку **classpath 'com.google.gms:google-services:4.1.0'** в ваш файл build.gradle уровня проекта (Project) в раздел dependencies, а затем добавьте строку **implementation 'com.google.firebase:firebase-core:16.0.1'** в ваш файл build.gradle уровня модуля things (Module: things) в раздел dependencies; после этого добавьте строку **apply plugin: 'com.google.gms.google-services'** в самый конец того же файла (вне последнего символа }). Теперь нажмите **Sync Now** для синхронизации и построения проекта. После того как синхронизация завершится, нажмите **Next** в консоли Firebase и запустите своё приложение (папка things), чтобы проверить наличие связи между приложением и Firebase. Если всё в порядке, вы должны увидеть зелёное сообщение Congratulations в консоли Firebase.

5.3. Создайте следующий класс NotificationManager.java в вашем проекте (things – java – package – клик правой кнопкой мыши – New – Java Class):

```
import android.os.AsyncTask;
import android.util.Log;

import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class NotificationManager {

    private static NotificationManager me = null;

    private NotificationManager() {}

    public static NotificationManager getInstance() {
        if (me == null)
            me = new NotificationManager();

        return me;
    }

    public void sendNotificaton(String message, String key) {
```

```
        (new FirebaseNotificationTask()).execute(new String[]{message, key});
    }

    private class FirebaseNotificationTask extends AsyncTask<String, Void, Void> {

        @Override
        protected Void doInBackground(String... strings) {
            String msg = strings[0];
            String key = strings[1];
            Log.d("Alm", "Send data");
            try {
                HttpURLConnection con = (HttpURLConnection) (new URL("http://fcm.
googleapis.com/fcm/send")).openConnection();
                con.setRequestMethod("POST");
                con.setRequestProperty("Authorization", "key=" + key);
                con.setRequestProperty("Content-Type", "application/json");
                con.setDoInput(true);
                con.setDoOutput(true);
                con.connect();

                String body = "{\n" +
                        "  \"to\": \"/topics/alarm\",\n" +
                        "  \"data\": {\n" +
                        "  \"message\": \"" + msg + "\"" +
                        "  }\n" +
                        "}";
                Log.d("Alm", "Body ["+body+"]");
                con.getOutputStream().write(body.getBytes());
                InputStream is = con.getInputStream();
                byte[] buffer = new byte[1024];
                while ( is.read(buffer) != -1)
                    Log.d("Alm", new String(buffer));
                con.disconnect();

            }
            catch(Throwable t) {
                t.printStackTrace();
            }

            return null;
        }
    }
}
```

Теперь откройте MainActivity.java и в методе onGpioEdge callback класса добавьте следующую строку после сообщения Log.d:

```
NotificationManager.getInstance().sendNotificaton("Alarm!", server_key);
```

где server_key – это ключ, который вы можете получить из консоли Firebase: перейдите в **Project Overview** в меню слева вверху с иконкой home; нажмите на псевдоним (alias) вашего проекта, созданный ранее (с помощью круглой иконки Android), затем нажмите на иконку настроек (шестерёнку) и потом в открывшемся окне перейдите на закладку **Cloud Messaging** – там вы найдёте значение server key (которое очень длинное).

5.4. Измените файл AndroidManifest.xml file в вашей папке things – добавьте следующие две строки перед тегом application, так как мы работаем с Firebase и нам нужен доступ в интернет:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Теперь наше Android Things-приложение готово посылать уведомления. Но нам нужно **приложение для смартфона**, чтобы получать эти уведомления. Такое приложение обычно называют приложением-компаньоном (**companion app**).

5.5. Наше Android-приложение-компаньон будет подписываться на канал, используемый нашим Android Things-приложением для отправки уведомлений; задействовать сервис для прослушивания входящих уведомлений; показывать уведомления пользователю. Самое интересное в том, что у нас уже есть мобильное приложение в нашем проекте – см. папку mobile. Всё, что нам нужно сделать, – это удалить стандартный элемент TextView со значением Hello World! из интерфейса приложения (mobile – res – layout – activity_main.xml) и добавить туда кнопку. В последней (на данный момент) версии Android Studio (3.1.4) существует ошибка с пользовательским интерфейсом на закладке **Design** в редакторе интерфейса приложения. Чтобы исправить эту ошибку, перейдите в файл build.gradle уровня приложения (Module: mobile) и замените строку

```
implementation 'com.android.support:appcompat-v7:28.0.0-rc02'
```

строкой

```
implementation 'com.android.support:appcompat-v7:27.1.1'
```

Затем синхронизируйте проект. После этого удалите TextView, добавьте кнопку (**Button**) и установите её свойство text в значение SUBSCRIBE через создание нового строкового ресурса. Затем вы должны скопировать **тот же самый** файл google-services.json, который вы скопировали ранее в папку things, в корень папки mobile проекта и изменить файл build.gradle уровня приложения (Module: mobile): добавьте строку **implementation 'com.google.firebase:firebase-core:16.0.1'** в файл build.gradle (Module: mobile) в раздел dependencies; после этого добавьте строку **apply plugin: 'com.google.gms.google-services'** в самый конец того же файла (вне последнего символа }). Дополнительно добавьте следующую строку в раздел dependencies и затем снова синхронизируйте проект:

```
implementation 'com.google.firebase:firebase-messaging:17.3.2'
```

Теперь добавьте следующий код в файл MainActivity.java в метод onCreate вашего мобильного приложения (папка mobile, а не things!):

```
Button but1 = (Button) findViewById(R.id.button);
but1.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
    public void onClick(View v) {
        FirebaseMessaging.getInstance().subscribeToTopic("alarm");
        String tkn = FirebaseInstanceId.getInstance().getToken();

        Toast.makeText(MainActivity.this,"Device subscribed to the channel", Toast.LENGTH_
LONG).show();
        Log.d("App", "Token ["+tkn+"]");
    }
});
```

Нам нужно создать ещё два класса. Один будет отвечать за ввод токена:

```
import android.util.Log;

import com.google.firebase.iid.FirebaseInstanceId;
import com.google.firebase.iid.FirebaseInstanceIdService;

public class FireIDService extends FirebaseInstanceIdService {

    @Override
    public void onTokenRefresh() {
        String tkn = FirebaseInstanceId.getInstance().getToken();
        Log.d("Not","Token ["+tkn+"]");

    }
}
```

А другой класс будет реализовывать сервис, который обрабатывает входящее пуш-уведомление (push notification):

```
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;
import android.util.Log;

import com.google.firebase.messaging.FirebaseMessagingService;
import com.google.firebase.messaging.RemoteMessage;

public class FireMsgService extends FirebaseMessagingService {

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        super.onMessageReceived(remoteMessage);

        Log.d("Msg", "Message received ["+remoteMessage+"]");

        // Create Notification
        Intent intent = new Intent(this, MainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 1410, intent,
                PendingIntent.FLAG_ONE_SHOT);

        String info = null;

        if (remoteMessage.getData().size() > 0) {
```

```
            info = remoteMessage.getData().get("message");
        }

        if (remoteMessage.getNotification() != null) {
            info = remoteMessage.getNotification().getBody();
        }

        NotificationCompat.Builder notificationBuilder = new NotificationCompat.
Builder(this)
                .setSmallIcon(R.drawable.ic_launcher_background)
                .setContentTitle("Message")
                .setContentText(info)
                .setAutoCancel(true)
                .setContentIntent(pendingIntent);

        NotificationManager notificationManager =
                (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

        notificationManager.notify(1410, notificationBuilder.build());
    }
}
```

Не стоит забывать и про объявление сервисов в файле AndroidManifest.xml. Добавьте в этот файл следующее внутри тега application:

```
<service
    android:name=".FireIDService">
    <intent-filter>
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
    </intent-filter>
</service>

<service
    android:name=".FireMsgService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>
```

Следующие строки, добавленные также внутрь тега application, устанавливают иконку уведомления и его цвет, которые мы будем видеть в панели уведомлений:

```
<meta-data
    android:name="com.google.firebase.messaging.default_notification_icon"
    android:resource="@drawable/ic_launcher_background" />
<meta-data
    android:name="com.google.firebase.messaging.default_notification_color"
    android:resource="@color/colorAccent" />
```

6. После этого всё, что вам нужно сделать, – это подключить ваш смартфон к компьютеру (режим отладки) и запустить приложение, выбрав ваш смартфон в качестве устройства для выполнения приложения в ADB, для того чтобы установить приложение на смартфон. Можно также сделать это по-другому – сгенерировать apk-файл приложения с помощью пункта главного меню Android Studio

Build -> Build APK(s), затем найти сгенерированный файл mobile-debug.apk, скопировать его на смартфон, установить приложение и открыть его. Нажмите на кнопку subscribe в вашем приложении, чтобы подписать ваше устройство (точнее, приложение) на канал уведомлений. Если плата Raspberry Pi 3 подключена к питанию и на ней запущено приложение из папки things, вы можете протестировать PIR-сенсор и получить уведомление на ваше мобильное устройство, которое подключено к интернету, где бы вы не находились. Как только система определит двигающийся объект, она свяжется с платформой Google Firebase посредством отправки сообщения с уведомлением. В ответ платформа Firebase пошлёт это сообщение на смартфон пользователя (вы должны увидеть зелёный квадрат в трее сообщений). Как правило, уведомление приходит сразу после запуска обоих приложений, так как вы будете находиться рядом с PIR-сенсором.

Третий проект в Android Things – система мониторинга окружающей среды

0. В этом проекте мы построим сложную систему интернета вещей с помощью операционной системы Android Things, которая измеряет некоторые физические параметры окружающей среды. Этот проект фокусируется на том, как использовать I2C (Inter-Integrated Circuit) с помощью Android Things, как использовать класс Sensor Manager и визуализировать данные, получаемые с сенсоров, с помощью светодиодов. Целью этого проекта является построение системы мониторинга окружающей среды, которая определяет температуру и давление. Вы должны помнить подобные эксперименты из практики по Arduino, но в этот раз мы будем использовать другие сенсоры, и реакция светодиодов будет иной.

Создайте новый проект в Android Studio. Дайте имя проекту, в следующем окне выберите дополнительно с Phone and Tablet платформу Android Things внизу и minimum SDK установите как API 24 (устанавливается по умолчанию). В следующем окне выберите Empty Activity, дайте имя activity, затем в следующем окне выберите Android Things Empty Activity; нажмите **Next** и **Finish**.

1. Нам понадобятся:
   - соединительные провода;
   - сенсор BME280 (или BMP280, если есть);
   - 4 резистора на 220 Ом;
   - макетная плата;
   - трёхцветный светодиод (с общим катодом – он есть в наборе с Arduino Uno);
   - красный светодиод;
   - плата Raspberry Pi 3;
   - HDMI-монитор;
   - кабель HDMI m-m;
   - адаптер питания / кабель для HDMI-монитора;
   - адаптер питания для платы Raspberry Pi 3 (5 В).

Сенсор BME280 выглядит следующим образом:

Рис. 116 ❖ Первый вариант сенсора BME280

В других исполнениях он может выглядеть так:

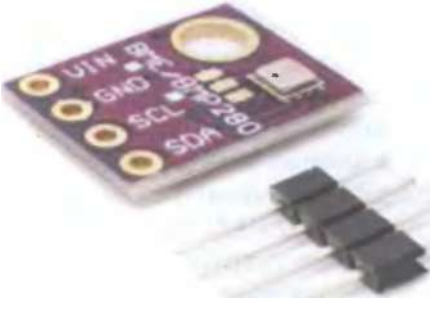


Рис. 117 ❖ Второй вариант сенсора BME280

Этот сенсор может измерять температуру, давление и влажность, в то время как сенсор BMP280 измеряет только температуру и влажность. Сенсор BME280 обладает следующими характеристиками:

- напряжение: 1.8–5 В (но может быть и максимум 3В, что должно быть написано на сенсоре; будьте осторожны и внимательны!);
- интерфейс: I2C (до 3.4 МГц), SPI (до 10 МГц);
- диапазон измеряемой температуры: от –40 °C до +85 °C;
- диапазон измеряемой влажности: от 0 до 100 %;
- диапазон измеряемого давления: от 300 до 1100 ГПа (мб);
- точность измерения температуры: 0.5 °C;

- точность измерения влажности: 3 %;
- точность измерения давления: 1 ГПа (мб).

Теперь немного о единицах измерения давления: 1 ГПа (Гектопаскаль) = $1*10^2$ Па = 100 Па, поэтому 1100 ГПа = 110 000 Па. 1 мб (миллибар) = 1 ГПа = 100 Па, поэтому 110 000 Па = 1100 мб. И наконец, 1 мм рт. ст. (миллиметр ртутного столба) = 1.33322387415 мб, поэтому, например, 735 мм. рт. ст. – это 980 мб, или 980 ГПа.

Перед подключением всех компонентов к плате убедитесь, что она отключена от питания, иначе вы можете повредить её саму или её компоненты.

Следующая схема с макетной платой показывает, как нужно соединить все компоненты и подключить их к Raspberry Pi 3 (рис. 7). В соответствии со схемой мы будем использовать следующие пины: Vin (пин на 3.3 В), GND (земля), SCK/SCL (тактовый сигнал, так как I2C-сенсор использует тактовый/синхронизирующий сигнал в своей работе) и SDI/SDA (пин данных). В некоторых исполнениях сенсора есть и другие пины, но мы не будем их использовать, так как мы будем подключать пины с помощью шины I2C.

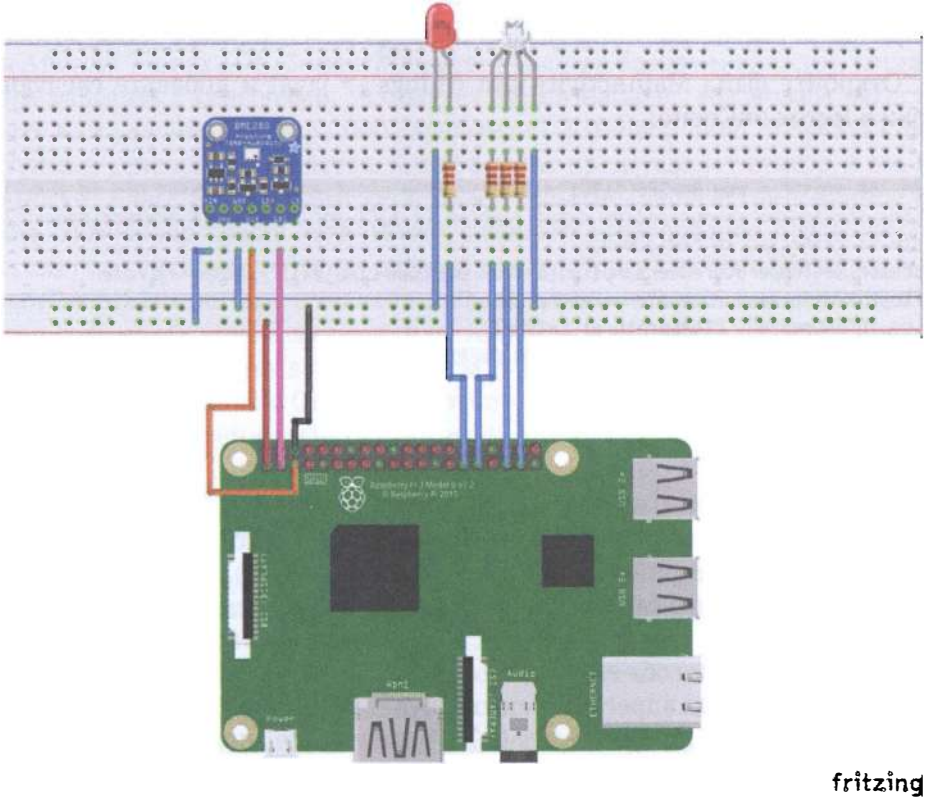


Рис. 118 ❖ Схема подключения для задания 3

2. Откройте файл manifests -> AndroidManifest.xml и добавьте следующую строку перед тегом application:

```
<uses-permission android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

Обычно для использования I2C-периферии нам нужен драйвер. Драйвер – это набор классов, отвечающих за общение между платой с Android Things и периферией. Более того, эти классы обрабатывают специальные протоколы, используемые периферией. Все драйверы, которые официально поддерживаются ОС Android Things, доступны на GitHub в папке **contrib-drivers** по адресу https://github.com/androidthings/contrib-drivers. Откройте файл build.gradle (Module: things) и добавьте следующую строку в раздел dependencies:

```
implementation 'com.google.android.things.contrib:driver-bmx280:1.0'
```

Дополнительно вам придётся поменять minSdkVersion с 24 на 27 в этом файле. Синхронизируйте проект, и теперь вы можете использовать BME280/BMP280-сенсор в проекте. Так как мы используем драйвер для этого сенсора, мы должны добавить следующее разрешение (permission) в файл AndroidManifest.xml перед тегом application:

```
<uses-permission android:name="com.google.android.things.permission.MANAGE_SENSOR_DRIVERS" />
```

3. Откройте файл MainActivity.java (things -> java) и добавьте следующие строки в метод onCreate:

```
try {
    Bmx280 sensor = new Bmx280("I2C1", 0x76); //the class accepts the SDA pin id for
handling the sensor communication details which is I2C1 for Rpi3; 0x76 address means the
SDO pin is connected to the ground
    sensor.setTemperatureOversampling(Bmx280.OVERSAMPLING_1X); //sampling rate
    float temp = sensor.readTemperature();
    Log.i("onCreate", "Temperature: "+temp);
    sensor.setPressureOversampling(Bmx280.OVERSAMPLING_1X);
    float press = sensor.readPressure();
    Log.i("onCreate", "Pressure: "+press);
} catch (IOException e) {
    e.printStackTrace();
}
```

Заметьте, что имя пина – I2C1, а следующее значение установлено в 0x76. Это уникальный адрес устройства, который обычно устанавливается с помощью пина SDO, потому что у каждого периферийного устройства, поддерживающего соединение по I2C, есть свой адрес. Если на вашем сенсоре есть отдельный SDO-пин, есть два способа его подключения. Вы можете подключить этот пин к питанию (Vcc), тогда адрес I2C-сенсора будет 0x77; и вы можете подключить SDO пин к земле, тогда адрес будет 0x76. В коде выше адрес установлен в значение 0x76, что означает, что даже если у вас нет SDO-пина на сенсоре, внутри сенсора этот пин есть, и он подключён к земле. Теперь подключите плату, монитор и запустите приложение: в логе Logcat вы увидите температуру

и давление окружающей вас среды (заметьте, что сенсор показывает значение давления в миллибарах):

```
09-27 12:42:24.507 2132-2132/? I/zygote: Late-enabling -Xcheck:jni
09-27 12:42:24.800 2132-2132/com.example.me.iotbook3 I/InstantRun: starting instant run
server: is main process
09-27 12:42:25.113 2132-2132/com.example.me.iotbook3 I/onCreate: Temperature: 19.92114
09-27 12:42:25.119 2132-2132/com.example.me.iotbook3 I/onCreate: Pressure: 718.10345
```

По желанию вы можете добавить к этим значениям показания влажности; строки кода аналогичны приведённым выше.

4. В приведённом коде мы читаем данные с сенсора только один раз. В нашем проекте мы хотим читать их постоянно. Поэтому удобно будет использовать другой подход. Этот подход аналогичен тому, который мы используем в операционной системе Android, когда приложению нужно отслеживать показания сенсоров смартфона. Как вы знаете, в настоящее время у смартфона есть несколько встроенных сенсоров, и, чтобы читать с них данные, мы используем sensor framework, предоставляемый Android SDK.

Ключевыми элементами этого фреймворка являются SensorManager, Sensor, SensorEvent и SensorEventListener. Эти классы и интерфейсы также входят в состав Android Things SDK. **SensorManager** является базовым классом, когда мы хотим работать с сенсорами. Используя SensorManager, мы можем регистрировать или удалять регистрацию слушателей (listeners) либо получать список доступных сенсоров устройства. Класс **Sensor** представляет сенсор и его возможности. Класс **SensorEvent** описывает событие, зарегистрированное сенсором. Экземпляр класса SensorEvent содержит информацию о сенсоре, данные, снятые сенсором, точность этих данных и метку времени. Наконец, класс **SensorEventListener** представляет callback-класс, который вызывается, когда сенсор считывает новые данные, или точность измерения данных меняется. Мы будем использовать все эти классы и ещё один класс – **SensorManager.DynamicSensorCallback**. Он полезен тогда, когда мы хотим получать уведомления при подключении или отключении от нашей платы *динамического сенсора*.

Откройте файл MainActivity.java (things -> java); закомментируйте предыдущий код и добавьте следующие строки перед методом onCreate:

```
SensorManager sensorManager;
TemperatureCallback tempCallback;
PressureCallback pressCallback;
Bmx280SensorDriver mySensorDriver;
BMX280Callback callback;
```

Теперь нам нужно создать **callback**-класс сенсора. Так как мы хотим получать уведомления о двух параметрах (температуре и давлении), нам нужны два слушателя (listeners), по одному на каждый параметр. Для значений температуры, получаемых от сенсора, callback-класс будет выглядеть так:

```
private class TemperatureCallback implements SensorEventListener {

    @Override
    public void onSensorChanged(SensorEvent event) {
        float val = event.values[0];
        Log.i("TemperatureCallback","Temperature: "+val);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        Log.i("TemperatureCallback","Accuracy: "+accuracy);
    }
}
```

А для значений давления – так:

```
private class PressureCallback implements SensorEventListener {

    @Override
    public void onSensorChanged(SensorEvent event) {
        float val = event.values[0];
        Log.i("PressureCallback","Pressure: "+val);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        Log.i("PressureCallback","Accuracy: "+accuracy);
    }
}
```

Метод **onSensorChanged** вызывается, когда доступно новое значение параметра, прочитанное сенсором; метод **onAccuracyChanged** – когда меняется точность измерений этого параметра.

Теперь необходимо зарегистрировать эти пользовательские callback-классы, чтобы получать события. Это возможно только тогда, когда Android Things-приложение уведомлено о том, что сенсор подключён, иначе нельзя зарегистрировать слушателей событий. Вот здесь-то приложение и использует SensorManager.DynamicSensorCallback для обработки событий. Добавьте следующий код в MainActivity.java:

```
private class BMX280Callback extends SensorManager.DynamicSensorCallback {
    @Override
    public void onDynamicSensorConnected (Sensor sensor) {
        int sensorType = sensor.getType();
        Log.i("BMX280Callback","On Sensor connected...");
        if (sensorType==Sensor.TYPE_AMBIENT_TEMPERATURE) {
            Log.i("BMX280Callback", "Temp sensor...");
            tempCallback = new TemperatureCallback();
            sensorManager.registerListener(tempCallback,sensor,SensorManager.SENSOR_DELAY_
NORMAL);
        } else if (sensorType==Sensor.TYPE_PRESSURE) {
            Log.i("BMX280Callback", "Pressure sensor...");
            pressCallback = new PressureCallback();
```

```
            sensorManager.registerListener(pressCallback,sensor,SensorManager.SENSOR_DELAY_
NORMAL);
        }
    }

    @Override
    public void onDynamicSensorDisconnected (Sensor sensor) {
        super.onDynamicSensorDisconnected(sensor);
    }
}
```

Существует 4 возможных значения для частоты снятия показаний с сенсора: SENSOR_DELAY_NORMAL – задержка около 2 секунд между показаниями; SENSOR_DELAY_UI – задержка в 0.6 секунды; SENSOR_DELAY_GAME – задержка в 0.2 секунды; SENSOR_DELAY_FASTEST – никакой задержки.

5. Чтобы собрать всё вместе, откройте файл MainActivity.java (things -> java) и в методе onCreate добавьте следующее (в коде есть одна строка с ошибкой, найдите её и исправьте в соответствии с вашими знаниями, полученными в предыдущем разделе этого задания):

```
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
callback = new BMX280Callback();
sensorManager.registerDynamicSensorCallback(callback);
try {
    mySensorDriver = new Bmx280SensorDriver(BoardPins.getSDAPin());
    mySensorDriver.registerTemperatureSensor();
    mySensorDriver.registerPressureSensor();
} catch (IOException e) {
    e.printStackTrace();
}
```

Заметьте, что, для того чтобы сделать наше приложение платонезависимым, мы используем новый метод в классе BoardPins.java, который мы создали в предыдущем задании; поэтому вам нужно скопировать этот класс из предыдущего проекта Android Studio и добавить в класс метод getSDAPin:

```
public static String getSDAPin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "I2C1";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}
```

И наконец, нам нужно закрыть соединение с сенсором, чтобы освободить пин SDA, используемый для общения с сенсором, чтобы другие приложения тоже могли через него обратиться к сенсору. Нам нужно удалить регистрацию слушателя значений от сенсора (sensor listener), удалить регистрацию слушателя определения статуса подключения сенсора к Android Things-плате и за-

крыть соединение с сенсором. Для этого добавьте следующий метод onDestroy в конец класса MAinActivity.java:

```
@Override
protected void onDestroy () {
    super.onDestroy();
    Log.i("MainActivity","onDestroy");
    sensorManager.unregisterListener(tempCallback);
    sensorManager.unregisterListener(pressCallback);
    sensorManager.unregisterDynamicSensorCallback(callback);
    try {
        mySensorDriver.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Теперь вы можете подключить плату и монитор к питанию и проверить, работает ли приложение. Сенсор надо придерживать, чтобы был стабильный контакт с пинами, как вы делали это с жидкокристаллическим монитором или RFID-модулем в ходе очередного задания по практике с Arduino, иначе значения, получаемые от сенсора, будут странные (как значения давления в следующем логе). В логе приложения будет видно следующее:

```
09-27 15:05:59.791 1972-1972/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
09-27 15:05:59.792 1972-1972/com.example.me.iotbook3 I/BMX280Callback: Temp sensor...
09-27 15:05:59.807 1972-1972/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Pressure sensor...
09-27 15:05:59.821 1972-1972/com.example.me.iotbook3 I/TemperatureCallback: Accuracy: 3
    Temperature: 29.686142
09-27 15:05:59.840 1972-1972/com.example.me.iotbook3 I/PressureCallback: Accuracy: 3
09-27 15:05:59.841 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure:
729.39264
09-27 15:05:59.856 1972-1972/com.example.me.iotbook3 D/vndksupport: Loading /vendor/lib/
hw/android.hardware.graphics.mapper@2.0-impl.so from current namespace instead of sphal
namespace.
09-27 15:05:59.872 1972-1972/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.701126
09-27 15:05:59.872 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure: 729.4043
09-27 15:05:59.912 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure: 729.4217
09-27 15:05:59.939 1972-1972/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.706121
09-27 15:05:59.955 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure: 729.4217
09-27 15:05:59.982 1972-1972/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.72111
09-27 15:05:59.998 1972-1972/com.example.me.iotbook3 I/PressureCallback: Pressure: 729.4332
```

Теперь вы знаете, как использовать I2C-сенсор в Android Things. Заключительный этап этого задания – в том, чтобы создать пользовательскую (кастомную) логику с помощью светодиодов, которая основана на показаниях сенсора.

6. Трёхцветный светодиод должен показывать текущее состояние по давлению, а красный светодиод показывает, не превысила ли определённый порог температура. Предположим, что есть два порога по давлению: LEVEL_1 = 1022.9 мб, или 767.2 мм рт. ст.; LEVEL_2 = 1009.14 мб, или 756.92 мм рт. ст. Логика нашего приложения будет следующей:

- если текущее давление больше LEVEL_1, то трёхцветный светодиод будет светить зелёным и красным цветом (т. е. жёлтым);
- если текущее давление между LEVEL_1 и LEVEL_2, то трёхцветный светодиод будет светить зелёным цветом;
- если текущее давление ниже LEVEL_2, то трёхцветный светодиод будет светить синим цветом.

Таким образом, трёхцветный светодиод будет представлять прогноз погоды:

- если давление выше LEVEL_1, погода будет стабильной;
- если давление между LEVEL_1 и LEVEL_2, погода будет облачной;
- если давление ниже LEVEL_2, погода будет дождливой.

Красный светодиод будет использоваться как сигнализация. Он будет загораться, если температура в помещении стала выше 30 °C.

Откройте файл MainActivity.java (things -> java) и добавьте следующие переменные перед методом onCreate:

```
PeripheralManager pManager;
Gpio redPin, greenPin, bluePin, redLedPin;
```

В методе onCreate добавьте следующую строку:

```
pManager = PeripheralManager.getInstance();
```

Добавьте следующий метод в тот же класс:

```
private void initRGBPins() {
        try {
            redPin = pManager.openGpio(BoardPins.getRedPin());
            redPin.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            redPin.setActiveType(Gpio.ACTIVE_HIGH);
            greenPin = pManager.openGpio(BoardPins.getGreenPin());
            greenPin.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            greenPin.setActiveType(Gpio.ACTIVE_HIGH);
            bluePin = pManager.openGpio(BoardPins.getBluePin());
            bluePin.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            bluePin.setActiveType(Gpio.ACTIVE_HIGH);
            redLedPin = pManager.openGpio(BoardPins.getRedLedPin());
            redLedPin.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            redLedPin.setActiveType(Gpio.ACTIVE_HIGH);
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

Этот метод открывает сообщение между пинами платы и подключёнными к ним светодиодами, устанавливает тип пинов в режим записи и затем уста-

навливает значения (напряжения) пинов (ACTIVE_HIGH – мы используем это значение для трёх цветов, так как у нас трёхцветный светодиод с общим катодом, поэтому ток течёт от GPIO-пинов платы на землю (пин земли) через общий катод; если у вас трёхцветный светодиод с общим анодом, вам нужно использовать значение ACTIVE_LOW, так как ток будет течь в обратном направлении – от питания 3.3 В через общий анод к GPIO-пинам платы, которые представляют собой землю (пины заземления)).

Как вы, наверное, заметили, мы используем ещё 4 метода из класса BoardPins.java, поэтому вы должны добавить их в него:

```
public static String getRedPin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "BCM6";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}

public static String getBluePin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "BCM26";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}

public static String getGreenPin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "BCM19";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}

public static String getRedLedPin() {
    switch (getBoardName()) {
        case RASPBERRY:
            return "BCM5";
        case EDISON_ARDUINO:
            return "IO4";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}
```

Затем добавьте следующую строку в метод onCreate:

```
initRGBPins();
```

Теперь нам необходимо реализовать логику приложения. Мы сделаем это в методах слушателей сенсора (sensor listener). Сначала изменим код слушателя значений давления от сенсора: добавим в раздел объявления переменных следующее:

```
private static final float LEVEL_1 = 1022.69f; //or 767.2f mm Hg
private static final float LEVEL_2 = 1009.14f; //or 756.92f mm Hg
private int currentWeather = -100;
```

А в метод **onSensorChanged** класса **PressureCallback** – следующее:

```
int newWeather = -200;
if (val >= LEVEL_1)
    newWeather = 1;
else if (val >= LEVEL_2 && val <= LEVEL_1)
    newWeather = 0;
else
    newWeather = -1;

if (newWeather != currentWeather) {
    currentWeather = newWeather;
    // Set the RGB color
    switch (newWeather) {
        case 1:
            setRGBPins(true, true, false);
            break;
        case 0:
            setRGBPins(false, true, false);
            break;
        case -1:
            setRGBPins(false, false, true);
            break;
    }
}
```

У нас нет метода setRGBPins, используемого, чтобы избежать изменения цвета трёхцветного светодиода каждый раз, когда сенсор считывает новое значение. Вместо этого приложение просто проверяет, может изменить цвет светодиода новое значение давления или нет; и если да – меняет с помощью вызова метода setRGBPins:

```
private void setRGBPins(boolean red, boolean green, boolean blue) {
    try {
        Log.i("setRGBPins", "Change RGB led color. Red ["+red+"] - Green ["+green+"] - Blue
["+blue+"]");
        redPin.setValue(red);
        greenPin.setValue(green);
        bluePin.setValue(blue);
    }
```

```
    catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
```

И наконец, мы должны реализовать логику работы красного светодиода. Для этого надо поменять код класса, отвечающего за слушатель показаний температуры. Сначала добавим следующую переменную в раздел объявления переменных:

```
boolean currentState = false;
```

И затем перейдём в метод **onSensorChanged** класса **TemperatureCallback** и допишем следующий код:

```
boolean turnOn = false;
if (val >= 30)
    turnOn = true;
else
    turnOn = false;

if (currentState != turnOn) {
    Log.d("TemperatureCallback", "Change RED led color. New state ["+turnOn+"]");
    try {
        redLedPin.setValue(turnOn);
        currentState = turnOn;
    }
    catch(IOException ioe) {
        ioe.printStackTrace();
    }
}
```

7. Кроме того, можно выводить текущие значения температуры и давления в пользовательский интерфейс приложения – вместо текстового значения Hello World! элемента TextView мы можем добавить ещё один TextView и менять их значения соответственно (things -> res -> layout -> activity_main.xml). Также мы можем добавить простой прогноз погоды, используя третий TextView в соответствии с логикой нашего приложения. После редактирования пользовательского интерфейса добавьте следующие переменные в начало класса MainActivity.java:

```
TextView text1, text2, text3;
```

В методе onCreate добавьте инициализацию этих переменных (помните о том, что надо установить сначала их ID в свойствах):

```
text1 = findViewById(R.id.text1); //temperature
text2 = findViewById(R.id.text2); //pressure
text3 = findViewById(R.id.text3); //simple weather forecast
```

И затем в методах onSensorChanged соответствующих классов-слушателей добавьте:

```
text1.setText("Temperature: "+String.valueOf(Math.round(val))+" C");// to the
TemperatureCallback class
text2.setText("Pressure: "+String.valueOf(Math.round(val))+" millibars"); // to the
PressureCallback class
```

Что касается элемента text3, измените конструкцию if – elseif – else метода onSensorChanged класса PressureCallback следующим образом (можно перевести на русский язык):

```
if (val >= LEVEL_1) {
    newWeather = 1;
    text3.setText("The weather will be stable");
}
else if (val >= LEVEL_2 && val <= LEVEL_1) {
    newWeather = 0;
    text3.setText("The weather will be cloudy");
}
else {
    newWeather = -1;
    text3.setText("The weather will be rainy");
}
```

Теперь можно протестировать наше приложение! Подключите к питанию Android Things-плату и монитор и запустите приложение. В логе Logcat (и в интерфейсе приложения) вы должны увидеть что-то, похожее на следующее:

```
09-28 11:11:58.348 1878-1878/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Temp sensor...
09-28 11:11:58.362 1878-1878/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Pressure sensor...
09-28 11:11:58.376 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Accuracy: 3
09-28 11:11:58.377 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.92688
09-28 11:11:58.395 1878-1878/com.example.me.iotbook3 I/PressureCallback: Accuracy: 3
09-28 11:11:58.396 1878-1878/com.example.me.iotbook3 I/setRGBPins: Change RGB led color.
Red [false] - Green [false] - Blue [true]
09-28 11:11:58.399 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9729
09-28 11:11:58.430 1878-1878/com.example.me.iotbook3 D/vndksupport: Loading /vendor/lib/
hw/android.hardware.graphics.mapper@2.0-impl.so from current namespace instead of sphal
namespace.
09-28 11:11:58.460 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.94686
09-28 11:11:58.462 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.98682
09-28 11:11:58.465 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9318
09-28 11:11:58.472 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.95435
09-28 11:11:58.520 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.01679
09-28 11:11:58.522 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9579
09-28 11:11:58.553 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.061749
```

```
09-28 11:11:58.555 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.95703
09-28 11:11:58.585 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.08173
09-28 11:11:58.617 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9202
09-28 11:11:58.625 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.116695
09-28 11:11:58.656 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.92365
09-28 11:11:58.687 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.121689
09-28 11:11:58.689 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.95514
```

Если температура достигает значения 30 °C (потому что вы придерживаете сенсор рукой), в логе появится соответствующее сообщение об изменении статуса красного светодиода:

```
09-28 11:12:14.684 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.985851
09-28 11:12:14.701 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 980.0391
09-28 11:12:14.731 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.995844
09-28 11:12:14.749 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 980.0391
09-28 11:12:14.786 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.98315
09-28 11:12:14.816 1878-1878/com.example.me.iotbook3 D/TemperatureCallback: Change RED led
color. New state [true]
09-28 11:12:14.817 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.00084
09-28 11:12:14.834 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9271
09-28 11:12:14.872 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.935
09-28 11:12:14.900 1878-1878/com.example.me.iotbook3 D/TemperatureCallback: Change RED led
color. New state [false]
09-28 11:12:14.902 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.995844
09-28 11:12:14.919 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.991
09-28 11:12:14.959 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.96313
09-28 11:12:14.987 1878-1878/com.example.me.iotbook3 D/TemperatureCallback: Change RED led
color. New state [true]
09-28 11:12:14.988 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.005833
09-28 11:12:15.005 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9709
09-28 11:12:15.045 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.991
09-28 11:12:15.089 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 980.0067
09-28 11:12:15.132 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.97876
09-28 11:12:15.175 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 980.0067
09-28 11:12:15.219 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.9866
09-28 11:12:15.249 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.01582
09-28 11:12:15.266 1878-1878/com.example.me.iotbook3 I/PressureCallback: Pressure:
979.96643
09-28 11:12:15.290 1878-1878/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.010828
```

А в пользовательском интерфейсе приложения вы увидите что-то вроде этого: округлённые значения температуры и давления + простой прогноз погоды. Сегодня 28 сентября 2018 года, и дождь идёт целый день, так что прогноз работает! На рис. 119 показан скриншот работающего на Raspberry Pi 3 приложения:

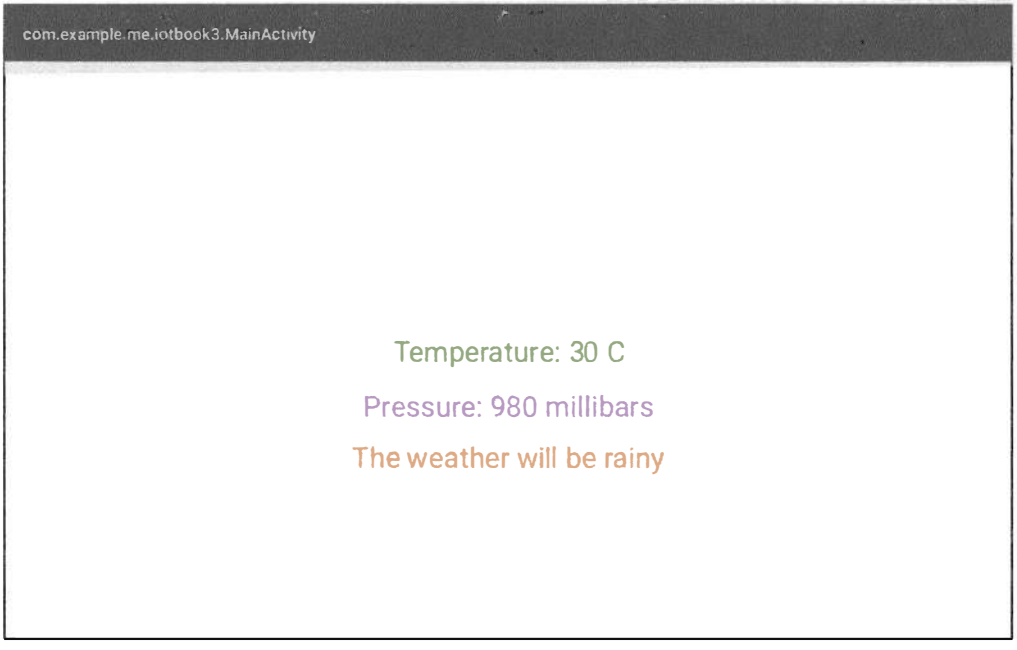


Рис. 119 ❖ Интерфейс Android Things-приложения мониторинга окружающей среды

8. Теперь вы, наверное, думаете, что задание, наконец, выполнено, но – нет! Мы могли бы создать пользовательский (кастомный) драйвер сенсора для класса BmxSensorDriver.java, но он уже содержит способность читать значения влажности с сенсора BME280, так как он содержит внутри класс HumidityUserDriver – вы, вероятно, уже видели эту возможность считывать влажность, когда писали код приложения. Поэтому было бы логично добавить измерение влажности в наше приложение. Вам придётся сделать это самостоятельно! Когда вы закончите, в логе вашего Android Things-приложения будет что-то вроде:

```
09-28 12:25:17.464 1618-1618/com.example.me.iotbook3 I/SensorManager: DYNS native
SensorManager.getDynamicSensorList return 3 sensors
09-28 12:25:17.647 1618-1618/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Temp sensor...
09-28 12:25:17.672 1618-1618/com.example.me.iotbook3 I/BMX280Callback: On Sensor
connected...
    Pressure sensor...
09-28 12:25:17.699 1618-1618/com.example.me.iotbook3 I/BMX280Callback: On Sensor connected...
```

```
    Humidity sensor...
09-28 12:25:17.727 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Accuracy: 3
09-28 12:25:17.728 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
27.987799
09-28 12:25:17.730 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.002787
09-28 12:25:17.730 1618-1618/com.example.me.iotbook3 I/PressureCallback: Accuracy: 3
09-28 12:25:17.732 1618-1618/com.example.me.iotbook3 I/setRGBPins: Change RGB led color.
Red [false] - Green [false] - Blue [true]
09-28 12:25:17.745 1618-1618/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.87885
09-28 12:25:17.780 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
28.042746
09-28 12:25:17.782 1618-1618/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.886
09-28 12:25:17.782 1618-1618/com.example.me.iotbook3 I/HumidityCallback: Accuracy: 3
09-28 12:25:17.783 1618-1618/com.example.me.iotbook3 I/humidityCallback: Humidity: 68.94584
09-28 12:27:42.581 1618-1618/com.example.me.iotbook3 I/humidityCallback: Humidity: 69.07395
09-28 12:27:42.599 1618-1618/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.907
09-28 12:27:42.615 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
29.995844
09-28 12:27:42.631 1618-1618/com.example.me.iotbook3 I/humidityCallback: Humidity: 69.07331
09-28 12:27:42.634 1618-1618/com.example.me.iotbook3 I/PressureCallback: Pressure: 979.935
09-28 12:27:42.650 1618-1618/com.example.me.iotbook3 D/TemperatureCallback: Change RED led
color. New state [true]
09-28 12:27:42.651 1618-1618/com.example.me.iotbook3 I/TemperatureCallback: Temperature:
30.00084
```

И интерфейс вашего приложения будет выглядеть, как показано на рис. 120.

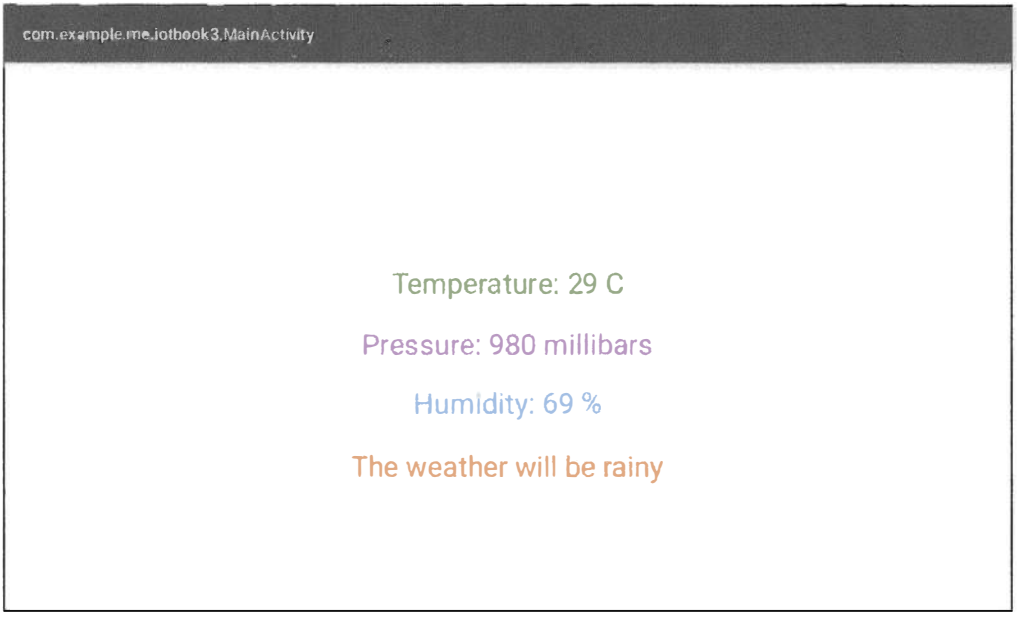


Рис. 120 ❖ Интерфейс Android Things-приложения мониторинга окружающей среды с влажностью

Четвёртый проект в Android Things – объединение Android Things с облачной платформой интернета вещей

0. В этом проекте мы научимся интегрировать ОС Android Things с облачными платформами интернета вещей (IoT cloud platforms). Это – важный аспект при разработке приложения для интернета вещей. Существует несколько сценариев, когда требуется передавать в облако данные, полученные от Android Things-плат. Мы изучим архитектуру облака интернета вещей, научимся конфигурировать облачную платформу интернета вещей, подключать Android Things-приложение к этой платформе и посылать (стримить) данные в реальном времени в облако, создавая инструментальные панели с графиками для анализа данных (dashboards).

К этому моменту мы научились разрабатывать Android Things-приложения, которые самодостаточны: они не взаимодействуют с внешними системами или платформами (кроме того, приложения с Google Firebase). Все данные, полученные через сенсоры, обрабатываются локально. Но обычно приложения для интернета вещей посылают данные в облако, поэтому облачные платформы интернета вещей играют важную роль. В настоящее время облачные платформы интернета вещей являются важной частью в экосистеме интернета вещей. Используя эти платформы, мы можем расширить сервисы, которые предлагаем пользователю, и раскрыть всю мощь плат с Android Things. Как только данные доступны на уровне облака интернета вещей, эти платформы могут обеспечить комплексный анализ, где требуется большая вычислительная мощность, применяя такие технологии, как машинное обучение, искусственный интеллект и анализ больших данных – задачи, которые невозможно выполнить на Android Things-платах. Но без последних не было бы данных для анализа.

Обычная облачная платформа интернета вещей предоставляет следующий набор сервисов: **сервис соединения/подключения, сервис хранения данных, сервис обработки событий, управление устройствами, визуализация данных, интеграция сервисов**.

Ядром сервиса подключения являются соединение и передача данных между облачной платформой интернета вещей и удалённой платой. Платформы поддерживают различные протоколы для упрощения процесса подключения: Rest API и HTTP, MQTT, CoAP и т. д. Другими словами, платформы предоставляют набор программных интерфейсов, которые могут быть использованы удалёнными платами интернета вещей для подключения и обмена данными. Кроме этого, они предоставляют набор SDK для различных плат, чтобы сделать процесс подключения быстрым и более простым.

Сервис хранения данных служит для хранения данных в облаке. Эти данные являются основой для других сервисов.

Эти первые два сервиса предоставляются почти всеми облачными платформами интернета вещей, в то время как сервис обработки событий является более сложным. Такой сервис основан на правилах и использует сохранённые

данные и события для запуска действий, которые могли бы иметь влияние на платы интернета вещей. Обычно все события и действия конфигурируются через веб-интрерфейс.

Сервис управления устройствами берёт на себя управление всеми устройствами интернета вещей, подключёнными к платформе. Иными словами, это централизованная административная консоль для удалённых устройств.

Визуализация данных – это сервис, предоставляемый некоторыми облачными платформами интернета вещей для создания инструментальных панелей (dashboards) с целью графической визуализации полученных данных с помощью гистограмм/графиков.

Сервис интеграции полезен, когда мы хотим интегрировать какие-то внешние сервисы типа сообщений по электронной почте, сообщений в Твиттере и т. д. и запускать их в соответствии с заранее сконфигурированными событиями.

На рынке существует несколько облачных платформ интернета вещей, например (совсем небольшой пример): Google IoT cloud, Microsoft Azure IoT, Amazon AWS IoT, Samsung Artik Cloud, Temboo, Ubidots.

Архитектура интернета вещей обычно следующая: уровень сенсоров -> платы интернета вещей -> облачные платформы интернета вещей -> высокоуровневые сервисы с пользовательским интерфейсом для конечного пользователя. Иногда 3-й и 4-й слои смешаны друг с другом.

Чтобы использовать облачную платформу интернета вещей, нам нужно сделать два шага: сконфигурировать проект интернета вещей в облачной платформе, предоставив всю информацию, включая тип данных, и создать клиента облачной платформы интернета вещей (Android Things-приложение), который обрабатывает подключение и посылает данные.

В качестве облачной платформы интернета вещей мы будем использовать Samsung Artik Cloud (https://artik.cloud). Это профессиональная платформа, предоставляющая почти все упомянутые выше сервисы. Она проста в использовании и предоставляет несколько SDK, которые упрощают процесс обмена данными. Мы вручную реализуем обмен данными между Android Things-платой и облаком Samsung Artik, используя его Rest API-механизм.

1. Создайте бесплатный аккаунт в Samsung Artik Cloud: перейдите по ссылке https://my.artik.cloud/ и затем зарегистрируйтесь (sign up) и залогиньтесь (sign in). После этого перейдите по адресу https://developer.artik.cloud/ и нажмите на кнопку **Create your first device type**. Заполните поля **device display name** и **unique name**, например как показано на рис. 121.

Рис. 121 ❖ Создание типа устройства (device type)

Нажмите на кнопку **Set up now**, чтобы настроить ваш аккаунт для разработки. В следующем окне выберите, какой проект у вас будет – групповой (**Team**) или индивидуальный (**Individual**) **project**, затем введите название своей должности и, наконец, введите информацию о вашей организации и метоположении, нажмите **Complete**. После этого вы увидите следующее сообщение в окне, показанном на рис. 121, и сможете нажать на кнопку **Create device type** – она станет активной.

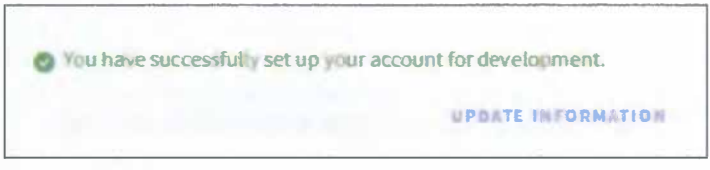


Рис. 122 ❖ Сообщение об успешном завершении конфигурации аккаунта в Artik Cloud

Возможно, вам придётся поменять поле **unique name**, если уникальное имя уже занято. После успешного нажатия на кнопку **Create device type** вы увидите следующее окно.

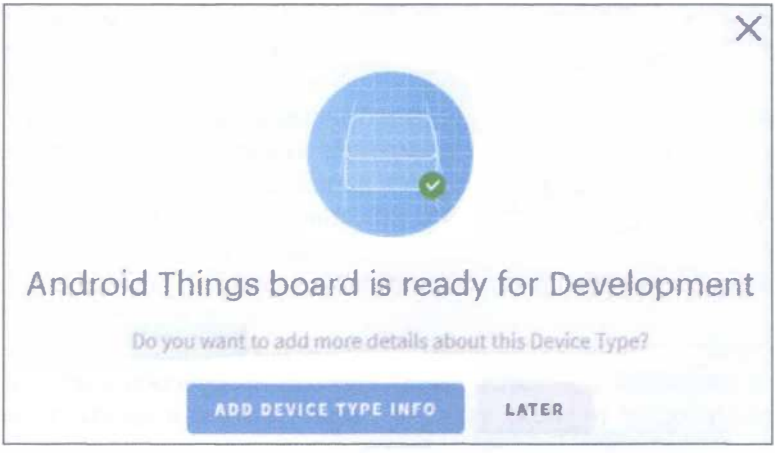


Рис. 123 ❖ Окно с сообщением о готовности системы к поддержке приложений для Android Things-платы

В этом окне нажмите кнопку **Later** и затем нажмите кнопку **New manifest**. Когда мы настраиваем манифест, мы уведомляем Artik Cloud о данных, которые наше приложение будет посылать, чтобы облачная платформа могла извлечь данные, посылаемые Android Things-приложением. В нашем предыдущем проекте мы измеряли три параметра: температуру, давление и влажность. Поэтому мы должны сконфигурировать три переменные в манифесте. Сначала мы настроим поле температуры.

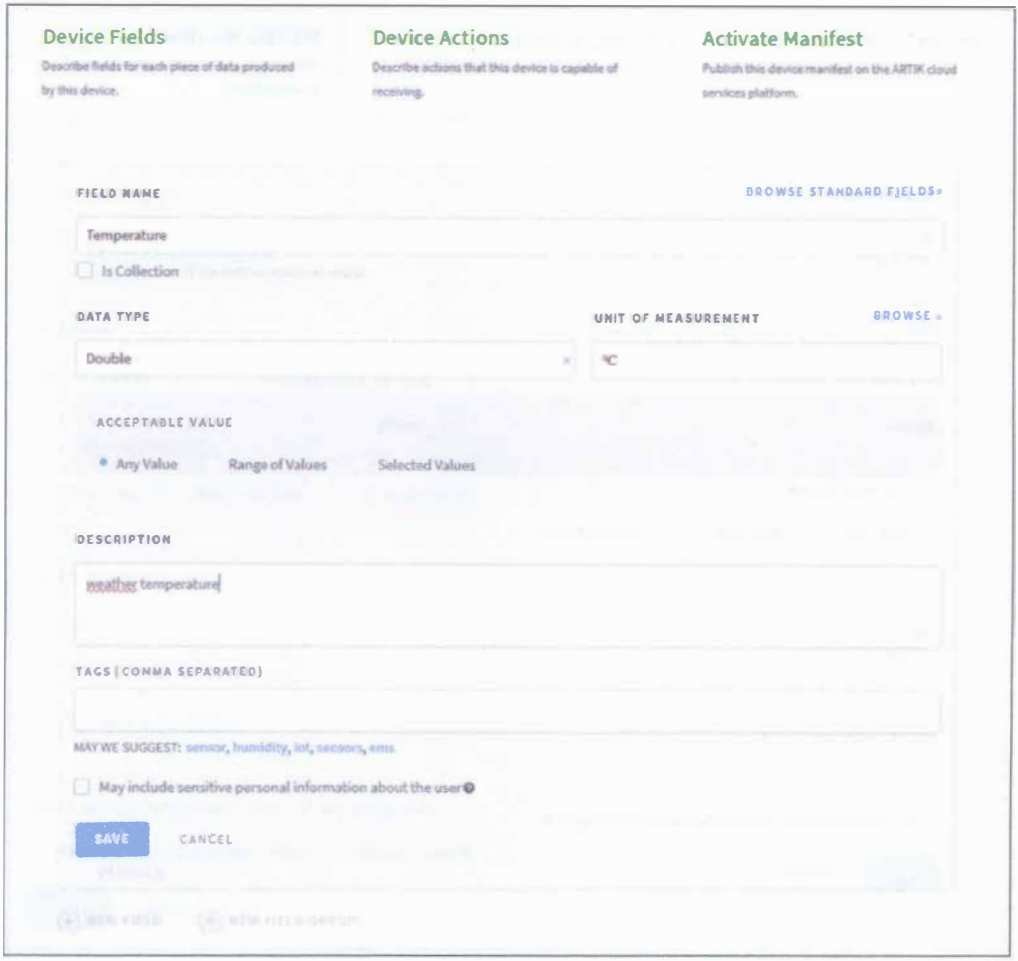


Рис. 124 ❖ Настройка поля/переменной температуры

Самое главное поле в этом окне – это поле **name**, которое мы используем для ссылки на переменную. Нажмите **Save** и затем нажмите **New Field** для переменной давления.

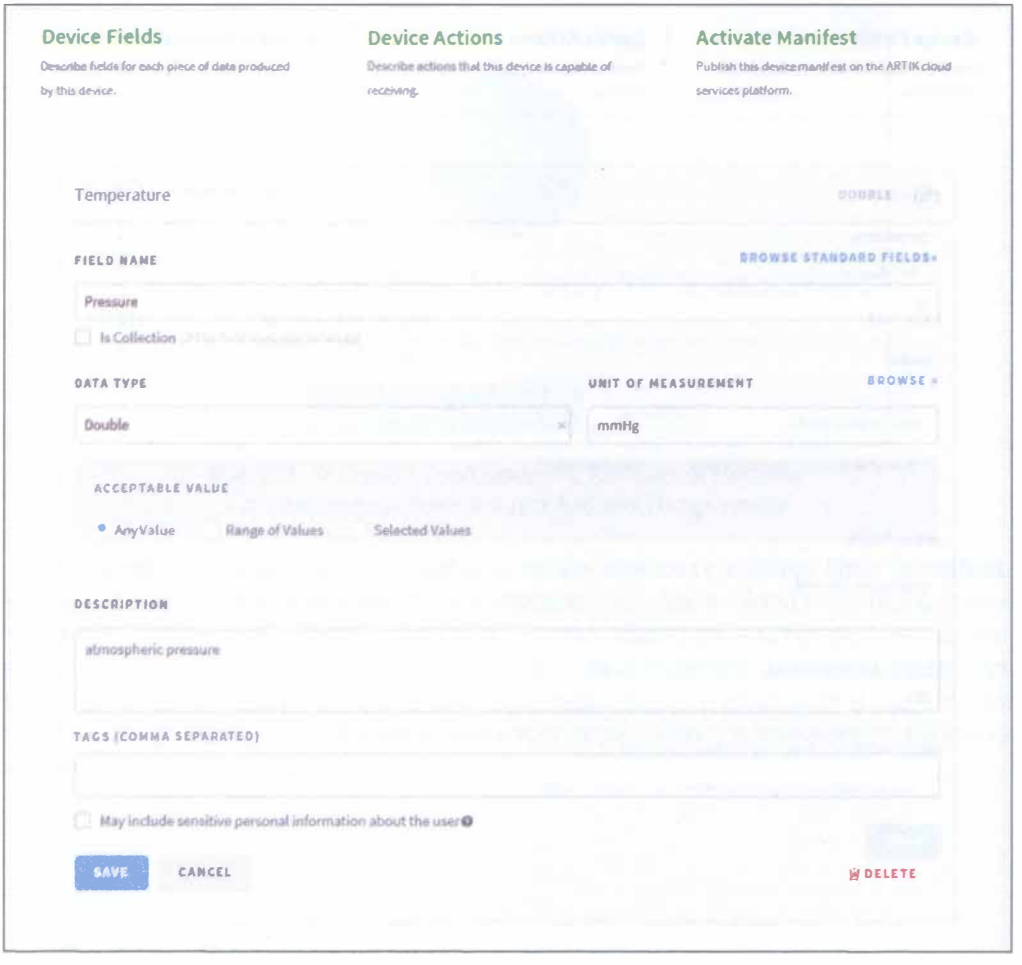


Рис. 125 ❖ Настройка поля/переменной давления

Далее нажмите кнопку **Save** и проделайте то же самое с влажностью.

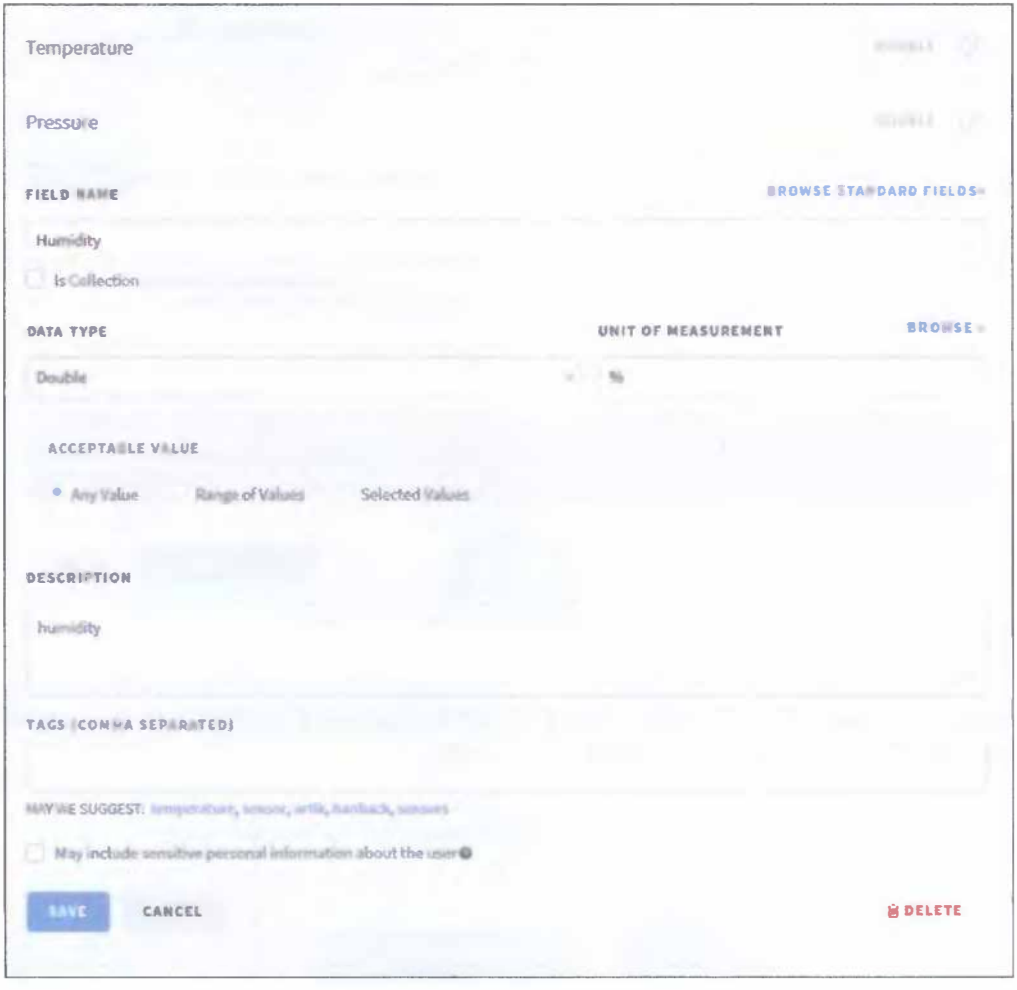


Рис. 126 ❖ Настройка поля/переменной влажности

Нажмите **Save**. После этого нажмите на закладку **Activate Manifest** и затем нажмите на кнопку **Activate Manifest**.

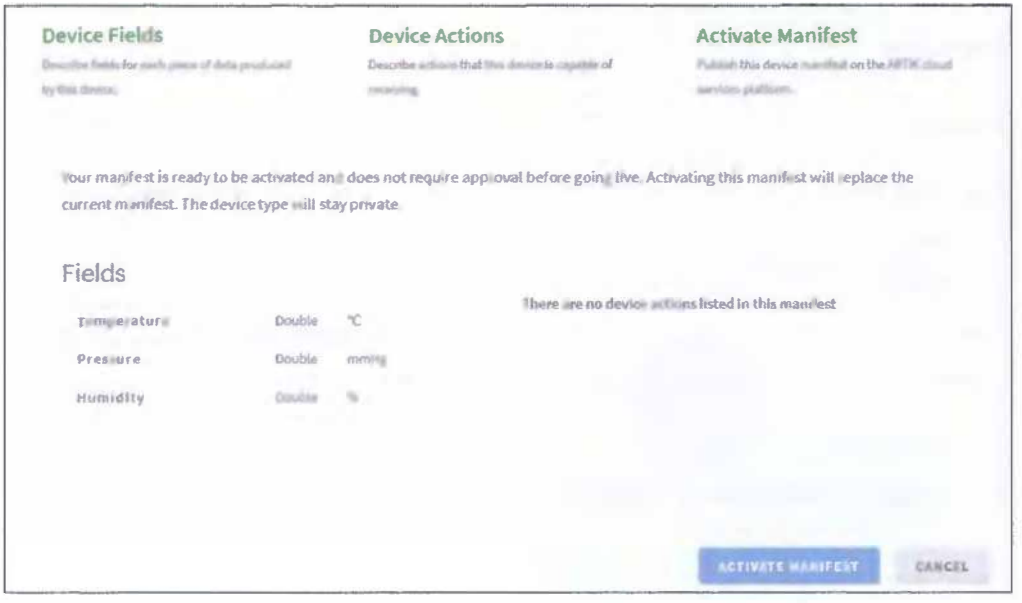


Рис. 127 ❖ Закладка Activate Manifest

Теперь нажмите **на My Cloud** вверху справа, чтобы перейти в панель инструментов (dashboard). На странице **Devices** вы увидите тип устройства вашей организации, который мы только что создали.

Рис. 128 ❖ Тип устройства в панели инструментов

Устройство – это экземпляр типа устройства, который мы создали. Разместите указатель мыши поверх этого типа и нажмите **Select**. Назовите ваше новое устройство, например:

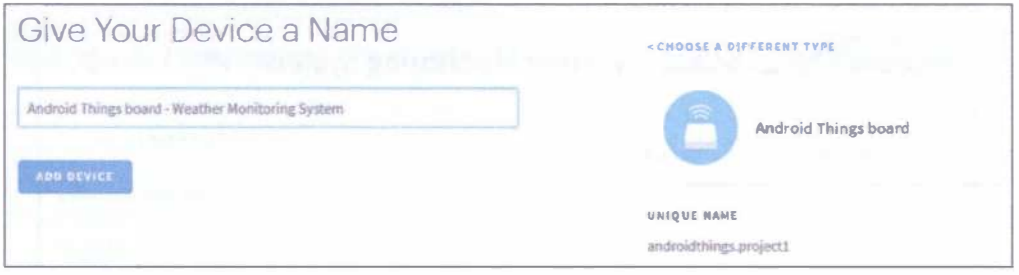


Рис. 129 ❖ Окно добавления нового устройства

И затем нажмите на кнопку **Add Device**, в результате откроется окно, изображённое на рис. 130.

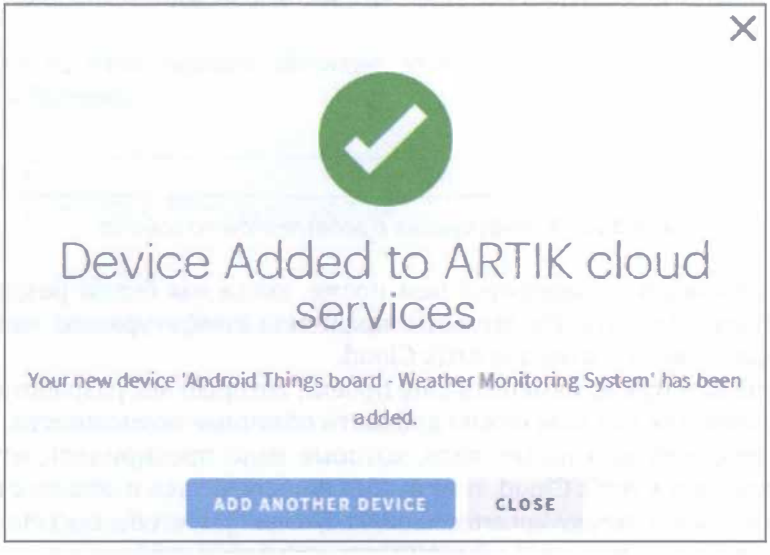


Рис. 130 ❖ Окно с сообщением об успешном добавлении нового устройства

Нажмите кнопку **Close**. Теперь если вы нажмёте на созданное устройство, то увидите следующее.

Рис. 131 ❖ Информация о добавленном устройстве

Эта информация понадобится нам позже, когда мы будем разрабатывать Android Things-клиента. На этом мы закончили конфигурацию нашего проекта интернета вещей в облаке Artik Cloud.

2. Теперь нам нужно изменить наш проект, который мы разработали в прошлом задании, так как нам нужно добавить облачные возможности. Но перед этим всё ещё есть некоторые шаги, которые надо предпринять, чтобы подключить клиента к Artik Cloud. Нам нужно подключиться к облаку с помощью следующей ссылки: **https://api.artik.cloud/v1.1/messages**, чтобы посылать данные об аутентификации нашего Android Things-клиента и сообщение, содержащее данные с сенсора и другую информацию. Затем для аутентификации клиента заголовок http-запроса должен содержать следующую строку: **Authorization: Bearer device_token**, поэтому нам нужен токен устройства (device token). Мы можем получить его, если нажмём на ссылку **Generate device token** во всплывающем окне, показанном на рис. 131. Сообщение, которое клиент посылает в облако, также должно обладать определённой структурой. Чтобы знать его структуру, перейдите в консоль разработчика (**https://developer.artik.cloud/**), нажмите на **Device Types** и затем – на **Android Things board**. Нажмите на кнопку **Edit Info**, затем – на пункт меню **Manifest** (в меню слева), и вы должны увидеть что-то вроде следующего:

Рис. 132 ❖ Информация из манифеста

Нажмите на **View Sample Message**, чтобы знать, как выглядит структура данных сообщения.

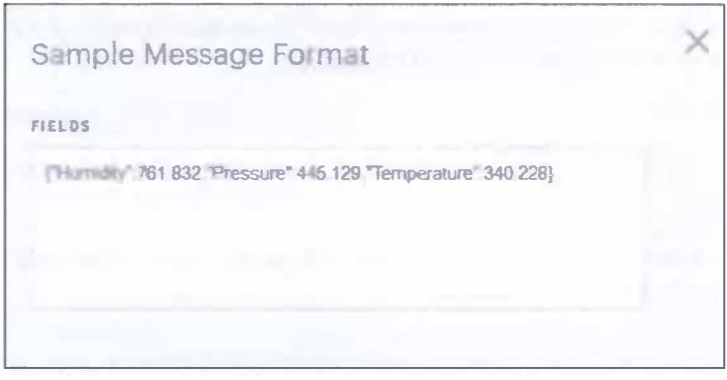


Рис. 133 ❖ Окно со структурой сообщения

Эти данные сенсора должны содержаться внутри **сообщения**, которое является обёрткой, добавляющей другую информацию к реальным данным и выглядящей так:

```
{
    "sdid":"device_id",
    "ts":timestamp,
    "data":
    {
        "Pressure":765.111,
        "Temperature":30.659,
        "Humidity":65.345
    }
}
```

Здесь device_id – это уникальный идентификатор устройства, а ts – метка времени. Теперь мы знаем, как построить сообщение и как извлечь параметры аутентификации.

2.1. Сейчас мы разработаем Android Things Artik-клиента. Мы используем Android Things-приложение из предыдущего практического задания. Мы можем использовать как библиотеку Android HTTP, так и пользовательскую (кастомную) библиотеку для обработки http-соединений. В этом проекте мы будем использовать библиотеку Volley (https://developer.android.com/training/volley/). Эта библиотека широко используется и предлагает интересные возможности. Также она упрощает управление http-соединением.

Откройте файл build.gradle(Module: things) и добавьте следующую строку в раздел dependencies:

```
implementation 'com.android.volley:volley:1.1.0'
```

Добавьте следующую строку в файл AndroidManifest.xml file, чтобы получить доступ к интернету через приложение:

```
uses-permission android:name="android.permission.INTERNET" />
```

Теперь создадим новый класс ArtikClient.java в папке things -> java -> packagename (в моём случае название пакета – com.example.me.IoTbook4) для обработки всех деталей обмена сообщениями.

```
public class ArtikClient {

private static String TAG = "Artik";
private static ArtikClient me;
private Context ctx;
private RequestQueue queue;
private static final String ARTIK_URL = "https://api.artik.cloud/v1.1/messages";
private String deviceId;
private String token;

    private ArtikClient(Context ctx, String deviceId, String token) {
        this.ctx = ctx;
        this.deviceId = deviceId;
        this.token = token;
        createQueue();
    }

}
```

ArtikClient – это синглтон (singleton), и он принимает Android Context, ID устройства и параметры токена, упомянутые ранее. Теперь нам нужно создать createQueue()-метод, который инициализирует очередь запросов Volley, чтобы приложение могло посылать запросы к Artik Rest API.

```
private void createQueue() {
    if (queue == null)
        queue = Volley.newRequestQueue(ctx.getApplicationContext());
}
```

Добавьте метод getInstance – нам он понадобится позже.

```
public static final ArtikClient getInstance(Context ctx, String deviceId, String token) {
    if (me == null)
        me = new ArtikClient(ctx,deviceId,token);
    return me;
}
```

Следующий метод, который надо добавить, – это sendData():

```
public void sendData(final double temp, final double press, final double hum) {

}
```

Он вызывает Artik Rest API, используя структуру сообщения, описанную ранее. Первый шаг добавления этого метода – создание экземпляра StringRequest, который представляет наш http-запрос. Добавьте следующее в метод sendData():

```
StringRequest request = new StringRequest(Request.Method.POST,
        ARTIK_URL,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                Log.d(TAG, "Response ["+response+"]");
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                error.printStackTrace();
            }
        }) {
```

Второй шаг – переопределить метод getHeaders(), чтобы настроить (кастомизировать) заголовки http-запроса, так как приложение должно посылать параметр заголовка Authorization, как предписано Artik-спецификациями:

```
@Override
public Map<String, String> getHeaders() throws AuthFailureError {
    Log.d(TAG, "Get headers..");
    Map<String, String> headers = new HashMap<String, String>();
    headers.put("Content-Type", "application/json");
    headers.put("Authorization", "Bearer " + token);
    return headers;
}
```

И третий шаг – переопределить метод getBody(), чтобы настроить тело сообщения, которое мы посылаем:

```
@Override
public byte[] getBody() throws AuthFailureError {
    Log.d(TAG, "Creating body...");

    try {
```

```
            JSONObject jsonRequest = new JSONObject();
            jsonRequest.put("sdid", deviceId);
            jsonRequest.put("ts", System.currentTimeMillis());
            JSONObject data = new JSONObject();
            data.put("Temperature", temp);
            data.put("Pressure", press);
            data.put("Humidity", hum);
            jsonRequest.put("data", data);

            String sData = jsonRequest.toString();
            Log.d(TAG, "Body:" + sData);

            return sData.getBytes();
        } catch (JSONException jsoe) {
            jsoe.printStackTrace();

        }
        return "".getBytes();
    }

};
```

Этот метод использует библиотеку JSON для построения JSON-сообщения. Он добавляет к сообщению все параметры, рассмотренные ранее.

Последний шаг для подготовки Artik-клиента – это добавление запроса в очередь, чтобы Volley смогла обработать его:

```
queue.add(request);
```

Эта строка завершает метод sendData().

2.2. Наконец, нам надо посылать данные из Android Things-приложения. Чтобы делать это, нам надо вызывать клиента из класса MainActivity.java (things -> java -> package name). Самый простой способ посылать данные в Artik Cloud – это использовать его API каждый раз, когда сенсор считывает новое значение. Это потребовало бы почти бесконечного обращения к Artik API. Поэтому, чтобы не перегружать Artik Cloud, лучший подход – это использование планировщика для отправки данных. С помощью планировщика Android Things-приложение посылает данные с определёнными временными интервалами. Мы можем настроить частоту отправки данных, тем самым больше влияя на поведение приложения и полосу пропускания данных, которую потребляет приложение.

Добавьте следующие переменные в секцию объявления переменных класса MainActivity.java (**поменяйте значения переменных device_id и token на значения вашего устройства**):

```
private static final int TIMEOUT = 1;
private static final double FACTOR = 0.750061561303;
private int tempCounter = 0;
private int pressCounter = 0;
private int humCounter = 0;
private double totalTemp;
private double totalPress;
```

```
private double totalHum;
private static String DEVICE_ID = "3c1e4c0b24f645258400e6e553bcc817";
private static String TOKEN = "04b92cb5070f48aba72f1e030325c6f9";
```

Мы должны немного поменять наши callback-классы. В классе TemperatureCallback в методе OnSensorChanged добавьте следующие строки в конец:

```
totalTemp += val;
tempCounter++;
```

В классе PressureCallback в методе OnSensorChanged добавьте следующие строки:

```
totalPress += val;
pressCounter++;
```

И в классе HumidityCallback в методе OnSensorChanged добавьте следующее:

```
totalHum += val;
humCounter++;
```

И затем добавьте следующий метод в класс MainActivity.java:

```
private void initScheduler() {
    ScheduledExecutorService scheduler =
            Executors.newSingleThreadScheduledExecutor();

    scheduler.scheduleAtFixedRate(new Runnable() {
        @Override
        public void run() {
            //Log.d(TAG, «Scheduler running...»);
            double mTemp = totalTemp / tempCounter;
            double mPress = totalPress / pressCounter * FACTOR;
            double mHum = totalHum / humCounter;

            totalTemp = 0;
            totalPress = 0;
            totalHum = 0;
            tempCounter = 0;
            pressCounter = 0;
            humCounter = 0;
            // call artik
            ArtikClient.getInstance(MainActivity.this,
                    DEVICE_ID, TOKEN).sendData(mTemp, mPress, mHum);

        }
    }, 1, TIMEOUT, TimeUnit.MINUTES);
}
```

В этом классе мы используем SchedulerExecutorService, который запускает конкретную задачу всё время с задержкой, указанной в переменной TIMEOUT. Задача определяется в методе run(). В этом методе приложение выполняет следующие шаги: вычислить среднее значение температуры за интервал времени между временем последней отправки данных в Artik и текущим временем; таким же образом вычислить средние значения давления и влажности; преобра-

зовать значение давления из миллибаров в мм рт. ст., которые требуются в консоли Artik Cloud; вызвать ArtikClient для отправки средних значений; сбросить общее значение переменных и общий счётчик полученных значений температуры, давления и влажности. И последний шаг – вызвать этот метод, чтобы запланировать задачу: в методе onCreate() добавьте следующую строку в конце:

```
initScheduler();
```

Можно видеть, что переменная TIMEOUT установлена в 1 минуту (потому что присутствует константа TimeUnit.***MINUTES***, и TIMEOUT равна 1). Вы можете изменить эти значения, если хотите. Теперь вы можете подключить к питанию плату и монитор и запустить приложение. В логе Logcat вы увидите примерно следующее:

```
2018-10-04 16:23:18.680 1588-1588/com.example.me.iotbook4 I/BMX280Callback: On Sensor
connected...
2018-10-04 16:23:18.680 1588-1588/com.example.me.iotbook4 I/BMX280Callback: Temp sensor...
2018-10-04 16:23:18.695 1588-1588/com.example.me.iotbook4 I/BMX280Callback: On Sensor
connected...
2018-10-04 16:23:18.695 1588-1588/com.example.me.iotbook4 I/BMX280Callback: Pressure
sensor...
2018-10-04 16:23:18.710 1588-1588/com.example.me.iotbook4 I/BMX280Callback: On Sensor
connected...
2018-10-04 16:23:18.710 1588-1588/com.example.me.iotbook4 I/BMX280Callback: Humidity
sensor...
2018-10-04 16:23:18.725 1588-1588/com.example.me.iotbook4 I/TemperatureCallback: Accuracy:
3
2018-10-04 16:23:18.727 1588-1588/com.example.me.iotbook4 I/TemperatureCallback:
Temperature: 27.84794
2018-10-04 16:23:18.727 1588-1588/com.example.me.iotbook4 I/PressureCallback: Accuracy: 3
2018-10-04 16:23:18.730 1588-1588/com.example.me.iotbook4 I/setRGBPins: Change RGB led
color. Red [false] - Green [false] - Blue [true]
2018-10-04 16:23:18.733 1588-1588/com.example.me.iotbook4 I/PressureCallback: Pressure:
727.2553
2018-10-04 16:23:18.757 1588-1588/com.example.me.iotbook4 I/TemperatureCallback:
Temperature: 27.852932
2018-10-04 16:23:18.759 1588-1588/com.example.me.iotbook4 I/PressureCallback: Pressure:
979.8375
2018-10-04 16:23:18.759 1588-1588/com.example.me.iotbook4 I/HumidityCallback: Accuracy: 3
2018-10-04 16:23:18.760 1588-1588/com.example.me.iotbook4 I/humidityCallback: Humidity:
68.93547
2018-10-04 16:23:19.619 1588-1626/com.example.me.iotbook4 D/Artik: Get headers..
2018-10-04 16:23:19.688 1588-1626/com.example.me.iotbook4 D/Artik: Creating body...
2018-10-04 16:23:19.692 1588-1626/com.example.me.iotbook4 D/Artik: Body:{"sdid":"3c1e4c0b2
4f645258400e6e553bcc817","ts":1538659399688,"data":{"Temperature":27.98967432975769,"Press
ure":725.0232714798924,"Humidity":50.68603856940018}}
2018-10-04 16:23:19.708 1588-1588/com.example.me.iotbook4 I/humidityCallback: Humidity:
49.181366
2018-10-04 16:23:19.709 1588-1588/com.example.me.iotbook4 I/TemperatureCallback:
Temperature: 28.09769
2018-10-04 16:23:19.712 1588-1588/com.example.me.iotbook4 I/PressureCallback: Pressure:
979.8723
```

```
2018-10-04 16:23:21.313 1588-1588/com.example.me.iotbook4 D/Artik: Response [{"data":{"mid
":"c5b4bf2ba7464f8d97edc2114aafc961"}}]
2018-10-04 16:23:29.545 1588-1627/com.example.me.iotbook4 D/Artik: Get headers..
2018-10-04 16:23:29.549 1588-1627/com.example.me.iotbook4 D/Artik: Creating body...
2018-10-04 16:23:29.552 1588-1588/com.example.me.iotbook4 I/humidityCallback: Humidity:
50.674202
2018-10-04 16:23:29.553 1588-1627/com.example.me.iotbook4 D/Artik: Body:{"sdid":"3c1e4c0b2
4f645258400e6e553bcc817","ts":1538659409550,"data":{"Temperature":28.236814498901367,"Pres
sure":734.9934985935055,"Humidity":51.02160032171952}}
2018-10-04 16:23:29.577 1588-1588/com.example.me.iotbook4 I/PressureCallback: Pressure:
979.9393
2018-10-04 16:23:29.593 1588-1588/com.example.me.iotbook4 I/humidityCallback: Humidity:
50.696575
2018-10-04 16:23:29.605 1588-1588/com.example.me.iotbook4 I/TemperatureCallback:
Temperature: 28.322474
2018-10-04 16:23:30.336 1588-1588/com.example.me.iotbook4 D/Artik: Response [{"data":{"mid
":"78b00da261fb43ccba50ac3f0b7cbcdf"}}]
```

Вы увидите, что от платформы Artik Cloud приходит ответ (response), уведомляющий приложение, что посланные данные приняты облаком Artik. Таким образом, можно переслать достаточно много показаний, которые позже можно анализировать в облаке.

Вы можете проверить значения, посланные в Artik Cloud, и можете создать панель инструментов (dashboard). Войдите в аккаунт Artik Cloud, перейдите по адресу https://my.artik.cloud/ и выберите **Charts** в меню сверху. Вы увидите следующее всплывающее окно.

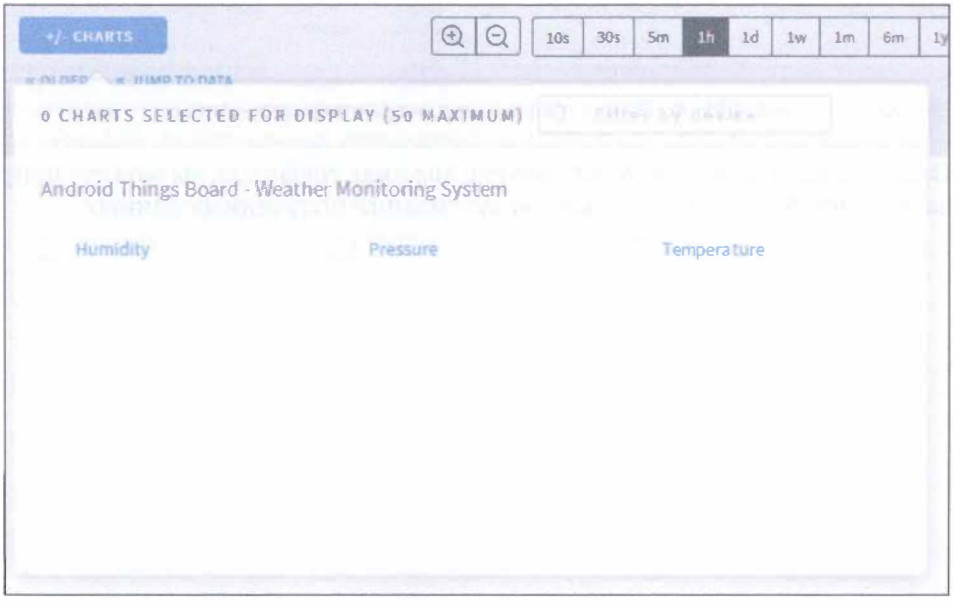


Рис. 134 ❖ Окно выбора переменных для отображения на графиках

Здесь вы можете добавить переменные, которые вас интересуют, которые мы создали в начале этого практического задания. Затем вы увидите данные, и вы можете поменять период времени, чтобы точно установить период, когда данные отсылались в облако, – это сделает график более понятным и наглядным, например таким:

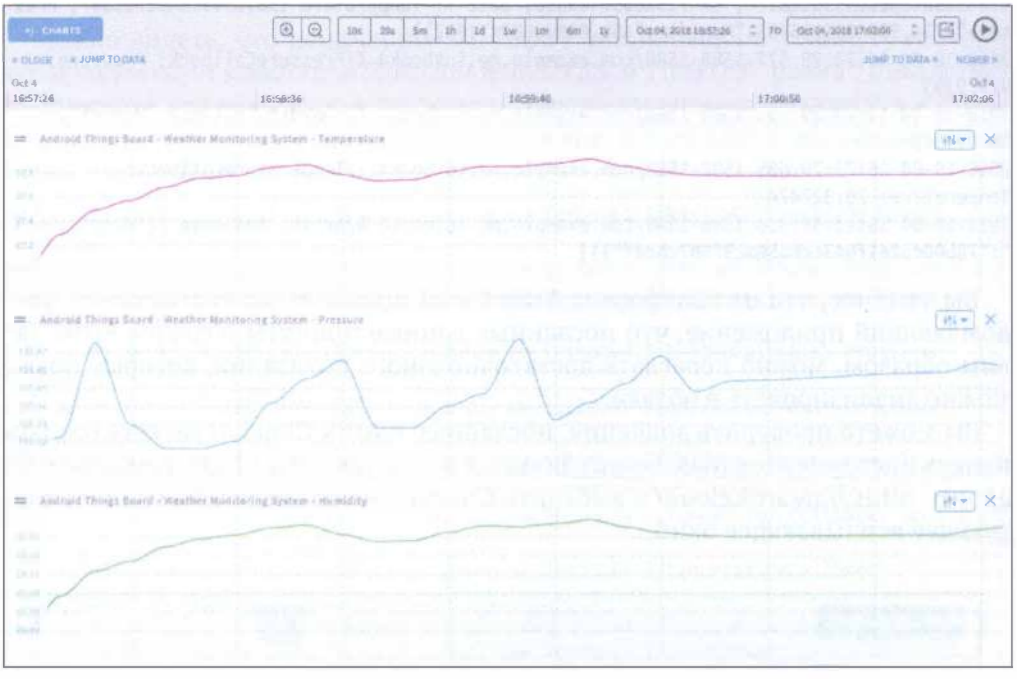


Рис. 135 ❖ Окно с графиками для всех трёх переменных

Около кнопки закрытия **X** есть другие два вида графиков, вы можете попробовать их, чтобы добиться лучшей визуализации полученных данных.

Что касается точных данных, отосланных в облако, – вы можете посмотреть их, используя пункт **Data Logs** верхнего меню. Логи данных (Data Logs) показывают вам все запросы, которые сделал Android Things-клиент в облако Artik Cloud.

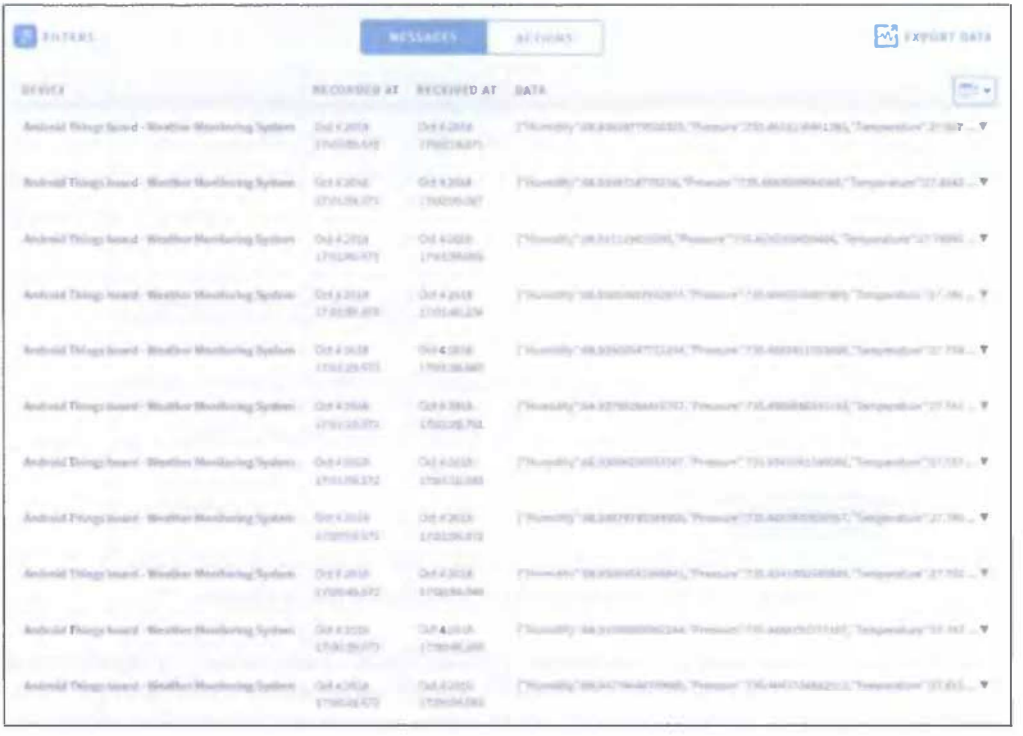


Рис. 136 ❖ Логи данных, переключённые на просмотр сообщений от клиента

Также вы можете экспортировать данные:

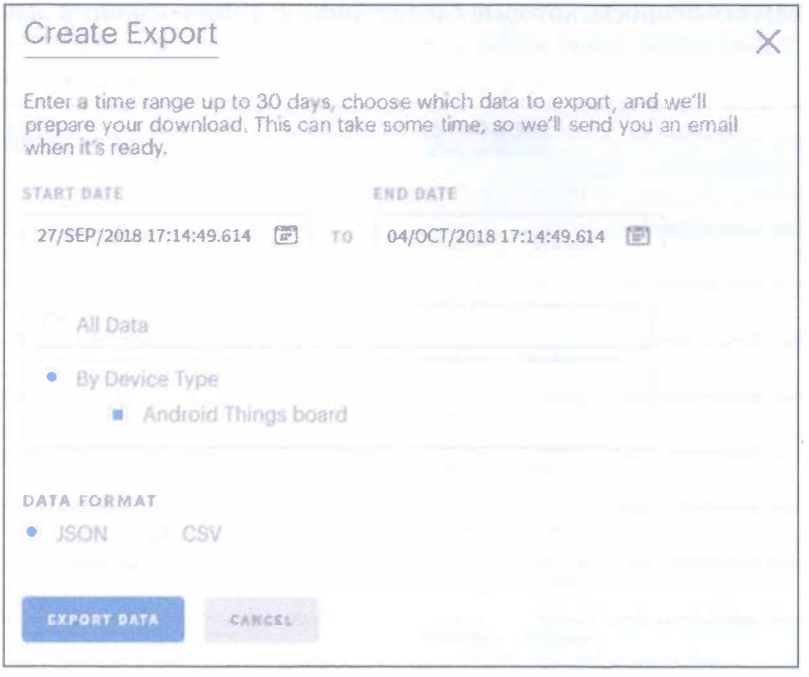


Рис. 137 ❖ Возможности Artik Cloud по экспорту данных

3. Эта часть задания посвящена добавлению голосовых способностей в Android Things. Платформы интернета вещей предлагают различные типы сервисов, не только для хранения данных. Есть некоторые платформы интернета вещей, которые предоставляют сервисы интеграции. Другими словами, они не сфокусированы на получении данных от сенсоров и их хранении, их цель – предоставление сервисов интеграции с другими облачными системами. Одна из таких платформ – **Temboo** (https://temboo.com/). Она предлагает большой набор сервисов интеграции, которые могут быть использованы, чтобы расширить возможности приложения интернета вещей. Temboo поддерживает несколько языков программирования и платформ интернета вещей, включая ОС Android.

Что мы хотим сделать – так это добавить голосовые возможности в наше Android Things-приложение, чтобы оно задействовало голосовой звонок с заранее настроенным сообщением, дабы проинформировать нас о том, что произошло какое-то конкретное событие. Android Things-приложение использует Temboo-сервис, который называется **choreo** и упрощает интеграцию с платформой Nexmo. **Nexmo** (https://www.nexmo.com/) – это голосовая облачная платформа. Наше Android Things-приложение с помощью Android Things-платы будет вызывать Temboo, когда температура будет больше либо равна 30 °C. В ответ Temboo будет вызывать Nexmo, которая будет совершать звонок на наш смарт-

фон, используя движок Text To Speech (TTS), чтобы преобразовать текст сообщения в человеческий голос, который мы слышим во время телефонного звонка.

3.1. Первый шаг – это настроить Temboo choreo, чтобы она могла общаться с платформой Nexmo. Сначала мы должны создать бесплатный аккаунт Nexmo для нашего приложения. При регистрации вы должны предоставить номер своего телефона и бизнес-адрес электронной почты (т. е. адрес почты, зарегистрированный в какой-то организации (HSE подойдёт), а не в бесплатных почтовых системах, таких как Mail.ru, Gmail, Yandex и т. д.). Затем вы должны верифицировать свой номер телефона и адрес электронной почты. Внимание: письмо с верификацией приходит в течение 5–10 минут. После верификации вы можете войти на сайт платформы и ответить на дополнительные вопросы, например такие:

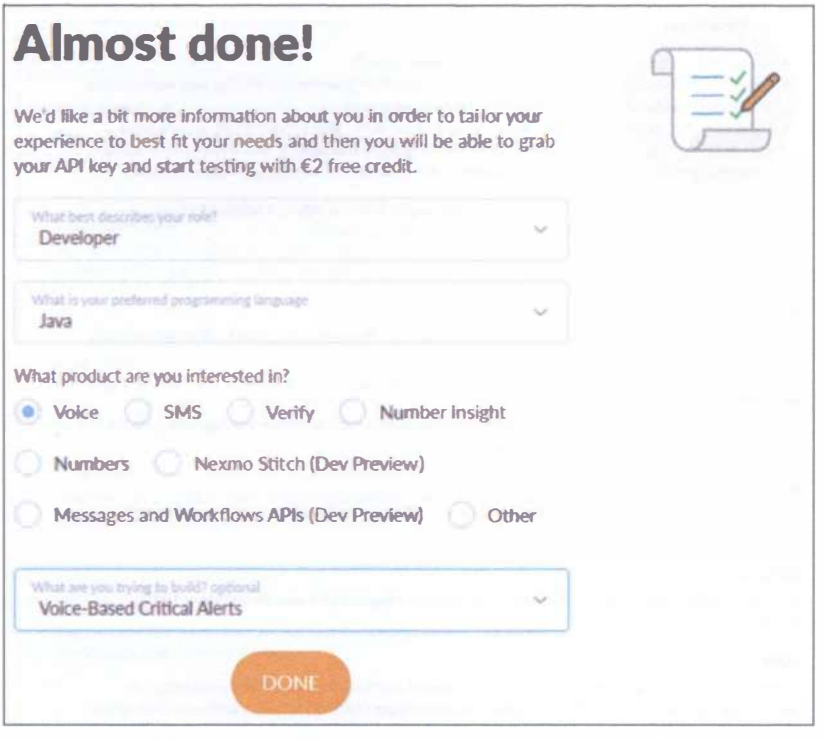


Рис. 138 ❖ Дополнительные вопросы платформы Nexmo

Нажмите кнопку **Done** и перейдите в панель инструментов консоли. Нам нужна эта консоль, чтобы получить два важных параметра: ключ (key) и секрет (secret). Нажмите на логотип Nexmo в верхнем левом углу консоли, и вы увидите эти параметры (вместе с приветственным сообщением и вашим номером телефона в разделе **Try the API**). Эта пара параметров используется платформой Temboo для аутентификации запроса к серверу.

Теперь нам нужно создать аккаунт Temboo (**https://temboo.com/**). Бесплатная пробная версия действует только 14 дней! Перейдите по ссылке **https://temboo.com/signup** и заполните все текстовые поля. Нажмите кнопку **Try it now**, затем прокликайте через 3 примера, потом нажмите **Start building**. Закройте приветственное всплывающее окно и найдите Nexmo в левой панели. Выберите её и заметьте, что в окне справа есть информация о функции TTS.

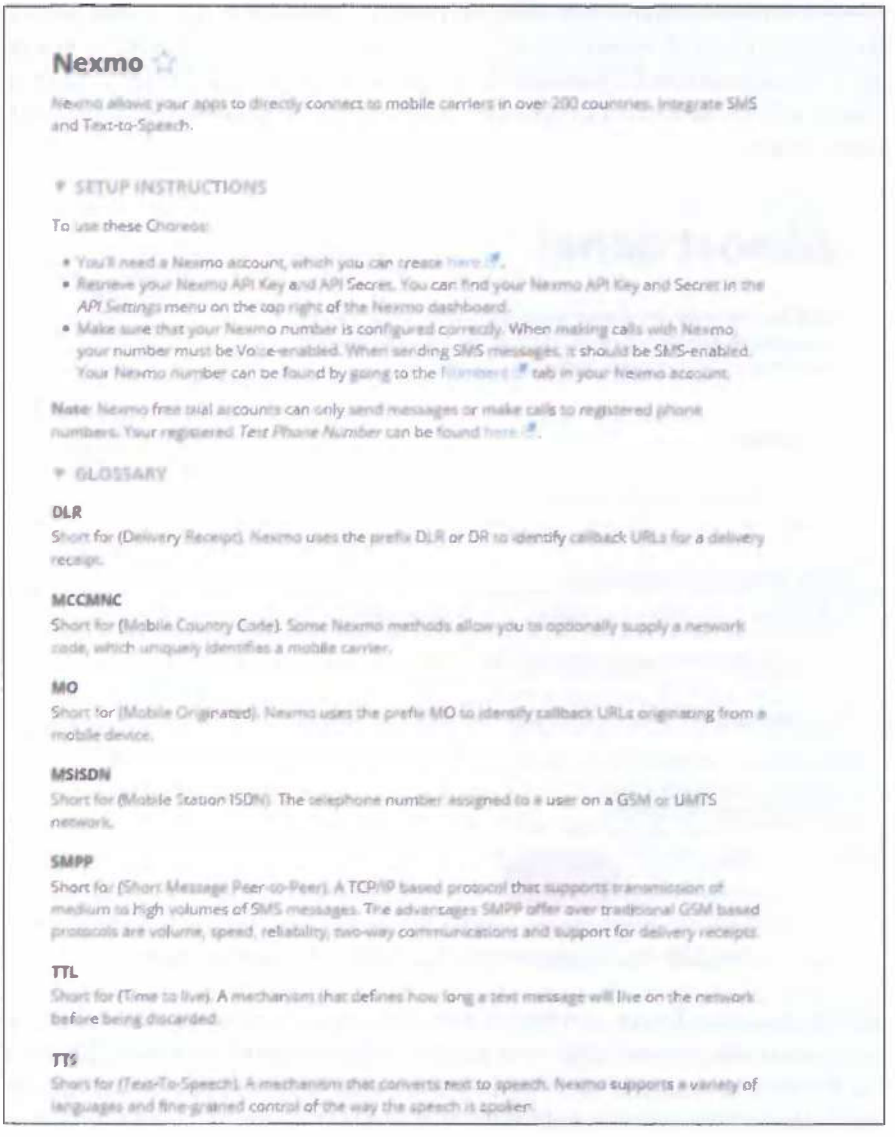


Рис. 139 ❖ Информация о поддерживаемых возможностях интеграции с Nexmo

Ниже перечня возможностей располагается список поддерживаемых функций, среди которого нет **Nexmo.Voice choreo** в последней на данный момент версии (2.22.0), но мы можем выбрать, например, SMS-функционал (под Bundles) и затем функцию SendMessage, и вы увидите следующее окно:

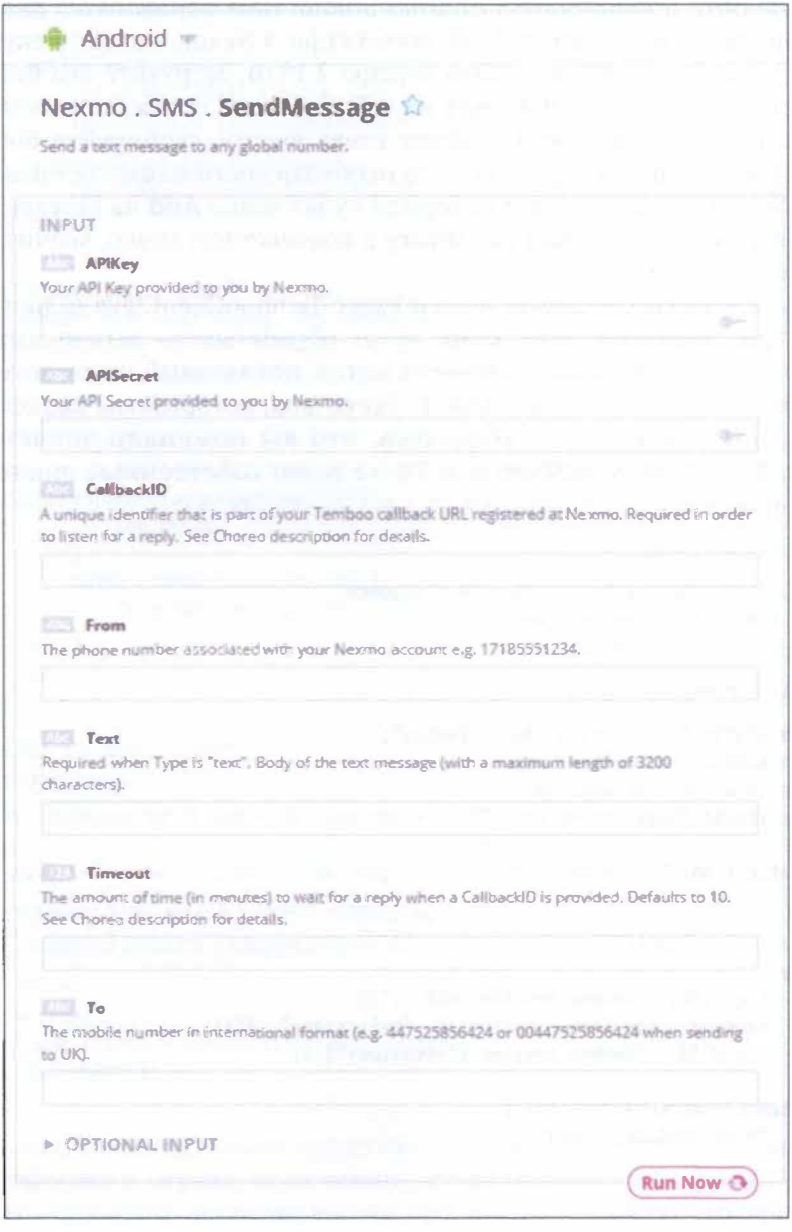


Рис. 140 ❖ Окно настройки функции отправки сообщений Nexmo

Вам нужно заполнить все поля, кроме CallbackID и Timeout. При этом важно, чтобы вверху вы выбрали Android, как показано на рис. 140. После заполнения всех полей вы автоматически получите код ниже, в разделе **code**. Теперь мы можем интегрировать код в наше Android Things-приложение – нам понадобятся некоторые части этого кода на следующих этапах.

3.2. Откройте приложение в Android Studio. Нам понадобятся две библиотеки из архива: temboo-android-sdk-core-xxx.jar и Nexmo-xxx.jar (текущая версия 2.22.0, мы будем использовать версию 2.19.0). Загрузите эти библиотеки из источников [17] и [18]. Добавьте их в библиотеки проекта (поменяйте вид с Android на Project в Project Explorer слева вверху, скопируйте библиотеки и вставьте их в папку things -> libs; по необходимости нажмите правой кнопкой мыши по каждой из них и выберите пункт меню **Add as Library** (если вы видите этот пункт меню **Add as Library** в контекстном меню, значит, необходимость есть)).

Теперь нам нужно добавить новый класс TembooClient.java (в папке things -> java ->packagename). Этот класс будет обрабатывать детали интеграции с Temboo. Ядром этого класса является метод, показанный ниже, который вызывает Temboo choreo и оборачивает choreo-код, который мы сконфигурировали в предыдущем пункте (**убедитесь, что вы поменяли значения констант ID, KEY, APIKey, APISecret и To на ваши собственные значения**, см. комментарии в коде с инструкциями, откуда что брать):

```
import android.util.Log;

import com.temboo.Library.Nexmo.Voice.TextToSpeech;
import com.temboo.core.TembooException;
import com.temboo.core.TembooSession;

public class TembooClient {

    private static final String TAG = "Temboo";
    private static TembooClient me;
    private TembooSession session;
    private static final String ID = ""; //enter your ID - the first parameter from the
Temboo CODE section: new TembooSession(
private static final String KEY = ""; //enter your KEY - the last parameter from the Temboo
CODE section: new TembooSession(

    private TembooClient() {
        try {
            Log.d(TAG, "Temboo session init...");
            session = new TembooSession(ID, "myFirstApp", KEY);
            Log.d(TAG, "Temboo session ["+session+"]");
        }
        catch (TembooException te) {
            te.printStackTrace();
        }
    }

    public static TembooClient getInstance() {
```

```
        if (me == null)
            me = new TembooClient();

        return me;
    }

    public void callTemboo() {
        Runnable r = new Runnable() {
            @Override
            public void run() {
                Log.d(TAG, "Call Temboo...");
                TextToSpeech textToSpeechChoreo = new TextToSpeech(session);

// Get an InputSet object for the choreo
                TextToSpeech.TextToSpeechInputSet textToSpeechInputs = textToSpeechChoreo.
newInputSet();

// Set inputs
textToSpeechInputs.set_APIKey(""); //enter your Key from Nexmo
textToSpeechInputs.set_Text("Hello, the temperature is too high");
textToSpeechInputs.set_To(""); // russian number pattern is 7XXXXXXXXXX
textToSpeechInputs.set_APISecret(""); //enter your Secret from Nexmo

// Execute Choreo
                try {
                    TextToSpeech.TextToSpeechResultSet textToSpeechResults =
textToSpeechChoreo.execute(textToSpeechInputs);
                    Log.d(TAG, "TTS Result ["+textToSpeechResults.get_Response()+"]");
                }
                catch (TembooException te) {
                    te.printStackTrace();
                }

            }
        };

        Thread t = new Thread(r);
        t.start();
    }

}
```

И последнее, что мы должны сделать, – это вызвать этот класс в методе onSensorChanged класса TemperatureCallback:

```
if (val >= 30) {
    turnOn = true;
    TembooClient client = TembooClient.getInstance();
    client.callTemboo();
}
```

Теперь мы, наконец, можем запустить приложение и проверить, **звонит ли нам платформа в случае, если температура больше либо равна 30 °C**. Заметьте, что при этом, согласно логике проекта из предыдущего задания, который мы использовали, красный светодиод будет гореть в этой ситуации!

Номер звонящего будет неизвестен. Вы можете изменить код, так как звонки будут поступать постоянно!

Пятый проект в Android Things – шпионский глаз

0. В этом задании мы создадим проект, который использует Android Things-плату, чтобы контролировать камеру и сервомотор, который мы используем для поворота камеры. Также мы научимся использовать PWM-пины платы (Pulse Width Modulation, широтно-импульсная модуляция). ОС Android Things поддерживает CSI-2-протокол для камеры (Camera Serial Interface, последовательный интерфейс камеры) так же, как и плата Raspberry Pi 3, у которой есть два CSI-2-порта – один для камеры и один для монитора.

1. Нам понадобятся:
- соединительные провода,
- модуль с Raspberry-камерой;
- сервомотор;
- плата Raspberry Pi 3;
- HDMI-монитор;
- кабель HDMI m-m;
- адаптер питания / кабель для HDMI-монитора;
- адаптер питания для платы Raspberry Pi 3 (5 В).

Модуль с Raspberry камерой выглядит так:

Рис. 141 ❖ Raspberry-камера

Модуль с камерой очень популярен в приложениях по охране дома и в ловушках с камерой для дикой природы. Камера обладает следующими характеристиками:

- напряжение: 5 В;
- ток, минимум: 1.8 А ;
- интерфейс: CSI-2;
- тип сенсора: Sony IMX 219 PQ CMOS ¼ дюйма;
- 8 MegaPixels максимум (3280×2464);
- поддерживаемые видеоформаты: 1080 p (30 fps), 720 p (60 fps), 640×480 p (90 fps);
- фокусное расстояние: 33 м.

И после практики на Arduino Uno 3 вам должен быть знаком сервомотор и PWM (ШИМ):

Рис. 142 ❖ Сервомотор

Сервомотор обладает следующими характеристиками:

- напряжение: 3–5 В;
- усилие на вале: 1.2 кг/см (4.8 В); 1.6 кг/см (6.0 В);
- ширина импульса: 2 мс;
- максимальный угол: 160 градусов.

Широтно-импульсная модуляция используется для того, чтобы формировать на выходе переменное напряжение, применяя цифровой сигнал. Она может быть использована для управления сервомоторами, интенсивностью освещения (светодиодами), звуком и аудио. ШИМ основана на изменении времени, при котором сигнал высокий. Есть два важных фактора в этой модуляции: **частота** и **рабочий цикл**. Например, рабочий цикл равен 50%, если сигнал высокий половину времени (периода), а другую половину периода – низкий. В Android Things нам нужно знать как частоту, так и рабочий цикл после открытия PWM-пина и перед активацией PWM-сигнала.

Перед подключением устройств к плате убедитесь, что она отключена от компьютера и питания, иначе вы можете повредить её или устройства. **Пожалуйста, будьте очень аккуратны с камерой!**

Следующая схема с макетной платой показывает, как подключить сервомотор и камеру к Raspberry Pi 3. Чтобы подключить камеру, сдвиньте вверх белый замок на CSI-2-порте для камеры, затем вставьте шлейф камеры так, чтобы контакты на шлейфе смотрели в противоположную сторону по отношению к белому замку (или так, чтобы синий ключ/полоса на конце шлейфа смотрел в сторону белого замка), затем опустите замок вниз (оба конца), чтобы обеспечить хороший контакт. Если камера подключена правильно, вы должны видеть её характеристики и картинку с трансляцией (live feed) с камеры в небольшом окне сверху характеристик, если зайдёте в пункт меню **Android Things -> Peripherals -> Camera** (включая поддерживаемые режимы работы и ID камеры: Camera 0). Если вы не видите «прямой эфир» с камеры в этом

окне или пункт меню **Camera** недоступен – значит либо вы подключили камеру неправильно, либо камера не работает (свойства камеры доступны в окне, открывающемся по щелчку на активном пункте меню **Camera**, но картинки с камеры нет).

Подключите сервомотор к плате и откройте маленькую упаковку (пакетик) с тремя насадками на серво и тремя шурупами – используйте их и металлический угловой держатель для камеры с двумя отверстиями (или любой другой угловой держатель), чтобы прикрепить сервомотор к камере. Вам понадобится крестовая отвёртка. Я рекомендую сначала ввернуть шурупы в небольшие дырки на белой насадке для сервомотора, чтобы расширить дырки, а затем вывернуть их обратно и ввернуть их снова, в этот раз продев через дырку в модуле камеры и дырку углового металлического держателя.

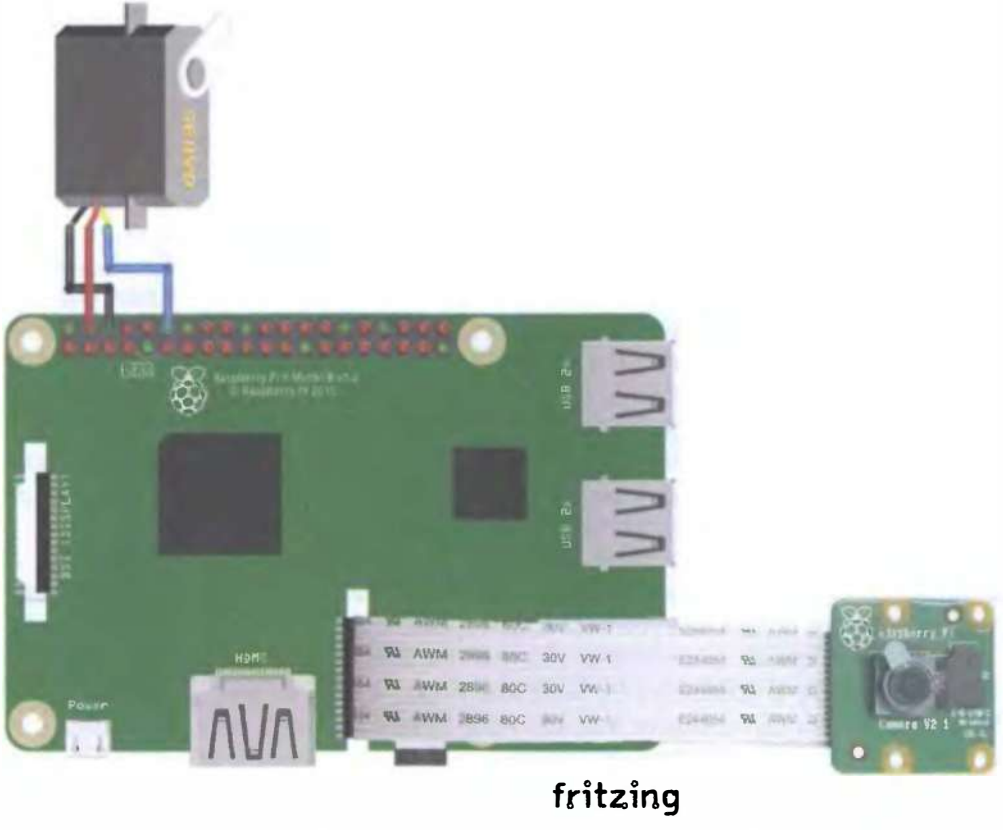


Рис. 143 ❖ Схема подключения для задания 5

2. Создайте новый проект в Android Studio. Дайте имя проекту, в следующем окне выберите дополнительно с Phone and Tablet платформу Android Things внизу и minimum SDK установите как API 24 (устанавливается по умолчанию).

В следующем окне выберите **Empty Activity**, дайте имя activity, затем в следующем окне выберите **Android Things Empty Activity**; потом нажмите **Next** и **Finish**.

Прежде всего нам нужно создать пользовательский интерфейс для нашего приложения, чтобы управлять сервомотором, который поворачивает камеру, и фотографировать помещение. Пользовательский интерфейс должен выглядеть так (две кнопки, чтобы поворачивать камеру влево/вправо (используйте для них значения **android:text=">"** и **android:text="<"**), и одна кнопка для фотографирования + ImageView, в котором будет отображаться фотография):

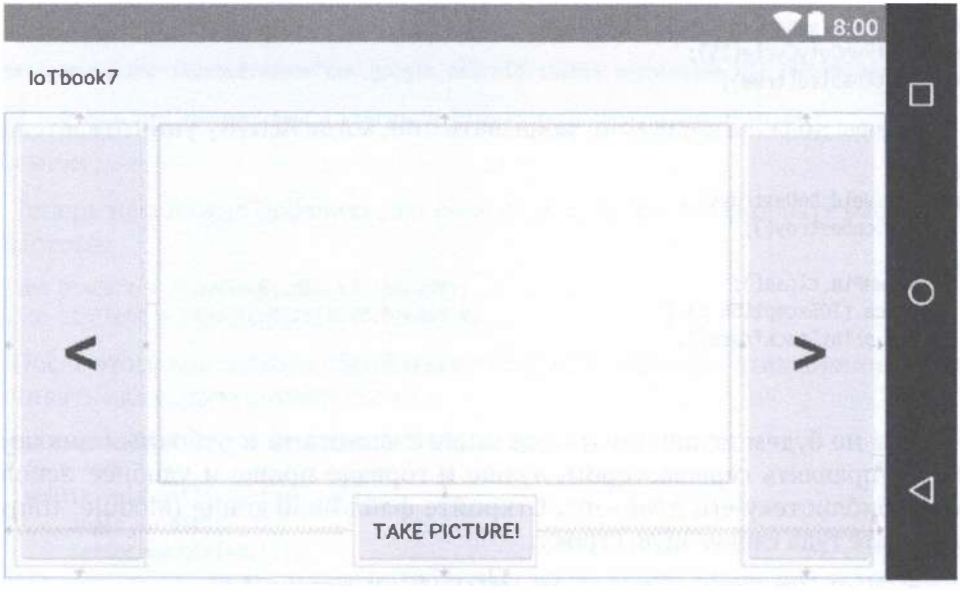


Рис. 144 ❖ Пример пользовательского интерфейса для задания 5

Чтобы открыть PWM-пин, мы могли бы использовать уже знакомый класс PeripheralManager (заметьте, что нам также нужен созданный ранее класс BoardPins в этом задании):

```
Pwm pwmPin;

PeripheralManager pm = PeripheralManager.getInstance();
try {
    pwmPin = (Pwm) pm.openPwm(getPWMPin()); // will give us «PWM0»
} catch (IOException e) {
    e.printStackTrace();
}
```

И следующий код должен быть добавлен в файл BoardPins.java:

```
public static String getPWMPin() {
    switch (getBoardName()) {
```

```
        case RASPBERRY:
            return "PWM0";
        case EDISON_ARDUINO:
            return "IO3";
        default:
            throw new IllegalArgumentException("Unsupported device");
    }
}
```

pwmPin – это экземпляр класса Pwm, у которого есть 4 метода: setPwmFrequencyHz(), setPwmDutyCycle(), setEnabled() и close(). Поэтому для частоты 50 Гц и 75 % рабочего цикла код будет такой:

```
pwmPin.setPwmFrequencyHz(50);
pwmPin.setPwmDutyCycle(75);
pwmPin.setEnabled(true);
```

Также вы должны корректно закрывать пин, когда Activity уничтожается:

```
@Override
protected void onDestroy() {
    super.onDestroy();
    try {
        pwmPin.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Но мы не будем использовать код выше с частотами и рабочими циклами. Чтобы управлять сервомотором, лучше и гораздо проще и удобнее использовать библиотеку его драйвера. Откройте файл build.gradle (Module: things) и добавьте туда следующую строку:

```
implementation 'com.google.android.things.contrib:driver-pwmservo:1.0'
```

Синхронизируйте проект, перейдите в файл MainActivity.java и добавьте следующий код в раздел объявления переменных класса:

```
private static final String TAG = MainActivity.class.getSimpleName();

private Servo mServo;

private Button btnPicture;
private ImageView imgView;

private int angle = 0;
private final int STEP = 15;
```

Затем добавьте следующий код внутрь класса:

```
private void initServo() {
    try {
        mServo = new Servo(getPWMPin());// will give us «PWM0»
        mServo.setAngleRange(0f, 180f);
```

```
            mServo.setEnabled(true);

        }
        catch(Exception e) {
            e.printStackTrace();
        }
}
```

В этом коде мы устанавливаем имя пина (**PWM0**) и минимальные и максимальные углы поворота вала сервомотора, которые определяют диапазон угла вращения. Затем мы активируем пин. Так как мы используем метод getPWMPin() класса BoardPins.java, то должны добавить разрешение управления периферией по входу и выходу в файл AndroidManifest.xml:

```
<uses-permission android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

В методе onCreate() мы должны вызвать рассмотренный метод:

```
initServo();
```

Теперь нам нужно добавить две кнопки, чтобы контролировать работу сервомотора:

```
Button btnLeft = findViewById(R.id.btnLeft);
Button btnRight = findViewById(R.id.btnRight);
```

После этого мы должны обрабатывать события нажатия этих кнопок и поворачивать вал серво соответственно:

```
btnLeft.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        angle += STEP;
        setServoAngle(angle);
    }
});

btnRight.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        angle -= STEP;
        setServoAngle(angle);
    }
});
```

И наконец, мы должны добавить метод setServoAngle():

```
private void setServoAngle(int angle) {
    if (angle > mServo.getMaximumAngle())
        angle = (int) mServo.getMaximumAngle();

    if (angle < mServo.getMinimumAngle())
        angle = (int) mServo.getMinimumAngle();

    try {
```

```
            mServo.setAngle(angle);
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

Вы можете запустить Android Things-приложение (вам придётся подключить плату к HDMI-монитору и использовать USB-мышь, которую также надо подключить к плате, либо подключить кабель USB-USB к входу сенсорного управления на мониторе и к плате и воспользоваться сенсорным экраном монитора, если у вас есть сенсорный экран), проверить, работает ли управление сервомотора через интерфейс приложения.

3. Теперь поработаем с камерой. Чтобы работать с ней, мы будем использовать пакет **android.hardware.camera2** (добавленный с API уровня 21). Он содержит класс **CameraManager**, чтобы определять и открывать камеру, класс **CameraDevice** для установки её свойств и возможностей и класс **CaptureSession** для получения фотографии и представления её на какой-то поверхности.

Добавим ещё один класс в проект, назовём этот класс AndroidCamera.java (things -> java -> packagename -> щелчок правой кнопкой мыши -> new -> Java class).

3.1. Прежде всего мы должны обнаружить камеру и убедиться, что она подключена к плате, поэтому мы используем класс CameraManager. Объявим некоторые переменные в классе AndroidCamera.java:

```
private Context ctx;
private static final String TAG = "AndroidCamera";
private String camId;
private CameraManager cManager;
private CameraDevice camera;
private CameraCaptureSession session;
private CameraListener listener;
private ImageReader iReader;

private HandlerThread imgHandler = new HandlerThread("ImageThread");

public AndroidCamera(Context ctx, CameraListener listener) {
    this.ctx = ctx;
    this.listener = listener;
}
```

И затем создадим новый метод initCamera():

```
public void initCamera() {

    cManager =  (CameraManager) ctx.getSystemService(Context.CAMERA_SERVICE);
    Log.d(TAG, "Camera Manager ["+cManager+"]");

    // Retrieve the list of cams
    try {
        String[] idCams = cManager.getCameraIdList();
        Log.d(TAG, "Camera Ids ["+idCams+"]");
        camId = idCams[0];
```

```
    } catch (CameraAccessException e) {
        e.printStackTrace();
    }

    // Initialize the ImageReader
    imgHandler.start();
    iReader = ImageReader.newInstance(640, 480 , ImageFormat.JPEG, 1);
    iReader.setOnImageAvailableListener(new ImageReader.OnImageAvailableListener() {
        @Override
        public void onImageAvailable(ImageReader reader) {
            listener.onImageReady(reader);
        }
    }, new Handler(imgHandler.getLooper()));
}
```

В этом методе мы сначала получаем ссылку на менеджер камеры, затем приложение перечисляет все подключённые к плате камеры. Мы будем использовать первую же найденную камеру (с индексом [0]). Также мы должны инициализировать контейнер картинки, который используется приложением для получения прямого доступа к данным, переведённым на какую-то поверхность. После создания обработчика (Handler), требуемого объектом ImageReader, и установки формата изображения мы используем ImageReader, который содержит только одну картинку. Затем мы подключаем слушателя (listener), чтобы этот класс получал уведомление, когда фотография становится доступной. В свою очередь, класс AndroidCamera использует другой слушатель (listener) для уведомления вызвавшего его класса (MainActivity.java) о том, что изображение доступно.

3.2. Следующий шаг – это реализовать метод, который используется для начала общения с камерой:

```
public void openCamera() {
    try {
        CameraCharacteristics characteristics = cManager.getCameraCharacteristics(camId);
        cManager.openCamera(camId, stateCallback, null);

    } catch (CameraAccessException e) {
        e.printStackTrace();
    } catch (SecurityException se)  {
        se.printStackTrace();
    }
}
```

3.3. Теперь нам нужно создать callback-класс, использованный на предыдущем шаге, чтобы получать уведомления, когда происходят события, связанные с открытием камеры:

```
private final CameraDevice.StateCallback stateCallback = new CameraDevice.StateCallback() {
    @Override
    public void onOpened(@NonNull CameraDevice camera) {
        Log.d(TAG, "Camera opened");
        AndroidCamera.this.camera = camera;
        listener.onCameraAvailable();
```

```
    }

    @Override
    public void onDisconnected(@NonNull CameraDevice camera) {
        Log.d(TAG, "Camera disconnected");
    }

    @Override
    public void onError(@NonNull CameraDevice camera, int error) {
        Log.d(TAG, "Camera Error" + error);
    }
};
```

Есть несколько методов, которые надо добавить в callback-класс. Мы заинтересованы в методе, вызывающемся тогда, когда открывается камера, потому что мы храним экземпляр **CameraDevice**, для того чтобы обращаться к подключённой камере на следующих шагах. В то же время в том же методе мы информируем вызвавший класс (caller) о том, что камера подключена.

3.4. Как только камера подключена, мы можем задействовать метод, чтобы сделать фотографию:

```
public void takePicture() {
    try {
        camera.createCaptureSession(Collections.singletonList(iReader.getSurface()),
                sessionCallback, null);
    }
    catch(Exception e) {
        e.printStackTrace();
    }

}
```

Приложение создаёт *сессию получения изображения (capture session)*, которая используется для фотографирования. Мы используем callback-класс, чтобы быть проинформированными об этих событиях. Заметьте, что метод **createCaptureSession** использует поверхность **ImageReader** для удержания фотографии.

3.5. Ниже представлен callback-метод для обработки *событий, связанных с сессией получения изображения (capture session events)*:

```
private CameraCaptureSession.StateCallback sessionCallback = new CameraCaptureSession.
StateCallback() {
    @Override
    public void onConfigured(@NonNull CameraCaptureSession session) {
        Log.d(TAG, "Camera configured");
        AndroidCamera.this.session = session;
        startCaptureImage();
    }

    @Override
    public void onConfigureFailed(@NonNull CameraCaptureSession session) {
        Log.e(TAG, "Configuration failed");

    }
};
```

Здесь важным является метод **onConfigured**, который вызывается, когда камера готова получить изображение и конфигурационный процесс закончен. Приложение использует этот метод для того, чтобы начать захват изображения.

3.6. Последний шаг – это разработка метода, который на самом деле получает изображение:

```
private void startCaptureImage() {
    try {
        CaptureRequest.Builder captureBuilder =
                camera.createCaptureRequest(CameraDevice.TEMPLATE_STILL_CAPTURE);
        captureBuilder.addTarget(iReader.getSurface());
        captureBuilder.set(CaptureRequest.CONTROL_AE_MODE, CaptureRequest.CONTROL_AE_MODE_
ON);
        Log.d(TAG, "Session initialized.");
        session.capture(captureBuilder.build(), captureCallback, null);
    }
    catch(CameraAccessException cae) {
        cae.printStackTrace();
    }

}
```

Этот метод подготавливает запрос, устанавливая некоторые параметры, и запускает сессию получения изображения. Как обычно, мы используем callback-метод, чтобы получать уведомления о событиях.

3.7. И наконец, мы определяем callback-интерфейс (или слушатель/listener), используемый классом AndroidCamera для уведомления вызывающего класса о наиболее важных событиях:

```
public static interface CameraListener {
    public void onCameraAvailable();
    public void onImageReady(ImageReader reader);
}
```

И добавляем отсутствующий captureCallback:

```
private CameraCaptureSession.CaptureCallback captureCallback = new CameraCaptureSession.
CaptureCallback() {
    @Override
    public void onCaptureCompleted(@NonNull CameraCaptureSession session, @NonNull
CaptureRequest request, @NonNull TotalCaptureResult result) {
        Log.d(TAG, "Capture completed");
        session.close();
    }
};
```

4. Теперь нам нужно изменить MainActivity.hava, чтобы закончить Android Things-приложение. Мы должны позаботиться о кнопке, которая фотографирует помещение. В методе onCreate добавьте следующее:

```
final AndroidCamera aCamera = new AndroidCamera(this, listener);
aCamera.initCamera();
aCamera.openCamera();
```

Камера была инициализирована. Теперь нам нужно добавить ссылку на ImageView, который будет показывать фотографию:

```
imgView = findViewById(R.id.img);
```

И – на кнопку:

```
btnPicture = findViewById(R.id.btnPicture);
btnPicture.setEnabled(false);
```

Изначально кнопка неактивна, пока камера не готова фотографировать. Для этого приложение использует слушатель, чтобы знать, когда камера готова, и активировать кнопку (в методе **onCameraAvailable**).

Событие onClick для этой кнопки выглядит так:

```
btnPicture.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.d(TAG, "Start capturing the image");
        aCamera.takePicture();
    }
});
```

Обработчик вызывает метод **takePicture**, чтобы сделать фотографию.

Наконец, приложение реализует callback-интерфейс, который мы использовали ранее в конструкторе:

```
private AndroidCamera.CameraListener listener = new AndroidCamera.CameraListener() {
    @Override
    public void onCameraAvailable() {
        Log.d(TAG, "Camera Ready");
        btnPicture.setEnabled(true);
    }

    @Override
    public void onImageReady(ImageReader reader) {
        Log.d(TAG, "Image ready");
        Image img1 = reader.acquireLatestImage();
        ByteBuffer bBuffer = img1.getPlanes()[0].getBuffer();
        final byte[] buffer = new byte[bBuffer.remaining()];
        bBuffer.get(buffer);
        img1.close();
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                imgView.setImageBitmap(BitmapFactory.decodeByteArray(buffer, 0, buffer.
length));

            }
        });
    }
};
```

В первом callback-методе, названном **onCameraAvailable()**, приложение активирует кнопку, как только камера готова. Во втором методе, названном

onImageReady, мы обновляем изображение, помещая фотографию в виджет **ImageView**. Код, содержащийся в этом методе, используется для извлечения изображения и его адаптации к виду, приемлемому в ImageView.

Возможно, вы уже заметили ошибку в методе openCamera класса AndroidCamera.java. Она возникла из-за того, что мы до сих пор не добавили разрешение на использование камеры в файл манифеста AndroidManifest.xml. Вы можете исправить ошибку, нажав на красную лампочку с восклицательным знаком (выберите **Add Permission CAMERA**), или добавить следующую строку в файл манифеста вручную:

```
<uses-permission android:name="android.permission.CAMERA" />
```

Поздравляю! Работа над приложением закончена, и теперь вы можете подключить к питанию вашу плату Raspberry Pi 3, HDMI-монитор, подключиться к плате через ADB и запустить приложение снова, чтобы проверить его и поэкспериментировать. После получения фотографии вы должны увидеть в интерфейсе вашего приложения что-то, похожее на рис. 145.

Рис. 145 ❖ Скриншот работающего приложения после получения фотографии с помощью камеры

Заключение

Прочитав эту книгу и пройдя все практические задания, вы получили базовый опыт в области интернета вещей и, что немаловажно, в основном практический опыт работы с реальным оборудованием, с реальными облачными инструментами для интернета вещей, наряду с некоторыми теоретическими положениями. Практический опыт гораздо ценнее, особенно для университетской среды, где, как правило, до сих пор делают упор на теоретические лекции, почти не уделяя времени практике. Вы вспомнили основы физики, закон Ома, увидели много датчиков, модулей и прочих сенсоров в работе, и у вас наверняка появились идеи разработки нескольких Android Things- или Arduino-приложений для своих нужд, при использовании различных комбинаций датчиков, модулей, индикаторов и программного кода.

Кроме того, возможно, вы усвоили правила определения ошибки в отдельно взятом проекте, если что-то не работает. Эти правила сложнее, чем если бы вам требовалось найти ошибку только в коде программы, т. е. с точки зрения программиста. Для определения ошибки начать нужно с анализа и проверки схемы подключения компонентов. При этом надо помнить, что подключение пинов на модуле к плате может не совпадать со схемой, так как на самих модулях есть собственная информация о назначении этих пинов. Затем – переходить к анализу кода: возможно, ошибка скрывается именно в нём. Если всё правильно и ошибка всё равно не уходит, возможно, надо попробовать поменять компоненты схемы, один за другим. Дело может быть совсем не в схеме или в коде, а в конкретном модуле, подключённом в схему, который просто работает неправильно или перестал работать. У меня был случай с камерой: я захотел выполнить проект с Хабрахабра по умной дверной консоли, которая автоматически открывает дверь с помощью сервомотора с RFID-картой изнутри помещения, если получен положительный ответ распознавания лица человека, подходящего к двери, с помощью облачного сервиса Google Vision, и всё это происходит в светлое время суток. Всё было хорошо, но проект не работал, так как не было изображения с камеры – сессия получения изображения не завершалась, а в коде не было никаких ошибок, так как камера определялась правильно, открывалась, и ей даже присваивался ID. В конечном итоге дело оказалось именно в камере, которая просто была бракованной, так как не могла передавать изображение даже в Android Things, если выбрать пункт меню Camera. Однако я этого не знал и искал ошибку в коде. А код у этого проекта был увесистый и состоял из нескольких классов. Таким образом, можно потратить массу времени совершенно впустую, поэтому важно возвращаться к этим трём этапам проверки и проходить их снова при необходимости, не застревая на каком-то конкретном этапе надолго, например на анализе кода. Важно так-

же понимание того, сколько займёт времени каждый из этих этапов, и сначала выполнять самые быстрые этапы, а трудные и времязатратные оставлять на крайний случай, когда ничего не помогает. Если же в нахождении ошибки не помогает ни один из перечисленных этапов, можно обратиться к аналогичным проектам в интернете, но такое случается крайне редко.

Помимо всего прочего, освоив это учебное пособие, вы получили некоторые навыки работы со средами программирования Arduino IDE и Android Studio, которые вам наверняка пригодятся в будущем, ведь Android Studio – это официальная среда разработки нативных Android-приложений. Я надеюсь, вам понравилась эта книга!

Список использованных источников

1. Комплект интернет-вещей Arduino Uno R3 Starter Learning Kit с RFID-модулем [Электронный ресурс] // Интернет-магазин onpad.ru. – URL: http://onpad.ru/shop/cubie/arduino/ardiuno_kit/1680.html.
2. Arduino IDE [Электронный ресурс] // Официальный сайт Arduino. Язык: английский. – URL: https://www.arduino.cc/en/Main/Software.
3. Arduino Create Online IDE [Электронный ресурс] // Официальный сайт Arduino. Язык: английский. – URL: https://create.arduino.cc/editor.
4. Первый урок-инструкция по Arduino IDE [Электронный ресурс] // Официальный сайт Arduino. Язык: английский. – URL: https://www.arduino.cc/en/Guide/ArduinoUno.
5. Библиотека Keypad.h для работы с кнопочным модулем 4×4 [Электронный ресурс] // Официальный сайт Arduino. Язык: английский. – URL: http://arduino.cc/playground/uploads/Code/Keypad.zip.
6. Исправленная библиотека DS1302.h для работы с модулем часов реального времени [Электронный ресурс] // Matt Sparks, Github. Язык: английский. – URL: https://github.com/msparks/arduino-ds1302.
7. Библиотека IRemote.h для работы с инфракрасным модулем [Электронный ресурс] // *Макаров С. Л.* Официальная страница курса «Экосистемы интернета вещей». Язык: английский. – URL: http://serjmak.com/2students/IoTE/IRremote.zip.
8. Исправленная библиотека MFRC522.h для работы с RFID-модулем RC522 [Электронный ресурс] // *Макаров С. Л.* Официальная страница курса «Экосистемы интернета вещей». Язык: английский. – URL: http://serjmak.com/2students/IoTE/mfrc522.zip.
9. Архив из 34 практических заданий на английском языке и некоторых библиотек [Электронный ресурс] // Интернет-магазин onpad.ru (ссылка взята из [1]). Язык: английский. – URL: https://yadi.sk/d/YUYx69mEm3xPV.
10. Открытая библиотека и редактор электронных компонентов и схем Fritzing [Электронный ресурс] // Friends-of-Fritzing foundation. Язык: английский. – URL: http://fritzing.org/home./
11. Резистор [Электронный ресурс] // Амперка вики. – URL: http://wiki.amperka.ru/конспект-arduino:резистор.
12. Светодиод [Электронный ресурс] // Амперка вики. – URL: http://wiki.amperka.ru/конспект-arduino:светодиод.

13. Код программы 1 для задания 34 [Электронный ресурс] // Загружено на сайт serjmak.com 12 октября 2018. – URL: http://serjmak.com/2students/IoTE/SetPassword.txt.
14. Код программы 2 для задания 34 [Электронный ресурс] // Загружено на сайт serjmak.com 12 октября 2018. – URL: http://serjmak.com/2students/IoTE/DoorCon.txt
15. Android Things Projects [Text] // *Francesco Azzola*. – Packt, 2017. – 232 p. – ISBN: 9781787289246.
16. Raspberry Pi 3 [Electronic resource] // Official Google site for Android Things developers, last updated: September 25, 2018. – URL: https://developer.android.com/things/hardware/raspberrypi#io-pinout.
17. Библиотека Nexmo (версия 2.19.0) [Электронный ресурс] // Загружено на сайт serjmak.com: 4 октября 2018. – URL: http://serjmak.com/2students/IoTE/Nexmo-2.19.0.jar.
18. Библиотека Temboo (версия 2.19.0) [Электронный ресурс] // Загружено на сайт serjmak.com: 4 октября 2018. – URL: http://serjmak.com/2students/IoTE/temboo-android-sdk-core-2.19.0.jar.
19. Build a Proof of Concept (former Hardware 101 page) [Электронный ресурс] // Сайт разработчиков под Android, Google Inc., обновлено 7 августа 2018. – URL: https://developer.android.com/things/get-started/proof-of-concept.
20. Raspberry Pi 3 Model B [Электронный ресурс] // Сайт ООО «Амперка», 2018. – URL: http://amperka.ru/collection/soc-boards/product/raspberry-pi-3-model-b.
21. Raspberry Pi 3 Official Starter Kit 16GB Black – Комплект с Raspberry Pi 3 [Электронный ресурс] // Интернет-магазин onpad.ru. – URL: https://onpad.ru/catalog/cubie/raspberrypi/raspberrypi_kit/kit_raspberrypi3/2279.html.
22. Учебное пособие по практическим занятиям в рамках курса «Экосистемы интернета вещей» (IoT Ecosystems) [Электронный ресурс] // Загружено на сайт serjmak.com: 12 ноября 2018. – URL: http://serjmak.com/2students/IoTE/Makarov_S.L._Uchebnoye_posobiye_po_IoT_2018.docx.

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Сергей Львович Макаров

Arduino Uno **и** Raspberry Pi 3:
от схемотехники к интернету вещей

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 16,41. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**