Программировать оборудование? Легко!

Все бывалые линуксоиды умеют писать софт, но чтобы программировать «железо», нужна большая любовь к компьютеру. Мы покажем, с чего начать...

Arduino крут. Крут, поскольку это миниатюрное устройство — примерно три на два дюйма [7x5 см] — имеет порт USB и программируемый чип. Он крут, потому что его можно запрограммировать, используя очень простой язык под названием Wiring. Но главное, вся документация производителя доступна по лицензии Creative Commons, и вы можете собрать устройство сами, если захотите. Правда, для большинства людей это, вероятно, все-таки экстрим, и поэтому продаются также готовые платы Arduino — причем по очень низкой цене. IDE для программирования Arduino доступна по лицензии GPL для множества платформ, и единственное, что стоит между вами и любезным вашему сердцу «железным» проектом — это плата Arduino, клевая идея и кое-какие мелкие детали, аккуратно втыкаемые в Arduino, чтобы сделать его гораздо более интересным.

Электроника и безопасность

Пожалуйста, учтите, что электронные компоненты очень чувствительны: статическое электричество может быть смертельно для вашего Arduino и других крошечных компьютерных частей, так что не шаркайте ногами и носите антистатический браслет. А также помните, что электронные компоненты могут, в свою очередь, быть опасны для вас — они часто содержат свинец, поэтому тщательно мойте руки после работы с вашим прибором!

Приступим к установке

Платы Arduino доступны в нескольких вариантах, но основные три – Arduino NG (“next generation” – «новое поколение»), Arduino NG Plus и Arduino Diecimila. Это не платы-конкуренты – NG появилась первой, потом ее сменила NG Plus, а за ней вышла Diecimila. Мы пользуемся NG, потому что купили ее давно, но сейчас при попытке приобрести ее вы обнаружите только Diecimila. В принципе, различия между этими платами минимальны: на NG Plus установлен процессор Atmega168, на NG – ATmega8, а Diecimila незначительно улучшена для более простой загрузки программ. Чипы ATmega8 и ATmega168 очень похожи; основное отличие в том, что в 168 больше места для программ, но для целей нашего урока это не станет проблемой – подойдет любая из трех.

Где взять плату Arduino? Рекомендуем PCB Europe (http://pcb-europe.net) – у них есть в продаже Diecimila за 22 евро (на момент написания статьи), и они с радостью ответят на вопросы, если вы не знаете точно, что вы хотите.

Теперь перейдем к вопросам посложнее: что втыкать в ваш Arduino? Сама плата имеет 14 цифровых разъемов и 6 аналоговых, а также встроенный светодиод и кнопку сброса, и нужно докупить дополнительные детали, чтобы что-то создать. Если вы из США, то, возможно, недалеко от вас есть магазин Radio Shack, где имеются наготове залежи прикольных штучек – просто подойдите к одному из продавцов, объясните ему, что вы пытаетесь сделать свой первый электронный проект, и уйдете с полной корзиной товаров долларов этак на 30. Если вы из Великобритании, то лучшее место разместить заказ на нужные

› Кабель USB с разъемами типа 2 (слева**) и 1 (**справа**) нужен для соединения Arduino с вашим ПК.**

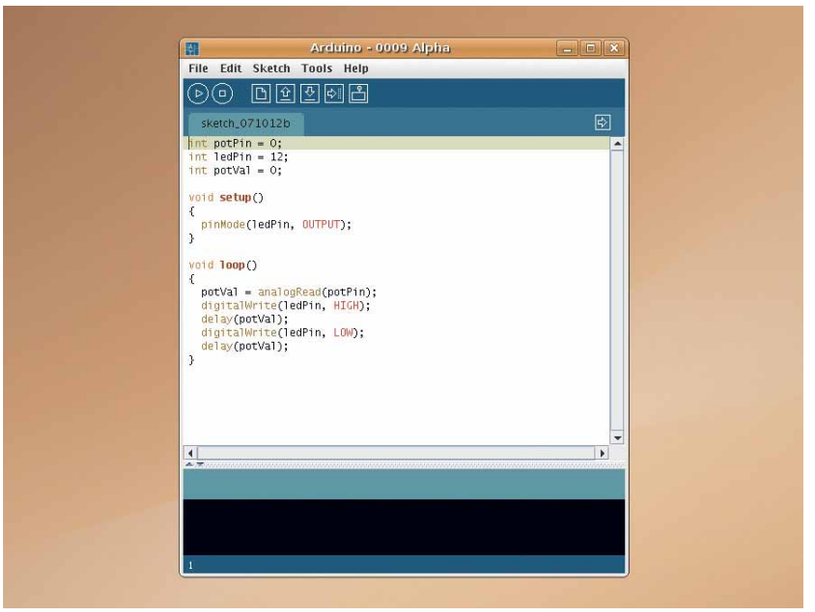
детали – www.maplin.co.uk; также рекомендую прихватить у них один из чудесных наборов Lucky Bags. Поразительно, сколько идей может подсказать случайная смесь деталей! Вне США и Великобритании, либо поищите хороший магазин электроники в вашей стране (говоря «электроника», мы не имеем в виду «продажа iPod»!), либо обратитесь в PCB Europe и купите один из комплектов деталей к Arduino.

Так или иначе, вот необходимый минимум деталей для нашего урока про Arduino:

» Макетная плата для сборки без пайки. Обычно это прямоугольник из белого пластика, со множеством отверстий, позволяющих прикрепить детали и соединить их электрически.

» Несколько проводков-перемычек. Можно накупить их разной длины, или нарезать самим – без разницы.

» USB-кабель Type A — Type B. Говоря по-человечески, это обычный соединительный кабель USB: на одном конце плоский, на другом почти квадратный.



» Несколько светодиодов. Лучше – разных цветов!

» Резисторы разных номиналов. Сопротивления более 10 кОм не понадобятся, но стоят они копейки, поэтому можете набрать всяких.

» Потенциометр.

» Фоторезистор.

Вы уложитесь при покупке вышеперечисленного в сумму менее 500 рублей, и даже еще останется, так что не бойтесь набрать лишнего – потом пригодится!

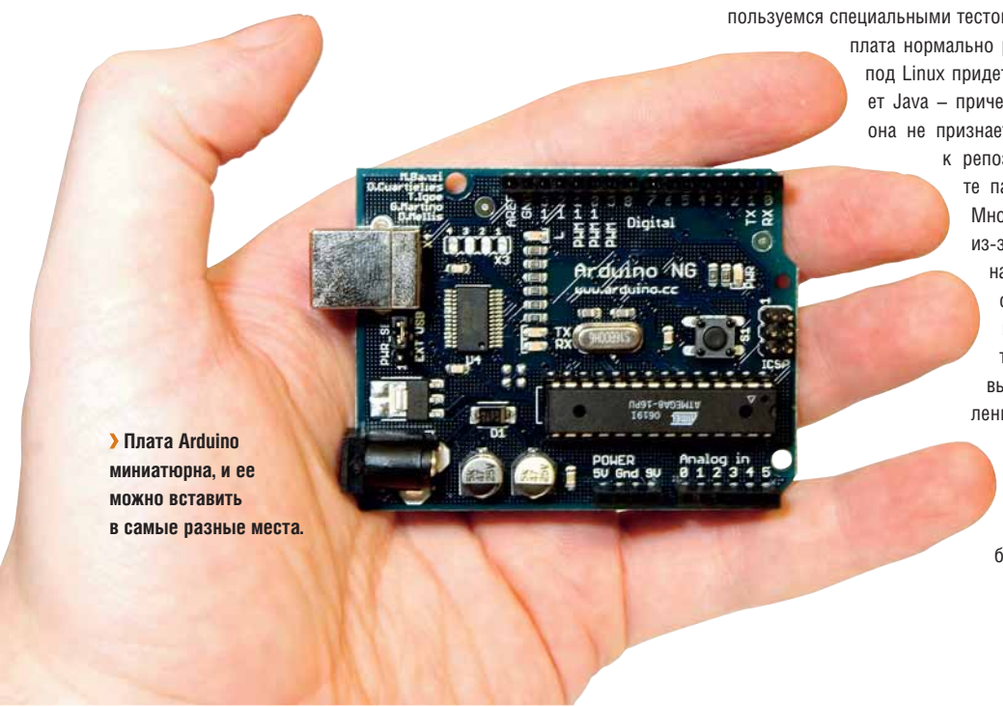
› IDE Arduino очень проста: предлагает подсветку кода, сохранение и загрузку и ничего более — зато вы можете загрузить все на вашу плату прямо отсюда.

«Первая вещь в любом программистском проекте обычно похожа на тест вроде 'hello world'.»

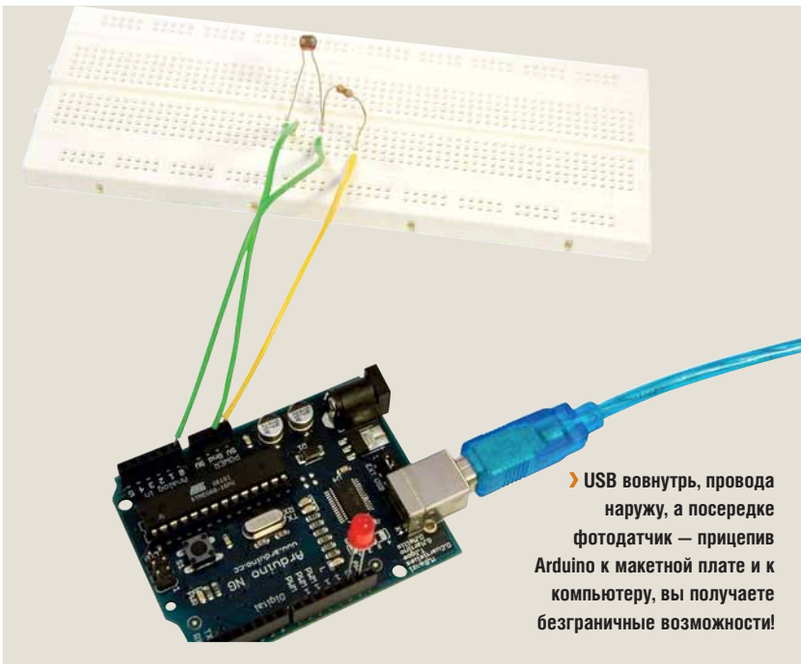
Мигалка

Первое, что делают в любом программистском проекте – аппаратном или каком еще – обычно нечто вроде "hello world": программу заставляют выдать во внешний мир простое сообщение, чтобы убедиться в правильности конфигурации.

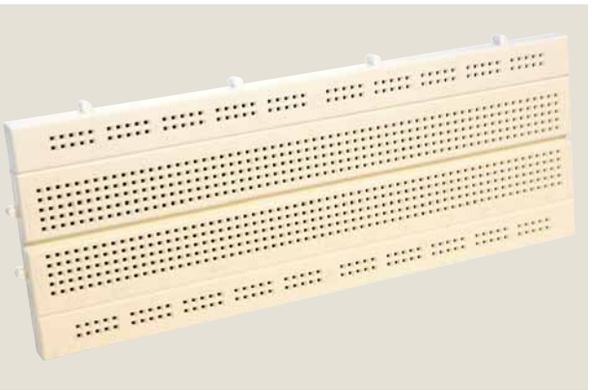
В Arduino встроено несколько светодиодов – есть, например, TX- и RX-светодиоды, мигающие при передаче и приеме данных. Мы воспользуемся специальными тестовыми светодиодами: они покажут, что плата нормально работает. Заставить Arduino работать под Linux придется хитростью, потому что она требует Java – причем официальной Java от Sun, клонов она не признает. Если у вас Ubuntu, подключитесь к репозиторию Multiverse, затем установите пакеты *un-java5-jre*, *gcc-avr* и *avr-libc*. Многие пользователи имеют проблемы из-за конфликта Arduino с поддержкой на клавиатуре системы Брайля; если она вам не нужна, сперва удалите пакет *brltty* – а если вы вставили плату Arduino при установленном *brltty*, выньте ее и воткните уже после удаления пакета. Установив программное обеспечение, запустите sudo update-alternatives --config java и выберите номер официальной Java от Sun. Этого должно быть достаточно, чтобы все заработало. »



› Плата Arduino миниатюрна, и ее можно вставить в самые разные места.



› USB вовнутрь, провода наружу, а посередке фотодатчик — прицепив Arduino к макетной плате и к компьютеру, вы получаете безграничные возможности!



› Полномерная макетная плата, не требующая пайки, обычно имеет около 60 разъемов в ширину и как минимум 10 в высоту. На этом рисунке контакты рядов соединены между собой, а контакты колонок не соединены.

» Если у вас не Ubuntu и не другая система на базе Debian, то вам опять же потребуются Java от Sun, *gcc-avr* и *avr-libc*, но зато в таких дистрибутивах, скорее всего, не придется беспокоиться об *update-alternatives*.

Мы включили IDE Arduino на DVD этого месяца – распакуйте ее на ваш рабочий стол и запустите команду arduino. При наличии проблем попробуйте запустить ее из терминала и посмотреть сообщения об ошибках. Запуская IDE впервые, вы должны будете ответить на вопрос о месте сохранения ваших программ – поддиректория в вашем домашнем каталоге вполне подойдет.

Итак, ПО для Arduino установлено; теперь подключим Arduino к компьютеру через USB-кабель. Если плата работает нормально, светодиод PWR (power, питание) будет гореть, а тестовый диод – мигать, показывая, что на плате все в порядке.

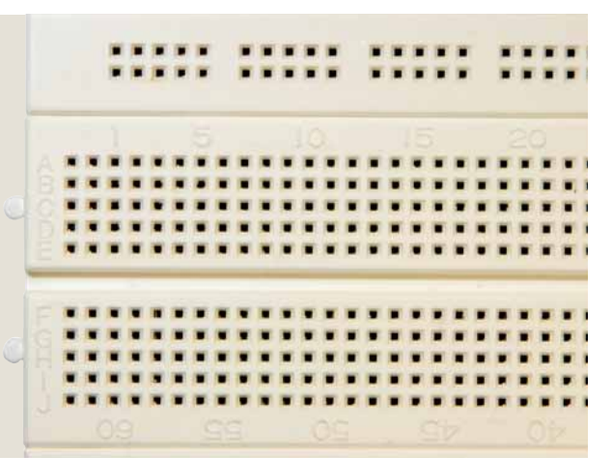
› Главное на плате Arduino — чип ATmega, но видны также аналоговые разъемы вверху и цифровые — внизу.

В редакторе Arduino зайдите в File > New для ввода нового кода. Файлы кода в Ардуинии называются “sketches” («наброски»), и с ними очень легко начать работать. Начнем мы с простого проекта: будем переключать светодиод из ВКЛ в ВЫКЛ, и я хочу показать вам код и убедиться, что он работает, а потом разобраться, как он работает.

Вот этот «набросок»:

```
int ledPin = 13;
void setup()
{
  pinMode(ledPin, OUTPUT);
}
void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(300);
  digitalWrite(ledPin, LOW);
  delay(100);
}
```

Теперь идем в Tools > Microcontroller и выбираем либо atmega8 (если у вас ArduinoNG), либо atmega168 (если у вас Diecimila). Если вы этого толком не знаете, пропустите данный шаг – Arduino определит процессор при подключении и выдаст ошибку, если обнаружит неправильный. Получив сообщение об ошибке, просто смените опцию! Когда процессор сконфигурируется, зайдите в Tools > Serial Port; вы увидите список возможных USB-устройств. В Linux, вероятно, надо будет выбрать /dev/ttyUSB0.



› В моей макетной плате ряды обозначаются цифрами, а колонки — буквами. Я так на них и ссылаюсь, чтобы вы смогли в точности скопировать мою схему.

› При повороте ручки потенциометр возвращает большее аналоговое значение — убедитесь, что используете analogRead()!

ОК, настройка закончена – пора выгружать ваш «набросок» на плату! В панели меню IDE, вы можете увидеть иконку со стрелкой вправо – это кнопка Upload [Выгрузка]. Для Diecimila вам достаточно нажать на кнопку прямо сейчас, и программа будет выгружена. В противном случае сперва нужно нажать на маленькую кнопку сброса на плате Arduino, подготовив ее к принятию нового «наброска», а затем нажать кнопку выгрузки. Светодиоды TX и RX секунду поморгают, затем секунд пять ничего не будет происходить (Arduino ждет, не поступят ли дальнейшие инструкции), и наконец ваш тестовый диод должен начать мигать. Заработало!

С этим маленьким проектом вы можете шагнуть дальше. Возьмите из набора светодиод и подключите его к цифровому контакту 13. Если

«Если светодиод вставлен неправильно, не беда: просто переверните его.»

вы раньше со светодиодами не работали, хорошенько рассмотрите его перед установкой: у него две ножки, причем разной длины, и одна сторона цветного пластикового ободка плоская. Таким образом указывается полярность: ножка покороче и плоский обод – это «минус». Подключая ваш светодиод к плате Arduino, убедитесь, что плюсовая ножка подключена к контакту 13, а минусовая – к Gnd (ground, земля), и если все будет нормально, вы увидите, как он мигает в такт тестовому светодиоду. Ошиблись – ничего страшного, просто побыстрее выдерните светодиод!

Как работает код

Теперь, когда плата Arduino работает правильно, объясню вам, как работает код, чтобы вы сами могли его модифицировать:

» int ledPin = 13 описывает переменную ledPin типа integer (целое число – ну, то есть, не вида 3,1) и присваивает ей начальное значение 13.

» setup() – функция по умолчанию в Arduino. Она вызывается при запуске вашей программы, чтобы вы могли задать начальную конфигурацию.

» pinMode(ledPin, OUTPUT) сообщает Arduino, что вы хотите посылать данные на контакт 13, а не считывать их.

» loop() – еще одна функция по умолчанию Arduino: она вызвается всякий раз, когда процессор ищет, что бы ему еще сделать.

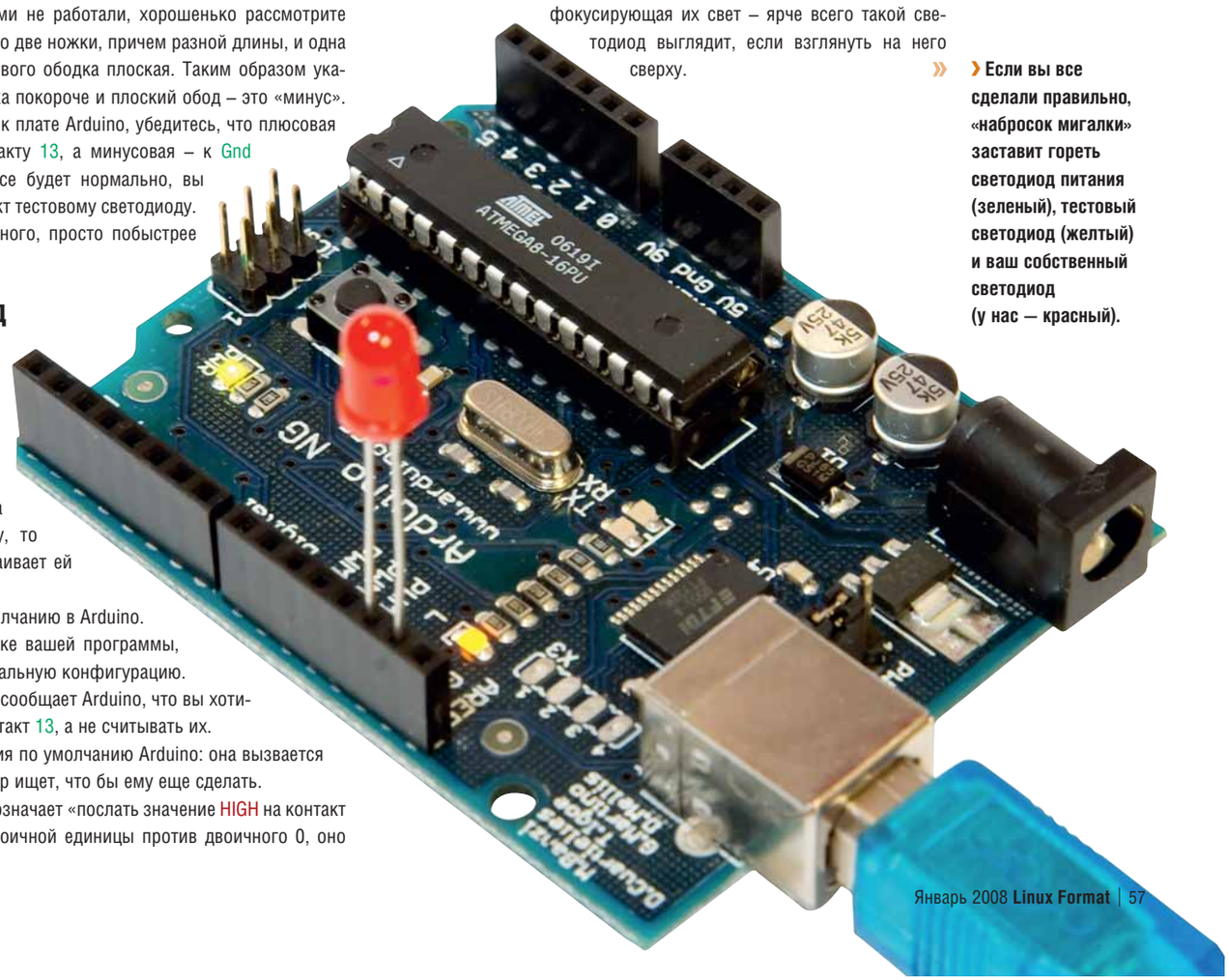
» digitalWrite(ledPin, HIGH) означает «послать значение HIGH на контакт 13»; HIGH – эквивалент двоичной единицы против двоичного 0, оно же – «ВКЛ», в отличие от «ВЫКЛ». Это включает светодиод.

» delay(300) заставляет процессор сделать паузу в 300 миллисекунд, т..е. примерно на треть секунды.

» digitalWrite(ledPin, LOW) выключает светодиод.

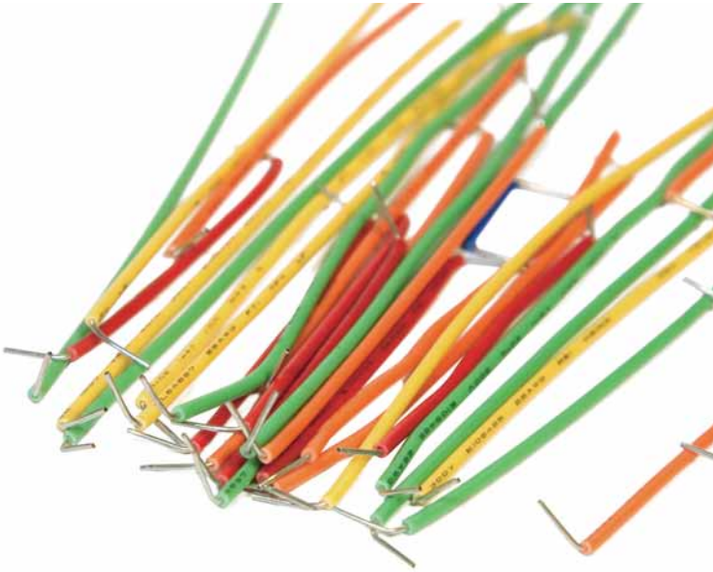
» delay(100) заставляет процессор ждать одну десятую секунды.

Вот и все! Функция loop() вызывается с частотой, равной частоте работы процессора, а вызовы delay() вставлены, чтоб разрешить процессору периодически перевести дух – иначе светодиод будет мигать так часто, что глазу не уследить, и покажется, что он всегда включен! Помните, что на большинство светодиодов насажена маленькая линза, фокусирующая их свет – ярче всего такой светодиод выглядит, если взглянуть на него сверху. »



› Соедините светодиод с цифровыми контактами 13/GND, да присмотритесь к длине его ножек, чтобы не перепутать полярность!

› Если вы все сделали правильно, «набросок мигалки» заставит гореть светодиод питания (зеленый), тестовый светодиод (желтый) и ваш собственный светодиод (у нас — красный).



› **Коробку нарезанных проводов можно купить в любом хорошем магазине радиоэлектроники. Чтобы ваша схема смотрелась круто, используйте провода разных цветов для разных типов соединения!**

› **Резисторы безумно дешевы и обычно продаются по пять штук; внимательно осмотрите полоски, чтобы определить их номинал.**

» Использование макетной платы

Макетная плата имеет металлические дорожки, спрятанные под отверстиями и соединяющие ряды контактов в электрическую цепь. Колонки не соединены, поэтому горизонтальных контактов нет – есть только вертикальные. Если у вас полномерная макетная плата типа нашей, то на ней между рядов имеется просвет. Ряды по краям просвета тоже не соединены между собой. Рассмотрим на примере, как это работает: модифицируем простую светодиодную систему введением макетной платы и двух проводов. В моей плате колонки пронумерованы с 1 до 60, а ряды обозначены буквами от A до J, как показано на фото на стр. 56, и я воспользуюсь этими обозначениями для указания, куда втыкать провода. Если у вас таких обозначений нет, мои все равно вам помогут, подсказав, где ряд, а где колонка, а больше нам ничего и не надо.

> «Шпаргалка подскажет, какие полоски каким номиналам резистора соответствуют.»

Итак, для перенесения проекта «мигалки» на макетную плату, соедините проводом Gnd с J33, а другим проводом – Digital 13 с J32. Это подводит ток к контактам I33 и I32, H33 и H32, G33 и G32, F33 и F32. Моя плата имеет просвет между рядами F и E, поэтому E32 и E33 не входят в схему. Теперь вам осталось только поместить ваш светодиод на правильное место, чтобы замкнуть цепь; по-вашему, это куда?

Если вы не страдаете схемобоязнью, можете просто тыкать в контакты, пока светодиод не загорится – «методом тыка» вы много чего изучите, и это очень забавно, поверьте! В данном примере, светодиод загорится в позиции F33 (минус, короткая ножка) и F32 (плюс, длинная ножка); то же произойдет при помещении его в колонки G, H или I.

Попробуем сделать что-нибудь еще: с другой стороны платы Arduino находятся аналоговые входные контакты Analog In, а также несколько помеченных как Power. Ток в них будет побольше – заметили на одном из них отметку 5V, а на втором 9V? Это, между прочим, вольты; при таком напряжении нужна осторожность – подав на светодиод слишком много, вы его спалите. Если вы приволокли мешок светодиодов, беда невелика; но вдруг у вас их только три-четыре, и они последние! К слову, вполне безопасно подсоединить их на пару секунд, просто чтобы убедиться, что они работают. Пожалуйста, сделайте это сейчас: отключите Arduino от питания, затем соедините 5V с J33, а Gnd рядом с 5V – с J32. Потом на секунду включите плату: ваш светодиод ярко засветится (а заодно и нагреется!). Убедившись, что он работает, отключите USB-кабель, чтобы светодиод погас.

Перейдем к более сложной задаче: я хочу познакомить вас с резистором. Сия маленькая деталь задает падение напряжения, в зависимости от своего номинала сопротивления. Номинал изображен маленькими цветными полосками на корпусе резистора, но простым смертным тут ловить нечего – лучше держать под рукой шпаргалку, разъясняющую, что какие полоски означают. Теперь соедините 5V с J38 – так, чтобы светодиод не был подключен к питанию. Для замыкания цепи соедините резистором I38 и J33, и вы снова увидите свечение светодиода – но чуть тусклее. Если цветные коды резисторов для вас китайская грамота, попробуйте поочередно вставлять разные резисторы и рассортируйте их по яркости свечения светодиода!

Чтение ввода

В фирменном комплекте деталей имеется фотоэлемент – датчик, возвращающий различные значения в зависимости от падающего на него света. Потребуется чуть больше проводов, дополнительный вызов функции, а также условный оператор, но на самом деле не так все сложно – и стоит затраченных усилий: ведь вы напишете код, благодаря которому Arduino сможет ориентироваться в своей среде! Моя макетная плата снабжена двумя специальными дорожками, помещенными ради удобства подвода питания, но я ими пользоваться не буду, потому что платы поменьше не всегда обладают подобной роскошью.

Сперва поместите ваш фотоэлемент на макетную плату. Я воткнул его в F34 и F37. Поместите мощный резистор в G34 и соедините его с I29, затем соедините проводом J29 и 5V – это подаст питание на фотоэлемент. Необходимо также соединить проводом G37 и контакт Gnd рядом с 5V, для замыкания электрической цепи. Чтобы считать данные с фотоэлемента, соедините другим проводом H34 с Analog 0, и ваша система готова для программирования!

«Набросок» для чтения данных с аналогового входа в Arduino очень прост: вам необходимо указать, с какого контакта вы будете брать данные фотоэлемента, а также сохранить эти данные. Тут пригодится функция analogRead(), возвращающая значение, которое вы можете использовать для работы. В виде «наброска» это выглядит так:

```
int ledPin = 13;
int lgtPin = 0;
int lgtVal = 0;
void setup()
{
  pinMode(ledPin, OUTPUT);
}
void loop()
{
  lgtVal = analogRead(lgtPin);
```

Оборудование в LXF

Мы в *Linix Format* очень редко занимаемся оборудованием вроде Arduino – фактически, это наш первый опыт! Поэтому, пожалуйста, напишите, как вы думаете: нужно ли продолжать эту статью, с применением большего количества деталей, более сложных «набросков» и, возможно, даже каких-нибудь полных проектов? Или лучше держаться от «железок» подальше? Дайте нам знать! letters@linuxformat.ru

> «Чтение с аналогового входа в «наброске» Arduino очень простое — через функцию analogRead().»

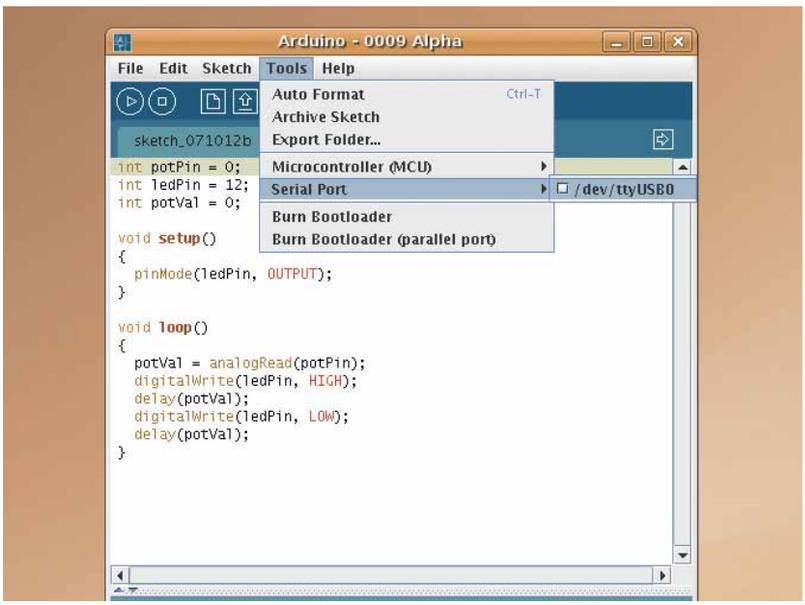
```
  digitalWrite(ledPin, HIGH);
  delay(lgtVal);
  digitalWrite(ledPin, LOW);
  delay(lgtVal);
}
```

Как вы думаете, что этот «набросок» должен делать? Если вы не уверены, попробуйте разобраться, загрузив его в вашу плату и прикрывая/открывая фотоэлемент!

Дерзать дальше...

Если вы раздобыли потенциометр, то заставить ваш фотоэлемент работать будет особенно легко: ваш провод от Analog 0 необходимо соединить со средней ножкой потенциометра; провод от Ground – с левой ножкой; а провод к питанию (через резистор) – с правой ножкой. Убедитесь, что потенциометр плотно вставлен в макетную плату – ножки у них обычно более толстые, так что придется на него поднажать!

Конечно, это только беглый обзор возможностей Arduino, и все потому, что настоящая магия заключается в программах, которые вы напишете. Мы показали вам, как собрать схему для чтения и записи данных, и следующий вопрос такой: что вы будете с ней делать? LXF



› Перед попыткой загрузки в плату убедитесь, что для Arduino выбран правильный порт — это обычно /dev/ttyUSB0 или нечто подобное.

Создай свой гаджет
Возвращение

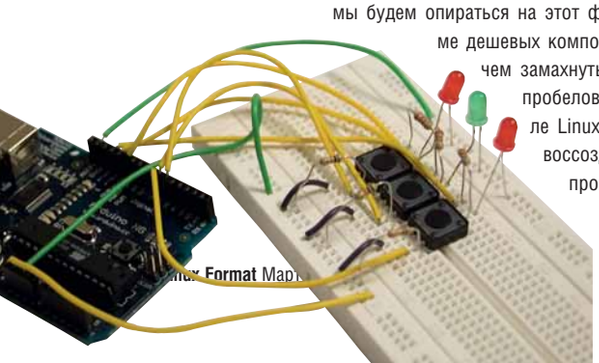
По многочисленным просьбам читателей, наш цикл аппаратного хакинга вернулся. **Грэм Моррисон** собирает игру типа «Саймон сказал», используя Arduino и несколько светодиодов.

Пропустили LXF100/101?

Предварительный материал для этого проекта содержался в статье про Arduino в **LXF100/101**. Если вы пропустили этот номер, можете скачать PDF со статьей с www.linuxformat.ru/download/LXF100-101.arduino.pdf.

Статья об аппаратном хакинге из LXF100/101 оказалась столь популярной, что мы решили продолжить. В статье говорилось об Arduino – небольшой печатной плате, включающей простое окружение ввода/вывода и USB-порт, которыми легко управлять, используя несложный язык программирования и IDE на Java. Самое замечательное то, что в ней все – от дизайна платы до управляющего ПО – открытое (Creative Commons для аппаратной схемы и GPL для ПО). В результате вокруг проекта Arduino образовалось сплоченное сообщество, и это также повлияло на выбор его как платформы для нашей «железной» статьи. Другая причина – дешевизна решения. Примерно за 2000 рублей вы можете приобрести полный набор для новичка, содержащий все, что нужно для начала, включая зуммер, светодиоды, сенсоры и кнопки – и, конечно, сам Arduino.

В статье этого месяца мы собираемся продолжить начатое и, опираясь на основы, раскроем чуть больше потенциала Arduino. После этого у вас будет вполне достаточно информации для уверенной работы над собственными проектами. В следующие несколько месяцев мы будем опираться на этот фундамент, не используя ничего, кроме дешевых компонентов и открытых кодов. Но прежде чем замахнуться на большее, заполним несколько пробелов, оставшихся от первого урока. В стиле Linux Format, мы превратим это в забаву, воссоздав классическую игру 80-х – Simon, простую игру на повторение [«Саймон сказал», игра, в которой все повторяют то, что делает или просит сделать ведущий, – *прим. пер.*]. Игроки должны нажимать клавиши в соответствии с проигранными Саймоном нотами. Сперва вы слушаете последовательность звуков, а затем пытаетесь воссоздать мелодию с помощью четырех больших кнопок на игровой системе. Последовательность становится все длиннее и сложнее для повторения.



Мы сделаем упрощенную версию этой игры, используя три светодиода (вместо нот) и три кнопки. Идея игры следующая: компьютер выбирает последовательность включения светодиодов, а игрок ее повторяет, нажимая на соответствующие кнопки. Каждый раз, когда игрок вводит ее правильно, мы удлиняем последовательность на единицу, и количество очков игрока будут зависеть от запомненной им последовательности.

Пишем игру

Вот что нам понадобится для построения собственного «Саймона»:

- **Плата Arduino.**
- **3 переключателя.**
- **3 светодиода.**
- **3 резистора по 1 КОм для переключателей.**
- **3 резистора по 1 КОм для светодиодов.**

Мы также будем использовать монтажную плату из оригинальной статьи для скрепления всех компонентов и проводов вместе.

Первым делом разместим переключатели. Они имеют четыре контакта, но нам нужны только два, на любой из сторон. Наши три переключателя поместим так, чтобы они выглядели как «мостик» над цен- »

тральной канавкой макетной платы, а соединять будем только две ножки по одну сторону канавки; другую сторону используем для присоединения светодиодов. Дизайн игры должен учитывать эргономику: люди захотят поиграть с вашим устройством, но не с проводами и резисторами, путающимися у них под руками. Кроме того, продумайте, как разместить светодиоды – они должны быть рядом с кнопками, чтобы было ясно, какой светодиод какой кнопке соответствует. У нас получилась не слишком успешная раскладка, и если у вас тоже отсутствуют навыки дизайна а-ля Джонатан Ив, вы можете обнаружить, что весь процесс лучше начать заново, хорошенько разобравшись в проблемах размещения проводов и резисторов.

Расставим кнопки

Следующая задача – подсоединить переключатели. Соединим выход питания 5 В на Arduino с одной из горизонтальных шин на углу платы. Как и полагается «шине», при соединении с одной точкой на горизонтальной линии сигнал передается в любую точку по всей длине платы. Шина обычно используется для передачи питания и заземления на различные компоненты, присоединяемые к центральной части макетной платы (также известной как терминальная полоса), и это именно наш случай. От горизонтальной линии, несущей теперь 5 В, мы должны «прокинуть» резистор к ножке каждого переключателя, чтобы не подавать на переключатели слишком большое напряжение. Если у вас нет макетной платы с шинами, это не мешает продолжить работу над нашим проектом, но тогда придется выполнять каждое соединение с платой Arduino вручную. Надо также подвести провод от ножки, на которую подано напряжение, к трем отдельным цифровым коннекторам Arduino, чтобы мы могли читать состояние каждой кнопки. Используем контакты 5, 6 и 7.

Еще надо соединить другую ножку каждого переключателя с «землей» на Arduino (мы использовали следующий контакт за выходом 5 В). Простейший способ сделать это – соединить все эти ножки с шиной, соседней с шиной питания, а эту новую шину подключить к контакту «земля» на Arduino. Вот и весь монтаж переключателей; установка светодиодов будет еще проще.

Подключим светодиоды

На другой стороне канавки, или паза, посреди макетной платы, надо теперь прикрепить наши три светодиода – поближе к кнопкам, но оставив достаточно места для присоединения земли (GND) к одной ножке и управляющего провода к другой. Мы использовали шину по другую сторону паза для соединения с GND и воткнули светодиоды отрицательной стороной (короткая ножка, или плоская сторона головки) прямо в нее. Другую ножку надо соединить через резистор проводом с одним из контактов ввода/вывода Arduino – у нас это контакты 10, 11 и 12.

До начала работы приведем список соединений, использованных нами на Arduino.

- **1-й переключатель** на Вход 7, светодиод на 12.
- **2-й переключатель** на Вход 6, светодиод на 11.
- **3-й переключатель** на Вход 5, светодиод на 10.
- **5** В питания на шину к трем переключателям.
- **GND** (Земля) на шину к трем переключателям.
- **Другой GND**, соединенный с шиной, используемой светодиодами.

Программирование

Ну вот, мы все собрали – теперь перейдем в программное окружение для написания кода, необходимого для игры. Мы рассмотрели работу в окружении в первой статье, это довольно просто. Клиент использует Java, и если вы испытываете проблемы с его запуском, вероятно, нужно убрать программное обеспечения терминала Брайля: оно конфликтует с установленным ПО Arduino.

Вверху нового файла исходного кода следует добавить строки, инициализирующие все наши соединения и глобальные значения:

```
#DEFINE MAXSEQ 20
int rand_array[MAXSEQ];
int ledPin1 = 12;
int ledPin2 = 11;
int ledPin3 = 10;
int inPin1 = 7;
int inPin2 = 6;
int inPin3 = 5;
int score = 1;
```

Первая директива **DEFINE** задает максимальное значение последовательности мигания светодиодов, и эта последовательность хранится в массиве, создаваемом в следующей строке. Массив – это строка элементов, и каждый элемент в нашем примере говорит плате Arduino, какой светодиод зажечь и какая кнопка должна быть нажата: короче, это число от 1 до 3. Строкой ниже мы сообщаем Arduino, к каким входам подключены светодиоды и переключатели (соответственно, **ledPin** и **inPin**). Последняя строчка создает глобальную переменную для хранения очков, отмечающую, до какой длины последовательности добрался игрок.

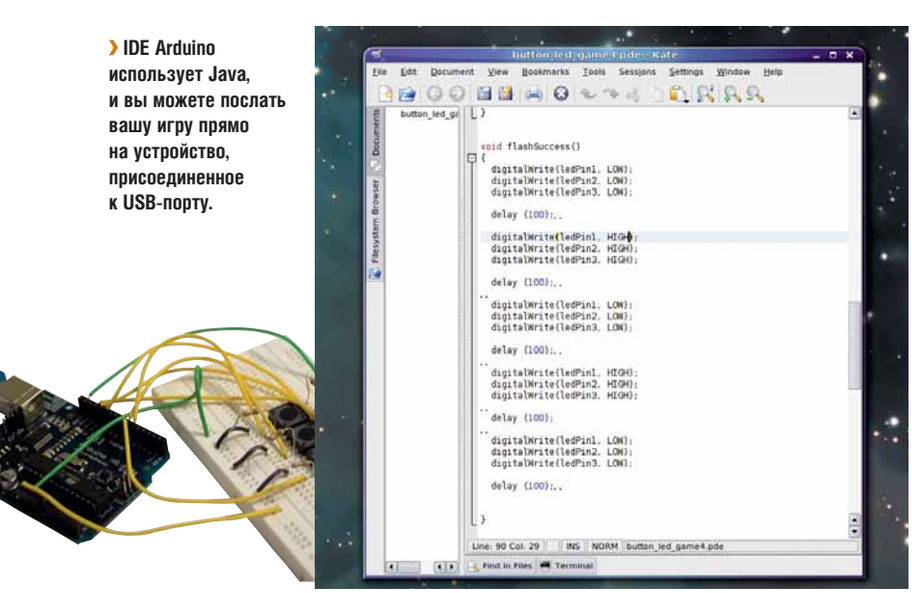
Теперь надо добавить функцию setup, используемую Arduino для настройки различных входов и выходов:

```
void setup() {
  long val_rand = 0;
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(inPin1, INPUT);
  pinMode(inPin2, INPUT);
  pinMode(inPin3, INPUT);
  randomSeed(analogRead(0));
  for (int i=0; i<=MAXSEQ; i++){
    rand_array[i] = random (3);
  }
}
```

В этом фрагменте кода мы делаем две вещи. Сперва мы говорим плате Arduino, какие контакты использованы как выходы (для мигания светодиодов), а какие – как входы (для считывания состояния кнопок). Мы также используем функцию **setup()** для заполнения массива случайными значениями, и делаем это в два приема. Функция **randomSeed** читает вход с ножки 0. Так как ножка не подсоединена, то с нее, фактически, снимается случайный шум, который мы используем как затравку для генератора случайных чисел, а генератор встроен внутрь цикла ‘for’, заполняющего массив случайными значениями между 1 и 3.

Покончив с конфигурацией, перейдем к логике программы.

Вот как должна протекать игра:



IDE Arduino использует Java, и вы можете послать вашу игру прямо на устройство, присоединенное к USB-порту.

1 Длина последовательности принимается 1. Мы проигрываем всю последовательность на светодиодах.
2 Игрок должен в ответ нажать на переключатель, соответствующий светодиоду.
3 Если игрок полностью повторил последовательность, ее длина увеличивается на 1.
4 Когда игрок допускает ошибку, игра повторяет последовательность текущей длины.

Мы осуществим это с помощью четырех функций. Первая будет считывать, какая кнопка нажата (мы назовем ее **readButtons**), вторая – проигрывать последовательность на светодиодах (**playSequence**), и понадобятся еще две функции для сигнализации об успехе попытки или сбое (**flashSuccess** и **flashFailure**). Вот функция **readButtons**:

```
int readButtons()
{
   int val1 = 0; int val2 = 0; int val3 = 0;
  do {
    val1 = digitalRead(inPin1); val2 = digitalRead(inPin2); val3 =
digitalRead(inPin3);
   } while (val1 == HIGH && val2 == HIGH && val3 == HIGH);
   if (val1==LOW){
     return 0;
   } else if (val2==LOW) {
     return 1;
   } else if (val3==LOW) {
     return 2;
   }
}
```

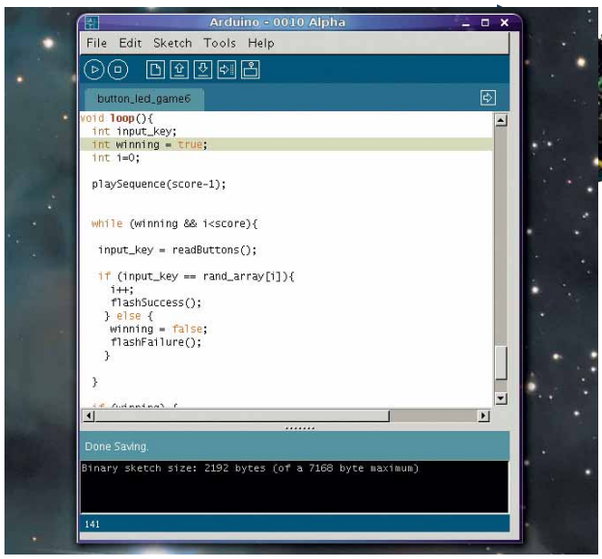
Она ожидает нажатия кнопки пользователем, а затем возвращает значение 0, 1 или 2, в зависимости от нажатой кнопки. Мы выбрали эти цифры, потому что они в точности соответствуют номерам, используемым для зажигания каждого светодиода. Важное замечание: переключатель возвращает значение HIGH, когда он отпущен и LOW – когда нажат. Так происходит потому, что в нормальном состоянии переключатель пропускает ток, а его нажатие разрывает цепь.

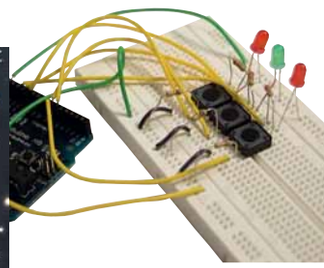
Далее добавим функцию, проигрывающую последовательность на светодиодах:

```
void playSequence(int count)
{
  for (int i=0; i<=count; i++){
    digitalWrite(ledPin1, LOW); digitalWrite(ledPin2, LOW);
digitalWrite(ledPin3, LOW);
    delay (200);
    switch (rand_array[i]) {
     case 0:
       digitalWrite(ledPin1, HIGH); break;
     case 1:
       digitalWrite(ledPin2, HIGH); break;
     case 2:
       digitalWrite(ledPin3, HIGH); break;
    }
    delay (200);
  }
  digitalWrite(ledPin1, LOW); digitalWrite(ledPin2, LOW);
digitalWrite(ledPin3, LOW);
}
```

Эта функция принимает один аргумент, **count**, зависящий от длины проигрываемой последовательности. Он используется как предел для цикла 'for', и внутри цикла мы переключаем светодиоды согласно массиву, несущему последовательность. Мы также должны обеспечить выключение светодиода до и после, а также задержку между вспышками, чтобы игрок мог различать свечение отдельных светодиодов.

Остались две функции, показывающие успех или ошибку нажатия кнопки. Будем выдавать две быстрых вспышки всех светодиодов при каждом правильном нажатии кнопки и длинную паузу, если игрок допустит ошибку. Здесь нет места для печати обоих функций, но вам необходимо сочетать вызовы функций **digitalWrite(ledPin1, LOW)** и **digitalWrite(ledPin1, HIGH)** для каждого светодиода, а затем запускать **delay (100)**. Как мы это сделали, показывает исходный код на www.linuxformat.co.uk/mag/arduino.





› **Не обязательно цепляться за скверный редактор Arduino. Используйте свой любимый редактор и сохраните код в той же директории эскизов.**

Последняя функция зовется **void loop()**, и без нее не обходится ни один проект Arduino: она заставляет устройство циклически работать, пока идет игра. Вот наша версия:

```
void loop(){
  int input_key;
  int winning = true;
  int i=0;
  playSequence (score-1);
  while (winning && i<score){
   input_key = readButtons();
   if (input_key == rand_array[i]){
     i++;
     flashSuccess();
   } else {
     winning = false;
     flashFailure();
   }
  }
  if (winning) {
  score++;
  }
}
```

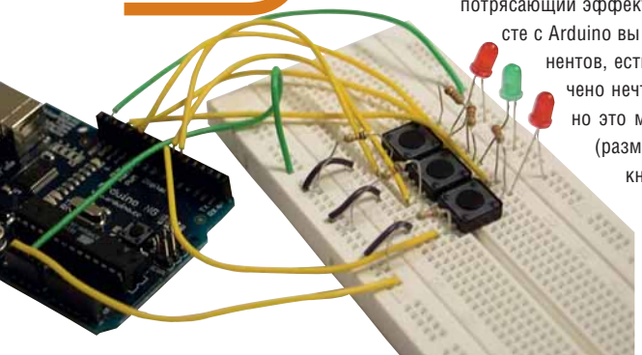
Эта функция воплощает программную логику, описанную ранее. Мы проигрываем последовательность (сперва это одна вспышка), потом погружаемся в цикл **while**, который выполняется, пока игрок нажимает правильные кнопки. Если игрок ошибся, играем опять, с последовательностью той же длины; а если он правильно воспроизвел последовательность, наградой будет новая игра с увеличенной длиной последовательности для запоминания (это делает **score++**). Если вы вставите сюда какие-нибудь классные дополнения, сообщите нам, и мы внесем их в наше собственное устройство! LXF

Аппаратный хакинг ПО НОВОЙ

ЧАСТЬ 2 Игра, которую мы создали месяц назад – немая, и это дает **Грэму Моррисону** повод добавить звук, используя только Arduino и динамик.

Месяц назад мы создали игру типа «Саймон сказал», используя Arduino – аппаратную платформу для простых и не очень электронных проектов. Мы разместили три кнопки и три светодиода на макетной плате и написали маленькую программу, передававшую на них случайную последовательность вспышек, которую игрок должен был повторить, нажимая на кнопки в заданном порядке. При каждом правильном воспроизведении последовательности ее длина увеличивалась на единицу, и все повторялось снова. Чем большей длины случайные последовательности вы запоминали, тем больше очков получали.

Мы собираемся надстроить проект прошлого месяца, добавив другую важную особенность – звук! Это очень просто, но оказывает потрясающий эффект на игровой процесс. Если вместе с Arduino вы купили стартовый набор компонентов, есть хороший шанс, что туда включено нечто вроде пьезоэлемента – обычно это маленькая черная круглая штука (размером примерно с «нажималку» кнопки), с маленькой дырочкой наверху. Пьезоэлемент способен как генерировать, так и улавливать звук, почему его и включают в стартовый набор. Если пьезоэлемента у вас нет, не горюйте – берите пример с нас. В магазин мы за ним не пошли, а раскурочили корпус старого компьютера, выудили оттуда динамик и припаяли два провода к контактам по краям магнита, управляющего динамиком (а вы, может, сумеете вынуть его прямо с проводами). Подойдет любой маленький динамик – вы удивитесь, какая масса электронных приборов содержит подобное. Если у вас валяется что-то сломанное и электронное, загляните вовнутрь.



Звук и свет

Подсоединив динамик к Arduino, придадим вспышкам светодиодов звук, чтобы каждой из них соответствовала определенная нота. Тогда случайная последовательность вспышек заодно будет сопровождаться сгенерированной мелодией, а игрок пусть повторяет последовательность вспышек/мелодию, нажимая на три кнопки.

Даже если игра «Саймон сказал», которую мы создаем, вам ни к чему, вы можете использовать то, что изучите на следующих страницах, для использования в больших проектах. Звук в Arduino будет просто работать, и есть немало приложений, где ему найдется применение.

Пропустили LXF100/101?

Основы этого проекта описывались в статье про Arduino в LXF100/101. Если вы пропустили этот материал, можете скачать PDF с ним с www.linuxformat.ru/download/LXF100-101.arduino.pdf.

Генерация звука

Существует несколько способов генерировать звук с помощью Arduino, разных уровней сложности. Мы воспользуемся простейшим, известным также как «битье по битам». Этот метод позволяет генерировать ноты различной тональности, но не более – просто «бип» некоторой длительности. Не спешите воротить нос – звук на самом деле очень приятный, и он знаком всем, кто вырос в 1980-х. Наш звук будет очень похож на тот, что порождали подлинные игры «Саймон» и много других электронных устройств, в том числе типичный «бип», который звучит при включении компьютера. Выбран он потому, что это самый простой и дешевый способ получения звука, и вам не нужны никакие аудиосредства, кроме динамика.

Если вы интересуетесь звуком и хотите сделать что-нибудь чуть более амбициозное, то есть другие методы генерации, использующие цифро-аналоговый преобразователь для воссоздания сэмплов и реальных звуков; есть и проекты Arduino, которые воспроизводят различные дискретные компоненты синтезаторов, включая осцилляторы, фильтры и огибающие, а также увязку всего этого с MIDI. Тем, кому интересны такие приложения, сайт Arduino предоставляет исчерпывающую информацию.

› Мы добыли динамик бесплатно, но его можно купить в любом магазине – это недорого.

Бип... Бип...

Но нам-то нужен всего лишь «бип». Смотрите, как легко это делается. Присоедините положительный контакт динамика к контакту питания 9 В на Arduino (на плате динамика должны находиться маленькие знаки «плюс» и «минус», показывающие полярность). Далее соедините минусовый провод с цифровым контактом 9 на Arduino – вот и все необходимые соединения. Если у вас уже собрана игра «Саймон сказал» из прошлого номера, эти два соединения должны располагаться параллельно проводам, идущим к и от светодиодов и кнопок на макетной плате. Осталось написать код, создающий звук. Основной принцип очень похож на переключение светодиодов из «включено» в «выключено», по крайней мере с точки зрения программирования. Поэтому метод и назван битье по битам. Сперва надо установить режим для контакта, подключенного к динамику, затем послать сигналы **HIGH** и **LOW** на тот же контакт, тем самым посылая цифровой сигнал по проводу на динамик. В результате мы получим небольшой «клик», так как катушка динамика в этот момент дернется. Если вы зарисуете движение на листке бумаги, то увидите, что динамик сгенерировал прямоугольный импульс – резкое движение вверх при включении и затем такое же резкое при спаде. Теперь, послав гораздо больше таких маленьких сигналов на динамик, мы сгенерируем больше таких импульсов, и суммарный эффект от движения катушки и диффузора динамика вперед и назад даст нам звуковой тон, высота которого будет зависеть от числа таких циклов в секунду.

> **«Наш звук будет похож на издаваемый исходной игрой “Саймон сказал”.»**

...Бип! Бип! Ура!

Хватит теории – перейдем к шумам. Откройте новый проект Arduino в Arduino IDE (или в любимом текстовом редакторе) и введите следующие строки в начале файла:

```
#define note_len 200000
int speakerOut = 9;
void setup() {
  pinMode(speakerOut, OUTPUT);
}
```

Эти строки говорят контроллеру Arduino, как мы хотим использовать контакты на плате. Теперь надо добавить код, генерирующий звук. Для этого просто вставьте следующую функцию:

```
void playTone(int note) {
  long elapsed_time = 0;
  while (elapsed_time < note_len) {
    digitalWrite(speakerOut,HIGH);
    delayMicroseconds(note / 2);
    digitalWrite(speakerOut, LOW);
    delayMicroseconds(note / 2);
    elapsed_time += (note);
  }
}
```

В цикле ‘**while**’, цифровой импульс посылается на динамик, используя функции **digitalWrite** – **HIGH** для включения и **LOW** для выключения. Между каждым применением **digitalWrite** задается задержка с помощью функции **delayMicroseconds**. Как следует из ее названия, она останавливает работу программы на число микросекунд, заданное в скобках. В нашем случае мы присвоили это число переменной ‘note’, так как именно от значения этой задержки зависит требуемый тон: чем меньше задержка, тем быстрее импульсы посылаются на динамик, и тон будет выше. Мы затем добавляем суммарную длину всех задержек в переменную **elapsed_time**, чтобы знать, сколько времени у нас проигрывается звук, и когда нота проиграется с длительностью, заданной в **note_len**, мы выйдем из цикла **while**.

Для создания звука, переходим к окончательной функции – **loop**. Она выполняется Arduino постоянно, и здесь можно вызвать функцию **playTone** для создания звука:

```
void loop() {
playTone (3830);
}
```

»

Поместим его в игру

Теперь, создав основу для генерации звука с помощью Arduino и дешевого динамика, применим изученное к нашей игре. Первый шаг – скопировать функцию **playTone** из нашего предыдущего примера в исходный код игры. Мы будем использовать ее похожим способом, вызывая ее, когда необходимо издать звук. Чтобы идти в ногу с новой функцией, нам также необходимо добавить похожую строку **pinMode** в функцию **setup()** и убедиться, что описания **note_len** и **speakerOut** скопированы в начало файла. Нам также надо задать через **#define** различные задержки при проигрывании функции **playTone**. Их использование позволит нам избежать голых чисел в качестве значений, какие мы использовали в функции **loop()**, и так как у нас всего три кнопки, нам нужно три тона:

```
#define play_c 3830
#define play_d 3400
#define play_e 3038
```

Мы взяли величину задержки с сайта Arduino, где подобраны готовые значения для каждой ноты. Теперь можно использовать константы **play_c**, **play_d** и **play_e** [в музыке, C, D и E – обозначения нот до, ре, ми, – *прим. ред.*] как аргументы функции **playTone** для задания требуемого тона. Следующее, что надо добавить – издание звука при вспышке соответствующего светодиода; мы можем сделать это с помощью **playSequence** – функции, которая зажигает светодиоды. Перескочив в середину этой функции, вы можете вспомнить, что на прошлом уроке мы использовали условие **case**: требуемый светодиод зажигался в зависимости от заданного значения в текущей позиции массива. Можно просто добавить команду **playTone** в каждое условие для проигрывания звука, когда загорается светодиод:

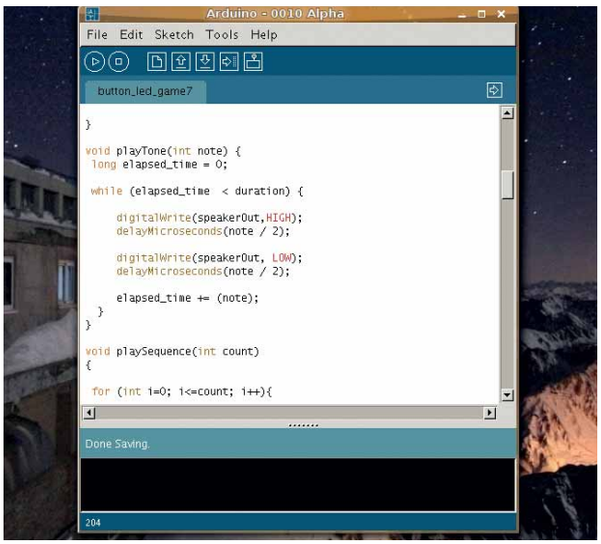
```
switch (rand_array[i]) {
  case 0:
    digitalWrite(ledPin1, HIGH);
    playTone(play_c);
    break;
  case 1:
    digitalWrite(ledPin2, HIGH);
    playTone(play_d);
    break;
  case 2:
    digitalWrite(ledPin3, HIGH);
    playTone(play_e);
    break;
  }
```

Как видите, мы задали три различных ноты для трех различных светодиодов, и они должны проигрываться так же, как и световая последовательность, которую должен скопировать игрок.

Теперь, обеспечив проигрывание, необходимо добавить похожую функцию в секцию ввода в коде, чтобы при нажатии игроком кнопки динамик выдавал соответствующий звук. В прошлом месяце мы создали функцию, названную **readButtons**, отвечающую за ввод. Вместо распределения всех нажатий кнопки по условиям **case**, мы использовали вложенные **if**, и сюда тоже надо добавить корректные значения **playTone**.

```
if (val1==LOW){
  playTone(c);
  return 0;
} else if (val2==LOW) {
  playTone(d);
  return 1;
} else if (val3==LOW) {
  playTone(e);
  return 2;
}
```

Наша программа научилась проигрывать ноты при нажатии кнопок. Только не перепутайте тоны: они должны совпадать в секциях программы для ввода и проигрывания, иначе игрок будет слышать различные ноты, когда он запоминает их и когда нажимает кнопки.



Все звуки в проекте генерируются с помощью функции playTone.

Вот и вся базовая звуковая функциональность. Осталось только одно дополнение, которое вы, возможно, захотите добавить – звуки реакции системы: при правильно введенной последовательности и при ошибке. Как вы можете помнить, у нас уже есть две функции, отвечающие за эти действия – **flashSuccess** и **flashFailure**, они обе были исключены из исходного кода по причине нехватки места. Вы можете захотеть добавить соответствующий унизительный звук провала, чтобы игрок осознавал свою ошибку – например, длинный низкий тон типа такого:

Мультиметры

Немного освоившись в играх с Arduino, вы можете наткнуться на одно полезное устройство – мультиметр [или тестер]. Это эквивалент инструментов отладки программиста для инженера-электронщика, и он неоценим, когда у вас что-то не заработает и надо выяснить, почему.

Он также является прекрасным инструментом для обучения, помогая вам увидеть, что и как происходит в вашей схеме. Другое хорошее качество мультиметра – его дешевизна, особенно для любителей, ведь базовые модели стоят несколько сотен рублей. Большинство имеют похожие режимы и функциональность, с круговым переключателем режимов и цифровым дисплеем [в России еще встречаются стрелочные индикаторы, – прим. пер.], отображающим выходные параметры. Мультиметром можно измерять напряжения, или сопротивления резисторов, но самая полезная функция для маленьких проектов Arduino – «проводимость». Это когда вы проверяете, соединены ли два компонента или участка цепи, подавая на них небольшое напряжение с мультиметра. Для этой цели Arduino должен быть отключен от источника питания – как и любая тестируемая схема – и тест проводимости генерирует звук, когда достаточное напряжение передается от одного сенсора к другому. На переключателе эта функция обычно изображается в виде ноты или динамика. Тест на проводимость – лучший способ проверить целостность ваших проводов, до перехода к тестированию отдельных компонентов.

Мультиметры бывают всяких форм и размеров, но большинство использует круговой переключатель для выбора функции и цифровой дисплей для отображения информации.

```
playTone(8000);
playTone(8000);
```

Но, может быть, вам захочется создать нечто более раздражающее. Звук, генерируемый динамиком, не обязан быть мелодичным, и можно устроить настоящую какофонию, быстро подавая ноты на динамик. Наша функция **playTone** имеет фиксированную задержку, и нам надо исправить это, если мы хотим менять звуки быстро. Решение – ввести еще один аргумент для длительности ноты. Для этого изменим функцию **playTone** следующим образом:

```
void playTone(int note, long note_len )
```

Необходимо также сменить значение **#define note_len 200000** в начале исходного кода на **#define duration 200000**, чтобы не путать имена переменных. Проделав этих два изменения, надо обеспечить, чтобы каждый раз при вызове **playTone** подставлялась длительность. Например:

```
playTone(play_c, duration);
```

Теперь у вас есть больший контроль над продолжительностью звука, генерируемого динамиком, и вы можете быстро передавать одну ноту за другой. Вот как мы заменили функцию **flashFailure** на устрашающий звук ошибки, вместо набора огоньков:

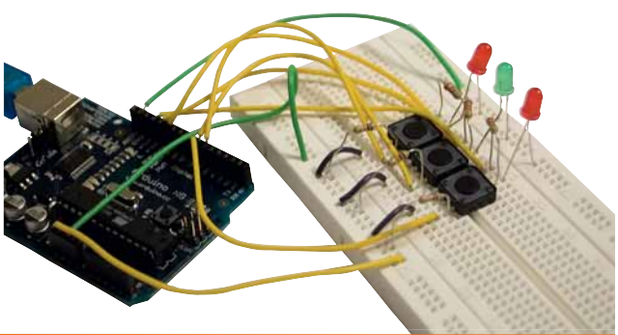
```
void flashFailure(){
for (int i=0; i<=10; i++){
  playTone (play_c, 10000);
  playTone (play_d, 10000);
  playTone (play_e, 10000);
}
}
```

Это просто цикл, который очень быстро проигрывает три ноты, имитируя битву лазерных пушек и ясно информируя игрока, что он сделал ошибку при попытке воспроизвести последовательность. Мы обнаружили, что при наличии звука можно обойтись без функции реакции на успех: функция ошибки достаточно информативна. Удалите **flashSuccess()** из **void loop()**, и увидите, что мы имели в виду.

Игра стала более интуитивной, и вы можете воссоздать последовательность нажатия кнопок гораздо быстрее, не дожидаясь всякий раз подтверждения. Со звуком весь проект доставляет больше удовлетворения, и хотя «бипы» немного упростили игру, зато вы можете повторять гораздо более длинные последовательности, особенно если у вас хорошая музыкальная память.

Через месяц

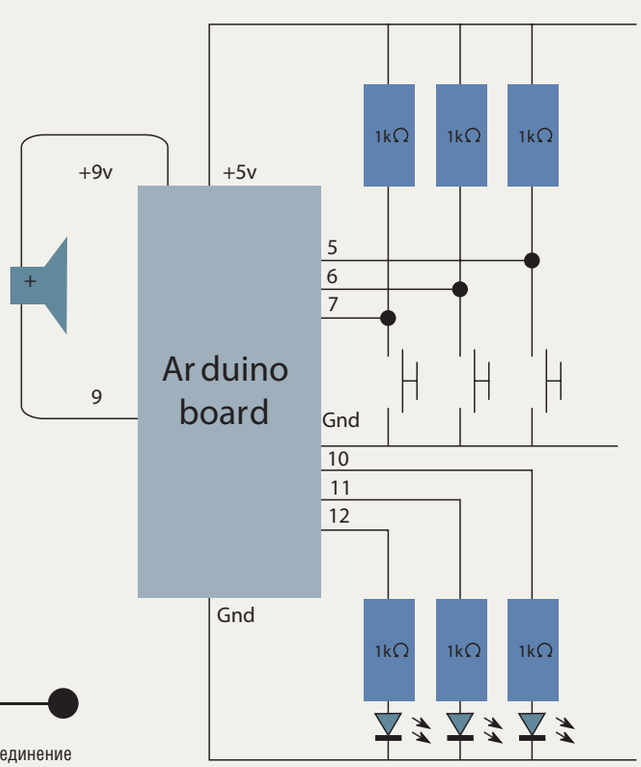
На следующем уроке мы сделаем другое важное дополнение к игровому механизму. Пока мы просто запускали световые последовательности, без каких-либо шансов узнать, выигрывает ли игрок, или уведомлений о том, как далеко он зашел на случай, если кто-то другой хочет добиться похожих результатов. Для решения данной проблемы мы добавим сохранение очков игрока и будем отображать их, когда игрок решит, что с него хватит. С этой целью мы запрограммируем семисегментный дисплей, напоминающий решения 1970-х, и используем его для вывода очков, а также для текущего статуса запущенной игры. До встречи, и удачи вам в аппаратном хакинге. LXF



Принципиальная схема

В прошлом месяце была упущена одна вещь, которая облегчает понимание устройства цепи. Для проектов, состоящих более чем из двух светодиодов, трудно описать словами, как все соединено. Мы собирались включить вам в помощь электросхему, но не хватило места. К счастью, нас выручил один из читателей: Стюарт Уоткисс [Stewart Whatkiss] прислал собственную схему проекта, и мы печатаем ее здесь, добавив динамик, чтобы отразить изменения урока этого месяца.

Даже если вы не видели таких схем со школьных времен, понять ее не трудно, и она очень упрощает работу по конструированию электрической цепи. Вы можете видеть различные выходные контакты платы Arduino, и как они подключены к резисторам (показанным с их сопротивлением в 1 КОм). Другие символы обозначают переключатели и светодиоды. Переключатели показаны как кратковременный разрыв цепи, а светодиоды – как перевернутые треугольники со стрелками, изображающими излучаемый свет, идущий от элемента. Еще одно устройство – динамик; ну, его-то легко узнать.



Arduino: Плюс

ЧАСТЬ 3: Прежде чем перейти к новому эксперименту, закончим проект. **Грэм Моррисон** завершит игру «Саймон сказал», добавив индикатор очков.

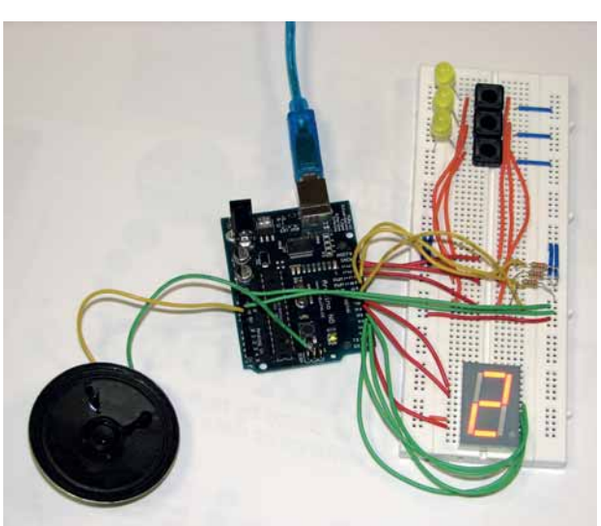
Наш эксперт

Грэм Моррисон
Разработчик свободного ПО, фанат KDE и энтузиаст электронной музыки. Он обсуждал синтезаторы не с одним ведущим членом сообщества FOSS.

Это третий, заключительный урок нашей серии «Создание игры “Саймон сказал” с помощью Arduino». Но не беспокойтесь, это не конец учебников по Arduino. Через месяц мы займемся совершенно другим. Игра у нас сейчас в отличной форме, с созвездием мигающих огоньков и шумным звуком. Но ваши успехи пока что можно оценить, только загибая пальцы при каждой верно угаданной мелодии. Для решения этой проблемы мы добавим счетчик очков, использовав новый компонент: классический семисегментный индикатор. Эти недорогие, технически несложные устройства ассоциируются с дешевой электроникой и применяются везде, от цифровых часов до микроволновых печей. Кроме того, они использовались во многих электронных играх из 70-х и 80-х, что добавляет ощущения подлинности нашему проекту.

Семисегментный индикатор

Существует несколько разновидностей семисегментного индикатора. Для нашего проекта мы использовали самые легкодоступные, известные под именем «с общим катодом». Другие виды могут не заработать (они способны менять полярность контактов), так что попытайтесь при возможности достать именно такой. Семисегментный индикатор состоит из, сами понимаете, семи отдельных элементов – черточек, из которых образуется цифра, и к каждому имеется контакт на задней стороне компонента. Таким образом, индикатор работает в точности как сборка из семи светодиодов. В зависимости от модели, вам также могут потребоваться семь резисторов для каждого соединения. К вящей путанице, многие дисплеи включают еще и восьмой элемент, который можно употребить как запятую, но мы его проигнорируем. Чтобы управлять семью элементами, для их включения и выключения нам понадобятся семь отдельных цифровых выходов на Arduino. К счастью, сосчитав использованные контакты, вы должны обнаружить, что у нас осталось ровно семь свободных. Прежде всего сдвинем соединения, чтобы их расположение стало чуть более логичным. Вот как мы сделаем:



› **Окончательную принципиальную схему игры см. на www.linuxformat.co.uk/mag/arduino.**

Семь элементов 0, 1, 2, 3, 4, 5, 6
Кнопки 7, 8, 9
Светодиоды 10, 11, 12
Динамик 13

Освободив место для индикатора на цифровом разъеме Arduino, следующим шагом воткните элемент на свободный участок макетной платы; удобно будет поместить его над центральной канавкой. Индикатор должен находиться над канавкой, а контакты окажутся по обе стороны от нее. Убедитесь, что вы видите, с какой дорожкой соединен каждый контакт, так как нам сейчас нужно подключить каждый контакт через дорожки к соответствующим выходам на Arduino. Семисегментные индикаторы имеют стандартную маркировку, где каждый элемент обозначен буквами A-G, двигаясь по часовой стрелке сверху. Сверьтесь с нашей диаграммой, как они помечены. Теперь подключайте каждый элемент, начиная с A, к соответствующему контакту на Arduino, начиная с 0 (нуля), так, чтобы задействовать все элементы. Кроме того, необходимо подключить GND (землю) макетной платы к GND на индикаторе. Вот и все нужные нам соединения. Теперь добавим программную логику для управления.

Рулим индикатором

Откройте исходный код нашего проекта и убедитесь, что переменные ledPin, inPin и speakerOut, присутствующие в верхней части файла, отражают новый порядок соединений Arduino. Теперь добавим переменные для индикатора, но сделаем это немного иначе, чтобы сэкономить место в журнале. Вместо семи различных чисел и семи присваиваний, используем массив, содержащий номер контакта для каждого сегмента. Преимущество такого подхода, помимо экономии чернил, заключается в том, что можно изменять порядок подсоединения контактов, просто корректируя значения в массиве.

```
/* DISPLAY ORDER: a,b,c,d,e,f,g */
int segPin[7]={0,1,2,3,4,5,6};
```

Кто не сталкивался с массивами ранее, представьте его себе как строку таблицы, причем размер массива определяется длиной строки. Сейчас мы собираемся создать еще один, только на этот раз он будет двумерным – похожим на электронную таблицу со строками и столбцами. Каждая строка содержит сигналы «вкл» и «выкл», чтобы зажигать необходимое количество сегментов на дисплее, и каждая колонка ссылается на определенный сегмент:

```
bool segNum[10][7]={
{1,1,1,1,1,1,0}, {0,1,1,0,0,0,0},
{1,1,0,1,1,0,1}, {1,1,1,1,0,0,1},
{0,1,1,0,0,1,1}, {1,0,1,1,0,1,1},
{1,0,1,1,1,1,1}, {1,1,1,0,0,0,0},
{1,1,1,1,1,1,1}, {1,1,1,1,0,1,1},
};
```

Нам надо хранить значения только «вкл» и «выкл», поэтому для двумерного массива выбран тип «bool». Это сокращение от Boolean [логический], то есть его значение может быть либо «истиной», либо «ложью» (1 или 0). Так мы сэкономим память, что особенно важно при работе с устройствами вроде Arduino, где ее всего ничего [еще большей экономии памяти можно добиться, закодировав

Счетчик очков

каждый элемент массива sigNum в виде двоичного числа, например, sigNum[0]=1111110b=126, т.е. символ нуля – прим.ред.]. Теперь инициализируем контакты, которые мы используем для индикатора – это можно сделать, добавив следующие строки в функцию setup():

```
for (int i=0; i<7; i++){
  pinMode(segPin[i], OUTPUT);
}
```

Цикл for перебирает массив segPin и инициализирует контакты, оответствующие текущему элементу массива. Итак, мы все настроили; осталось только написать функцию вывода нужных нам чисел. Назовем ее displayNum:

```
void displayNum (int number) {
  if (number <= 9){
    for (int i=0; i<7; i++){
      if (segNum[number][i]){
        digitalWrite(segPin[i], HIGH);
      } else {
        digitalWrite(segPin[i], LOW);
      }
    }
  }
}
```

Она проще, чем кажется. Первое условие if – это просто проверка на ошибку, чтобы убедиться, что мы не пытаемся вывести номера больше, чем 9 (индикатор не способен показывать такие числа). После этого мы перескакиваем в цикл for, который зажигает или гасит каждый элемент дисплея в соответствии со значениями, хранящимися в строке, определенной переменной number – ее значение мы передаем в функции. Все, что нам сейчас нужно сделать, это вызвать функцию с нужным числом очков.

Так как цифр можно изобразить всего 10, показывать число шагов на индикаторе нелепо: даже самый посредственный игрок сумеет заработать больше девяти очков. Вместо этого будем показывать нечто вроде уровней, деля число очков на три и выводя уровень на индикатор. В результате через каждые три успешные шага счет на дисплее будет увеличиваться на единицу – итого 27 шагов, чтобы добраться до девятого уровня; задача достаточно сложная. Для вывода уровня просто добавьте displayNum(score/3) в цикл if(winning) в конце нашей программы. Цикл станет таким:

```
if (winning) {
  displayNum(score/3);
  score++;
}
```

Вот и все, что нужно сделать – ну, почти все: не решен лишь один вопрос. Если вы подключили выход динамика к контакту 13, как говорилось в начале данного урока, вы уже могли заметить проблему: уровень звука на входе 13 еле слышим. А все потому, что контакт 13 на самом деле предназначен для тестирования: он уже содержит резистор, снижающий и ток, и громкость динамика. К счастью, в Arduino есть недокументированная функция, которая нам поможет. Контакт 3 разъема ICSP на плате Arduino (следующий после переключателя reset) – это еще один контакт 13, но без резистора; можете подключить динамик к нему и вновь обрести звук. Но будьте осторожны: если динамик по-прежнему подключен к 9-вольтовому источнику, короткое замыкание с одним из других контактов на разъеме ICSP может повредить вашу плату Arduino. Убедитесь, что питание отключено и ваш динамик не присоединен к другим контактам.

Наша простая игра «Саймон сказал» теперь полностью функциональна. Конечно, как и все великие сериалы, она еще только началась. Наш код нуждается в хорошей проверке на ошибки, и неплохо бы добавить еще одну кнопку для перезапуска текущей игры, так как нажатие reset на плате отнимает слишком много времени. А можете придумать, как перенести схему с макетной платы и установить ее в подходящий корпус, в стиле семейных развлечений 1970-х. Удастся – пришлите нам фотографию. LXF

Скорая помощь

Для увеличения уровня очков до 16 можно использовать шестнадцатеричный дисплей. Увеличьте размер segNum до [16][7], замените проверку на ошибки в displayNum на number<=15 и добавьте следующие строки к элементам массива: {1,1,1,0,1,1,1}, {0,0,1,1,1,1,1}, {1,0,0,1,1,1,0}, {0,1,1,1,1,0,1}, {1,0,0,1,1,1,1}, {1,0,0,0,1,1,1}

Декодируем индикатор

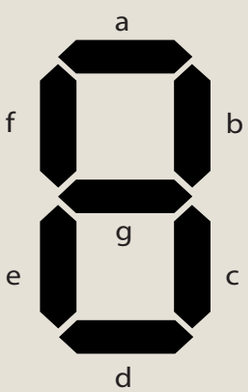
Иногда бывает трудно определить, какой элемент на семисегментном индикаторе к какому контакту подключен. Если у вас нет описания, то проще всего выяснить это методом проб и ошибок. Подключите индикатор к Arduino и напишите простую программу для запуска элементов в определенном порядке. Сделав задержку достаточно долгой, вы легко разглядите, какие именно элементы в настоящее время подключены к определенным контактам. Кстати, мы именно это и делаем в нашей программе – просто измените значения элементов массива на следующее (вместо символов, которые там были заданы):

```
bool segNum[10][7]={
{1,0,0,0,0,0,0}, {0,1,0,0,0,0,0},
{0,0,1,0,0,0,0}, {0,0,0,1,0,0,0},
{0,0,0,0,1,0,0}, {0,0,0,0,0,1,0},
{0,0,0,0,0,0,1}, {0,0,0,0,0,0,0},
{0,0,0,0,0,0,0}, {0,0,0,0,0,0,0},
};
```

Теперь замените основной цикл следующим:

```
for (int i=0; i<9; i++){
  displayNum(i);
  delay (300);
}
```

Тут перебираются все элементы индикатора, через длинные паузы. Без информации, куда подцеплен каждый элемент, последовательность на индикаторе покажется случайной, но если записать порядок, в котором зажигались элементы, вы сможете определить, какой контакт чем управлял, и либо перецепить провода, либо изменить порядок контактов в массиве segPin.



Через месяц Мы с Arduino будем бить в барабан, как макака в цирке.