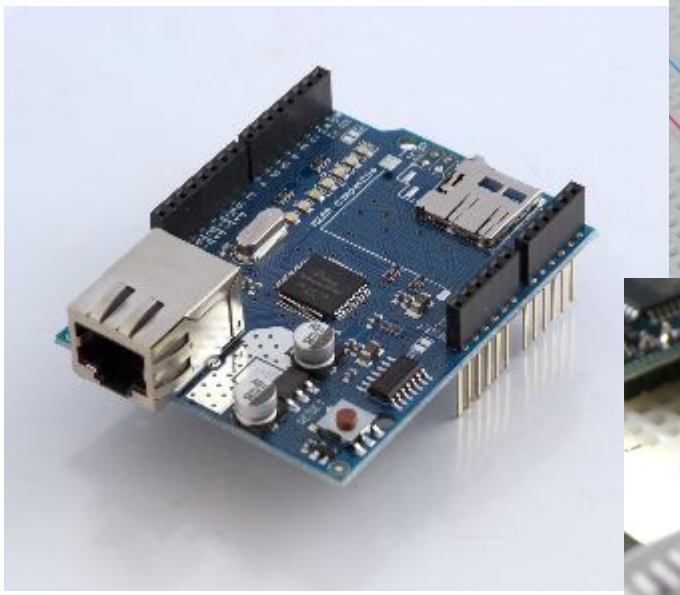
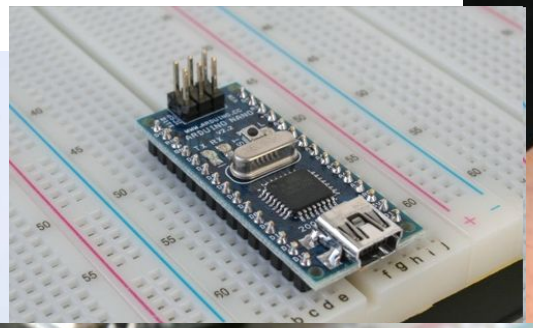
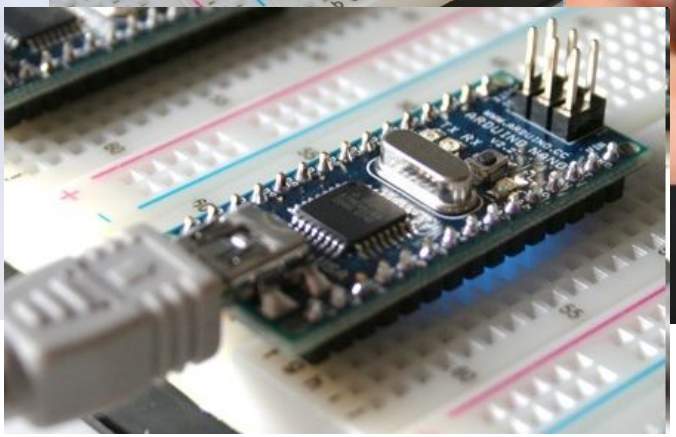
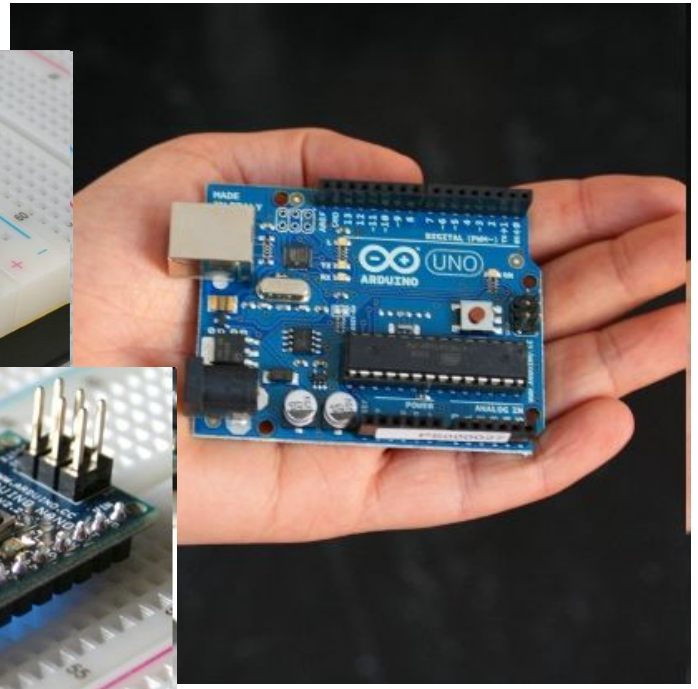
Введение в Arduino



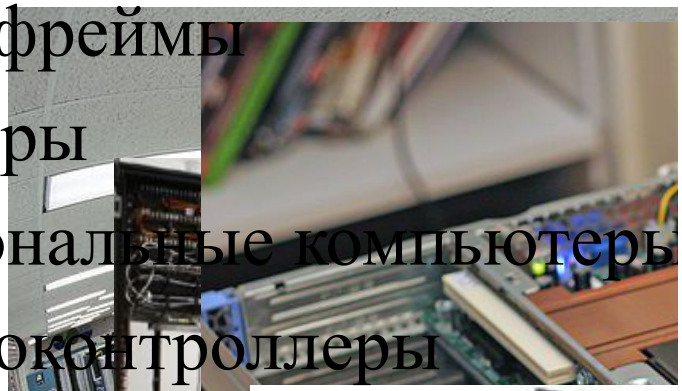


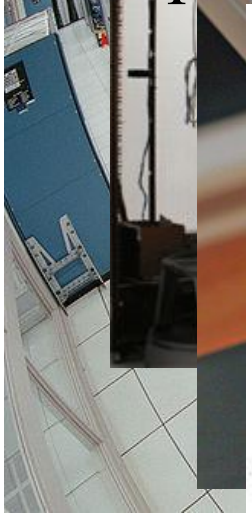


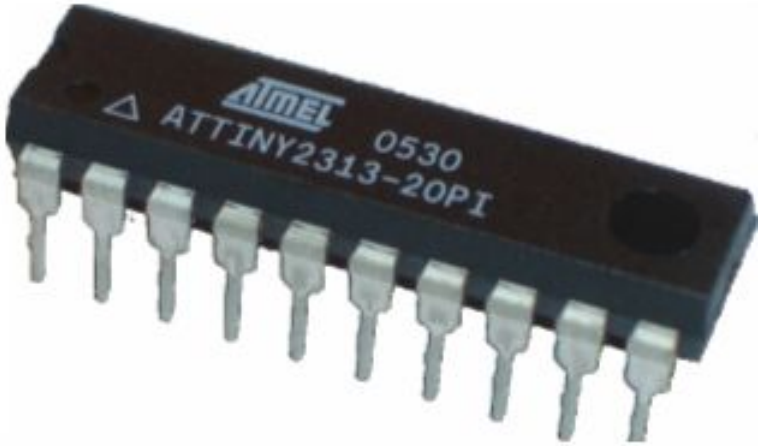


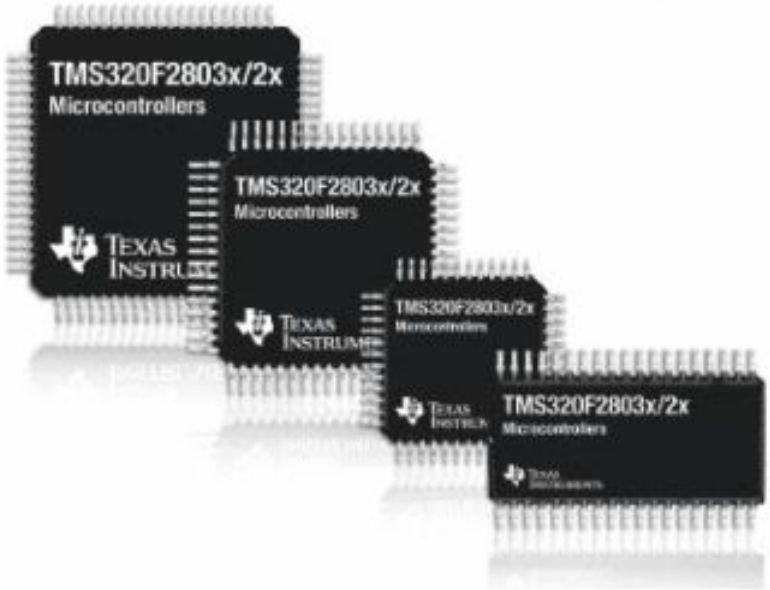
Виды компьютеров

- Суперкомпьютеры
- Мейнфреймы
- Серверы
- Персональные компьютеры
- Микроконтроллеры









Области использования МК

- Промышленность
- Медицина
- Транспорт
- Робототехника
- Бытовая техника, умный дом
- Игрушки

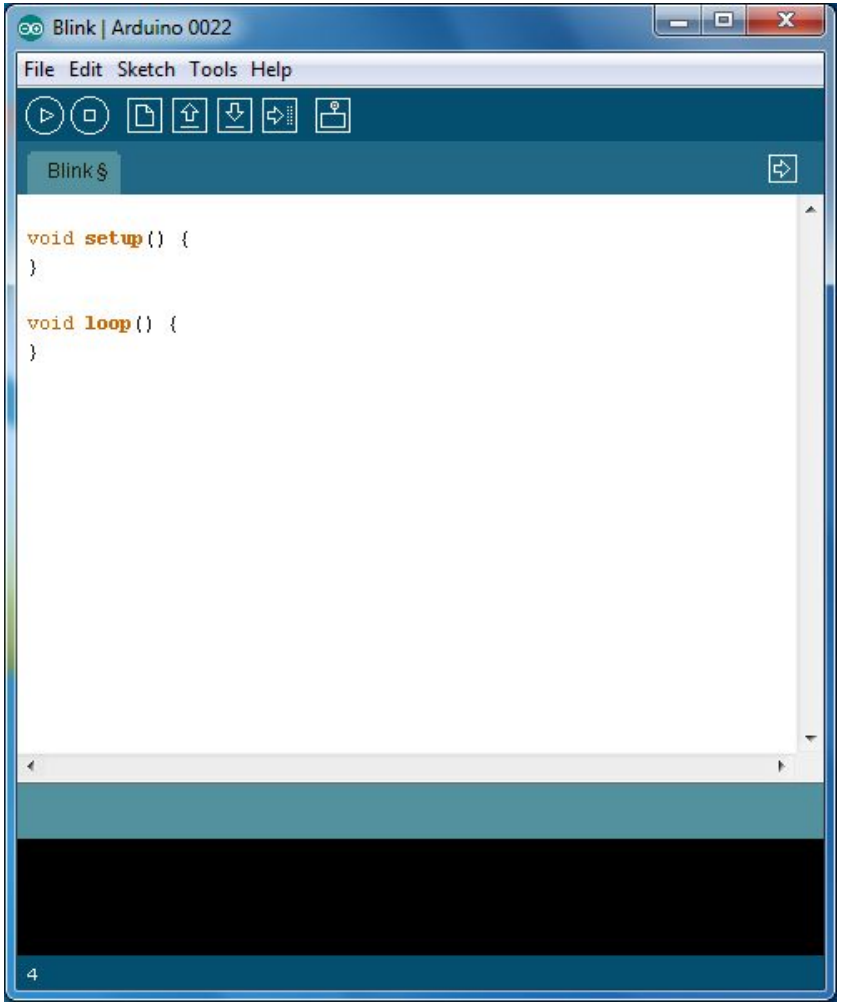
Параметры МК

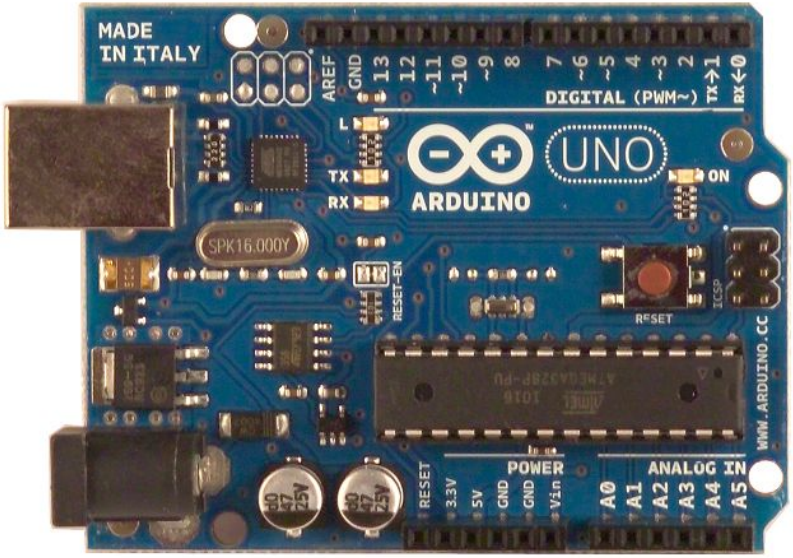
- Наличие/отсутствие
  - ОЗУ, ПЗУ
  - возможности перепрошивки
  - встроенного генератора тактовой частоты
  - сторожевого таймера
  - периферии
- Архитектура: 8, 16, 32 бит
- Различная частота процессора
- Специального назначения

Платформа Arduino

Электронный конструктор и удобная платформа быстрой разработки электронных устройств для новичков и профессионалов

- Среда разработки
- Платы

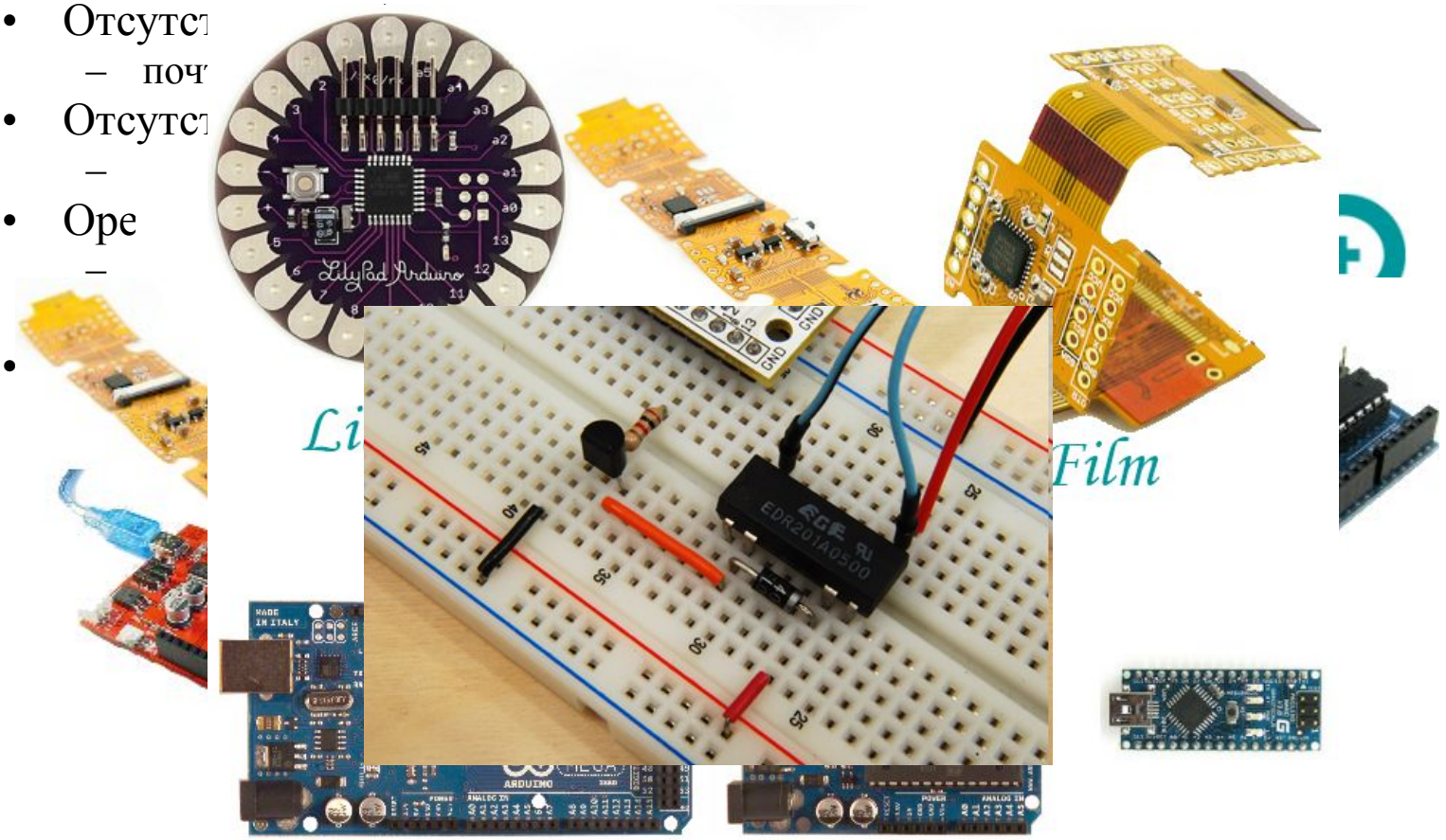




http://http://лекции.техблог.рфhttp://лекции.техблог.рф/http://лекции.техблог.рф/введение_в_ардуиноhttp://лекции.техблог

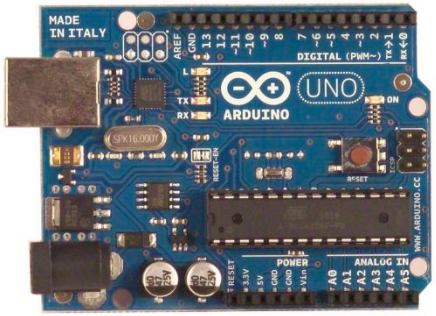
Популярность платформы

- Низкий порог входа в мир МК
- Разнообразие плат. Две версии носимых плат: *LilyPad* и *Seeeduino Film*
- Кроссплатформенность среды разработки. Переносимость кода для разных плат Arduino.
- Отсутст
  - поч
- Отсутст
  -
- Ope
  -
-

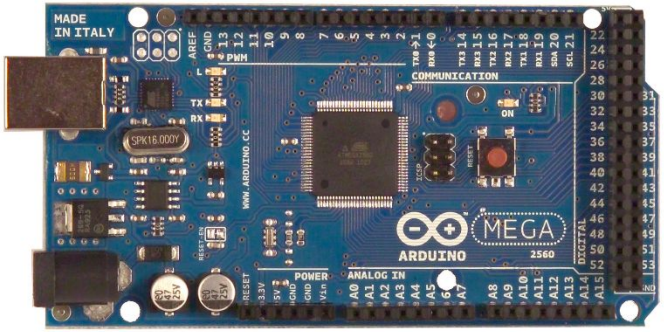


//лекции.техблог.

Основные платы



- Uno
  - базовая платформа Arduino
  - 14 цифровых входов/выходов (из них 6 ШИМ)
  - 6 аналоговых входов
  - 1 последовательный порт UART
  - программируется через USB с токовой защитой
  - дополняется платами расширения



- Mega2560
  - 54 цифровых входа/выхода (из них 14 ШИМ)
  - 16 аналоговых входов
  - 4 последовательных порта UART
  - дополняется платами расширения
  - программируется через USB

- Nano
  - 14 цифровых входов/выходов (6 могут использоваться как выходы ШИМ)
  - 8 аналоговых входов
  - программируется через Mini-USB

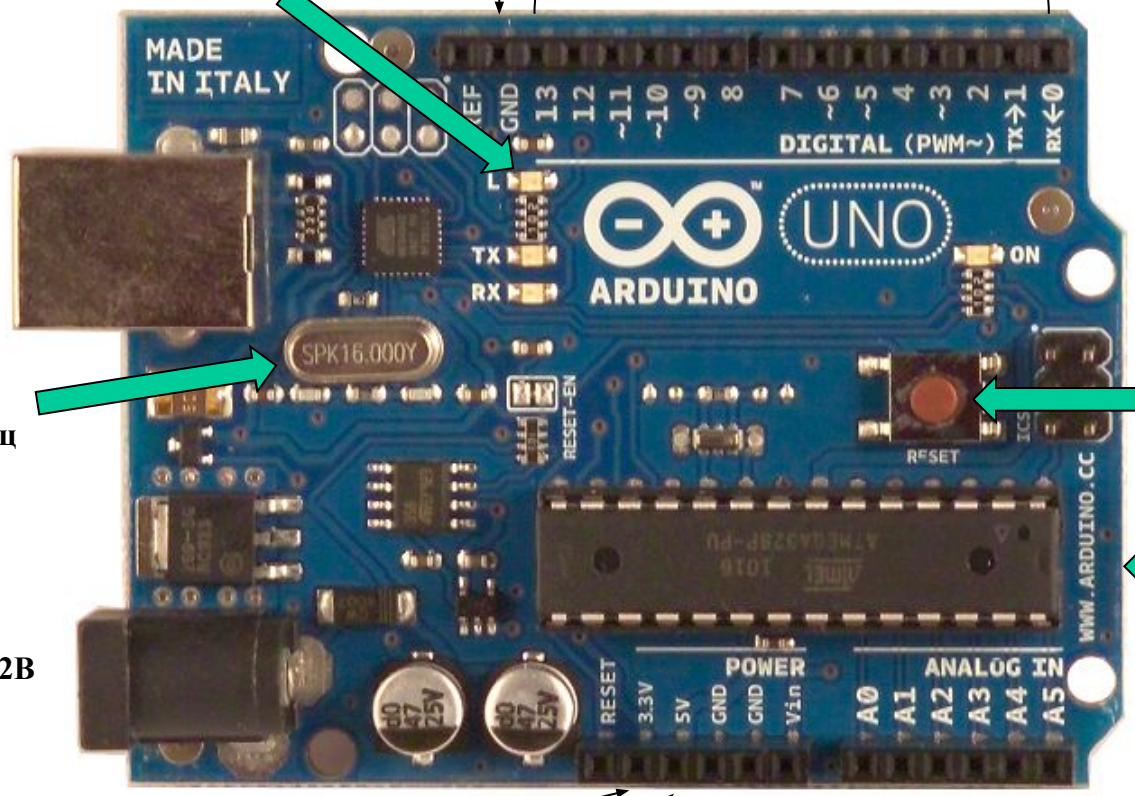
Платы расширения

GSM+GPS

Ethernet

LCD

Плата Arduino Uno



СВЕТОДИОД НА ВЫВОДЕ №13

ЗЕМЛЯ

ЦИФРОВЫЕ ВЫВОДЫ (ШИМ ~)

РАЗЪЕМ USB 5В

КВАРЦЕВЫЙ РЕЗОНАТОР 16 МГц

КНОПКА ПЕРЕЗАГРУЗКИ

МК ATmega328

РАЗЪЕМ ПИТАНИЯ 7-12В ЦЕНТР +

ВЫХОД 3.3В +

ВЫХОД 5В +

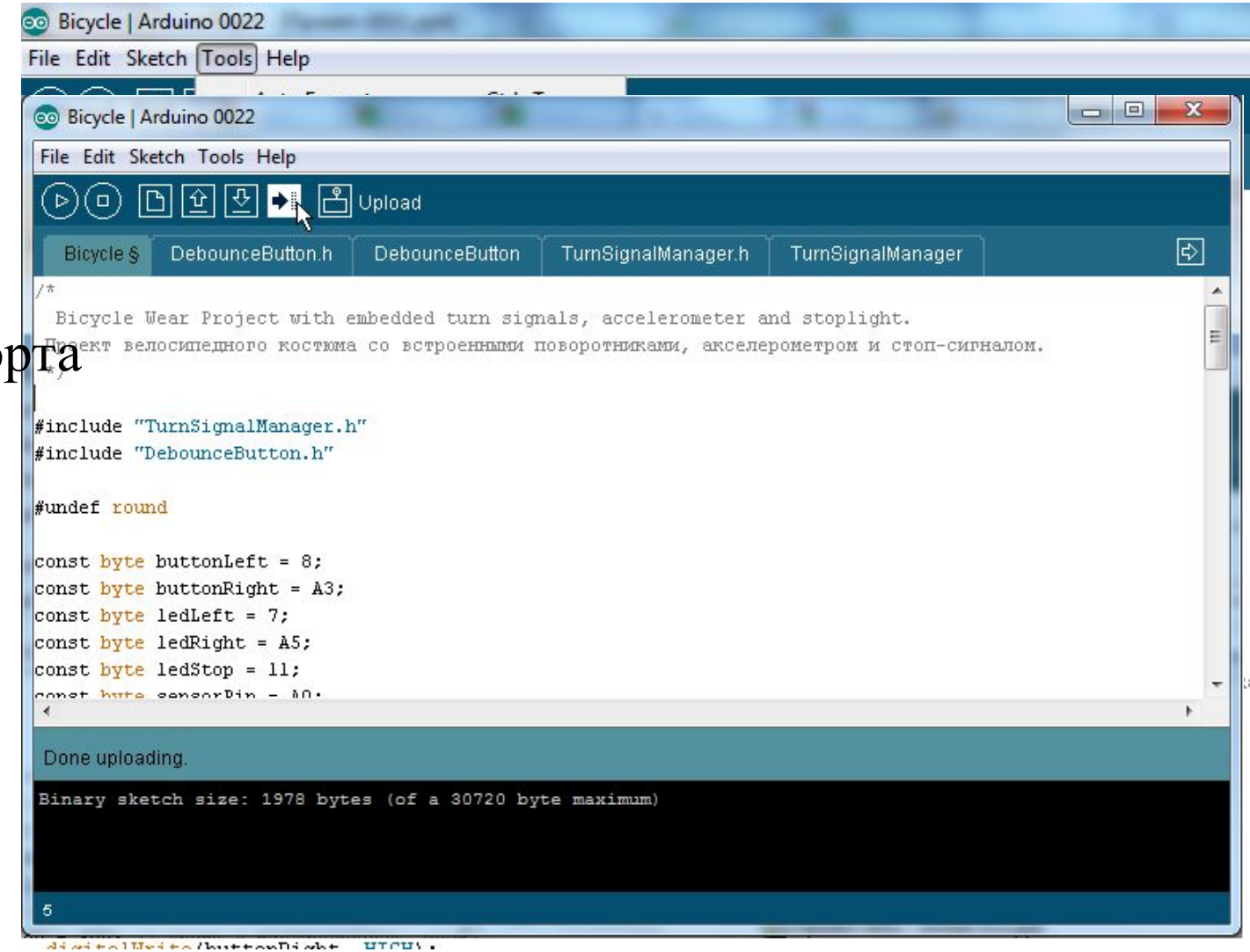
ЗЕМЛЯ

АНАЛОГОВЫЕ ВХОДЫ

ВХОД +

Среда разработки

Выбор платы
Выбор COM-порта
Прошивка



Ток, напряжение, сопротивление

- Высота жидкости подобна напряжению
- Чем больше разность уровней, тем больше энергия
- Другое название напряжения – разность потенциалов
- Чем больше разность уровней тем быстрее и сильнее поток

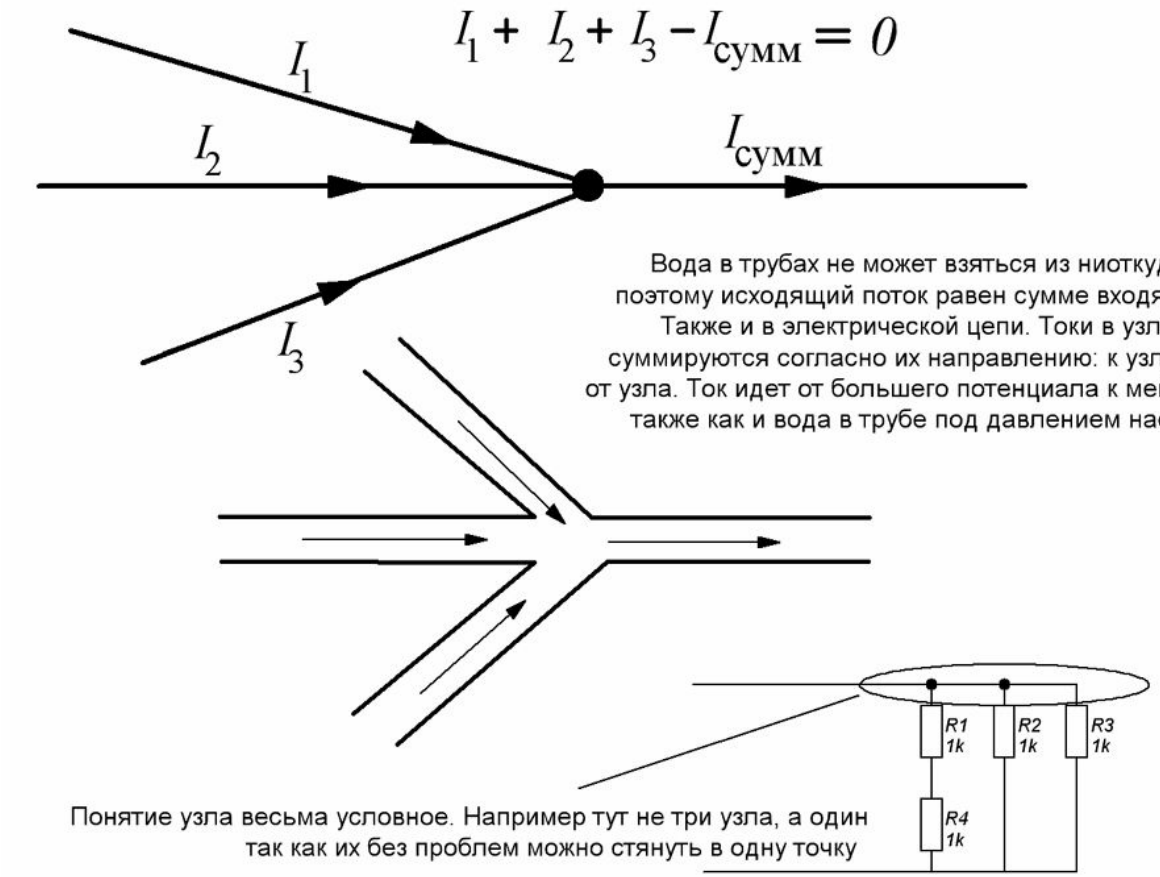
- Трение жидкости о стенки трубы, а также выполнение какой-либо работы, обеспечивают сопротивление
- Чем сильней сопротивление тем медленней поток
- При большом сопротивлении можно увеличить поток, подняв давление – разность потенциалов

Закон Ома

- Сила тока в цепи прямо пропорциональна напряжению и обратно пропорциональна полному сопротивлению цепи
- I = U / R
- U – величина напряжения в вольтах
- R – сумма всех сопротивлений в омах
- I – протекающий по цепи ток в амперах

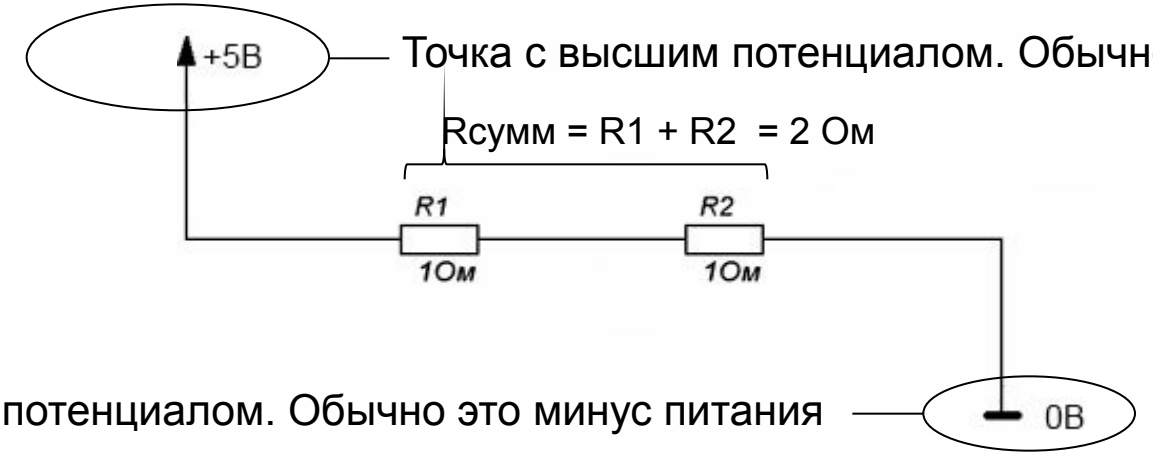
Закон Кирхгофа

$$\sum_{j=1}^{n} I_j = 0$$



Закон Ома на практике

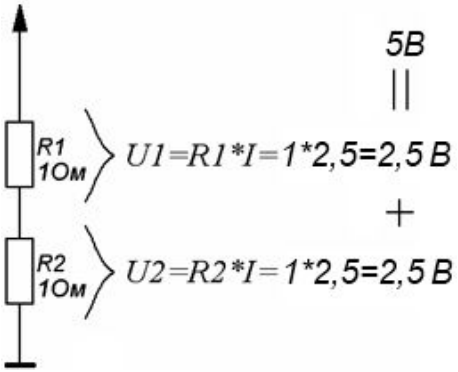
$$I = \frac{U}{R}$$

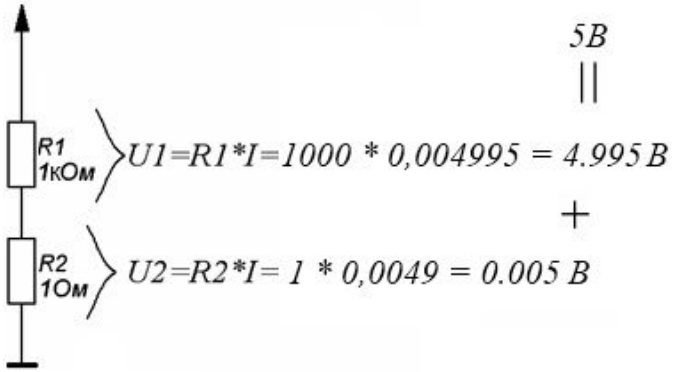


$$I = \frac{5B}{2Ом} = 2.5A$$

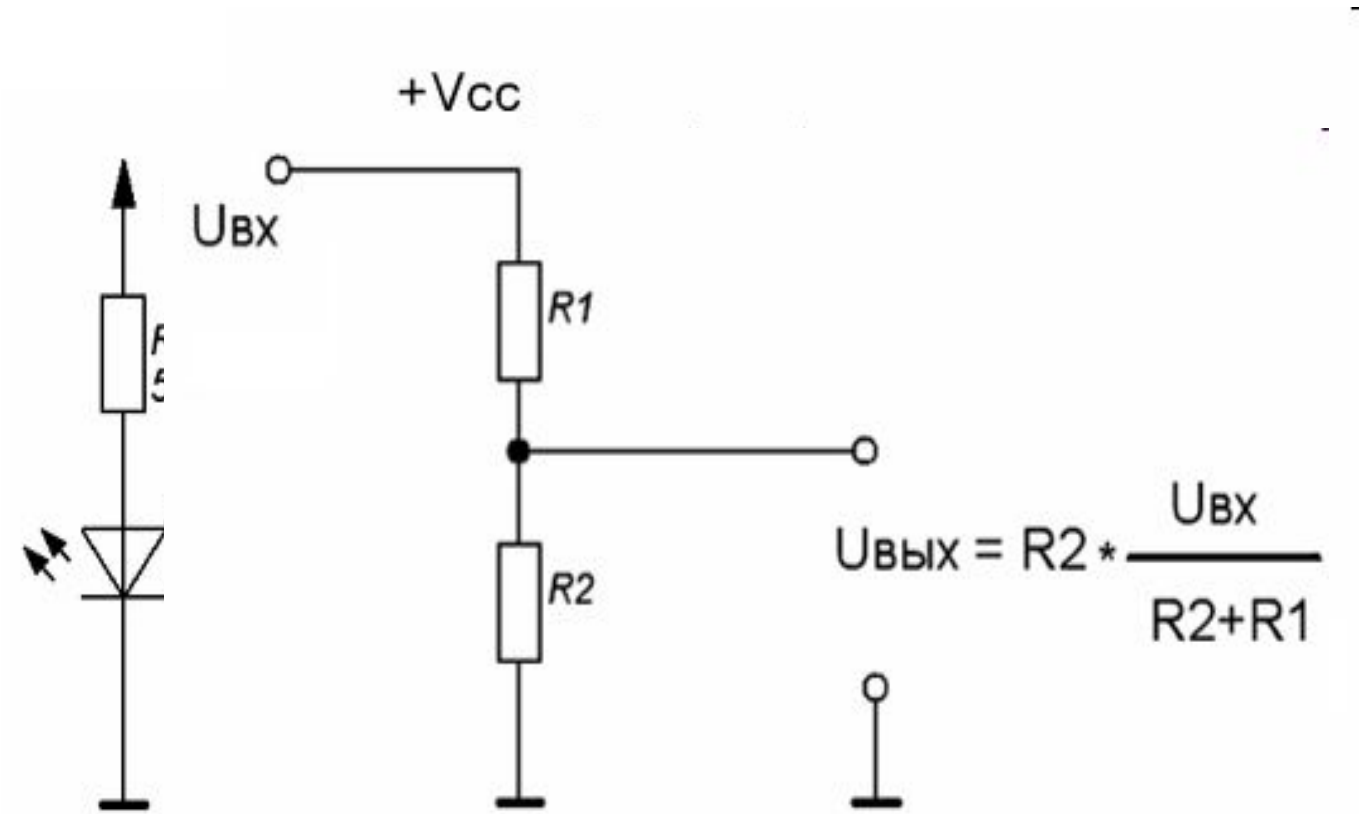
Точка с нулевым потенциалом. Обычно это минус питания

Распределение напряжения в зависимости от сопротивления:



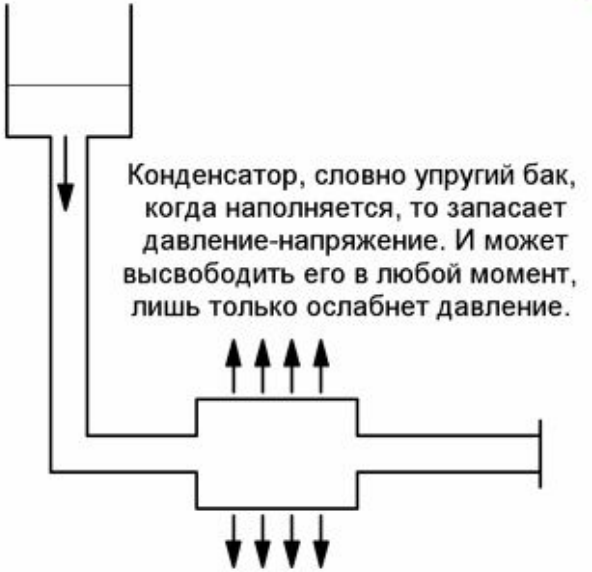


Резистор

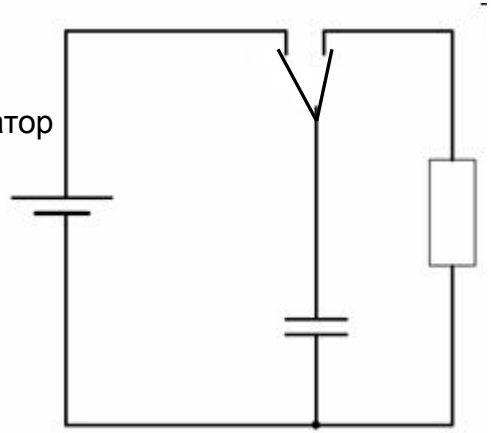


ОПростейший делитель напряжения

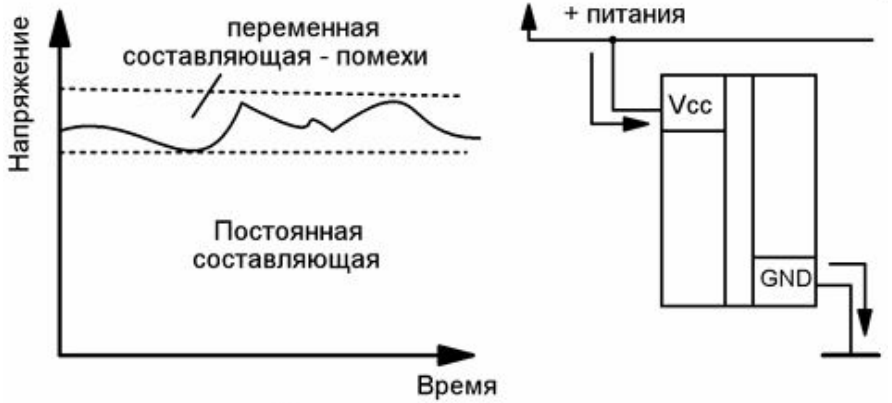
Конденсатор



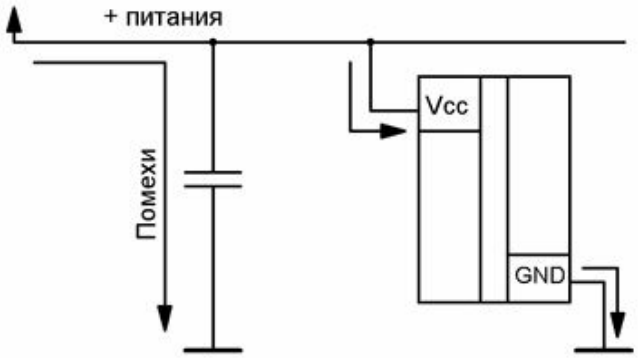
Сейчас конденсатор заряжается от источника



Но если переключить рубильник на другую цепь, то произойдет разряд конденсатора на резистор

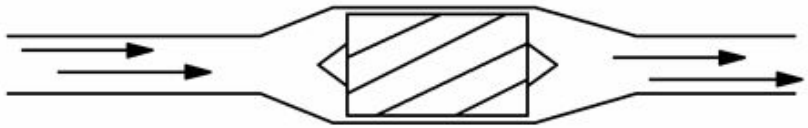


Конденсатора нет, и все напряжение вместе с помехами идет через девайс. Что явно не идет ему на пользу

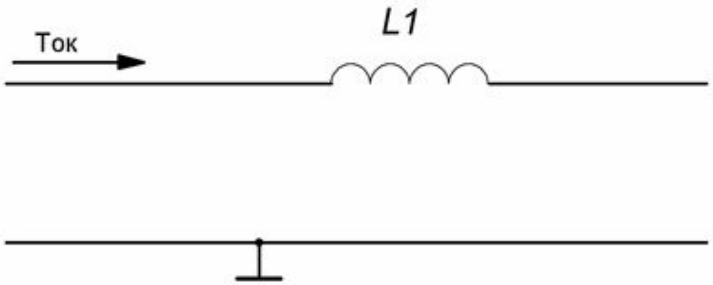


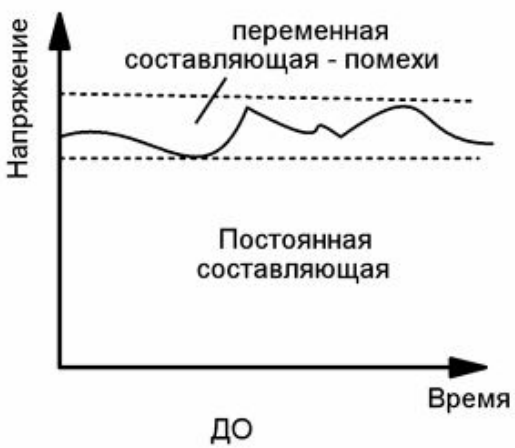
Когда кондер есть, то переменная составляющая в основном идет через него сразу на землю, т.к. сопротивление переменному току у него явно меньше, чем сопротивление девайса.
Ну, а постоянная составляющая уже идет через девайс. Так как конденсатор ее никогда не пропустит через себя

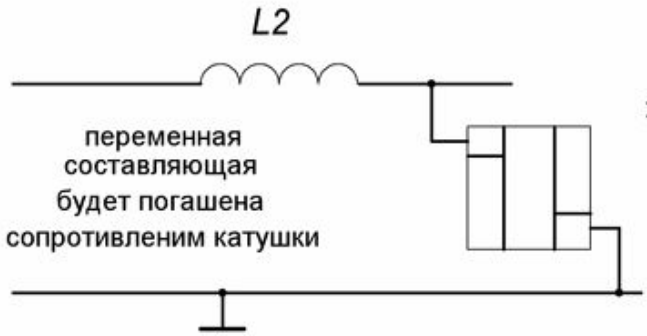
Катушка индуктивности



Катушка похожа на массивную турбину



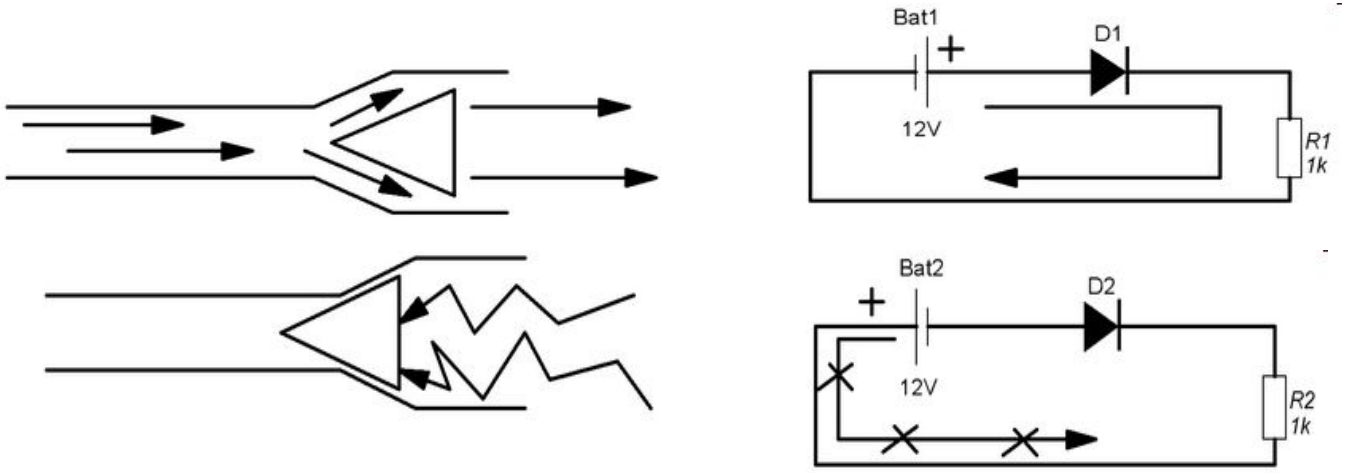




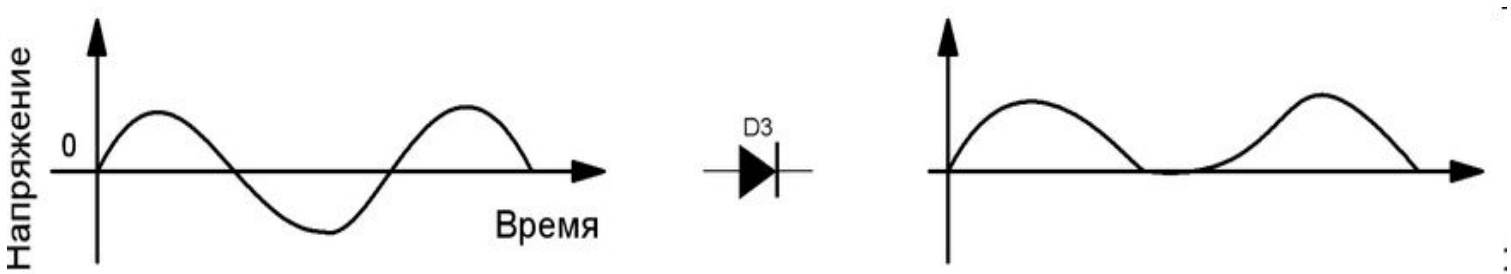
Переменная составляющая "завязнет" на индуктивном сопротивлении катушки и сильно ослабнет, а постоянная составляющая пройдет через катушку практически без потерь. В итоге, на девайсе будет почти "чистое" постоянное напряжение

http://http://лекции.техблог.рфhttp://лекции.техблог.рф/http://лекции.техблог.рф/введение_в_ардуиноhttp://лекции.техблог

Диод

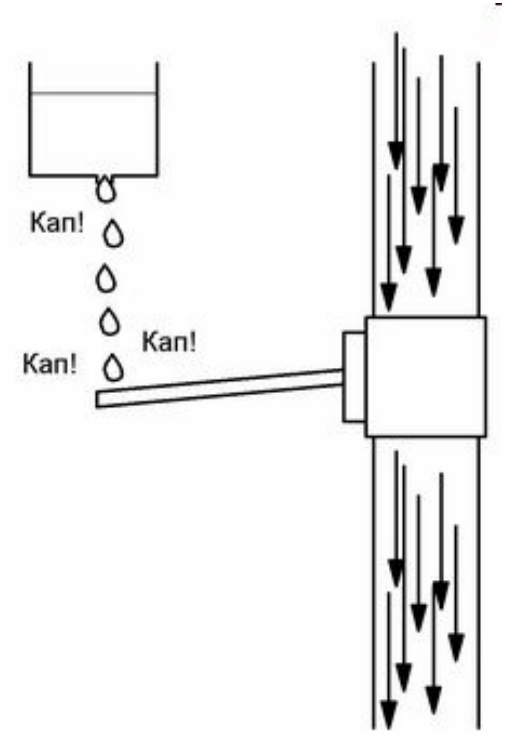


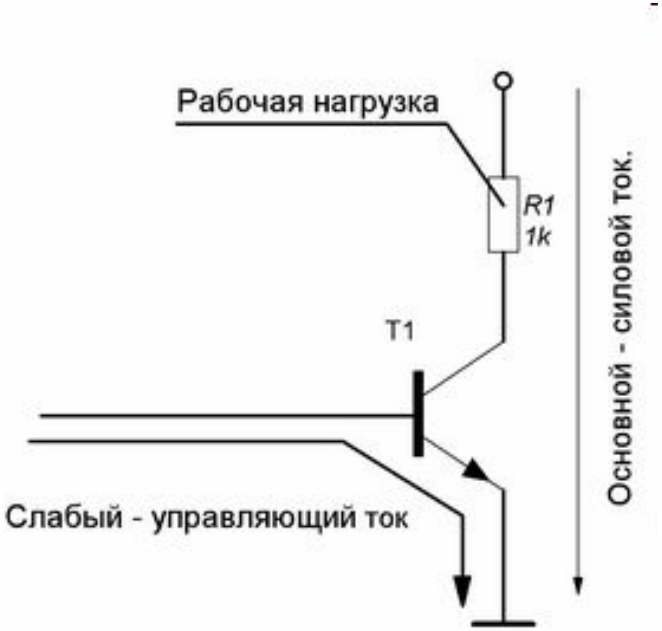
Диод - тот же ниппель, только электрический



Диод пропустил через себя только положительную полуволну переменного сигнала
Все, что было ниже нуля (т.е. шло в другом направлении) "завязло" на диоде.

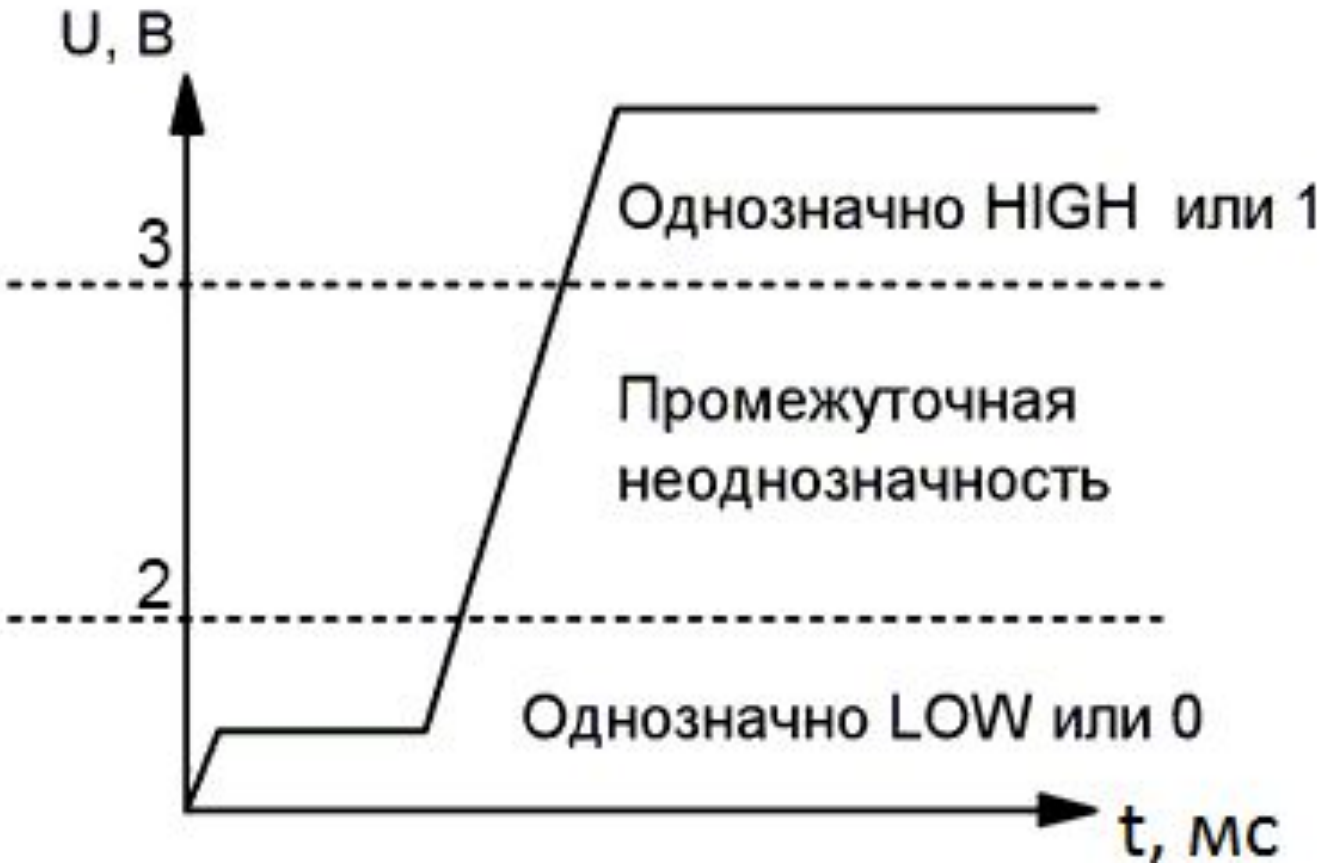
Транзистор



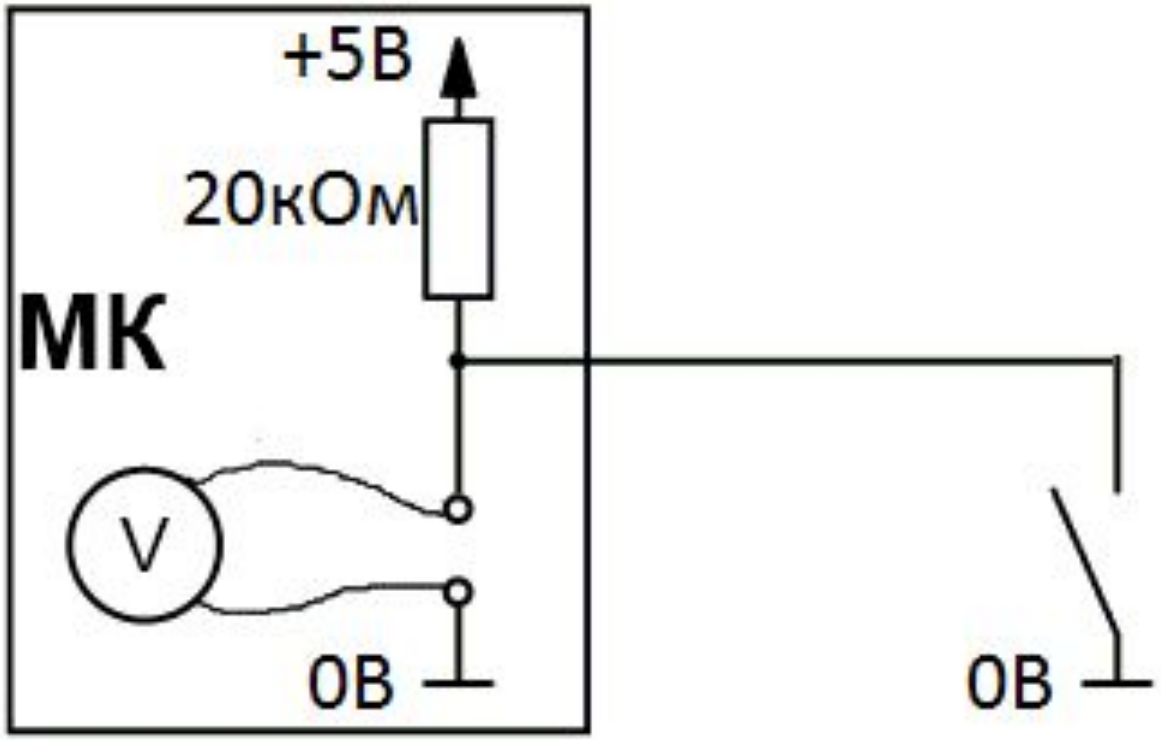


- Транзистор подобен вентилю, где крошечная сила может управлять могучим потоком энергии, в сотни раз превышающим управляющий
- Транзистор позволяет слабым сигналом, например с ноги микроконтроллера, управлять мощной нагрузкой типа двигателя или лампочки.
- Если не хватит усиления одного транзистора, то их можно соединять каскадами

Понятие нуля и единицы

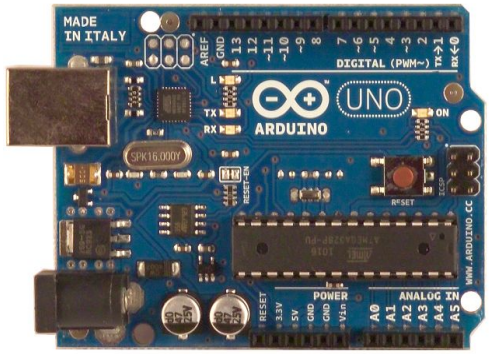


Подтяжка выводов до нужного напряжения

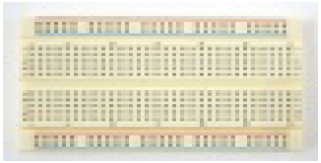


Предполетная подготовка

- Проверить наличие Arduino-совместимой платы и USB-кабеля

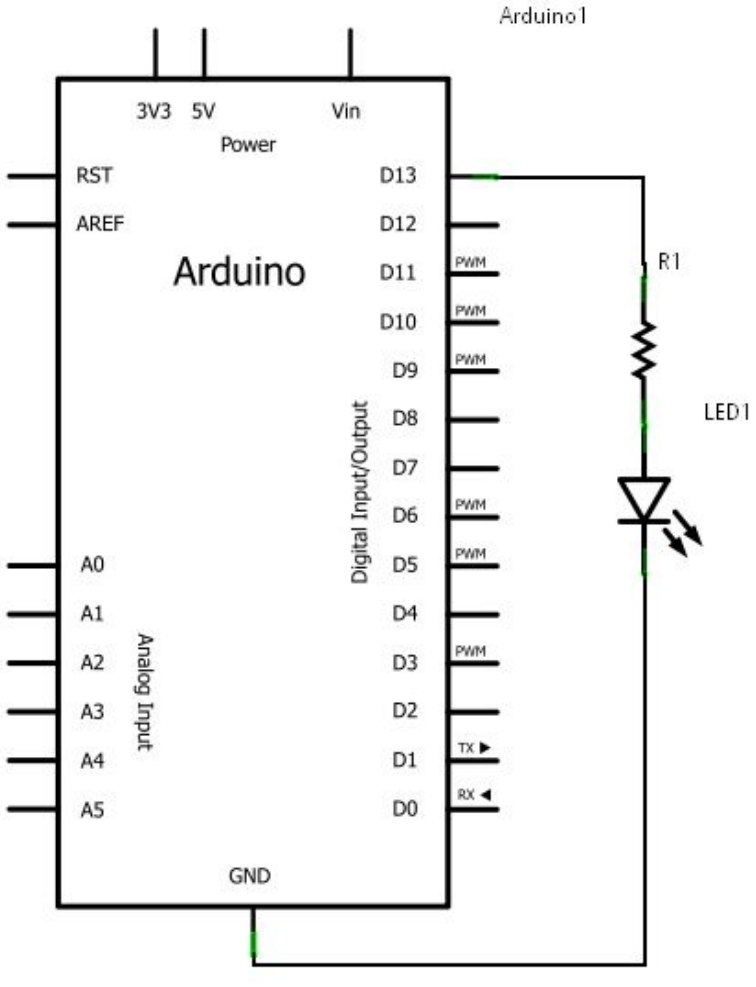


- (Опционально) Проверить наличие макетной платы, соединительных проводов, диода и резистора на 150-500 Ом



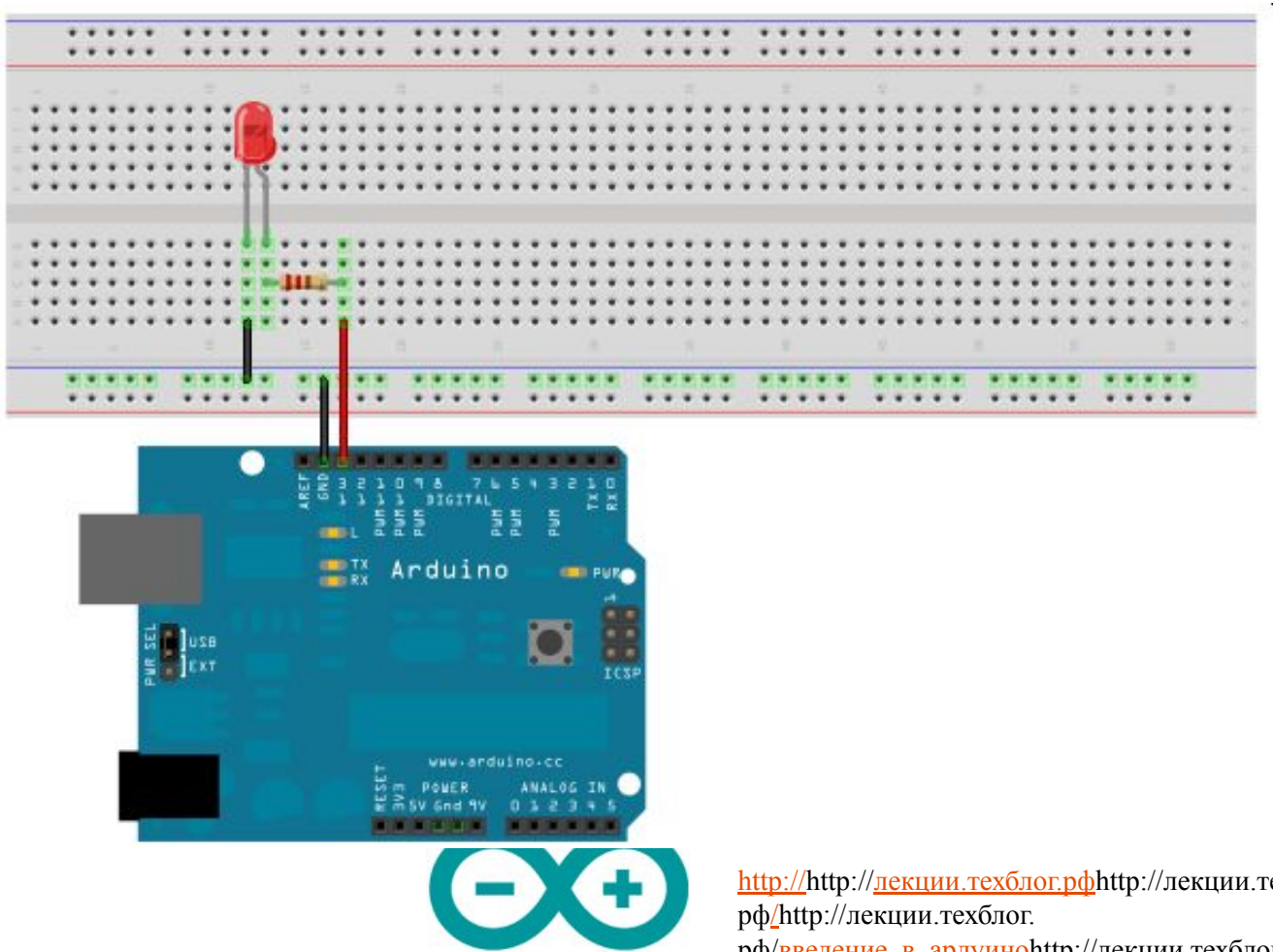
Предполетная подготовка

Принципиальная схема



Предполетная подготовка

Макетная плата



Полет

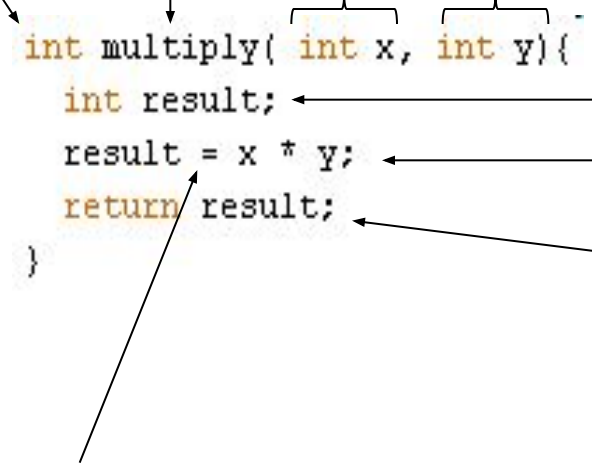
```
/*
  Blink.
  Включает светодиод на секунду, затем выключает на секунду в цикле.
*/


// Инициализация. Метод вызывается только 1 раз, когда стартует скетч, после подачи питания
// или после сброса платы. Используется для инициализации переменных, определения режимов
// работы выводов, запуска используемых библиотек
void setup() {
  pinMode(13, OUTPUT); // назначить 13-й вывод как выход
}

// Бесконечный цикл. После выполнения setup(), данный метод вызывается каждый раз после
// завершения последнего оператора в цикле
void loop() {
  digitalWrite(13, HIGH);   // включить светодиод на 13 выводе
  delay(1000);              // подождать 1 секунду = 1000 миллисекунд
  digitalWrite(13, LOW);    // выключить светодиод на 13 выводе
  delay(1000);              // подождать 1 секунду = 1000 миллисекунд
}
```

Разбор полета (1)

Тип возвращаемого значения, либо void, если ничего не возвращаем

Имя функции

Параметры функции и их тип



Объявление переменной типа int

Инициализация переменной значением

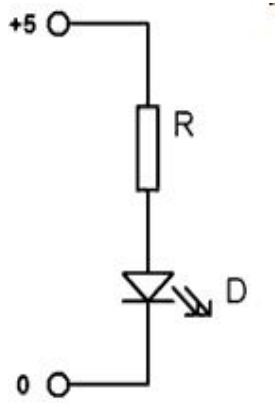
Прекращение выполнения функции и возврат значения типа int

Оператор присваивания

Разбор полета (2)

Характеристики диода:

- Тип корпуса
- Угол рассеивания, градусы
- Типовой (рабочий) ток, А
- Падение (рабочее) напряжения, В
- Цвет свечения (длина волны), нм



Пример: $U_{светодиода} = 2B, \quad I_{светодиода} = 20мА$

$$U_{резистора} = U_{питания} - U_{светодиода} = 5B - 2B = 3B$$

$$R_{резистора} = \frac{U_{резистора}}{I_{светодиода}} = \frac{3B}{0.02A} = 150\ Ом$$

Разбор полета (3)

Недостаток программы: если мы захотим поменять вывод №13 на другой, мы должны внести исправления в нескольких местах.

Решение: введем глобальную переменную, хранящую номер вывода

```
/*
  Blink2.
  Включает светодиод на секунду, затем выключает на секунду в цикле.
*/

int ledPin =  13; // Глобальная переменная. Используется, чтобы хранить номер вывода

// Инициализация.
void setup() {
  pinMode(ledPin, OUTPUT); // назначить ногу ledPin как выход
}

// Бесконечный цикл.
void loop() {
  digitalWrite(ledPin, HIGH);   // включить светодиод на ноге ledPin
  delay(1000);               // подождать 1 секунду = 1000 миллисекунд
  digitalWrite(ledPin, LOW);    // выключить светодиод на ноге ledPin
  delay(1000);               // подождать 1 секунду = 1000 миллисекунд
}
```

Разбор полета (4)

Недостаток программы: слишком много дублированного кода внутри цикла

Решение: введем глобальную переменную, хранящую текущее значение напряжения

```
/*
  Blink3.
  Включает светодиод на секунду, затем выключает на секунду в цикле.
*/

int ledPin =  13;   // Используется, чтобы хранить номер вывода
boolean ledState = HIGH; // Используется, чтобы хранить текущее значение светодиода ВКЛ/ВЫКЛ

// Инициализация.
void setup() {
  pinMode(ledPin, OUTPUT); // назначить ногу ledPin как выход
}

// Бесконечный цикл.
void loop() {
  ledState = !ledState;  // присвоить переменной ledState противоположное значение (воскл. знак означает отрицание)
  digitalWrite(ledPin, ledState);   // задать на ноге ledPin значение ledState
  delay(1000);  // подождать 1 секунду = 1000 миллисекунд
}
```

Разбор полета (5)

Недостаток программы: delay(1000) означает, что процессор МК простаивает 1 секунду и мы не можем обрабатывать датчики/делать вычисления. Фактически, мы замедлили его работу до частоты 1 Гц вместо 16 МГц. Если сработает датчик, то мы сможем отследить его через 1 сек вместо 1/16000000 сек., либо вообще не сможем отследить событие

```
/*
  Blink4.
  Включает светодиод на секунду, затем выключает на секунду в цикле, не используя функцию Delay
*/

int ledPin =  13;   // Используется, чтобы хранить номер вывода
boolean ledState = HIGH; // Используется, чтобы хранить текущее значение состояния светодиода ВКЛ/ВЫКЛ
long previousTimeStamp = 0; // Будет хранить время последнего изменения состояния светодиода

// Инициализация.
void setup() {
  pinMode(ledPin, OUTPUT); // назначить ногу ledPin как выход
}

// Бесконечный цикл.
void loop() {
  // Объявляем локальную переменную currentTimeStamp, и инициализируем ее значением функции millis()
  // Функция millis() возвращает количество миллисекунд с момента начала выполнения текущей программы на плате Arduino
  long currentTimeStamp = millis();

  // проверяем, а не изменить ли нам напряжение на светодиоде? Меняем, только если текущее время в мс отличается
  // от времени последнего изменения больше чем на 1000 мс
  if( currentTimeStamp - previousTimeStamp > 1000 )
  {
    previousTimeStamp = currentTimeStamp; // запоминаем текущее время как время последнего изменения
    ledState = !ledState; // присвоить переменной ledState противоположное значение (воскл. знак означает отрицание)
    digitalWrite(ledPin, ledState); // подождать 1 секунду = 1000 миллисекунд
  }
}
```

Типы данных (ознакомится)

- Логический (булевый) тип данных — **boolean.** Может принимать одно из двух значений true или false. boolean занимает в памяти один байт
- **Char** (символ)Переменная типа **char** занимает 1 байт памяти и может хранить один алфавитно-цифровой символ (литеру). При объявление литеры используются одиночные кавычки: 'A' (двойные кавычки используется при объявлении строки символов - тип string: "ABC").
- **Byte -** тип данных byte 8-ми битное беззнаковое целое число, в диапазоне 0..255.
- **Int** (целое)о дин их наиболее часто используемых типов данных для хранения чисел. int занимает 2 байта памяти, и может хранить числа от -32 768 до 32 767
- **unsigned int** - (беззнаковое целое) число, также как и тип **int** (знаковое) занимает в памяти 2 байта. Но в отличие от **int**, тип **unsigned int** может хранить только положительные целые числа в диапазоне от 0 до 65535 (2^16)-1)

http://http://лекции.техблог.рфhttp://лекции.техблог.рф/http://лекции.техблог.рф/введение_в_ардуиноhttp://лекции.техблог

- **long** (длинное) используется для хранения целых чисел в расширенном диапазоне от -2,147,483,648 до 2,147,483,647.**long** занимает 4 байта в памяти
- **Unsigned long** (без знака длинное) используется для хранения положительных целых чисел в диапазоне от 0 до 4,294,967,295 (2^32 - 1) изанимает 32 бита (4 байта) в памяти**.**
- **float** (плавающий) служит для хранения чисел с плавающей запятой. Этот тип часто используется для операций с данными, считываемыми с аналоговых входов. Диапазон значений — от -3.4028235E+38 до 3.4028235E+38. Переменная типа **float**занимает 32 бита (4 байта) в памяти
- **Double** (двойной), в отличие от большинства языков программирования, имеет ту же точность, что и тип **float** и занимает также 4 байта памяти

- Базовая структура программы для Arduino довольно проста и состоит, по меньшей мере, из двух частей. В этих двух обязательных частях, или функциях, заключён выполняемый код

```
void setup()
{
  statements;
}

void loop()
{
  statements;
}
```

Где setup() — это подготовка, а loop() — выполнение. Обе функции требуются для работы программы.

Перед функцией setup - в самом начале программы, обычно, идёт, объявление всех переменных. setup - это первая функция, выполняемая программой, и выполняемая только один раз, поэтому она используется для установки режима работы портов (pinMode()) или инициализации последовательного соединения

```
void setup()
{
  pinMode(pin, OUTPUT);      // устанавливает 'pin' как выход
}
```

Следующая функция loop содержит код, который выполняется постоянно — читаются входы, переключаются выходы и т.д. Эта функция — ядро всех программ Arduino и выполняет основную работу.

```
void loop()
{
  digitalWrite(pin, HIGH);   // включает 'pin'
  delay(1000);               // секундная пауза
  digitalWrite(pin, LOW);    // выключает 'pin'
  delay(1000);               // секундная пауза
}
```

- Функция — это блок кода, имеющего имя, которое указывает на исполняемый код, который выполняется при вызове функции. Функции void setup() и void loop() уже обсуждались

Могут быть написаны различные пользовательские функции, для выполнения повторяющихся задач и уменьшения беспорядка в программе. При создании функции, первым делом, указывается тип функции. Это тип значения, возвращаемого функцией, такой как 'int' для целого (integer) типа функции. Если функция не возвращает значения, её тип должен быть void. За типом функции следует её имя, а в скобках параметры, передаваемые в функцию.

```
int delayVal()
{
  int v;                    // создаём временную переменную 'v'
  v  = analogRead(pot);     // считываем значение с потенциометра
  v /= 4;                   // конвертируем 0 - 1023 в 0 - 255
  return v;                 // возвращаем конечное значение
}
```

{} фигурные скобки

Фигурные скобки (также упоминаются как просто «скобки») определяют начало и конец блока функции или блока выражений, таких как функция void loop() или выражений (statements) типа for и if.

За открывающейся фигурной скобкой { всегда должна следовать закрывающаяся фигурная скобка }. Об этом часто упоминают, как о том, что скобки должны быть «сбалансированы». Несбалансированные скобки могут приводить к критическим, неясным ошибкам компиляции, вдобавок иногда и трудно выявляемым в больших программах.

переменные

- Переменные — это способ именовать и хранить числовые значения для последующего использования программой. Само название - переменные, говорит о том, что переменные - это числа, которые могут последовательно меняться, в отличие от констант, чьё значение никогда не меняется. Переменные нужно декларировать (объявлять), и, что очень важно - им можно присваивать значения, которые нужно сохранить. Следующий код объявляет переменную inputVariable, а затем присваивает ей значение, полученное от 2-го аналогового порта:

```
int inputVariable = 0;              // объявляется переменная и
                                    // ей присваивается значение 0
inputVariable = analogRead(2); // переменная получает значение
                                    // аналогового вывода 2
```

- Переменные могут быть объявлены в начале программы перед void setup(), локально внутри функций, и иногда в блоке выражений таком, как цикл for. То, где объявлена переменная, определяет её границы (область видимости), или возможность некоторых частей программы её использовать.

- Глобальные переменные таковы, что их могут видеть и использовать любые функции и выражения программы. Такие переменные декларируются в начале программы перед функцией setup(). Локальные переменные определяются внутри функций или таких частей, как цикл for. Они видимы и могут использоваться только внутри функции, в которой объявлены.

```
int value;                          // 'value' видима
                                    // для любой функции
void setup()
{
  // no setup needed
}

void loop()
{
  for (int i=0; i<20;)              // 'i' видима только
  {                                 // внутри цикла for
    i++;
  }
  float f;                          // 'f' видима только
}                                   // внутри loop
```

массивы

```
int myArray[5];     // объявляет массив целых длиной в 6 позиций
myArray[3] = 10;    // присваивает по 4у индексу значение 10
```

Чтобы извлечь значение из массива, присвоим переменной значение по индексу массива:

```
x = myArray[3];     // x теперь равно 10
```

арифметика

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

http://http://лекции.техблог.рфhttp://лекции.техблог.рф/http://лекции.техблог.рф/введение_в_ардуиноhttp://лекции.техблог.

операторы сравнения

```
x == y      // x равно y
x != y      // x не равно y
x <  y      // x меньше, чем y
x >  y      // x больше, чем y
x <= y      // x меньше, чем или равно y
x >= y      // x больше, чем или равно y
```

логические операторы

Logical AND:

```
if (x > 0 && x < 5)        // true, только если оба
                           // выражения true
```

Logical OR:

```
if (x > 0 || y > 0)        // true, если любое из
                           // выражений true
```

Logical NOT:

```
if (!x > 0)                // true, если только
                           // выражение false
```

константы

- true/false
- high/low Эти константы определяют уровень выводов как HIGH или LOW и используются при чтении или записи на логические выводы. HIGH определяется как логический уровень 1, ON или 5 вольт(3-5), тогда как LOW — 0, OFF или 0 вольт(0-2)