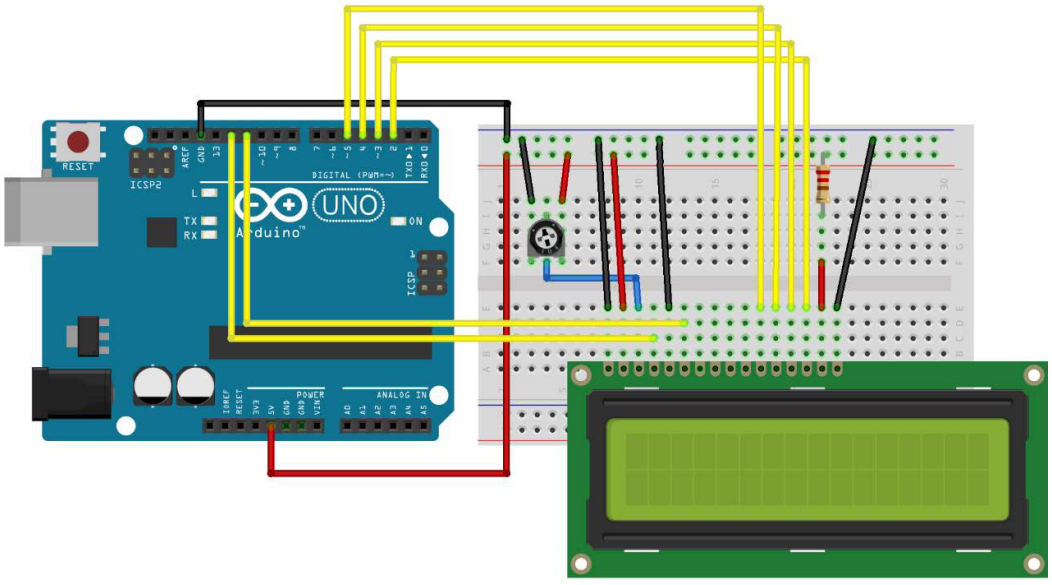
Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Тверской государственный университет»

С.В. СОРОКИН, И.С. СОЛДАТЕНКО

ОСНОВЫ РАЗРАБОТКИ И ПРОГРАММИРОВАНИЯ РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

Учебное пособие



ТВЕРЬ 2017

УДК 004.896(075.8)
ББК 3966.093я73-1+3816.2я73-1
С65

Рецензенты:

Доктор технических наук, доцент
С.В. Новикова
Кандидат технических наук, доцент
В.Л. Волушкова

Сорокин С.В., Солдатенко И.С.
С65 Основы разработки и программирования робототехнических систем: учеб. пособие. – Тверь: Твер. гос. ун-т, 2017. – 157 с.

ISBN 978-5-7609-1232-9

Пособие посвящено изучению основных принципов моделирования и построения электронных схем, программирования микроконтроллеров и роботов. В первой части пособия рассматриваются вопросы разработки собственных электронных схем, используется широкий набор компонент: от транзисторов и светодиодов до сервомоторов и плат Arduino, позволяющих запрограммировать интеллектуальное управление элементами схемы. Вторая часть посвящена более сложным вопросам, а именно программированию логики поведения роботов: перемещение в пространстве, объезд препятствий, взаимодействие с окружающей средой.

Предназначено для студентов направления 15.03.06 Мехатроника и робототехника, а также других направлений подготовки, изучающих программирование в качестве одной из профильных дисциплин (в частности, 02.03.02 Фундаментальная информатика и информационные технологии и другие).

УДК: 004.896(075.8)
ББК 3966.093я73-1+3816.2я73-1

Печатается по решению научно-методического совета
Тверского государственного университета.

ISBN 978-5-7609-1232-9

Оглавление

Введение

Настоящее учебное пособие можно считать первым шагом на пути освоения «практического» инженерного моделирования, на котором учащиеся познакомятся с основными принципами моделирования и построения электронных схем, программирования микроконтроллеров и роботов.

Пособие разделено на две части. Первая посвящена вопросам конструирования электронных схем, а также программирования логики работы некоторых ее компонентов, как например, микроконтроллеров Arduino. Лабораторные работы выполняются в бесплатной виртуальной лаборатории https://circuits.io/lab/.

Данная виртуальная лаборатория позволяет создавать собственные электронные схемы, используя широкий набор компонент: от транзисторов и светодиодов, до сервомоторов и плат Arduino, позволяющих запрограммировать интеллектуальное управление элементами схемы. Первая часть пособия позволит обучающимся познакомиться с низкоуровневыми вопросами робототехники: работой с сенсорами, датчиками, сервомоторами, устройствами питания, микроконтроллерами – со всем тем, из чего впоследствии строится сложная, многокомпонентная модель робота.

Вторая часть учебного пособия посвящена более высокоуровневым вопросам программирования поведения робота. Здесь большее внимание уделяется не сбору информации с датчиков и посылки сигналов приводам, а вопросам интеллектуального управления поведением робота: перемещению в пространстве, исследованию территории, обнаружению коллизий и реагированию на них, взаимодействию с окружающей средой. Все лабораторные работы также проводятся в бесплатной виртуальной среде V-Rep, которую можно скачать с сайта разработчика http://www.coppeliarobotics.com.

V-Rep предоставляет виртуальную среду моделирования с интегрированной средой разработки, которая позволяет как строить различные робототехнические устройства: от манипуляторов до свободно перемещающихся на плоскости или в воздушно-водной среде роботов, так и моделировать их поведение. В библиотеке системы есть большое количество уже заранее

созданных роботов, поведение всех узлов и компонентов которых можно задавать посредством скриптов.

Задачи учебного пособия

Обучающие:

- познакомить учащихся с основными принципами конструирования и программирования роботов;
- сформировать и развить абстрактное и логическое мышление;

Развивающие:

- развить навык написания компьютерных программ для управления сложными процессами и явлениями (на примере роботов);
- сформировать учебную мотивацию и мотивацию к творческому поиску;
- развить творческий и рациональный подход к решению поставленных задач;
- развивать у обучающихся элементы технического мышления, изобретательности, образное и пространственное мышление;
- развить логическое мышление.

Требования к предварительной подготовке

Для освоения курса обучающиеся должны иметь базовые навыки программирования на одном из языков высокого уровня, изучаемых в школе. Пособие содержит краткое описание языков Си и Lua, используемых в тексте, но оно недостаточно для людей, не имеющих опыта программирования на других языках.

Необходимо также иметь начальные знания электротехники в рамках школьного курса физики:

- физические величины: сила тока, напряжение и сопротивление;
- приборы для измерения этих величин;
- закон Ома;
- правила Кирхгофа;
- последовательное и параллельное соединение сопротивлений.

Для изучения дополнительной литературы и дальнейшего саморазвития в данном направлении желателен навык чтения технической литературы на английском языке.

Целевая аудитория и развиваемые компетенции

Настоящее учебное пособие предназначено для студентов направлений «15.03.06 Мехатроника и робототехника», «02.03.02 Фундаментальная информатика и информационные технологии», «01.03.02 Прикладная математика и информатика», «09.03.03 Прикладная информатика», а также других направлений подготовки, изучающих программирование в качестве одной из профильных дисциплин. Материал пособия может быть использован для организации учебной практики первых курсов, а также при проведении лабораторных практикумов по программированию.

Учебное пособие подготовлено в соответствии с ФГОС ВО по перечисленным выше направлениям подготовки и направлено на развитие следующих компетенций:

1. *по направлению подготовки 15.03.06 Мехатроника и робототехника:*
   - способность к самоорганизации и самообразованию (ОК-7);
   - способность решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности (ОПК-6);
   - способность участвовать в качестве исполнителя в научно-исследовательских разработках новых робототехнических и мехатронных систем (ПК-9);
2. *по направлению подготовки 02.03.02 Фундаментальная информатика и информационные технологии:*
   - способность к самоорганизации и самообразованию (ОК-7);
   - способность применять в профессиональной деятельности современные языки программирования и языки баз данных, методологии системной инженерии, системы автоматизации проектирования, электронные библиотеки и коллекции, сетевые технологии, библиотеки и пакеты программ, современные профессиональные стандарты информационных технологий (ОПК-2);
   - способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных

моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям (ОПК-3);
- способность использовать современные инструментальные и вычислительные средства (ПК-3);
- решать задачи профессиональной деятельности в составе научно-исследовательского и производственного коллектива (ПК-4);
- способность применять на практике международные и профессиональные стандарты информационных технологий, современные парадигмы и методологии, инструментальные и вычислительные средства (ПК-8);

3. *по направлению подготовки 01.03.02 Прикладная математика:*
   - способность к самоорганизации и самообразованию (ОК-7);
   - способность приобретать новые научные и профессиональные знания, используя современные образовательные и информационные технологии (ОПК-2);
   - способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям (ОПК-3);
4. *по направлению подготовки «09.03.03 Прикладная информатика»:*
   - способность к самоорганизации и самообразованию (ОК-7);
   - способность разрабатывать, внедрять и адаптировать прикладное программное обеспечение (ПК-2).

Авторы выражают признательность Сорокиной Ирине Владимировне за подготовку материала для глав третьего раздела, а также рецензентам за ценные критические замечания.

Тема 1

Встроенные системы

1.1 Основные понятия о встроенных системах

Итак, что такое встроенные системы? Можно дать простое определение: встроенные системы – это компьютерные системы, которые совсем не похожи на компьютеры. В таких системах сложность компьютера скрыта от пользователя.

Скорее всего у вас есть компьютер, например, ноутбук или что-то подобное. И скорее всего у вас возникали сложности в его использовании. Скажем, вы хотели установить новое программное обеспечение, но произошел конфликт, и вы не смогли этого сделать. Так иногда бывает с компьютерными играми. Вы купили новую игру, а она не работает, потому что ей нужна новая версия драйверов видеокарты. Вы устанавливаете новые драйвера, а у вас перестает работать другая игра, работавшая до этого, так как ей не подходят новые драйвера. Или новые драйвера могут вообще не работать с вашей видеокартой и нужно купить еще и новую видеокарту. Все эти проблемы возникают из-за универсальности наших «обычных» компьютеров – мы используем их для решения очень разных задач, причем мы пытаемся их все решать с помощью одного и того же вычислительного устройства.

Со встроенными системами все не так. Как правило, встроенная система используется для какой-то одной конкретной задачи. Если

это камера, то она должна снимать изображения или видео. Если это автомобиль – он должен ездить как автомобиль. Внутри них может находиться сложная компьютерная система, но только она встроена в устройство и поэтому ее сложность скрыта от пользователя, который знает, как работать с этими системами через простой интерфейс.

Например, цифровая камера использует тот же интерфейс, что и старая механическая камера – нужно нажать кнопку и она сделает фотографию. Внешне цифровая камера управляется так же, как механическая, но внутри она устроена гораздо более сложно, за счет чего пользователь получает дополнительные преимущества. Ему не нужно больше проявлять пленку, он может сразу же увидеть результат и даже более того – тут же на камере выполнить его предварительную обработку, которую раньше можно было сделать только на компьютере. Еще раз повторим, что при этом вся эта сложность скрыта от пользователя за простым и понятным интерфейсом.

Встроенные системы не всегда взаимодействуют с пользователем непосредственно – они могут делать это через другое устройство. Что это значит? Давайте рассмотрим USB накопитель, который хранит данные. Эта вещь, как мы понимаем, не взаимодействует непосредственно с человеком, ведь у нас же нет разъема для подключения такого накопителя к своему телу или чего-то вроде этого! Мы подключаем его к компьютеру, и потом взаимодействуем с ним через компьютер, чтобы получить доступ к файлам. И это не просто микросхема памяти – в накопителе есть микрокомпьютер, который управляет его работой. Таким образом, это тоже встроенная система, хотя она непосредственно не взаимодействует с человеком.

То же самое может быть и в других устройствах, например, антиблокировочная система внутри автомобиля. Человек нажимает на педаль тормоза, а педаль тормоза, в свою очередь, связана с антиблокировочной тормозной системой, таким образом, человек взаимодействует с автомобилем, а в автомобиле взаимодействуют друг с другом его подсистемы.

Важной особенностью встраиваемых систем, отличающей их от обычных компьютеров, является то, что в их работе ключевым фактором является эффективность. Это означает, что не всегда достаточно просто выполнять нужную функцию, чтобы решать

задачу. Можно сказать, что способ решения должен быть нестандартным. Например, он должен работать быстро, или он должен работать с низким энергопотреблением, или он должен быть дешевым. И в этом на самом деле заключается большая разница между разработкой встраиваемых систем и, например, традиционным дизайном программного обеспечения.

Когда изучают программирование, то в качестве конечной цели обычно ставится решение какой-то конкретной задачи. Достаточно написать код, который делает то, что нужно, например, сортирует список. При этом чаще всего не требуется, чтобы алгоритм использовал минимально возможное количество памяти, и чтобы список из миллиона элементов сортировался за 0.01 секунды. Иногда такие ограничения важны и студенты соответствующих направлений подготовки даже изучают целые дисциплины, посвященные эффективным алгоритмам и структурам данных, но на практике в обычном программировании решаемые задачи не всегда предъявляют такие повышенные требования к эффективности – достаточно, чтобы программа просто работала.

Но для встроенных систем эффективность очень важна. Недостаточно просто заставить такую систему работать. Она должна делать это эффективным образом. Причиной этих ограничений является то, что для большинства этих устройств очень важно снижение их себестоимости, как например для устройств потребительской электроники. Или же от их работы может зависеть жизнь и здоровье людей, если, например, такие устройства используют врачи или военные.

Рассмотрим первый пример – с потребительским устройством. Допустим, вы разрабатываете новый телефон. Для таких устройств производственные затраты, а также затраты на проектирование, и время выхода на рынок являются первичными. Вы хотите заработать деньги на устройстве, поэтому его стоимость должна быть низкой, ведь никто не будет покупать новый телефон за слишком дорогую цену, тем более, что у конкурентов она скорее всего будет меньше. Кроме того, важна и скорость выхода на рынок. Те же самые конкуренты сделают все, чтобы выпустить свой телефон раньше вас и завоевать бóльшую часть рынка. Все это накладывает жесткие ограничения на «начинку» вашего телефона – его производительность, энергопотребление, объем памяти, количество

дополнительных функций и т.д. Слишком «толстая» программная начинка будет тратить больше процессорного времени, работать медленнее, а батарейка будет садиться раньше. С другой стороны, слишком «тощая» программная начинка превратит ваше устройство в «кирпич», по которому в лучшем случае можно будет только делать звонки, а о каких-либо дополнительных функциях можно будет и не мечтать.

С другой стороны, рассмотрим применение в медицине. От медицинского оборудования зависит жизнь людей, поэтому здесь вы больше заботитесь о производительности и надежности, а не о конечной цене. Если электронное устройство контролирует работу сердца человека, лучше, чтобы оно было надежным, даже если из-за этого оно будет стоить дороже. Очевидно, что под надежностью понимаются не только физические характеристики устройства. Программное обеспечение подобных устройств тоже должно быть надежным. Например, современный кардиостимулятор «слушает» сердечный ритм человека, анализирует его и, если в нем возникает пауза или какое-либо другое нарушение, начинает генерировать импульсы. Теперь представьте, что будет, если программное обеспечение кардиостимулятора начнет «притормаживать» так же, как это иногда делают программы на нашем компьютере, усиленно «шурша» при этом жестким диском. Или еще хуже – возьмет и зависнет. Результат, думаю, будет очевиден.

Обсудим отличия в процессе разработки встроенных устройств и обычных компьютеров.

Одним из наиболее важных отличий является то, в отличие от настольных и портативных компьютеров, которые могут запускать программы любого типа, встроенные устройства, как правило, разрабатываются под конкретную задачу (или набор связанных задач). Современные телефоны являются исключением, потому что они размывают это различие. Телефоны начинались как типичные встроенные системы, но сегодня они практически превратились в полноценные универсальные компьютеры. Тем не менее, большинство устройств, таких как камеры, электроника автомобиля, бытовая техника и прочее сделаны для решения конкретных задачи. Например, камера – чтобы снимать. И она делают все, что связанно с этой деятельностью камеры, не делая других вещей. Это свойство является общим для большинства встроенных систем. Это важно,

потому что это изменяет способ разработки устройства. Когда вы знаете, что оно будет решать только одну задачу, вы проектируете его для этой конкретной задачи, а не на все случаи жизни.

Таким образом, разработка фокусируется на одном применении, в отличие от систем общего назначения, таких как ноутбук или настольный компьютер. Универсальные ноутбуки и компьютеры часто оказываются более мощными, чем необходимо для решения той задачи, для которой они используются. Это происходит потому, что они должны быть готовы работать с любым типом вычислительных задач. Так, например, возьмем стандартный компьютер, скажем с четырехъядерным процессором. У вас может быть четырехъядерный Core i7, который работает на частоте 4 ГГц. Как правило, вы не используете весь вычислительный потенциал этой машины. Вам совсем не нужно четыре процессора для запуска MS Word или PowerPoint, независимо от того, что вы в них делаете. Вам также не нужно 4 ГГц для запуска этих приложений. Компьютер успевает сделать миллионы различных операций до того, как вы напечатаете очередную букву. Откройте, ради интереса, диспетчер задач своего ноутбука и посмотрите на процент использования времени центрального процессора. Как правило (если вы не играете в игру в данный момент и не рендерите 3D-сцену), большая часть приложений спит и только единицы потребляют десятые доли процента (сам диспетчер задач, например). То есть 99% процентов ресурсов центрального процессора не используется!

Но иногда эта мощность все-таки нужна. Например, если вы обрабатываете фотографии или видео. Такие приложения требуют всю вычислительную мощность, которая есть в компьютере. Другой пример – это игры со сложной графикой, требующие огромную вычислительную мощность. Поэтому, когда вы в них играете, вы используете все возможности вашего компьютера. Но в остальное время эта мощность не используется.

В этом смысле универсальные компьютеры очень неэффективны. Если вы хотите играть в новые игры, вам придется купить дорогой мощный компьютер с хорошей видеокартой. Или, вы можете купить игровую приставку, которая сможет запускать те же игры, может быть даже быстрее, и она будет дешевле компьютера. Приставку можно использовать только для игр, но она делает это лучше, чем компьютер. То есть ее архитектура более эффективна для решения

данной задачи. И большую часть времени вы будете использовать ее если не на полную мощность, то по крайней мере на большую часть мощности, так как будете только играть на ней.

Таким образом, конкретное применение позволяет повлиять на процесс разработки системы. Вы можете включить в нее только те блоки, которые действительно необходимы. Благодаря этому становится возможной более высокая эффективность конструкции. Это означает, что, если я знаю, что мое оборудование предназначено только для игр, то есть определенные компоненты, которые мне не нужны в моей конструкции. Зачем покупать то, что не требуется? Рассмотрим фотоаппарат. Я знаю, что эта вещь должна делать некоторые виды обработки фото и видео. Я знаю, что нужно управлять объективом и вспышкой. Но камере никогда не придется рендерить 3D-графику, раздавать окружающим Wi-Fi, подключать к себе и обслуживать десяток периферийных устройств и выполнять много других функций. Поэтому я могу просто включить только то программное обеспечение и аппаратные средства, которые мне нужны для выполнения моих конкретных задач. И это сделает устройство дешевле и эффективнее. Увы, но за все надо платить. Общее правило нашей жизни, которое выполняется и для вычислительных систем – чем более универсальное устройство, тем менее оно эффективно, и наоборот. Конечно, врач общей практики знает про зубы больше, чем обычный человек, но разве вы пойдете к нему, если у вас заболит зуб? Нет, вы предпочтете узкоспециализированного, и от того более эффективного для решения данной задачи зубного врача.

Еще одним большим отличием встраиваемых систем по сравнению с обычными настольными компьютерами и ноутбуками является то, что аппаратное и программное обеспечение, как правило, разрабатываются совместно. Если, скажем, вы хотите купить Microsoft Word или другую программу, вы, очевидно, можете приобрести ее отдельно от компьютера. И их сделали две совершенно разные фирмы. Например, ваш ноутбук может быть сделан фирмой Asus, а Word – фирмой Microsoft. Одна фирма делает оборудование, а другая – программы. Для встроенных системы все наоборот. Саму TV-приставку и программную оболочку для нее делает одна и та же компания. Мы говорим «как правило», потому что, конечно же, из всего есть исключения. Например, компания Apple делает все сама –

и компьютеры, и ноутбуки, и телевизоры, и телефоны, и операционные системы, и офисные приложения, иными словами – весь спектр универсальных и встроенных систем.

Почему удобно делать встроенные системы полностью самому? Потому что я могу сделать оборудование именно таким, которое нужно для запуска именно того программного обеспечения, которое мне нужно. Я могу сделать их соответствующими друг другу. Весь процесс проектирования становится более эффективным. Мы купим только те компоненты, которые нужны, чтобы построить систему, которую мы хотим. Мы не будем говорить, любая система. Мы возьмем только те компоненты, которые нужны для нашей конкретной системы. И мы будем писать код, который использует только их.

Это отличает встроенные системы от систем общего назначения, в которых нужно иметь оборудование на все случаи жизни и программное обеспечение для всех случаев жизни, даже если это потребуется всего лишь один раз в этой самой жизни.

Вопросы для самоконтроля

1. Выберите верные утверждения:
   a) Встроенная система используется для одной конкретной задачи.
   b) Встроенные системы всегда взаимодействуют непосредственно с пользователем.
   c) Одной из особенностей встраиваемых систем является то, что для них очень важна эффективность.
2. Определите, какие из черт обычно присущи встроенным системам:
   a) Встроенные устройства обычно разрабатываются для конкретной задачи.
   b) Аппаратное и программное обеспечение разрабатывается отдельно.
   c) Встроенные системы многофункциональны и универсальны.

1.2 Компоненты встроенных систем

В этом разделе мы поговорим о структуре встраиваемых систем. Конкретно, мы обсудим структуру аппаратных средств, посмотрим на ее различные компоненты, причем основной компонент – микроконтроллер – обсудим более подробно в следующем разделе.

На Рис. 1.1 схематически изображена общая схема. Встроенная система должна получать данные из внешнего мира, обрабатывать их и затем выводить данные во внешний мир. Так что, в первую очередь, она имеет набор датчиков для приема данных. Эти датчики могут получать информацию о внешнем мире по-разному, поэтому существует много различных типов подобных устройств.

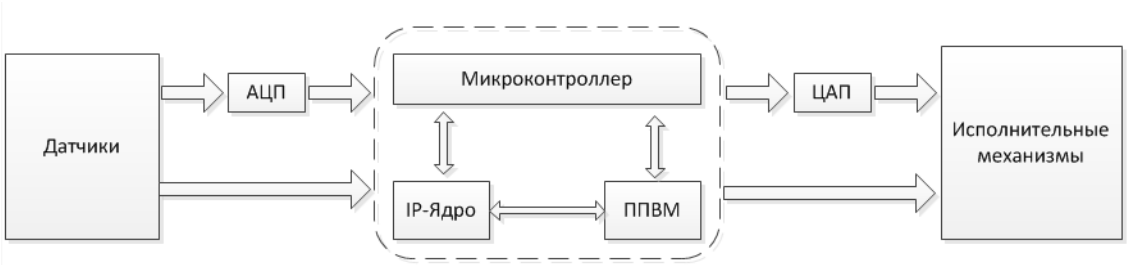


Рис. 1.1: Устройство встроенных систем

Самый простой тип датчика – это кнопка или что-то подобное, которая получает данные в очень простом виде: нажата или не нажата. Система может получать и звуковую информацию, используя микрофон. Видеокамеры – это тоже датчики, которые получают информацию в виде изображения. Сенсорный экран позволяет не только выводить, но и получать информацию. И так далее – существует большое множество самых разнообразных типов датчиков, через которые встроенная система может получать информацию.

Итак, входящая информация поступает в систему и дальше она попадает в ее ядро, которое занимается ее обработкой. В конце этого процесса, когда система решила, что делать с поступившей информацией или какое решение нужное принять на ее основе, она должна выдать какие-то результаты. Чаще всего это означает – произвести какой-то действие во внешнем мире. Это делается с помощью исполнительных механизмов или приводов, которые показаны в правой части схемы на Рис. 1.1.

Исполнительным механизмом может быть светодиод, который может гореть или мигать. Например, он может сигнализировать, что видеокамера записывает или что кончается батарейка. Но есть и другие исполнительные механизмы. Например, внутри камеры есть моторы, с помощью которых происходит управление объективом и наведение резкости. Эти двигатели являются исполнительными механизмами, а их движение – это выход системы.

Существует множество различных типов исполнительных механизмов: динамики воспроизводят звук, лампы позволяют системе выводить свет, экран также выводит свет, но в более детальном и «управляемом» виде.

Так что, если вы теперь посмотрите на встроенную систему, то увидите, что она находится посередине. Она получает данные от датчиков, что-то с ними делает, а затем посылает соответствующие сигнал на исполнительные устройства, чтобы заставить их что-то сделать в реальном мире в ответ на данные, которые она получила. В этом смысле встроенная система обеспечивает связь между датчиками и исполнительными механизмами.

В центре встроенной системы находится микроконтроллер. Более подробно мы обсудим его в следующих главах. Кроме микроконтроллеров могут быть и другие компоненты. Два из тех, которые встречаются достаточно часто, показаны на Рис. 1.1: это IP-ядра и ППВМ.

IP это английское сокращение от intellectual property, то есть интеллектуальный продукт. IP-ядро представляет собой готовые блоки для проектирования устройств, например, это может быть готовая микросхема, которая выполняет одну функцию. Конечно «одну функцию» не надо понимать слишком буквально – IP-ядро может выполнять несколько функций, но это тесно связанные функции, ориентированные на какую-то конкретную задачу. То есть этот блок не является универсальной программируемой микросхемой общего назначения. Наоборот, это микросхема, которая просто умеет выполнять некоторый небольшой набор связанных между собой функций. Такие устройства могут быть очень полезны, и к тому же они дешевы при производстве в больших объемах. Пусть у вас есть достаточно общая система, имеющая большое количеством различных подсистем, как, например, мобильный телефон. Все сотовые телефоны должны выполнять

определенные алгоритмы обработки звука. Можно сделать специальную микросхему, которая будет выполнять такую обработку и ее можно будет продавать производителям телефонов. Так как телефонов производят очень много, то и микросхем потребуется много. Чтобы сделать одну такую микросхему, потребуется много времени, труда и денег и это было бы крайне невыгодно, а вот если необходимо произвести сотни тысяч или даже миллионы подобных микросхем, то себестоимость каждой из них становится очень низкой.

IP-ядра делают для распространенных задач – для тех, которые возникают снова и снова и на которые есть большой спрос. Если какая-то задача встречается только в одной конкретной системе, то для нее не имеет смысла делать IP-ядро.

Рассмотрим, например, сетевые контроллеры. Они встречаются во многих системах. Любое устройство, которое подключается к сети Интернет, должно иметь соответствующее устройство. Такую вещь можно реализовать в виде IP-ядра – специализированной части аппаратных средств, обеспечивающей только функцию взаимодействия с компьютерной сетью. Другой пример – аудио и видео кодеки, которые занимаются кодированием аудио и видео, их сжатием и распаковкой. Эти операции выполняются в большом числе встроенных устройств, поэтому это как раз пример такой функции, которую можно было бы поместить в IP-ядре.

Когда вы делаете встроенную систему, то будет полезно не «изобретать велосипед», а воспользоваться уже готовыми IP-ядрами. Вы можете посмотреть каталоги фирм, которые делают IP-ядра, например, Texas Instruments. Такие фирмы выпускают множество специализированных микросхем, и вы сможете найти то, что вам нужно. Например, если требуется реализовать сжатие по стандарту MPEG, то у Texas Instruments вы найдете несколько десятков микросхем, которые его делают, и выбрать подходящую.

IP-ядра должны взаимодействовать с микроконтроллером, который является центром всей системы. Он управляет IP-ядрами и командует, когда они должны начинать работу, передает им информацию и получает результат. Например, если IP-ядро выполняет сжатие видео, то микроконтроллер говорит ему, когда начинать сжимать данные и какие данные сжимать. Для этого микроконтроллер передает ему определенную последовательность

сигналов, а когда микросхема заканчивает свою работу она посылает определенные сигналы микроконтроллеру, сообщая, что результат готов.

Другой вид компонентов, который применяется во встроенных системах, это программируемые пользователем вентильные матрицы, или ППВМ.

Мы не будем рассматривать ППВМ в этом пособии, потому что это достаточно сложные, хотя и интересные устройства, но вы должны знать, что это такое. ППВМ – это аппаратные устройства, а если быть точнее – аппаратно программируемые микросхемы. Что это значит? Обычная микросхема состоит из огромного количества построенных на базе транзисторов логических вентилей. Вентили, в свою очередь, соединены между собой очень сложными многочисленными связями, по которым, собственно, и путешествуют управляющие и информационные сигналы. Схема объединения этих микроскопических устройств и обусловливает логику выполняемых микросхемой операций, но только эта схема «печатается» на заводе. В свою очередь ППВМ можно образно сравнить с набором юного мехатроника, состоящего из нескольких десятков транзисторов, резисторов, лампочек, кнопок, проводов, клемм, батареек, колесиков, приводов и прочих элементов. Имея этот набор, вы можете собрать из него аппаратное устройство, которое будет выполнять какую-то задачу – например, машинку, умеющую ездить по поверхности. Затем, если вам понадобится другое устройство, например, лодка с мотором, которая способна плавать по воде, вы разберете машинку на составные части и перекомпонуете их в кораблик. Это совсем образно можно представить как процесс аппаратного программирования набора юного мехатроника. ППВМ – это сложная, хитро устроенная микросхема, которая позволяет менять рисунок взаимосвязей своих логических компонент, т.е. по сути *аппаратную конфигурацию* устройства. Здесь важно еще раз подчеркнуть, что речь идет не о перепрошивке программной начинки устройства, а об изменении его аппаратной структуры.

Что это нам дает? Главное преимущество ППВМ в том, что они работают быстрее, чем обычные микропроцессоры, выполняющие те же функции, но программным путем. ППВМ может заменить специализированную микросхему, такую как IP-ядро. С ППВМ не требуется разработка и производство специальной микросхемы, так

как можно взять ППВМ и запрограммировать ее на выполнение нужной задачи, а это гораздо дешевле. Специализированные же микросхемы, в свою очередь, могут работать еще быстрее, чем ППВМ, и они будут дешевле, если их производить в больших количествах, потому что они не включают в себя механизмы переконфигурирования, которые есть в ППВМ. Но это обосновано только для случая микросхем с широко востребованной логикой. Иными словами, все, как обычно, зависит от решаемой задачи, под которую вы подбираете наиболее эффективную конфигурацию.

Вопросы для самоконтроля

1. Выберите верные утверждения:
   a) Встроенная система сама генерирует данные, производит расчеты и выводит их во внешний мир.
   b) Встроенная система получает данные из внешнего мира с помощью набора датчиков.
   c) Светодиод может выступать в роли датчика.
   d) Сканер отпечатков пальцев может выступать в роли датчика для встроенной системы.
   e) IP-ядра изготавливают под узкоспециализированные задачи конкретных систем.
   f) IP-ядра – готовые блоки для проектирования устройств.
   g) Аналогом программирования для ППВМ выступает изменение соединения элементов.
2. Что работает быстрее: программа, выполняющаяся на обычном микропроцессоре или ППВМ, реализующая ту же логику, что и программа?

1.3 Микроконтроллеры

В этом параграфе мы продолжим говорить об аппаратных компонентах встроенных систем, а именно о микроконтроллерах.

Микроконтроллер является центром встроенной системы, поэтому в этом учебной пособии мы уделим много внимания именно

работе с ним. На Рис. 1.2 вы видите одну из плат Arduino. Это не микроконтроллер, это печатная плата с множеством элементов, среди которых, присмотревшись, вы сможете увидеть большую черную прямоугольную микросхему. Вот это как раз и есть микроконтроллер, который выполняет программы, управляющие устройством.

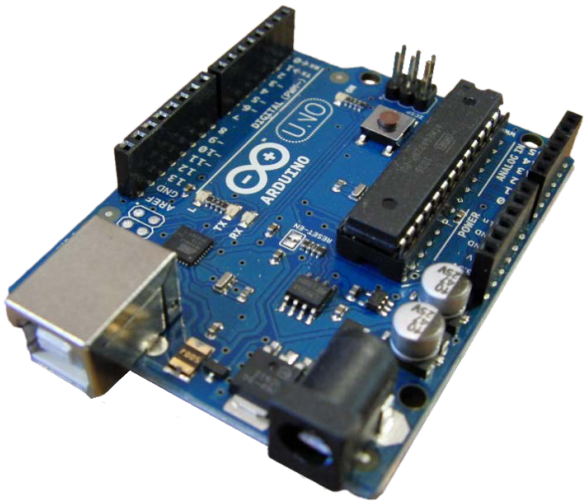


Рис. 1.2: Плата Arduino

Итак, работа данной микросхемы состоит в том, чтобы выполнять код, и это центр всей системы. Он считывает данные из других компонентов, и он также управляет другими компонентами. Чем отличается микроконтроллер от микропроцессора, который используется в обычных компьютерах? Микроконтроллер обычно меньше и слабее, чем микропроцессор, поэтому, когда говорят о последнем, то имеют в виду микросхему от компании Intel или AMD, устанавливаемую в настольный компьютер или ноутбук, и которая работает на очень большой скорости.

Микроконтроллер же, как правило, существенно более слабый. При этом с точки зрения функционирования он делает примерно то же самое, но только во встроенных системах. С этим связаны и ограничения. Все компоненты встроенных систем должны быть энергоэффективными и дешевыми и для них, чаще всего, не требуется производительность как у процессора с 6 ядрами и

частотой 4 ГГц. Если нужно реализовать *конкретное* взаимодействие с пользователем, то такая производительность скорее всего и не требуется, поэтому для встроенных систем обычно нужен дешевый процессор, потребляющий мало энергии.

Какова производительность микроконтроллеров в абсолютном выражении? Сейчас достаточно часто используется частота от 16 MHz, хотя может быть даже и меньше, например, 8 или даже 4 MHz. Это почти в несколько сот раз медленней, чем процессор обычного компьютера. Конечно, могут применять и более быстрые микросхемы – до 500 MHz или даже до 1 GHz. Это требуется, например, для обработки видеоизображений.

Другое отличие от обычных компьютеров состоит в том, что в обычных компьютерах процессор и память – это отдельные микросхемы, причем обычно память представляет собой несколько микросхем. Мы можем менять и добавлять память независимо от процессора. В микроконтроллерах начального уровня все объединено в одну микросхему, то есть по своей сути такой микроконтроллер – это мини-компьютер, сделанный на одной микросхеме, на которой находится и микропроцессор, и память, и другие необходимые компоненты. Конечно же, памяти у них тоже гораздо меньше, чем в настольных компьютерах, но такой вариант может быть удобен, так как нужно только минимальное количество внешних компонентов, чтобы микропроцессор мог работать. В самых простых случаях требуется только питание.

В более производительных микроконтроллерах используют внешнюю память, которая устанавливается отдельно. Это менее удобно, но зато позволяет выбрать, сколько памяти нам нужно, не переплачивая за не нужный в конкретном устройстве ее объем. Обычно так делают, когда микроконтроллеру для работы требуется достаточно большой объем памяти.

Итак, микроконтроллер – это интегрированная микросхема, которая выполняет программу. На Рис. 1.3 микроконтроллер выглядит немного по-другому, чем то, что мы видели раньше. На этом рисунке мы заглянули внутрь черной оболочки и увидели то, как на самом деле выглядит микросхема. Это маленький кремниевый чип, спрятанный в защитный корпус.

Фотография на Рис. 1.3 показывает микропроцессор после того, как чип был установлен в корпус.

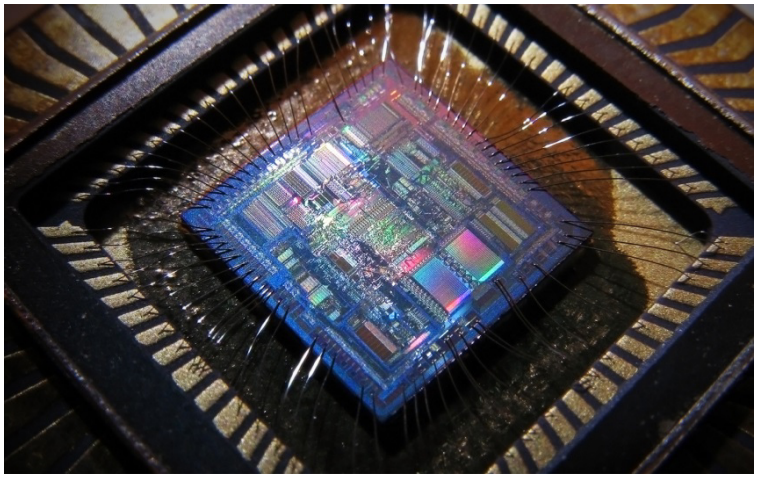


Рис. 1.3: Микроконтроллер

Сам корпус представляет собой черный материал, который окружает чип. Он нужен, прежде всего, для защиты, а кроме того, для охлаждения микросхемы, так как помогает более эффективно отводить тепло. Это необходимо, потому что все чипы сильно нагреваются во время работы. Настолько сильно, что если вы попробуете коснуться открытого чипа во время его работы, то получите такие же ощущения, как и от раскаленной лампочки (по количеству выделяемого тепла современные процессоры можно сравнить с обычными 100 ватными лампочками).

Кроме того, вы видите, что по всему периметру чипа находятся контактные площадки. Это то, посредством чего микросхема общается с внешним миром. Они соединены проволочками с чипом и выведены на маленькие кусочки металла на внешней стороне корпуса, так чтобы можно было припаять микроконтроллер к остальным компонентам системы.

Обычно микроконтроллер ставится на печатную плату, медные дорожки которой соединяют его с другими компонентами системы. Данная микросхема является центром системы, она посылает команды другим компонентам и получает от них данные. Чтобы она могла обработать эти данные, ее нужно запрограммировать. В отличие от ППВМ, аппаратную архитектуру микропроцессора менять нельзя. Зато в его память можно загрузить программу в виде набора инструкций, написанных на понятном ему языке, и попросить его ее выполнить.

Существует много языков программирования, на которых можно писать подобные программы. В рамках этого учебного пособия мы будем использовать язык Си для программирования микроконтроллера, установленный на плате Arduino (Рис. 1.2). Кроме Си используют и другие языки. Прежде чем выполнить, программу, очевидно, надо где-то сохранить. Следовательно, внутри микроконтроллера должна быть память для программы. Сейчас это, как правило, флэш-память, такая же, как в USB накопителях.

Флэш-память – это энергонезависимая память, то есть такая, из которой данные не стирается после отключения питания. Когда питание появляется вновь, микроконтроллер начинает выполнение программы с самого начала.

Где, на чем и как пишут программы для подобных устройств? Обычно это делают на обычном ноутбуке или настольном компьютере. Это связано с тем, что как уже упомянуто ранее, микроконтроллеры сравнительно медленные и слабые. Поэтому программу пишут и компилируют на мощном компьютере, а затем окончательная версия скомпилированной программы должна быть записана в память микроконтроллера. Делается это с помощью специального устройства, называемого *программатором* (Рис. 1.4).

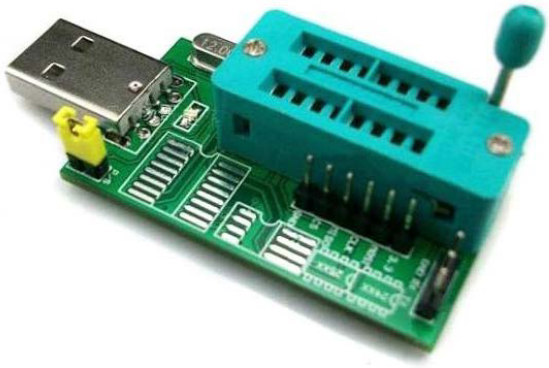


Рис. 1.4: Программатор

Это небольшая плата, которая с помощью USB присоединяется к компьютеру, а к расположенному на ней слоту подключается микроконтроллер. Есть специальные программы на компьютере, которые связываются с этим устройством и записывают через него программу. Это один из способов запрограммировать микроконтроллер. С таким программатором микроконтроллер программируют отдельно, а потом уже устанавливают в систему, где

он будет работать. Иногда бывает так, что в устройстве уже предусмотрен специальный разъем, и тогда микроконтроллер программируют прямо в системе. Но в нашем случае мы не будем так делать, потому что мы используем Arduino. Для нее не нужен отдельный программатор, так как он уже встроен в плату. Можно просто взять Arduino и, подключив ее к USB порту ноутбука или настольного ПК, программировать ее непосредственно.

Вопросы для самоконтроля

1. Отметьте верные утверждения:
   a) Работа микроконтроллера состоит в том, чтобы выполнять код.
   b) Микропроцессор меньше и слабее микроконтроллера.
   c) Процессор и память микроконтроллера начального уровня находятся на разных микросхемах.
   d) Для повышения производительности микроконтроллера используют внешнюю память.
2. Как называется устройство, с помощью которого окончательную версию программы переносят с компьютера на микроконтроллер?

1.4 Аналоговые и дискретные сигналы

В этой части мы обсудим аналоговые и цифровые сигналы. Ранее мы говорили о встроенных системах и их интерфейсе взаимодействия с окружающим миром и видели, что на входе есть набор сенсоров, которые получают информацию извне. Сенсоры могут быть очень простыми, такими как кнопка, а могут быть и сложными, как, например, микрофон, датчик освещенности или камера. С другой стороны, у встроенной системы есть исполнительные механизмы, которыми могут быть лампочки, светодиоды, динамики, экраны или что-то еще. И те, и другие обеспечивают связь встроенной системы с внешним физическим миром. Они измеряют физические параметры или наоборот, производят какие-то физические действия, а физические сигналы

чаще всего являются аналоговыми. В то же время ядро системы – микроконтроллер – работает в дискретном окружении – в мире «нулей и единиц» и знать не знает об аналоговых сигналах. Прежде чем информация из внешнего мира поступит на обработку в микропроцессор она должна быть *оцифрована*, то есть переведена из аналоговой формы в цифровую.

В чем же разница между этими формами представления сигналов? Можно привести такое сравнение – разница между аналоговыми и цифровыми сигналами такая же, как разница между действительными и целыми числами. Если вы помните, действительные числа являются непрерывными, то есть на любом интервале, какой бы вы не взяли, существует бесконечное число таких чисел. С другой стороны, вы можете взять сколь угодно большой интервал, но количество целых чисел на нем будет конечно. Собственно, разница между аналоговыми и цифровыми сигналами, по существу, именно в этом.

Давайте рассмотрим какой-нибудь «аналоговый» пример, например, свет. Предположим, на нас падает свет от лампы, и его яркость можно регулировать. Как правило, яркость света – это аналоговый параметр. Можно взять лампу, подключенную через диммер, и сделать ее немного ярче или немного темнее. В зависимости от того, насколько точное управление, можно получить очень много возможных значений между полностью выключенным и полностью включенным светом. То же самое со звуком. Можно говорить громко, можно тихо. Можно говорить с промежуточной громкостью. Громкость можно плавно регулировать между максимально громкой и максимально тихой. Это тоже аналоговое явление. Все непрерывные явления нашего мира (по крайней мере именно так их воспринимает человек) – например, время, расстояние, масса, яркость, сила, даже вкус и тому подобное – являются аналоговыми.

Физики, конечно, могут возразить, сказав, что на самом деле это не так, что мир дискретный. Существующие на субатомном уровне элементарные частицы могут находиться только в строго определенных состояниях, и это дискретный мир. Возможно, это так, но для наших человеческих чувств мир является аналоговым. Мы не можем воспринимать состояния отдельных элементарных частиц, поэтому нам кажется, что все параметры могут изменяться плавно.

Но главная проблема даже не в этом, а в том, что различных значений аналогового сигнала может быть очень-очень-очень много, настолько много, что у современной вычислительной системы просто не хватит ресурсов работать со всем его многообразием. С другой стороны, дискретное явление – это то, что может находится только в фиксированном (сравнительно, небольшом) числе разных состояний, например, «есть» или «нет», как выключатель света на стене – он или включен, или выключен. В этом случае есть всего два уровня яркости.

Другим примером могут быть часы. Бывают цифровые часы, которые показывают цифры. А бывают часы со стрелками, и если стрелки движутся плавно, то стрелка может находиться в любой промежуточной позиции, в том время как цифровые часы могут показать только целые часы, минуты, секунды.

Мы говорим об этом потому, что встроенные системы взаимодействуют с реальным миром, который, в рамках нашего восприятия, во многом аналоговый. Пусть, например, требуется датчик, который измеряет яркость света, и пусть он будет аналоговым, чтобы быть в состоянии измерить не только, что свет выключен или включен, но и степень его яркости. Проблема состоит в том, что, как уже было сказано выше, микроконтроллеры являются цифровые системами, то есть они понимают только цифровые данные, или если более конкретно – нули и единицы. Для того, чтобы программа на микроконтроллере имела возможность использовать информацию от датчиков, нужно чтобы аналоговый сигнал был преобразован в цифровое значение. И это то, для чего нужно аналого-цифровое преобразование.

Мы не будем подробно обсуждать сам процесс, так как он является сравнительно сложным. Достаточно понимать, что берут аналоговое вещественное значение и преобразуют его в целое число, которое является *приближением* к исходному аналоговому значению. Всегда есть какая-то ошибка, но это нормально, так как мы можем сделать ошибку настолько маленькой, чтобы она нас устраивала.

Итак, аналогово-цифровое преобразование (сокращенно АЦП) требуется на многих входах. На многих, но не на всех, так как некоторые из них могут быть уже цифровыми, например, кнопки. Кнопки естественным образом являются цифровыми: кнопка или нажата, или отпущена.

На выходе – со стороны исполнительных механизмов, часто может потребоваться обратное преобразование из цифрового сигнала в аналоговый. Например, микроконтроллер должен выводить звук через громкоговорители. Акустические системы являются аналоговыми устройствами, они нуждаются в аналоговом сигнале, а микроконтроллер выводит нули и единицы, поэтому тут нужно преобразование цифрового сигнала в аналоговый. Так что очень часто можно видеть преобразование аналогового сигнала в цифровой на входе системы и цифрового в аналоговый на выходе. Рассмотрим пример такого преобразования.

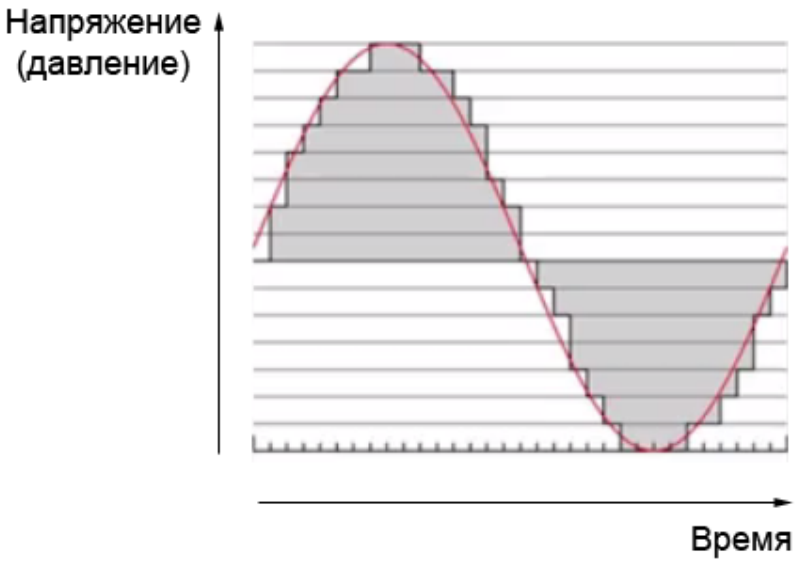


Рис. 1.5: Пример аналогово-цифрового преобразования

Пусть есть какой-то сигнал, который изменяется со временем, как показано на Рис. 1.5. Например, это звуковая волна, являющаяся изменением давления со временем. Если это чистый тон, то давление будет изменяться по функции синуса, как показано плавной линией. По оси Y задается давление воздуха, которое воспринимается микрофоном. Роль микрофона состоит в том, чтобы преобразовать давление в напряжение, которое уже будут воспринимать электронные системы. Микрофон воспринимает давление и выдает напряжение, которое изменяется так же, как изменяется давление, поэтому ось Y подписана еще и как напряжение.

Но после микрофона это все еще аналоговый сигнал, изображенный в виде гладкой, непрерывной, красной линии на графике, плавно изменяющейся во времени. Значения этого сигнала

– по-прежнему действительные числа, ведь напряжение может принимать любое промежуточное значение между максимумом и минимумом.

На этой картинке можно видеть также график ступенчатого дискретного сигнала, который может принимать только определенные значения, изображенные в виде горизонтальных линий. Эти значения соответствуют целым числам 0, 1, 2, 3 и так далее. Собственно, цифровой сигнал может принимать только такие значения, и не может принимать промежуточные.

Таким образом, аналоговый сигнал с некоторой погрешностью приближается цифровым ступенчатым сигналом. Как это делается? Для этого с определенным интервалом по времени записываются значения аналогового сигнала, а точнее ближайшие дискретные значения, которые у нас есть. Именно это делает аналогово-цифровое преобразование.

В случае обратного преобразования все происходит наоборот – цифровой сигнал посылается на исполнительный механизм, например, на динамики, которые и преобразуют его, в данном случае в звуковую волну.

Вопросы для самоконтроля

1. Выберите датчики, с которых может поступать аналоговая информация:
   a) Кнопка.
   b) Датчик, измеряющий уровень шума.
   c) Датчик освещенности.
   d) Трехпозиционный переключатель.
2. Определите верные утверждения:
   a) Микроконтроллеры работают как с цифровыми, так и аналоговыми сигналами.
   b) Аналогово-цифровое преобразование приближает аналоговый сигнал цифровым.
   c) На исполнительный механизм (например, динамик) посылается сигнал, модифицированный с помощью аналогово-цифрового преобразователя.

Тема 2

Аппаратное обеспечение

2.1 Электронные компоненты

В этой теме мы поговорим о документации на электронные компоненты. Рассмотрим, к примеру, вот этот транзистор:

Рис. 2.1: Транзистор

Если набрать в поисковике название этого транзистора, можно найти его описание и техническую документацию. Первая страница описания может выглядеть примерно так, как изображено на Рис. 2.2. В документации содержится очень много информации, которую в первое время может быть сложно понять.

FAIRCHILD
SEMICONDUCTOR®

July 2008

FGH40N60SFD
600V, 40A Field Stop IGBT

Features

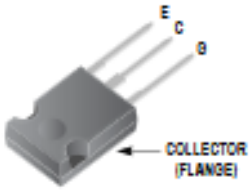
- High current capability
- Low saturation voltage: $V_{CE(sat)}$ =2.3V @ I_C = 40A
- High input impedance
- Fast switching
- RoHS compliant

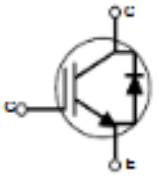
Applications

- Induction Heating, UPS, SMPS, PFC

General Description

Using Novel Field Stop IGBT Technology, Fairchild's new series of Field Stop IGBTs offer the optimum performance for Induction Heating, UPS, SMPS and PFC applications where low conduction and switching losses are essential.





Absolute Maximum Ratings

Symbol	Description		Ratings	Units
V_{CES}	Collector to Emitter Voltage		600	V
V_{GES}	Gate to Emitter Voltage		± 20	V
I_C	Collector Current	@ T_C = 25°C	80	A
	Collector Current	@ T_C = 100°C	40	A
I_{CM} (*1)	Pulsed Collector Current	@ T_C = 25°C	120	A
P_D	Maximum Power Dissipation	@ T_C = 25°C	290	W
	Maximum Power Dissipation	@ T_C = 100°C	116	W
T_J	Operating Junction Temperature		-55 to +150	°C
T_{stg}	Storage Temperature Range		-55 to +150	°C
T_L	Maximum Lead Temp. for soldering Purposes, 1/8" from case for 5 seconds		300	°C

Notes:
*1: Repetitive rating: Pulse width limited by max. junction temperature

Thermal Characteristics

Symbol	Parameter	Typ.	Max.	Units
$R_{\theta JC}$(IGBT)	Thermal Resistance, Junction to Case	-	0.43	°C/W
$R_{\theta JC}$(Diode)	Thermal Resistance, Junction to Case	-	1.45	°C/W
$R_{\theta JA}$	Thermal Resistance, Junction to Ambient	-	40	°C/W

Рис. 2.2: Пример страницы с описанием транзистора

К тому же многие компоненты производятся за границей и описания есть только на английском языке. Иногда можно найти его и на

русском языке, например, на сайте www.chipdip.ru, но чаще всего оно будет очень кратким, содержащим только самые основные параметры. На Рис. 2.3, например, изображены пол страницы документации на 14 различных транзисторов, а если посмотреть в английскую версию, то там на каждый транзистор будет по 14 страниц. Поэтому лучше учить технический английский язык, что, к счастью, не очень сложно сделать, так как множество наиболее часто используемых слов из области информационных технологий, как и любой другой узкоспециализированной сферы, сравнительно небольшое. Если вы только начинаете разбираться с электроникой, то вы вряд ли сможете понять все, что написано в документации. Но это не страшно, чаще всего понимать все целиком даже и не требуется.

IGBT ТРАНЗИСТОРЫ

Infineon technologies

IGBT транзисторы фирмы **Infineon** выполнены по NPT технологии, которая позволила значительно улучшить рабочие характеристики приборов.
IGBT транзисторы выпускаются в разнообразных корпусах. Технология EmCon фирмы Infineon позволяет интегрировать в одном корпусе транзистор и быстродействующий обратный диод.
Диапазон рабочих токов 2–30 А, рабочее напряжение до 1200 В.
Диапазон рабочих температур: -55...+150°C.

СИСТЕМА ОБОЗНАЧЕНИЙ

SG	P	30	N	120
1	2	3		4

1. **Тип транзистора**
   BUP - IGBT
   SG - быстродействующий IGBT
   SK - быстродействующий IGBT с обратным диодом
2. **Тип корпуса:** D – TO-252AA, I – TO-262, B – TO-263AB, P (или 2) – TO-220AB, W – TO-247AC, 3 – TO-218AB
3. **Ток коллектора, А**
4. **Напряжение коллектор-эмиттер (x10), В**

Наименование	Частотный диапазон, кГц	Напряж. коллектор-эмиттер, В	Напряжение коллектор-эмиттер (откр.), В	Ток коллектора, А	Мощность рассеивания, Вт	Тип корпуса	Семейство
BUP212	10	1200	3.4	22	125	TO-220AB	IGBT
BUP213			3.3	32	200		
BUP313			3.4	32	200	TO-218AB	
BUP314			3.4	52	300		
BUP311D			3.4	20	125		IGBT с обратным диодом
BUP313D			3.4	32	200		
BUP314D			3.4	42	300		
SGP30N60	30	600	2.5	30	250	TO-220AB	быстродействующий IGBT
SGW30N60						TO-247AC	
SGP02N120		1200	3.1	2	62	TO-220AB	
SGP07N120		1200	3.1	7	125	TO-220AB	
SGW25N120		1200	3.1	25	313	TO-247AC	
SKP15N60		600	2.3	15	139	TO-220AB	быстродействующий IGBT с обратным диодом
SKW25N120		1200	3.7	25	313	TO-247AC	

ТИПЫ КОРПУСОВ

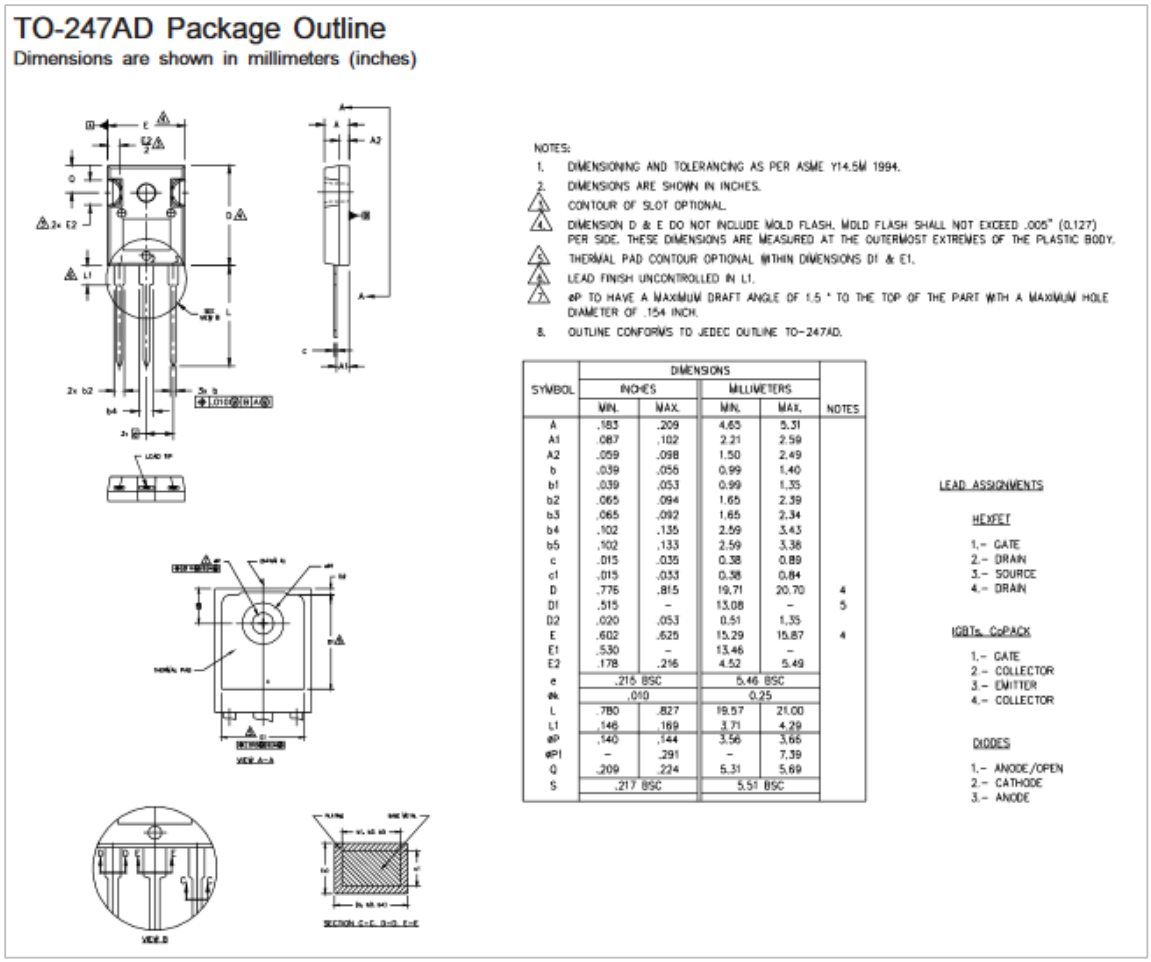
TO-220AB

TO-247AC

TO-218AB

Рис. 2.3: Пример документации по транзистору

Итак, какую информацию мы можем почерпнуть из документации? Одна вещь, которая всегда там есть, это размеры корпуса. Так как различные компоненты встроенных систем всегда нужно соединять вместе, инженеры должны знать, какой у них размер, чтобы можно было разработать подходящую схему платы.



SYMBOL	DIMENSIONS				NOTES
	INCHES		MILLIMETERS		
	MIN.	MAX.	MIN.	MAX.	
A	.183	.209	4.65	5.31	
A1	.087	.102	2.21	2.59	
A2	.059	.098	1.50	2.49	
b	.039	.055	0.99	1.40	
b1	.039	.053	0.99	1.35	
b2	.065	.094	1.65	2.39	
b3	.065	.092	1.65	2.34	
b4	.102	.135	2.59	3.43	
b5	.102	.133	2.59	3.38	
c	.015	.035	0.38	0.89	
c1	.015	.033	0.38	0.84	
D	.776	.815	19.71	20.70	4
D1	.515	–	13.08	–	5
D2	.020	.053	0.51	1.35	
E	.602	.625	15.29	15.87	4
E1	.530	–	13.46	–	
E2	.178	.216	4.52	5.49	
e	.215 BSC		5.46 BSC		
øk	.010		0.25		
L	.780	.827	19.57	21.00	
L1	.146	.169	3.71	4.29	
øP	.140	.144	3.56	3.66	
øP1	–	.291	–	7.39	
Q	.209	.224	5.31	5.69	
S	.217 BSC		5.51 BSC		

Рис. 2.4: Чертеж транзистора

Поэтому, в документации в обязательном порядке будут указаны размеры, как, например, в чертеже транзистора на Рис. 2.4. Часто на чертеже размеры показаны буквами, а рядом указана таблица, в которой написано, какой букве какой размер соответствует. В качестве единиц измерения обычно используют и дюймы, и миллиметры.

Габариты компонентов важны, потому что, если, например, использовать макетную плату с шагом отверстий, в которые вставляются компоненты, в 2.54 мм, то с ней можно будет использовать только те компоненты, шаг между ножками которых равен или кратен 2.54 мм, иначе их просто не получится вставить. Очевидно, что все это надо знать до покупки соответствующих деталей.

Во-вторых, если нужно спроектировать очень компактное устройство, то размеры исходных элементов, очевидно, тоже будут

очень важны, так как придется выбирать самые маленькие компоненты, чтобы уложится в заданные размеры.

В документации, конечно, есть и другая информация. Там обязательно есть электрические и температурные параметры. Температурные параметры могут быть важны, если устройство должно работать зимой на улице или наоборот, оно предназначено для производств с высокой температурой. Если устройство не будет использоваться в экстремальных условиях, то температурные характеристики не важны.

Электрические параметры включают такие важные характеристики, как, например, значения минимальных и максимальных напряжений и токов, которые может выдержать компонент, а также рабочие диапазоны этих величин. Это важно, потому что нужно, быть уверенным, что все компоненты совместимы по напряжению. Например, микроконтроллер работает на напряжении 5 вольт и, соответственно, выдает все сигналы с напряжением 5 вольт, а какой-то другой компонент, например, IP-ядро, работает на уровне 3.3 вольта. Такие компоненты нельзя соединить напрямую, они не смогут друг с другом работать, и может быть вообще «сгорят». В этом случае понадобятся специальные устройства, которые будут переводить уровни напряжения от 5 к 3.3 вольтам и обратно. Или же нужно сразу подбирать компоненты, работающие на одном напряжении.

Как уже было упомянут, обычно в документации есть два вида электрических характеристик. Один, который будет указан в начале, это максимальные допустимые значения (на английском «Absolute maximum ratings»). Эти характеристики точно не надо превышать, потому что компонент, скорее всего, сразу же сгорит. Например, транзистор с Рис. 2.1 не выдержит, если подать больше 600 вольт между одной из пар его выводов и больше 20 между другими. И он не может пропускать больше 80 ампер. 80 ампер, это, конечно, очень много, и речь идет об очень мощном транзисторе, но, если ток будет еще больше, то он быстро перегреется и сгорит.

Второй вид электрических характеристик, указанных в документации, это так называемые рабочие характеристики компонентов, то есть те, с которыми они могут работать в штатном режиме.

Объемы и сложность технической документации сильно зависят от самих компонент, которые могут быть очень разными. Некоторые из них очень простые, как например, резисторы, которые можно описать всего несколькими параметрами. Но чем сложнее компонент, тем больше у него будет параметров и тем сложнее будет документация. Например, микросхемы могут быть очень сложными, и документация на них может иметь размер в сотни страниц. В частности микроконтроллеры – это очень сложные микросхемы, для описания которых требуется не только указать внешние параметры вроде размеров, напряжения и прочих характеристик, но еще и полностью описать внутреннюю логику работы микросхемы, что иногда может занимать больше места, чем все внешние характеристики.

К счастью, в настоящем учебном пособии нам не нужно глубокого понимания всех деталей работы микроконтроллера и поэтому нам не потребуется изучать его объемную документацию. Мы будем использовать плату Arduino, а для нее есть библиотеки, позволяющие простыми средствами использовать возможности установленных на этих платах микроконтроллеров. Хотя, конечно, нужно помнить, что библиотеки, скрывая детали реализации, также скрывают и все потенциальные возможности. Могут быть такие задачи, которые можно решить с помощью микроконтроллера, но для которых нет библиотеки, или библиотека решает их немного не так, как нужно. Бывает, что в микроконтроллере какой-то компонент может работать в разных режимах, а библиотека позволяет использовать только один из них. Или библиотека делает это медленнее, чем, если управлять микроконтроллером напрямую. Поэтому, в последствии вам безусловно необходимо будет научиться понимать эти огромные руководства, научиться напрямую работать с микроконтроллером.

Вопросы для самоконтроля

1. Выберите верные утверждения:
   a) Обычно в документации есть два вида электрических характеристик: предельные и рабочие.
   b) У всех компонентов шаг выводов одинаковый.

c) В документации содержится информация о температурных и электрических параметрах.

2. В каких случаях транзистор сгорит (Рис. 3.2 этого раздела):
   a) Подали 800 вольт между одной из пар выводов.
   b) Температура окружающей его среды +30C.
   c) Ток через транзистор составил 100 ампер.

2.2 Микроконтроллеры

В этом разделе мы обсудим характеристики микроконтроллеров. Когда разрабатывают встроенное устройство, то одно из первых решений, которое нужно принять, это какой микроконтроллер использовать для управления системой.

Не выбрав микроконтроллер, нельзя начать писать управляющую программу. Схему и печатную плату тоже нельзя начать делать, потому что не понято даже, какое питание нужно и какой размер контроллера и сколько у него ножек для подключения.

Для этого можно посмотреть в документацию и попытаться понять, какой микроконтроллер имеет те характеристики и возможности, которые нужны. Как уже было сказано в предыдущем разделе, нам в рамках этого учебного пособия такие решения принимать не придется, потому что мы будем использовать платы Arduino, а на них уже есть определенный микроконтроллер, который мы и будем использовать. Но если вы будете разрабатывать настоящие устройства, то вам придется решать и эту задачу, поэтому далее мы кратко обсудим, какие основные характеристики нужно учитывать.

Одна из основных характеристик – это разрядность микроконтроллера. Что она означает? Как и в большинстве современных вычислительных систем память, с которой работает микроконтроллер, логически и схематически отделена от него (хоть и может находиться с ним на одном чипе), и поэтому микроконтроллер не может напрямую выполнять операции с данными, которые в ней записаны (например, складывать). Для этого ему нужно скопировать эти данные в специальные ячейки памяти, которые как раз являются непосредственной частью микросхемы и называются *регистрами*. После того, как операция

будет выполнена, данные из регистров, как правило, копируются обратно в память. Так сделано потому, что регистровая память, которая работает на тех же скоростях, что и микроконтроллер, является очень дорогой, поэтому ее не может быть много. С другой стороны, основная память, вынесенная в свою собственную микросхему, делается по несколько иной технологии, что позволяет ее существенно удешевить и, соответственно, сделать ее больше, правда за все надо платить, поэтому подобная память, хоть ее и много, работает существенно медленнее, чем регистровая – в десятки, а иногда и в сотни раз! Поэтому основной цикл работы современного микропроцессора выглядит следующим образом: скопировать нужные данные из основной памяти в регистры, выполнить операцию над регистрами, скопировать результат обратно во внешнюю память или продолжить вычисления над значениями, все еще находящимися в регистрах.

Данные, которые находятся в регистрах, как, собственно, и везде в компьютерах, представлены в двоичном виде, то есть в виде нулей и единиц, а регистры, в свою очередь, имеют конкретный размер, определяющий, сколько двоичных разрядов или сколько бит можно в него записать. Собственно, разрядность и есть данный размер регистра – количество бит, которые можно в него записать.

Если взять, например, очень простой микроконтроллер, то он может иметь разрядность 8 бит (иногда говорят, что он является 8-ми битным). Это значит, что для обработки данных он использует регистры размером в 8 бит. В такой регистр можно записывать числа от 0 до 255, потому что 255 это самое большое число, которое можно записать в двоичном виде, используя 8 разрядов. Так что такой процессор может работать непосредственно только с числами, значение которых не больше 255. Если нужно работать с числами больше, то даже для самых простых операций придется писать программу. Например, для сложения эта программа будет складывать сначала младшие 8 бит числа, а потом, учитывая возможный перенос, старшие 8 бит числа. Если вы не программируете на ассемблере, а программируете, например, на языке Си, то вам, скорее всего, не придется самим писать такую программу, так как ее для вас сгенерирует компилятор, но все равно такое сложение будет состоять из нескольких мелких операций процессора и будет выполняться медленнее, чем когда он изначально

имел бы бóльшую разрядность и мог бы уместить числа целиком в своих регистрах.

Другой распространенный вариант разрядности – это 32 бита. 32 битный процессор работает с числами, которые можно представить последовательностью из 32 нулей и единиц и поэтому он должен иметь такой же размер регистров. Он также должен иметь механизмы, которые позволяют ему быстро передавать данные такого размера в и из памяти. Такой процессор при прочих равных условиях будет работать быстрее, если нужно обрабатывать числа больше 255.

Итак, разрядность многое говорит о производительности процессора. Грубо говоря, чем больше разрядность, тем выше производительность, потому что можно обработать больше информации за один раз.

Разрядность это всего лишь один из факторов, которые определяют производительность микроконтроллеров и микропроцессоров. Таких факторов очень много, что делает сравнение производительности сложной задачей, на которую не всегда можно дать однозначный ответ. Но есть еще один важный параметр, который влияет на производительность. Это тактовая частота.

Тактовая частота – это скорость, с которой процессор выполняет одну простую инструкцию. Вернее сказать, это количество простых инструкций, которые процессор выполняет за 1 секунду. Процессор с тактовой частотой 8 MHz выполняет 8 миллионов операций в секунду, а если частота процессора 1 GHz, то он выполняет миллиард операций в секунду. Здесь принципиально понятие простой операции. Это далеко не всегда соответствует одному оператору языка программирования. Это даже не всегда соответствует одной операции языка ассемблера. Даже если взять простейшую операцию – скопировать значение из ячейки памяти в регистр микроконтроллера, то этот процесс может занимать сразу несколько тактов его работы. Это связано с тем, что как уже было сказано выше, основная память работает существенно медленнее процессора и пока запрос на информацию дойдет до нее и данные будут переданы назад, центральный процессора успеет совершить много тактов. Или, например, бывают достаточно сложные инструкции, которые выполняются не за один, а за несколько тактов – скажем операции,

работающие сразу с массивом данных и т.д. Конечно, в современных архитектурах микроконтроллеров и микропроцессоров используется большое количество хитрых и сложных приемов, позволяющих повысить производительность системы, несмотря на подобный перекос в скоростях работы различных компонент системы и сложности выполняемых инструкций. Поэтому можно считать, что тактовая частота напрямую влияет на производительность системы, и если нужно, чтобы инструкции выполнялись быстрее, а значит, программа выполнялась быстрее, то нужна и более высокая тактовая частота.

Итак, с точки зрения производительности, с одной стороны есть простые 8-битные микроконтроллеры, работающие на частотах, например, 4, 8, 16 MHz. С другой стороны, есть 32 битные микроконтроллеры, которые работают на частотах 500 MHz и больше, 1 GHz. Это очень большая разница в производительности, но и, очевидно, в цене и, что не менее важно, в энергопотреблении. Для каждой задачи нужно выбирать подходящий контроллер.

Входы и выходы микросхемы тоже очень важны, и они часто являются узким местом – ресурсом, которого не хватает. Может быть, вы слышали, что есть такой закон Мура. Этот закон утверждает, что с каждым годом микросхемы становятся все сложнее и быстрее. Однако, хотя производительность быстро растет, число контактов, через которые микроконтроллер общается с внешним миром, остается примерно одинаковым. Контакты (их еще называют портами ввода-вывода), являются в виду этого ценным ресурсом. Грубо говоря, если микроконтроллер должен быть связан еще с пятью устройствами и каждое из них требует определенное число контактов для связи, то нужно сложить их число для всех 5 устройств, и получим минимальное число контактов, которое должно быть у микросхемы. И это может сразу отсеять много вариантов. Таким образом, число входов и выходов определяет, насколько удобно соединять микроконтроллер с другими устройствами и сколько таких связей может быть. Это очень важный фактор.

Поддержка коммуникационных протоколов – другой важный момент, который описывает возможности связи микроконтроллера с другими компонентами. Для чего нужны эти протоколы? Как уже было неоднократно сказано, микроконтроллер должен обмениваться данными с другими компонентами. Они передают друг другу много

данных, и делают это строго определенным образом. Чаще всего недостаточно просто передать 0 или 1. Если нужно обмениваться большим количеством данных (а помимо самих данных есть еще и управляющие, синхронизирующие, контрольные и прочие сигналы), то обычно устанавливается порядок, в котором осуществляются коммуникации. Этот порядок, или набор правил, и называется протоколом.

Существует много протоколов: UART, i²C, SPI, USB и другие. Мы рассмотрим часть из них позднее. Это общеизвестные протоколы, используемые микросхемами и другими компонентами, причем очевидно, одно устройство не обязательно поддерживает их все. Например, какая-то микросхема для сжатия звука может работать с использованием только i²C, и это нужно знать и учитывать, потому что тогда и микроконтроллер должен поддерживать i²C, иначе он не сможет связаться с этой микросхемой.

Так что поддержка стандартных протоколов – это тоже важный параметр. Важно не только то, сколько контактов для общения есть, но и какие протоколы реализованы во всех компонентах системы.

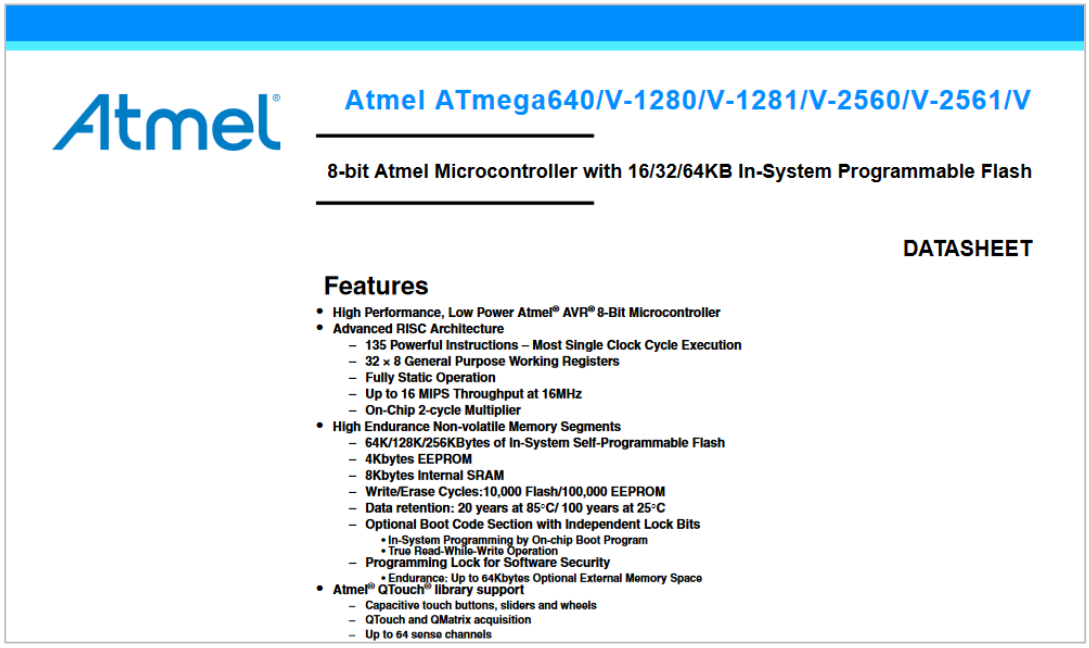
Еще одним важным компонентом микроконтроллера является таймер. Он может использоваться для того, чтобы измерять интервалы времени или создавать задержку на определенное время, для того, чтобы генерировать специальные сигналы для управления приводами, но об этом мы поговорим позже. Один из вариантов использования таймера – оцифровка звука. Например, нужно записывать уровни звука (напряжение с микрофона) с частотой 4 KHz. Для этого необходимо, чтобы кто-то запускал измерение в фиксированные интервалы времени 4 тысячи раз в секунду. Этим как раз и занимается таймер.

Цифро-аналоговые и аналого-цифровые преобразователи, о которых мы говорили ранее в этой главе, также могут быть встроены в микроконтроллер.

Еще один важный аспект, который следует учитывать при выборе микроконтроллера – это режимы низкого потребления энергии. Встроенные системы часто работают от батареек, и важно, чтобы устройство потребляло как можно меньше энергии и работало как можно дольше. Чтобы сократить потребление питания используют микроконтроллеры с режимами пониженного потребления, в которых неиспользуемые подсистемы отключаются в те моменты,

когда они не нужны. Такие системы управления питанием могут быть очень сложными и поддерживать несколько режимов (например, обычный, сон, глубокий сон и так далее). Как правило, эти и другие основные характеристики описаны в начале технической документации, так что можно быстро посмотреть и понять, есть они в этом микроконтроллере или нет. Для того, чтобы понять, как именно они работают и как их использовать, потребуется уже более подробно изучить документацию.

Возможно, один из самых важных ресурсов любого микроконтроллера – это память, чтобы хранить программы и данные. Рассмотрим, какие виды памяти встречаются. На Рис. 2.5 мы видим начало документации на один из микроконтроллеров ATmega фирмы Atmel, которые могут использоваться на платах Arduino.


Atmel

Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V

8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash

DATASHEET

Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 × 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles:10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
    - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix acquisition
  - Up to 64 sense channels

Рис. 2.5: Страница с документацией на контроллер Arduino

Здесь говорится, что это 8 битный микроконтроллер и что он может работать на частоте до 16 MHz. Процессор умеет выполнять 135 различных инструкций, большая часть из которых выполняются за один такт. Большинство остальных параметров описывают виды памяти, которые в нем есть.

Почему бывает так много видов памяти? Одна из причин этого в необходимости баланса между ценой и объемом. Есть разные

способы сделать память, но всегда получается, что чем она быстрее, тем дороже.

Самая быстрая память – регистровая. Процессор может за один такт прочитать и записать данные из нескольких регистров, например, из двух или трех. Например, он может прочитать данные из двух регистров, сложить их и записать в третий регистр, и он это успевает все это сделать за один такт.

Регистры быстрые, но они дорогие и поэтому их мало. Например, в этом микроконтроллере ATMega 32 восьмибитных регистра общего назначения. Это те регистры, где можно хранить значения переменных, с которыми работает программа. Есть еще специальные регистры, использующиеся для того, чтобы хранить режимы работы процессора, адрес текущей выполняемой инструкции и т.д., но их нельзя использовать напрямую для обработки данных.

Очевидно, что 32-х регистров мало, чтобы хранить данные и программу. Их хватает только для того, чтобы организовать текущий процесс вычислений, записывая в них промежуточные результаты. Для хранения основных данных используется *оперативная память* (или ее еще называют основной). В процессоре на Рис. 2.5 она называется SRAM. Ее гораздо больше – тут ее целых 8 килобайт. Она существенно дешевле, но и медленнее. Как уже говорилось ранее, современным процессорам, которые используются в настольных компьютерах, нужны сотни тактов, чтобы прочитать значение из памяти. В медленных микроконтроллерах разница не так велика, но все равно нужно больше времени, чтобы прочитать данные из SRAM, чем из регистра.

Другая проблема с памятью это энергонезависимость. Регистры и SRAM теряют свое содержимое, если выключить питание. Очевидно, что в микроконтроллере должна постоянно храниться программа, которую он выполняет, иначе пропадает всякий смысл его использования, так как после включения микропроцессор не будет знать, что ему делать. В обычных компьютерах для решения этой проблемы используют так называемую *внешнюю память* – жесткие диски. В микроконтроллерах с этой целью используют флэш-память, которая является такой же самой, как и в USB-накопителях. Правда у нее есть свои недостатки. Во-первых, она имеет ограниченное число циклов перезаписи, и может испортиться, если в нее часто что-нибудь записывать. Во-вторых, данные в нее

записываются только большими фрагментами (это обусловлено ее внутренней архитектурой), аоэтому, она подходит для хранения программы, но не очень хорошо подходит, если нужно хранить какие-то данные, которые часто меняются во время работы устройства и должны сохраняться при выключении питания.

Поэтому есть еще четвертый вид памяти, который называется EEPROM. Это тоже энергонезависимая память, но она сделана по другой технологии. Ее можно перезаписывать отдельными байтами и у нее гораздо больше рабочих циклов записи, так что ее можно менять часто. В частности, из инструкции на Рис. 2.5 мы видим, что флэш-память допускает 10000 циклов перезаписи, а EEPROM – до 100000 циклов. Но она дороже и поэтому ее меньше, чем флэш-памяти.

Изначально может быть сложно понять, сколько потребуется памяти в будущем для конкретного устройства. Хотя некоторые вещи можно оценить. Пусть, например, наша встроенная система будет обрабатывать звук длительностью в 1 секунду и с частотой 4 KHz, то это значит, что потребуется 4000 байт, если для хранения каждого отсчета используется 1 байт. Может быть потребуется два таких буфера – один обрабатывается, а в другой в это же время идет запись нового звука. Кроме этого, потребуется память под разные дополнительные переменные. Оценить код программы сложнее. Если писать программу на языке Си, то сколько она займет, заранее непонятно. Если программа простая, то она займет, например, несколько килобайт, а более сложная и большая программа может занять после компиляции существенно больше. Это станет понятно, когда уже большая часть программы написана. Поэтому может получиться так, что в выбранный процессор программа не помещается – тогда нужно попытаться написать программу более компактно или использовать процессор, в котором больше памяти.

Вопросы для самоконтроля

1. Выберите верные утверждения:
   а) При выборе микроконтроллера главную роль играет его тактовая частота, тогда как разрядность – маловажная деталь.

b) Разрядность сильно влияет на производительность процессора.
c) Производительность микроконтроллеров растет медленно, а число контактов для общения с внешним миром растет быстро.
d) Протоколы используются при обмене данными между микроконтроллером и другими компонентами.
e) Таймер, цифро-аналоговые и аналого-цифровые преобразования, память – все это может входить в микроконтроллер.

2. Какие из следующих видов памяти являются энергонезависимыми:
   a) Оперативная память (SRAM).
   b) Флэш-память.
   c) Регистры.
   d) EEPROM

2.3 Программирование микроконтроллеров

В этом разделе мы поговорим о программировании микроконтроллеров. Точнее, как обрабатывается программа перед тем, как ее выполнит микроконтроллер.

Ключевым моментом здесь является то, микроконтроллер, как вы, наверное, догадываетесь, на самом деле не выполняет буквально тот код, который пишут программисты. Сначала он должен быть обработан транслятором, а затем уже он выполняется микроконтроллером.

Зачем нужен этот дополнительный этап? Он нужен потому, что микроконтроллер, как и любой другой процессор, не понимает языков Си, или C++, Java, Паскаль, Питон и любые другие языки, на которых обычно пишутся программы. Он понимает и может выполнять только свой собственный машинный язык.

Возьмем процессор Intel. Он понимает машинный язык X86. Другие процессоры, например, микроконтроллеры Atmel, которые стоят на платах Arduino, понимают свой машинный язык. Различные семейства процессоров используют свой собственный язык. Что он из себя представляет? Это набор простых инструкций, закодированных

в двоичном формате с помощью нулей и единиц. Так что если вы посмотрите непосредственно на машинный код, то все, что вы увидите – это нули и единицы. Это, конечно, редко кто делает, но если и делает, то обычно код записывается в шестнадцатеричном формате – так он занимает меньше места и чуть более удобен для восприятия. Но в любом случае машинный язык не выглядит читаемым для человека, хотя это то, что машина на самом деле выполняет.

Для того, чтобы сделать программы более удобочитаемыми их переводят на язык ассемблера, которые стоит из коротких и понятных мнемонических команд, вроде `ADD`, `SUB`, `MUL`, `MOV`, `JMP` и т.д. Никакой выразительной мощности ассемблер не добавляет, так как по сути, это однозначное отображение из машинных кодов в слова. Он существует только ради удобства чтения машинных кодов человеком. Например, вместо `10110110 10111001 01100110 01010101` (или в шестнадцатеричном формате – `B6 B9 66 55`) будет написано `ADD R1, R2, R3`, что означает сложить значения в регистрах `R2` и `R3` и поместить результат в `R1`.

Хоть язык ассемблера и придумали для облегчения чтения и записи программ, но все-таки это очень низкий уровень. Ассемблерный язык называется языком с метками, потому что напротив любой инструкции можно поставить метку, а затем перейти к ней с помощью команд условного и безусловного перехода (вроде goto в высокоуровневых языках). Причем переходы по меткам – это единственный способ организовать ветвления и циклы, так как в подобных языках нет ни циклов for или while, ни возможности объединить инструкции в блоки кода как в структурированных языках программирования. Можно писать код на языке ассемблера, но это трудно, и мы не будем делать это в данном пособии. Иногда это необходимо, чтобы достичь максимальной производительности. Мы будем работать с языками более высокого уровня – Си, C++, Python, Lua.

Таким образом, программа, написанная на языке высокого уровня, должна быть переведена на машинный язык, прежде чем ее сможет выполнить микроконтроллер. Существует два способа, которыми это можно сделать, в зависимости от того, используется ли *компилируемый* или *интерпретируемый* язык. Если использовать компилируемый язык, то программа на этом языке сразу целиком

переводится в машинный код. После такого перевода вы получите то, что называется исполняемым файлом, содержащим именно тот код, который выполняется каждый раз при запуске программы. Примеры компилируемых языков – Си, C++, Паскаль и другие. Когда мы будем использовать платы Arduino, мы будем работать с упрощенным языком C++.

С другой стороны, кроме компиляции, есть также интерпретация. В интерпретируемой среде инструкции на языке высокого уровня, преобразуются в машинный код каждый раз во время выполнения с помощью интерпретатора. Примерами таких языков является Питон, Java, Lua и другие. Недостаток интепретируемых программ в том, что они работают существенной медленнее, ведь процессор выполняет не их, а код интерпретатора, который, в свою очередь, выполняет вашу программу! Скомпилированная же программа, как уже было сказано ранее, занимает центральный процессор целиком и полностью без всяких посредников. Но интерпретация имеет и преимущества, поскольку она освобождает программиста от определенных задач. Так, например, если в Си вы хотите использовать динамический массив, вам нужно вызвать специальную функцию для получения памяти, а затем не забыть освободить эту память. Это очень сложно отследить и сделать корректно в больших программах, поэтому часто возникают ошибки. С другой стороны, в интерпретируемых языках вы просто используете готовый динамический массив, а интерпретатор решает все задачи с управлением памятью. Поэтому какой язык использовать, как всегда, зависит от решаемой задачи.

Вопросы для самоконтроля

1. Расположите в порядке от низкого к высокому уровню следующие способы написания кода:
   a) Код на Си, Java, Python.
   b) Машинный код.
   c) Язык ассемблера.
2. Выберите верные утверждения:
   a) Python является компилируемым языком.

b) Инструкции компилируемого языка целиком переводятся в машинный код только один раз.
c) Пример компилируемого языка – C++.
d) Код, написанный на компилируемом языке, выполняется медленнее, чем код, написанный на интерпретируемом языке.

Тема 3

Arduino

3.1 Платформа Arduino. Плата Arduino Uno

Arduino — это электронный конструктор и удобная платформа быстрой разработки электронных устройств для новичков и профессионалов. Платформа пользуется огромной популярностью во всем мире благодаря удобству и простоте языка программирования, а также открытой архитектуре и программному коду. Устройство программируется через USB без использования программаторов.

Микроконтроллер на плате программируется при помощи языка Arduino (основан на языках Си и C++) и собственной среды разработки, которая доступна для бесплатного скачивания. Проекты устройств, основанные на Arduino, могут работать самостоятельно, либо же взаимодействовать с программным обеспечением на компьютере. Платы могут быть собраны пользователем самостоятельно или куплены уже в сборе, причем исходные чертежи схем являются общедоступными, пользователи могут применять их по своему усмотрению.

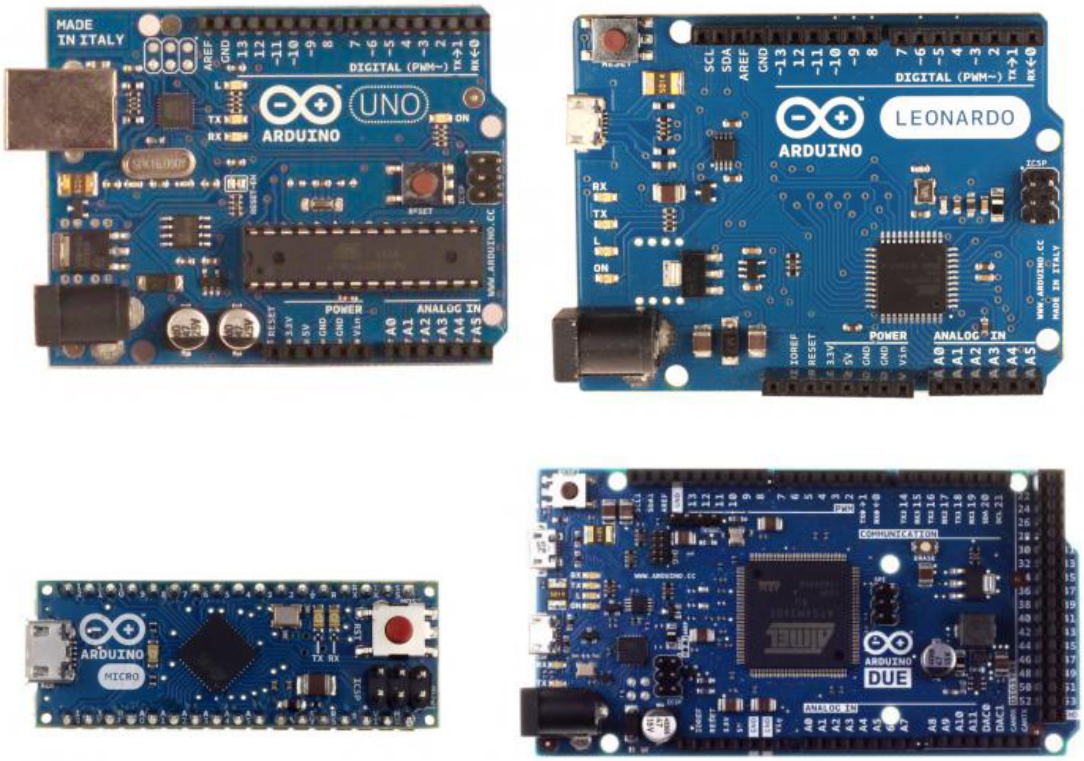


Рис. 3.1: Примеры плат Arduino

В настоящее время выпущено несколько плат из серии Arduino, некоторые из которых изображены на Рис. 3.1. В данном учебном пособии мы будем использовать Arduino Uno, но рассматриваемые примеры должны заработать на любой из них, с учетом некоторых особенностей использования выводов у разных плат, о которых будет рассказано позже.

Большинство плат Arduino построены на 8-битных микроконтроллерах фирмы Atmel. Процессоры работают на тактовой частоте 16 MHz. Платы содержат все необходимые для работы микроконтроллера компоненты, включая схемы питания и кварцевый резонатор. Рабочее напряжение в большинстве случаев 5 вольт. Программирование микроконтроллера можно осуществлять, просто подключив его к компьютеру через порт USB.

На плате есть разъемы, позволяющие подключать внешние схемы к большинству выходов микроконтроллера. Эти разъемы позволяют использовать цифровые и аналоговые входы и выходы, ШИМ генераторы, различные цифровые интерфейсы. Для Arduino выпускается множество плат расширения, которые позволяют

использовать плату как основу для управления роботами, подключаться к компьютерным сетям через Ethernet или Wi-Fi, превращать ее в цифровой фотоаппарат и так далее.

Так как среда Arduino очень популярна, то многие разработчики микроконтроллеров делают свои платы, совместимыми с ней. Это позволит применить навыки, полученные при работе с Arduino, с другими, в том числе гораздо более мощными микроконтроллерами.

Рис. 3.2: Плата Arduino Uno

Плата Arduino Uno, изображенная на Рис. 3.2, построена на базе микроконтроллера *ATmega328*. Платформа имеет 14 цифровых входов/выходов (6 из которых могут использоваться как выходы ШИМ), 6 аналоговых входов, кварцевый генератор 16 МГц, разъем USB, разъем питания, разъем для программатора (ICSP) и кнопку перезагрузки. Для того, чтобы она заработала, ее необходимо подключить к компьютеру посредством кабеля USB, либо подать питание при помощи адаптера AC/DC или батареи.

3.2 Среда разработки Arduino

В этом разделе мы познакомимся со средой разработки Arduino IDE (Integrated Development Environment – интегрированная среда разработки). Эту среду нужно использовать, если у вас есть реальная плата Arduino.

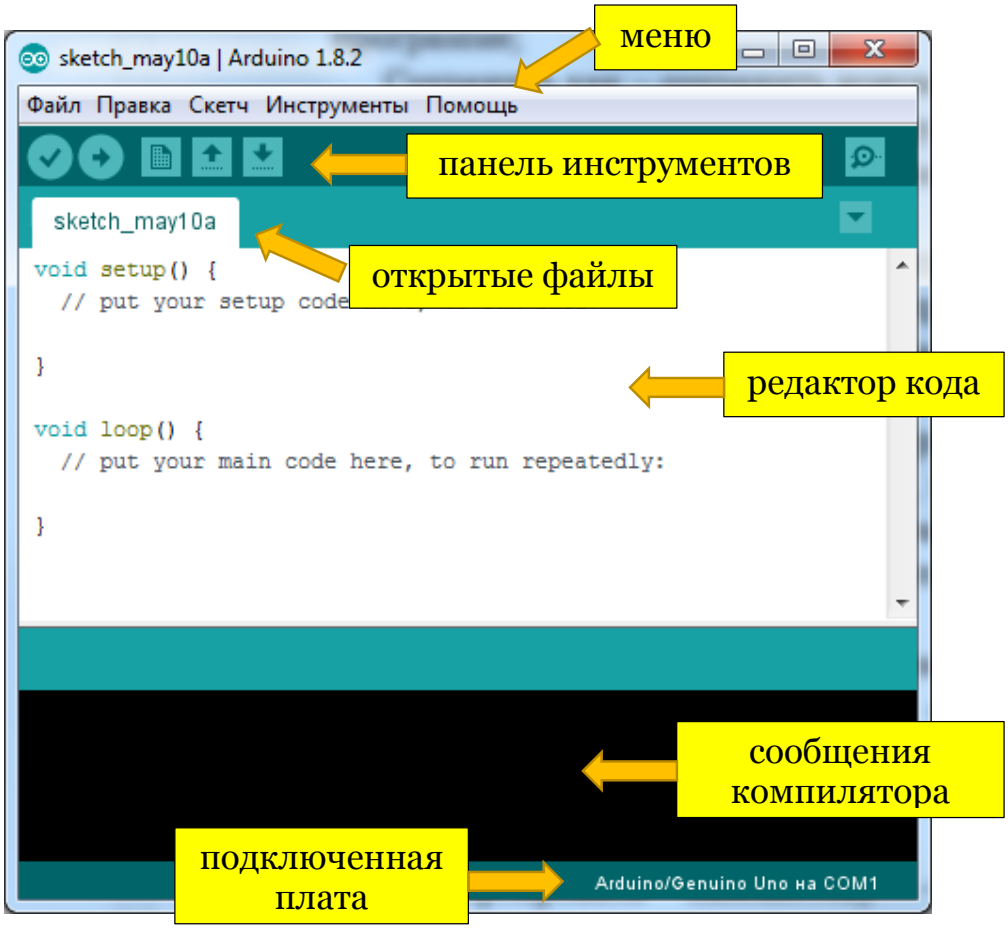


Рис. 3.3: Области окна среды разработки Arduino

Строка меню редактора (Рис. 3.3) включает в себя следующие главные элементы: файл, правка, скетч, сервис и справка. Рассмотрим подробнее каждый из них.

В пункте *Файл* можно найти команды, отвечающие за создание новой программы, загрузки с диска уже существующей, сохранения ее изменений, а также команды для загрузки программы на микроконтроллер:

– «Создать» – создать новую программу (в данной среде программы называются *скетчами*);

– «Открыть» – открыть существующую программу;
– «Папка со скетчами» – открыть программу из заданной папки;
– «Примеры» – открыть пример программы;
– «Закрыть» – закрыть текущее окно;
– «Сохранить» – сохранить изменения;
– «Сохранить как» – сохранить программу в новом файле;
– «Загрузить» – загрузить программу в Arduino;
– «Загрузить с помощью программатора» – загрузить программу посредством программатора;
– «Настройка печати» – настройка принтера;
– «Печать» – вывод на печать кода программы;
– «Настройки» – настройки редактора;
– «Выход» – выход из Arduino IDE.

Пункт меню *Правка* содержит команды, связанные с редактирование текста программы, включающие в себя копирование, вставку, настройку отступов и поиск.

В разделе *Скетч* размещаются команды для управления компиляцией программы:
– «Проверить/Компилировать» – компилировать программу;
– «Показать папку скетчей» – открыть системную папку с программами;
– «Добавить файл» – добавить к проекту файл с данными или программой;
– «Импортировать библиотеку» – подключить к программе библиотеку из списка установленных.

Пункт меню *Сервис* включает в себя вспомогательные функции для работы с самим микроконтроллером:
– «Автоформатирование» – автоматическая расстановка отступов, переносов строк и т.п.;
– «Архивировать скетч» – архивация папки с программой, и сохранение архива в указанное место;
– «Монитор порта» – открыть окно для обмена данными с микроконтроллером;
– «Плата» – выбор текущей платы (в нашем случае Arduino Uno);
– «Последовательный порт» – выбор порта, к которому подключено устройство;

– «Программатор» – выбор программатора (нами в учебном пособии не используется);
– «Записать загрузчик» – запись программы загрузчика в микроконтроллер (также нами не используется).

Наконец, меню *Справка* содержит подробное описание всех функций самого редактора Arduino IDE, а также всевозможные команды и приемы работы с платформой.

На панель инструментов вынесены в виде кнопок наиболее часто используемые функции среды. Их назначение описано в таблице 3.1.

Таблица 3.1: Кнопки на панели инструментов

Кнопка	Описание
	Проверить/Компилировать программу
	Загрузить программу в Arduino
	Создать новую программу
	Открыть существующую программу
	Сохранить программу
	Монитор последовательного порта

Непосредственно текст программы создается и редактируется в окне редактора кода. Это окно представляет собой типичный текстовый редактор с подсветкой синтаксиса программы.

В нижней части окна Arduino IDE имеется область, служащая для вывода уведомлений и сообщений об ошибках, возникающих в процессе компиляции программы или во время загрузки программы в микроконтроллер.

3.3 Онлайн-эмулятор Arduino

Для того, чтобы выполнять упражнения данного учебного пособия, не обязательно покупать плату (хотя это и предпочтительней, если у вас есть такая возможность). Мы будем использовать веб-

приложение Autodesk CIRCUITS, которое позволяет моделировать Arduino и различные электронные компоненты прямо в веб-браузере без необходимости загрузки и установки чего-либо себе на компьютер. Данная система абсолютно бесплатна и все, что вам нужно для работы с ней – выход в Интернет.

Заходим на сайт http://circuits.io и нажимаем в верхнем правом углу кнопку «Sign up for free». Можно зарегистрироваться непосредственно в системе или использовать возможность входа с помощью уже имеющихся учетных записей от Google, Facebook и других сервисов.

Чтобы создать новую схему, нажимаем кнопку «New Electronics Lab». Вы попадете в редактор электронных моделей, в котором пока есть одна пустая макетная плата.

Эта плата состоит из четырех областей (см. Рис. 3.4). Сверху и снизу идут по два ряда отверстий, предназначенных для создания шин питания. Все отверстия одного ряда связаны друг с другом. Предполагается, что через одно из отверстий к этим шинам подводят напряжение питания (положительный или отрицательный контакт), а затем, через другие отверстия к питанию можно будет подключать компоненты нашей системы.

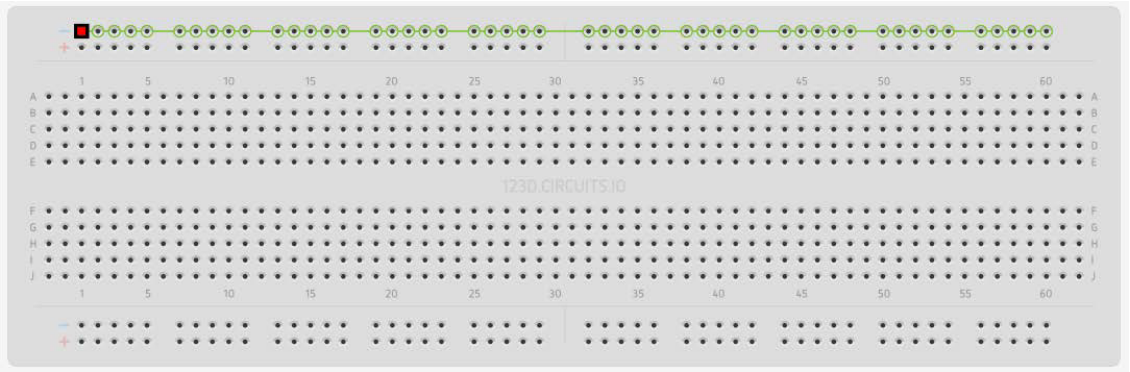


Рис. 3.4: Пример платы

Отверстия в каждой из двух центральных областей, напротив, связаны вертикально: они соединены в рядах A-B-C-D-E и F-G-H-I-J. Эти области используются для установки и подключения электронных устройств.

Чтобы создать схему, нужно в первую очередь добавить компоненты. Для этого нажимаем кнопку «+ Components» в верхнем правом углу экрана (Рис. 3.5).

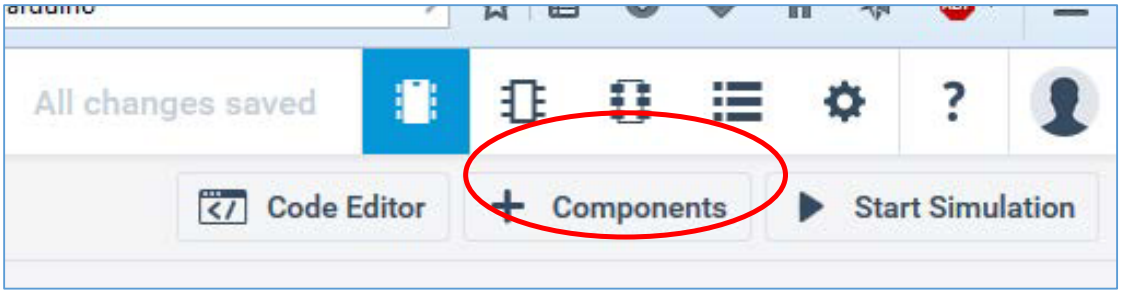


Рис. 3.5: Кнопка «+ Components»

Снизу появится панель, в которой можно выбирать различные компоненты, поддерживаемые эмулятором.

Давайте выберем светодиод и батарею и разместим их на нашей макетной плате. Если выводы (контакты) компонентов совпадут с отверстиями платы, то эмулятор будет считать, что они туда вставлены и есть контакт.

Если выделить мышью компонент, то справа сверху появится окно с его свойствами. С его помощью можно, например, задать цвет светодиода или поменять сопротивление резистора (Рис. 3.6).

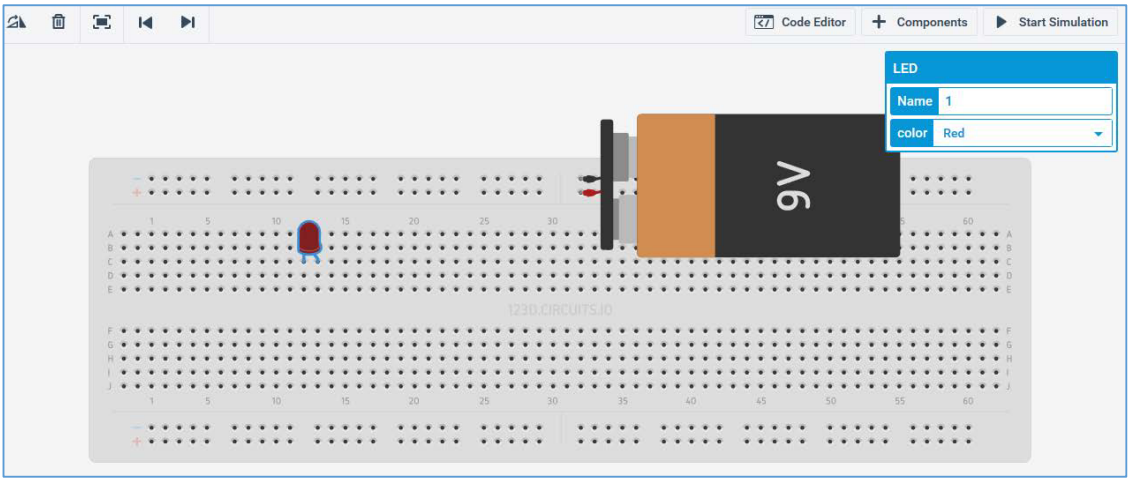
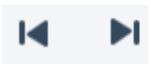


Рис. 3.6: Пример окна со свойствами

Расположенные в левом верхнем углу кнопки позволяют управлять редактором модели. Их описание приведено в таблице 3.2.

Чтобы соединить компоненты проводами, достаточно щелкнуть мышью на контактах, которые надо соединить. Появится изображение провода. Если щелкнуть по выделенному проводу, то на нем появится точка, с помощью которой провод можно изогнуть.

Таблица 3.2: Кнопки на панели инструментов circuits.io

Кнопка	Описание
	Поворачивает компонент
	Удаляет компонент (можно также нажать на клавишу <Delete> на клавиатуре)
	Уменьшает масштаб, так чтобы все компоненты поместились на экране (увеличить масштаб нельзя)
	Позволяют отменить предыдущее действие и вернуть его после отмены

Давайте подключим светодиод проводами к шине питания. Чтобы посмотреть, как работает схема, нажмите на кнопку «▶ Start Simulation» в правом верхнем углу экрана (Рис. 3.7).

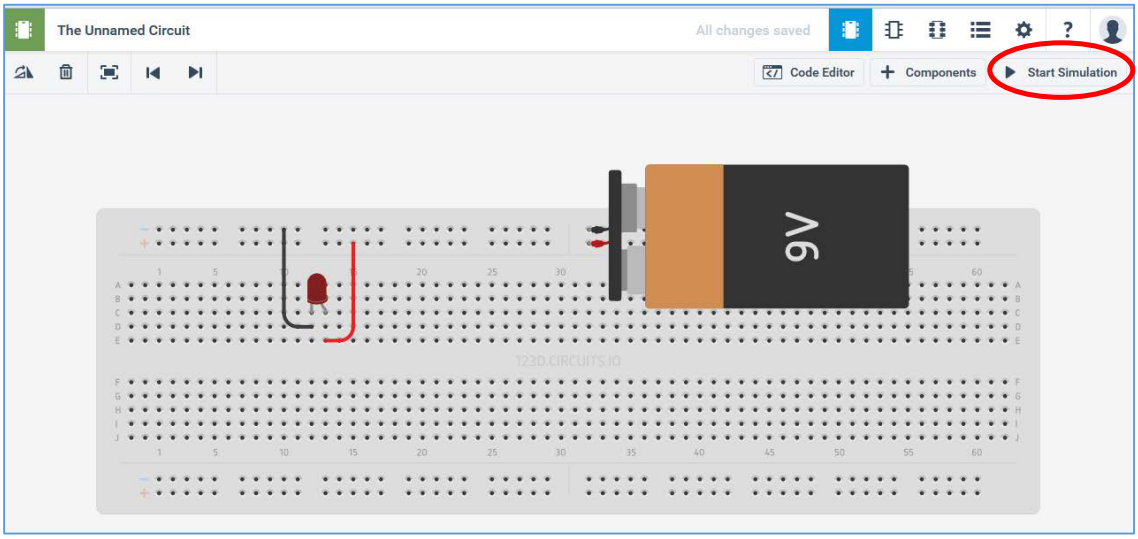


Рис. 3.7: Пример подключения светодиода

Что произошло? Симулятор показывает, что наш светодиод перегорел. Светодиоды нельзя напрямую подключать к источнику питания, особенно напряжением 9 вольт.

На этом примере хорошо видно одно из полезных свойств использования моделей – сжечь виртуальный светодиод можно совершенно бесплатно, и вам не потребуется бежать в магазин за новым. Моделирование позволяет проверить правильность схемы до

того, как мы потратим силы и время на ее изготовление. Можно проверить, заработает что-то или нет, поэкспериментировать с разными компонентами и способами соединения, и уже после того, как мы убедимся, что все работает как надо, можно купить компоненты и собрать реальную схему.

К сожалению, модели работают недостаточно хорошо, или, может быть лучше сказать, что они работают слишком хорошо, и показывают, как наше устройство работало бы в идеальном мире. В реальном мире всегда есть какие-то шумы, помехи, незначительные отклонения от идеального поведения, которые в некоторых случаях могут повлиять на работу устройства, которое до этого безупречно работало в эмуляторе.

Давайте исправим нашу схему, чтобы светодиод не перегорал. Для этого надо добавить в нее резистор на 430 Ом, последовательно соединенный со светодиодом. Для изменения проводов, нужно сначала выделить провод, а затем перетащить одну из его ключевых точек. Удалить промежуточную точку можно выделив ее и нажав <Delete> на клавиатуре.

Запустим схему. Теперь она работает нормально, светодиод загорелся (Рис. 3.8).

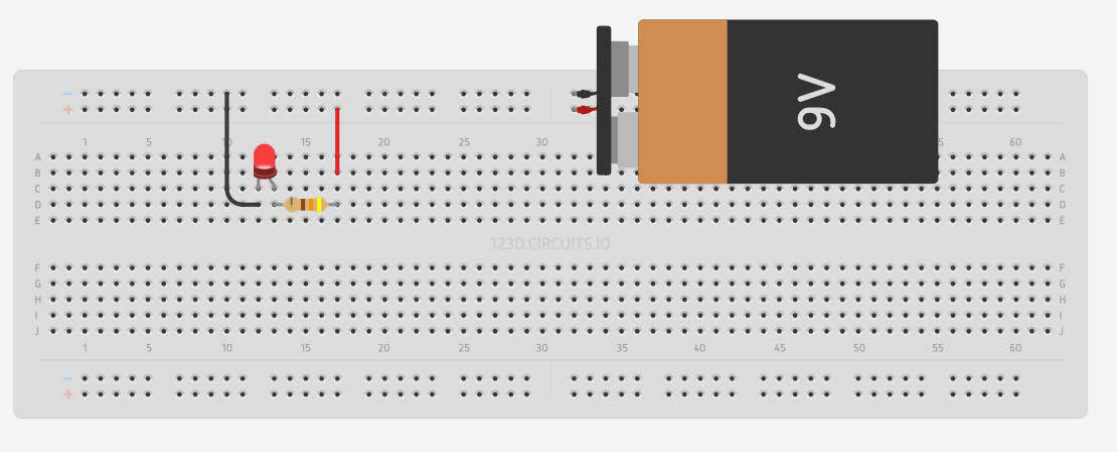


Рис. 3.8: Работа светодиода

В верхнем правом углу экрана есть кнопки, которые позволяют переключаться между разными режимами редактирования модели. Описание кнопок дано ниже в Таблице 3.3.

Таблица 3.3: Кнопки переключения между режимами редактирования

Кнопка	Описание
	Модель макетной платы (сейчас мы работаем в этом режиме)
	Электрическая принципиальная схема
	Редактор печатной платы (можно сделать свою печатную плату)
	Список компонентов (можно распечатать перед походом в магазин)
	Свойства проекта (можно задать название, описание и т.д.)
	Справка по системе
	Ваша личная страница в системе

Давайте заглянем на страницу с принципиальной схемой. На ней мы видим схему, соответствующую соединениям, которые мы сделали на макетной плате (Рис. 3.10). Компоненты можно вращать и перемещать мышью. Если их аккуратно разложить, то получится вполне читаемая схема.

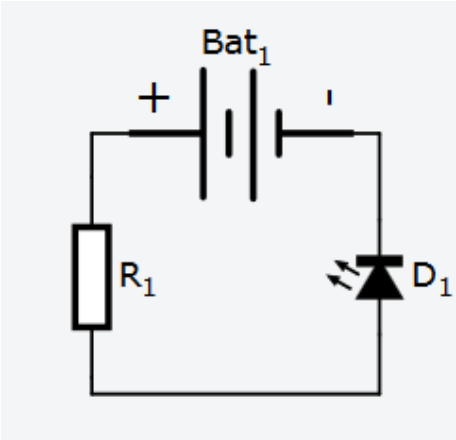


Рис. 3.10: Пример схемы

Редактировать схему (добавлять и удалять компоненты и связи между ними) в этом режиме нельзя. К сожалению, нельзя и перераспределить руками соединения, они всегда прокладываются автоматически, поэтому сложные схемы могут получаться достаточно неряшливыми.

Код для плат Arduino можно писать прямо в веб-интерфейсе среды Autodesk CIRCUITS. Для этого нужно нажать кнопку «Code Editor». Эта функция будет доступна только после добавления платы Arduino или других программируемых компонентов в модель.

Написав программу ее можно запустить, нажав кнопку «↑ Upload & Run» в верхней части редактора кода. Там же имеются и другие кнопки:

- Libraries – показывает поддерживаемые эмулятором библиотеки Arduino и позволяет их подключить;
- Download Code – скачать разработанный код на свой компьютер;
- Debugger – отладчик кода;
- Serial Monitor – показывает сообщения от Arduino.

Задание 3.1

Соберите схему, рассмотренную выше. Проверьте ее работу.

Задание 3.2

Добавьте в схему мультиметр. С помощью кнопки переключите его в режим амперметра и подключите последовательно со светодиодом. Запустите эмуляцию и проверьте ток через светодиод. Рабочий ток обычных светодиодов не превышает 20 mA.

3.4 Программирование на языке Си

Программирование в среде Arduino осуществляется на языке Си с минимальным набором элементов C++. В дальнейшем будем, для краткости, называть этот язык программирования языком Си.

Рассмотрим основные особенности синтаксиса этого языка, знание которых необходимо для написания программ.

Каждое выражение заканчивается символом точки с запятой. Например:

```
a = b + c;
```

Тело функций и составных операторов (`if`, `else`, `for`, `while`) выделяются фигурными скобками (аналогично `BeginEnd` в языке Pascal). Например:

```
if(a > 0) {
     b = a+1;
}
```

Строки выделяются двойными кавычками:

```
lcd.print("some text");
```

а символы выделяются одинарными кавычками:

```
symbol = 'a';
```

Подключение библиотек осуществляется с помощью конструкции:

```
#include <math.h>
```

Комментарии в программе начинаются двумя прямыми слешами:

```
// это комментарий
```

Объявление переменной в языке Си осуществляется с помощью конструкции вида:

```
тип_переменной имя_переменной;
```

Пример:

```
int x, y; // объявлены переменные x и y, имеющие целый тип
```

В программах для Arduino можно использовать следующие типы для числовых данных:

- `byte` – целое число от `0` до `255`;
- `int` – целое число от `-32768` до `+32767`;
- `word` – целое число от `0` до `65535`;
- `long` – целое число от `-2147483648` до `2147483647`;

– `float` и `double` – числа с плавающей точкой, соответствуют типу `float` для обычных компьютеров, тип `double` в настоящее время эквивалентен `float`.

Массивы в языке Си объявляются конструкцией вида:

```
тип_элемента имя_массива[размер];
```

Пример:

```
int values[10]; // массив из десяти целых чисел
```

В языке Си предусмотрен тип для хранения символов и строк, это тип `char`. Строки представляют собой массивы типа `char`, в которых последний символ имеет код `0`, что обозначает конец строки. Пример:

```
char my_str[10]; // строка не более девяти символов длиной
```

Тип `boolean` используется для представления логических значений `true` (истина) и `false` (ложь). Кроме того, любое целое значение может использоваться как логическое, при этом `0` означает ложь, а любое ненулевое значение – истину.

Последний тип данных, который мы рассмотрим – тип `void`. Это «пустой» тип, который используется при объявлении функций, не принимающих аргументов или не возвращающих результат (аналогично процедурам в языке Pascal).

Указание слова `const` перед объявлением переменной говорит о том, что ее значение не может быть изменено.

В языке Си используется стандартный набор математических операторов, включающий `+`, `-`, `*` и `/`. Для вычисления остатка от деления используется %.

Кроме того, есть их специальные комбинации с оператором присваивания, позволяющие сократить запись:

```
x += 4;  // Означает x = x + 4
```

Такие формы есть для всех операторов.

Для увеличения и уменьшения числа на единицу есть инкременты и декременты:

```
x++;     // Означает x = x + 1
y--;     // Означает y = y – 1
```

Набор операторов сравнения тоже стандартен и включает операторы <, >, <=, >=. Равенство записывается с помощью двух знаков равно (==), а неравенство записывается как !=. Логические выражения можно вычислять с помощью операторов «и» – &&, «или» – ||. Логическое отрицание записывается с помощью восклицательного знака !.

Условный оператор имеет следующий синтаксис:

```
if(условие)
     код, выполняемый, если условие истинно
```

Если в случае истинности условия необходимо выполнить несколько операторов, то они объединяются в один блок с помощью фигурных скобок:

```
if(условие){
     код, выполняемый, если условие истинно
}
```

Полная форма условного оператора включает блок else, код в котором выполняется, если условие ложно:

```
if(условие) {
     код, выполняемый, если условие истинно
}
else {
     код, выполняемый, если условие ложно
}
```

Оператор switch позволяет осуществить выбор одной из веток кода в зависимости от значения числовой переменной, например:

```
switch(mode) {
     default:
     case 0:
          код для режима 0
          break;
     case 1:
          код для режима 1
          break;
     case 2:
          код для режима 2
```

```
        break;
}
```

При этом значение переменной `mode` определяет, в какую точку перейдет исполнитель программы внутри тела оператора. Дальше будут выполняться все действия, если не встретится `break`, который прервет дальнейшее исполнение кода внутри `switch`.

Если значение выражения не соответствует ни одной из меток, то выполняется код, начиная с метки `default`. В данном примере из-за отсутствия `break` после `default` управление попадет в «действия для режима `0`».

Цикл `while` позволяет выполнять код до тех пор, пока условие верно. Он имеет синтаксис:

```
while( условие )
    код, выполняемый в цикле
```

Более сложным является цикл `for`:

```
for(инициализация; условие; шаг)
    код, выполняемый в цикле
```

В этом цикле выражение, указанное в блоке «инициализация», выполняется один раз перед началом цикла. Обычно в нем задают начальные значения для переменных цикла. Условие проверяется каждый раз в самом начале очередной итерации цикла и если оно станет ложно, то цикл останавливается. «Шаг» выполняется после выполнения тела цикла и обычно увеличивает (или уменьшает) переменные цикла. В блоках «инициализация» и «шаг» можно изменить несколько переменных, если написать соответствующие выражения через запятую.

Пример цикла, вычисляющего сумму арифметической прогрессии:

```
int i, sum;

for(i = 0, sum = 0; i <= 10; i++)
   sum += i;
```

Код программы на языке Си должен быть обязательно разбит на функции. Для объявления функции используется следующий синтаксис:

```
тип_функции имя_функции( параметры ) {
    код, выполняемый в рамках функции
    return результат_функции;
}
```

Тип функции сообщает тип значения, которое будет возвращать функция. Например, стандартная функция `cos`, вычисляющая косинус, возвращает значение типа `float`.

Имя функции может быть любой строкой, начинающейся с буквы, и содержащей только буквы, цифры и символы подчеркивания.

Параметры – это перечень переменных, которые принимают значения, передаваемые в функцию. Переменные указываются через запятую, сначала пишут тип параметра, а затем – его название.

Чтобы вернуть значение из функции используется оператор `return`. Он немедленно прерывает выполнение функции, а ее результатом становится значение указанного после `return` выражения.

Если функция ничего не возвращает, то `тип_функции` указывается как `void`. Если функции не передается никаких аргументов, то можно ничего не писать внутри круглых скобок или можно написать там слово `void`.

Ниже приведен пример функции, суммирующей два числа типа `int`:

```
int sum(int a, int b) {
    int result;
    result = a + b;
    return result;
}
```

Для вызова функции необходимо написать ее имя и в круглых скобках указать через запятую значения аргументов, например:

```
r = sum(3, 10);
```

В программах на языке Си можно объявить переменные за пределами любых функций. Такие переменные называются глобальными. Их можно использовать в любых функциях, расположенных после объявления переменной. Например, в этой программе переменная `led` используется в функциях `setup()` и `loop()`.

```
const int led = 2;

void setup() {
     pinMode(led, OUTPUT);
}

void loop() {
     digitalWrite(led, HIGH);
     delay(1000);
     digitalWrite(led, LOW);
     delay(1000);
}
```

Вопросы для самоконтроля

1. Какие из следующих строк не соответствуют синтаксису языка Си:
   a) `int x;`
   b) `int x, char u`
   c) `int x, y;`
   d) `float int x;`
2. Какое значение примет переменная `b` после выполнения этого кода:

```
int a = 2;
char b = ‘ ‘;
switch(a) {
     default:
     case 0:
          b = ‘a’;
          break;
     case 1:
          b = ‘b’;
          break;
     case 2:
          b = ‘c’;
```

```
        break;
}
```

3. Чему будет равно значение переменной `k` в функции `setup()` в результате выполнения следующего блока кода:

```
int func (int b){
     int a = 0;
     if (b % 2 == 0 || a > 0)
          return a;
     else
          return (a+1);
}
void setup(){
     int k = func(5);
}
```

4. Сколько раз выполнится тело следующего цикла:

```
int i = 4;
while(i >= 0) {
     i--;
}
```

3.5 Структура скетча

В предыдущем разделе мы говорили о языке программирования, который используется в Arduino. Этот язык очень похож на язык Си, но есть и некоторые отличия.

Во-первых, когда пишут программу для Arduino, она обязательно должна быть написана в одном файле, который называют скетч. Вы можете часто встретить такое сочетание слов «скетч для Arduino», это значит программа для Arduino.

Есть отличие в структуре скетча от структуры обычной программы на Си, и вызвано оно различием между работой программы на обычном компьютере и во встроенных системах. Когда выполняют программу на обычном компьютере, то она запускается, выполняет какие-то действия, а потом завершается. Если программу пишут на языке Си, то за ее выполнение отвечает функция `main()`.

При запуске программы запускается именно эта функция. Из нее программист уже может вызвать остальные функции программы в том порядке, в котором они должны выполняться. Когда функция `main()` возвращает управление с помощью `return`, программа завершается.

Но для программы, управляющей микроконтроллером, такое поведение не имеет смысла. Пока на устройство подано питание, микроконтроллер должен управлять устройством, а значит, он должен выполнять свою программу. Эта программа никогда не завершается сама, она перестает работать, только если выключить микроконтроллер.

Поэтому структура программы для Arduino отличается от структуры обычной программы на языке Си. В программе для Arduino не должно быть функции `main()`, вместо нее должны быть две функции: `setup()` и `loop()`.

Первая из этих функций, функция `setup()`, вызывается один раз в самом начале выполнения программы сразу после того, как микроконтроллер включили. Она должна подготовить микроконтроллер и все устройства к работе.

Обычно в этой функции настраивают микроконтроллер, например, задают, какие его выводы будут использоваться как входы, а какие – как выходы и так далее. В этой функции можно вывести на экран приветствие для пользователя (если у устройства есть экран). Или каким-то другим образом приготовить устройство к работе. Может быть, нужно что-то включить, или наоборот, убедиться, что ничего лишнего пока не включено.

Вторая функция, функция `loop()`, выполняется все время, пока работает наше устройство. Если она завершится, то среда Arduino запустит ее заново. В этой функции как раз и нужно реализовать алгоритм работы устройства. Обычно в ней опрашивают подключенные к микроконтроллеру сенсоры, или сами входы микроконтроллера, анализируют полученную информацию и выдают команды исполнительным устройствам.

Можно считать, что где-то в библиотеке Arduino уже есть функция `main()`, которая устроена примерно так:

```
void main() {
    setup();
```

```
    while(true)
        loop();
}
```

Такая функция там действительно есть, скорее всего она устроена более сложно, она может делать еще какие-то действия по управлению микроконтроллером, но общая структура именно такова. А нам нужно написать две функции `setup()` и `loop()`.

Как и в обычных программах, мы можем создавать и любые другие свои функции и вызывать их из `setup()` и `loop()`. Если вы делаете что-то простое, то можно написать весь код просто внутри `setup()` и `loop()`. Так стоит сделать, если весь их код помещается в несколько строчек. Если делать что-то сложное, то стоит разбить код на отдельные функции.

В программе Arduino можно вызывать стандартные функции, которые есть в библиотеке. Для этого не нужно добавлять директив `#include`. В обычном языке Си это нужно делать даже для стандартных функций. Одна из таковых, которая будет часто использоваться, это функция `delay( время )`, делающая задержку на указанное время. Время задается в миллисекундах, поэтому, чтобы подождать 1 секунду, надо написать:

```
delay(1000);
```

Есть еще несколько стандартных функций, которые мы будем использовать, и которые можно сразу писать в программе. Эти функции управляют базовыми возможностями микроконтроллера, и мы их рассмотрим, когда они нам понадобятся.

Большим преимуществом среды Arduino является ее большая популярность. Многие производители выпускают платы расширения, которые можно подключать к Arduino и использовать вместе с ней. Для того чтобы такие платы было удобно использовать, для них пишут библиотеки, содержащие удобные функции для работы с этими устройствами. Пишут такие библиотеки и просто для отдельных компонентов, которые можно подключить к плате Arduino. Мы будем использовать, например, библиотеку для управления жидкокристаллическим экраном.

Чтобы использовать библиотеку, нужно в начале программы поместить директиву `#include`:

```
#include < имя_библиотеки.h >
```

Например, библиотека для работы с дисплеем называется `LiquidCrystall`, и чтобы ее использовать, надо сначала написать

```
#include <LiquidCrystal.h>
```

В языке Arduino для работы с библиотеками используют элементы языка C++. Фактически, сами библиотеки пишут на C++, но в скетче можно использовать только очень простые возможности этого языка. Язык C++ позволяет создавать классы. Классы – это пользовательские типы данных, описывающие набор полей и операции, которые с ними можно делать. С помощью этого механизма в C++ можно сделать, например, класс «дисплей» и переменную такого типа. Такие переменные называют объектами. Потом можно работать с этими «сложными» переменными. Чтобы сделать такую переменную, ее надо объявить, как объявляют обычные переменные. Например, для работы с дисплеем надо сделать переменную типа `LiquidCrystall`:

```
LiquidCrystal lcd(4, 5, 6, 7, 8, 9);
```

Тут `LiquidCrystall` – это название типа, его пишут в документации на библиотеку. `lcd` – это название нашей переменной, можно использовать любое имя, как удобно. Цифры – это параметры, которые передаются объекту при создании. О том, какие параметры надо указать, тоже пишут в документации. Для дисплея это, на самом деле, номера выводов микроконтроллера, к которым он подключен. Позже мы рассмотрим работу с дисплеями более подробно.

После того, как переменная создана, мы можем с ней работать. Для этого используют специальные функции, которые доступны только для этой переменной. Такие функции еще называют «методами класса». Вызывают их следующим способом:

```
имя_переменной.имя_функции( параметры );
```

Например, текст на дисплей выводит функция `print`, и ее можно вызвать следующим образом:

```
lcd.print("Hello!");
```

Так как мы указываем сначала имя переменной, то функция `print` вызывается именно для этой переменной. Можно подключить несколько дисплеев и сделать для каждого из них свою переменную.

Вопросы для самоконтроля

1. Какие функции обязательно должны присутствовать в скетче для Arduino?
   a. `setup()`
   b. `main()`
   c. `startup()`
   d. `loop()`
   e. `init()`
2. Выберите верные утверждения:
   a. «Класс» – это тип данных, а «объект» – это значение такого типа.
   b. Для использования любой стандартной функции в среде Arduino, надо в начале программы написать директиву `#include` и указать имя библиотеки.
   c. Библиотеки пишут для упрощения работы с различными устройствами.
   d. Чтобы работать с двумя одинаковыми устройствами, для которых есть библиотека, надо будет два раза подключить эту библиотеку с помощью `#include`.
   e. Чтобы работать с двумя одинаковыми устройствами, для которых есть библиотека, надо будет один раз подключить библиотеку с помощью `#include` и создать два объекта соответствующего класса.

Тема 4

Цифровой ввод-вывод

4.1 Цифровые выходы. Работа со светодиодом

Одно из простейших действий, которое может выполнить микроконтроллер, это переключение состояния контактов цифровых входов-выходов. Любой из цифровых выходов микроконтроллера можно программным способом переключить между высоким уровнем (+5 V) и низким (0 V). С помощью такого переключения происходит управление простыми устройствами.

Переключение осуществляется функцией `digitalWrite`, вызов которой имеет вид:

```
digitalWrite( номер_контакта, уровень_сигнала );
```

Параметр `уровень_сигнала` может принимать два значения: `HIGH` (высокий, +5 V) или `LOW` (низкий). Параметр `номер_контакта` – это число, соответствующее номеру контакта на плате Arduino.

Обратите внимание, что одни и те же выводы микроконтроллера могут выступать как в роли цифровых выходов, так и в роли цифровых входов. Поэтому перед их использованием необходимо указать, в какой роли будет использоваться контакт: входа или выхода. Это делается с помощью функции `pinMode`:

```
pinMode( номер_контакта, режим_контакта );
```

Ее аргумент `режим_контакта` может принимать значения: **`OUTPUT`** (выход) и **`INPUT`** (вход). Таким образом, чтобы установить на выходе №2 высокий уровень сигнала, достаточно выполнить следующую программу:

```
const int pin = 2;

void setup() {
     pinMode(pin, OUTPUT);
     digitalWrite(pin, HIGH);
}

void loop() {
}
```

Одно из наиболее простых для подключения исполнительных устройств, которым можно управлять с помощью цифрового выхода, это *светодиод*. Это устройство представляет собой полупроводниковый прибор, способный излучать свет при пропускании через него электрического тока в прямом направлении (от анода к катоду). На Рис. 4.1 показан внешний вид светодиода и его обозначение на электрической схеме.

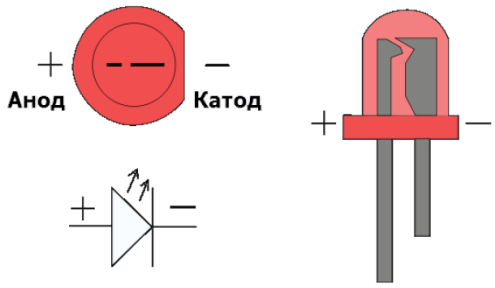


Рис. 4.1: Внешний вид светодиода и его обозначение на электрической схеме

Для того чтобы правильно включить светодиод в электрическую цепь, необходимо отличать катод от анода. Сделать это можно по трем признакам:

1) анод светодиода имеет более длинный проводник;
2) со стороны катода корпус светодиода немного срезан;
3) внутри светодиода катод шире, чем анод.

В современной микроэлектронике применяются миниатюрные светодиоды для поверхностного монтажа. Такие индикаторы, например, имеются на Arduino Uno для информирования пользователя о состоянии системы.

Важно отметить, что рабочее напряжение питания светодиода варьируется от 1.85 до 2.5 вольт при рекомендуемой силе тока не более 20мА. Чтобы сила тока не превысила это значение, при подключении светодиода к выходу микроконтроллера, который выдает напряжение 5 V, в цепь следует добавить последовательный ограничивающий резистор сопротивлением от 200 до 500 Ом. В противном случае ток через светодиод будет слишком большим, что может повредить как диод, так и вывод микроконтроллера. На Рис. 4.2 представлена электрическая схема подключения светодиода к Arduino Uno, а также модель собранного устройства.

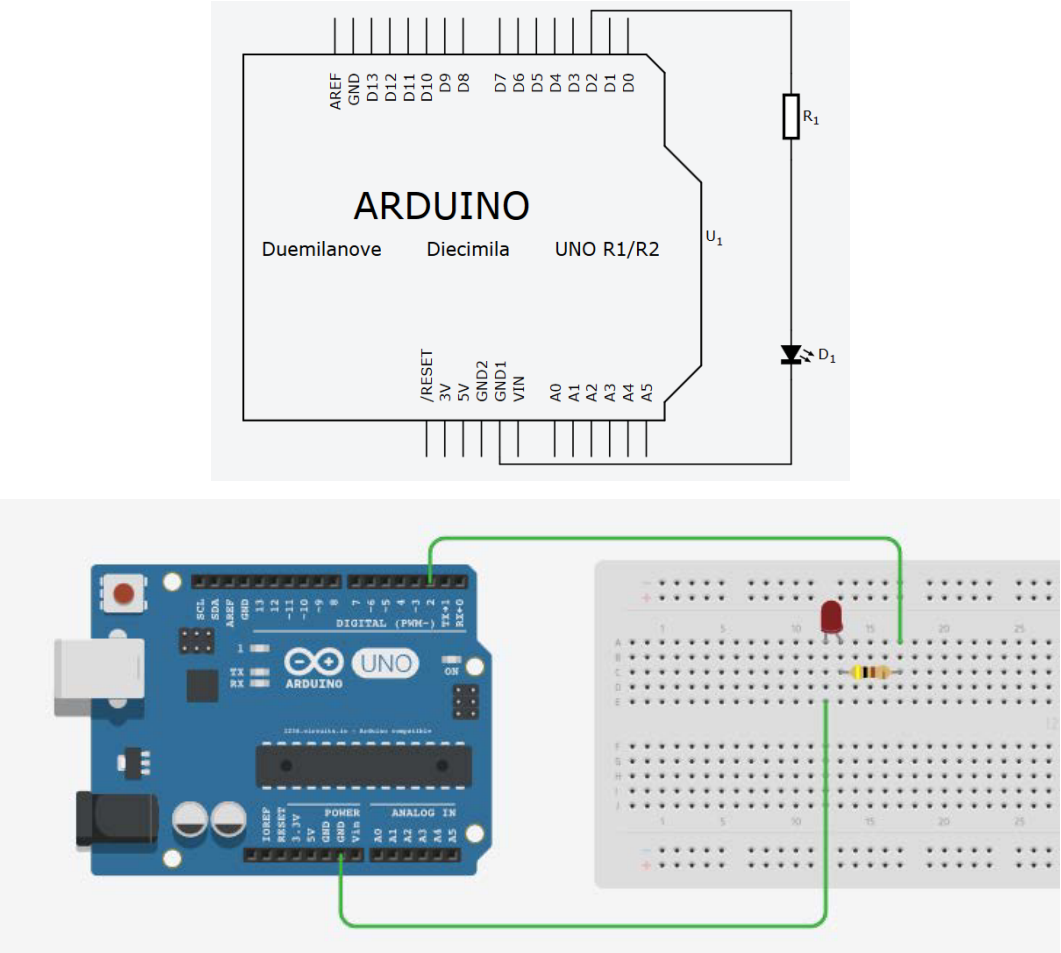


Рис. 4.2: Электрическая схема подключения светодиода и модель собранного устройства

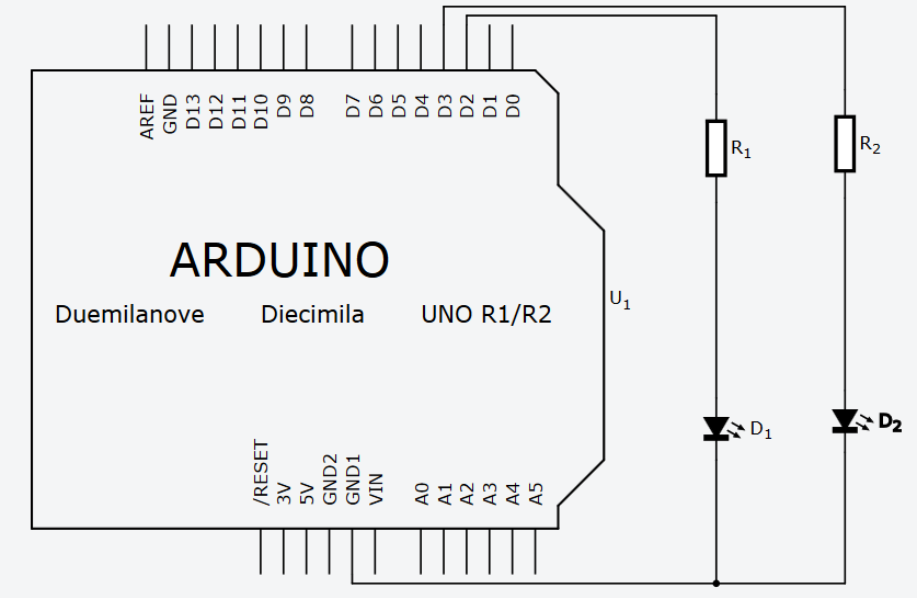
Попробуем «помигать» светодиодом. Для этого мы будем последовательно зажигать его, передавая на ногу №2 сигнал `HIGH`, а затем гасить с помощью сигнала `LOW`. Между включением и выключением светодиода обязательно нужно поставить задержку в несколько сотен миллисекунд, иначе мы не заметим мигания. Вспомним, что контроллер Arduino работает на частоте 16MHz, поэтому он может включать и выключать светодиод тысячи раз в секунду. Получаем следующую программу:

```
int led = 2;

void setup() {
     pinMode(led, OUTPUT);
}

void loop() {
     digitalWrite(led, HIGH);
     delay(1000);
     digitalWrite(led, LOW);
     delay(1000);
}
```

Аналогичным образом можно подключить еще один светодиод (Рис. 4.3). Для подключения сразу двух светодиодов к земле используем шину питания на макетной плате.



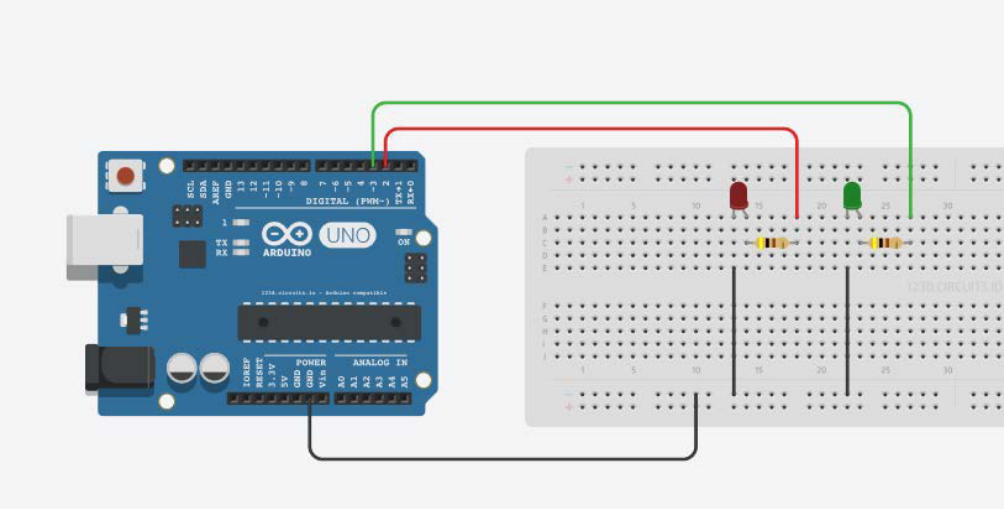


Рис. 4.3: Схема подключения двух светодиодов

Имеем следующий код:

```
int led_r = 2;
int led_g = 3;

void setup() {
     pinMode(led_r, OUTPUT);
     pinMode(led_g, OUTPUT);
}

void loop() {
     digitalWrite(led_r, HIGH);
     digitalWrite(led_g, HIGH);
     delay(1000);
     digitalWrite(led_r, LOW);
     digitalWrite(led_g, LOW);
     delay(1000);
}
```

Задание 4.1

Реализуйте на базе разработанной схемы программу «проблесковые маячки». Светодиоды должны попеременно мигать с интервалом 0.3 сек.

Задание 4.2

Подключите к Arduino три светодиода (красный, желтый и зеленый) и реализуйте программу, заставляющую светодиоды загораться в режиме светофора, повторяя следующий цикл:

1) зеленый светодиод горит 3 секунды;
2) зеленый светодиод мигает 3 секунды;
3) желтый светодиод зажигается на 1 секунду;
4) красный светодиод зажигается на 3 секунды.

4.2 Вывод информации через последовательный порт

Arduino может передавать текстовые сообщения на персональный компьютер. Для того чтобы этим воспользоваться, необходимо в функции `setup` инициализировать последовательный порт устройства:

```
Serial.begin(9600);
```

Значение 9600 означает скорость передачи данных в битах в секунду. 9600 – это одна из стандартных скоростей последовательного порта. Непосредственно для вывода короткого сообщения используется функция `print`:

```
Serial.print(“text”);
```

Пример программы:

```
void setup() {
  Serial.begin(9600);
  Serial.print("Hello!");
}
void loop() {
}
```

Чтобы увидеть эти сообщения в Autodesk CIRCUITS включите Serial Monitor на панели управления над редактором кода.

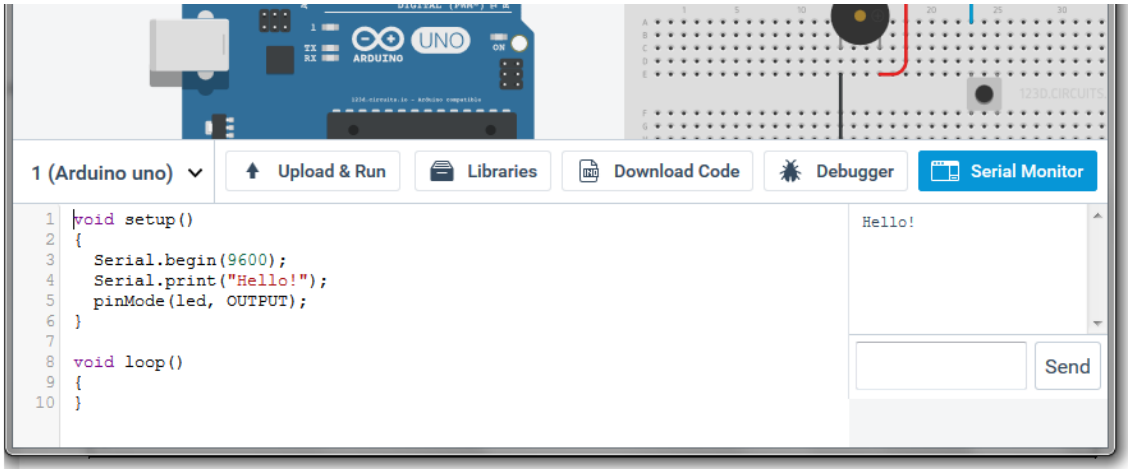


Рис. 4.4: Пример работы Serial Monitor

Если вы используете настоящую плату Arduino и редактор Arduino IDE, используйте имеющееся в нем окно «Монитор порта».

Задание 4.3

Напишите программу, которая будет каждую секунду передавать через последовательный порт увеличивающееся на единицу число.

4.3 Цифровые входы, подключение кнопок и выключателей

Выключатель – это прибор, который позволяет замыкать и размыкать электрическую цепь. Смена состояния выключателя может происходить разными способами, в зависимости от типа устройства.

Механические выключатели используются непосредственно человеком. К такому типу относятся различные тумблеры, кнопки, клавиши и рубильники. Электромагнитные и электронные, напротив, применяются в автоматических системах и управляются при помощи электрических сигналов. Самым известным электромагнитным выключателем является *реле* (relay). Примером электронного выключателя может служить *транзистор* (transistor).

В нашей работе мы будем использовать простой механический выключатель (pushbutton), который замыкает два своих контакта при нажатии. Для удобства монтажа их часто делают с 4-мя контактами, у которых выводы соединены попарно.

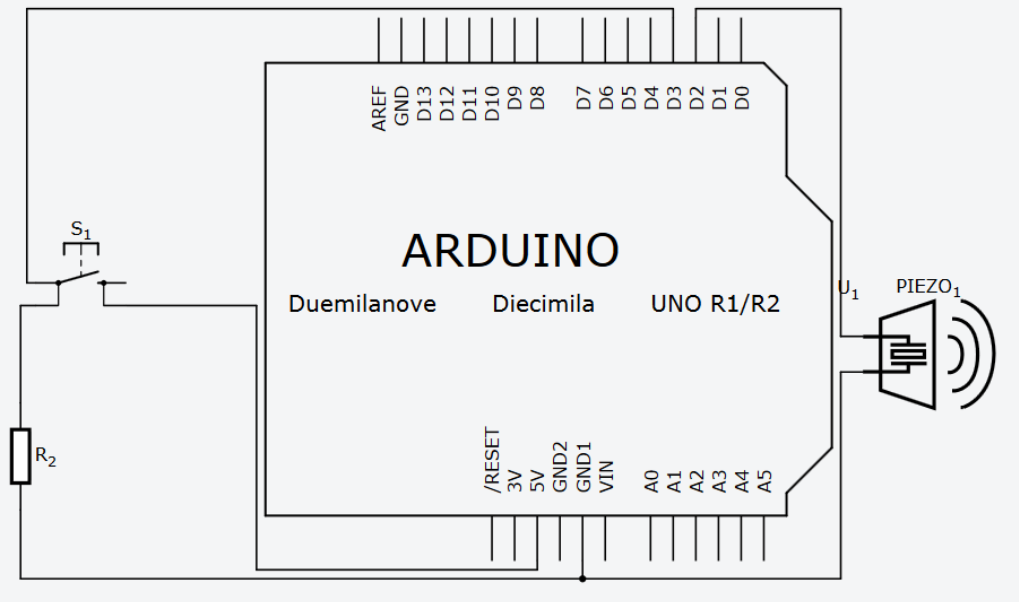
Для чтения цифрового сигнала с одного из контактов Arduino, необходимо воспользоваться функцией `digitalRead`:

```
результат = digitalRead( номер_контакта );
```

После вызова этой функции переменная `результат` будет хранить уровень цифрового сигнала, детектируемый на соответствующем контакте.

Чтобы читать состояние кнопки, ее нужно подключить таким образом, чтобы при ее нажатии на входе был высокий уровень напряжения (`5` V), а при отпускании – низкий (`0` V). Для этого подключим последовательно между шиной питания и землей кнопку и резистор, а вход микроконтроллера подключим посередине между ними (см. Рис. 4.5). Тогда, если кнопка не нажата, то выход микроконтроллера будет соединен через резистор с землей и будет иметь низкий уровень, а если кнопку нажать, то получится, что вход соединен с питанием и на нем будет высокий уровень.

При такой схеме подключения через резистор потечет ток, который зависит от его сопротивления. Чтобы схема не потребляла много лишнего электричества, возьмем резистор побольше, например, 10 kOm.



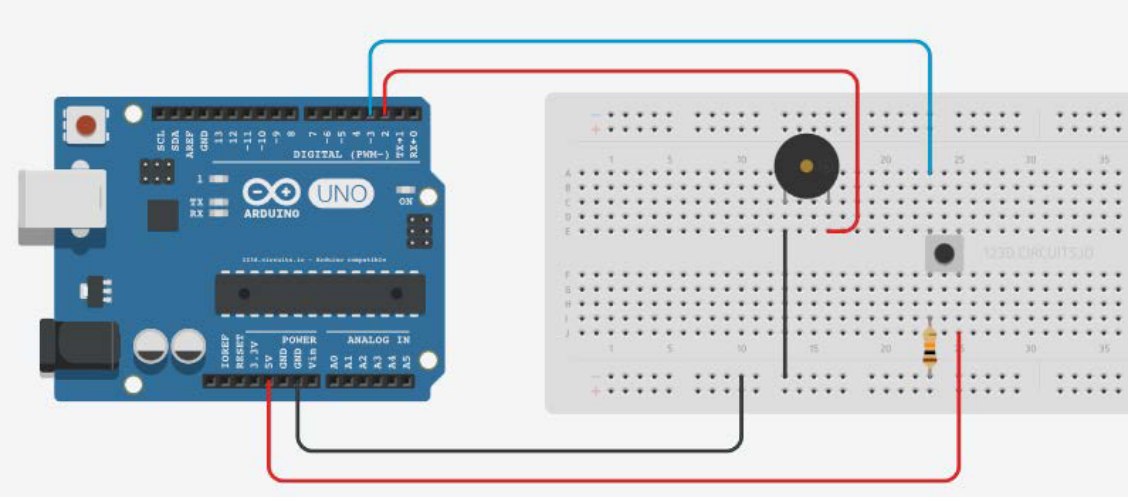


Рис. 4.5: Пример подключения резистора

Напишем программу для вывода звукового сигнала по нажатию кнопки. Для этого используем еще одну компоненту – *зуммер*, который издает звуковой сигнал, когда на него подается высокий уровень напряжения. После запуска программа начинает постоянно проверять состояние кнопки. Если кнопка нажата, то зуммер включается, а если отпущена – выключается:

```
const int btn = 3;
const int buzz = 4;
byte val = 0;

void setup() {
     // установка 3-го контакта в режим ввода
     pinMode(btn, INPUT);
     // установка 4-го контакта в режим вывода
     pinMode(buzz, OUTPUT);
}

void loop() {
     val = digitalRead(btn);

     // условие «нажата кнопка»
     if(val == HIGH)
          // перевод зуммера в активное состояние
          digitalWrite(buzz, HIGH);
```

```
    else
        // перевод зуммера в неактивное состояние
        digitalWrite(buzz, LOW);
}
```

Теперь модифицируем программу, чтобы нажатие на кнопку переключало звуковой сигнал, то есть нажимаем первый раз – зуммер включается и работает, нажимаем второй раз – выключается и так далее. Для этого введем дополнительную переменную `state`, в которой будем хранить текущее состояние зуммера.

Нажимаем кнопку один раз: в переменную состояния записывается `1`, нажимаем второй – `0`, третий – снова `1` и т.д. А зуммер будем активировать в зависимости от того, какое значение хранится в этой переменной:

```
const int btn = 3;
const int buzz = 4;
byte val = 0;
bool state = 0;

void setup() {
    // установка 3-го контакта в режим ввода
    pinMode(btn, INPUT);
    // установка 4-го контакта в режим вывода
    pinMode(buzz, OUTPUT);
}

void loop() {
    val = digitalRead(btn);

    // условие смены состояния
    if(val == HIGH) {
        state = !state;
        delay(200);
    }
    if(state == true) // условие активации зуммера
        // перевод зуммера в активное состояние
        digitalWrite(buzz, HIGH);
```

```
    else
        // перевод зуммера в неактивное состояние
        digitalWrite(buzz, LOW);
}
```

Обратите внимание на задержку в `0.2` секунды после фиксации нажатия на кнопку. Эта задержка нужна, чтобы исключить влияние так называемого *дребезга контактов*. Когда кнопка нажимается, ее контакты не сразу переходят в замкнутое состояние, они успевают несколько раз очень быстро замкнуться и разомкнуться, что может быть воспринято микроконтроллером как несколько разных нажатий. Задержка позволяет пропустить время, пока контакт не стабилизируется.

Задание 4.4

Сделайте схему, в которой к Arduino подключено две кнопки, светодиод и зуммер. Напишите программу так, чтобы после нажатия на одну из кнопок, светодиод начинал мигать, а зуммер издавать периодический сигнал (лампочка и зуммер должны работать синхронно: когда лампочка горит, зуммер издает сигнал, когда лампочка потухает, зуммер тоже выключается). Оба сигнала выключаются по нажатию второй кнопки.

Задание 4.5

Сделайте схему и напишите программу для генератора сигнала SOS. В схеме должны присутствовать одна кнопка, светодиод и зуммер. После нажатия на кнопку, зуммер начинает передавать сигнал SOS. Параллельно с зуммером светодиод дублирует сигнал световыми импульсами. Выключается генератор с помощью второго нажатия кнопки.

Напоминание: сигнал SOS представляет собой последовательность коротких и длинных сигналов, чередующихся следующим образом: три коротких, три длинных, три коротких, пауза, три коротких, три длинных, три коротких, пауза, ...

Задание 4.6

Сделайте игру на реакцию. В схеме должны присутствовать две кнопки, два светодиода и зуммер. После запуска программы зуммер начинает издавать короткие импульсы через неравные промежутки времени. Каждый игрок должен как можно быстрее нажать на кнопку сразу после сигнала зуммера. У игрока, нажавшего свою кнопку первым, на 2 секунды зажигается светодиод.

Задание 4.7

Сделайте модель кодового замка. В схеме присутствуют три кнопки, зеленый и красный светодиоды, а также зуммер. После запуска программы горит красный светодиод. Пользователю необходимо нажать три кнопки в правильной последовательности. Если кнопки нажаты правильно, загорается зеленый светодиод, в противном случае зуммер издает пять длинных сигналов.

Тема 5

Индикация

5.1 Семисегментный индикатор

Мы уже познакомились со светодиодом, который является одним из наиболее часто используемых индикаторов. Обычным светодиодом легко информировать пользователя о каких-то бинарных событиях, то есть событиях, о которых важно знать, наступили они или нет: включении или выключении какого-либо устройства, о превышении пороговых значений и тому подобном.

Но что, если требуется сообщать пользователю более сложную информацию, например, числовую? Другими словами, что если прибор должен сообщать измеряемые значения явно, в виде чисел? Для этих целей в электронике часто используется *сегментный (или семисегментный) светодиодный индикатор*. Этот прибор представляет собой набор обычных светодиодов, расположенных в виде восьмерки таким образом, что, зажигая некоторые из них, можно получить контуры различных цифр.

Конструктивно, светодиодные индикаторы подразделяются на приборы с общим катодом и с общим анодом. Общий катод, к примеру, означает, что все светодиоды внутри индикатора соединены вместе катодами, а их аноды разведены по отдельным контактам. Именно такой тип индикатора используется в нашем учебном пособии.

Внешний вид и обозначение на схеме сегментного индикатора представлены на Рис. 5.1. Все сегменты промаркированы буквами латинского алфавита, начиная с a и заканчивая g. Как правило, счет ведут с самого верхнего сегмента по часовой стрелке. Точка маркируется отдельно, словом dot (точка).

Кроме контактов, соединенных с сегментами, на схеме видны два контакта `com` и один `dot`. Слово `dot` используется для обозначения точки, а `com` – это общий катод (или анод). Он часто бывает выведен в двух местах на индикаторе.

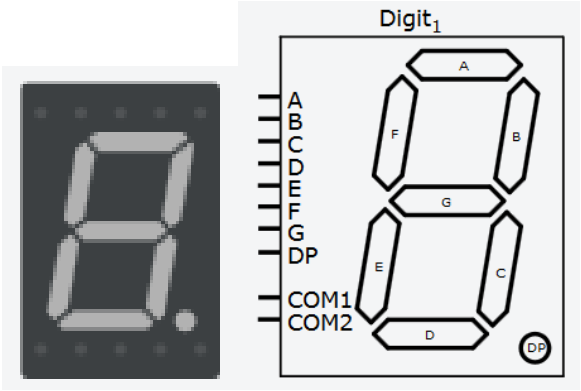
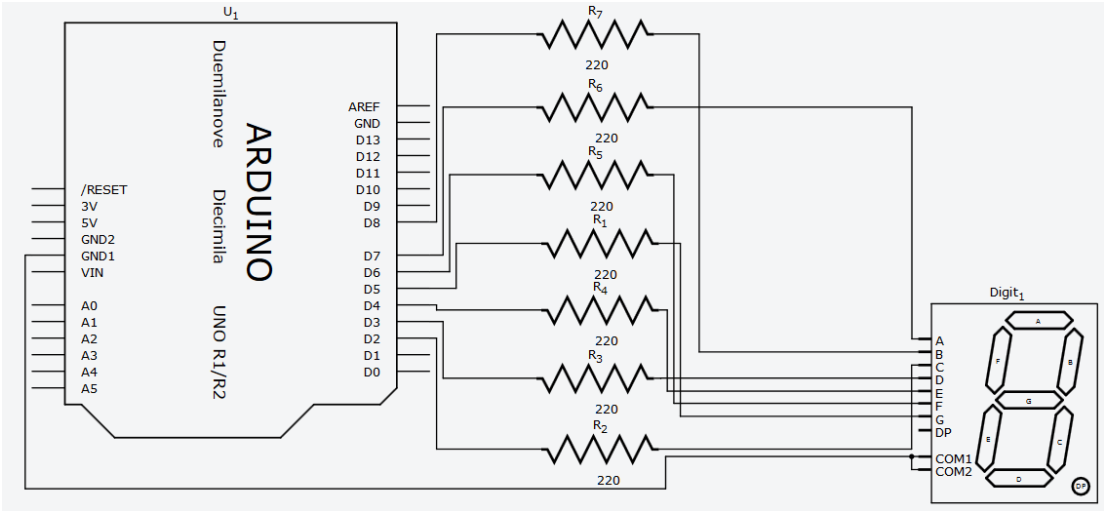


Рис. 5.1: Сегментный индикатор

Подключение семисегментного индикатора осуществляется так же, как и обычного светодиода. Так как в индикаторе присутствуют 7 диодов, потребуется использовать 7 выводов микроконтроллера и 7 резисторов.



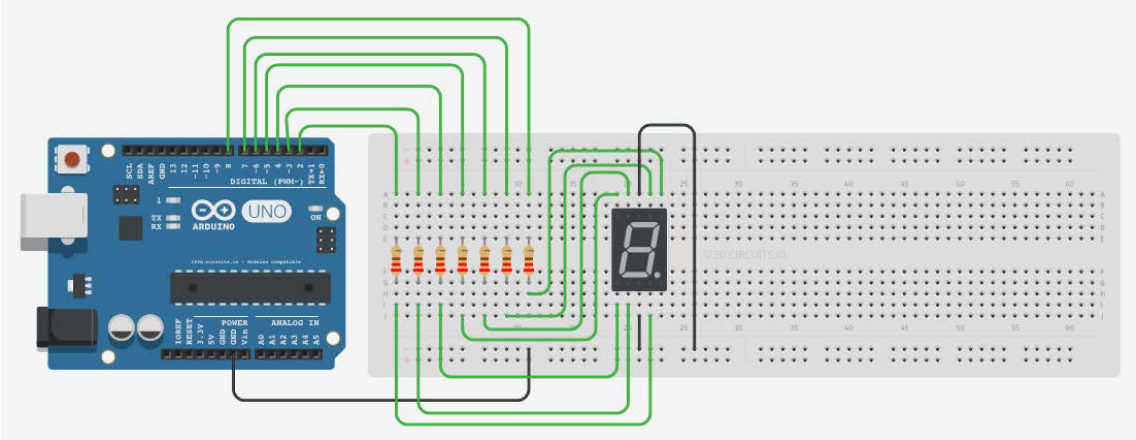


Рис. 5.2: Пример подключения семисегментного индикатора

Программа, приведенная ниже, отображает цифру 5, включая сегменты a, f, g, c, d.

```
int seg_a = 7;
int seg_b = 8;
int seg_c = 2;
int seg_d = 3;
int seg_e = 4;
int seg_f = 6;
int seg_g = 5;

void setup() {
     pinMode( seg_a, OUTPUT );
     pinMode( seg_b, OUTPUT );
     pinMode( seg_c, OUTPUT );
     pinMode( seg_d, OUTPUT );
     pinMode( seg_e, OUTPUT );
     pinMode( seg_f, OUTPUT );
     pinMode( seg_g, OUTPUT );
     digitalWrite( seg_a, HIGH );
     digitalWrite( seg_f, HIGH );
     digitalWrite( seg_g, HIGH );
     digitalWrite( seg_c, HIGH );
     digitalWrite( seg_d, HIGH );
}
void loop() {}
```

Задание 5.1

Реализуйте программу, осуществляющую анимацию сегментов. После включения платы на индикаторе должны по очереди загораться и потухать сегменты в порядке: верхний, правый верхний, правый нижний, нижний, левый нижний, левый верхний и т.д.

Задание 5.2

Реализуйте светофор с обратным отсчетом. Добавьте в созданный ранее в Задании 4.2 светофор семисегментный индикатор, на котором будет выводиться количество секунд, оставшихся до конца очередной фазы его работы. Увеличьте длительность фазы до 9 секунд.

Задание 5.3

Реализуйте секундомер. Для этого соберите схему с кнопкой и сегментным индикатором. При запуске платы на индикаторе высвечивается `0`. После нажатия на кнопку каждую секунду цифра увеличивается на единицу. После `9` счетчик переходит в `0` и считает дальше. При втором нажатии на кнопку счет останавливается, а при третьем нажатии значение на индикаторе сбрасывается на `0`. Дальше все повторяется заново.

5.2 Жидкокристаллический дисплей

Самый информативный и вместе с тем самый сложный вид индикаторов – это *дисплеи*. В нашем учебном пособии мы будем работать с *жидкокристаллическим индикатором*, представителей которого можно встретить во многих электронных устройствах: электронные часы, калькуляторы, экраны старых сотовых телефонов – все они используют подобные ЖК индикаторы.

Мы будем использовать дисплей 16 x 2, что означит, что он умеет отображать 2 строки по 16 символов. Внешний вид дисплея и его обозначение на схеме показаны на Рис. 5.3.

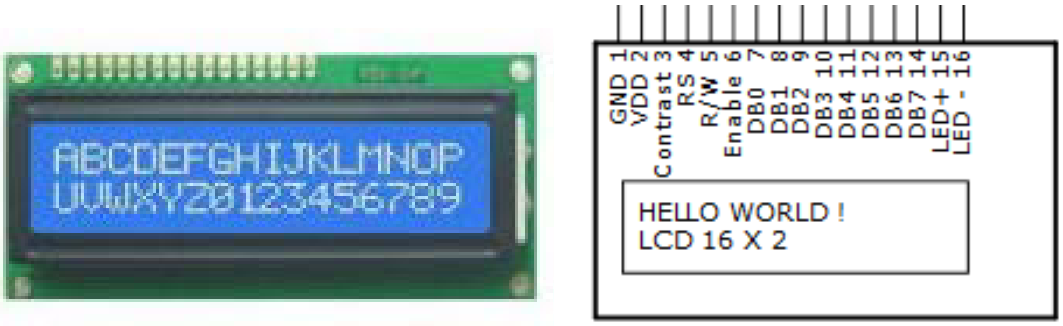


Рис. 5.3: Внешний вид дисплея и его обозначения на схеме

Назначение контактов индикатора следующее:

- GND – земля (минус питания);
- VDD – питание +5В;
- Contrast – регулятор контраста дисплея;
- RS – выбор регистра;
- R/W – направление передачи данных (запись/чтение);
- Enable – синхронизация;
- DB0…DB7 – шина данных;
- Led- – катод подсветки;
- Led+ – анод подсветки.

Схема и вид модели подключения дисплея показаны на Рис. 5.4. Кроме подключения информационных сигналов и питания нужно обеспечить регулировку контраста дисплея. Для этого подсоединяют потенциометр 10 kOm таким образом, как показано на схеме. Подробнее о потенциометрах речь пойдет в следующей главе. Если нужна подсветка дисплея, то ее подключают с помощью контактов LED- и LED+ через резистор не менее 100 Ом.

Для работы с ЖК дисплеями подобного типа различных размеров в среде Arduino IDE имеется специальная библиотека LiquidCrystal. Для того, чтобы воспользоваться этой библиотекой в программе, необходимо в начале кода указать следующую директиву:

```
#include <LiquidCrystal.h>
```

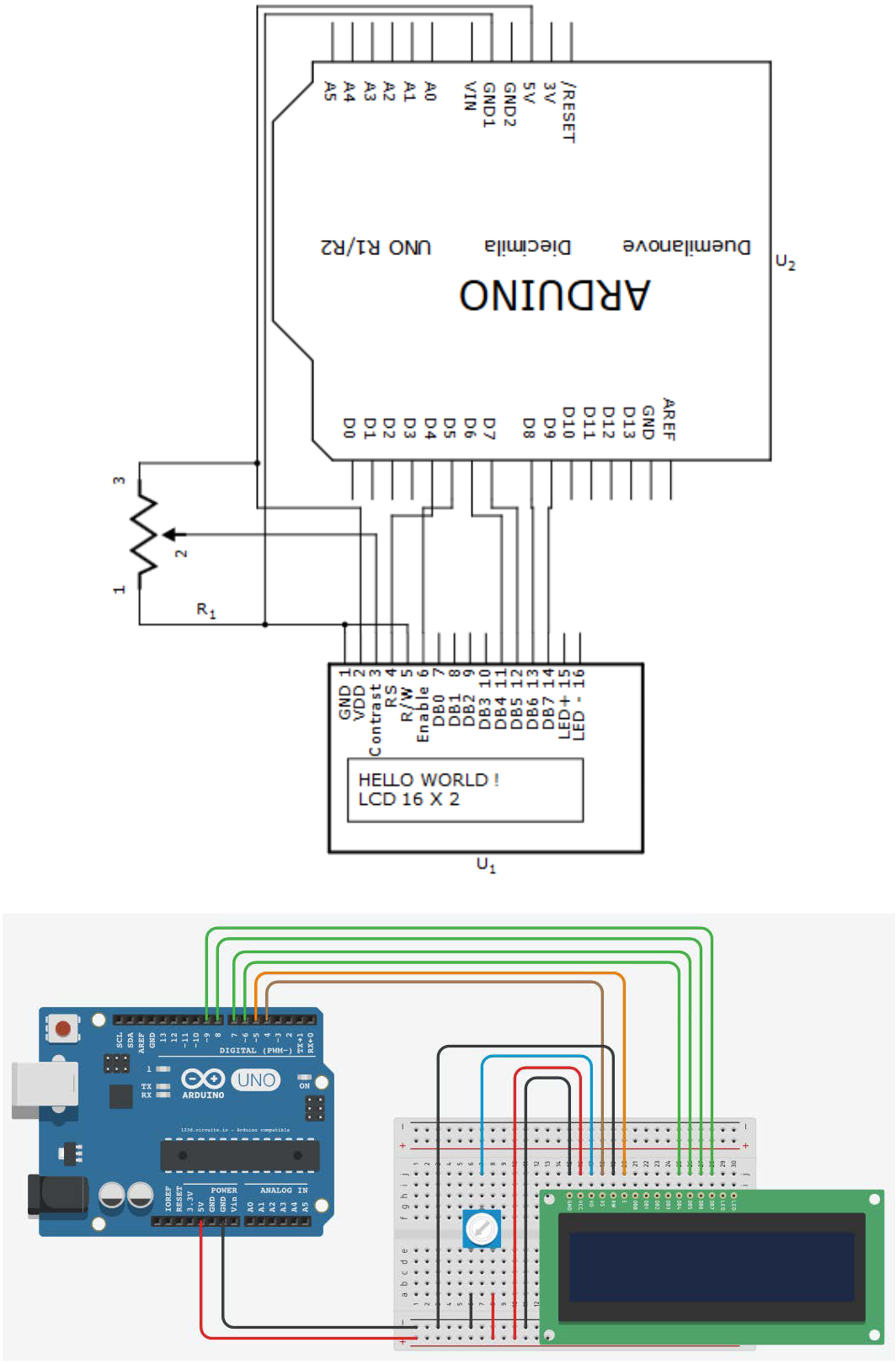


Рис. 5.4: Пример подключения дисплея к общей схеме

После этого надо инициализировать дисплей, указав, какие именно выводы используются для подключения к Arduino:

```
LiquidCrystal lcd(4, 5, 6, 7, 8, 9);
```

где первый аргумент – контакт RS, второй – Enable, а с третьего по шестой – DB4..DB7. Другие выводы дисплея подключать к Arduino не требуется.

Вывод текста на дисплей осуществляется с помощью функции `print` следующим образом:

```
lcd.print( текст_или_переменная );
```

В качестве аргумента функции можно передавать текст, выделенный двойными кавычками, либо переменные любых типов.

В библиотеке предусмотрена функция для явного указания позиции вывода:

```
lcd.setCursor( номер_колонки, номер_строки );
```

Так, для вывода символа во вторую строку потребуется вызвать `setCursor` со следующими значениями аргументов:

```
// колонка с индексом 0, строка с индексом 1
lcd.setCursor( 0, 1 );
```

Попробуем вывести на дисплей сообщение «hello, world!»:

```
#include <LiquidCrystal.h>

// инициализация дисплея
LiquidCrystal lcd(4, 5, 6, 7, 8, 9);

void setup() {
    // параметры дисплея: 16 символов, 2 строки
    lcd.begin(16, 2);
    // вывод на дисплей текста
    lcd.print("hello, world!");
}

void loop() {
}
```

Задание 5.4

Реализуйте секундомер с выводом времени на дисплей. Для этого добавьте к схеме подключения дисплея две кнопки. Напишите программу, которая по одной кнопке запускает и останавливает секундомер, а по второй – сбрасывает его показания в `0`. Когда секундомер запущен, на дисплее должно отображаться прошедшее с момента его запуска время в формате `минуты:секунды`.

Задание 5.5

Реализуйте вывод бегущей строки “`Hello, world!`”. После подачи питания в первой строке дисплея должен появиться перемещающийся справа налево текст.

Тема 6

Работа с аналоговыми сигналами

6.1 Сенсоры. Резистивные сенсоры

В предыдущих главах для ограничения силы тока через светодиод мы использовали резисторы. Как было тогда отмечено, существует множество резисторов разного номинала, рассчитанных на разную мощность нагрузки. Кроме обычных резисторов есть аналогичные приборы, но только с изменяемым сопротивлением, которые называются *потенциометрами* (или *реостатами*).

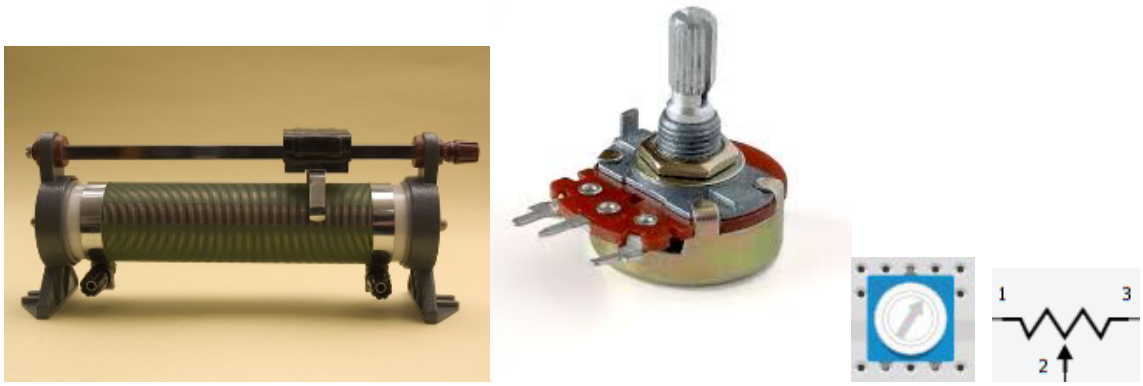


Рис. 6.1: Реостаты

На Рис. 6.1 показано несколько изображений реостатов, его обозначение на электрических схемах и в моделях Autodesk CIRCUITS.

Для чего нужен потенциометр? Во-первых, исходя из описания прибора, его можно использовать для задания требуемого сопротивления в случае, если нет подходящего резистора или нужна точная регулировка или настройка уже готового устройства. Во-вторых, с помощью потенциометра можно сделать регулируемый делитель напряжения – устройство, которое позволяет передавать на нагрузку только часть напряжения от источника. В этой главе мы будем использовать реостат именно в роли делителя напряжения, варьируя напряжение на одном из аналоговых входов Arduino. Такой реостат является простейшим аналоговым датчиком, который сообщает угол поворота вокруг своей оси.

Во время работы с кнопками мы уже познакомились с функцией `digitalRead`, которая умеет считывать цифровой сигнал с определенного вывода контроллера. У этой функции существует аналоговая версия `analogRead`, которая может делать то же самое, но только для аналогового сигнала:

```
результат = analogRead( номер_контакта );
```

После ее вызова в программе переменная `результат` будет хранить уровень аналогового сигнала, считанный с соответствующего контакта. При этом, в отличие от цифрового сигнала, функция `analogRead` возвращает число от `0` до 1023, где `0` означает напряжение `0` вольт, а 1023 – 5 вольт. Промежуточным напряжениям будут соответствовать промежуточные значения.

Подключать потенциометр надо к аналоговым входам `A0..A5` платы Arduino как показано на Рис. 6.2.

Напишем программу, которая будет измерять положение потенциометра раз в полсекунды и отправлять его через последовательный порт:

```
const int analog_R = A0;

void setup() {
     pinMode(analog_R, INPUT);
     Serial.begin(9600);
}
```

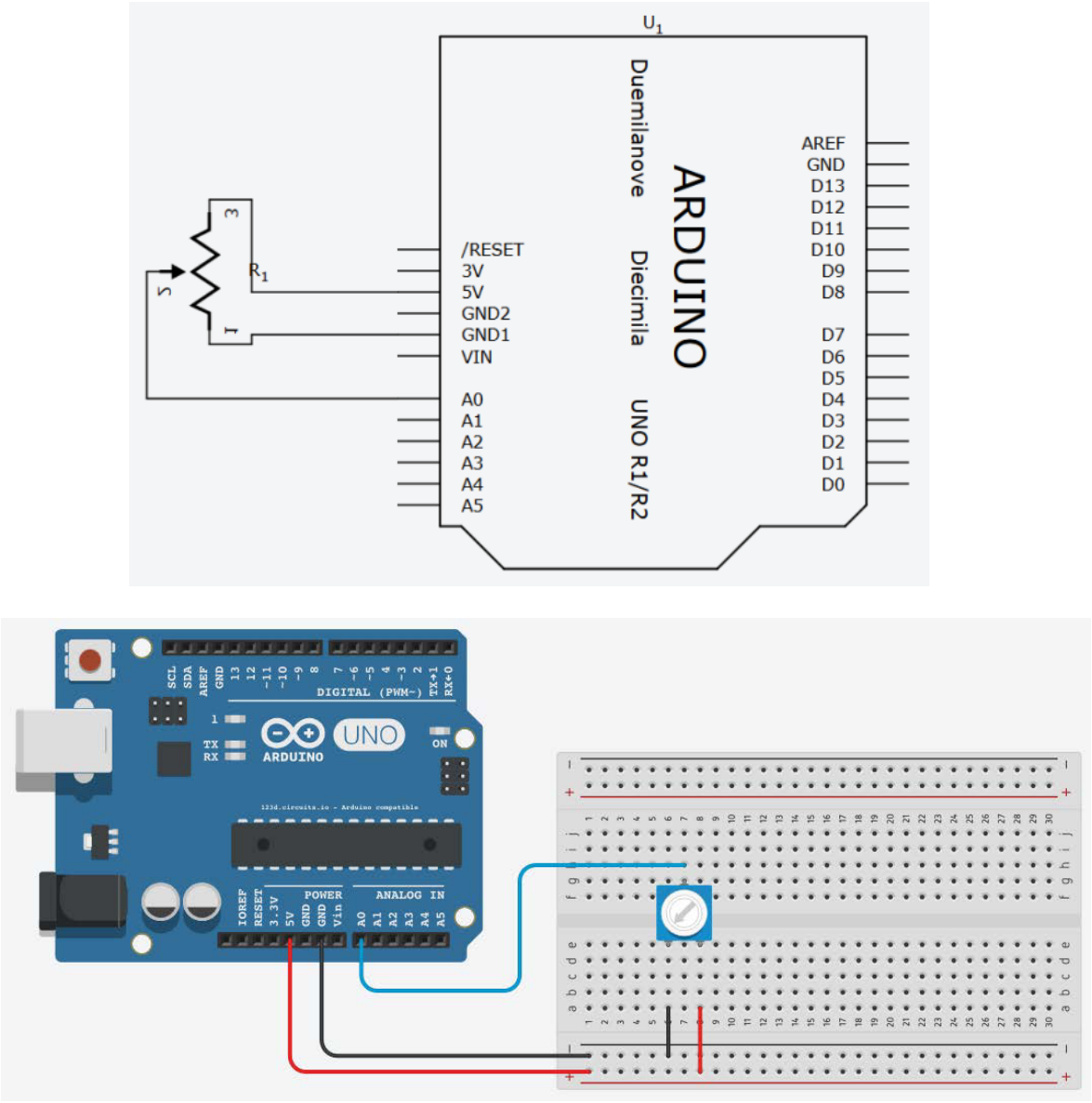


Рис. 6.2: Пример подключения реостата

```
void loop() {
     int result = analogRead( analog_R );
     Serial.println( result );
     delay(500);
}
```

Обратите внимание, что если вы будете использовать несколько аналоговых датчиков, то при последовательном чтении показания с них могут считываться неустойчиво. Это может происходить потому, что датчики влияют друг на друга в момент переключения

АЦП между разными входами. В таком случае можно показания с каждого входа считывать два раза с задержкой между замерами и отбрасывать результаты первого чтения.

Задание 6.1

Разработайте схему с потенциометром и жидкокристаллическим экраном, которая бы выводила на экран текущее значение, читаемое с потенциометра.

Задание 6.2

Разработайте схему с потенциометром и семисегментным индикатором, которая бы отображала напряжение на потенциометре в вольтах. Нужно показывать одну цифру напряжения, так чтобы показание `0` на аналоговом входе отображалось как «`0`» на индикаторе, а 1023 – как «`5`».

6.2 Аналоговый датчик температуры

Распространенным вариантом датчиков с аналоговым выходом являются датчики температуры. Мы будем использовать датчик TMP36, который изображен на Рис. 6.3.

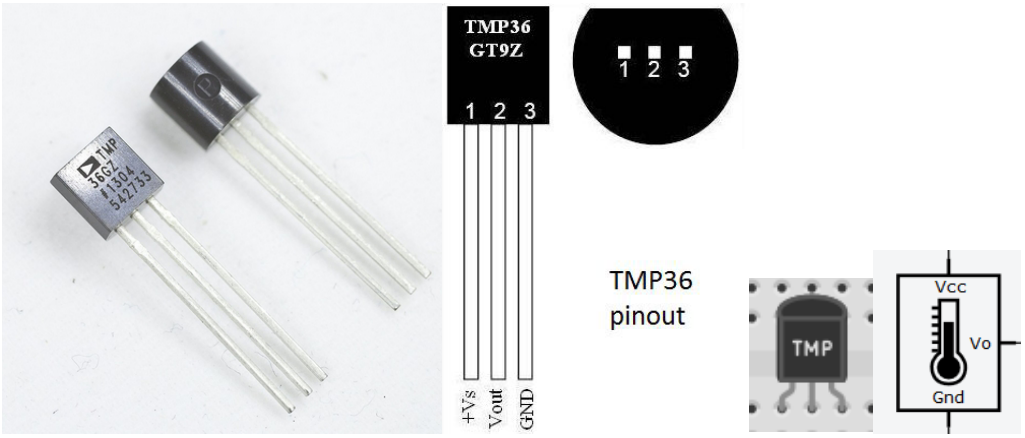


Рис. 6.3: Аналоговый датчик температуры TMP36

Назначение его контактов следующее:

- Vs – питание 2.7 - 5.5В (в нашем случае +5В);
- Vout – выходное напряжение, зависящее от температуры;
- Gnd – земля.

Пример подключения датчика изображен на Рис. 6.4.

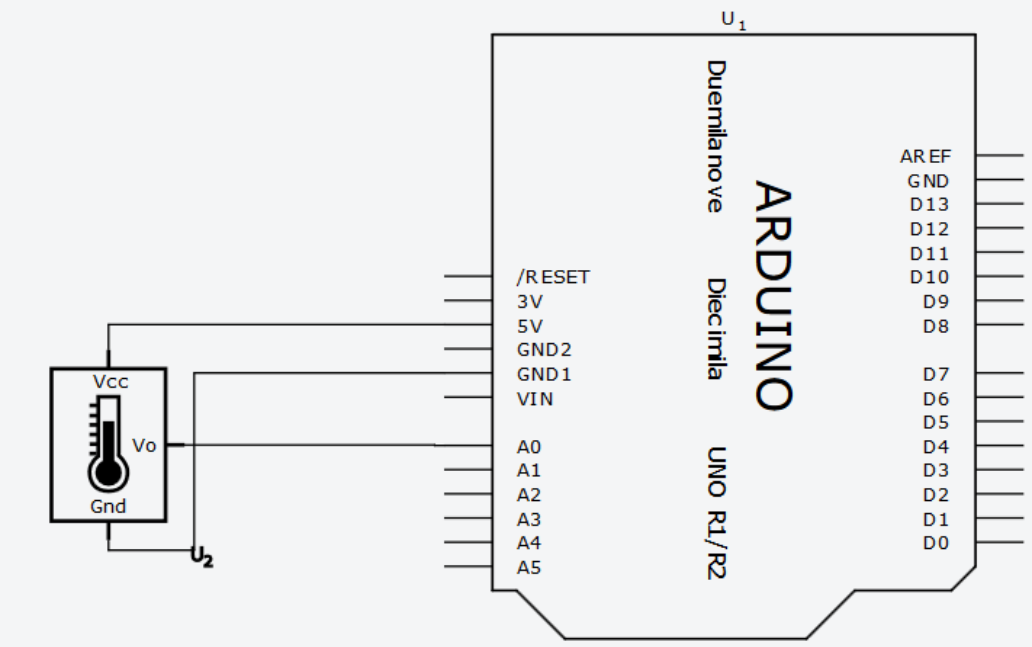
Выходное напряжение сенсора зависит только от температуры и не зависит от напряжения питания. Для того чтобы рассчитать температуру, нужно воспользоваться формулой:

$$T = \frac{V_{out} - 500}{10},$$

где V_{out} – выходное напряжение датчика в милливольтах, T – температура в градусах Цельсия. Учитывая, что напряжению 5 V соответствует значение 1023, получаем следующую формулу:

$$T = \frac{Reading * \frac{5000}{1023} - 500}{10},$$

где $Reading$ – значение, полученное в результате вызова функции `analogRead`.



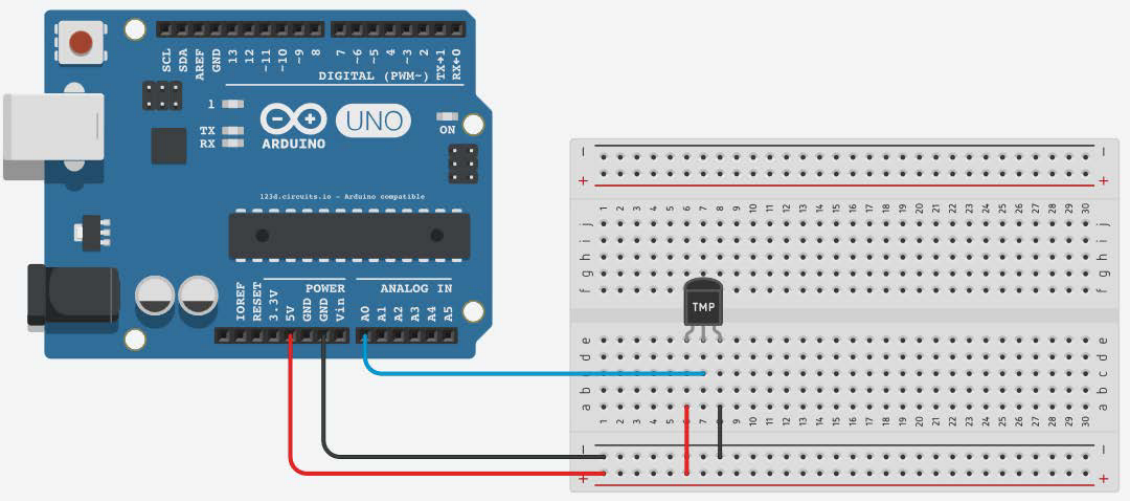


Рис. 6.4: Пример подключения аналогового датчика температуры

Задание 6.3

Реализуйте программу, которая каждую секунду выводит в последовательный порт температуру, измеренную аналоговым датчиком. Температура должна выводиться в градусах Цельсия.

Рекомендация: для расчетов используйте значения с плавающей точкой (`float`). Для этого следует задавать константы в виде «`5000.0`».

Задание 6.4

Соберите схему, включающую светодиод, зуммер и датчик температуры. При превышении температуры заданного значения (например, 25 С), должен загораться светодиод и подаваться периодические сигналы зуммером.

6.3 Датчик света

В этом разделе мы рассмотрим еще один аналоговый датчик – датчик света, который называют *фоторезистором* (Рис. 6.5).

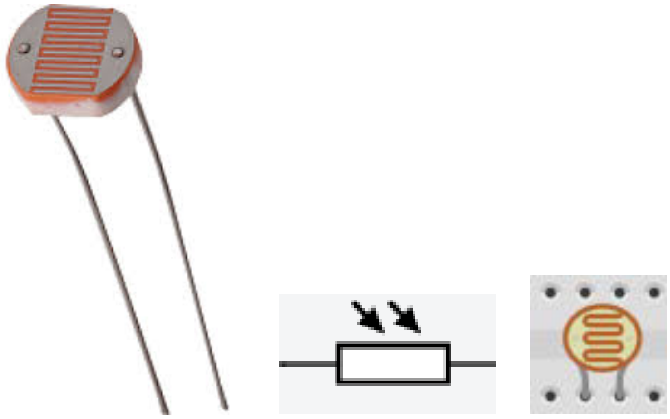


Рис. 6.5: Фоторезистор

Фоторезистор – это простой компонент с двумя входами, который изменяет свое сопротивление в зависимости от уровня освещенности. В отличие от датчика температуры TMP36 фоторезистор не выдает напряжение, зависящее от измеряемого им параметра. Он может только изменять свое сопротивление.

Чтобы считать показания фоторезистора, надо сначала превратить изменение сопротивления в изменение напряжения. Для этого используют схему, которая называется *делителем напряжения*. Соединим один вывод фоторезистора с напряжением питания, а ко второму подключим обычный резистор, соединенный с землей. Напряжения на резисторах будут пропорциональны их сопротивлениям. Например, если сопротивление фоторезистора 20 kOm, а обычного резистора – 10 kOm, то напряжение на фоторезисторе будет в два раза больше, чем на обычном. Если, как в нашем случае, напряжение питания 5 вольт, то эти значения составят 3.33 и 1.66 вольта. При изменении сопротивления фоторезистора напряжения будут меняться. Соединив среднюю точку между резисторами с аналоговым входом платы Arduino можно измерять напряжение на обычном резисторе и делать выводы об освещенности.

Фоторезисторы, которые обычно идут в наборах с платами Arduino, как правило изменяют сопротивление от 200 kOm (полная темнота) до 1 kOm (яркость 10 люкс). Экспериментируя с датчиком, можно заметить, что показания на аналоговом входе изменяются не линейно, это особенность работы датчика.

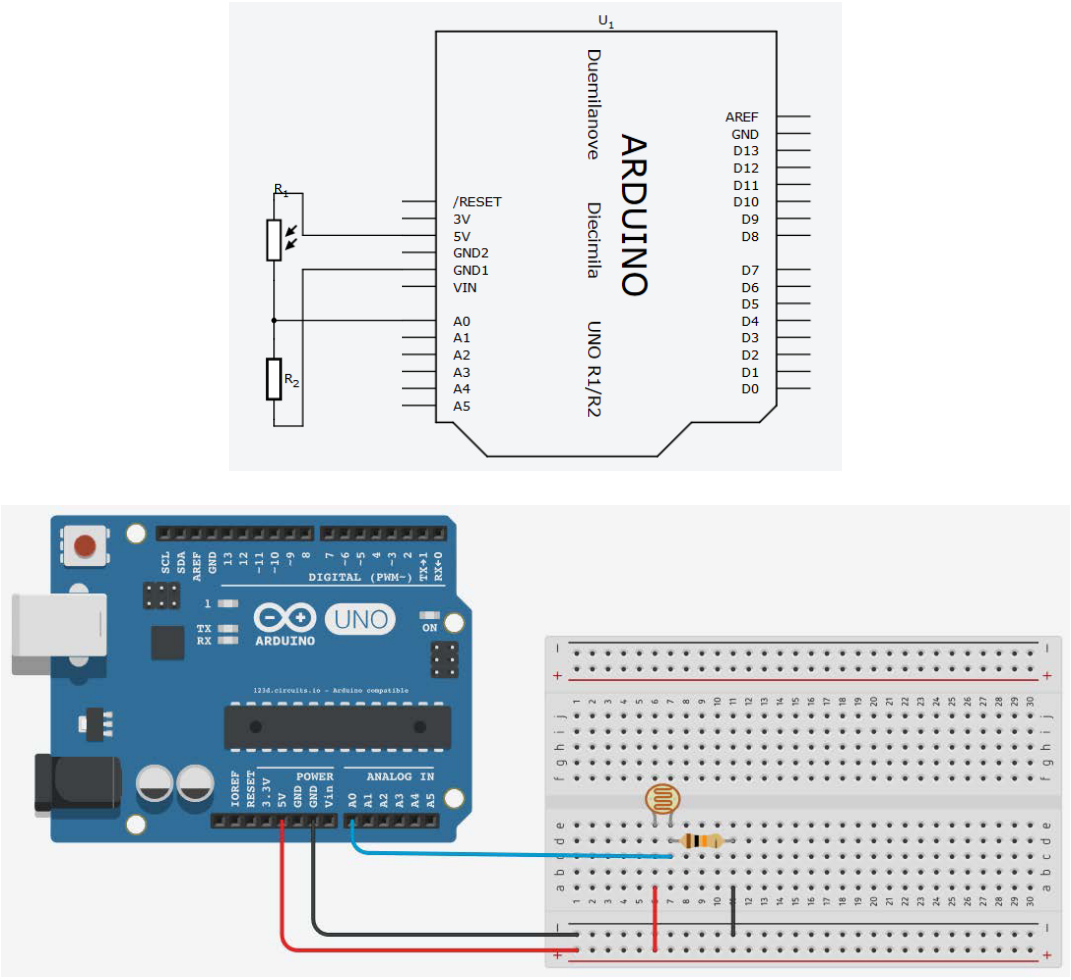


Рис. 6.6: Пример подключения фоторезистора

Задание 6.5

Реализуйте автоматический выключатель света. Подключите к плате Arduino фоторезистор и несколько светодиодов. Запрограммируйте микроконтроллер таким образом, чтобы он включал светодиоды при снижении уровня освещения и включал их, если опять становится светлее.

В этой задаче может быть разумно ввести два отдельных порога освещенности – один, на котором освещение включается, и другой, немного больший, на котором выключается. Это поможет избежать мигания света в случае, если яркость близка к пороговой, так как показания реального датчика всегда содержат небольшой шум.

Тема 7

Управление электроприводами

7.1 Приводы. Аналоговые приводы. PWM

Как нам уже известно, микроконтроллер умеет работать исключительно с цифровыми данными. Он легко может выполнять арифметические операции над ними, принимать и передавать цифровые сигналы. Например, мы можем зажечь светодиод, подав на него положительный сигнал, равный напряжению питания контроллера. Для того чтобы погасить светодиод, просто отключим его от питания. Получается, для управления мы используем только ноль или единицу, что и называется цифровым управлением.

Но что делать, если нам нужно зажечь этот самый светодиод только на половину яркости? Или запустить двигатель на 30% его мощности? Для того чтобы это сделать, нам потребуется преобразовать цифровой сигнал в некое подобие аналогового сигнала с заданным уровнем напряжения.

Для этой цели можно использовать специальное устройство, называемое *цифроаналоговым преобразователем* (ЦАП, DAC). ЦАП умеет генерировать нужный уровень напряжения, который задается микроконтроллером в цифровом виде. Однако, такой способ избыточен для многих задач. Кроме того, если потребуется управлять мощным двигателем, придется использовать очень дорогой ЦАП.

Другой способ генерации сигнала с заданным уровнем напряжения называется *широтно-импульсной модуляцией* (ШИМ, PWM). Этот способ позволяет обойтись чисто цифровыми устройствами, что значительно упрощает схему и тем самым значительно удешевляет процесс создания устройства.

Идея этого метода состоит в том, чтобы подавать энергию на двигатель (или источник света) не постоянно, а с перерывами: короткое время напряжение есть, а потом его нет, потом опять есть и так далее. Если подать такой сигнал на лампу, то она будет светиться более тускло, чем если подавать на нее напряжение постоянно, ведь часть времени она не получает электричества. То же самое с двигателем – он будет крутиться медленнее. Регулируя время подачи и отсутствия напряжения, можно заставить лампочку светиться с разной яркостью или двигатель крутиться с разной скоростью.

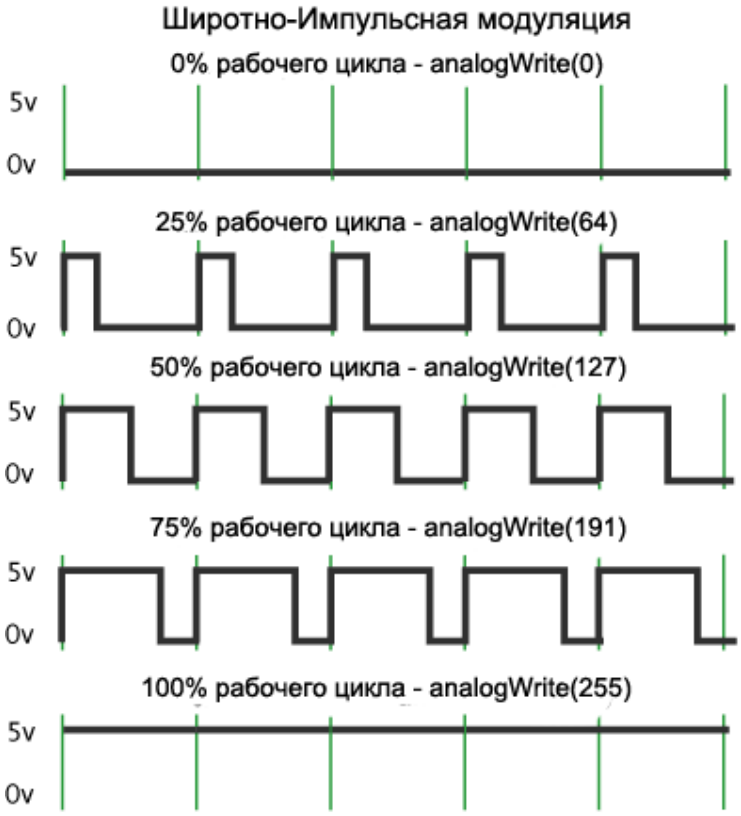


Рис. 7.1: Сигналы широтно-импульсной модуляции

На Рис. 7.1 показаны сигналы широтно-импульсной модуляции. Одним из основных параметров ШИМ является процент времени,

в течение которого на выход подается высокий уровень напряжения. На верхнем графике этот процент равен `0`, и напряжение всегда остается на нулевом уровне. С таким сигналом лампочка не будет гореть, а мотор будет стоять на месте. С другой стороны, сигнал с заполнением 100% рабочего цикла – это постоянно выведенный высокий уровень напряжения. Он заставит подключенный исполнительный механизм работать на максимальной мощности. Промежуточные значения позволят регулировать, насколько ярко будет светиться лампочка или как быстро будет крутиться мотор.

Другой важный параметр ШИМ сигнала – это его частота. Она определяет время, которое занимает один цикл изменения сигнала. На Рис. 7.1 это время показано зелеными линиями. Если мы управляем светодиодом и частота будет слишком низкой, то вместо плавной регулировки яркости мы увидим мигающий светодиод, так как светодиоды очень быстро реагируют на включение и выключение питания.

С лампой накаливания эффект будет менее заметным, так как она излучает свет за счет нагрева нити, а нить остывает гораздо медленнее, чем выключается светодиод. Тем не менее, такой эффект можно будет заметить и на обычных лампах накаливания, если сильно уменьшить частоту.

В Arduino частота ШИМ равна 500 kHz, что означает, что интервал времени между двумя зелеными линиями равен 2 миллисекунды.

Чтобы выдать ШИМ сигнал на один из портов Arduino, используют функцию `analogWrite`:

```
analogWrite( номер_контакта, уровень_сигнала );
```

После вызова `analogWrite` на выходе будет генерироваться ШИМ сигнал с заданной шириной импульса до следующего вызова `analogWrite` (или вызова `digitalWrite` или `digitalRead` на том же порту входа/выхода).

Уровень сигнала задается числом от `0` до 255, где `0` соответствует `0`% заполнения, а 255 – 100%.

Необходимо учитывать, что генерация ШИМ сигнала поддерживается не на всех выходах микроконтроллера. На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11, а на плате Arduino Mega

порты 2-13. На более ранних версиях плат Arduino `analogWrite` работал только на портах 9, 10 и 11. Выводы, поддерживающие этот режим, на схеме платы обозначают аббревиатурой ШИМ или PWM.

Отметим, что функция `analogWrite` никак не связана с аналоговыми входами и с функцией `analogRead`. В данном случае совпадение есть только в названии. Для вызова `analogWrite` нет необходимости устанавливать тип входа/выхода функцией `pinMode`.

Рассмотрим программу, увеличивающую яркость светодиода. Эта программа используется с такой же схемой, которая использовалась в самом первом примере подключения светодиода.

```
int led = 2;

void setup() {
     analogWrite( led, 0 );
     delay(1000);
     analogWrite( led, 32 );
     delay(1000);
     analogWrite( led, 64 );
     delay(1000);
     analogWrite( led, 128 );
     delay(1000);
     analogWrite( led, 255 );
     delay(1000);
}

void loop() {
}
```

Обратите внимание, что эмулятор Autodesk CIRCUITS не показывает включение светодиода на низких уровнях заполнения ШИМ сигнала.

Задание 7.1

Реализуйте программу, которая будет обеспечивать плавное загорание, а затем плавное затухание светодиода с периодом 1 секунда.

Задание 7.2

Соберите схему с светодиодом и потенциометром и реализуйте программу, которая будет регулировать яркость светодиода на основании положения потенциометра.

7.2 Управление электрическим двигателем

Нам уже известно, что микроконтроллер устройство, которое оперирует малыми токами и напряжениями. Так большинство современных микроконтроллеров работают с напряжениями 5 вольт, 3.3 вольта либо еще меньше. При этом сила тока, проходящего через выводы микроконтроллеров, обычно не должна превышать 20...40 миллиампер.

Такого тока достаточно, чтобы зажечь светодиод или включить зуммер. Однако для управления электрическим двигателем мощности микроконтроллера недостаточно. Даже для небольшого двигателя потребуется напряжение 5-6 Вольт и сила тока в сотни миллиампер. Такая нагрузка непременно уничтожит любой микроконтроллер за доли секунды.

Поэтому для управления двигателем нужно использовать дополнительные устройства, которые бы позволили управлять большими токами, проходящими через двигатель. В самом простом случае в роли такого ключа можно использовать мощные транзисторы. Однако удобнее использовать специальные микросхемы-драйверы двигателей. Такие микросхемы содержат все необходимые компоненты для коммутации тока и позволяют управлять направлением и скоростью вращения двигателя.

Мы будем использовать микросхему L293D (Рис. 7.2), которая позволяет управлять сразу двумя двигателями с напряжением питания от 4.5 до 36 вольт и постоянным током до 600 мА. Этого достаточно, чтобы управлять двигателями небольшого робота.

Микросхема позволяет управлять сразу двумя двигателями, при этом ее выходы распределены так, что с одной стороны находится управление одним двигателем, а с другой – другим. Их работа совершенно идентична, поэтому мы рассмотрим только одну ее сторону.

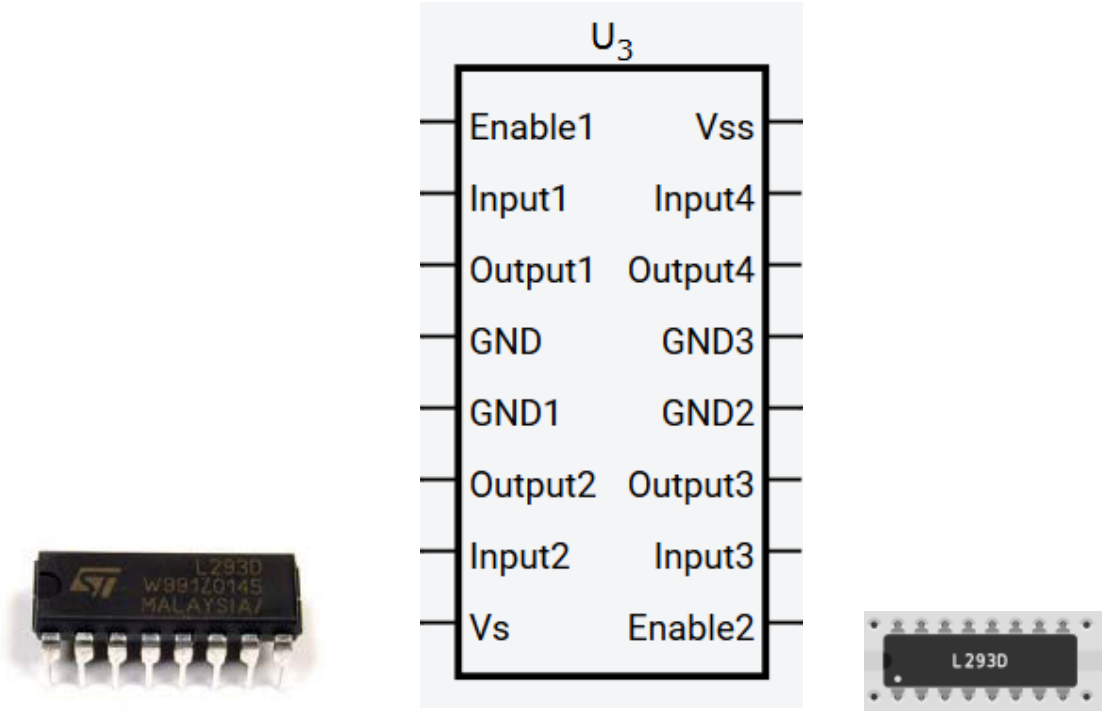


Рис. 7.2: Микросхема L293D

Несомненным плюсом данной микросхемы является раздельное питание ее логической части, напряжение которого лежит в пределах 4.5-5 вольт (контакт **VSS**), и силовой части питания двигателей (контакт **VS**). Дело в том, что мощные двигатели создают помехи в цепях питания, которые могут приводить к сбоям и перезапускам микроконтроллера. Поэтому желательно разделять питание микроконтроллера и двигателей.

Разберем теперь выводы для управления первым (левым на схеме) двигателем. Вывод `ENABLE1` – это главный вход управления левым каналом. Без сигнала единицы (или ШИМ) на нем, двигатель работать не будет вне зависимости от того, что подается на входы `INPUT1` и `INPUT2`.

Выводы `INPUT1` и `INPUT2` задают направление вращения мотора. Чтобы заставить двигатель вращаться в одну сторону, надо подать логическую единицу на вывод `INPUT1`, а на `INPUT2` подать логический ноль. Для смены направления нужно поменять местами: на INPUT1 подать `0`, а на `INPUT2` – 1.

При подаче одинаковых сигналов на `INPUT1` и `INPUT2` мотор вращаться не будет, следовательно, вращение можно остановить либо подачей логического нуля на вывод `ENABLE1` при любой

конфигурации `INPUT1` и `INPUT2`, либо одинаковыми сигналами на `INPUT1` и `INPUT2`, не изменяя уровня сигнала на выводе `ENABLE1`.

Выводы GND соединяются с отрицательным полюсом источника питания (земля). Оставшиеся выводы `OUTPUT1` и `OUTPUT2` служат непосредственно для подключения мотора.

Соберем схему, в которой подключим один двигатель с помощью микросхемы L392D (Рис. 7.3). Обратите внимание, что для двигателя используется отдельный источник питания, при этом «земля» обоих источников соединена. Вход `ENABLE2`, управляющий вторым мотором, отключен благодаря подключению его к земле, так как в этой схеме второй двигатель не используется.

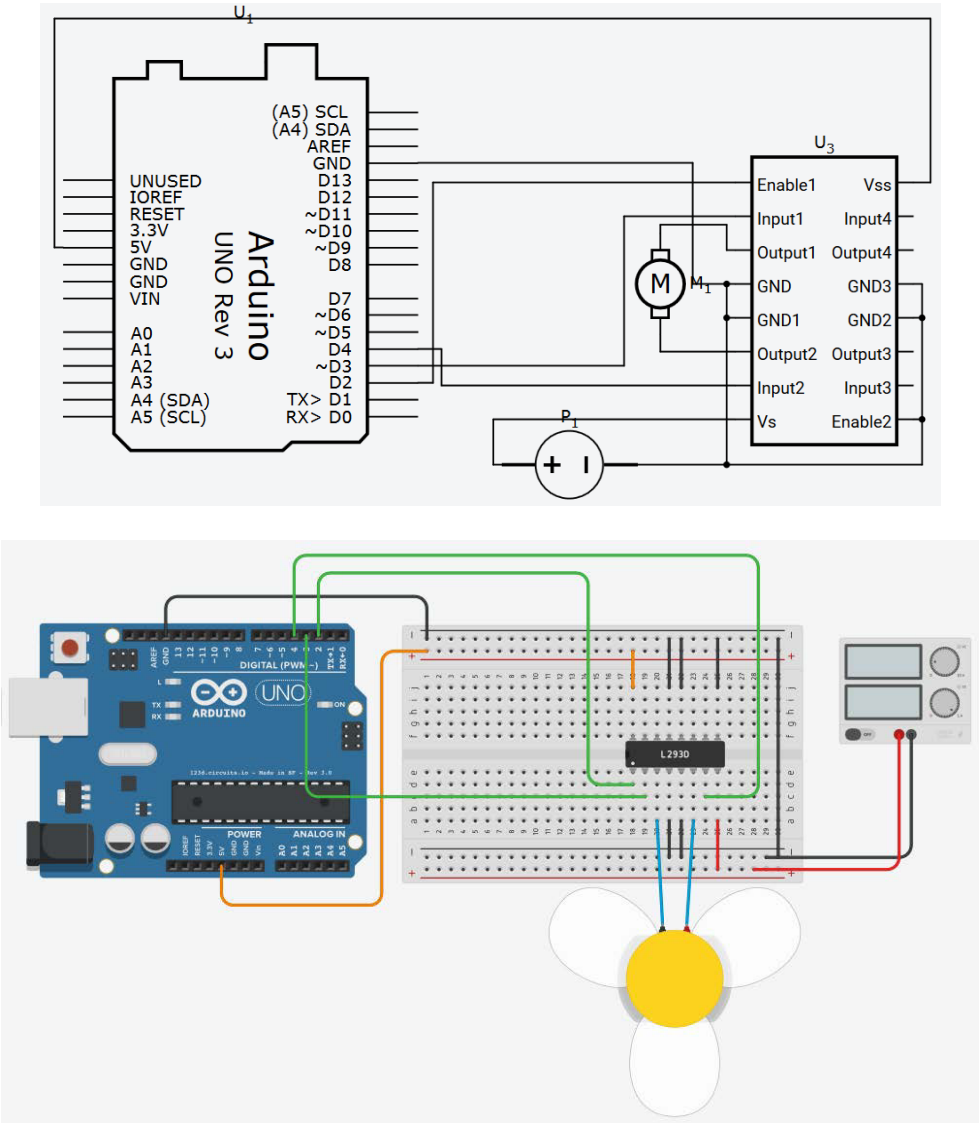


Рис. 7.3: Пример подключения микросхемы L293D

Следующая программа демонстрирует включение двигателя с помощью подачи ШИМ сигнала на вход ENABLE1:

```
const int m_enable = 2;// выход сигнала ENABLE1
const int m_dir1 = 3;  // выход выбора направления INPUT1
const int m_dir2 = 4;  // выход выбора направления INPUT2

void setup() {
    // конфигурация контактов на режим выходов
    pinMode(m_enable, OUTPUT);
    pinMode(m_dir1, OUTPUT);
    pinMode(m_dir2, OUTPUT);

    // выбор направления вращения
    digitalWrite(m_dir1,HIGH);
    digitalWrite(m_dir2,LOW);

    // управление вращением с помощью PWM
    analogWrite(m_enable, 150);
}
void loop() {
}
```

При резком включении двигателей микросхема L293D начинает очень сильно нагреваться по причине того, что при старте сразу на полную мощность, для того чтобы сдвинуться с места, мотор потребляет ток, хоть и кратковременно, но минимум в 2-5 раз больше установившегося рабочего значения. При резкой смене направления вращения появляется еще больший скачок тока, так как приходится не только сдвинуть якорь с места, как это было в первом случае, но и побороть его инерцию движения в другом направлении.

Для снижения этих факторов следует разгонять двигатели плавно – чем дольше, тем лучше (в разумных пределах). При изменении направления вращения следует дать промежуток времени для самостоятельной остановки двигателя (доли секунды), а затем снова плавно разгонять. Приложив палец к микросхеме, можно почувствовать разницу. Поэкспериментируйте и найдите золотую середину между нагревом и скоростью реагирования для вашей задачи.

Задание 7.3

Измените программу, написанную выше, чтобы двигатель попеременно вращался то в одну сторону, то в другую. Период смены направления – пять секунд.

Задание 7.4

Реализуйте плавное управление скоростью работы двигателя. При включении двигатель должен начать плавно увеличивать обороты и за 5 секунд достичь максимальной скорости. Затем он должен также плавно остановиться и начать вращаться в другую сторону.

Задание 7.5

Реализуйте схему для управления движением робота, которая должна обеспечить управление двумя двигателями и содержать 4 кнопки, нажатием на которые определяется направление движения:

- первая кнопка – двигаться вперед (оба мотора вперед);
- вторая – поворот направо (правый мотор вперед, левый – назад);
- третья – поворот налево (левый мотор вперед, правый назад);
- четвертая – двигаться назад (оба мотора вращаются назад).

7.3 Управление сервоприводами

Еще одним электромеханическим исполнительным механизмом, часто применяющимся в робототехнике и моделировании, является *серводвигатель* (*сервопривод*, *сервомашинка*), изображенный на Рис. 7.4.

В отличие от обычных двигателей, серводвигатель умеет поворачивать вал строго на заданный угол. Это полезное свойство сервомашинок часто используют в авиамоделировании для управления

элеронами, рулями высоты и тому подобным. Мощные серводвигатели можно использовать для создания механических рук-манипуляторов.

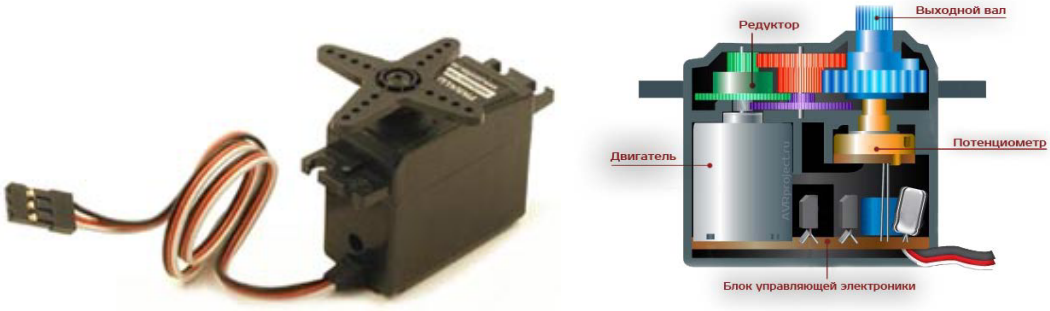


Рис. 7.4: Серводвигатель

Устроен такой двигатель достаточно сложно. В верхней части прибора размещается шестеренный редуктор, который позволяет значительно увеличить крутящий момент двигателя постоянного тока за счет снижения скорости его вращения. Ниже расположен потенциометр, задача которого – определять на какой угол повернут выходной вал редуктора. Наконец, в глубине корпуса находится небольшая плата управления, которая и делает серводвигатель умным. Эта плата постоянно отслеживает текущее положение вала и корректирует его в случае, если вал пытается уйти из заданной позиции.

Управляется серводвигатель с помощью ШИМ сигнала. Но, в отличие от обычного двигателя постоянного тока, здесь уровень ШИМ задает не скорость вращения, а угол поворота. Кроме того, так как серводвигатель содержит всю необходимую силовую электронику, управлять им можно непосредственно с порта микроконтроллера.

В Arduino IDE имеется специальная библиотека для работы с серводвигателями – `Servo`. Для подключения двигателя необходимо создать соответствующий объект:

```
Servo myservo;
```

Затем в функции `setup` выполнить его инициализацию:

```
myservo.attach( номер_контакта );
```

Здесь `номер_контакта` – номер ножки микроконтроллера, к которому подключен управляющий вход серводвигателя. Этот контакт должен поддерживать работу в режиме PWM. Поворот вала на угол осуществляется функцией `write`:

```
myservo.write( угол );
```

Угол задается в градусах и может принимать значения от 90° до +90° в зависимости от модели серводвигателя.

Схема подключения серводвигателя показана на Рис. 7.5. В данном случае тоже лучше использовать отдельное питание для привода.

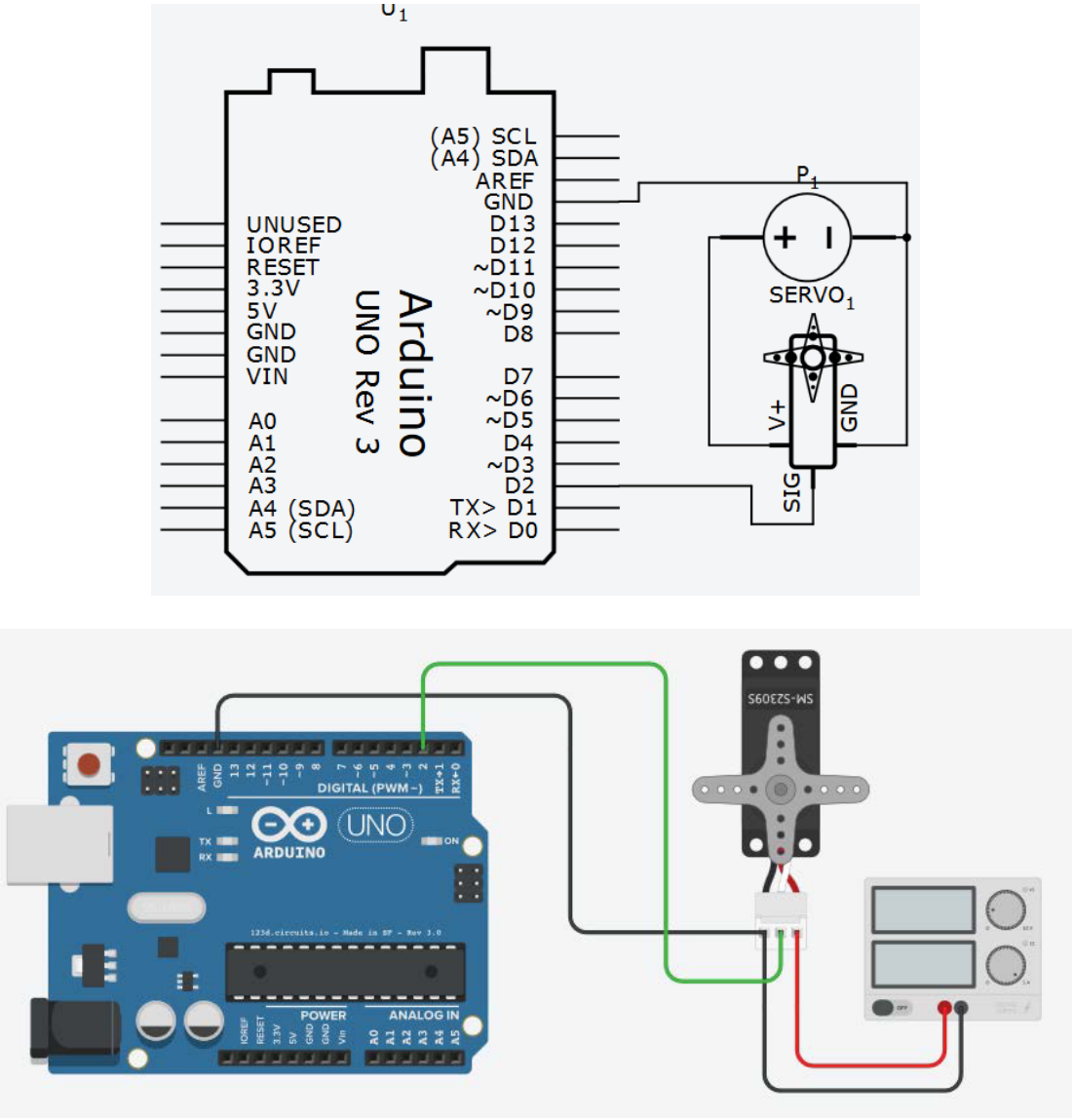


Рис. 7.5: Пример подключения серводвигателя

Далее приведен пример программы, демонстрирующей настройку библиотеки Servo и поворот оси привода на 90 градусов:

```
#include <Servo.h>

Servo myservo;
int servo_pwm = 2;

void setup() {
     // сообщаем, куда подключен привод
     myservo.attach( servo_pwm );
     // поворачиваем вал привода на +90 градусов
     myservo.write( 90 );
}

void loop() {
}
```

Задание 7.6

Реализуйте программу, обеспечивающую плавный поворот серводвигателя между крайними положениями за 5 секунд.

Задание 7.7

Соберите схему с серводвигателем и датчиком температуры. Реализуйте стрелочный термометр – серводвигатель должен занимать положение, задаваемое показанием термодатчика.

Тема 8

I²C и библиотека Wire

8.1 Шина I²C

I²C это сокращение от «Inter-Integrated Circuit». В далеких 1970-х годах подразделение по производству полупроводниковых компонентов фирмы Philips (теперь это фирма NXP) поняла необходимость разработки и стандартизации простого протокола для передачи данных между микросхемами. В результате они разработали протокол, который сейчас называют I²C или TWI (Two-Wire Interface). Этот протокол позволяет обмениваться информацией, используя только два провода, обозначаемые SDA и SCL (Рис. 8.1). По первому передаются данные, а второй используется для сигнала синхронизации.

Шина I²C (Рис. 8.1) позволяет соединять сразу несколько устройств. В сети должно быть хотя бы одно *ведущее устройство (Master)*, которое инициализирует передачу данных и генерирует сигналы синхронизации. Другие устройства являются *ведомыми (Slave),* они передают данные по запросу ведущего. У каждого ведомого устройства есть уникальный адрес, по которому ведущий и обращается к нему. Адрес устройства указывается в документации. К одной шине I²C может быть подключено до 127 устройств, в том числе несколько ведущих. К шине можно подключать устройства

в процессе работы, она поддерживает так называемое «горячее подключение».

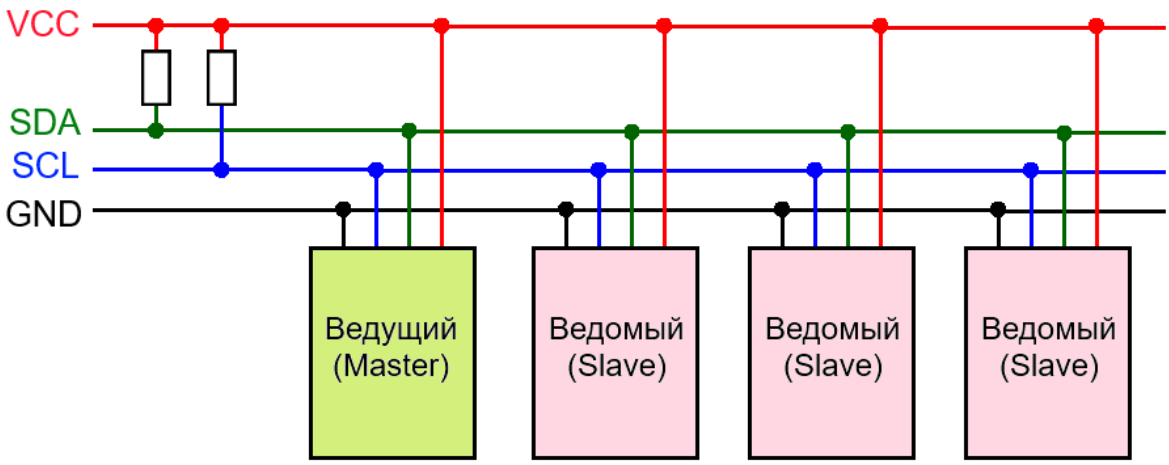


Рис. 8.1: Шина I²C

Как видно из Рис. 8.1, линии SCL и SDA должны быть соединены с питанием VCC через резисторы. Это необходимо для нормальной работы I²C. На платах Arduino (точнее, на установленных в них микроконтроллерах) такие резисторы уже присутствуют, поэтому у нас не будет необходимости добавлять их в схему.

Arduino использует для работы по интерфейсу I²C два порта, которые могут находиться на разных платах в разных местах. Наиболее распространен вариант, когда аналоговый порт A4 соответствует SDA, аналоговый порт A5 соответствует SCL. Такая схема используется, например, в Arduino UNO и Arduino Nano. Для других моделей расположение этих портов показано в Таблице 8.1.

Таблица 8.1: Расположение портов SDA и SCL

Плата	Пин SDA	Пин SCL
Arduino Uno, Nano, Pro и Pro Mini	A4	A5
Arduino Mega	20	21
Arduino Leonardo	2	3
Arduino Due	20, SDA1	21, SCL1

Для работы с протоколом I²C у Arduino есть штатная библиотека `Wire`, позволяющая взаимодействовать с I²C/TWI-устройствами как в режиме мастера, так и в режиме ведомого. Для инициализации библиотеки используется функция `begin`, которая может вызываться в двух вариантах:

```
Wire.begin();
```

или

```
Wire.begin( адрес );
```

Здесь `адрес` – цифровой адрес ведомого устройства, если Arduino подключается к шине в роли ведомого. Если адрес не указывать, то Arduino будет работать как мастер.

Для того чтобы мастер мог передать ведомому данные, он должен вызвать три функции. Сначала он вызывает функцию

```
beginTransmission(адрес);
```

аргументом которой является номер ведомого. Затем он передает сами данные:

```
write(значение);
```

Эту функцию можно вызывать несколько раз. В конце передачи мастер вызывает функцию `endTransmission`, в момент вызова которой как раз и происходит реальная отправка всех данных.

Например, чтобы отправить устройству с адресом 55 байт `val`, надо выполнить следующий код:

```
Wire.beginTransmission(55); // передача для устройства 55
Wire.write(val);            // отправка байта val
Wire.endTransmission();     // завершение передачи
```

Мастер может запросить ведомого передать ему данные. Для этого мастер сначала вызывает функцию

```
requestFrom(адрес, размер);
```

которой передает `адрес` ведомого и число байт, которые ожидает принять. После этого мастер может получить по очереди все байты ответа, вызывая функцию `read`. Функция `available` возвращает число байт, которые доступны для чтения.

Таким образом, обычный код чтения данных от ведомого выглядит следующим образом:

```
// запрашиваем 6 байт от ведомого #55
Wire.requestFrom(55, 6);
// ведомый может прислать меньше, чем просили
while (Wire.available()) {
      // получаем один байт как символ
      char c = Wire.read();
      // отправляем принятый байт на ПК
      Serial.print(c);
}
```

Работа с библиотекой `Wire` в режиме ведомого происходит другим образом. Мастер знает, когда он хочет что-то передать или получить и он может просто вызвать необходимые функции. Однако ведомый заранее не знает, в какой момент он получит данные или запрос от мастера.

Поэтому при написании программы для ведомого необходимо сначала написать функцию, которая будет реагировать на действия мастера, а потом зарегистрировать ее в библиотеке `Wire`. Библиотека сама вызовет нашу функцию в тот момент, когда зафиксирует действия мастера.

Таких функций может быть две. Одна используется для приема данных (если он передал их с помощью `beginTransmission`/ `send`/ `endTransmission`), а вторая – если мастер запросил данные с помощью `requestFrom`. Такие функции называются функциями-обработчиками.

Рассмотрим сначала первый вариант. Первым делом надо написать функцию, которая будет принимать данные. Эта функция может иметь любое имя, но она должна получать один аргумент – число байт, передаваемых мастером, и не должна возвращать никаких значений. Внутри нее можно читать данные с помощью функций `available` и `read`. Например:

```
void receiveEvent(int howMany) { // имя может быть другим
  while (Wire.available()) { // пока есть данные
      char c = Wire.read(); // читаем один байт
      Serial.print(c); // передаем его на компьютер
  }
}
```

Обработчик следует зарегистрировать в функции `setup` программы с помощью `onRecieve` следующим образом:

```
// надо указать имя обработчика
Wire.onReceive(receiveEvent);
```

Второй вариант, когда ведомый должен отправить данные по запросу мастера, реализуется похожим образом. Сначала нужно написать обработчик, который в данном случае будет иметь пустой список параметров. Внутри нее достаточно просто вызвать функцию `send` для отправки данных. Желательно отправить столько байт, сколько от нас ожидает мастер:

```
void requestEvent() {
  Wire.write("data12"); // ответ длиной 6 байт
}
```

Регистрируется такой обработчик запроса с помощью функции `onRequest`:

```
Wire.onRequest(requestEvent);
```

В настоящее время выпускается множество устройств, работающих по шине I²C. Это могут быть цифровые датчики температуры, микросхемы памяти, электронные компасы, жидкокристаллические дисплеи, микросхемы-радиоприемники и многое другое. Все эти компоненты можно подключить к микроконтроллеру, используя всего два вывода, что очень удобно. Для многих из них есть библиотеки для Arduino, которые сделают их использование еще более легким, чем работа через библиотеку `Wire`.

8.2 Пример использования I²C в Autodesk CIRCUITS

К сожалению, в эмуляторе Autodesk CIRCUITS нет моделей для I²C устройств. Поэтому в нашем учебном пособии мы будем использовать эту шину для организации связи между двумя платами Arduino. Это тоже очень интересная и полезная возможность, которая позволит продемонстрировать работу как в режиме мастера, так и в режиме ведомого.

Мы будем использовать простую схему, показанную на Рис. 8.2. В этой схеме есть две платы Arduino, к которым подключено по

светодиоду. К одной из плат, выступающей в роли мастера, подключена кнопка. Обе платы соединены шиной I²C через выводы A4 и A5. Земля питания плат также объединена. Так как в платах Arduino используются встроенные резисторы для шины I²C, то нет необходимости соединять линии шины с питанием отдельными резисторами.

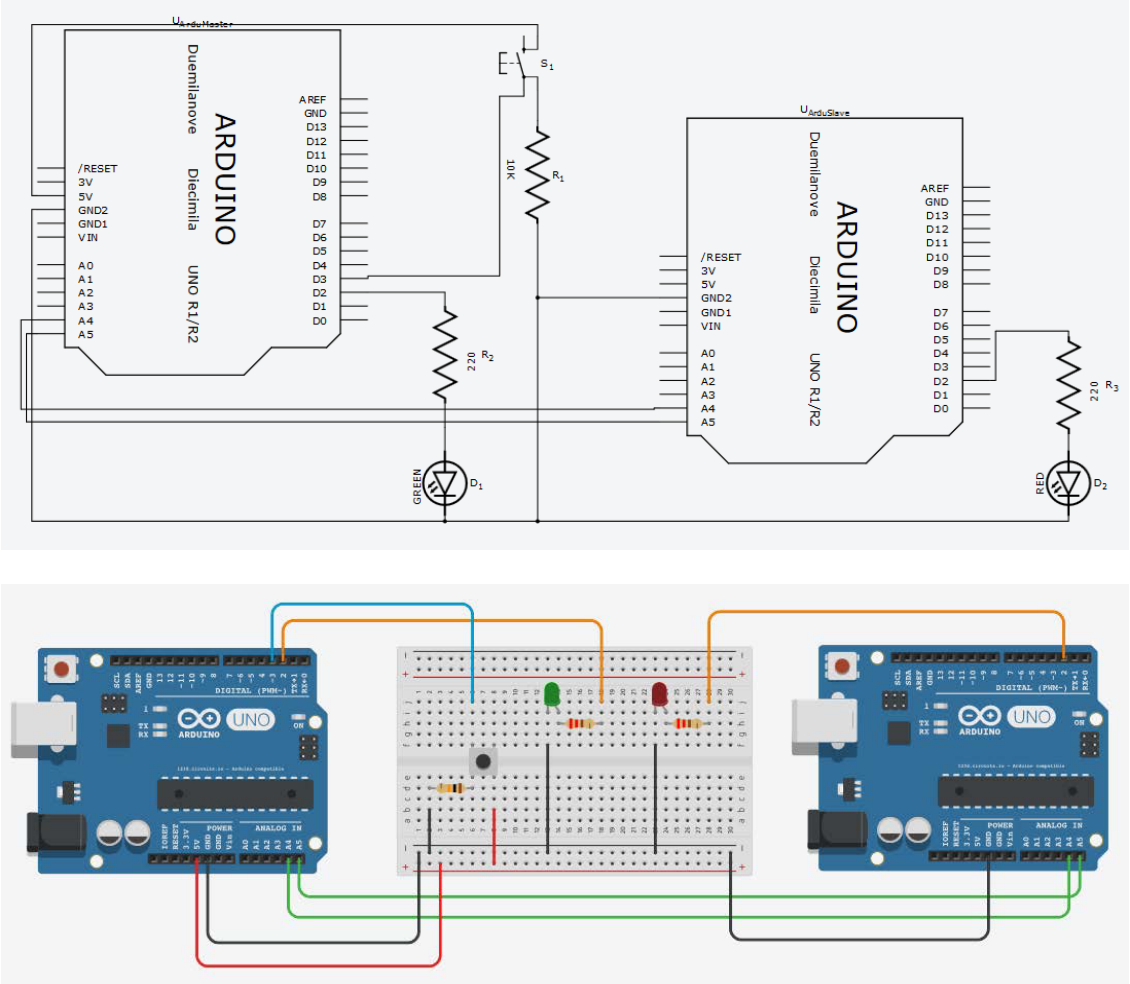


Рис. 8.2: Пример подключения шины i²C

При нажатии на кнопку мастер будет зажигать свой светодиод, и отправлять команду, состоящую из одного символа «H» ведомому. При отпускании кнопки мастер выключает светодиод и отправляет команду «L». Ведомый будет зажигать и выключать свой светодиод по командам от мастера.

При программировании двух плат в среде Autodesk CIRCUITS нужно выбирать плату, для которой пишется программа, с помощью

выпадающего списка, находящегося в левой верхней части редактора кода (см. Рис. 8.3).

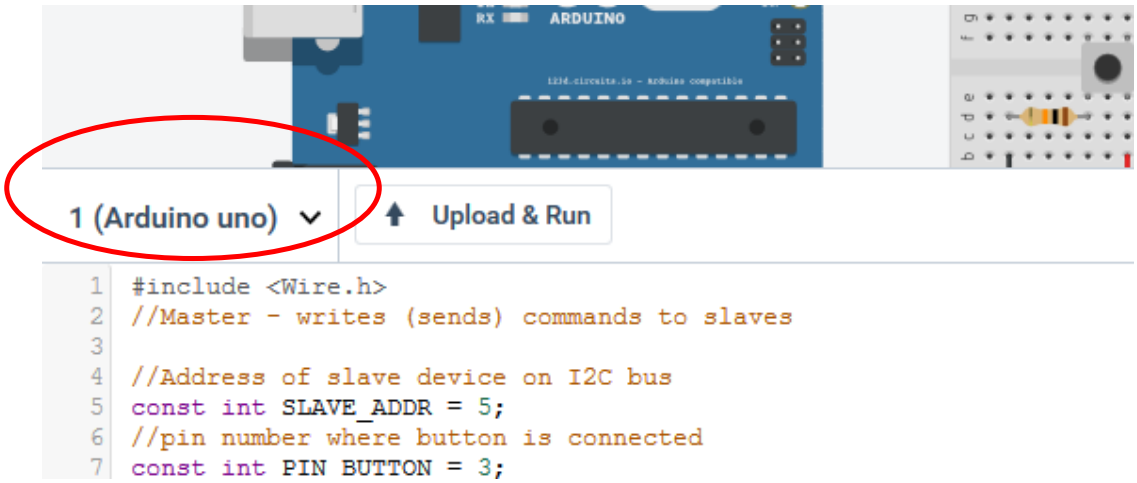


Рис. 8.3: Выбор платы в среде Autodesk CIRCUITS

Рассмотрим код для мастера:

```
#include <Wire.h>

const int SLAVE_ADDR = 5;   // адрес ведомого на шине I2C
const int PIN_BUTTON = 3;   // вывод с кнопкой
const int PIN_LED = 2;      // вывод со светодиодом
const char BTN_PRESSED = 'H';    // сообщение включить диод
const char BTN_RELEASED = 'L';  // сообщение выключить диод
int buttonState = 0;        // состояние кнопки

void setup() {
  Wire.begin(); // эта плата настраивается мастером
  pinMode(PIN_BUTTON,INPUT); // настройка вывода для кнопки
  pinMode(PIN_LED, OUTPUT); // настройка вывода для диода
}

void loop() {
  // читаем состояние кнопки
  buttonState = digitalRead(PIN_BUTTON);

  if (buttonState == HIGH) {
      // включаем свой диод
```

```
      digitalWrite(PIN_LED, HIGH);
      // посылаем команду ведомому
      Wire.beginTransmission(SLAVE_ADDR);
      Wire.write(BTN_PRESSED);
      Wire.endTransmission();
  } else {
      // выключаем диод
      digitalWrite(PIN_LED, LOW);
      // посылаем команду ведомому
      Wire.beginTransmission(SLAVE_ADDR);
      Wire.write(BTN_RELEASED);
      Wire.endTransmission();
  }
  delay(30);
}
```

Теперь рассмотрим код ведомого:

```
#include <Wire.h>

const int LED = 2;          // вывод со светодиодом
const int SLAVE_ADDR = 5;   // адрес ведомого на шине I2C

void setup() {
  Wire.begin(SLAVE_ADDR);        // эта плата будет ведомым
  Wire.onReceive(receiveEvent); // регистрация обработчика
  pinMode(LED, OUTPUT);     // настраиваем вывод светодиода
  digitalWrite(LED, LOW);
}

void loop() {
  // Здесь ничего не происходит.
  // Вся работа в обработчике дальше.
}

// Эта функция вызывается, когда приходят данные по I2C
void receiveEvent(int howMany) {
  // цикл чтения данных
```

```
  while(Wire.available()) {
      char c = Wire.read();  // читаем один байт
      if( c == 'H' )         // разбираем команду
          digitalWrite(LED, HIGH); // включаем диод
      else if ( c == 'L' )
          digitalWrite(LED, LOW);  // выключаем диод
  }
}
```

Задание 8.1

Соберите схему, в которой две платы Arduino с тремя светодиодами (красный, желтый и зеленый) были соединены шиной I²C. Реализуйте работу в режиме светофора – одна плата должна показывать сигналы для одной дороги, а другая – для перпендикулярной (если на одной плате красный, на второй – зеленый и наоборот). Реализуйте управление одного светофора другим через отправку команд по шине I²C. Обмен должен производится в режиме «мастер пишет, ведомый читает».

Задание 8.2

Соберите схему из двух плат Arduino. К одной плате подключен жидкокристаллический экран, а ко второй – датчик температуры. Платы соединены по шине I²C. Плата с экраном будет выступать в роли мастера, который запрашивает данные (температуру) у ведомого и отображает ее на экране.

Тема 9

Среда моделирования V-REP

9.1 Моделирование роботизированных систем

Среда Autodesk CIRCUITS позволяет моделировать работу электронных схем и микроконтроллеров. С ее помощью можно разрабатывать и отлаживать различные электронные схемы, а также программировать логику работы некоторых ее компонентов, как например, микроконтроллеров Arduino. Но на этом ее возможности заканчиваются. В ней можно разработать, например, электронику для робота, но нельзя проверить, как этот робот будет ездить или двигать манипулятором. В этой части учебного пособия мы познакомимся с системой, которая позволяет осуществлять моделирование реального физического мира. Мы займемся уже не реализацией низкоуровневых задач вроде «как сделать так, чтобы этот мотор крутился», а управлением на более высоком уровне – «как сделать робота, который движется, уклоняется от препятствий, следует заданной цели».

Сегодня возможности использования робототехники огромны. Роботы используются везде: начиная с изучения планет Солнечной системы и заканчивая уборкой помещений. Объединение в роботах трех подсистем: актуатора, сенсора и блока управления делает их эффективными в реальном мире, но усложняет виртуальную

симуляцию. Одним из инструментов по моделированию роботизированных систем является бесплатный программный комплекс V-REP. Наравне с традиционными подходами к моделированию, которые есть и в других тренажерах, V-REP добавляет несколько дополнительных. Данный тренажер обладает технологией встроенных скриптов, которые заменяют различные типы контроллеров в имитационной модели, что позволяет делать эти модели чрезвычайно портативными и масштабируемыми.

Встроенные скрипты представляют собой наиболее мощную отличительную особенность V-REP. Они делятся на основной и дочерние скрипты. Основной цикл («основной сценарий») моделирования является Lua-скриптом, который решает общие функциональные задачи. Например, он вызывает разные подсистемы для обработки кинематики или динамики объектов моделируемого мира. Основной сценарий также отвечает за вызов дочерних скриптов каскадным способом.

Дочерние скрипты, в отличие от основного, прикрепляются к конкретному объекту или конкретной части моделирования в процессе цикла моделирования. Они являются неотъемлемой частью сценария объекта, и будут повторяться вместе с ним. Как таковой дочерний скрипт представляет собой портативный и масштабируемый элемент управления: в нем есть один единый файл, содержащий определение модели вместе с ее функционалом, нет проблемы совместимости на разных платформах, нет необходимости в явной компиляции, никакого конфликта между несколькими версиями одной и той же модели и др. Дочерние скрипты могут быть запущены в потоковой или не потоковой реализации.

9.2 Интерфейс программы V-REP

Программа V-REP состоит из нескольких составных частей:

– *окно консоли*: во время запуска приложения появляется окно консоли, но во время моделирования оно будет спрятано. При необходимости можно изменить эти настройки, вызвав User settings (настройки пользователя) с помощью кнопки  . В этом окне отображаются загружаемые плагины и их процедуры

инициализации. Его можно использовать только для вывода информации (например, с помощью команд языка Си `printf` или `std::cout` с использованием соответствующего плагина). Более подробно про плагины можно узнать, пройдя по следующей ссылке: http://www.coppeliarobotics.com/helpFiles/en/pluginTutorial.htm;

– *окно приложения* используется для того, чтобы отображать, редактировать и моделировать;
– *диалоговые окна* являются промежуточным звеном между пользователем, сценой и объектами сцены и оказывают помощь во взаимодействии между ними.

На Рис. 9.1 можно видеть классический вид приложения V-REP:

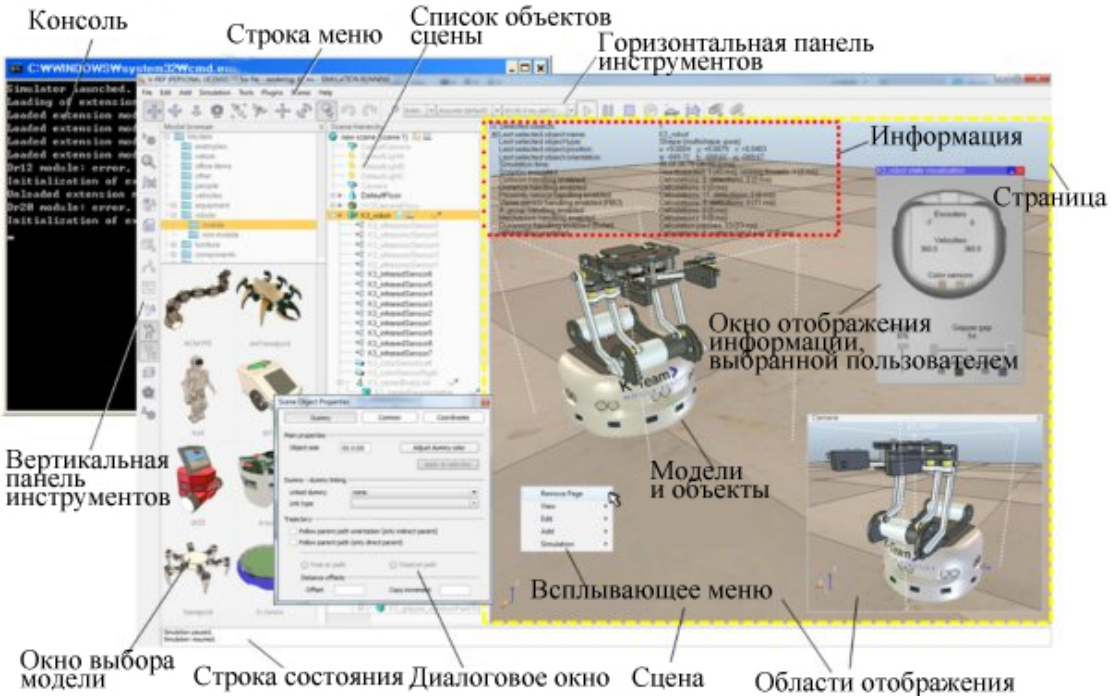


Рис. 9.1: Окно приложения V-REP

Составные части окна приложения V-REP:

1. Строка меню.

2. Панели инструментов – используются для доступа к наиболее часто используемым функциям программы. На Рис. 9.2 изображены кнопки, расположенные на панели инструментов, и их краткое описание.

3. Окно выбора модели – является видимым по умолчанию, но может быть скрыто с помощью соответствующей кнопки. В верхней

части окна отображается структура папок, а в нижней – всевозможные виды моделей (содержащихся в выбранной папке). Любую из имеющихся моделей можно при желании поместить на сцену.

4. Список объектов сцены – отображает древовидную структуру объектов, в которую можно добавлять элементы или удалять. Чтобы открыть окно свойств любого объекта, достаточно дважды щелкнуть по его значку, для присвоения объекту другого имени – дважды щелкнуть по имени. Перетаскивая один объект к другому, можно установить отношения связи между ними (сделать одного из них «родителем»).

5. Страница. Каждая сцена может содержать до 8 страниц, которые, в свою очередь, могут включать бесконечное число областей отображения.

6. Области отображения – используются для показа объектов и окружающих их предметов, расположенных на сцене, с помощью камер, графиков или видеодатчиков.

7. Информация – приводится для конкретных объектов или предметов, также показывает состояния и параметры моделирования. Внешний вид может быть изменен с помощью кнопок в верхнем левом углу поля.

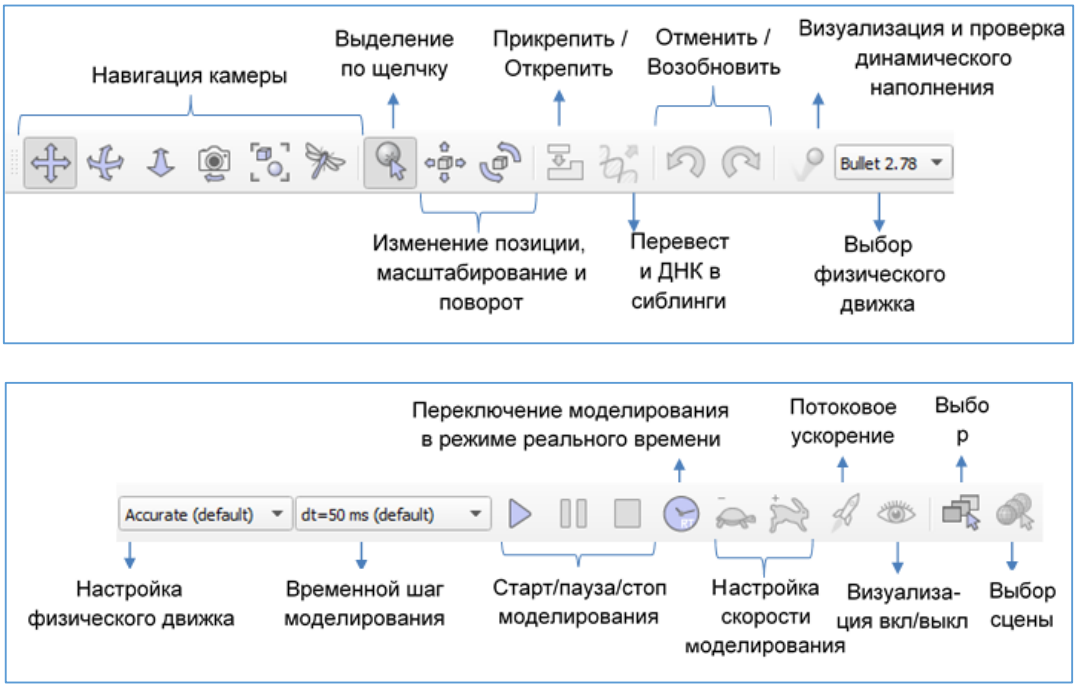


Рис. 9.2: Меню программы V-REP

8. Окно отображения информации, выбранной пользователем – настраиваемое пользователем окно, отражающее необходимую информацию или диалог с пользователем.

9. Всплывающее меню (Рис. 9.3) – появляется при щелчке правой кнопкой мыши.

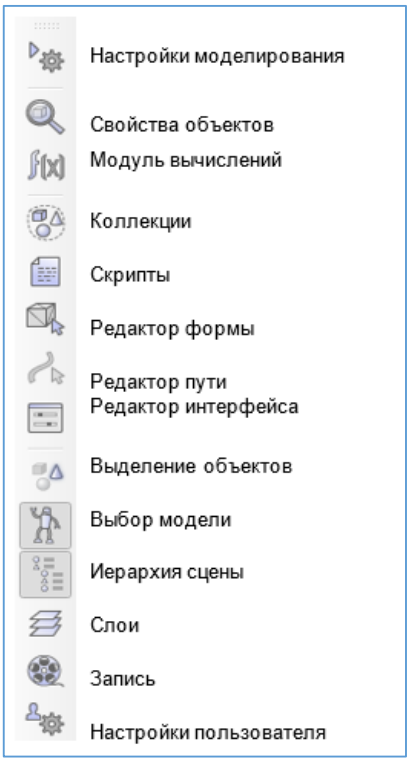


Рис. 9.3: Контекстное меню программы V-REP

Вопросы для самоконтроля

1) Как называется область программы, где можно установить связь родитель-дети между объектами сцены?
   a) Области отображения (Views).
   b) Список объектов сцены (Scene hierarchy).
   c) Окно выбора модели (Model browser).
   d) Консоль (Console window).
2) Скорость моделирования управляется с помощью кнопок:
   a) Стрелками на горизонтальной панели инструментов.
   b) Значком кинопленки на вертикальной панели инструментов.

c) Зайчиком и черепахой на горизонтальной панели инструментов.

3) Какие по счету (слева-направо) из кнопок панели инструментов на Рис. 9.4 отвечают за положение объектов?

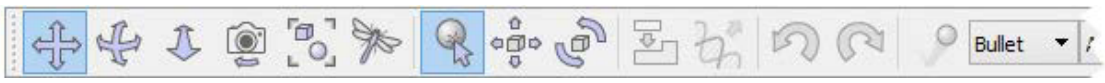


Рис. 9.4: Панель инструментов

9.3 Создание простого робота в среде V-REP

В качестве первого упражнения мы попробуем спроектировать простого мобильного робота ШаБота, используя возможности программы V-REP. ШаБот – это сокращение от двух слов – «робот» и «шар», так как наш робот будет иметь шаровидную форму.

Начать стоит с тела нашего робота. Добавляем примитив шар диаметром 0.2 к сцене [Add --> Primitive shape --> Sphere, X-size = 0.2] и нажимаем <OK>. Созданный шарик появится в видимом слое 1 по умолчанию и будет подвижным и взаимодействующим, то есть способным реагировать на столкновения с другими предметами (обладающими свойством взаимодействия) при помощи физического движка. Если требуется изменить свойства объекта, можно его выделить и нажать на кнопку  на панели инструментов «Scene Object Properties», в появившемся окне выбрать «Show dynamic properties dialog» (свойства «Body is respondable» и «Body is dynamic» включены).

Начинаем моделирование (с помощью кнопки с треугольником на панели сверху справа или нажав <CTRL>+<пробел> в окне сцены), копируем и вставляем созданный шарик [Edit --> Copy selected objects] и [Edit -> Paste buffer] (или <CTRL>+<C>, <CTRL>+<V>): в итоге два шарика столкнутся и откатятся друг от друга. Остановите моделирование (кнопка с квадратом) и скопированный шар автоматически удалится. С помощью кнопки Simulation settings  можно управлять настройками моделирования.

Также потребуется, чтобы тело нашего ШаБота можно было обрабатывать с помощью модулей расчетов (например, minimum

distance calculation module). Доступ к нему возможен через кнопку Calculation module properties  .

Проверьте, что у объекта шар были подключены такие свойства, как «Collidable», «Measurable», «Renderable» и «Detectable» (кнопка  , вкладка «Common»). Поставьте галочки, если свойства не подключены. Во вкладке «Shape» можно изменить внешний вид шарика.

Далее открываем окно «Object/item position/orientation» с помощью кнопки  (вкладка «Position/Translations»), в поле «Object/item translation & position scaling operations» вводим значение 0.02 для «Along Z(m)» и проверяем, что в поле «Relative to» выбрано «World». Кликаем по «Translate selection». Наши действия должны привести к тому, что все выделенные объекты передвинутся на 2 см вдоль абсолютной оси Z и шарик немного поднимется.

В окне «Scene hierarchy» дважды кликните по названию шарика и дайте ему имя. Например, *bubbleRob,* и нажмите <Enter>.

На следующем шаге мы будем добавлять датчик близости (proximity sensor), чтобы наш ШаБот знал о приближении к препятствиям: [Add --> Proximity sensor --> Cone type]. Вызовите окно «Object/item position/orientation»  (вкладка «Orientation/ Rotations»), в разделе «Object/item rotation operations» введите 90 в полях «Around Y(m)» и «Around Z(m)», кликните «Rotate selection». Далее во вкладке «Position/Translations» в поле «Object/item position» введите 0.1 в поле «X-coord» и 0.12 в поле «Z-coord». Теперь датчик близости корректно прикреплен к нашему ШаБоту.

Дважды кликните по значку датчика в окне «Scene hierarchy» и откройте его свойства. Кликните «Show volume parameter», откроется окно «Detection Volume Properties». Введите в поле «Offset» значение 0.005, «Angle» – 30 и «Range» – 0.15. Вернитесь к окну свойств сенсора и выберите «Show detection parameters». В открывшемся окне уберите галочку (если она стоит) напротив «Don't allow detections if distance smaller than» и закройте окно. Далее поменяйте имя сенсора на *bubbleRob_sensingNose*, дважды кликнув в окне «Scene hierarchy» по значку сенсора.

Для того, чтобы прикрепить сенсор к роботу, выделите значок сенсора и, зажав <CTRL>, значок робота в окне «Scene hierarchy», далее [Edit --> Make last selected object parent].

В итоге наш ШаБот будет выглядеть следующим образом:

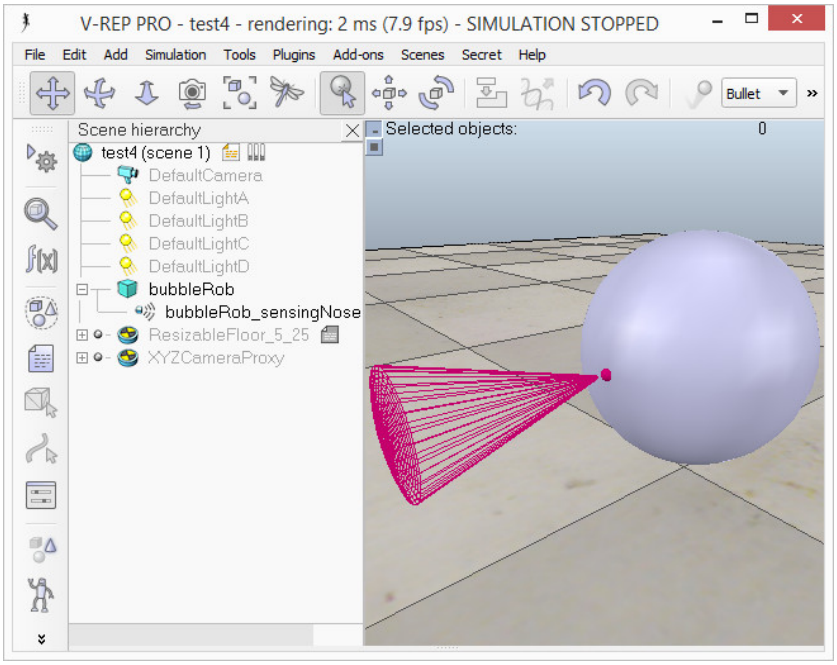


Рис. 9.5: Пример ШаБота

Далее нам следует позаботиться о возможности передвижения ШаБота при помощи колес.

Мы создадим отдельную сцену [File --> New scene]. Работать в нескольких сценах – это достаточно удобно, т.к. можно создавать отдельные элементы и работать над ними.

Добавьте к сцене объект цилиндр [Add --> Primitive shape --> Cylinder] с параметрами измерений [0.08, 0.08 (автоматически пропишется), 0.02]. Аналогично шару ставим галочку у свойств «Collidable», «Measurable», «Renderable» and «Detectable».

В окне «Object/item position/orientation» устанавливаем абсолютную позицию в (0.05, 0.1, 0.04) и абсолютную ориентацию в (-90,0,0). Мы получили левое колесо и надо переименовать объект в *bubbleRob_leftWheel*. Копируем колесо, устанавливаем по абсолютной оси Y значение -0.1 и даем имя *bubbleRob_rightWheel*. Копируем оба колеса, вставляем на первую сцену.

Теперь добавим шарнир (или мотор) к колесам: [Add --> Joint --> Revolute]. Новые объекты появляются всегда в одном месте сцены

и нам потребуется перенести мотор к центру колеса: выделяем мотор и левое колесо (с <CTRL>), заходим в «Position/Translations», секция [Object/item position --> Apply to selection]. Во вкладке «Orientation/Rotations» секции «Object/item orientation» нажимаем «Apply to selection».

Переименуйте мотор в *bubbleRob_leftMotor*. Откройте его свойства (дважды кликнув на значок у мотора в окне «Scene hierarchy»), нажмите кнопку «Show dynamic properties dialog», откроется окно «Joint Dynamic Properties», в нем ставим 2 галочки: «Motor enabled» и «Lock motor when target velocity is zero».

Аналогично добавляем мотор для правого колеса и проделываем с ним те же процедуры, называем *bubbleRob_rightMotor*.

Далее присоединяем левое колесо к левому мотору (выделяем оба объекта, [Edit--> Make last selected object parent]), правое – к правому, и оба мотора – к *bubbleRob*. В итоге получится следующий ШаБот:

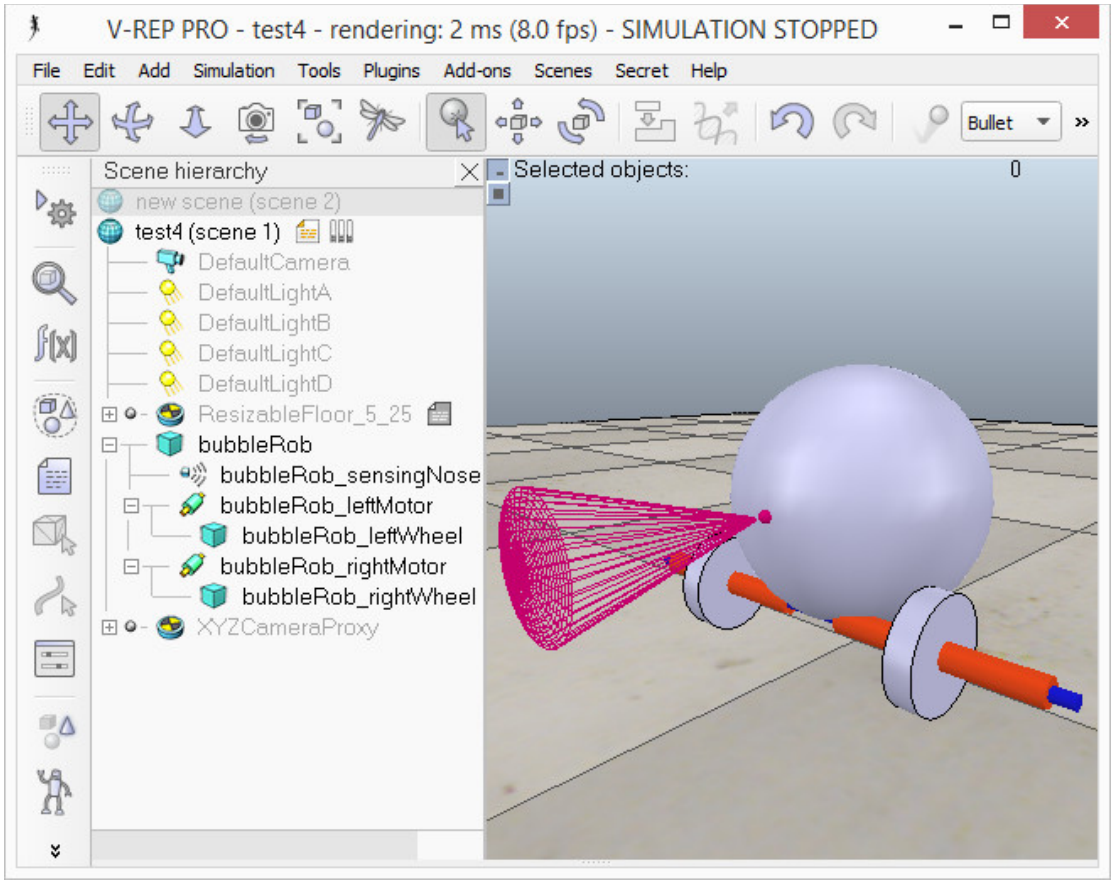


Рис. 9.6: ШаБот с колесами

Запустите моделирование. Робот опрокинулся! Почему? Потому что мы забыли дать ему третью точку опоры на плоскость. Остановите моделирование.

Добавим маленькое поддерживающее колесико сзади. В новой сцене добавьте шар (sphere) с диаметром 0.05 и установите ему свойства «Collidable», «Measurable», «Renderable» и «Detectable», далее переименуйте его в *bubbleRob_slider*. В окне «Scene Object Properties» нажмите «Show dynamic properties dialog» и в появившемся окне в качестве материала (Material) выберите «noFrictionMaterial». Чтобы связать сделанный шарик с роботом, добавляем датчик силы: [Add --> Force sensor]. Переименовываем его в *bubbleRob_connection* и поднимаем его вверх на 0.05 *(«*Object/item position/orientation», вкладка «object/item position», «Z-coord»). Закрепляем шарик за датчиком силы (как это делали с колесами и моторами), копируем оба объекта и вставляем в основной сцене. Далее надо передвинуть датчик силы на -0.07 по оси X и прикрепить к телу робота.

Если Вы запустите моделирование, то заметите, что поддерживающий шарик и тело робота постоянно слегка сталкиваются друг с другом и как бы дрожат. Это происходит потому, что в оба объекта (*bubbleRob_slider* и *bubbleRob)* сталкиваются друг с другом. Чтобы этого избежать, в [Scene Object Properties -> *Show* dynamic properties] (окно «Dynamics properties») для *bubbleRob_slider* укажите 00001111 в качестве «local respondable mask» (уберите галочки там, где должны быть 0), а для *bubbleRob* – 11110000. Перед выполнением этих действий не забудьте остановить моделирование. Должен ШаБот как на Рис. 9.7.

Проверьте, что у Вас получилась такая же иерархия объектов и запустите моделирование. Наш ШаБот почти не движется, ведь его моторы заблокированы. Остановите моделирование.

Стабильность динамических моделей тесно связана с массами и инерцией вовлеченных нестатических объектов. Следует помнить, что массы объекта и соединенного с ним мотора или датчика силы не должны отличаться более, чем в *10 раз*, иначе сенсор или мотор будут работать некорректно.

Попробуем исправить нашего робота. Выберите 2 колеса и поддерживающий шарик, и в окне «Dynamics properties» кликните 3 раза «M=M*2 (for selection)».

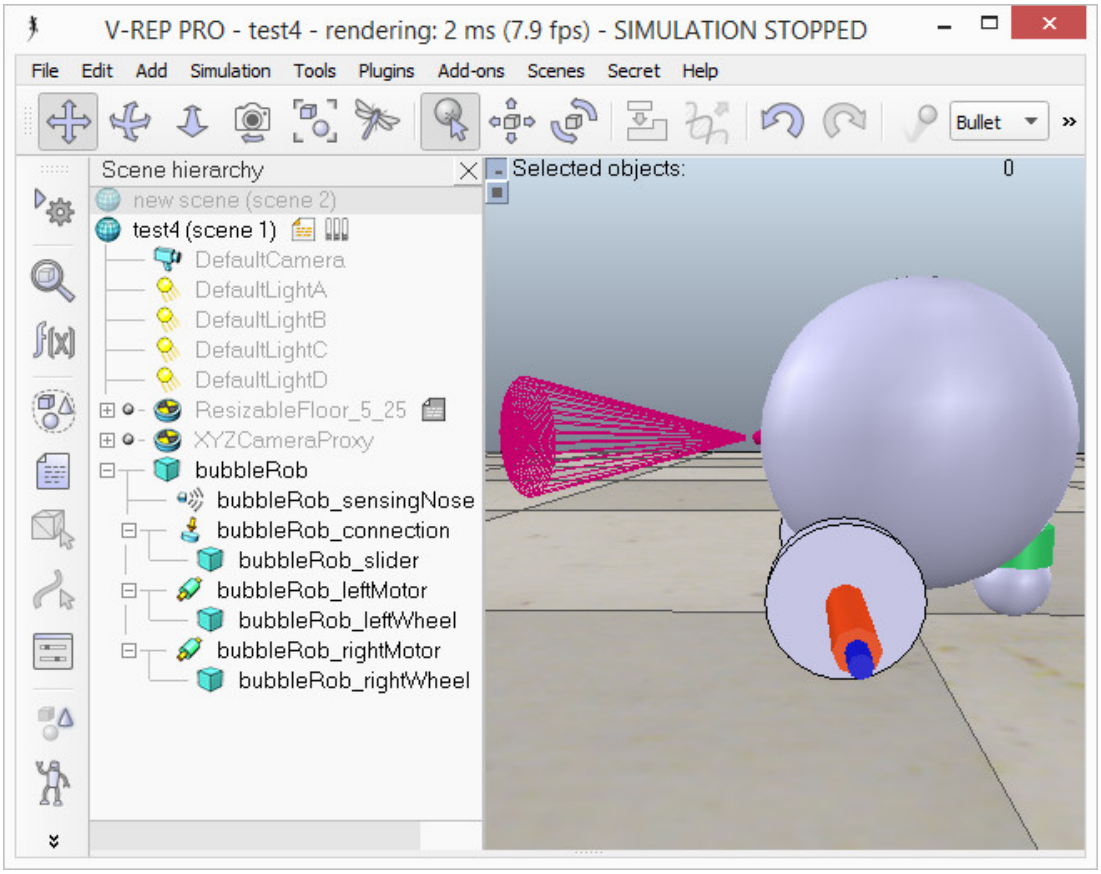


Рис. 9.7: Пример устойчивого ШаБота

В итоге масса каждого из выбранных объектов увеличится в 8 раз. Проделайте те же действия с инерциями этих объектов: «I = I*2 (for selection)». Запустите моделирование: стабильность достигнута.

Для обоих моторов в окне «Dynamics properties» установите «Target velocity» (скорость) равной 50.

Запустите моделирование: теперь ШаБот может двигаться вперед и в итоге упадет в пропасть. Остановите моделирование и верните «Target velocity» значение `0` для обоих моторов.

Объект bubbleRob является базовым для всех объектов, формирующих модель ШаБот. Определим коллекцию объектов, которые представляют ШаБота. Для этого следует открыть окно «Collection»: [Tools --> Collections] (или кликнуть по кнопке ). В появившемся окне выберите «Add new collection». Новая коллекция появится в «Collection list». Выделите ее, в окне «Scene hierarchy» выделите объект *bubbleRob,* в окне коллекций нажмите

«Add». Теперь наша коллекция содержит все объекты дерева иерархии, начинающегося с *bubbleRob*. Для переименования коллекции дважды щелкните по имени и введите *bubbleRob_collection*. Окно коллекции можно закрыть.

Вопросы для самоконтроля

1. Как называется свойство в окне «Dynamic properties», которое отвечает за подвижность объектов и способность испытывать гравитацию?
2. Определите способы установки для объектов связи «родитель-дети»:
   a) Меню [Edit --> Make last selected object parent]
   b) Последовательно щелкнуть по объектам мышью с зажатой клавишей <Alt>.
   c) Перетаскивание в окне «Scene hierarchy» одного объекта на другой.
3. Во сколько раз может отличаться масса объекта и соединенного с ним мотора или датчика силы, чтобы датчик или мотор работали корректно?
   a) в 2 раза,
   b) в 4 раза,
   c) в 8 раз,
   d) в 16 раз.
4. В каком окне можно изменить положение объекта в пространстве?

Задание 9.1

Переместите созданного в данном разделе робота на 1 вправо (относительно самого робота) и на 1 вперед, не изменяя положение по высоте, и поверните на 120 градусов влево вокруг своей оси.

Тема 10

Программирование в среде V-Rep

10.1 Язык Lua

Среда V-Rep поддерживает работу с разными языками программирования. В частности, можно использовать такие языки как C++, Python. Но лучше всего использовать язык Lua, так как именно этот язык является внутренним языком системы V-Rep, и поэтому из него доступно больше всего возможностей по управлению средой, программы, написанные на нем, легче всего интегрировать в систему.

Lua – интерпретируемый язык с динамической типизацией (переменные получают типы в процессе присваивания значений). Интерпретатор языка является свободно распространяемым. Допускается императивный, объектно-ориентированный и функциональный стили написания кода. По возможностям, идеологии и реализации язык ближе всего к JavaScript, однако Lua отличается более мощными и гораздо более гибкими конструкциями.

Считается, что Lua – простой, легкий и понятный язык программирования, который можно интегрировать с программами, написанными на других языках. Убедимся в этом, рассмотрев базовые конструкции языка.

10.2 Основные конструкции

Комментарии

Однострочные комментарии пишутся через 2 дефиса

```
-- my first lua app: hello.lua

--[[
А через 2 дефиса и 2 квадратные скобки пишутся
многострочные комментарии.
--]]
```

Переменные, ветвление, цикл

```
num = 42 -- Все числа с плавающей точкой.
s = 'walternate'  -- Неизменяемые строки как в Python.
t = "double-quotes are also fine"
t = nil -- Уничтожение t; в Lua есть сборщик мусора.

-- Циклы пишутся так (обязательное do/end):
while num < 50 do
   num = num + 1  -- Операторы ++ и -- отсутствуют.
end

-- Пример условного оператора (обязательное then/end):
if num > 40 then
   print('over 40')
elseif s ~= 'walternate' then  -- ~= означает не равно.
   -- Равенство проверяется с помощью ==
   -- Оператор == подходит для сравнения строк.
   io.write('not over 40\n')  -- Стандартный вывод
else
   --Все переменные по умолчанию глобальные.
   thisIsGlobal = 5  -- Предпочтителен горбатый стиль
```

```
   -- Способ сделать локальную переменную:
   local line = io.read() --Чтение следующей строки текста

  -- Конкатенация производится с помощью оператора ..:
  print('Winter is coming, ' .. line)
end
```

Неопределенные переменные возвращают `nil`. Следующая строка не вызовет ошибку:

```
foo = anUnknownVariable  -- Теперь foo равно nil.
aBoolValue = false
```

Только `nil` и `false` возвращают `false`; `0` и `''` возвращают `true`!

```
if not aBoolValue then print('twas false') end
```

Аналог операции `a?b:c` из Си:

```
ans = aBoolValue and 'yes' or 'no'  --> 'no'
```

Цикл `for`:

```
karlSum = 0
for i = 1, 100 do  -- Границы включаются (1 и 100).
  karlSum = karlSum + i
end
```

Чтобы двигаться по циклу задом-наперед, можно использовать "`100, 1, -1`" (сначала пишется откуда начинаем, далее – до какого значения и шаг):

```
fredSum = 0
for j = 100, 1, -1 do
   fredSum = fredSum + j
end
```

Другой способ задания циклов:

```
repeat
  print('the way of the future')
  num = num - 1
until num == 0
```

Функции

```
function fib(n)
   if n < 2 then return 1 end
   return fib(n - 2) + fib(n - 1)
end

-- Функция может возвращать сразу много значений:
function bar(a, b, c)
   print(a, b, c)
   return 4, 8, 15, 16, 23, 42
end

x, y = bar('zaphod')  -- Напечатается "zaphod  nil nil".
```

Теперь `x = 4, y = 8`, остальные значения (`15..42`) игнорируются. Функции могут быть локальными и глобальными. При вызове функции с одним параметром скобки не требуются:

```
print 'hello'  -- Сработает без ошибок.
```

Таблицы

Таблицы в Lua являются единственным структурным элементом, они сочетают в себе свойства массива, хэш-таблицы («ключ»-«значение»), структуры, объекта.

Использование таблиц в качестве словарей, по умолчанию ключи являются строками:

```
t = {key1 = 'value1', key2 = false}
```

К ключам можно получить доступ через точку:

```
print(t.key1) -- Напечатает 'value1'.
t.newKey = {} --Добавит новую пару ключ-значение.
t.key2 = nil  -- Удалит key2 из таблицы.
```

Использование не строки в качестве ключа:

```
u = {['@!#'] = 'qbert', [{}] = 1729, [6.28] = 'tau'}
print(u[6.28])  -- Напечатает "tau"
```

Ключ соответствует значению чисел или строк, но идентичен для таблиц:

```
a = u['@!#']  -- Теперь a = 'qbert'.
b = u[{}]  -- Кажется, что значение будет равно 1729, но
оно nil:
```

b = nil из-за того, что поиск завершился неудачно, т.к. используемый ключ не идентичен объекту, в котором хранится исходное значение (из-за использования таблицы).

Для функции с одним оператором скобки не требуются:

```
function h(x) print(x.key1) end
h{key1 = 'Sonmi~451'}  -- Напечатает'Sonmi~451'.

for key, val in pairs(u) do  -- Цикл по таблице.
   print(key, val)
end
```

Использование списков и массивов

Ключи в списках имеют тип int:

```
v = {'value1', 'value2', 1.21, 'gigawatts'}
for i = 1, #v do  -- #v это размер v для списков.
  print(v[i])  -- Индексы начинаются с единицы!
end
```

'list' – это не тип, v – таблица с последовательными целочисленными ключами, обрабатываемыми как список.

Тема 11

Управление роботами

11.1 Ручное управление роботом

В этой части мы научимся моделировать скорость робота с помощью «OpenGl-based custom UI». Чтобы его открыть, используйте кнопку  . Нажмите «Add new user interface» и введите значение 2 для поля «Client y-size», далее <OK>. Посередине окна появится сделанная заготовка для интерфейса. Переименуйте его в *bubbleRob_ui* (Рис. 11.1).

С помощью клавиши <shift> выделите все пустые клетки интерфейса и кликните «Insert merged button», после чего поверх выбранных клеточек появится большая кнопка. В поле справа обратите внимание на значение поля «Button handle» (равное 3) – во время программирования можно обратиться к этой кнопке, используя это число (аналог id). При желании мы можем его поменять.

Далее выберите в качестве типа кнопки «Slider» (поле type). Кнопка станет полем прокрутки. Снимите выделение со всех клеток, выберите bubbleRob в поле «UI is associated with». Последний шаг дает возможность автоматического копирования и сохранения в случае, если *bubbleRob* копируется или сохраняется в качестве

модели. Установите галочку в поле «UI is visible only during simulation» и закройте окно создания интерфейса.1

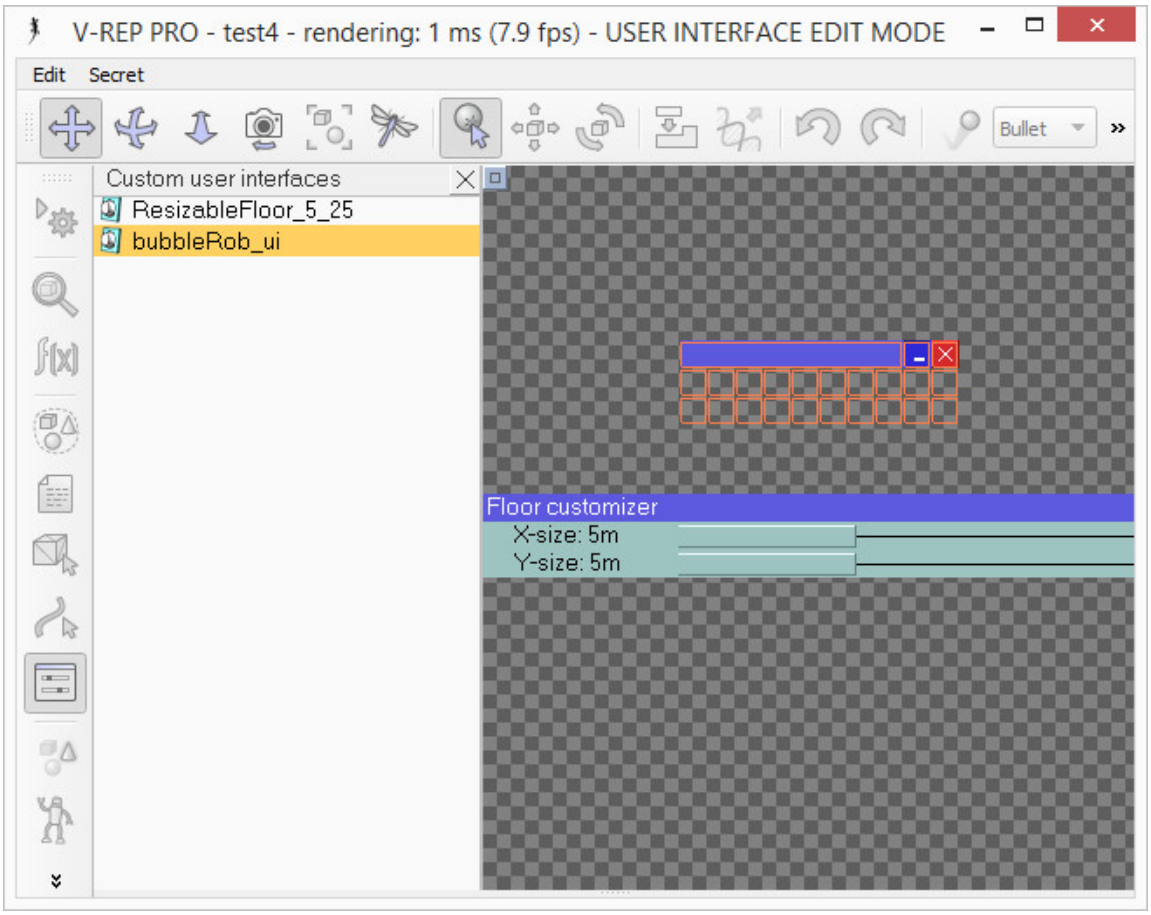


Рис. 11.1: Окно OpenGL based custom UI

Нам также потребуется написать немного кода, который будет контролировать поведение ШаБота. Чтобы добавить простой скрипт, ассоциированный с роботом, выделите *bubbleRob* и щелкните [Add --> Associated child script --> Non threaded]. В данном случае мы делаем не потоковый (Non threaded) скрипт, т.к. он является простым и не будет тормозить моделирование.

Скрипты можно модифицировать, добавлять и удалять, нажав кнопку  или [Tools --> Scripts].

Дважды кликните по появившемуся значку скрипта возле *bubbleRob* в окне «scene hierarchy» – откроется окно с добавленным скриптом. Вставьте в редакторе кода следующий фрагмент и закройте его:

```
-- это условие сработает только в первый запуск скрипта
if (sim_call_type==sim_childscriptcall_initialization) then
   -- дескриптор bubbleRob
   bubbleRobBase =
          simGetObjectAssociatedWithScript(sim_handle_self)
   -- далее дескриптор bubbleRob ассоциируется с созданным
   -- слайдером скорости:
   ctrl = simGetUIHandle("bubbleRob_ui")
   -- прописывается название слайдера:
   simSetUIButtonLabel(ctrl,0,simGetObjectName(
                                 bubbleRobBase) .. " speed")
   -- дескриптор левого мотора
   leftMotor = simGetObjectHandle("bubbleRob_leftMotor")
   -- дескриптор правого мотора
   rightMotor = simGetObjectHandle("bubbleRob_rightMotor")
   -- дескриптор датчика близости
   noseSensor = simGetObjectHandle("bubbleRob_sensingNose")
   -- установка минимальной и максимальной скорости
   -- каждого из моторов
   minMaxSpeed = {50*math.pi/180, 300*math.pi/180}
end
if (sim_call_type == sim_childscriptcall_actuation) then
   -- считывание данных со слайдера скорости:
   speed = minMaxSpeed[1]+(minMaxSpeed[2]-minMaxSpeed[1])*
                                 simGetUISlider(ctrl,3)/1000
   simSetJointTargetVelocity(leftMotor,speed)
   simSetJointTargetVelocity(rightMotor,speed)
end
```

Запустите моделирование. Робот должен ехать быстрее, если переводить ползунок в крайнее правое положение, и медленнее -- если в левое.

Задание 11.1

Поменяйте значение поля «Button handle», указав в нем, к примеру, цифру 5. Каким образом следует теперь изменить код?

Задание 11.2

Добавьте 2 аналогичных слайдера так, чтобы один из них управлял скоростью левого колеса, а другой – правого. В итоге ШаБот научится поворачивать, и им можно будет управлять.

11.2 ШаБот: объезжаем препятствия

В этой части мы научим нашего робота объезжать препятствия. Но сначала решим следующую задачу: как определять минимальное расстояние между ШаБотом и другими объектами? Для этого откройте окно «Distance»: [Tools --> Calculation module properties], либо кнопка  и вкладка «Distance calc». Щелкните «Add new distance object» и выберите «[collection] bubbleRob_collection - all other measurable objects in the scene». Переименуйте «distance» в *bubbleRob_distance* и закройте окно.

Если Вы запустите моделирование, то не увидите никаких отличий, но теперь измеряется расстояние между ШаБотом и объектами. Проблема в том, что у нас нет никаких объектов, расстояние до которых можно измерить – следует добавить препятствия. Но этим мы займемся чуть позже.

Следующим шагом будет добавление графика, чтобы можно было наблюдать расстояние и траекторию движения робота. Для этого щелкаем [Add --> Graph] и сразу переименовываем его в *bubbleRob_graph*. Закрепляем его за bubbleRob и устанавливаем абсолютные координаты в (0,0,0.005).

Открываем свойства графика двойным щелчком по значку в «scene hierarchy». Снимаем галочку «Display XYZ-planes» и кликаем «Add new data stream to record», чтобы создать объекты, записывающие положение робота в координатах по трем осям. В появившемся окне выбираем «Object: absolute x-position» для «Data stream type» и «*bubbleRob_graph*» для «*O*bject / item to record». В списке окна свойств появится новый элемент списка. Повторяем операцию для координат по оси Y и Z. Мы получили 3 потока для записи x-, y-, z-траекторий. Переименовываем их соответственно:

Data в bubbleRob_x_pos, Data0 в bubbleRob_y_pos и Data1 в bubbleRob_z_pos.

Добавим еще один поток – для записи минимального расстояния между роботом и окружающими предметами: щелкаем «Add new data stream to record» и выбираем «Distance: segment length» для «Data stream type» и «*bubbleRob_distance*» для «Object / item to record». Переименуем его в *bubbleRob_obstacle_dist*.

Далее для всех потоков кроме *bubbleRob_obstacle_dist* снимаем флажок «Visible» в секции «Time graph properties», таким образом видимым останется только информация о дистанции. Должно получиться в итоге окно как на Рис. 11.2.

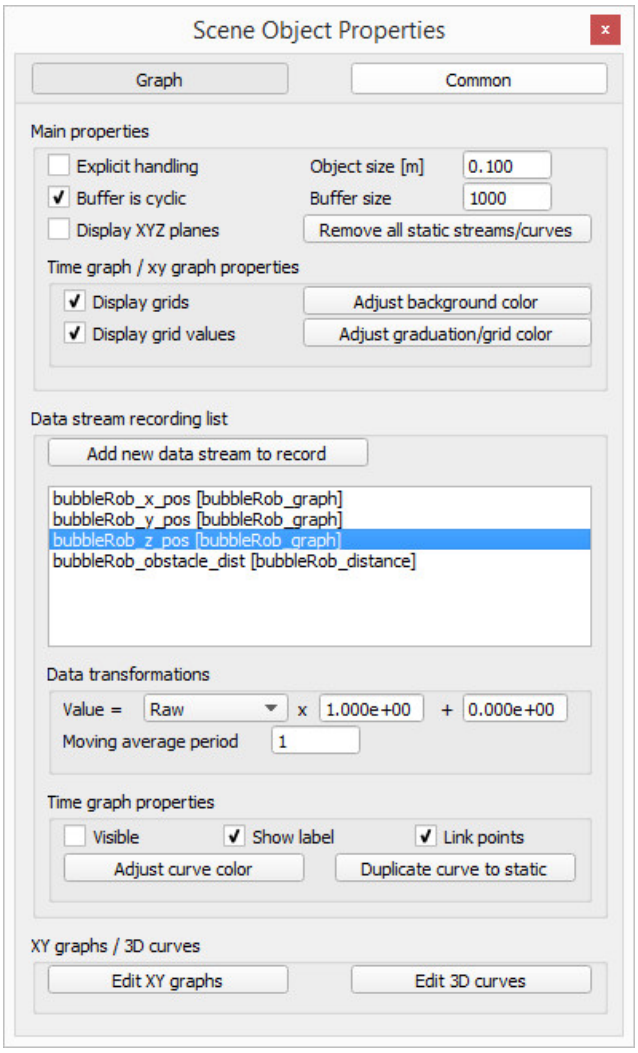


Рис. 11.2: Окно Scene Object Properties

Далее добавим 3D график, отражающий траекторию передвижения ШаБота: щелкаем по «Edit 3D curves», откроется окно «3D curves» (Рис. 11.3), далее «Add new curve». В появившемся окне выбираем «*bubbleRob_x_pos*» для «X-value» и «*bubbleRob_y_pos*» и «*bubbleRob_z_pos*» для «Y-value» и «Z-value», соответственно. Переименовываем *Curve* в *bubbleRob_path*. Выбираем «Relative to world» и устанавливаем значение ширины линии, равное 4, в поле «Curve width». Закрываем все окна, относящиеся к графике.

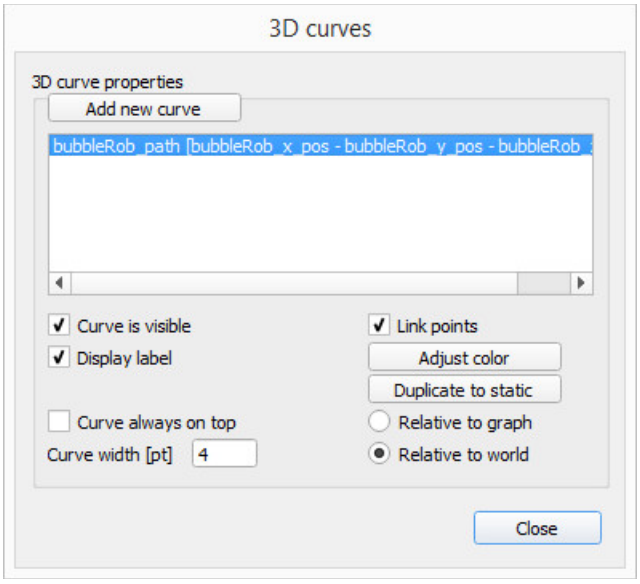


Рис. 11.3: Окно 3D curves

Для одного из моторов установите значение скорости (Target velocity), равное 50, запустите моделирование и пронаблюдайте траекторию ШаБота.

Остановите моделирование и верните скорость мотора, равную нулю.

Далее добавим окошко для отслеживания информации о минимальном расстоянии: правой кнопкой мыши щелкаем по окну сцены [Add --> Floating view]. В окне иерархии выделяем *bubbleRob_graph*, в окне «Floating view» щелкаем правой кнопкой мыши [View --> Associate view with selected graph]. Теперь в этом окне будет отображаться нужная нам информация. Но пока она не отображается... надо добавить препятствия!

Добавим обычный цилиндр с измерениями сторон 0.1, 0.1, 0.2. Чтобы при столкновении он оставался неподвижным и на него не

влияла гравитация, но при этом он оставался действующим препятствием для нестатичных объектов, мы отключим свойство «Body is dynamic» в окне «Dynamic properties». И, как обычно, сделаем его «Collidable», «Measurable», «Renderable» и «Detectable».

Окружим ШаБота стеной из цилиндров. Для этого выделите сделанный цилиндр, нажмите кнопку  , откроется окно. Не закрывая окно, с помощью левой кнопки мыши передвигайте цилиндр. Если одновременно с кнопкой мыши нажать клавишу <Shift>, то цилиндр будет передвигаться более плавно и с меньшим шагом. При необходимости с помощью клавиши <Ctrl> можно поднимать или опускать предметы (с нажатой левой кнопкой мыши). Поставьте столбик на некотором расстоянии от ШаБота. С помощью клавиш <CTRL>+<C> и <CTRL>+<V> создайте несколько копий цилиндра и окружите ими робота. Удобнее всего это делать, наблюдая за сценой сверху (используйте кнопки управления видом на сцену  или страница 6 с помощью кнопки ).

Установите скорость левого мотора равной 50 и запустите моделирование: окно с графиком отражает расстояние до ближайшего препятствия. Остановите моделирование и верните нулевое значение скорости.

Далее следует превратить ШаБота в целостную модель. Кликните по значку объекта *bubbleRob,* во вкладке «Common» отметьте галочками «Object is model base» и «Object/model can transfer or accept DNA» – появится белый пунктир вокруг всех объектов. Выделите 2 мотора, датчик близости и график и во вкладке «Common» поставьте галочку «Don't show as inside model selection», после чего кликните «Apply to selection» – эти объекты теперь не обведены белым пунктиром. Выделите 2 мотора и датчик силы и в этом же окне переставьте галочку, которая стоит в клетках «camera visibility layer», на позицию 10 (считая справа налево сверху вниз) – мы перевели моторы и датчик в скрытый слой (слои 9-16 являются по умолчанию скрытыми). Изменить настройки слоев можно по кнопке  или [Tools -->Layers]).

В заключение выберите датчик близости, 2 колеса, поддерживающее колесико и график и поставьте галочку в поле «Select base of model instead» (свойства объектов, вкладка «Common», «Apply to

selection»): теперь при клике на любой объект ШаБота выделяется весь робот целиком – с ним проще работать, невозможно случайно что-то в нем изменить. Отдельные объекты все еще можно выделять – либо в окне «Scene hierarchy», либо с одновременным нажатием клавиш <Ctrl>+<Shift>. В итоге ШаБот должен выглядеть следующим образом:

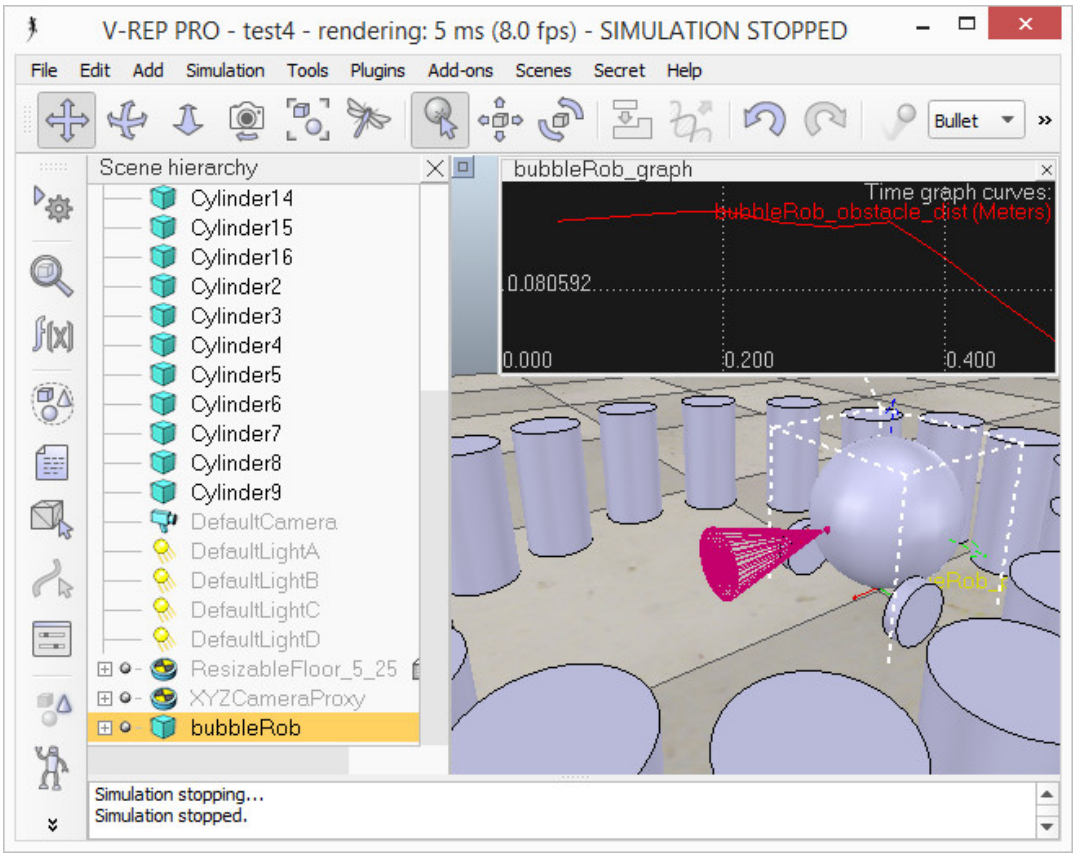


Рис. 11.4: ШаБот в новой среде

Следующим действием добавим видеодатчик на ту же позицию, что и датчик близости. Для этого выберите [Add --> Vision sensor --> Perspective type], прикрепите видеодатчик к датчику близости и установите положение видеодатчка равное (0,0,0) относительно родительского объекта (Parent Frame) в окне «Object/Item position/orientation» (в областях «Object / item position» и «Object / item orientation»). Удостоверьтесь, что сенсор невидим, не выделяется белым пунктиром и при клике по нему выделяется модель целиком (установите соответствующие галочки в свойствах объекта).

Для дальнейшей настройки зайдите в свойства видеодатчика, установите значение 1 в поле, соответствующем «Far clipping plane» (цилиндры должны быть внутри куба «видимости»), а для «Resolution x» и «Resolution y» значение 256. Далее щелкните по кнопке «Show filter dialog». В выпадающем списке выберите «Edge detection on work image» и щелкните «Add filter» (добавить фильтр). Поставьте его на вторую позицию (кнопка со стрелкой вверх). Дважды кликнув по новому фильтру установите «Threshold» (пороговое значение), равное 0.2. Нажмите ОК. Добавьте окошко на сцену (клик правой кнопкой мыши [Add --> Floating view]) и ассоциируйте его с видеодатчиком ([View --> Associate view with selected vision sensor] при выделенном видеодатчике). Запустите моделирование. Теперь мы можем видеть глазами ШаБота! Остановите моделирование.

Научим нашего робота объезжать препятствия. Для этого будем модифицировать скрипт, принадлежащий объекту *bubbleRob*. Скрипты можно модифицировать, добавлять и удалять, нажав кнопку  или [Tools --> Scripts].

Дважды кликните по значку скрипта возле *bubbleRob* в окне «scene hierarchy» – откроется окно с добавленным скриптом. Скопируйте в редактор кода следующий фрагмент (полностью заменив предыдущий код) и закройте окно редактора:

```
-- это условие сработает только в первый запуск скрипта
if (sim_call_type==sim_childscriptcall_initialization) then
   -- дескриптор bubbleRob
   bubbleRobBase =
          simGetObjectAssociatedWithScript(sim_handle_self)
   -- далее дескриптор bubbleRob ассоциируется
   -- с созданным слайдером скорости:
   ctrl = simGetUIHandle("bubbleRob_ui")
   -- прописывается название слайдера:
   simSetUIButtonLabel(ctrl,0,simGetObjectName(
                                  bubbleRobBase).." speed")
   -- дескриптор левого мотора
   leftMotor = simGetObjectHandle("bubbleRob_leftMotor")
   --дескриптор правого мотора
   rightMotor = simGetObjectHandle("bubbleRob_rightMotor")
```

```
   -- дескриптор датчика близости
   noseSensor = simGetObjectHandle("bubbleRob_sensingNose")
   -- установка минимальной и максимальной скорости
   -- каждого из моторов
   minMaxSpeed = {50*math.pi/180, 300*math.pi/180}
   -- для определения направления движения ШаБота
   backUntilTime = -1
end

if (sim_call_type == sim_childscriptcall_actuation) then
    -- считывание данных со слайдера скорости:
   speed = minMaxSpeed[1]+(minMaxSpeed[2]-minMaxSpeed[1])*
                                simGetUISlider(ctrl,3)/1000

   -- берем данные с датчика близости
   result = simReadProximitySensor(noseSensor)
   -- при возникновении проблем, включается обратный ход:
   if (result > 0) then
      backUntilTime = simGetSimulationTime() + 4
   end
   if (backUntilTime < simGetSimulationTime()) then
      -- при движении вперед выбирается скорость
      -- согласно слайдеру
      simSetJointTargetVelocity(leftMotor,speed)
      simSetJointTargetVelocity(rightMotor,speed)
   else
      --[[ если производится движение назад, то скорость
           устанавливается на минимум и робот движется
           по кривой, скорость левого колеса уменьшаем
           в 2 раза и делаем отрицательной (направление
           движения – назад)
      --]]
      simSetJointTargetVelocity(leftMotor, -speed/2)
      – скорость правого колеса уменьшаем в 8 раз
      -- и делаем отрицательной
      simSetJointTargetVelocity(rightMotor, -speed/8)
   end
```

```
end
if (sim_call_type == sim_childscriptcall_sensing) then
end
if (sim_call_type == sim_childscriptcall_cleanup) then
end
```

Запустите моделирование. Теперь ШаБот двигается, объезжая препятствия! Попробуйте поменять скорость робота во время моделирования, скопируйте и вставьте на сцену еще одного робота, поэкспериментируйте.

Возможны подтормаживания моделирования из-за подсчета минимального расстояния до препятствий (его можно убрать, сняв галочку «Enable all distance calculations» в окне «Calculation Modules», вкладка «Distance calc.»).

Вопросы для самоконтроля

1. Какие из пунктов необходимо отметить для того, чтобы выбранный объект мог взаимодействовать с роботами, их сенсорами и окружающими предметами:
   a) Collidable (способность к столкновению)
   b) Renderable (визуализируемость)
   c) Detectable (обнаруживаемость)
   d) Measurable (измеримость)
2. Как называется окно, в котором может отражаться график (при его добавлении)?
   a) Graph view
   b) Data view
   c) Floating view
3. Куда поворачивает ШаБот, построенный в этой главе, при столкновении с препятствием?

Задание 11.3

Скорректируйте код скрипта так, чтобы ШаБот при столкновении с препятствием поворачивал в другую сторону.

11.3 ШаБот: движение по линии

В данном разделе мы постараемся расширить возможности ШаБота и добавить способность следования по проложенному пути (линии).

Загрузите файл со сценой с созданным ранее ШаБотом.

Сперва создадим первый из трех видеодатчиков: [Add --> Vision sensor --> Orthographic type]. В свойствах датчика установите параметры, согласно окну на Рис. 11.5.

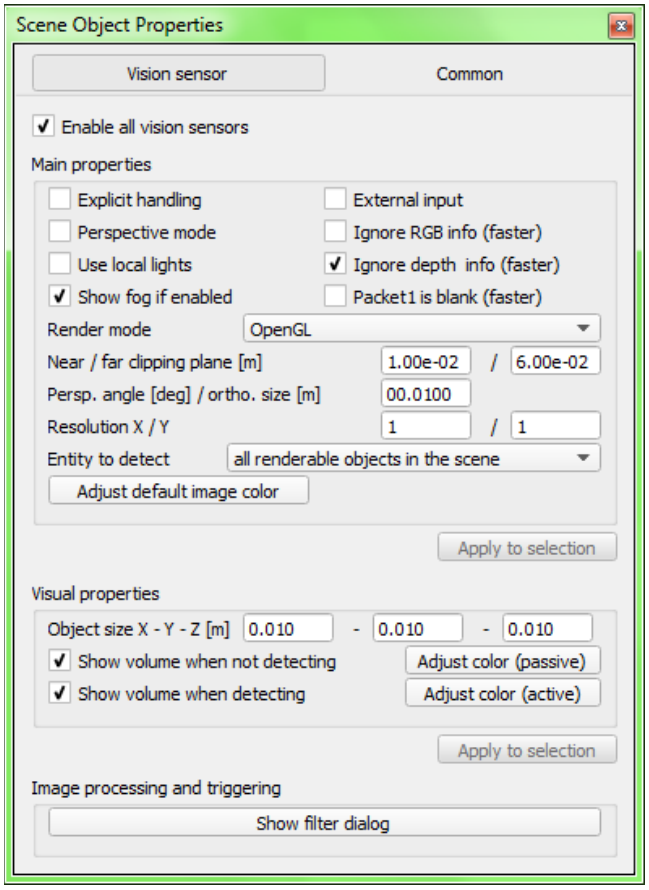


Рис. 11.5: Пример установки параметров

Чтобы видеодатчик находился на полу, в окне изменения положения объекта установите значения [180;0;0] для «*Alpha-Beta-Gamma*».

Существует несколько вариантов чтения информации с видеодатчика. Так как созданный датчик имеет всего 1 пиксел и легко управляем, то можно запрашивать с него информацию

о среднем значении яркости изображения. В более сложных случаях можно пропускать информацию через несколько фильтров.

Теперь скопируем и дважды вставим видеодатчик, переименуем и получим 3 объекта: *leftSensor*, *middleSensor* и *rightSensor*. Установите *bubbleRob* в качестве их родительского объекта.

Подкорректируем расположение датчиков. Для этого в окне изменения положения объектов установите следующие значения (абсолютные координаты):

– левый датчик (leftSensor): [0.2;0.042;0.018];
– средний датчик (middleSensor): [0.2;0;0.018];
– правый датчик (rightSensor): [0.2;-0.042;0.018].

Теперь изменим окружение. Удалите несколько цилиндров-препятствий перед ШаБотом. Сделайте тропинку, по которой будет следовать робот. Чтобы было удобнее, переключитесь на вид сверху (страница 6 с помощью кнопки ). Далее [Add --> Path --> Circle type]. Перейдите к окну изменения положения объекта и выполните одно из предложенных действий:

– при выделенной линии, <Ctrl>+клик мыши на одной из ее контрольных точек – таким образом Вы сможете переносить их в нужные позиции;
– при выделенной линии, используя кнопку  , перейдите в режим редактирования линии и придайте ей желаемую форму.

Избегайте чрезмерно сильных изгибов в линии, она не должна быть извилистой.

Далее в свойствах линии снимите галочки у следующих позиций: «Show orientation of points», «Show path line» и «Show current position on path». И щелкните кнопку «Show path shaping dialog». Установите галочку у поля «Path shaping enabled», в поле «Type» выберите «horizontal segment» и введите 4.0 в поле «Scaling factor». Установите черный цвет линии.

Все вроде бы готово, но осталась проблема: так как и пол, и линия имеют одну и ту же координату по оси Z, то иногда виден пол, иногда линия и может случиться конфликт, когда мы пустим ШаБота по тропинке. Поэтому переместим линию по координате Z на 0.5мм.

Последним шагом будет добавление скрипта к *bubbleRob,* чтобы он знал, как ездить по линии. Для этого открываем скрипт,

расположенный около имени объекта в Scene hierarchy, и заменяем его на следующий фрагмент кода:

```
if (sim_call_type==sim_childscriptcall_initialization) then
   bubbleRobBase =
          simGetObjectAssociatedWithScript(sim_handle_self)
   ctrl = simGetUIHandle("bubbleRob_ui")
   simSetUIButtonLabel(ctrl,0,simGetObjectName(
                                  bubbleRobBase).." speed")
   leftMotor = simGetObjectHandle("bubbleRob_leftMotor")
   rightMotor = simGetObjectHandle("bubbleRob_rightMotor")
   noseSensor = simGetObjectHandle("bubbleRob_sensingNose")
   minMaxSpeed = {50*math.pi/180, 300*math.pi/180}
   backUntilTime = -1
   floorSensorHandles = {-1, -1, -1}
   floorSensorHandles[1]=simGetObjectHandle("leftSensor")
   floorSensorHandles[2]=simGetObjectHandle("middleSensor")
   floorSensorHandles[3]=simGetObjectHandle("rightSensor")
end

if (sim_call_type == sim_childscriptcall_cleanup) then
end

if (sim_call_type == sim_childscriptcall_sensing) then
end

if (sim_call_type == sim_childscriptcall_actuation) then
   speed = minMaxSpeed[1]+(minMaxSpeed[2]-minMaxSpeed[1])*
                                 simGetUISlider(ctrl,3)/1000
   result = simReadProximitySensor(noseSensor)
   if (result > 0) then
      backUntilTime = simGetSimulationTime() + 4
   end

   sensorReading = {false, false, false}
   for i = 1,3,1 do
      result,data=
                  simReadVisionSensor(floorSensorHandles[i])
```

```
      if (result >= 0) then
         -- в data[11] содержится среднее значение
         -- яркости изображения
         sensorReading[i] = (data[11] < 0.3)
      end
      print(sensorReading[i])
   end

   rightV = speed
   leftV = speed

   if sensorReading[1] then
      leftV = 0.03 * speed
   end

   if sensorReading[3] then
      rightV = 0.03 * speed
   end

   if sensorReading[1] and sensorReading[3] then
      backUntilTime = simGetSimulationTime() + 2
   end

   if (backUntilTime < simGetSimulationTime()) then
      simSetJointTargetVelocity(leftMotor,leftV)
      simSetJointTargetVelocity(rightMotor,rightV)
   else
      simSetJointTargetVelocity(leftMotor,-speed/2)
      simSetJointTargetVelocity(rightMotor,-speed/8)
   end
end
```

Для отслеживания работы видеодатчиков добавьте на сцену окно отображения с помощью вызова контекстного меню сцены [Add --> Floating view] и контекстное меню окошка [View --> Associate view with selected vision sensor].

Также следует удалить все лишние датчики из предыдущего кода на ваше усмотрение. Запускайте моделирование!

Задание 11.4

Измените цвет линии на зеленый и поменяйте соответствующим образом код в скрипте для ШаБота, чтобы он распознавал линию и ездил по ней. Подсказка: обратите внимание на переменную, отвечающую за яркость линии.

Вопросы для самоконтроля

1. С помощью какого приспособления роботы могут распознавать линию, по которой им следует двигаться?
   a) Proximity sensor (датчик близости);
   b) Vision sensor (видеодатчик);
   c) Force sensor (датчик силы).
2. Зачем линия, по которой движется робот, приподнимается над «полом»?
   a) Если управление по видео сработает недостаточно быстро, робот сможет зацепиться за линию колесом и это поможет ему удержаться на траектории.
   b) Если линия будет находиться вровень с полом, то ее изображение на датчике робота может перекрываться изображением пола.
   c) Чтобы использовать датчик расстояния для контроля нахождения робота на линии.

Ответы к вопросам для самоконтроля

1.1

1) 1, 3
2) 1

1.2

1) 2, 4, 6, 7
2) ППВМ

1.3

1) 1, 4
2) программатор

1.4

1) 2, 3
2) 2

2.1

1) 1, 3
2) 1, 3

2.2

1) 2, 4, 5
2) 2, 4

2.3

1) 3-2-1
2) 2, 3, 4

3.3

1) 2, 4
2) b = 'c'
3) k = 1
4) 5

3.4

1) 1, 4
2) 1, 3, 5

9.1

1) 2
2) 3
3) 8, 9

9.2

1) Body is dynamic

2) 1, 3
3) 1, 2, 3
4) Object/item position/orientation

11.2

1) 1, 2, 3, 4
2) 3
3) налево

11.3

1) 2
2) 2

Литература

Раздел 1 «Введение»

1. Электроника для всех [Электронный ресурс]. – Режим доступа: http://easyelectronics.ru.
2. Официальный сайт сообщества разработчиков встроенных систем [Электронный ресурс]. – Режим доступа: http://embedders.org.
3. Форум разработчиков встроенных систем [Электронный ресурс]. – Режим доступа: http://caxapa.ru/sitemap.html.
4. Harper K. Honey, I Programmed the Blanket [Электронный ресурс] // The New York Times. – Режим доступа: http://www.nytimes.com/1999/05/27/technology/honey-i-programmed-the-blanket.html
5. Wolf W.H. Hardware-Software Co-Design of Embedded Systems [Электронный ресурс]. – Режим доступа: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=3602704FA35174F9A8FFBDEBC59F7932?doi=10.1.1.135.1147&rep=rep1&type=pdf

Раздел 2 «Моделирование электронных схем и программирование микроконтроллеров на примере Arduino»

6. Мамичев Д.И. Простые роботы своими руками или несерьёзная электроника [Текст]. – М.: Солон-Пресс, 2016. – 144 с.
7. Петин В.А. Проекты с использованием контроллера Arduino [Текст]. – 2-е изд. – СПб.: БХВ-Петербург, 2015. – 464 с.
8. Боксел Д. Изучаем Arduino. 65 проектов своими руками [Текст]. – СПб.: Питер, 2017. – 400 с.
9. Шилдт Г. Полный справочник по C [Текст]. – 4-е изд. – М.: Издательский дом «Вильямс», 2004. – 704 с.
10. Поляков К. Язык программирования Си. [Электронный ресурс]. – Режим доступа: http://kpolyakov.spb.ru/school/c.htm.
11. Arduino.ru [Электронный ресурс]. – Режим доступа: http://www.arduino.ru
12. Уроки для Arduino [Электронный ресурс]. – Режим доступа: https://lesson.iarduino.ru
13. Блог об Arduino, электронике и ИТ [Электронный ресурс]. – Режим доступа: http://soltau.ru/index.php/arduino
14. Robocraft.ru [Электронный ресурс]. URL: http://robocraft.ru.

15. Я презираю Arduino [Электронный ресурс] // Geektimes. – Режим доступа: https://geektimes.ru/post/255760/
16. Официальная документация проекта Arduino [Электронный ресурс]. – Режим доступа: https://www.arduino.cc/en/Guide/HomePage
17. C Tutorial [Электронный ресурс] // Сайт Cprogramming.com. – Режим доступа: http://www.cprogramming.com/tutorial/c-tutorial.html
18. Basic Sensor Interfacing Techniques [Электронный ресурс] // Сайт SensorWiki.org. – Режим доступа: http://www.sensorwiki.org/doku.php/tutorials/basic_sensor_interfacing_techniques
19. Arduino Shield List [Электронный ресурс] // Сайт SparkFun.com. – Режим доступа: https://learn.sparkfun.com/tutorials/arduino shields

Раздел 3 «Моделирование роботов»

20. Мамичев Д.И. Роботы своими руками. Игрушечная электроника [Текст]. – М.: Солон-Пресс, 2015. – 160 с.
21. Конюх В.Л. Основы робототехники [Текст]: учеб. пособие для вузов. – Ростов на Дону: Феникс, 2008. – 281 с.
22. Ловин Д. Создаем робота-андроида своими руками [Текст] / пер. с англ. Г. Мельникова. – М.: ДМК-пресс, 2007. – 312 с.
23. Официальная документация V-REP. [Электронный ресурс]. – Режим доступа: http://www.coppeliarobotics.com/ helpFiles/
24. Программируем роботов – бесплатный робосимулятор V-REP. Первые шаги. [Электронный ресурс] // Хабрахабр. – Режим доступа: https://habrahabr.ru/company/makeitlab/blog/253357/
25. V-REP – гибкая и масштабируемая платформа для робомоделирования [Электронный ресурс] // Geektimes. – Режим доступа: https://geektimes.ru/post/260370/
26. Learn Lua in 15 Minutes. [Электронный ресурс]. – Режим доступа: http://tylerneylon.com/a/learn-lua/
27. Lua за 60 минут. [Электронный ресурс]. – Режим доступа: https://zserge.wordpress.com/2012/02/23/Lua-за-60-минут/
28. Cyberbotics' Robot Curriculum [Электронный ресурс]. – Режим доступа: https://en.wikibooks.org/wiki/Cyberbotics%27_Robot_Curriculum
29. Saffiotti A. Fuzzy Logic techniques for autonomous vehicle navigation [Электронный ресурс]. – Режим доступа: http://www.aass.oru.se/Living/FLAR/Tutorial/HTML/ toc.html

Учебное издание

СОРОКИН Сергей Владимирович
СОЛДАТЕНКО Илья Сергеевич

ОСНОВЫ РАЗРАБОТКИ И ПРОГРАММИРОВАНИЯ РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

Учебное пособие
(программирование, инженерное моделирование, робототехника)

По направлениям бакалавриата:
мехатроника и робототехника, фундаментальная информатика
и информационные технологии, прикладная математика
и информатика, прикладная информатика

Компьютерная верстка И.С. Солдатенко
Подписано в печать 15.05.2017. Формат 60×84 $\frac{1}{16}$.
Усл.печ.л. 9,81. Тираж 200 экз. Заказ № 238.
Редакционно-издательское управление
Тверского государственного университета.
Адрес: 170100, г. Тверь, Студенческий пер. 12, корпус Б.
Тел РИУ: (4822) 35-60-63.