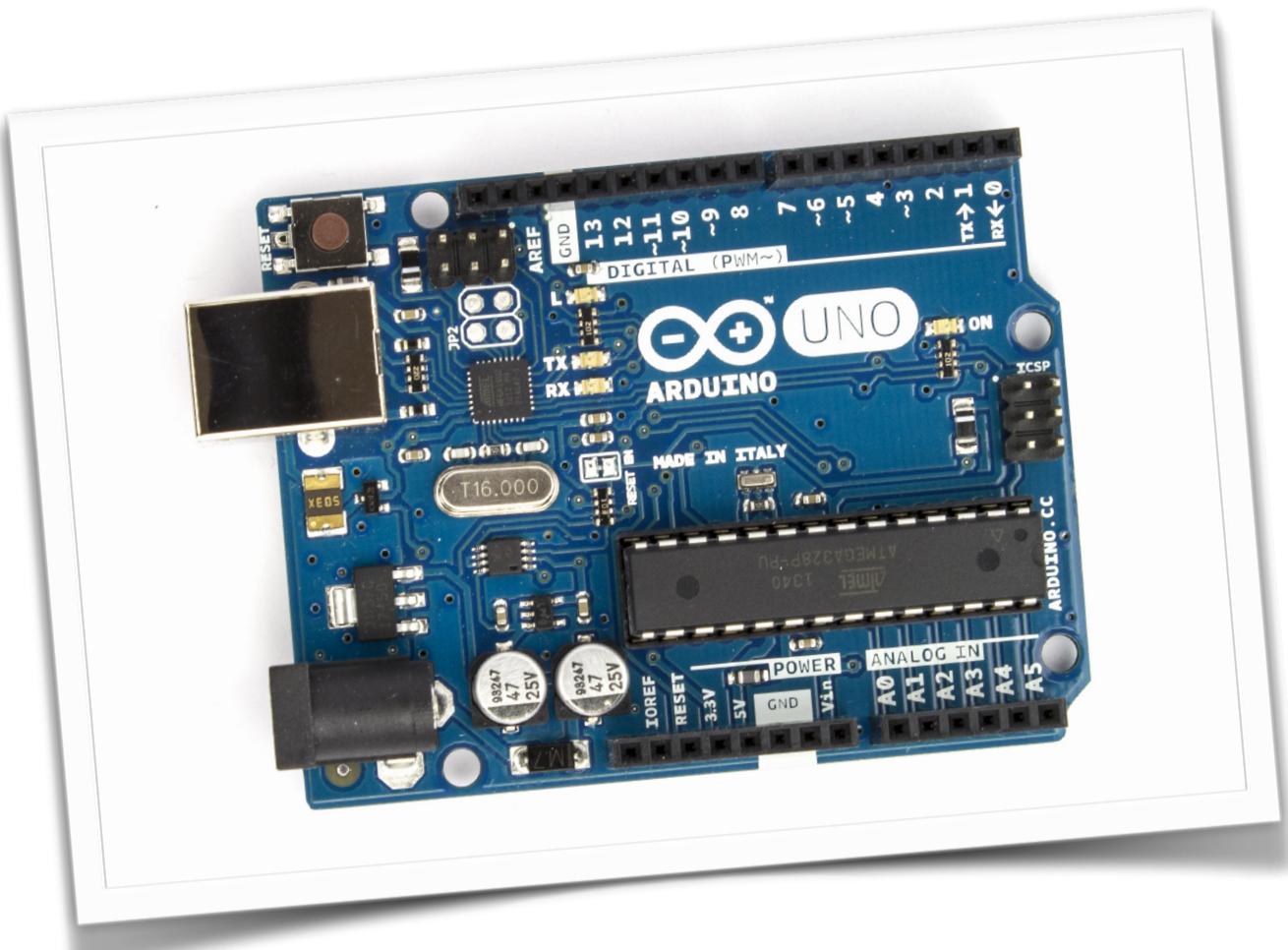


# Arduino

"На пальцах"



Или как студенту, не знающему  
ничего о микроконтроллерах,  
научиться программировать  
платформу Arduino с нуля.



Минск 2017

# Введение

Несмотря на все усилия учителей по физике, по окончании школы у большинства людей остаются в головах лишь смутные понятия о каких-то конденсаторах, резисторах и Вольтах с Омами, но связать эти знания воедино и извлечь из них практическую пользу никто толком не может :с

Разгадка этого парадокса ясна: на уроках физики в школе дают очень много теории (короче воды) при почти полном отсутствии практики. Вот и получается, что мало кто из школьников держал в руках настоящий транзистор илиставил опыт с зарядом конденсатора. А теория, не подкрепленная практикой, тяжело усваивается и очень легко забывается – на следующий же день после экзамена (это в лучшем случае).

Дабы облегчить усвоение материала я вводил ряд упрощений. Совершенно бредовых и антинаучных, но более менее наглядно показывающих суть процесса. Методика "канализационной электрики" успешно показала себя в полевых испытаниях, а посему будет использована и тут. Хочу лишь обратить внимание, что это всего лишь наглядное упрощение, справедливое для общего случая и конкретного момента, чтобы понять суть и к реальной физике процесса не имеющая практически никакого отношения. Зачем оно тогда? А чтобы проще запомнить, что к чему и не путать напряжение и ток и понимать как на все это влияет сопротивление, а то я от студентов такого наслушался...

В этой книжке - помощнике даются краткие теоретические сведения, которые ни в коем случае не могут заменить много страничные труды уважаемых авторов учебников и книг по электронике. Но зато этот текст написан доступно и увлекательно, поэтому может дополнить другие, более серьёзные работы (я пошутил, "канализационная электрика" ничего серьёзного дополнить не может)).

Макетная плата позволяет собирать первые простые схемы без пайки. Тебе не потребуется бегать в поисках паяльника и прочих инструментов. Ещё одно важное преимущество технологии сборки схем на макетной плате – из ограниченного набора деталей можно собрать больше десятка разных схем.

Но тем не менее, уметь паять должен каждый уважающий себя радиолюбитель, поэтому без советов по пайке не обойтись и здесь!)

Ну что, готов дерзать, рвать и метать? Ну, это я, конечно же, про изучение микроконтроллерной аппаратной платформы Arduino :) **Вперёд!**

\* В тексте присутствуют вставки с таким вот шрифтом. Это означает мысли в слух, или же какие то сноски, уточнения.

## Начало начал

Если сравнить электроцепь с канализацией, то источник питания это сливной бачок, текущая вода – **ток**, давление воды – **напряжение**, а несущиеся по трубам нечистоты – **полезная нагрузка**. Чем выше сливной бачок, тем больше потенциальная энергия воды, находящейся в нем, и тем сильней будет напор-ток проходящий по трубам, а значит больше нечистот-нагрузки он сможет смыть.

Кроме нечистот, потоку препятствует трение о стенки труб, образуя потери. Чем толще трубы тем меньше потери (ты теперь понимаешь почему аудиофилы для своей мощной акустики берут провода потолще ;)).

Итак, подведем **итог**. Электроцепь содержит источник, создающий между своими полюсами разность потенциалов – напряжение. Под действием этого напряжения **ток** устремляется через нагрузку туда, где потенциал **ниже**. Движению тока препятствует сопротивление, образуемое из полезной нагрузки и потерь. В результате напряжение-давление ослабевает тем сильней, чем больше сопротивление. Ну, а теперь, положим нашу канализацию в математическое русло.

**Сила тока в цепи пропорциональна напряжению и обратно пропорциональна полному сопротивлению цепи.**

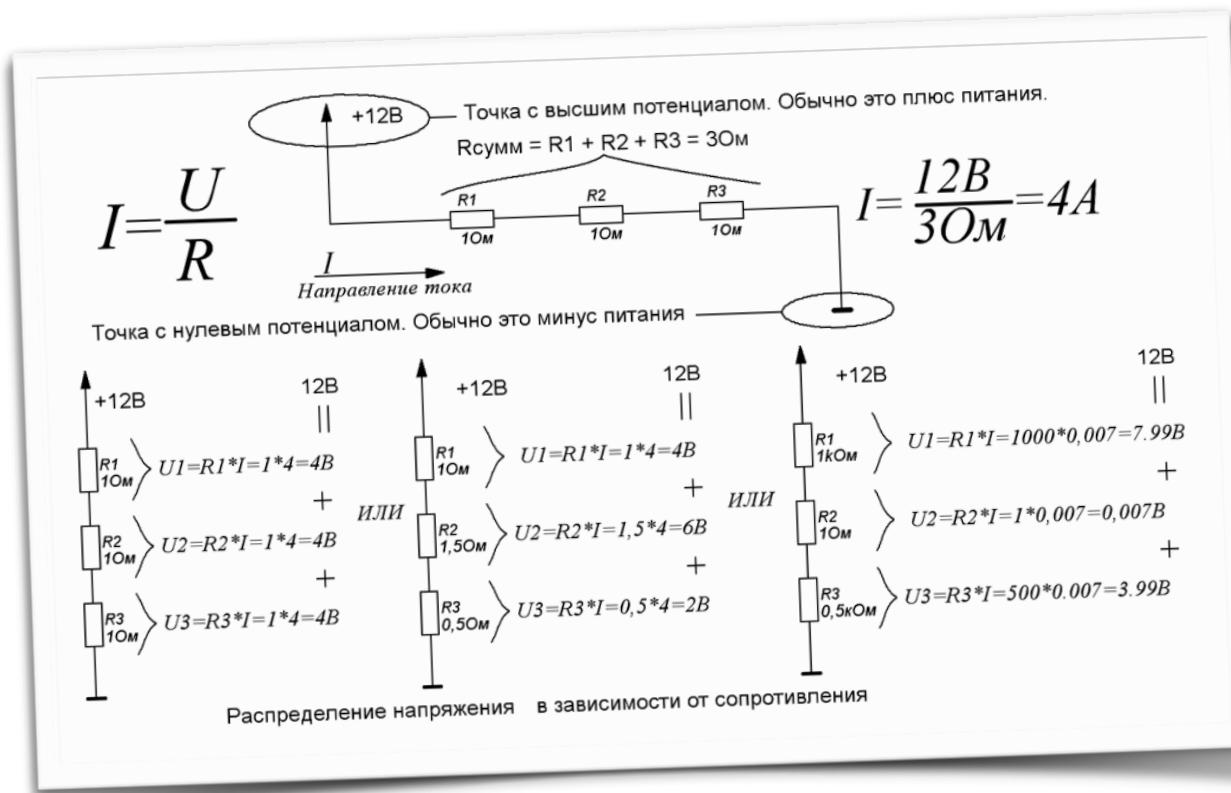
$$I = \frac{U}{R}$$

**U** – величина напряжения в вольтах.

**R** – сумма всех сопротивлений в омах.

**I** – протекающий по цепи ток.

Для примера просчитаем простейшую цепь, состоящую из **трех** сопротивлений и одного источника. Схему я буду рисовать не так как принято в учебниках по ТОЭ, а ближе

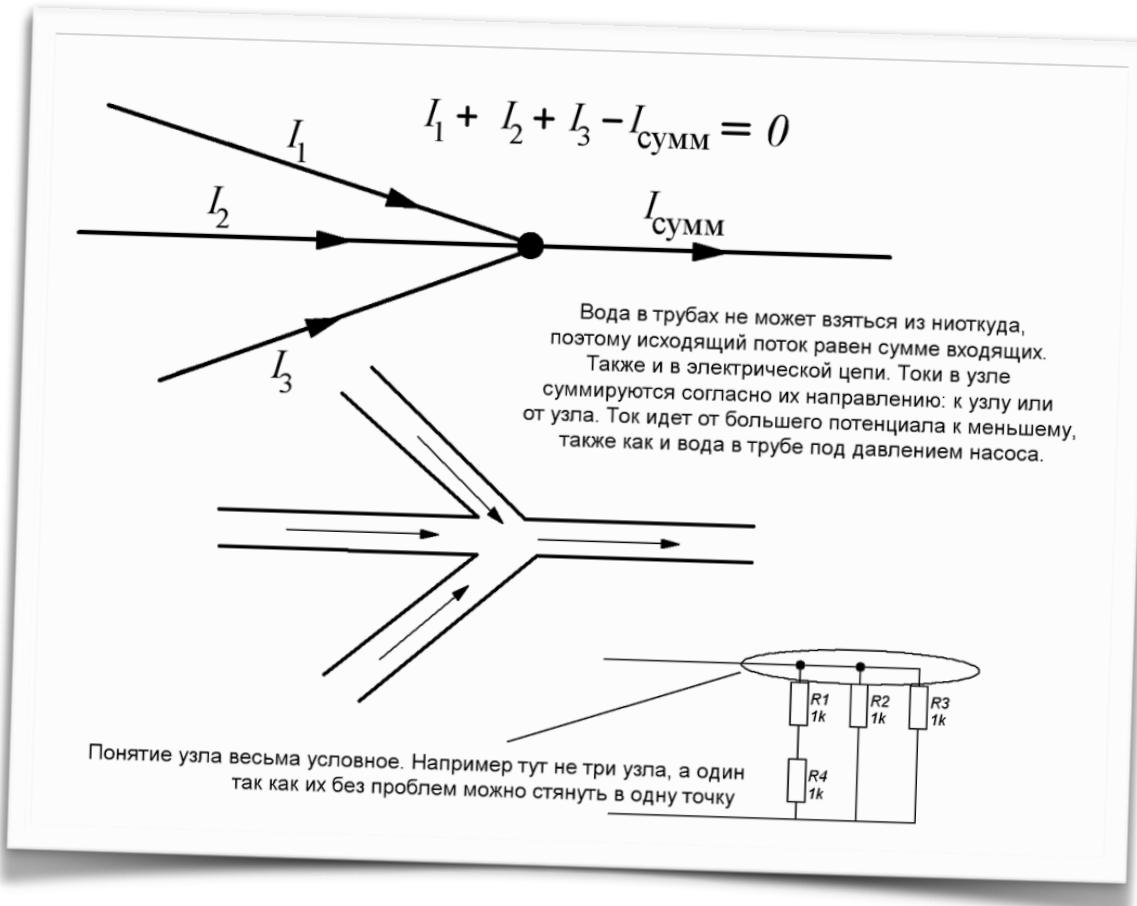


к реальной принципиальной схеме, где принимают точку **нулевого** потенциала – **корпус**, обычно равный минусу питания, а плюс считают точкой с потенциалом равным напряжению питания.

Для начала считаем, что напряжение и сопротивления у нас известны, а значит нам нужно найти ток. **Сложим** все сопротивления (о правилах сложения сопротивлений читай на врезке), дабы получить общую нагрузку и поделим напряжение на получившийся результат – ток найден! А теперь посмотрим как распределяется напряжение на каждом из сопротивлений. Выворачиваем закон Ома наизнанку и начинаем вычислять.  $U=I \cdot R$  поскольку ток в цепи един для всех последовательных сопротивлений, то он будет постоянен, а вот сопротивления разные. Итогом стало то, что  $U$  источника =  $U_1 + U_2 + U_3$ . Исходя из этого принципа можно, например, соединить последовательно 50 лампочек рассчитанных на 4.5 вольта и спокойно запитать от розетки в 220 вольт – ни одна лампочка не перегорит.

А что будет если в эту связку, в серединку, всандалить одно здоровенное сопротивление, скажем на КилоОм, а два других взять поменьше – на один Ом? А из расчетов станет ясно, что почти все напряжение выпадет на этом большом сопротивлении.

## Закон Кирхгоффа



Согласно этому закону **сумма** токов вошедших и вышедших из узла равна **нулю**, причем токи втекающие в узел принято обозначать с **плюсом**, а вытекающие с **минусом**. По аналогии с нашей канализацией – вода из одной мощной трубы разбегается по кучи мелких. Данное правило позволяет вычислять примерный потребляемый ток, что иногда бывает просто необходимо при расчете принципиальных схем.

## Мощность и потери

Мощность которая расходуется в цепи выражается как произведение напряжения на ток.

$$P = U * I$$

Потому чем **больше** ток или напряжение, тем больше **мощность**. Т.к. резистор (или провода) не выполняет какой либо полезной нагрузки, то мощность, выпадающая него это потери в чистом виде. В данном случае мощность можно через закон ома выразить так:

$$P = R * I^2$$

Как видишь, увеличение сопротивления вызывает увеличение мощности расходующееся на потери, а если возрастает ток, то потери увеличиваются в квадратичной зависимости. В резисторе вся мощь уходит в **нагрев**. По этой же причине, кстати, аккумуляторы нагреваются при работе – у них тоже есть внутреннее сопротивление, на котором и происходит рассеяние части энергии.

Вот для чего аудиофилы для своих сверхмощных звуковых систем берут толстенные медные провода с минимальным сопротивлением, чтобы снизить потери мощности, так как токи там бывают немалые.

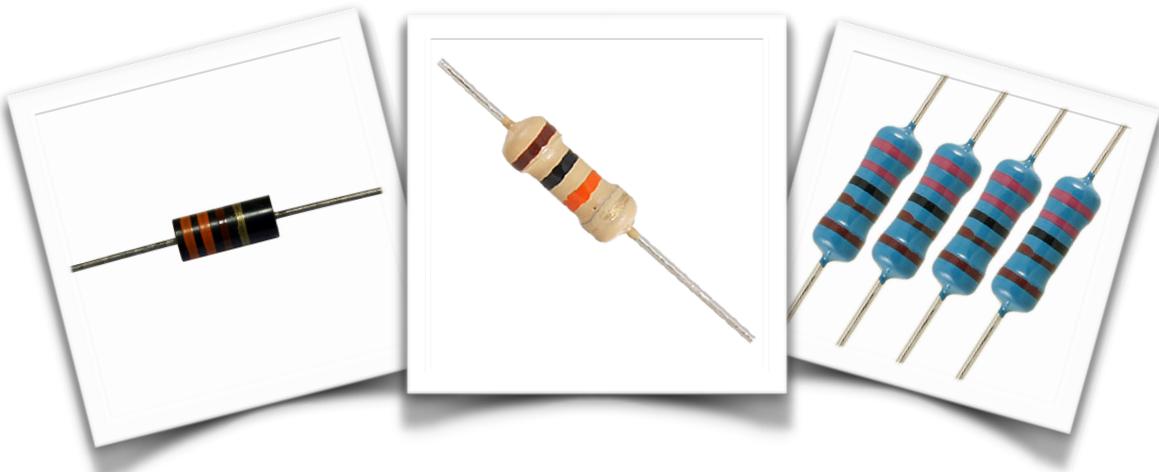
Есть закон полного тока в цепи, правда на практике мне он никогда не пригодился, но знать его не помешает, поэтому утюни из сети какой либо учебник по ТОЭ (теоретические основы электротехники) лучше для средних учебных заведений, там все гораздо проще и понятней описано – без ухода в высшую математику.

## Основные элементы электрических схем

**Основными** элементами электронных схем являются: Резистор, транзистор, диод, конденсатор и индуктивность. Обо всём по порядку.

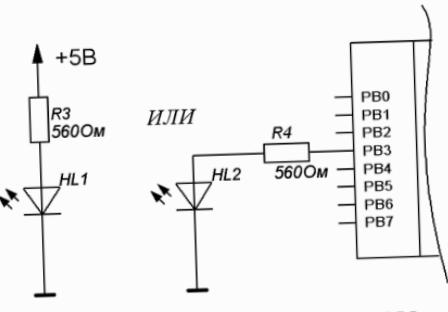
### Резистор

Он же **сопротивление**, на схеме выглядит белым узким прямоугольником (на буржуцких схемах часто обозначен угловатой пружинкой) – замечательная деталь! Отличается тем, что **не делает** вообще **ничего**. Тупо **потребляет** энергию и греется на этом. Основное предназначение в схеме это либо токоограничение, либо

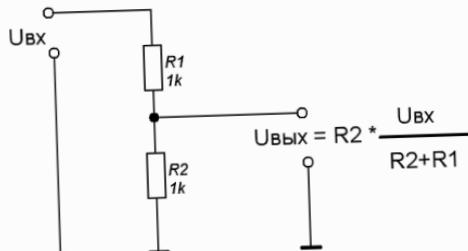


перераспределение напряжения.

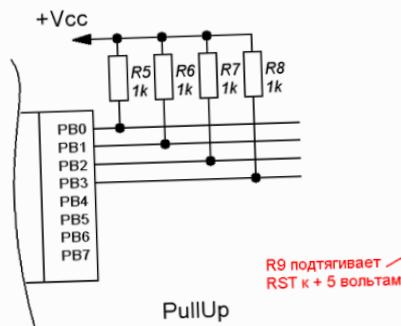
Непонятно? Сейчас поясню. Вот, например, светодиод. Ему для работы нужен мизерный ток, порядка 20 миллиампер, но вот беда – его сопротивление мало, поэтому



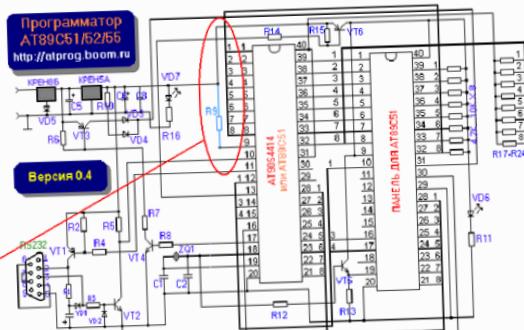
Ограничитель тока для светодиода



Простейший делитель напряжения



Подтяжка выводов до нужного напряжения



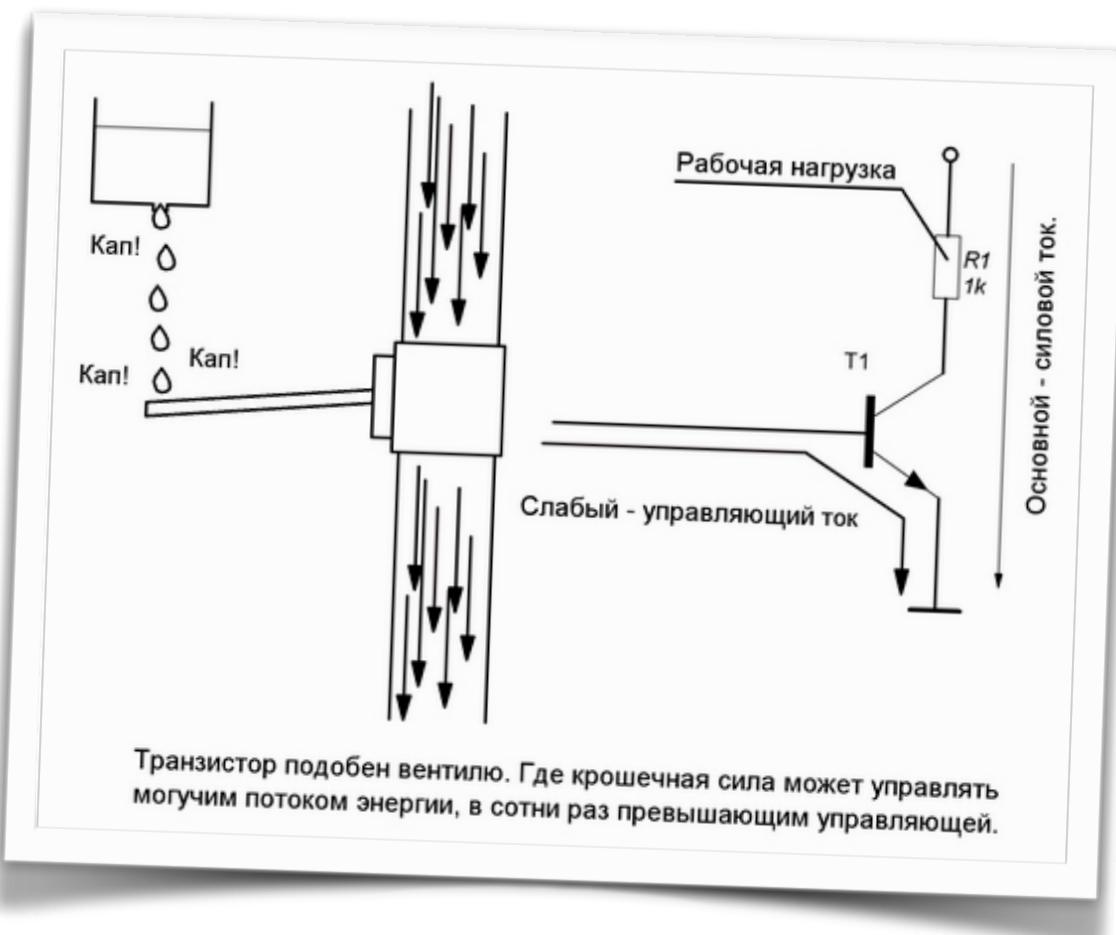
если его воткнуть напрямую в 5 вольт, то через него ломанётся ток в 400 миллиампер. От такой нагрузки бедняжка пожелтеет, позеленеет, а потом и вовсе загнется, источая вонь. Что делать? Правильно – поставить последовательно ему **резистор**, чтобы он ограничил ток, не пустив излишнюю мощность на хилый диодик. Даже если диод теперь тупо закоротить, то ток в цепи не превысит того, который разрешит резистор, исходя из закона Ома.

Второе популярное применение это **делители напряжения**. Цель делителя – **разделить входное напряжение** пропорционально номиналам резисторов и подать часть этого напряжения в нужную точку схемы. Это часто приходится делать при согласовании между сигналами разных напряжений. Делитель представляет из себя два последовательно соединенных резистора. Один из которых подсоединен к точке нулевого потенциала (корпус), а второй к напряжению которое нужно поделить. Средняя точка между резисторами это выход нашего поделенного напряжения. Ток в последовательной цепи везде одинаков, а вот сопротивление разное, а значит напряжение (по закону Ома) разделится на резисторах **пропорционально** их сопротивлениям. Однаковые резисторы – напряжение пополам, а если нет, то уже надо вычислять где как.

# Транзистор

Жуткая вещь, в детстве все не мог понять как он работает, а оказалось все просто. В общем, транзистор можно сравнить с управляемым **вентилем**, где крохотным усилием мы управляем мощнейшим потоком. Чуть повернул рукоятку и всё ненужное умчалось по трубам, открыл посильней и вот уже все вокруг захлебнулось в нечистотах. Т.е. выход пропорционален входу умноженному на какую то величину. Этой величиной является **коэффициент усиления**. Делятся эти девайсы на **полевые и биполярные**.

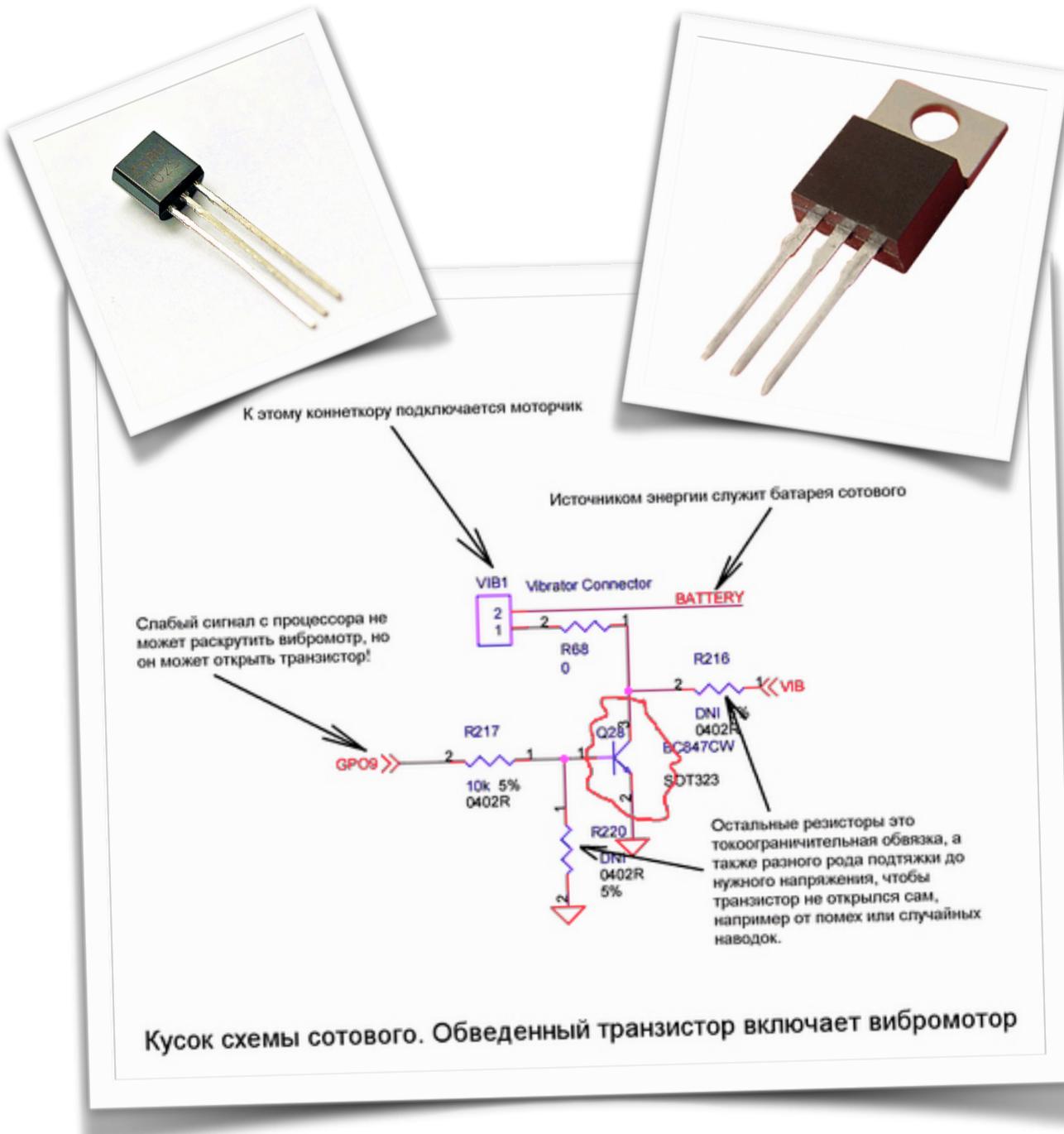
В **биполярном** транзисторе есть эмиттер, коллектор и база (смотри рисунок условного обозначения). Эмиттер он со стрелочкой, база обозначается как прямая площадка между эмиттером и коллектором. Между эмиттером и коллектором идет большой ток полезной нагрузки, направление тока определяется стрелочкой на эмиттере. А вот между базой и



эмиттером идет маленький управляющий ток. Грубо говоря, величина управляющего тока влияет на сопротивление между коллектором и эмиттером. **Биполярные** транзисторы бывают **двух типов: p-n-p и n-p-n** принципиальная разница только лишь в направлении тока через них.

**Полевой** транзистор отличается от биполярного тем, что в нем сопротивление канала между истоком и стоком определяется уже не током, а напряжением на затворе. Последнее время полевые транзисторы получили громадную популярность (на них построены все микропроцессоры), т.к. токи в них протекают микроскопические, решающую роль играет напряжение, а значит потери и тепловыделение минимальны.

Обозначение транзисторов или камень преткновения всех студентов. Как запомнить тип биполярного транзистора по его условной схеме? Представь что стрелочка это направление твоего движения на машине... Если едем в стенку то дружный вопль "Писец Нам Писец"



Короче, транзистор позволит тебе управлять слабеньkim сигналом, например с ноги микроконтроллера, управлять мощной нагрузкой типа реле, двигателя или лампочки. Если не хватит усиления одного транзистора, то их можно соединять каскадами – один за другим, все мощней и мощней. А порой хватает и одного могучего полевого **MOSFET** транзистора. Посмотри, например, как в схемах сотовых телефонов управляется виброзвонок. Там выход с процессора идет на затвор силового MOSFET ключа.

## Диод

Это такая хитрая фиговина, **пропускающая ток только в одну сторону**. Его можно сравнить с **ниппелем**. Применяется, например, в выпрямителях, когда из переменного тока делают постоянный. Или когда надо отделить обратное напряжение от прямого. Погляди в схему программатора (там где был пример с делителем). Видишь стоят диоды, как

The diagram illustrates the properties of a diode through several components and graphs:

- Visuals:** Two photographs of diodes are shown: one black diode labeled "P6" and one orange diode.
- Analogy:** A comparison is made between a diode and a nipple, stating that it only lets current flow in one direction.
- Circuit Diagrams:** Two simple circuit diagrams are shown. The first circuit, labeled "Bat1 12V", contains a battery, a diode D1, and a resistor R1. The second circuit, labeled "Bat2 12V", contains a battery, a diode D2, and a resistor R2.
- Graphs:**
  - A graph titled "Диод - тот же ниппель, только электрический" shows a sine wave representing "Напряжение" (Voltage) on the vertical axis and "Время" (Time) on the horizontal axis. The wave oscillates above and below zero. A horizontal dashed line represents the "Номинальное напряжение стабилизации стабилитрона" (Nominal stabilization voltage of a Zener diode). The portion of the wave above this line is shaded, while the portion below is labeled "D3" with an arrow pointing to the diode symbol, indicating it is reverse-biased.
  - A graph showing the output signal "К девайсу" (To device) versus "Время" (Time). It shows the original sine wave being clipped at the positive peak by the diode, resulting in a square-wave-like pulse train.
  - A graph showing the output signal "Входное" (Input) versus "Время" (Time). It shows the original sine wave being clipped at the negative peak by the diode, resulting in a square-wave-like pulse train.
- Textual Description:** A note states: "Диод пропустил через себя только положительную полуволну переменного сигнала. Все, что было ниже нуля (т.е. шло в другом направлении) "заязло" на диоде." (The diode only passed the positive half-wave of the AC signal. All that was below zero (i.e., went in the other direction) "stuck" on the diode.)
- Stabilizer Diagram:** A graph shows a Zener diode (stabilizer) connected in series with a resistor and an input voltage source. The output voltage is constant at the Zener voltage level, indicated by a horizontal dashed line. The text "Номинальное напряжение стабилизации стабилитрона" (Nominal stabilization voltage of a Zener diode) is labeled near the top of the graph.
- Text:** A note explains: "При превышении напряжения сверх номинала стабилитрон откроется и излишек напряжения уйдет на землю. Резистор нужен, чтобы стабилитрон не сгорел при этом - для ограничения тока." (When the voltage exceeds the nominal value, the Zener diode will turn on and the excess voltage will go to ground. A resistor is needed so that the Zener diode does not burn out during this - for current limitation.)

думаешь, зачем? А все просто. У микроконтроллера **логические уровни** это **0 и 5 вольт**, а у СОМ порта единица это минус 12 вольт, а ноль плюс 12 вольт. Вот диод и **отрезает** этот минус 12, образуя 0 вольт. А поскольку у диода в прямом направлении проводимость не идеальная (она вообще зависит от приложенного прямого напряжения, чем оно больше,

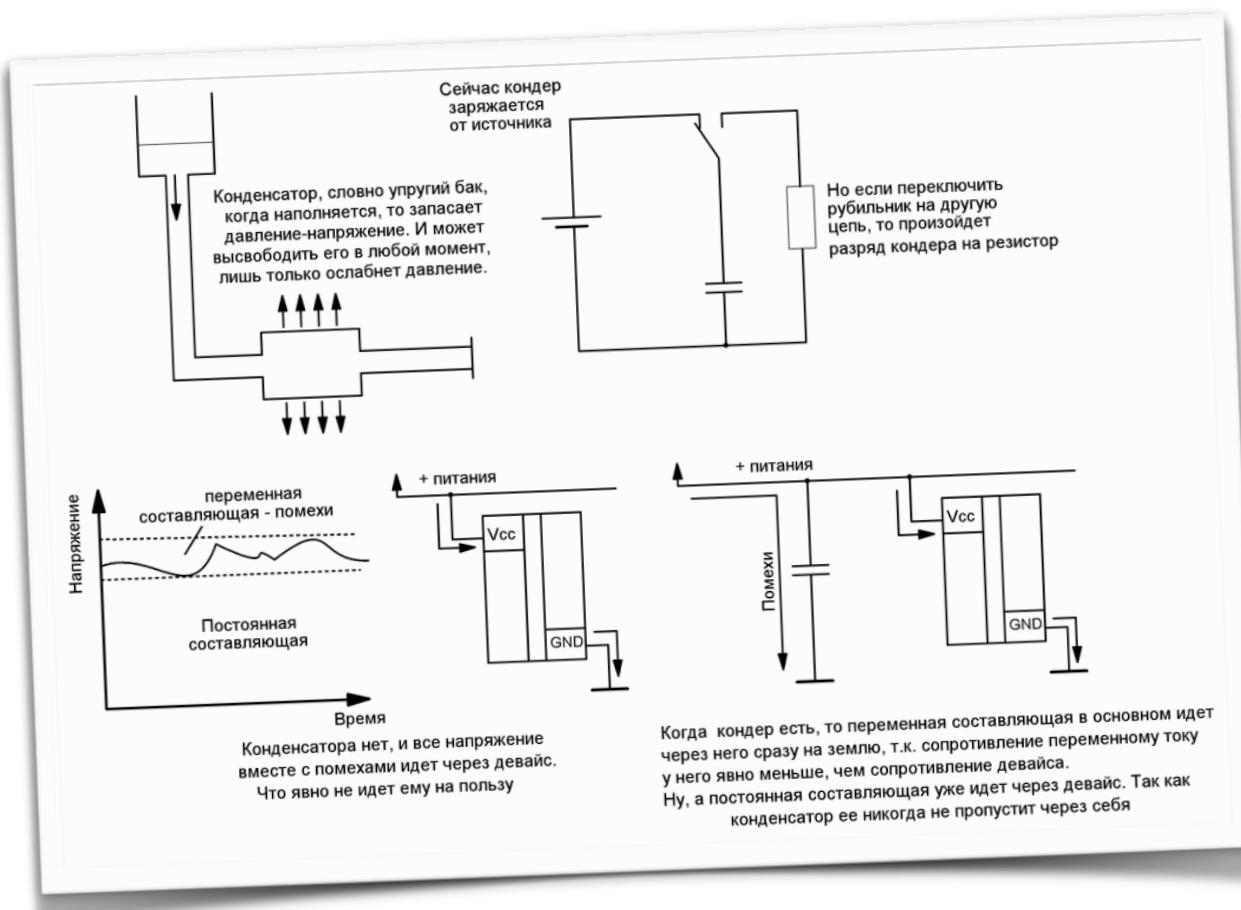
тем лучше диод проводит ток), то на его сопротивлении упадет примерно 0.5-0.7 вольта, остаток, будучи поделенным резисторами надвое, окажется примерно 5.5 вольт, что не выходит за пределы нормы контроллера.

Выводы диода называют **анодом и катодом**. Ток течет от анода к катоду. Запомнить где какой вывод очень просто: на условном обозначении стрелочка и палочка со стороны катода как бы рисуют букву К вот, смотри —К|—. К= Катод! А на детали катод обозначается полоской или точкой.

Есть еще один интересный тип диода – **стабилитрон**. Особенностью его является то, что в прямом направлении он работает как обычный диод, а вот в обратном его срывает на каком либо напряжении, например на 3.3 вольта. Подобно ограничительному клапану парового котла, открывающемуся при превышении давления и стравливающему излишки пара. Стабилитроны используют когда хотят получить напряжение заданной величины, вне зависимости от входных напряжений. Это может быть, например, опорная величина, относительно которой происходит сравнение входного сигнала. Им можно обрезать входящий сигнал до нужной величины или используют его как защиту. Также есть такой зверь как **супрессор**. Тот же стабилитрон, только куда более мощный и часто двунаправленный. Используется для защиты по питанию.

## Конденсатор

Он же **емкость** – еще один вид пассивных элементов. На схеме обозначен как две одинаковые параллельные черточки. В отличии от резистора, конденсатор это **нелинейный** элемент. По нашей канализационной аналогии его можно сравнить с **резиновым баком**. Вначале, когда он пуст, вода резко его заполняет, растягивая стенки.



Постепенно, когда стенки растянутся до предела, его сопротивление возрастет настолько, что поток воды остановится. А если убрать внешнее давление, то хлынет обратно.



Так же и электрический конденсатор, когда он не заряжен, то его сопротивление можно принять **нулю**, а вот когда зарядится, то бесконечностью, обрывом. Ток через него идет только лишь в момент **заряда или разряда**. После отсоединения источника тока конденсатор сам начинает действовать как **источник**, пока не разрядится.

Конденсаторы в электронике в основном используют как **фильтрующие элементы**, удаляющие помехи. Здоровенные конденсаторы на силовых цепях в блоках питания служат для подпитки системы при пиковых нагрузках, **сглаживая** просадки напряжения. Основан этот эффект на том, что конденсатор не пропускает постоянный ток, вот переменная составляющая через него проходит на ура. Сопротивление конденсатора переменной составляющей тока зависит от частоты этой составляющей. Чем выше частота, тем меньше сопротивление конденсатора. В итоге, все высокочастотные помехи, идущие поверх постоянного напряжения, глушатся через конденсатор на землю, оставляя после себя чисто постоянное напряжение. Сопротивление конденсатора переменной составляющей также зависит и от емкости кондера, поэтому ставя конденсаторы с разной емкостью можно отсеять разные частоты.

Емкостное сопротивление рассчитывается так:

$$X_C = 1/W * C$$

Где  $X_C$  – емкостное сопротивление в омах

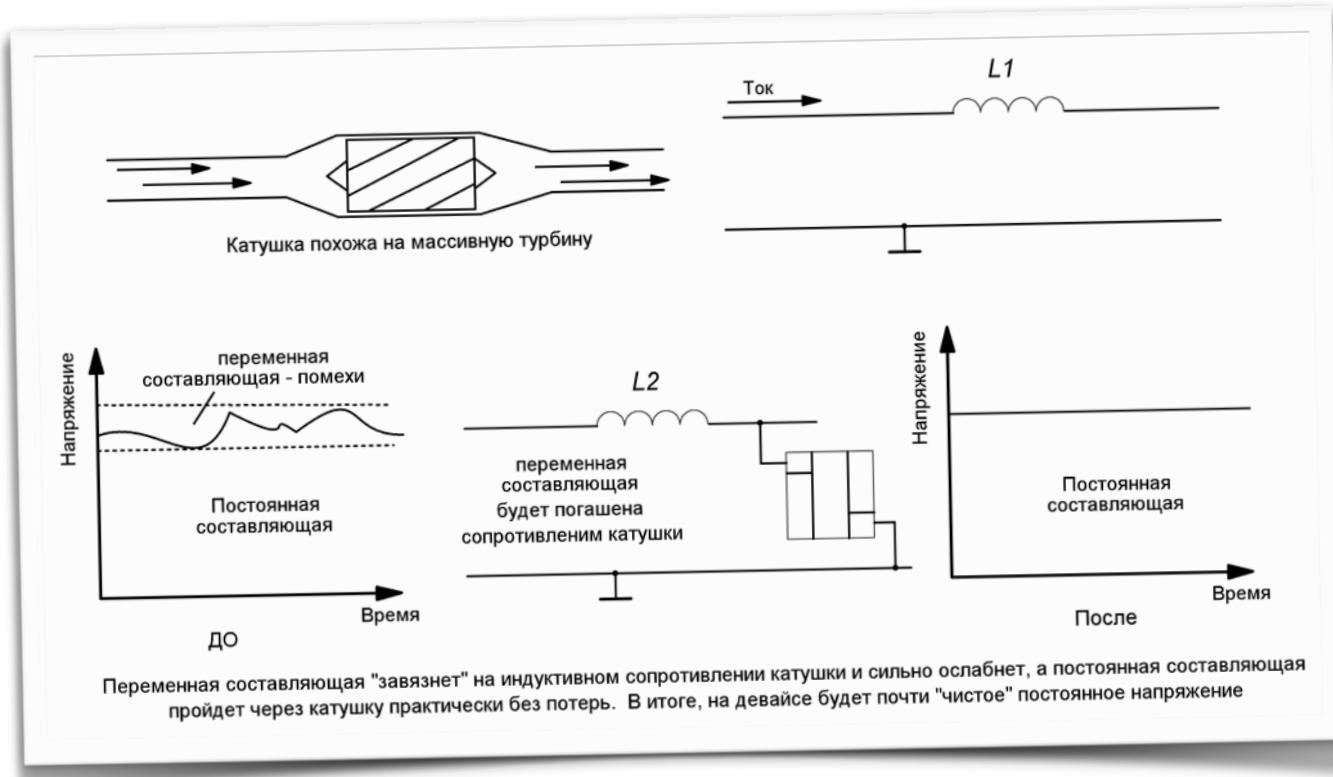
$C$  – емкость в фардах

$W$  – угловая частота переменной составляющей в радиан/с

Конденсатор может служить времязадающим элементом в разного рода **генераторах** – от него будет зависеть **частота** генерации, либо в качестве формирователя **импульса**.

## Индуктивность

В народе **катушка**, грубо говоря, это кусок проволоки намотанный на каркас. В эту группу входят и **дроссели** и разного рода **фильтры**, а также некоторые **антенны**. Также индуктивностью обладает всё, что имеет обмотку, несмотря на то, что это не главное свойство, например двигатели или электромагниты. А значит это надо будет учитывать при проектировании цепей. Увязать индуктивность в нашу канализационную теорию было нелегко, но немного пораскинув мозгами мы таки придумали.



В гидро модели катушка похожа на турбину с неслабой инерцией, где величина инерции является прообразом индуктивности. На стабильно текущий поток турбина, будучи раскрученной этим же потоком, не влияет никак, но стоит потоку ослабнуть, как турбина начнет за счет своей инерции подталкивать его. И наоборот, если турбина остановлена, то при появлении потока она будет его тормозить, пока не раскрутится. Чем **больше** инерция, тем сильней будет **сопротивление** изменению потоку.



Так и катушка индуктивности препятствует изменению тока, протекающего через неё. Основное применение катушки в колебательных контурах генераторов и в фильтрах. Т.к. катушка имеет отличное свойство пропускать через себя постоянную составляющую и подавлять переменную. В паре с конденсатором они образуют отличный Г или П образный фильтр.

## От теории к практике

Я рад тому, что многие поддержали проект-факультатив «**Arduino на пальцах**» и дали добро на создание данной книжечки. Я надеюсь это будет очень увлекательно. **Arduino** - это открытая платформа с готовым микроконтроллером, на котором можно создавать разнообразные проекты такие как: домашняя метеостанция, выключение-включение диодов в зависимости от освещения, управление приборами в доме (умный дом), роботов по типу машинок и кранов, приборы для измерения радиации, спектра частот и т.п.



**Обо всём** об этом мы и будем говорить подробней и здесь, и вживую на факультативе. Всё это будет подаваться для вас в очень **весёлой** или просто **интересной** форме, так сказать от студента к студенту, и не важно, работали ли вы с arduino раньше и уже знаете кое-что из радиоэлектроники(радиотехники) или вы новичок, который слышит и видит всё это в первый раз, но вы всё равно сможете узнать для себя здесь много интересного. Так же, помимо arduino, многие студенты смогут узнать немного теории во всё той же шуточной и интересной форме по предметам, которые имеют место быть на факультете **Радиофизики и Компьютерных Технологий БГУ**, и которые у тех же самых студентов будут в будущем.

Всё сделано для вашей же благополучной учёбы, чтобы вам было учиться и познавать радиофизику интересно и легко!)

## Необходимости

Что же нам может понадобиться для того, чтобы помимо теории, которой было тут написано уже немало, начать работать с платформой arduino? Правильно! детали (хе-хе), вот одни из них:

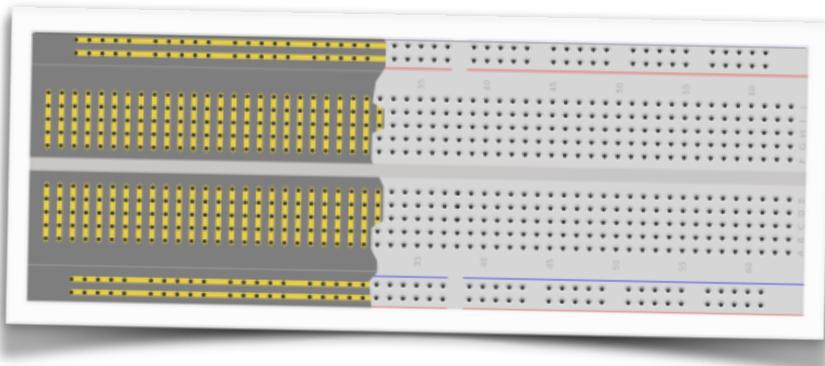
1. Микроконтроллер Arduino. Их бывает несколько типов, подробнее про каждый мы поговорим по отдельности, но в основе основ лежит Arduino Uno. Она та нам и нужна.



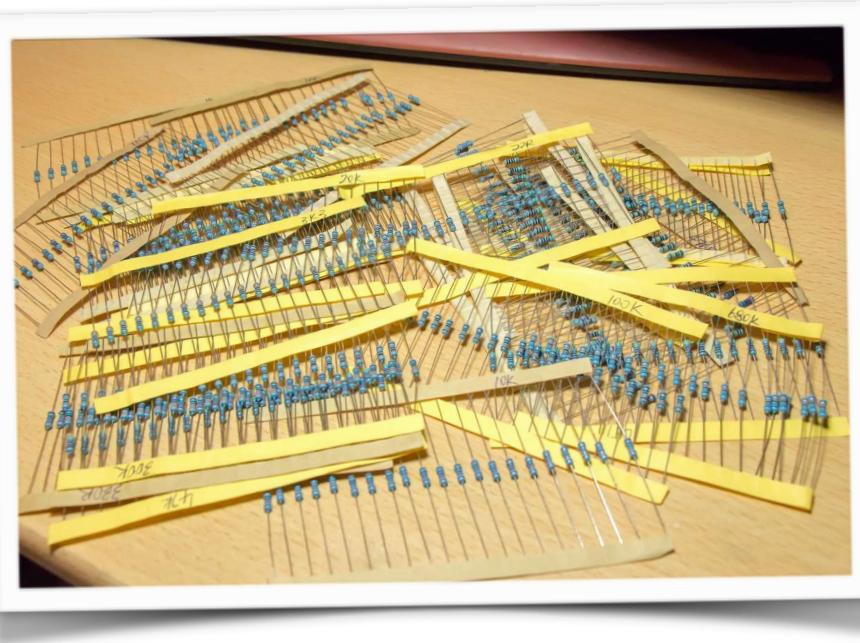
2. Кабель для подключения той самой железяки к компьютеру.  
Чорт, он слишком сильно похож на утку, ору не могу, остановите меня



3. Макетная плата, или, как говорят уже знающие радиолюбители Breadboard. **Запомни как соединяются между собой контактные площадки!!!**



4. Набор резисторов, пару транзисторов, немного конденсаторов и светодиодов. Так же, возможно, по желанию студентов, может понадобиться небольшая пьезо-пищалка, чтобы сделать свою музыку с помощью данного микроконтроллера (а это, скажу я вам, очень прикольно).



5. Ноутбук или компьютер, чтобы можно было программировать нашу Arduino  
ну, конечно же, я изображу здесь макарский MacBook (хотеть \*\_\*)



6. Блокнотик или небольшая тетрадка, но это не обязательно, чтобы в случае чего записывать или код, или что-либо ещё. (Немножко листов для записей будет в конце книги)
7. Особое желание постичь данную шалам-балам приблуду :)

**Вроде всё. Теперь можем начать.**

## Но сначала...

И так, в общем ещё раз: Arduino – это **открытая** source платформа, позволяющая создавать различные устройства и приборы, которые человек может использовать в своих или чьих-то нуждах и т.п.

А т.к. платформа открытая, то это значит, что производители **выложили все исходники** плат, контактов и материалов по данному микроконтроллеру на своём сайте в открытый доступ. Если вы захотите, то можете сделать собственную arduino по схемам, взятым с официального сайта, прошивать её через ISP программатор, но... разве оно вам надо? Когда в наше время китайцы свободно продают добротные и дешевые клоны arduino на aliexpress'е и купить их за гроши может каждый))

По этой причине если вы увидите Arduino Uno с разными микроконтроллерами (в версии ЧИП и ДИП), то не переживайте, всё нормально) значит у вас клон. На заметку, **оригинальные** платы arduino стоят по **40\$** за штуку.

Я, в настоящее время, третий раз помогаю нашему университету с ардуино проектами: делаю проект, отдаю его нужным людям (привет Иак), а они, в свою очередь, едут с этим проектом по областным городам в школы, чтобы там завлечь и сагитировать будущих абитуриентов поступать на факультет РФИКТ.

*Да, возможно, благодаря мне вы и решили поступать на РФИКТ)*

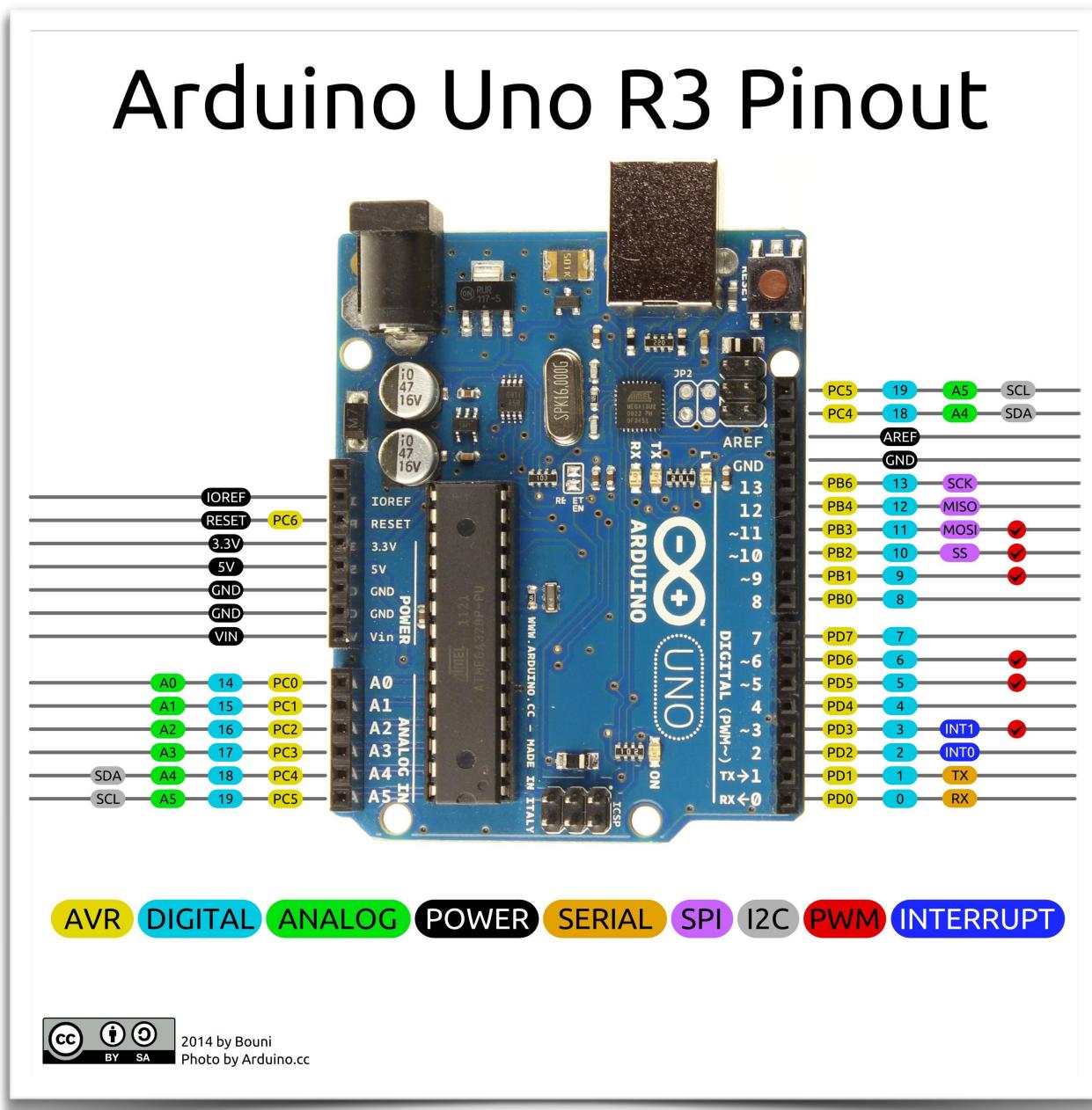
Arduino построены на микроконтроллерах от фирмы Atmel, и называются (в нашем случае) ATMega328, имеют 32 кБайта flash памяти, 32 пина подключений, работают на частоте 20 кГц, и ядро имеет 8-ми битную архитектуру. Скажете мало?

*(Ойой, мой компьютер может больше, фигня эта ваша Arduino, отписываюсь)*

А вот не тут то было! Не торопитесь делать выводы, сейчас вы всё поймёте.

# Строение Arduino Uno

Давайте на примере Arduino Uno **разберём** всё же, что из себя она представляет. Что у неё есть и чем она может похвастаться:



Как вы можете заметить, все порты, находящиеся на arduino можно разделить на **3 колонны**:

1. Цифровые порты (13 штук).
2. Аналоговые порты (6 штук).
3. Порты питания (6 штук).

**Цифровые** порты, как и аналоговые, нужны для того, чтобы **подключать** различные цифровые датчики, значения которых, в основном может принимать значения **0 и 1 (0 и 255 градаций)**. Так же некоторые цифровые порты могут контролировать количество принимаемых значений посредством **ШИМ** портов (это 3, 5, 6, 9, 10 и 11 порты). На 13 порту, кстати, имеется встроенный светодиод :)

**Аналоговые** порты нужны для того же самого, отличия лишь в том, что они могут контролировать принимаемые и передаваемые данные в пределах от **0** до **1023** градаций - то есть более точные, чем цифровые порты, но их меньше на плате чем цифровых по причине поедания микроконтроллером огромного количества вычислительных ресурсов.

### Миром правит цифра.

Ну и остались порты питания. Нужны для того, чтобы подводить питание к датчикам (не к arduino, у неё **отдельное** питание через **usb**). Имеются такие контакты как **5v** - источник напряжения в 5 вольт, **GND** - она же земля и она же минус питания, **3v3** - 3.3 вольта питания для очень требовательных датчиков.

Если нам необходимо подключить датчик к питанию, необходимо взять 2 проводка: **5v** и **GND**, и подсоединить их к соответствующим контактам на датчике, которые так же могут называться **Vcc** и **GND**, или же **VO** и **GND**.

## Время программировать!

Наконец-то мы можем взять ноутбуки и начать программировать на языке Arduino.

**Программировать** ардуино можно **двумя** способами:

1. Программирование через **ISP** программатор. Но этот метод программирования не очень удобный, да и чистые плюсы (C programming language) или assembler знает далеко не каждый, хотя нынешние второкурсники (**немногочисленные, к сожалению**), могут похвастаться тем, что уже немного освоили этот язык)

2. Программирование через **специальную** **программу** со своим собственным языком, которую можно скачать на сайте (<https://www.arduino.cc>) , ссылка на сайт также присутствует в группе факультатива вконтакте: ([https://vk.com/arduino\\_on\\_fingers](https://vk.com/arduino_on_fingers)).

Данный язык, как я его считаю, является бегемотом: **смесью** языков **C++** и **Java.Script** - уж больно стиль этих языков похож и слит в единое целое в этой программе. Программировать на этом языке может даже пятиклассник, уж на столько он **простой**, поэтому мы не будем выпендриваться и входить в исключение и сами возьмёмся за него.



## Первая программа

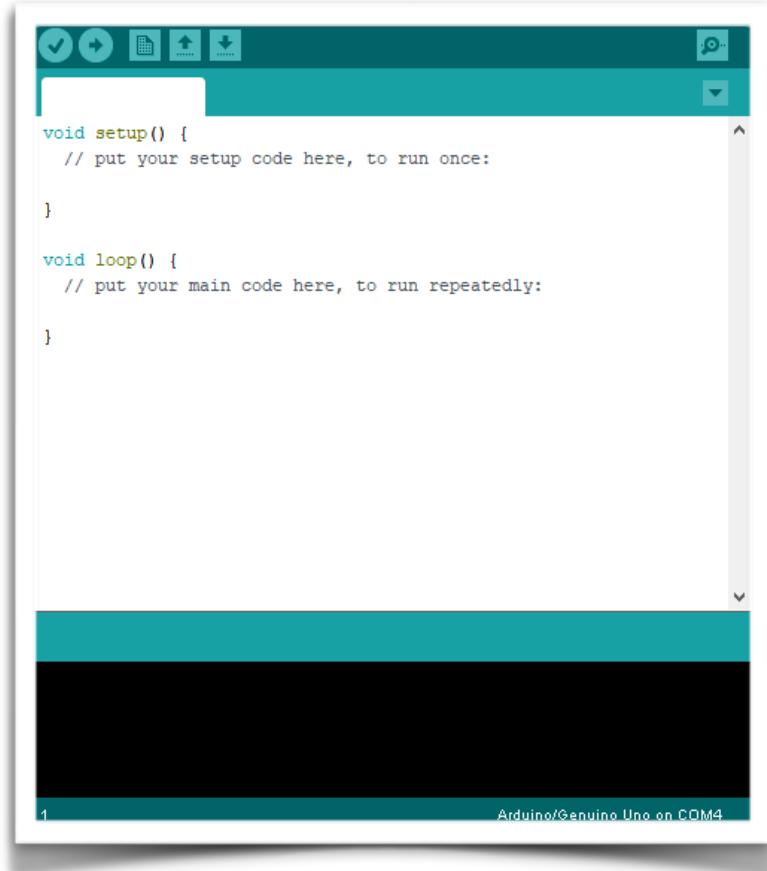
Давайте, на личном примере, напишем самую простенькую программку: моргаем светодиодом, который есть на плате длительностью в 1 секунду, пол секунды он не горит, и потом всё опять.

Здесь нам даже не нужен будет цикл, потому что в программной среде arduino предусмотрено 2 функции: **void setup()** и **void loop()**.

**void setup()** – функция, которая выполняется единожды, при загрузке программы на плату arduino и может повториться только в том случае, если мы заново включим arduino в сеть или же при нажатии кнопки «reset», присутствующей на плате

**void loop()** – соответственно функция, которая выполняется постоянно, когда плата подключена к источнику питания и при повторе функции не требует нажатия кнопки «reset».

Кстати, на следующей странице вы можете увидеть окно (новой) программы при вашем запуске.



```
void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to run repeatedly:
}
```

Arduino/Genuino Uno on COM4

Ничего не напоминает? Нет? Ну ладно :)

И так, перед нами **окно** программы. По умолчанию сразу же заполнено функциями (сетап) и (луп) (буду писать **так** и далее, для удобства), но перед функциями мы можем задавать значения или добавлять какие-нибудь необходимые библиотеки (правда похоже на C++?). Как было сказано ранее, на 13 порту платы ардуино имеется встроенный светодиод, вот с ним то мы и будем работать.

Для начала перед функцией (сетап) приписываем целочисленное значение нашему светодиоду – прописываем константы. В данном случае, пишем **int diode = 13;** что означает что на 13-ом порту у нас есть диод, которому мы присвоили целочисленное значение. Теперь перейдем к нашим функциям, начнём с (сетап). Проинициализируем контакты в качестве выхода.

**Все** контакты на плате Arduino могут принимать **2 значения: вход или выход**. Для инициализации будем использовать команду **pinMode(x, OUTPUT/INPUT)**, где x – название переменной (в нашем случае **diode**), а OUTPUT/INPUT – оставляем что-то одно на выбор в зависимости от того, какая у нас программа. В нашем случае пишем **pinMode (diode, OUTPUT);** так как диод у нас должен быть источником извлечения информации, то есть для нас информация это его мигания.

Теперь перейдем к нашей бесконечной функции (луп). Всё что мы сделаем, это напишем, чтобы светодиод светил 1 секунду и не светил пол секунды. Для того, чтобы всё это осуществилось, используем команду **digitalWrite(x, HIGH/LOW)** – пример аналогичный с **pinMode**.

Получается:

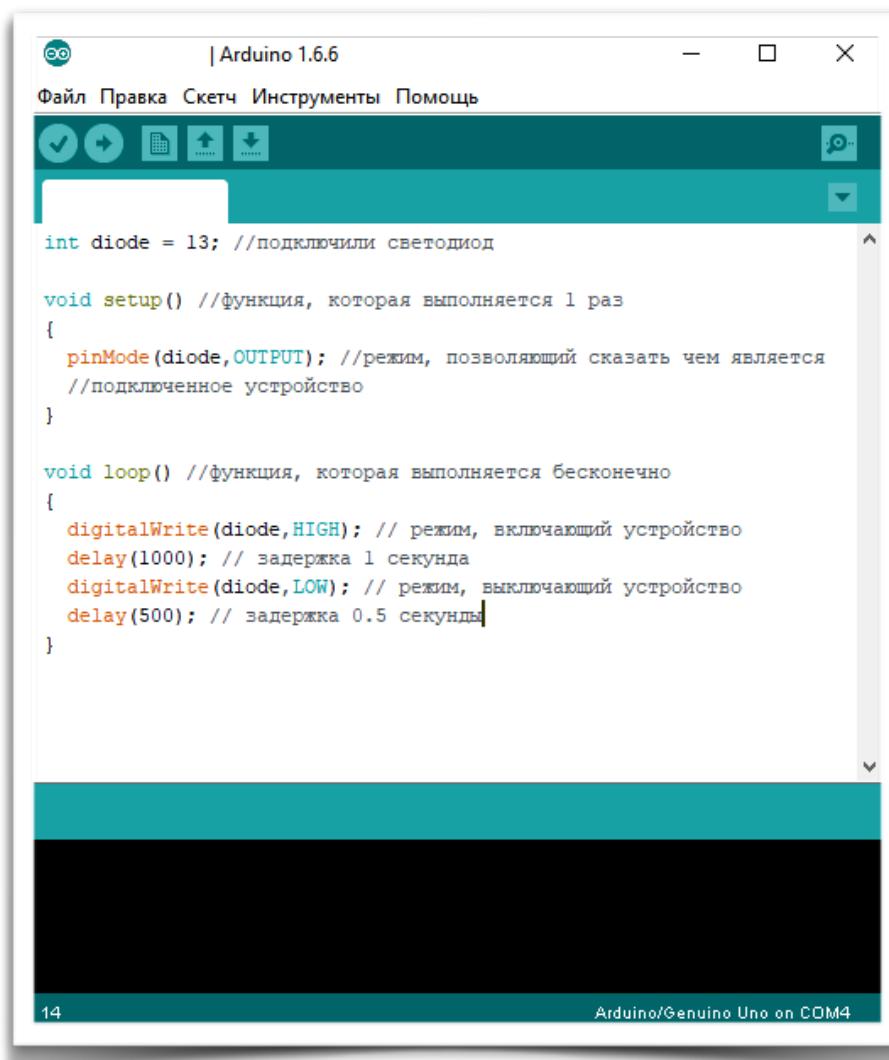
```
digitalWrite(diode, HIGH);
delay(1000);
digitalWrite(diode, LOW);
delay(500);
```

Вот и всё. Что такое **delay?** Всё просто. Delay – это команда задержки, а в скобочках указывается количество миллисекунд, которое надо подождать, чтобы продолжить программу. И того в 1-ой секунде 1000 миллисекунд, 500 миллисекунд – 0,5 секунды.

**Поздравляю**, вы написали свою первую программу!

Раньше светодиоды были качественнее, чем сейчас. Светодиоды делали из арсенида галия, который имел определенное падение напряжения в зависимости от излучаемого света, а теперь же, большинство светодиодов выпускаются из вещества, которое светит в ультрафиолетовом диапазоне и просто покрываются разноцветными пластиковыми кожухами, в итоге что мы получаем: тот же самый цвет, необходимый нам, но одинаковое падение напряжения вне зависимости от цвета. Как проверить светодиод? Есть несколько способов. Например можно взять светодиоды и с помощью вольтметра измерять падения напряжений на последних, или же, если дома есть лампа, которая светит ультрафиолетом, то посветить на светодиоды: в итоге мы должны увидеть светящуюся в ультрафиолете подложку катода и анода светодиода, что будет означать что светодиод ультрафиолетового диапазона. Мда, так и живём, до чего дошли китайцы.

Вот, как в итоге должен выглядеть ваш код  
Всё расписано, всё хорошо. Осталось лишь **записать** эту программу в arduino.



The screenshot shows the Arduino IDE interface with the title bar "Arduino 1.6.6". The menu bar includes "Файл", "Правка", "Скетч", "Инструменты", and "Помощь". Below the menu is a toolbar with icons for file operations. The main code editor window contains the following C++ code:

```
int diode = 13; //подключили светодиод

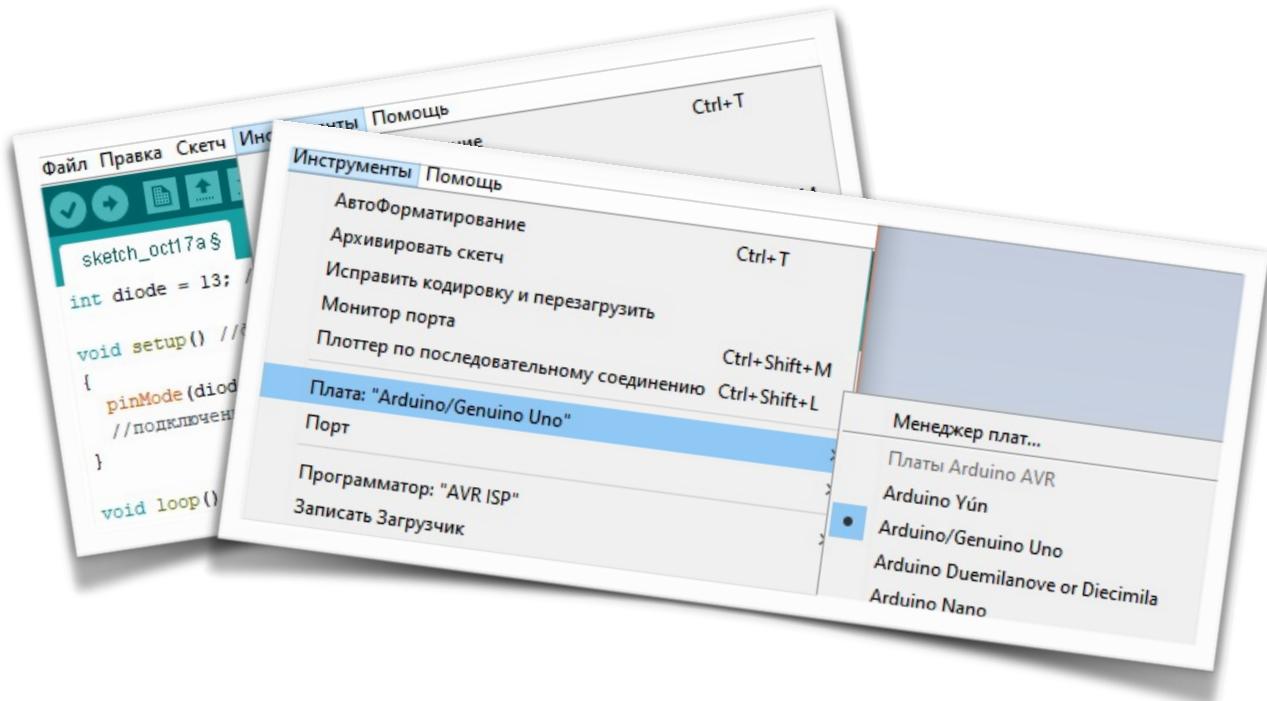
void setup() //функция, которая выполняется 1 раз
{
    pinMode(diode,OUTPUT); //режим, позволяющий сказать чем является
    //подключенное устройство
}

void loop() //функция, которая выполняется бесконечно
{
    digitalWrite(diode,HIGH); // режим, включающий устройство
    delay(1000); // задержка 1 секунда
    digitalWrite(diode,LOW); // режим, выключающий устройство
    delay(500); // задержка 0.5 секунды
}
```

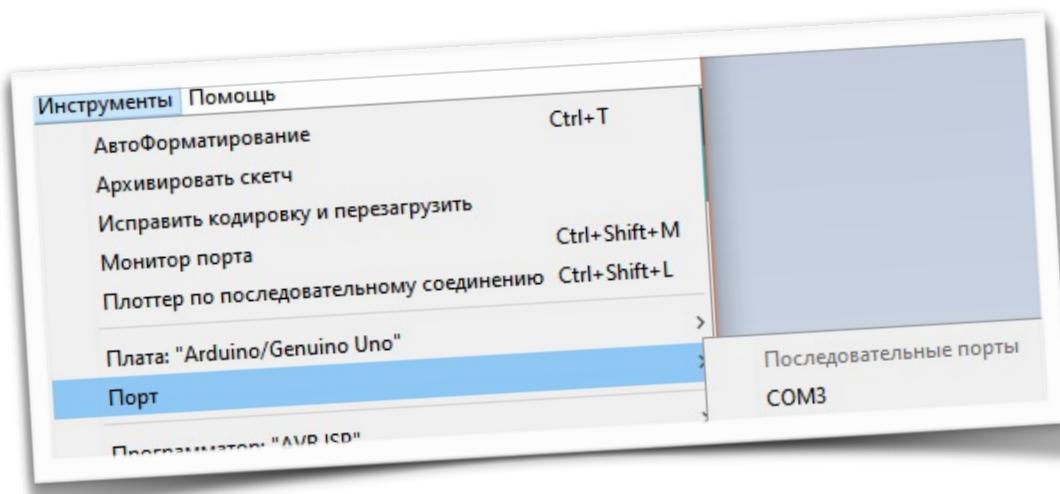
The status bar at the bottom left shows "14" and at the bottom right shows "Arduino/Genuino Uno on COM4".

## Заливай!

Кликаем на кнопку «**инструменты**», далее мы должны выбрать с какой arduino мы работаем, для этого выбираем параметр «**плата:**» и там выбираем «**Arduino/ Genuino Uno**» если мы работаем с arduino uno или же «**Arduino Nano**» если работаем с arduino nano, соответственно.



Далее надо убедиться, что arduino подключена к **правильному** порту, для этого нажимаем кнопку «**порт**» и убеждаемся в правильном номере порта, обычно это порты com3 или com5, но у вас, конечно же, может отличаться. **Вообще, это не очень важно.**



Всё сделали, всё подключили, но arduin'e всё равно от этого ни холодно ни жарко. Пора уже **заливать** в неё наш прекраснейший код. Для того, чтобы залить программу, надо сначала её **проверить**. Для этого, нажимаем «**галочку**», которая находится под меню программы, и, если компилятор не выдал ошибок в программе, предложит сохранить вашу программу на компьютере. Программы в arduino называются «**скетчами**», сохраняйте ваши скетчи в одной общей папке, чтобы потом вы смогли по названию найти нужный скетч и использовать его в будущих проектах, и не забудьте **изменить** название вашего проекта на то, которое вы сможете потом в будущем легко определить ;)

Проверили, сохранили, всё хорошо. Но скетч до сих пор не на arduino? Правильно!) Мы только сделали все проверки, а теперь только можем «**заливать**» скетч на arduino, для этого нажимаем кнопку «→», которая находится справа от «галочки».

Всё. Вот теперь, действительно, можно увидеть то, что светодиод, находящийся на 13-ом порту нашей arduino, моргает с такой частотой, которую задали мы.



## Самостоятельная работа №1

Хех, нет, не пугайтесь. Ничего сложного. Всего-навсего вам надо будет переделать вашу первую программу.

У вас теперь есть **2** светодиода. Вам надо, в программе, сделать так, чтобы сначала загорался один светодиод, потом второй, а потом первый гаснет и второй завершает. Интервал времени любой, который вам удобен, но советую что-то около 1 секунды.

Ответы будут прилагаться на следующих страницах, сразу же после условия задачи. Но вы не жульничайте и не торопитесь перелистывать на ответ. Попробуйте проверить свои силы и освоение пройденного материала.

*"Оттого, что человек много ест, он не становится здоровее, чем тот, который довольствуется только необходимым: точно так же и ученый — это не тот, кто много читает, а тот, кто читает с пользой".*

Аристипп

А вот и ответ к данной задаче :) У вас должно было получиться что-то похожее.

Обратите внимание, что каждому датчику, а в данном случае светодиоду, необходим свой int, так как он является отдельным устройством.

The screenshot shows the Arduino IDE interface with the following code:

```
int diode = 13; //подключили светодиод
int diod = 12; //подключили второй светодиод

void setup() //функция, которая выполняется 1 раз
{
    pinMode(diode,OUTPUT); //режим, позволяющий сказать чем является
    //подключенное устройство
    pinMode(diod,OUTPUT);
}

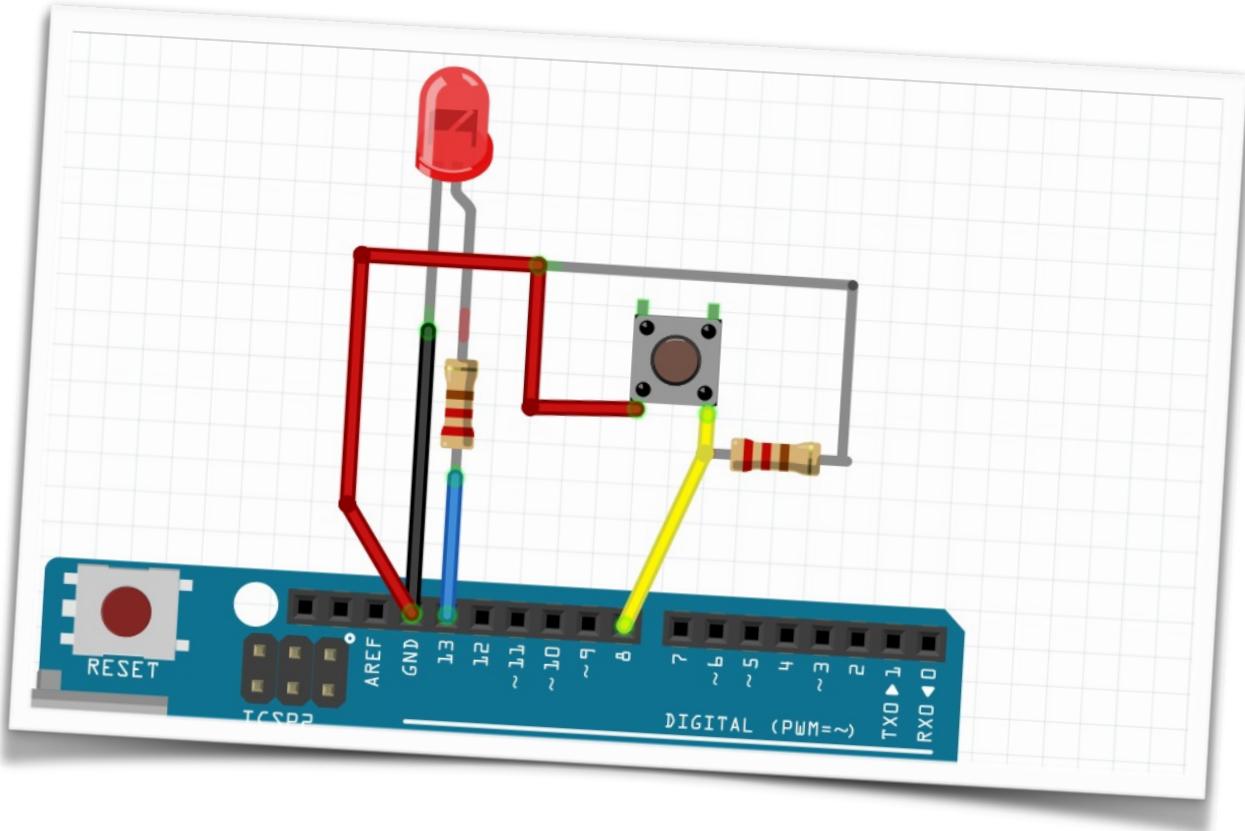
void loop() //функция, которая выполняется бесконечно
{
    digitalWrite(diode,HIGH); // режим, включающий устройство
    delay(1000); // задержка 1 секунда
    digitalWrite(diod,HIGH);
    delay(1000);
    digitalWrite(diode,LOW); // режим, выключающий устройство
    delay(1000); // задержка 1 секунда
    digitalWrite(diod,LOW);
    delay(1000);
}
```

The code initializes two pins, 13 and 12, as output pins. The `setup()` function sets the mode for both pins. The `loop()` function alternates between turning the two LEDs on and off at one-second intervals. The status bar at the bottom indicates "Arduino/Genuino Uno on COM4".

## Новый уровень

Переходим к чему-нибудь посложнее. Соберем схему, которая будет включать и выключать светодиод по нажатию кнопки. Здесь нам помимо светодиода и кнопки понадобятся 2 резистора: на 220 Ом и на 10 кОм. 220 Ом нужны для ограничения тока и напряжения на светодиоде, а 10 кОм надо для того, чтобы напряжение на нашей кнопке оставалось постоянным и убирались помехи. Можно использовать второй резистор куда гораздо большего номинала, но нам хватит и этого.

Ваш лучший помощник при работе с ардуинто и простыми схемками –



это breadboard. С помощью него можно собрать любую схему прежде чем паять всё на плате и заодно проверить работоспособность.



The image shows the Arduino IDE interface. The top bar contains standard icons for file operations (New, Open, Save, Print, Undo, Redo) and a settings gear icon. Below the toolbar is a code editor window displaying the following sketch:

```
int switchPin = 8;
Int ledPin = 13;

void setup()
{
    pinMode(switchPin, INPUT);
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    If (digitalRead(switchPin) == HIGH)
    {
        digitalWrite(ledPin, HIGH);
    }
    else
    {
        digitalWrite(ledPin, LOW);
    }
}
```

The code sets up pin 8 as an input and pin 13 as an output. In the loop, it reads the state of the switch connected to pin 8. If the switch is open (HIGH), it turns on the LED connected to pin 13. If the switch is closed (LOW), it turns off the LED.

А вот и сам код. Вроде бы как написали, самое время заливать его в arduino.

Всё работает здорово. Но это очень простой код, он будет работать и без микроконтроллера, надо что то посерьёзнее. Например путь у нас будет так: мы нажимаем на кнопку и светодиод загорается, ещё раз нажимаем и тухнет.

Для этого немного переделаем наш код:

Для удобства изобразим его не в виде картинки, а текста

```
int switchPin = 8;
int ledPin = 13;
boolean lastButton = LOW;
boolean ledOn = false; // переменные для запоминания состояния системы

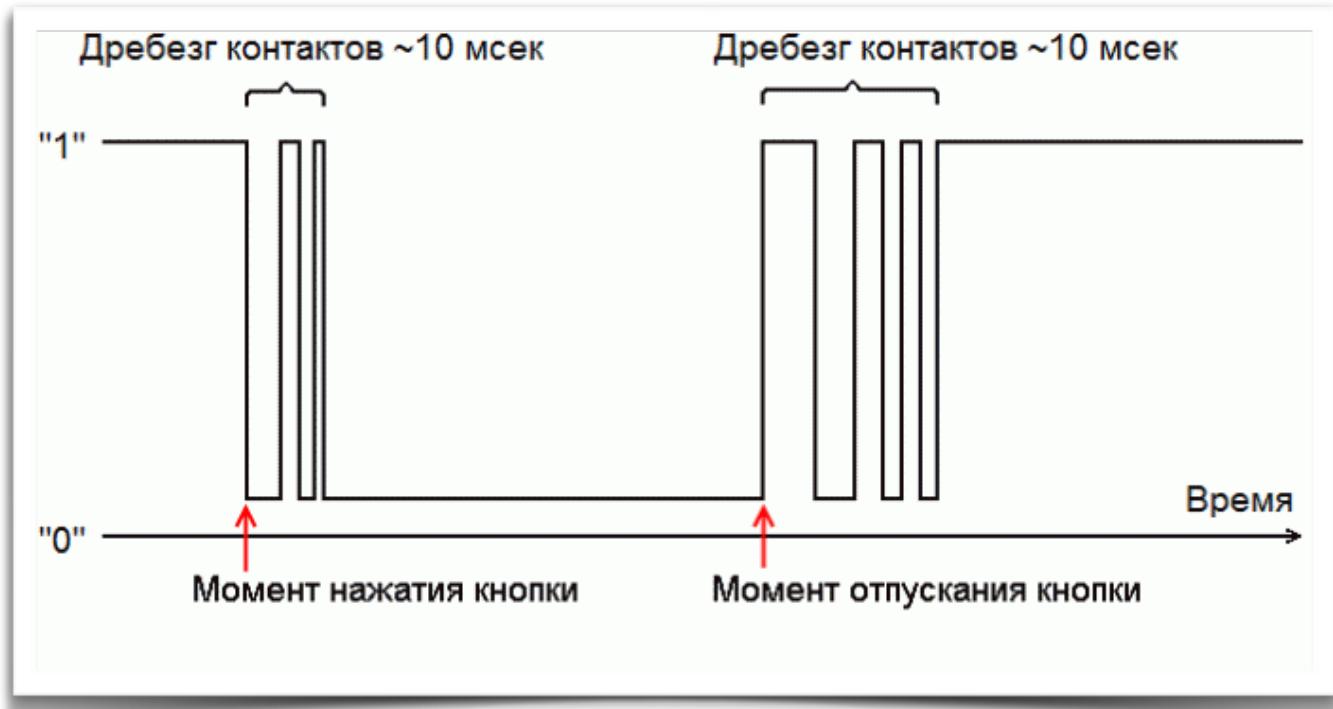
void setup()
{
    pinMode(switchPin, INPUT); // кнопка у нас является вводом информации, не так ли?
    pinMode(ledPin, OUTPUT); // а вот светодиод выводом
}

void loop()
{
    If (digitalRead(switchPin) == HIGH && lastButton == LOW)
    {
        ledOn = !ledOn; // перевернем значение чтобы включить светодиод
        lastButton = HIGH;
    }
    else
    {
        lastButton = digitalRead(switchPin); // читаем сигнал с кнопки
    }
    digitalWrite(ledPin, ledOn);
}
```

Сделали прогу, заливаем, смотрим. Хм, как-то **странны**, она работает периодически... очень странно. Что-то здесь не так.

В чём здесь дело?

Если вам кажется, что всё дело в **дребезжании** кнопки, то вы правы. Дребезжание кнопки, это когда наш сигнал **продолжает** поступать далее на доли миллисекунд **после** того, как мы уже **отпустили** кнопку:



## Что же с этим делать?

Давайте допишем программу, чтобы справиться с этим:

```
int switchPin = 8;
int ledPin = 13;
boolean lastButton = LOW;
boolean ledOn = false; // переменные для запоминания
// состояния системы
boolean currentButton = LOW;

void setup()
{
    pinMode(switchPin, INPUT);
    pinMode(ledPin, OUTPUT);
}

boolean debounce(boolean last) // делаем
функцию которая добавит нам значение,
принимаемое до нажатия кнопки
{
    boolean current = digitalRead(switchPin); // узнаем значение кнопки сейчас
    if (last != current) // если состояние кнопки изменилось
    {
        delay(5); // делаем задержку кнопки на 5 миллисекунд
        current = digitalRead(switchPin); // считываем значение кнопки по новой,
        // предполагая получить устаканившееся значение
    }
    return current; // возвращаем устаканившееся значение
}

void loop()
{
    currentButton = debounce(lastButton);
    if (lastButton == LOW && currentButton == HIGH)
    {
        ledOn = !ledOn; // перевернем значение чтобы включить светодиод
    }
    lastButton = currentButton;
    digitalWrite(ledPin, ledOn);
}
```



Теперь попробуем запустить программу по новой, давайте попробуем.  
Работает чётко!

# Выбор Arduino

Какую же всё-таки arduino выбрать начинающему радиолюбителю? В чём различия arduino между собой? Почему их так много разных?



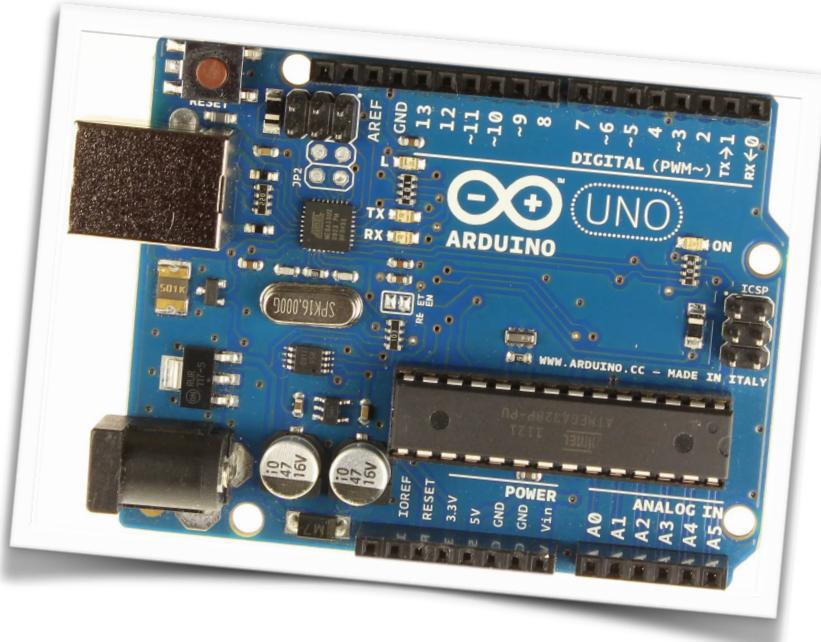
Вопросов много, так же как и ответов) Начнем по порядку:

В чём различия arduino: как я уже упоминал на факультативе, в большинстве моделей arduino, кроме размера, никаких **различий нет**.

Почему их так много разных: каждая сделана **под свои какие-то условия**, в основном, опять же, зависящие от размеров. Есть и **исключения**, но о них позже.

какую же arduino выбрать чтобы познакомиться с платформой:

1. **Arduino Uno** - основа всех основ. Это, пожалуй, самый лучший выбор для самого что не есть начинающего радиолюбителя, ардуиномана и т.п.



Средний размер, микроконтроллер ATmega 328, большое количество контактных входов/выходов и 2 источника питания: через usb **type-b** по сути являющиеся так же программаторным входом (в некоторых китайских клонах *micro usb* - что не советую вообще) или же круглое гнездо как у большинства электронных устройств, находящихся дома. Питание от 5 до 9 (12, но не желательно) вольт.

+ Хороший вариант для начинающего радиолюбителя

+ Достаточное количество выводов

+ Диапазон питания от 5 до 12 вольт

+ Микроконтроллер ATmega328 (8 бит)

+ 2 источника питания

+ Не требуется программатор

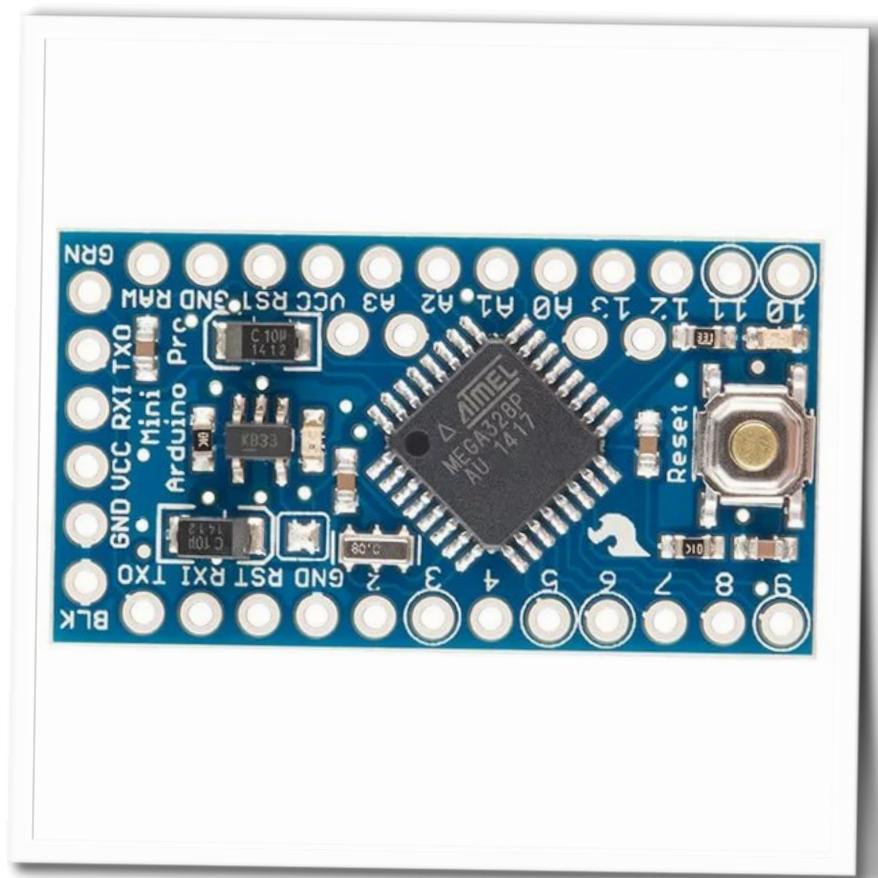
- Средняя цена (от 4 до 8 баксов)

- Размер чуть меньше ладони (есть платы и меньше)

- Может продаваться с разъемом micro usb вместо usb type-b

- Не у всех есть usb type-b, но если покупать комплект обычно его кладут вместе с платой

2. **Arduino pro mini** - вариант для тех, кому важна компактность и совсем малые габариты устройства. Плату можно разместить для удобства прямо на макетной плате (брэдборде). Всё тот же микроконтроллер ATmega328, на 1 или 2 входа/выхода больше чем у Arduino Uno, но вот питание...



Питается и программируется **только** через программатор, который **необходимо докупать** отдельно на всё том же aliexpress'e , а ещё и подключить его правильно к определенным ножкам это тоже проблема. Вариант не для начинающего ардуиномана, уже для более опытного пользователя.

- + Цена
- + Компактность

+ Чуть большее количество выводов чем у ардуино Уно

+ Микроконтроллер ATmega328 (8 бит)

+ Можно использовать сразу с макетной платой

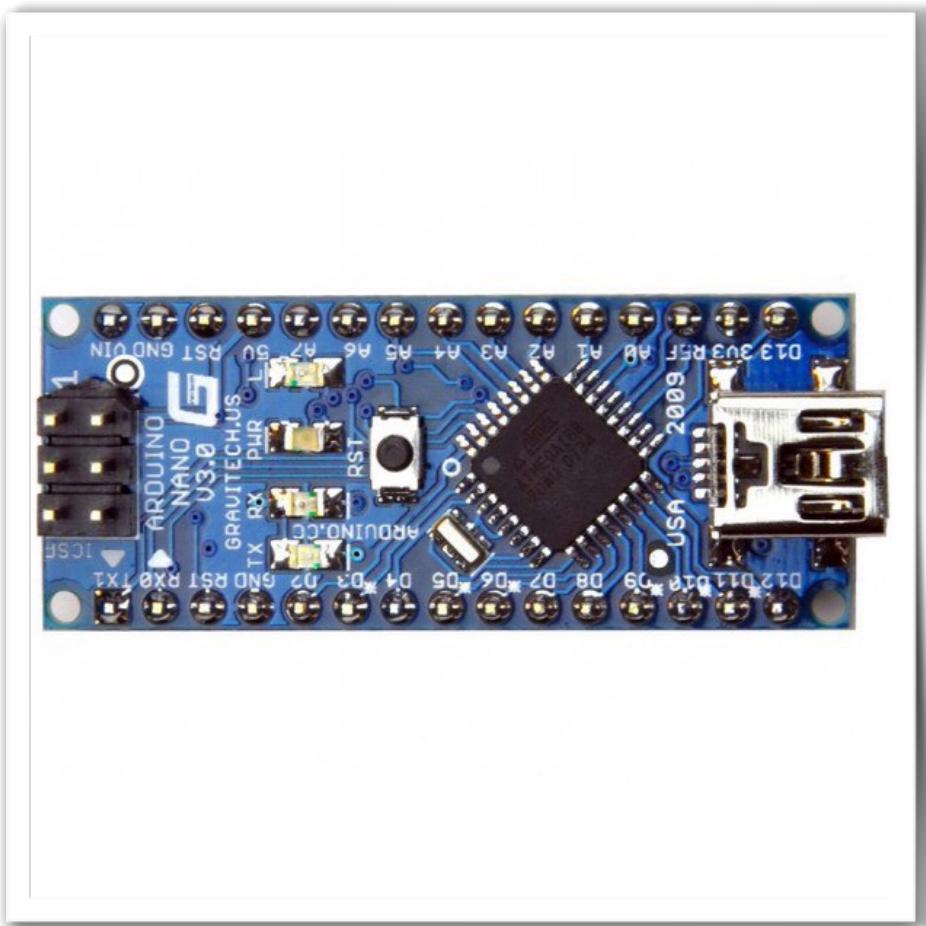
- Нужен программатор отдельно, который стоит столько же

- Питание только от 5 вольт

- Не очень хороший вариант для начинающего радиолюбителя

- При покупке с aliexpress'a необходимо допаять ножки

**3. Arduino nano** - так же как и Arduino Uno отличный вариант. Совмещает в себе все лучшие качества Arduino Uno и Arduino pro mini: компактность, можно воткнуть в макетную плату и работать на месте, такое же количество выводов, всё тот же микроконтроллер ATmega328 и программируется от mini usb. Питание только от него же, то есть 5 вольт.



+ Так же неплохой вариант для начинающего

+ Цена (необходимо искать правильную цену на алиэкспрессе)

+ Компактность

+ Можно использовать сразу с макетной платой

+ Микроконтроллер ATmega328 (8 бит)

+ Чуть больше выводов чем у ардиуно Уно

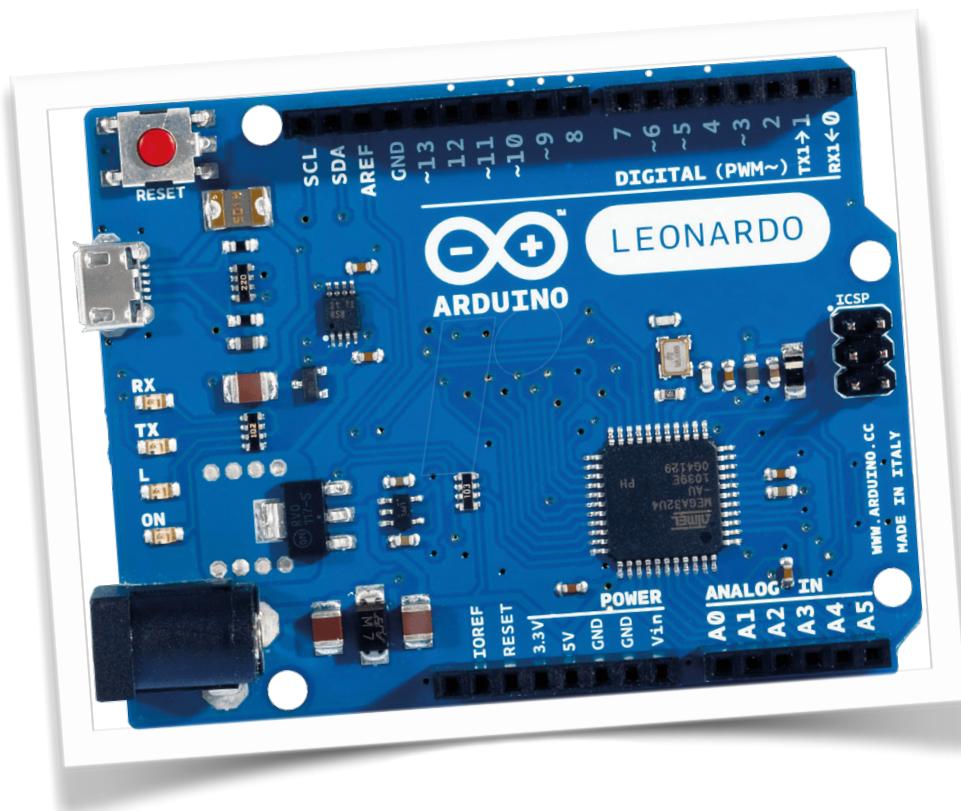
+ Не нужен программатор

- Питание только от 5 вольт

- Шнур mini usb (НЕ ПУТАТЬ С micro usb) не всегда у всех есть дома

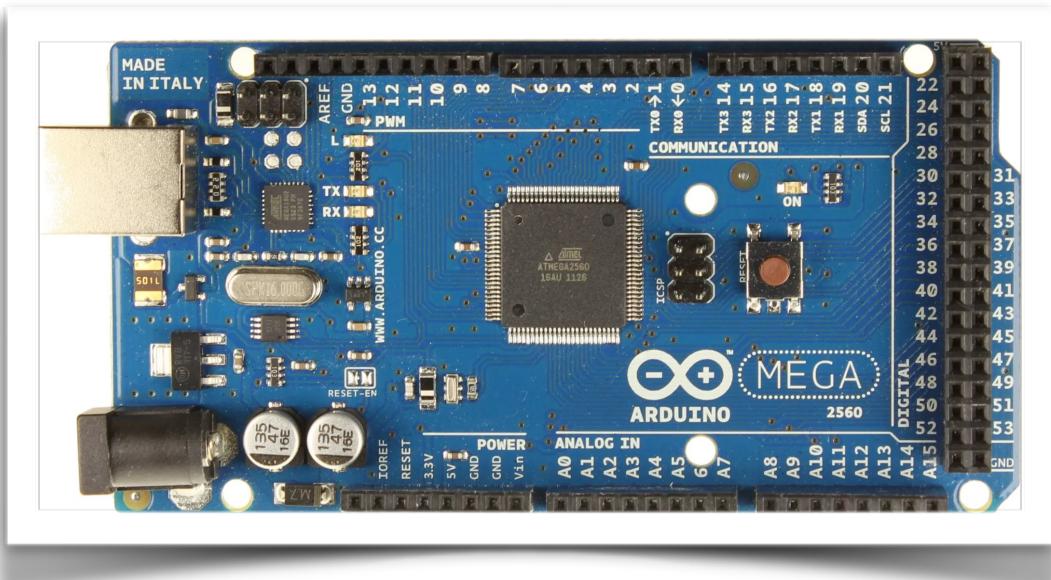
- При покупке с алиэкспресса необходимо допаять ножки

4. **Arduino Leonardo** - старший брат Arduino Uno. Различие лишь в том, что у него предназначений может быть больше, а всё это связано с наличием другого микроконтроллера на борту ATmega32U4. Так же благодаря этому микроконтроллеру инженерам и разработчикам платы удалось уменьшить количество элементов на самой плате (стало меньше резисторов, конденсаторов, транзисторов). На основе ардуино Леонардо гораздо легче сделать такие устройства как компьютерную мышь, геймпад и ещё парочку устройств, которым необходимо огромное количество вычислений в секунду.



- + Большой функционал чем у Arduino Uno
- + Меньшее количество элементов
- + 2 разъема питания (5 - 12 вольт)
- + Достаточное число входов/выходов
- + Выходы преобразованы для работы с более сложными элементами
- + Третий вариант по покупке начинающему радиолюбителю
- + Микроконтроллер ATmega32U4
- + Не требуется программатор
  
- За больший функционал платим ценой (цена около 8-14 баксов)
- Не всем начинающим радиолюбителям нужна такая мощь
- Разъем micro usb (компактно - да, прочно - не очень)
- Не компактно (размер Arduino Uno)

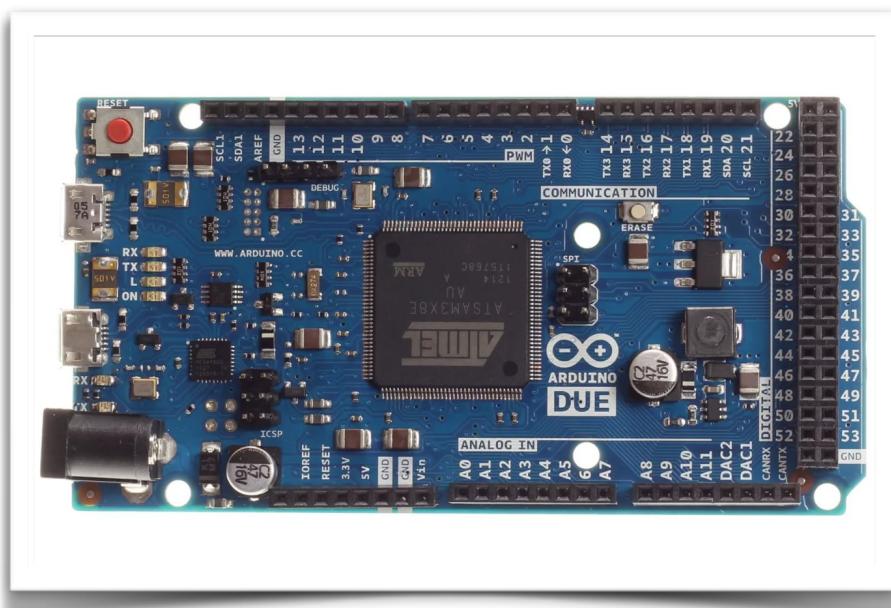
5. **Arduino Mega** - может являться вариантом для начинающего любителя, но всё же не необходимым. На плате присутствует в несколько раз большее количество входов/выходов, чем у Arduino Uno и Leonardo вместе взятых. Другой микроконтроллер ATmega2560 подстать количеству контактов. Всё остальное же, кроме размера, соответствует остальным платам.



- + Четвертый вариант для начинающего
- + Микроконтроллер ATmega2560 (8 бит)
- + Большее количество памяти в нём
- + 2 источника питания (5-12 вольт)
- + Не требуется программатор
- + Функционал такой же как и у выше написанных плат
- + Огроменное количество выходов/входов для подключения большего количества устройств

- Огромный размер (как 2 платы Arduino Uno и Leonardo)
- За размер и память доплачиваем (цена около 7-10 баксов)
- Вариант вариантом, но мощь избыточная для начинающего

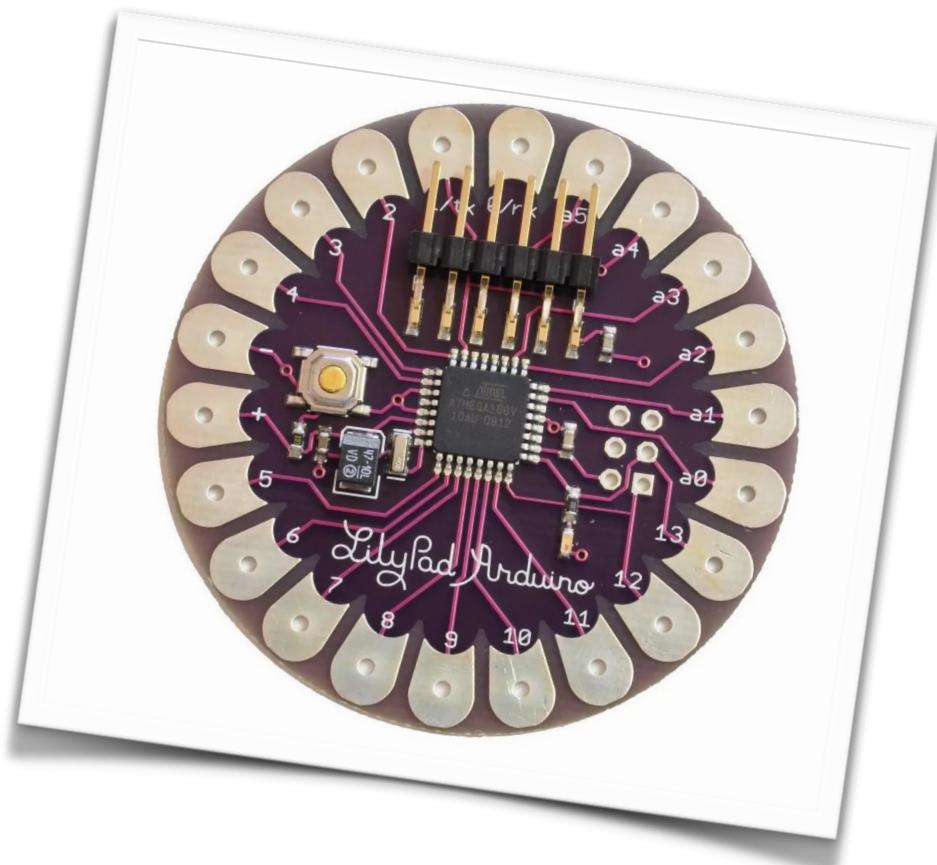
**6. Arduino Due** - самый бесполезный вариант как для начинающего, так и частично для среднего радиолюбителя. В плату встроен микроконтроллер Atmel SAM3X8E ARM Cortex-M3, который в отличие от вышеописанных собратьев обладает вместо 8 битной шиной 32 битной!



Размеры устройства и внешний вид напоминают Arduino Mega, имеется 3 входа питания: круглое гнездо и 2 разъема micro usb - один для программирования, второй просто для питания (в общем геморой какой то слишком заумный, их ещё и 5 раз попутать можно). И самый огроменный минус... Плата может питаться только от 5 вольт, а устройства, подключенные к ней только до 3.3 вольт - что есть очень хреново. Зато, если работать с этой платой "не дышав и не двигаясь" то у неё очень большой мощный потенциал. Но скажите... вам это надо? мне лично нет. Может в будущем закажу такую плату, но это будет не раньше чем в начале следующего года.

- + Мощный микроконтроллер Atmel SAM3X8E ARM Cortex-M3 (32 бита)
- + Огромное количество входов/выходов
- + 3 источника питания
- + Разъем micro usb (но это не всегда хорошо, если честно)
- + Не требуется программатор
- + Огромный потенциал
- Цена. Она тоже космическая
- Питание только от 5 вольт. Если подаём больше - плате кирдык
- Огромный размер как у Ардуино Мега
- Устройства, подключенные в плату пытаются только от 3.3 вольт
- Очень неустойчивая плата в плане возможного выхода из строя
- Ненадобность для начинающих и средних радиолюбителей
- Избыточная мощь для простых и даже средних проектов
- Micro usb, которые можно легко сломать/вырвать
- 2 разъема одинаковые с разными назначениями, можно путать

7. **Arduino LilyPad** - без лишних слов, просто супер компактный вариант, который может использоваться (и для этого, в общем-то предназначался) для пошива под одежду, чтобы сделать, например майку со светодиодами, или рюкзак, который может показывать температуру. Но вместо него можно использовать ардуиноnano без ножек или про мини при наличии программатора

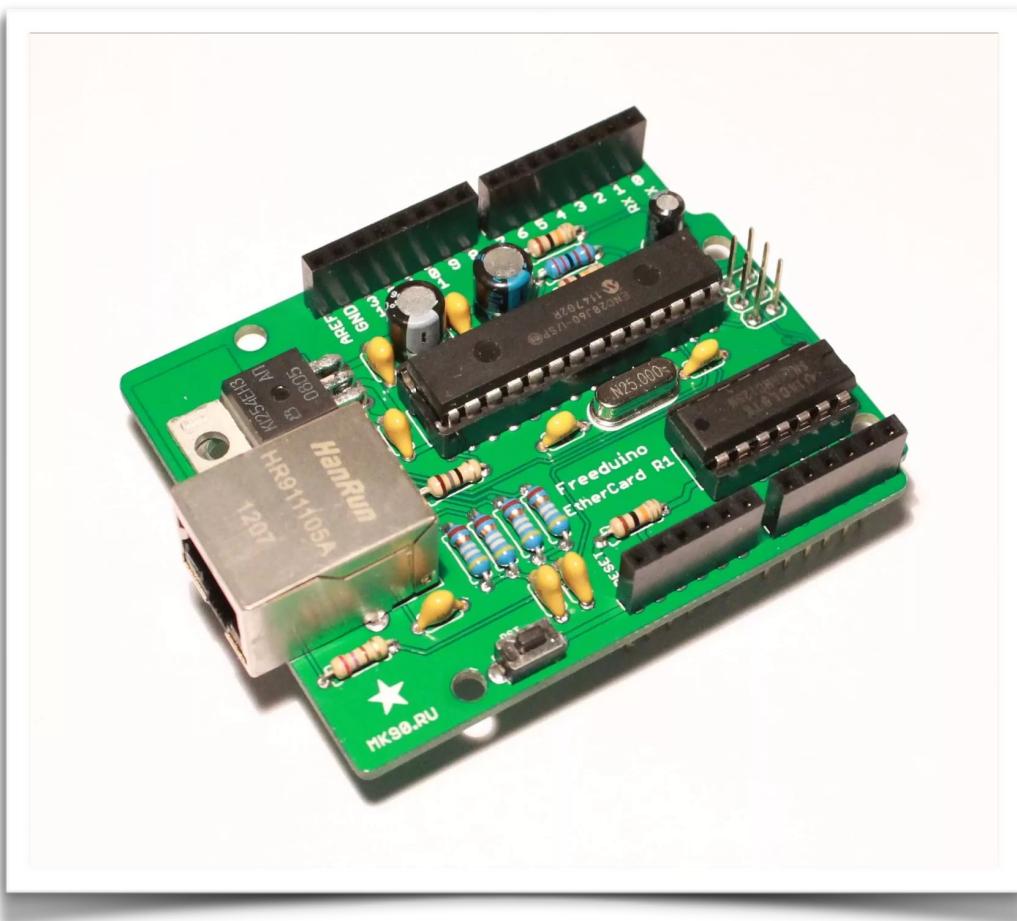


- + Компактность
- + Использование
- + Цена

- Тип предназначения
- Есть аналоги
- Вид платы
- Не подходит для начинающего радиолюбителя

## 8. Различные клоны Arduino. Мы этими клонами пользуемся сейчас)

Это различные **Funduino**, **Japanduino**, **Induino**, **Freeduino** и тому подобное (видоизмененные названия, с окончанием **-duino**).



плюсы и минусы перечислять не буду, кроме одного минуса, пожалуй, а так потенциал и производительность полностью идентичная.

- Не всегда определяется компьютерами и не всегда можно найти для них драйвера

Вот и всё. Я надеюсь я не сильно загрузил вас этой информацией, но как получилось так получилось) Зато теперь, если вы ещё не определились с выбором платы, уже можете примерно знать что вам нужно ;)

# Отступление

## Основные команды для программирования

Написали мы с вами целых 2 программы, а **основные команды**, для того чтобы программировать на языке arduino так и не освоили. Пора исправлять это недоразумение.

Команд для программирования очень много, но мы будем рассматривать только те, которые пользуются, так сказать, большей популярностью (используются чаще всего).

Команды языка программирования Arduino можно разделить на **три** раздела:

### 1. Операторы

**setup()** - Оператор подготовки. Выполняется один раз при включении arduino.

**loop()** - Оператор действия. Выполняется бесконечно, пока не отключится питание.

#### Управляющие операторы

**if** - "если" (если выполняется, сделать что-то).

**if else** - "если... иначе" (если выполняется, сделать что-то... иначе сделать другое).

**for** - Оператор условия.

**while** - "пока" (пока выполняется условие, делать что-то).

**do while** - "делать... пока" (делать что-то пока выполняется условие).

**break** - Прервать функцию.

**continue** - Продолжить функцию.

**return** - Вернуть значения.

#### Синтаксис

**;** - Точка с запятой ставится всегда при завершении строки в коде.

**{}** - Фигурные скобки ставятся всегда в циклах и условиях.

**//** - Однострочный комментарий.

**/\* \*/** - Многострочный комментарий.

#### Арифметические операторы

**=** - Равенство.

**+** - Сумма..

**-** - Разность.

**\*** - Умножение.

**/** - Деление.

**%** - Взятие числа по модулю.

#### Операторы сравнения

**==** - Точное равенство.

**!=** - Не равно.

**<** - Меньше.

**>** - Больше.

**<=** - Меньше или равно.

**>=** - Больше или равно.

#### Логические операторы

**&&** - И.

**||** - Или.

**!** - Логическое отрицание.

#### Унарные операторы

**++** - Прибавить исходное число.

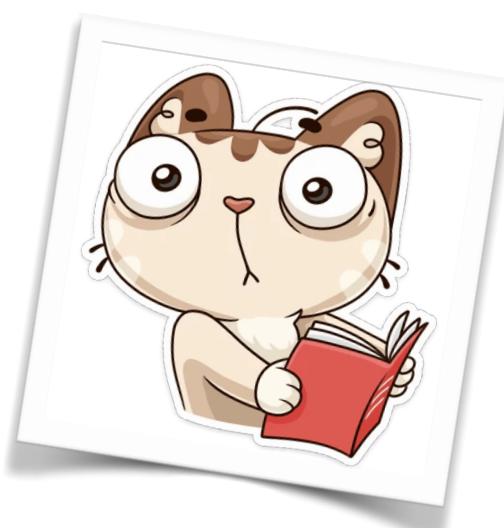
**--** - Отнять исходное число.

**+ =** - Прибавить какое-то определенное число (отличное от заданного).

**- =** - Отнять какое-то определенное число (отличное от заданного).

**\* =** - Умножить на какое-то определенное число (отличное от заданного).

**/ =** - Разделить на какое-то определенное число (отличное от заданного).



## 2. Данные

### Константы

**HIGH | LOW** - Значения "вкл" и "выкл" для датчиков.

**INPUT | OUTPUT** - Обозначение предназначения датчика. Ввод или вывод информации.  
**true | false** - Значения "правда" и "ложь" (для данных типа boolean).

### Типы данных

**boolean** - Логический тип данных, принимающих значения **true** и **false**.

**int** - Целочисленный тип данных.

**float** - Тип данных для хранения значений с плавающей запятой.

**double** - Тип данных, как и float, позволяющий иметь большую точность вычислений.

**массив (array)** - Массив некоторых значений, он же в простонародье "матрица".

## 3. Функции

### Цифровой ввод/вывод

**pinMode()** - Устанавливает режим работы порта как вход или выход.

**digitalWrite()** - Позволяет дать команду цифровому порту.

**digitalRead()** - Позволяет считать данные с цифрового порта.

### Аналоговый ввод/вывод

**analogRead()** - Позволяет дать команду аналоговому порту.

**analogWrite()** - Позволяет считать данные с аналогового порта.

### Работа со временем

**millis()** - Даёт задержку в н-ое количество миллисекунд. Почти не перегружается.

**delay()** - Даёт задержку в н-ое количество миллисекунд. Перегружается каждые 7 дней.

### Математические функции

**min()** - Возвращает минимальное значение.

**max()** - Возвращает максимальное значение.

**abs()** - Возвращает модуль числа.

**constrain()** - Показывает нет ли необходимости заполнить функцию.

**map()** - Перенос значений одного диапазона в другой.

**pow()** - Возведит значение в заданную степень

**sq()** - Функция возвращает квадрат числа, заданного параметром.

**sqrt()** - Функция вычисляет квадратный корень числа, заданного параметром.

### Тригонометрические функции

**sin()** - Синус.

**cos()** - Косинус.

**tan()** - Тангенс.

### Генераторы случайных значений

**random()** - генерирует случайное значение.

### Функции передачи данных

**Serial.begin()** - Функция, позволяющая начать работать с монитором порта.

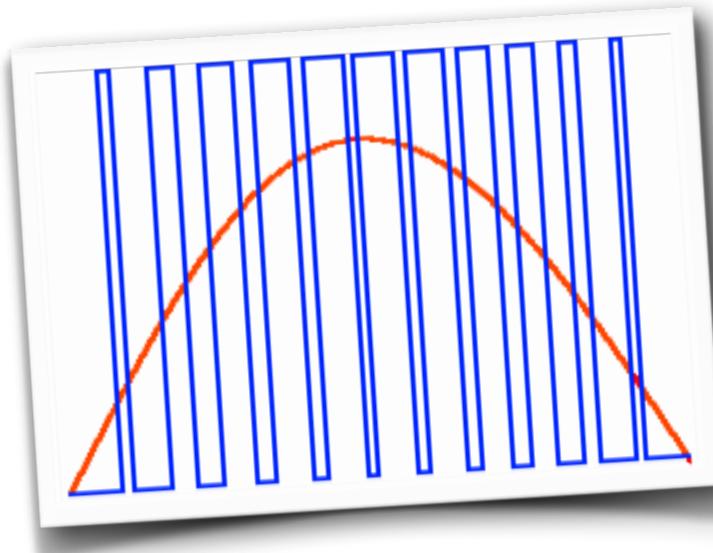
**Serial.print()** - Позволяет выводить на монитор порта данные из других функций.

Теперь, зная зачем нам та или иная команда (функция) мы можем подробнее рассмотреть задачки с макетной платой и arduino.

## ШИМ

### Широтно-импульсная модуляция

Широтно-импульсная модуляция (**ШИМ**) это способ **управления** мощностью на нагрузке с помощью изменения скважности импульсов при **постоянной амплитуде и частоте** импульсов.



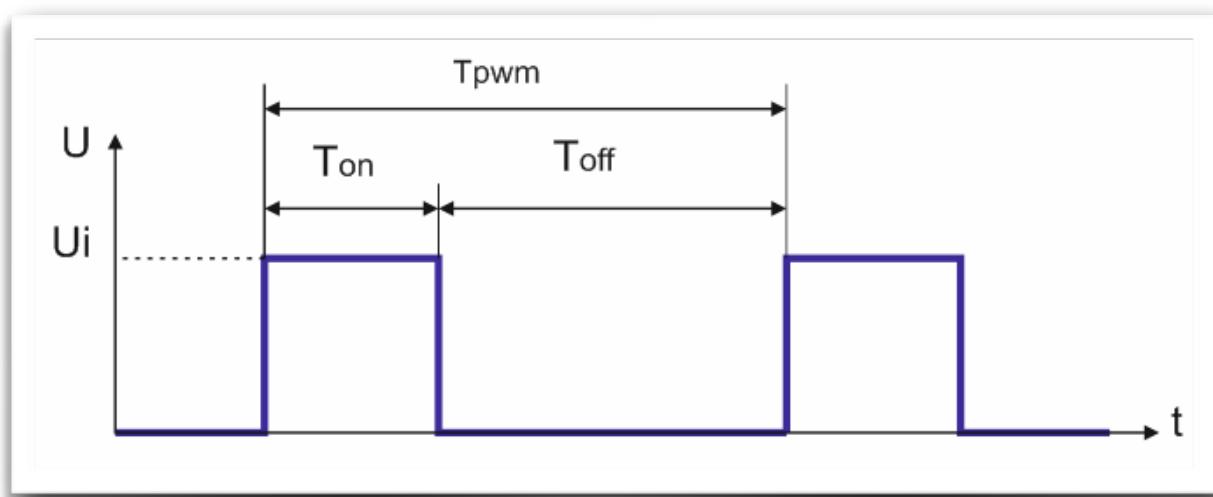
Можно выделить **две** основные области применения широтно-импульсной модуляции:

1. Во вторичных источниках питания, различных регуляторах мощности, регуляторах яркости источников света, скорости вращения коллекторных двигателей и т.п. В этих случаях применение ШИМ позволяет значительно увеличить КПД системы и упростить ее реализацию.

2. Для получения аналогового сигнала с помощью цифрового выхода микроконтроллера. Своеобразный цифро-аналоговый преобразователь (ЦАП). Очень простой в реализации, требует минимума внешних компонентов. Часто достаточно одной RC цепочки.

Принцип регулирования с помощью ШИМ – изменение ширины импульсов при постоянной амплитуде и частоте сигнала.

На диаграмме можно увидеть основные параметры ШИМ сигнала:



**Ui** - амплитуда импульсов ;

**Ton** – время активного (включенного) состояния сигнала;

**Toff** – время отключенного состояния сигнала;

**Tpwm** – время периода ШИМ.

Даже интуитивно понятно, что мощность на нагрузке пропорциональна соотношению времени включенного и отключенного состояния сигнала.

Это соотношение определяет коэффициент заполнения ШИМ:

$$Kw = Ton / Tpwm.$$

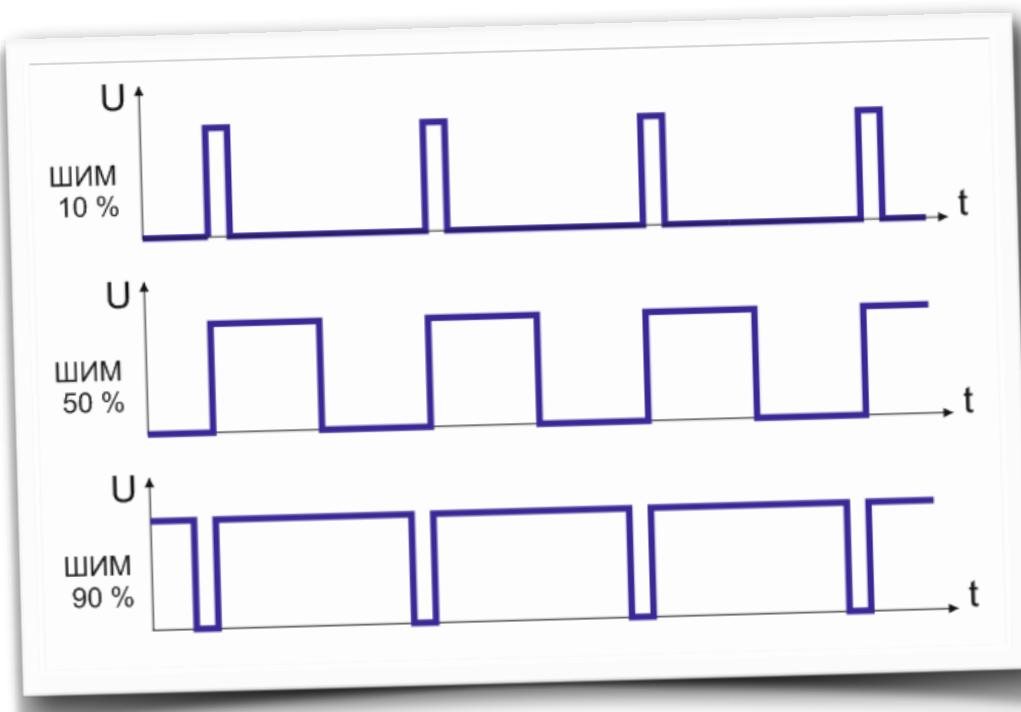
Он показывает, какую часть периода сигнал находится во включенном состоянии. Может меняться:

от 0 – сигнал всегда выключен;

до 1 - сигнал все время находится во включенном состоянии.

Чаще используют процентный коэффициент заполнения. В этом случае он находится в пределах от 0 до 100%.

Среднее значение электрической мощности на нагрузке строго пропорционально



коэффициенту заполнения. Когда говорят, что ШИМ равен, например, 20%, то имеют в виду именно **коэффициент заполнения**.

Платы Arduino на базе микроконтроллеров ATmega168/328 имеют **6** аппаратных широтно-импульсных модуляторов. Сигналы ШИМ могут быть сгенерированы на выводах **3, 5, 6, 9, 10, 11**. Об этом я писал выше.

Управление аппаратными ШИМ осуществляется с помощью системной функции **analogWrite()**.

```
void analogWrite(pin, val);
```

Функция переводит вывод в режим ШИМ и задает для него коэффициент заполнения. Перед использованием `analogWrite()` функцию `pinMode()` для установки вывода в режим “выход” вызывать необязательно.

Аргументы:

`pin` – номер вывода для генерации ШИМ сигнала.

`val` – коэффициент заполнения ШИМ. Без дополнительных установок диапазон `val` от 0 до 255 и соответствует коэффициенту заполнения от 0 до 100 %. Т.е. разрядность системных ШИМ в Ардуино 8 разрядов.

`analogWrite(9, 25); // на выводе 9 ШИМ = 10%`

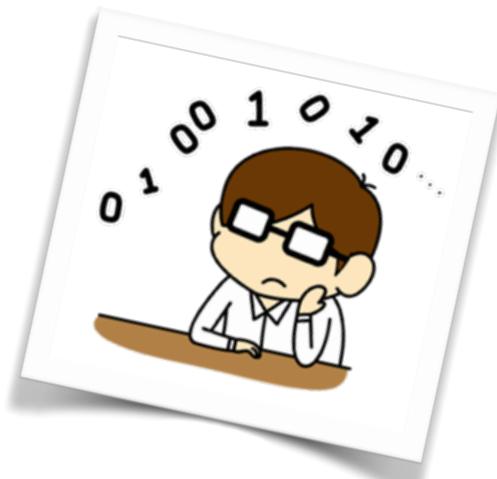
Теперь, используя эти знания, немного видоизменим нашу последнюю программу, чтобы по нажатию кнопки светодиод начинал гореть всё ярче и ярче. “А когда он уже будет гореть на максимум, что будет?”. Хороший вопрос, и такой же хороший ответ: когда светодиод дойдёт до своего максимума (а это 5 нажатий каждое по 51 градации до 255. Почему? Эти числа очень хорошо и без остатка делятся друг на друга, так что нам в самый раз) мы просто возьмём и сбросим его последнее значение обратно в 0, и того у нас получится замкнутый круг.

```
int switchPin = 12;
int diode = 11;
boolean lastB = LOW;
boolean currentB = LOW;
int diodelevel = 0;

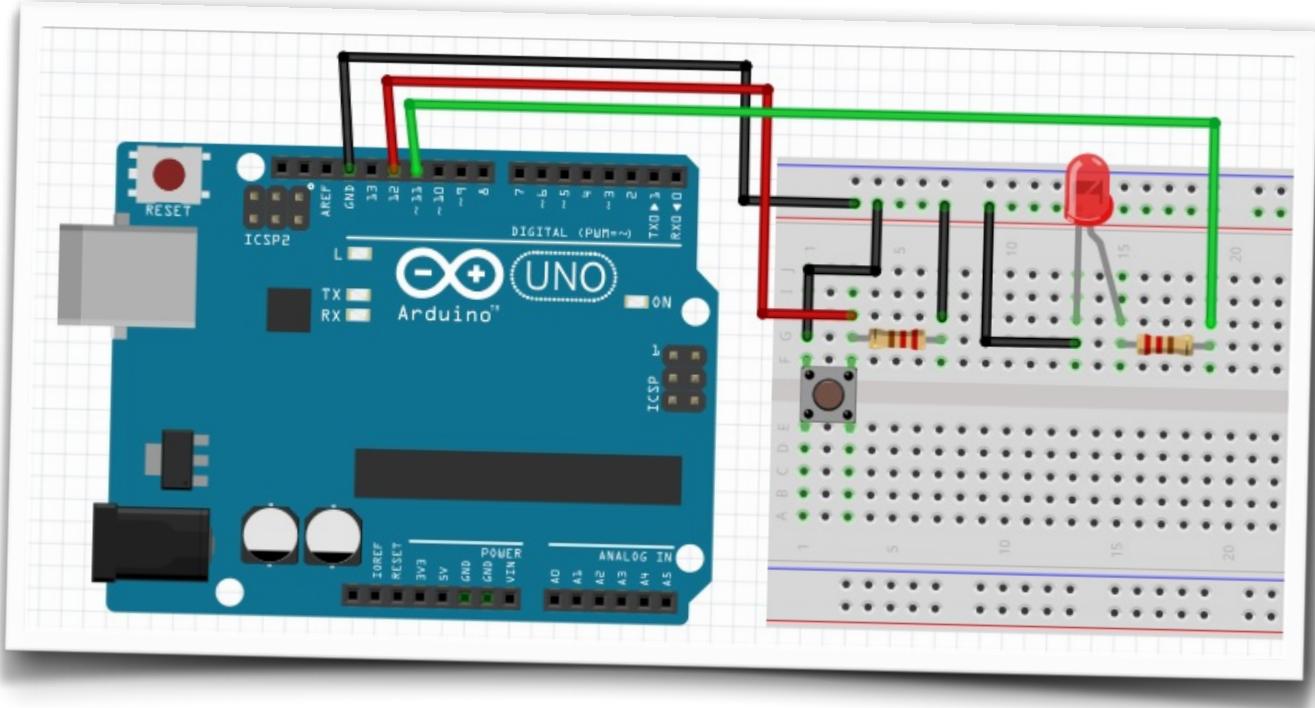
void setup()
{
    pinMode(switchPin, INPUT);
    pinMode(diode, OUTPUT);
}

boolean debounce(boolean last)
{
    boolean current = digitalRead(switchPin);
    if (last != current)
    {
        delay(5);
        current = digitalRead(switchPin);
    }
    return current;
}

void loop()
{
    currentB = debounce(lastB);
    if (lastB == LOW && currentB == HIGH)
    {
        diodelevel = diodelevel + 51;
    }
    lastB = currentB;
    if (diodelevel > 255) diodelevel = 0;
    analogWrite(diode, diodelevel);
}
```



А вот как выглядит эта схема на макетной плате:

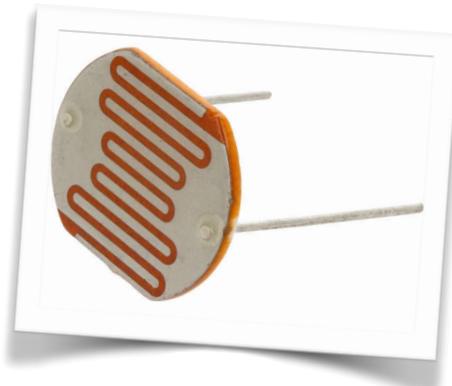


## Автоматическое управление светом

**ШИМ** - очень крутая штука. Но хочется уже сделать так, чтобы не надо было нажимать большое количество раз на кнопку, дабы подстроить уровень свечения светодиода. Хочется сделать так, чтобы arduino **сама** всё это рассчитывала и **контролировала**. Что ж, не вопрос! В этом нам поможет такая штука как **фоторезистор**.

Фоторезисторы дают вам возможность определять интенсивность освещения. Они маленькие, недорогие, требуют мало энергии, легки в использовании, практически не подвержены износу. Именно из-за этого они часто используются в игрушках, гаджетах и приспособлениях. Конечно же, DIY-проекты на базе Arduino не могли обойти своим вниманием эти замечательные датчики.

Фоторезисторы по своей сути являются резисторами, которые изменяют свое сопротивление (измеряется в Ом) в зависимости от того, какое количество света попадает на их чувствительные элементы. Как уже говорилось выше, они очень дешевые, имеют различные размеры и технические характеристики, но в большинстве своем не очень точные. Каждый фоторезистор ведет себя несколько иначе по сравнению с другим, даже если они из одной партии от производителя. Различия в показаниях могут достигать 50% и даже больше! Так что рассчитывать на прецизионные измерения не стоит. В основном их используют для определения общего уровня освещенности в конкретных, "локальных", а не "абсолютных" условиях.

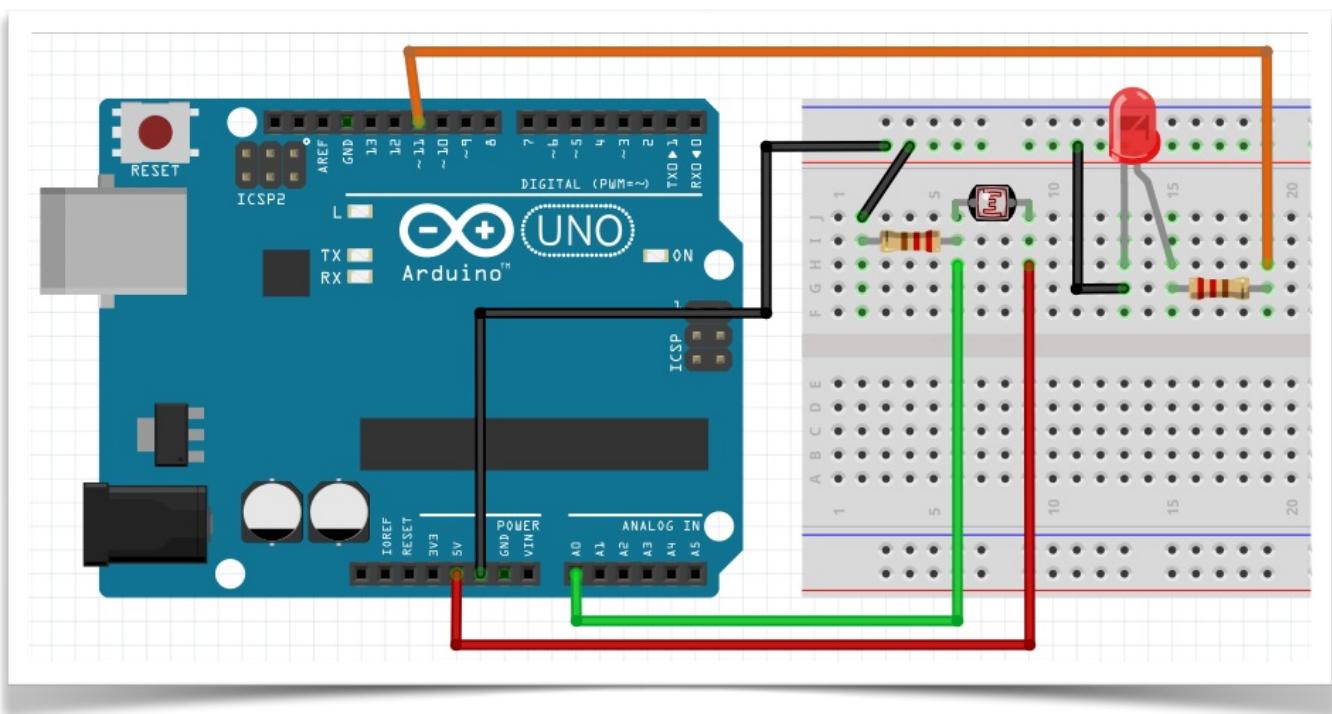


**Фоторезисторы** являются отличным выбором для решения задач вроде "вокруг темно или светло", "есть ли что-то перед датчиком (что ограничивает поступление света)", "какой из участков имеет максимальный уровень освещенности".

Так как фоторезисторы по сути являются **сопротивлением**, они **не имеют полярности**. Это значит, что вы можете их подключать их ноги 'как угодно' а они будут работать!

Фоторезисторы реально **неприхотливы**. Вы можете их припаять, установить их на монтажную плату, использовать клипсы для подключения. Единственное, чего стоит делать - слишком часто изгибать 'ноги', так как они запросто могут отломаться.

**Самый** простой вариант использования: подключить одну ногу к источнику питания, вторую - к земле через понижающий резистор. После этого точка между резистором с постоянным номиналом и переменным резистором - фоторезистором - подключается к аналоговому входу микроконтроллера. На рисунке ниже показана схема подключения к Arduino и код.



```

int fotosensor = 0;
int d = 9;

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

void loop()
{
    int val = analogRead(fotosensor); // задаём переменную, на которую будем получать
                                     // сигналы с фотодиода
    val = constrain(val, 850, 1023); // ограничиваем и дополняем получаемыми данными
                                     // диапазон значений с фотодиода
    int Level = map(val, 850, 1023, 0, 255); // вписываем диапазон сигналов фотодиода в
                                     // диапазон сигналов яркости светодиода
    analogWrite(d, Level);
}

```

## Сервопривод

От простого к всё более и более сложному, переходим на сервоприводы. Что такое **сервопривод**? Сервопривод (сервомотор) является важным элементом при конструировании различных роботов и механизмов. Это точный исполнитель, который имеет **обратную связь**, позволяющую точно управлять движениями механизмов. Другими словами, получая на входе значение управляющего сигнала, сервомотор стремится поддерживать это значение на выходе своего исполнительного элемента.

Обратная связь (ОС) в широком смысле означает отзыв, отклик, ответную реакцию на какое-либо действие или событие. Спасибо Википедии за это.

Сервоприводы широко используются для **моделирования** механических движений роботов. Сервопривод состоит из **датчика** (скорости, положения и т.п.), **блока управления** приводом из механической системы и **электронной схемы**. Редукторы (шестерни) устройства выполняют из металла, карбона или пластика. Пластиковые шестерни сервомотора не выдерживают сильные нагрузки и удары.

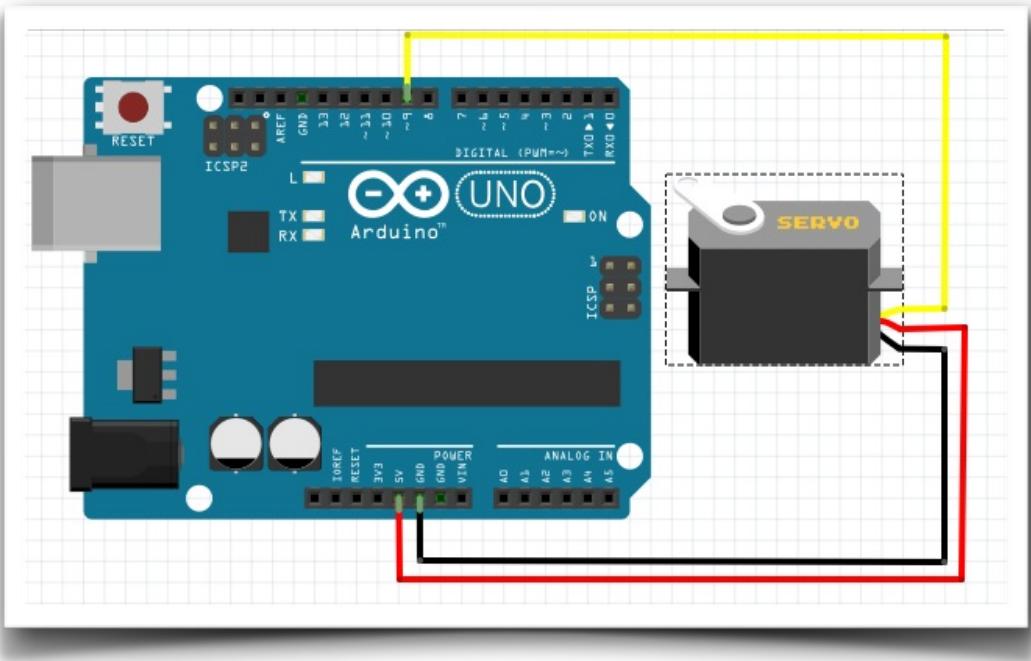


Сервомотор имеет встроенный **потенциометр**, который соединен с выходным валом. Поворотом вала, сервопривод меняет значение напряжения на потенциометре. Плата **анализирует** напряжение входного сигнала и **сравнивает** его с напряжением на потенциометре, исходя из полученной разницы, мотор будет вращаться до тех пор пока не выравняет напряжение на выходе и на потенциометре.

А вот он (слева) :) Спасибо, КЭП.

Схема подключения сервопривода к Arduino обычно следующая: **черный** провод присоединяется к GND, **красный** провод присоединяется к 5V, **оранжевый/желтый** провод присоединяется к аналоговому выводу с ШИМ (Широтно Импульсная Модуляция). Управление сервоприводом на Ардуино достаточно просто, но по углам поворота сервомоторы бывают на **180°** и **360°**, что следует учитывать в робототехнике.

Как же всё это выглядит на практике? Easy (^\_~) вот схема подключения и код.



```
#include <Servo.h> // используем библиотеку для работы с сервоприводом
```

```
Servo servo; // объявляем переменную servo типа Servo

void setup()
{
    servo.attach(10); // привязываем привод к порту 10
}

void loop()
{
    servo.write(0); // ставим вал под 0
    delay(2000); // ждем 2 секунды
    servo.write(180); // ставим вал под 180
    delay(2000); // ждем 2 секунды
}
```

И это всё? Ну, если вам показалось это лёгким, давайте сделаем с сервомашинкой что-нибудь посложнее. Объединим сервопривод и кнопку.



**Сделаем** программу, которая будет работать по принципу шлагбаума на парковках: когда машина подъезжает и сует билет в автомат по приёму парковочных талонов, шлагбаум открывается. Машина проезжает, шлагбаум закрывается. Мы же сделаем следующее: В спокойном положении вал сервопривода находится в положении **0**. Когда мы нажмём кнопку, вал сервопривода поднимется (**провернётся**) на определенное количество градусов, а при отпускании кнопки будет **возвращаться** в своё первоначальное состояние.

```

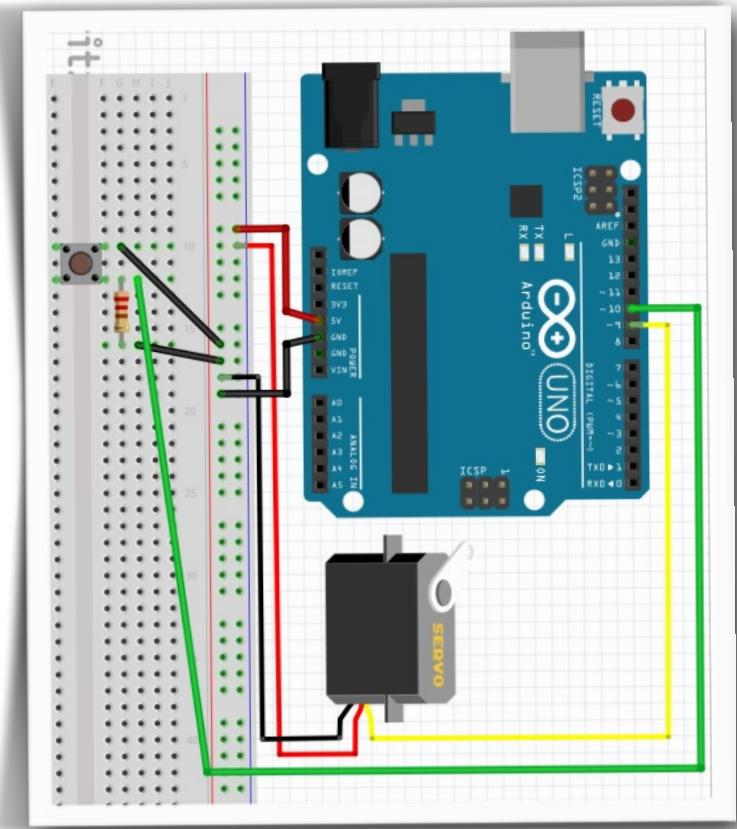
#include <Servo.h>

Servo servo;
int servoPin = 9;
int button = 10;

void setup()
{
    pinMode(button, INPUT);
    servo.attach(servoPin);
}

void loop()
{
    if (digitalRead(button) == HIGH)
    {
        servo.write(45);
    }
    else
    {
        servo.write(0);
    }
    delay(500);
}

```



## Самостоятельная работа № 2

На этот раз придётся пораскинуть мозгами, чтобы всё получилось правильно. **Задача** состоит в следующем: Взять фотодиод, сервомашинку, и заставить последнюю двигаться на определенный угол в зависимости от степени освещения.

Для решения данной задачи надо вспомнить некоторые команды из предыдущих задач.

### Последовательность действий:

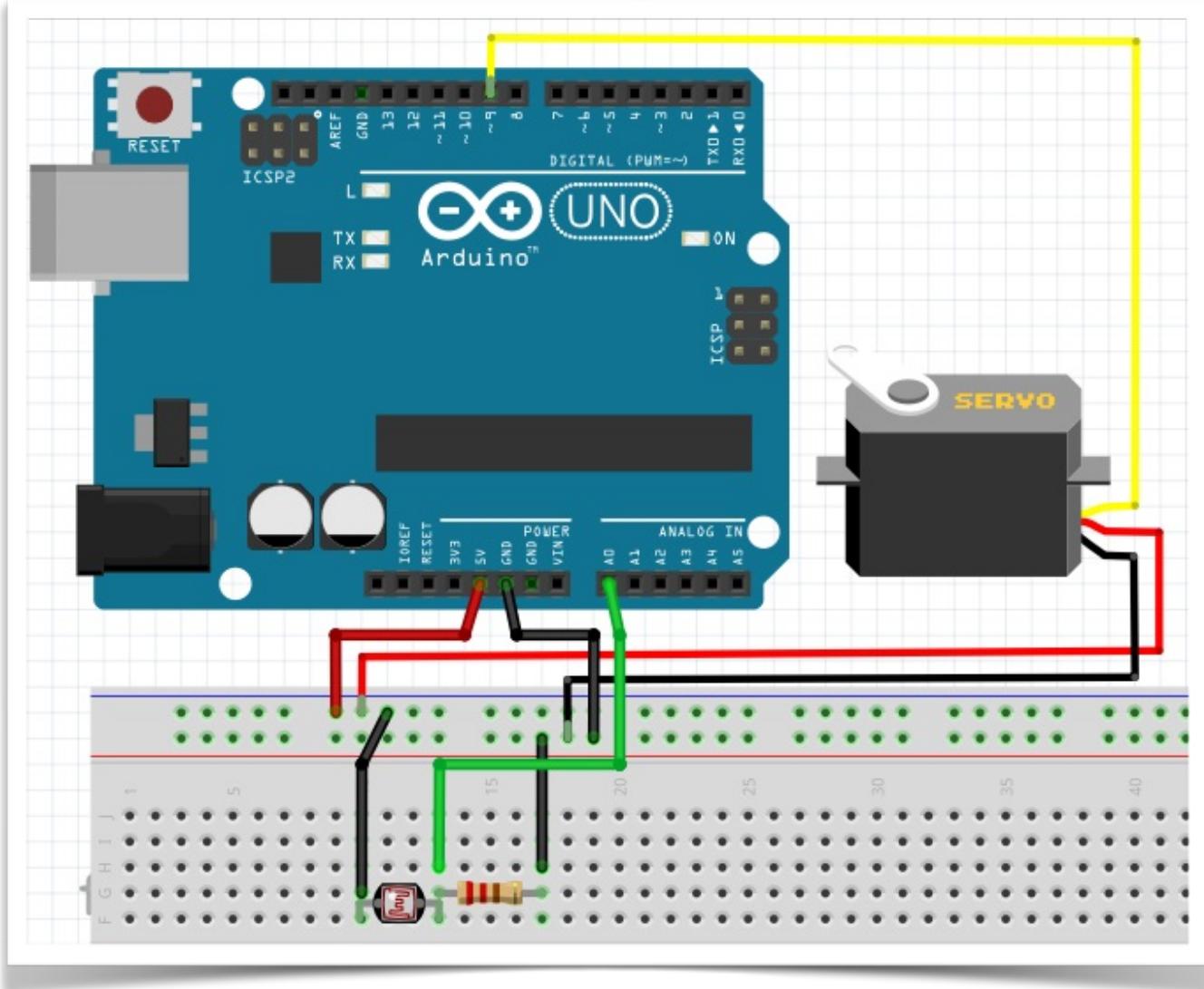
1. Получить какие-то данные с фотодиода.
2. Создать переменную, которая будет принимать на себя эти данные.
- 3 . Совместить диапазон полученных данных с фотодиода с диапазоном данных уровня яркости светодиода.
4. Обозначить это как новую переменную.
5. "сказать" сервомашинке делать то, что велит ей последняя переменная.



"Цель учения — достичь наибольшего удовлетворения в получении знаний".

Сюнь-цы

А вот и ответ!) Я надеюсь вы не подсматривали сразу, ведь так не интересно :c



```
#include <Servo.h>
Servo servo;
int servoPin = 9;
int photores = 0;

void setup()
{
    servo.attach(servoPin);
}

void loop()
{
    int photo = analogRead(photores); // переменная принимающая значения с
фоторезистора
    int pos = map(photo, 0, 1023, 0, 180); // совмещаем значения двух функций в одну новую
    servo.write(pos); // выдаём результат
}
```

## Музыка на Arduino

Несмотря на то, что Arduino это аппаратная платформа, на ней тоже можно делать **музыку**. Правда **восьмибитную**, но это никак не останавливает энтузиастов переделывать уже известные хиты :) Одним из примеров являются переделанные хиты, воспроизведённые на **дискетных** приводах, так называемая **Arduino floppy music**.



К сожалению, ещё толком не освоив основы основ Arduino, говорить про музыку на дискетах рановато, но её можно делать и на буззерах! Что такое буззер (**buzzer**)? Буззер - это небольшая пьезо пищалка, которая может генерировать звуки определённой частоты. В принципе её можно сравнить с небольшим **динамиком**, которым она, по сути, и является.



Можно потихоньку приступать. Но что же нам надо для того, чтобы начать писать музыку на Arduino? Для начала, как ни странно, **ноты**. Здесь они выглядят либо в виде чисел, в основном трёхзначных, либо в виде констант. Для первой программы сделаем в виде чисел, чтобы было понятнее.

так непонятно же! Ну смотрите: когда вы издаёте звук (говорите, щёлкаете пальцами, стучите по батареям, взрываете петарды, играете на пианино или гитаре) в воздухе распространяются звуковые **волны**. Как и любая волна, звуковая имеет такие параметры как **амплитуда** и **частота**. Знакомо? Я думаю да, по крайней мере слышали. Каждая нота распространяется с определённой частотой, а эти самые **цифры** в программе и **являются частотой**.  
Круг замкнулся.

Перед вами **раскладка** пианино с названием нот и их частотой звучания. Данная табличка может вам понадобится если вы захотите заранее задать ноты как константы, чтобы не писать большой код из чисел.

Конечно, от этого ничего не изменится, потому что вместо огромного количества чисел у вас будет огромное количество названий нот, хе хе хе.

Но никто и не говорил, что программировать это очень быстро. В различных устройствах, пусть даже и не очень большого размера коды, зашифтованные в микроконтроллеры имеют огроменные размеры, потому что им надо очень точно рассчитывать те или иные значения, поступающие из вне или бороздящие внутри самого контроллера.

То же самое и с нашим кодом для собственной музыки. Он не сложный, но большой.

На следующей странице приложен **код** песни. Сама мелодия состоит из двух целочисленных массивов. Один отвечает за **порядок** нот в песне, а второй за **скорость** воспроизведения последних.

Попробуйте скопировать данный код себе на компьютер, подключив Arduino и пищалку.

Сможете ли вы угадать, какая мелодия спрятана в коде? ;)

A0	27.5	A0#	29.135
B0	30.068		
C1	32.703	C1#	34.648
D1	36.708	D1#	38.891
E1	41.203		
F1	46.654	F1#	46.249
G1	48.999	G1#	51.913
A1	55.000	A1#	58.270
B1	61.735		
C2	65.406	C2#	69.296
D2	73.416	D2#	77.782
E2	82.407		
F2	87.307	F2#	92.499
G2	97.999	G2#	103.83
A2	110.000	A2#	116.54
B2	123.47		
C3	130.81	C3#	138.59
D3	146.83	D3#	155.56
E3	164.81		
F3	174.61	F3#	185.00
G3	196.00	G3#	207.65
A3	220.00	A3#	233.08
B3	246.94		
C4	261.63	C4#	277.18
D4	293.66	D4#	311.13
E4	329.63		
F4	349.23	F4#	369.99
G4	392.00	G4#	415.30
A4	440.00	A4#	466.16
B4	493.88		
C5	523.25	C5#	554.37
D5	587.33	D5#	622.25
E5	659.25		
F5	698.46	F5#	739.99
G5	783.99	G5#	830.61
A5	880.00	A5#	932.33
B5	987.77		
C6	1046.5	C6#	1108.7
D6	1174.7	D6#	1244.5
E6	1318.5		
F6	1396.9	F6#	1480.0
G6	1568.0	G6#	1681.2
A6	1760.0	A6#	1864.7
B6	1979.5		
C7	2093.0	C7#	2217.5
D7	2349.3	D7#	2489.0
E7	2637.0		
F7	2793.8	F7#	2960.0
G7	3136.0	G7#	3322.4
A7	3520.0	A7#	3729.3
B7	3951.1		
C8	4186.0		

```

const int Pin_tone = 9; // номер порта зуммера
const byte COUNT_NOTES = 39; // количество нот

// частоты ноты
int frequencies[COUNT_NOTES] =
{
  392, 392, 392, 311, 466, 392, 311, 466, 392,
  587, 587, 587, 622, 466, 369, 311, 466, 392,
  784, 392, 392, 784, 739, 698, 659, 622, 659,
  415, 554, 523, 493, 466, 440, 466,
  311, 369, 311, 466, 392
};

// длительность нот
int durations[COUNT_NOTES] =
{
  350, 350, 350, 250, 100, 350, 250, 100, 700,
  350, 350, 350, 250, 100, 350, 250, 100, 700,
  350, 250, 100, 350, 250, 100, 100, 100, 450,
  150, 350, 250, 100, 100, 100, 450,
  150, 350, 250, 100, 750
};

void setup()
{
  pinMode(Pin_tone, OUTPUT); // настраиваем контакт на выход
}

void loop()
{
  for (int i = 0; i <= COUNT_NOTES; i++ )
  { // цикл от 0 до количества нот
    tone(Pin_tone, frequencies[i], durations[i] * 2); // включаем звук, определенной частоты
    delay(durations[i] * 2); // пауза для заданной ноты
    noTone(Pin_tone); // останавливаем звук
  }
}

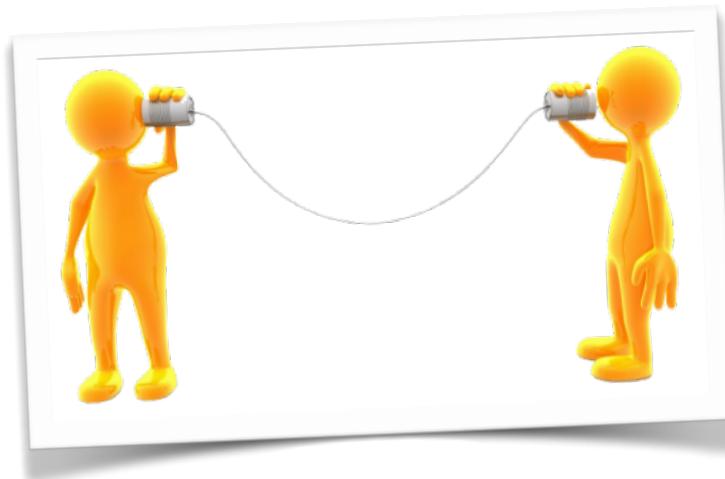
```



"Порядок больше всего помогает ясному усвоению".  
Цицерон Марк Туллий

## Serial port

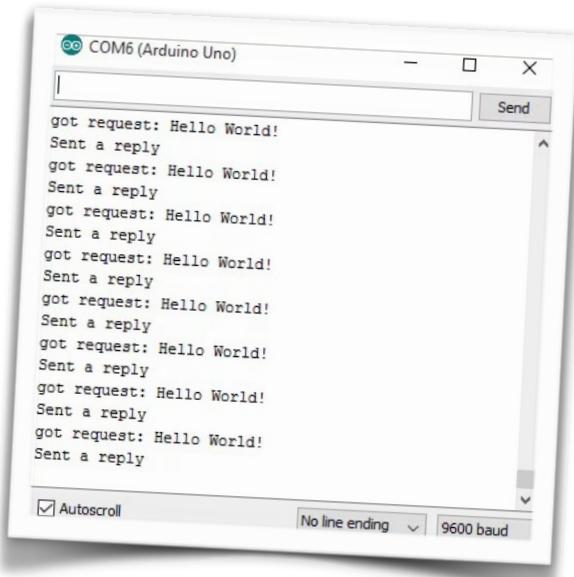
### Или как Arduino может взаимодействовать с внешними устройствами



Набор функций **Serial** служит для связи устройства Arduino с компьютером или другими устройствами, поддерживающими последовательный интерфейс обмена данными. Все платы Arduino имеют хотя бы один **последовательный** порт (UART, иногда называют USART). Для обмена данными Serial используют **цифровые** порты ввод/вывода 0 (**RX**) и 1 (**TX**), а также USB порт. Важно учитывать, что если вы используете функции Serial, то **нельзя одновременно** с этим использовать порты 0 и 1 для других целей.

Среда разработки Arduino имеет встроенный монитор последовательного интерфейса (**Serial monitor**). Для начала обмена данными необходимо запустить монитор нажатием кнопки Serial monitor и выставить ту же скорость связи (baud rate), с которой вызвана функция begin().

Плата Arduino Mega имеет три дополнительных последовательных порта: Serial1 на портах 19 (RX) и 18 (TX), Serial2 на портах на портах 17 (RX) и 16 (TX), Serial3 на портах на портах 15 (RX) и 14 (TX). Чтобы использовать эти порты для связи с компьютером понадобятся дополнительные адаптеры USB-to-serial, т.к. они не подключены к встроенному адаптеру платы Mega. Для связи с внешним устройством через последовательный интерфейс соедините TX порт вашего устройства с RX портом внешнего устройства и RX порт вашего устройства с портом TX внешнего и соедините "землю" на устройствах. (**Важно!** Не подключайте эти порты напрямую к RS232 порту, это может повредить плату).



Сам код для общения с монитором порта выглядит следующим образом:

```
void setup()
{
    Serial.begin(9600); // задаём частоту, по
которой мы будем общаться с Arduino
}

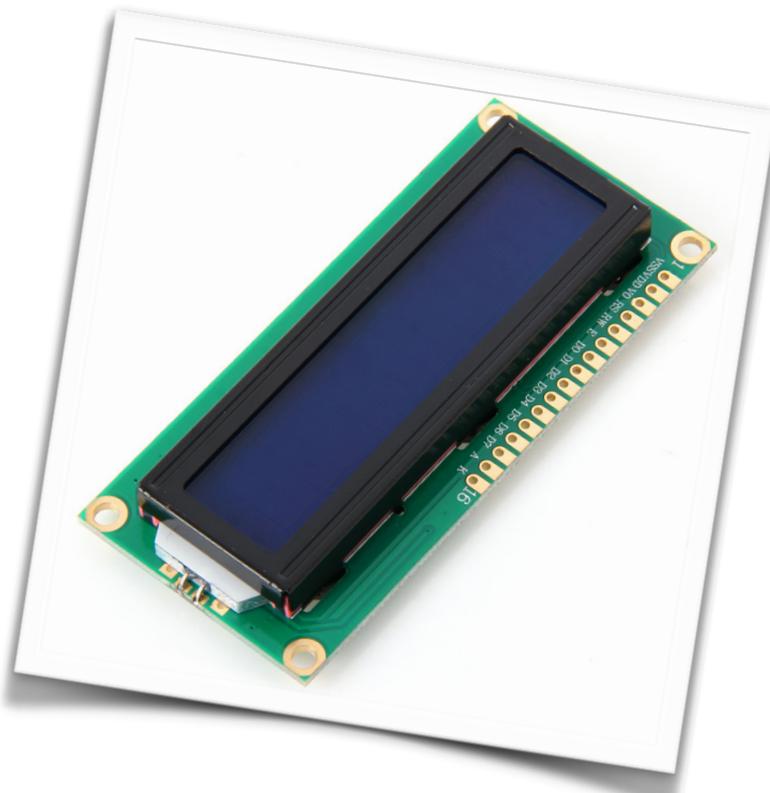
void loop()
{
    Serial.println("hello world!"); // пишем в
мониторе порта текст
    delay (1000); // задаём скорость
повтора текста
}
```



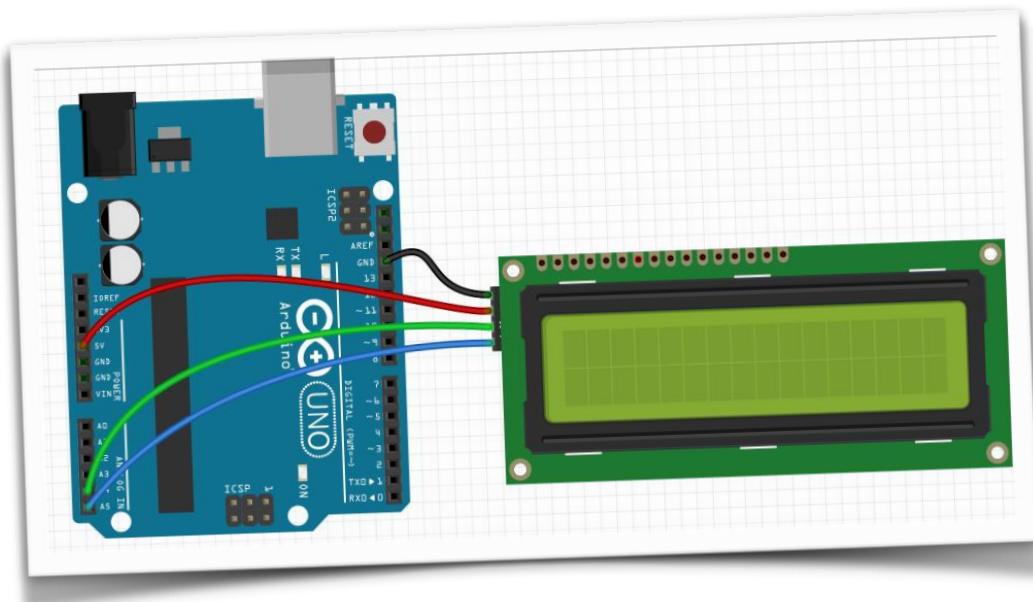
## I2C

**I2C** — последовательная двухпроводная **шина** для связи интегральных схем внутри электронных приборов, известна, как **I<sup>2</sup>C** или **IIC** (англ. Inter-Integrated Circuit). I<sup>2</sup>C была разработана фирмой Philips в начале 1980-х годов, как простая 8-битная шина для **внутренней** связи между схемами в управляющей электронике (например, в компьютерах на материнских платах, в мобильных телефонах и т.д.).

В простой системе I<sup>2</sup>C может быть **несколько ведомых** устройств и **одно ведущее** устройство, которое инициирует передачу данных и синхронизирует сигнал. К линиям SDA (линия данных) и SCL (линия синхронизации) можно подключить несколько ведомых устройств. Часто ведущим устройством является контроллер Arduino, а ведомыми устройствами: **часы** реального времени или **LCD Display**.



Жидкокристаллический дисплей 1602 с I2C модулем подключается к плате Ардуино всего **4** проводами — 2 провода данных и 2 провода питания. Подключение дисплея 1602 проводится стандартно для шины I2C: вывод SDA подключается к порту A4, вывод SCL – к порту A5. Питание LCD дисплея осуществляется от порта +5V на Arduino. Смотрите подробнее схему подключения ЖК монитора 1602 на фото ниже.



После подключения LCD монитора к Ардуино через I2C вам потребуется **установить** библиотеку LiquidCrystal\_I2C.h для работы с LCD дисплеем по интерфейсу I2C и библиотека Wire.h (имеется в стандартной программе Arduino IDE)

```
#include <Wire.h> // библиотека для управления устройствами по I2C
#include <LiquidCrystal_I2C.h> // подключаем библиотеку для LCD 1602

LiquidCrystal_I2C lcd(0x27,20,2); // присваиваем имя lcd для дисплея 20x2

void setup()
{
    lcd.init(); // инициализация LCD дисплея
    lcd.backlight(); // включение подсветки дисплея

    lcd.setCursor(0,0); // ставим курсор на 1 символ первой строки
    lcd.print("I LOVE"); // печатаем сообщение на первой строке

    lcd.setCursor(0,1); // ставим курсор на 1 символ второй строки
    lcd.print("ARDUINO"); // печатаем сообщение на второй строке
}

void loop()
{
    // процедура void loop() в скетче не используется
}
```



## Самостоятельная работа № 3

Попробуйте сделать программу, которая выводит на экран дисплея **фразу** «hello world!», **а затем** через некоторое время фразу «my project»

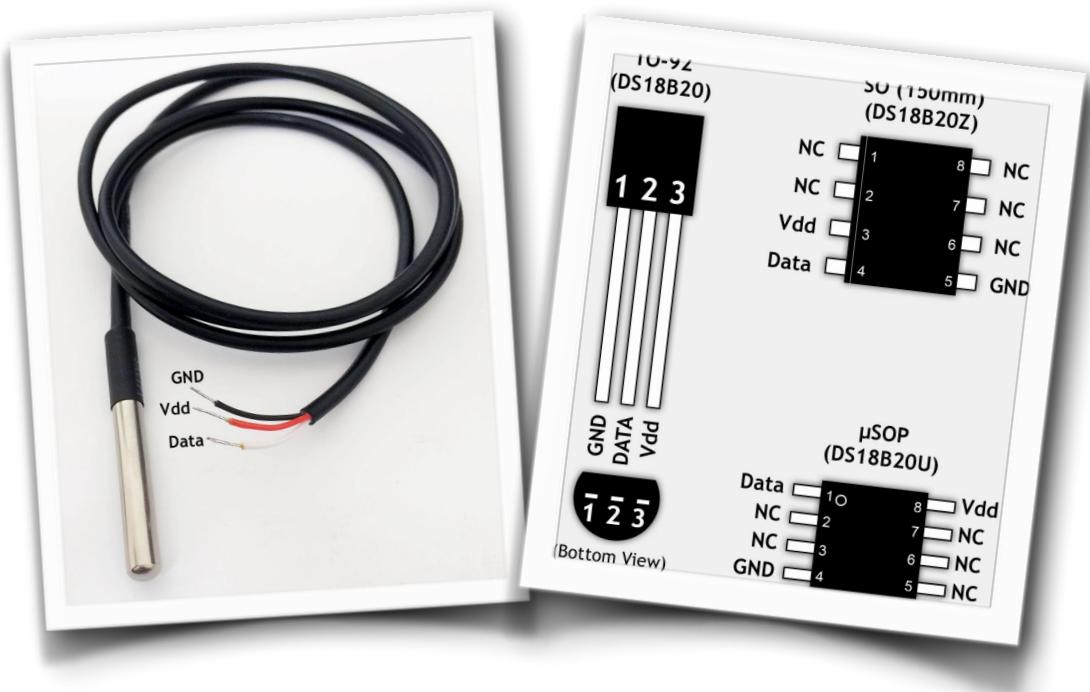
Вот кусок кода-ответа к задаче :)

```
void loop()
{
    lcd.clear(); // очищаем дисплей
    lcd.setCursor(0,0); // ставим курсор в начальное положение первой строки
    lcd.print("hello world!"); // пишем
    delay(1000); // делаем задержку
    lcd.setCursor(0,1); // ставим курсор в начальное положение второй строки
    lcd.print("my project"); // пишем
}
```

## Датчик температуры Ds18b20

**DS18B20** - это цифровой датчик температуры. Датчик очень прост в использовании. Во-первых, он цифровой, а во вторых - у него всего лишь **один контакт**, с которого мы получаем полезный сигнал. То есть, вы можете подключить к одному Arduino одновременно огромное количество этих сенсоров. Пинов будет более чем достаточно. Мало того, вы даже можете подключить несколько сенсоров к одному pinu на Arduino! Но обо всем по порядку.

DS18B20 имеет различные **форм-факторы**. Так что выбор, какой именно использовать, остается за вами. Доступно три варианта: 8-Pin **SO** (150 mils), 8-Pin **μSO**P, и 3-Pin **TO-92**. Серфинг по eBay или Aliexpress показывает, что китайцы предлагают версию **TO-92 в водозащищенном корпусе**. То есть, вы можете смело окунать подобное чудо в воду, использовать под дождем и т.д. и т.п. Эти сенсоры изготавливаются с тремя выходными контактами (черный - **GND**, красный - **Vdd** и белый - **Data**).



Различные **форм-факторы** датчиков DS18B20 приведены на рисунке выше.

DS18B20 удобен в использовании. Запитать его можно через контакт **data** (в таком случае вы используете всего два контакта из трех для подключения!). Сенсор работает в диапазоне напряжений от **3.0 В** до **5.5 В** и измеряет температуру в диапазоне от **-55°C** до **+125°C** (от -67°F до +257°F) с точностью ±0.5°C (от -10°C до +85°C).

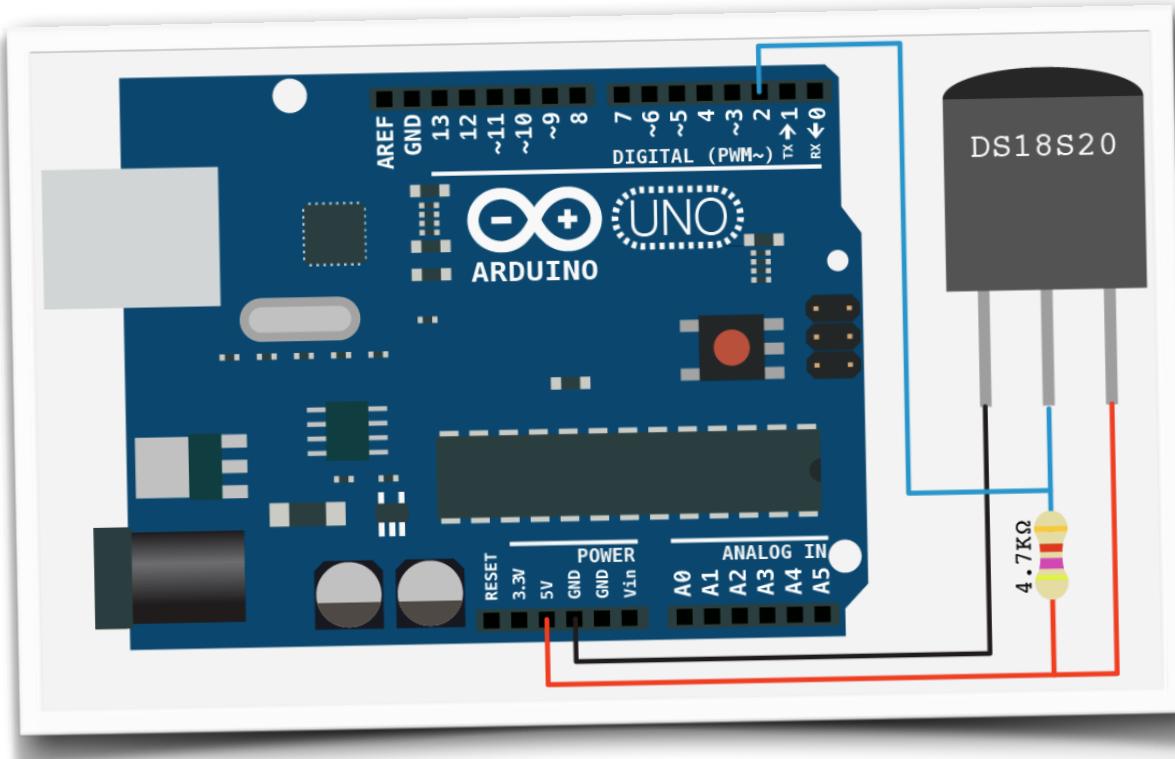
Еще одна крутая фича: вы можете подключить **параллельно** вплоть до **127 датчиков!** и считывать показания температуры с каждого отдельно. Не совсем понятно, в каком проекте подобное может понадобится, но подключить два сенсора и контролировать температуру в холодильнике и морозильной камере можно. При этом вы оставите свободными кучу пинов на Arduino... В общем, фича приятная.

Датчик подключается элементарно. Контакт **GND** с DS18B20 подключается к **GND** на Arduino. Контакт **Vdd** с DS18B20 подключается к **+5V** на Arduino. Контакт **Data** с DS18B20 подключается к любому цифровому pinу на Arduino. В данном примере используется **пин 2**.

Единственное, что необходимо добавить из внешней дополнительной обвязки - это подтягивающий **резистор на 4.7 КОм**.

**Схема** подключения DS18B20 к Arduino показана ниже (в скетче, который будет приведен ниже, проверьте строки 10 и 65. В них указаны пины, к которым вы подключали контакт сигнала с датчика и режим питания!)

Ещё одно уточнение. Для работы с данным датчиком понадобится библиотека `<OneWire.h>`, которая поможет упростить наш код с данным датчиком, а так же позволит arduino увидеть его.



А вот и сам код:

```
#include <OneWire.h> // пример использования библиотеки OneWire DS18S20, DS18B20,
DS1822
OneWire ds(2); // на пине 2 (нужен резистор 4.7 КОм)

void setup(void)
{
    Serial.begin(9600);
}

void loop(void)
{
    byte i;
    byte present = 0;
    byte type_s;
    byte data[12];
    byte addr[8];
    float celsius;
    if (!ds.search(addr)) // получение адреса
    {
        ds.reset_search();
        delay(250);
        return;
    }
    if (OneWire::crc8(addr, 7) != addr[7]) // получение адреса
    {
        return;
    }
    Serial.println();
    switch (addr[0]) // для определения типа датчика
    {
        case 0x10:
        type_s = 1;
        break;
        case 0x28:
        type_s = 0;
        break;
        case 0x22:
        type_s = 0;
        break;
        default:
        return;
    }
    ds.reset();
    ds.select(addr);
    ds.write(0x44); // начинаем преобразование, используя ds.write(0x44,1) с "паразитным"
питанием
    delay(1000); // 750 может быть достаточно, а может быть и не хватит
    // мы могли бы использовать тут ds.depower(), но reset позаботится об этом
    present = ds.reset();
    ds.select(addr);
    ds.write(0xBE);
    for (i = 0; i < 9; i++)
    {
        // нам необходимо 9 байт
        data[i] = ds.read();
    }

    Serial.println();
```

```

// конвертируем данный в фактическую температуру
// так как результат является 16 битным целым, его надо хранить в
// переменной с типом данных "int16_t", которая всегда равна 16 битам,
// даже если мы проводим компиляцию на 32-х битном процессоре
int16_t raw = (data[1] << 8) | data[0];
if (type_s)
{
    raw = raw << 3; // разрешение 9 бит по умолчанию
    if (data[7] == 0x10)
    {
        raw = (raw & 0xFFFF) + 12 - data[6];
    }
}
else
{
    byte cfg = (data[4] & 0x60); // при маленьких значениях, малые биты не определены,
давайте их обнулим
    if (cfg == 0x00) raw = raw & ~7; // разрешение 9 бит, 93.75 мс
    else if (cfg == 0x20) raw = raw & ~3; // разрешение 10 бит, 187.5 мс
    else if (cfg == 0x40) raw = raw & ~1; // разрешение 11 бит, 375 мс
    // разрешение по умолчанию равно 12 бит, время преобразования - 750 мс
}
celsius = (float)raw / 16.0;
Serial.print(" Temperature = ");
Serial.print(celsius);
Serial.print(" Celsius, ");
}

```

## Упрощая задачи

Не думайте, что изучая данный материал, вы сможете программировать только так, как написано либо здесь, либо в интернете. Это не так. В данном случае будет продемонстрирован код работы ШИМ без фотодиода, а только лишь с одним светодиодом.

Вот и код

```

int diode = 11; // подключаем светодиод
int bri = 0; // переменная уровня яркости
int fade = 3; // шаг изменения уровня яркости
unsigned long currentTime; // переменная отображающая настоящее время свечения
unsigned long loopTime; // переменная отображающая время повтора свечения
void setup()
{
    pinMode (diode, OUTPUT);
    currentTime = millis(); // задаём настоящему времени шкалу измерения (в миллисекундах)
    loopTime = currentTime; // приравниваем настоящее время к времени повтора
}

void loop()
{
    currentTime = millis();
    if (currentTime >= (loopTime + 20)) // прибавляем шаг к яркости
    {
        analogWrite(diode, bri);
        bri = bri + fade;
        if (bri == 0 || bri == 255) // уменьшаем или увеличиваем яркость если она достигла
максимума или минимума
    }
}

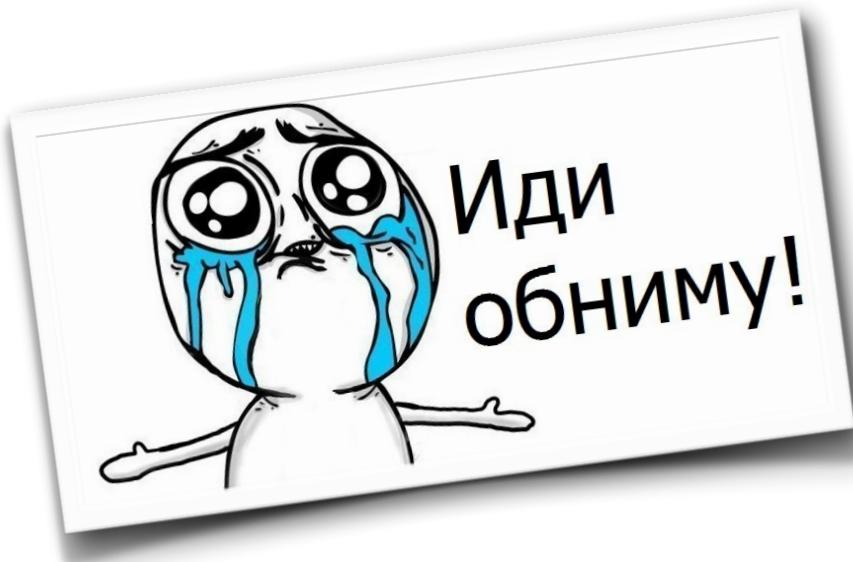
```

```
    fade = -fade; // для этого инвертируем шаг (направление шага)
}
loopTime = currentTime;
}
```

## В заключении...

Эх, ребятки, как бы мне не хотелось подходить к концу, чего я конечно же не ожидал от данной книги, которая рассчитывалась как **небольшой проект** на 10-20 страниц, а вышла на **полстони**, но **большую часть основ взаимодействия с arduino вы уже освоили**, читая эту книгу. Я надеюсь мои объяснения были понятны для вас и вы с большим удовольствием листали данную книженцию в поисках **нового и интересного. Не останавливайтесь** только лишь на прочтении, **двигайтесь** дальше, **пробуйте** создавать свои проекты на платформе arduino, а так же **осваивайте** другие труды по теме электричества, ибо с ними вы как рыба в воде :)

Что можно сказать по итогу... **Поздравляю** вас с окончанием изучения курса "**Arduino "На пальцах""**! Ждите второй части книги, в которой поговорим о более глубоких вещах, связанными с данной аппаратной платформой.



До встречи!

# Содержание

<b>Введение.....</b>	<b>2</b>
<b>Начало начал.....</b>	<b>3</b>
<b>Закон Кирхгофа.....</b>	<b>4</b>
<b>Мощность и потери.....</b>	<b>5</b>
<b>Основные элементы электрических схем.....</b>	<b>5</b>
<b>Резистор.....</b>	<b>5</b>
<b>Транзистор.....</b>	<b>7</b>
<b>Диод.....</b>	<b>9</b>
<b>Конденсатор.....</b>	<b>10</b>
<b>Индуктивность.....</b>	<b>12</b>
<b>От теории к практике.....</b>	<b>13</b>
<b>Необходимости.....</b>	<b>14</b>
<b>Но сначала.....</b>	<b>16</b>
<b>Строение Arduino Uno.....</b>	<b>17</b>
<b>Время программировать!.....</b>	<b>18</b>
<b>Первая программа.....</b>	<b>18</b>
<b>Заливай!.....</b>	<b>21</b>
<b>Самостоятельная работа № 1.....</b>	<b>22</b>
<b>Новый уровень.....</b>	<b>24</b>
<b>Выбор Arduino.....</b>	<b>28</b>
<b>Основные команды для программирования.....</b>	<b>35</b>
<b>ШИМ.....</b>	<b>37</b>
<b>Автоматическое управление светом.....</b>	<b>40</b>
<b>Сервопривод.....</b>	<b>42</b>
<b>Самостоятельная работа № 2.....</b>	<b>44</b>
<b>Музыка на Arduino.....</b>	<b>46</b>
<b>Serial Port.....</b>	<b>49</b>

I2C.....	50
<b>Самостоятельная работа № 3.....</b>	<b>51</b>
<b>Датчик температуры Ds18b20.....</b>	<b>52</b>
<b>Упрощая задачи.....</b>	<b>55</b>
<b>В заключении.....</b>	<b>56</b>
<b>Черновые листы.....</b>	<b>59</b>
<b>Об авторе и предназначение книги.....</b>	<b>62</b>

void loop()

{

}

## void loop()

{

}

## void loop()

{

}

## Об авторе и предназначение книги

Данная книга является сборником материалов и написана для студентов Белорусского Государственного Университета факультета Радиофизики и Компьютерных Технологий, а так же для всех желающих с целью освоения аппаратной платформы Arduino и изучения основ электроники, схемотехники.

Не исключено, что в данной книге могут содержаться материалы, а в частности и изображения, полностью или частично повторяющие материал, находящийся в открытом доступе в сети интернет.

Автор книги: Менчиков Юрий, студент Белорусского Государственного Университета 4 курса факультета Радиофизики и Компьютерных Технологий.

Почта: [yura.menschikov@icloud.com](mailto:yura.menschikov@icloud.com)



Дата написания книги

24.10.2017