Автономное учреждение дополнительного образования
Ханты-Мансийского автономного округа – Югры
«Мастерская талантов «Сибириус»

Методическая разработка конспекта-занятия «Основы программирования Arduino на примере создания прототипа брелока, сигнализирующего о своем местоположении»

Разработал:
Цигенгагель Никита Романович
Педагог дополнительного образования
Отдела развития детского технопарка
«Кванториум», г. Радужный
АУ ДО «Мастерская талантов «Сибириус»

г. Радужный, 2025 г.

Методическая разработка конспекта-занятия «Основы программирования Arduino на примере создания прототипа брелока, сигнализирующего о своем местоположении»

Аннотация

В настоящее время электроника стала чем-то совершенно обыденным для человека. В жизни людей постоянно и повсеместно используются такие достижения техники, как компьютеры, смартфоны, умные микроволновые печи, стиральные машины и т.д. Подобные устройства необходимо не только спроектировать и собрать, но и написать для них программное обеспечение, т.к. внутри таких изделий практически всегда есть «мини-компьютер», который управляет всеми датчиками, лампочками, моторами, экранами и т.д.

В рамках данного занятия мы попробуем создать своё электронное устройство в виде прототипа брелока, который подаст пользователю звуковой и световой сигнал, если тот его потеряет. Заодно с этим мы познакомимся с платформой Tinkercad, научимся собирать схемы на основе Arduino Uno и познакомимся с основами программирования в процессе написания прошивки для нее на языке C++.

Длительность занятия:

2,5 академических часа.

План занятия:

<table>
<tr><td colspan="2">Этап 1. Основы программирования. Сборка прототипа устройства.</td></tr>
<tr><td>Продолжительность</td><td>Цель этапа</td></tr>
<tr><td>75 мин</td><td>Знакомство с инструментом «цепи» платформы Tinkercad. Сборка схем на основе Arduino Uno. Знакомство с основными понятиями программирования: переменные, функции, операторы ветвления, циклы.</td></tr>
<tr><td colspan="2">Содержание:
Создание прототипа «умного» брелока, сигнализирующего о своем местоположении в случае потери. Подключение светодиодов к плате. Работа с зуммером. Изменение режимов работы устройства по нажатию кнопки.</td></tr>
<tr><td colspan="2">Этап 2. Выполнение самостоятельного задания 1.</td></tr>
<tr><td>Продолжительность</td><td>Цель этапа</td></tr>
<tr><td>20 мин</td><td>Закрепить навыки обучающихся по сборке электронных схем в Tinkercad. Отточить навыки работы со стандартными функциями.</td></tr>
<tr><td colspan="2">Содержание:
Модернизировать устройство, добавив в схему второй светодиод. Мигание светодиодов в режиме сигнализации должно быть попеременным. При выходе из режима сигнализации оба светодиода должны гореть непрерывно.</td></tr>
<tr><td colspan="2">Этап 3. Выполнение самостоятельного задания 2.</td></tr>
<tr><td>Продолжительность</td><td>Цель этапа</td></tr>
<tr><td>20 мин</td><td>Закрепить навыки обучающихся по сборке электронных схем в Tinkercad. Отточить навыки работы со стандартными функциями и циклом while.</td></tr>
<tr><td colspan="2">Содержание:
Модернизировать устройство, добавив в схему вторую кнопку. При нажатии кнопки 1, устройство должно выйти из режима сигнализации, а при нажатии на кнопку 2 снова войти в него.</td></tr>
</table>

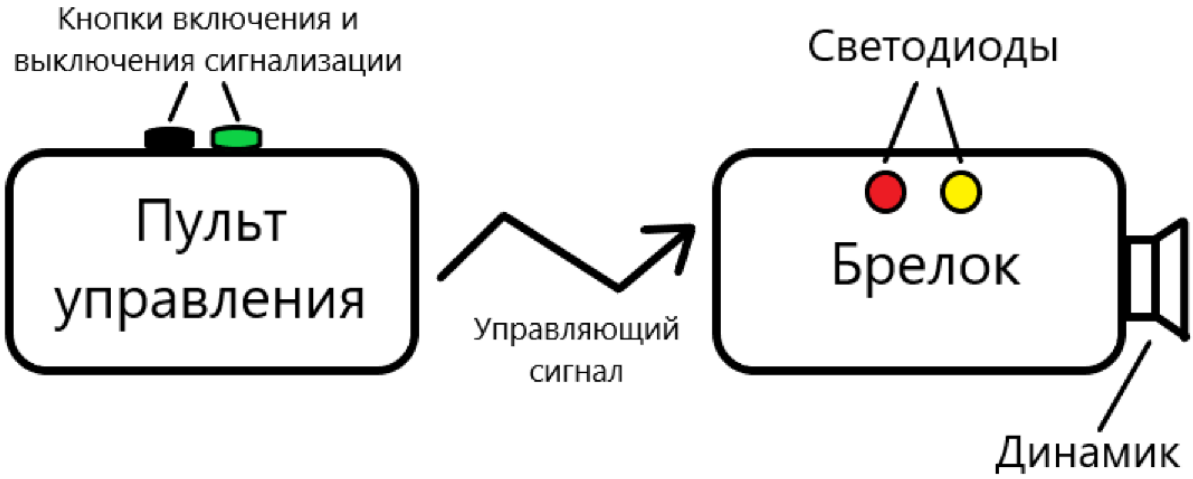
Легенда

В жизни мы часто кладём куда-то свои вещи, а потом забываем, где они находятся, и, порой, долго не можем их найти. Всё бы ничего, если эта вещь не слишком важная – найдется как-нибудь сама, – но что, если это ключи от квартиры или кошелек с деньгами? В такие моменты жалеешь, что не можешь «позвонить» на потерявшийся предмет.

В рамках этого задания мы создадим модель прототипа такого брелока, который можно прикрепить к какому-либо предмету и, в случае необходимости, подать на него сигнал, который заставит его запищать и моргать лампочками.

Содержание занятия

Итак, начнем создание нашего прототипа. В идеале концептуальная схема такого устройства должна быть следующей:



В состав системы должен входить пульт управления, который должен посылать на брелок управляющие сигналы по нажатию кнопки. Если какой-либо предмет, который прикреплен к брелоку, потерялся, пользователь нажмет на кнопку включения сигнализации, после чего брелок замигает светодиодами и начнет издавать звуковой сигнал. Когда потерянная вещь будет найдена, можно будет отключить сигнализацию по нажатию специальной кнопки.

Пока что у нас недостаточно знаний, чтобы создать сразу готовое устройство (а точнее, два), поэтому мы лишь создадим модель прототипа такого брелока. Здесь все будет намного проще – нужно создать такое устройство, которое будет по нажатию одной кнопки входить в режим сигнализации, моргая светодиодами и издавая писк, а по нажатию другой кнопки – выключать светодиоды и пищалку, и ждать повторного включения режима сигнализации.

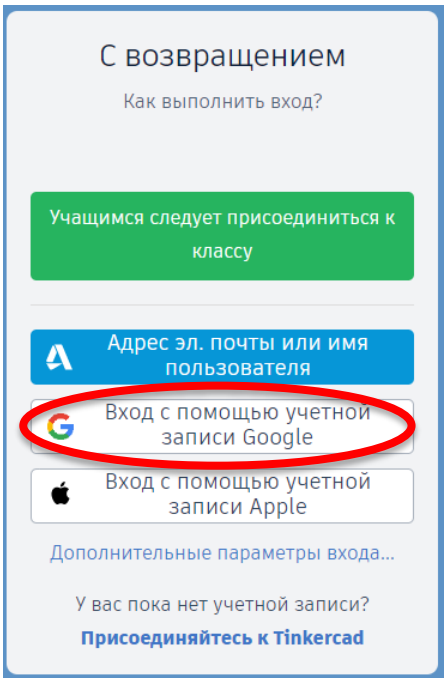
Инструмент «цепи» в Tinkercad

Так как мы создаем только модель прототипа, без физического исполнения, будем работать на платформе Tinkercad, которая, кроме всего прочего, позволяет симулировать поведение плат Arduino.

Для начала заходим на сайт https://www.tinkercad.com и нажимаем кнопку «Войти» в правом верхнем углу:

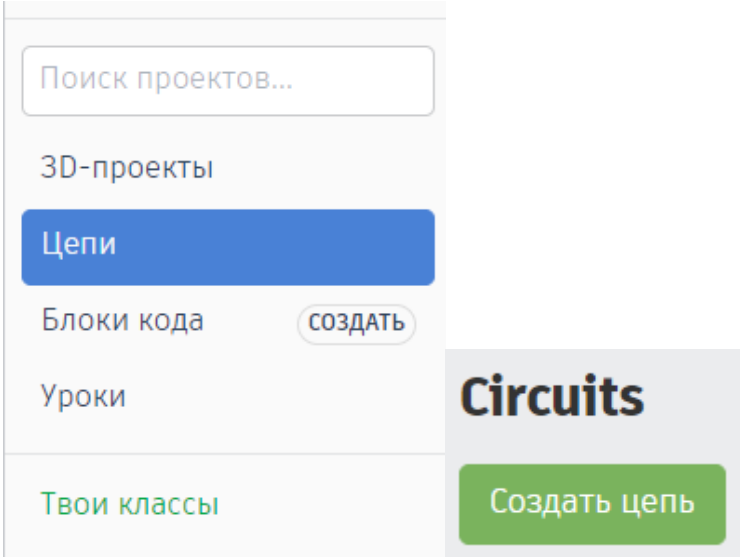


Если у вас зарегистрирован Google-аккаунт, нажмите по кнопке «Вход с помощью учетной записи Google».

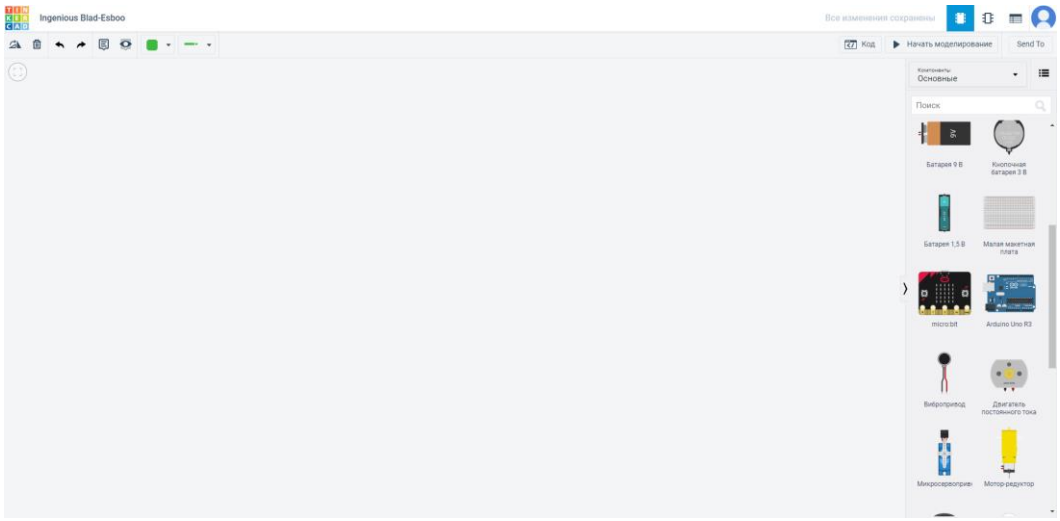


Если у вас нет Google-аккаунта, зарегистрируйте его. Так будет проще всего входить на платформу Tinkercad.

После того, как вы вошли в учетную запись, переходите во вкладку «Цепи» и нажмите на кнопку «Создать цепь»:



После этого у вас появится окно проекта:

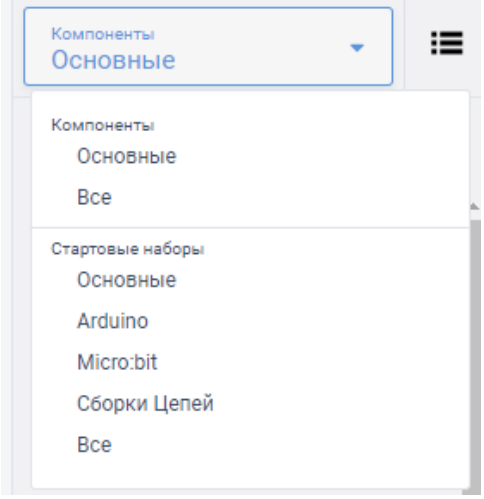


Создание модели прототипа

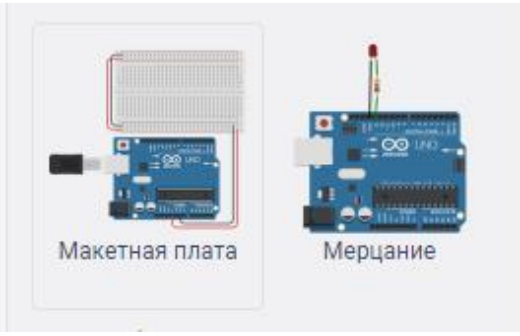
Ну что ж, приступим к разработке нашего брелока. С основными моментами работы Tinkercad познакомимся по ходу дела.

Для начала нам понадобится разместить в рабочей области «мозг» нашего устройства, плату контроллера Arduino Uno, которая будет управлять светодиодами и пищалками, которые мы к ней подключим.

В правой верхней части находим меню выбора компонентов и кликаем по пункту Arduino:

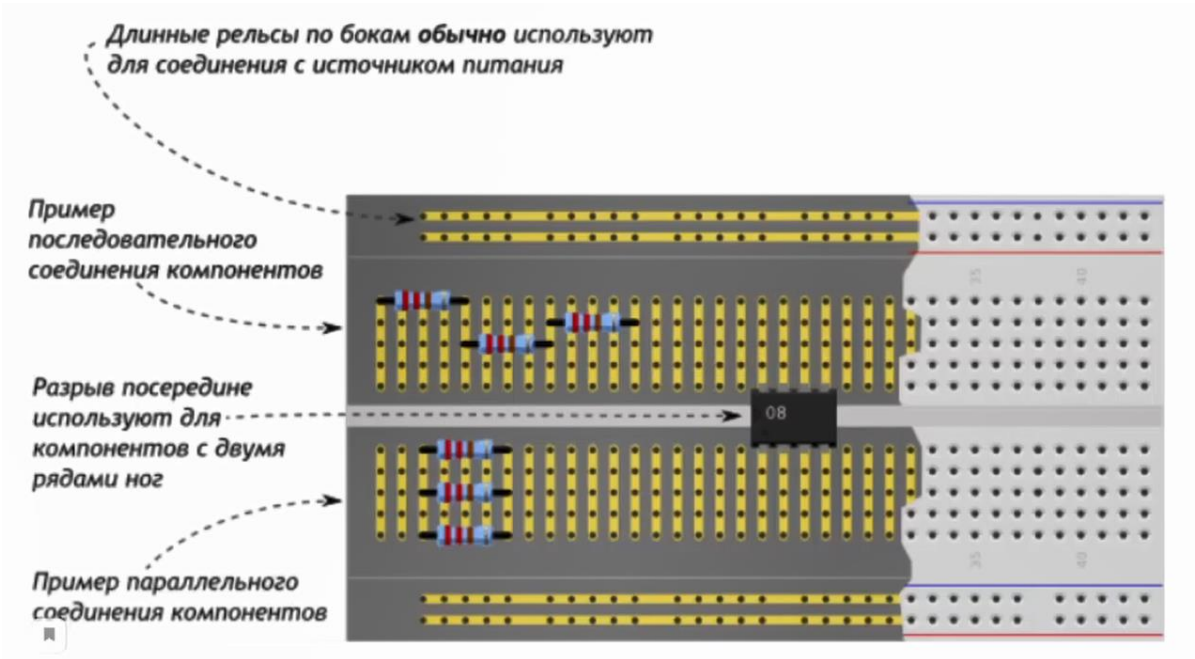


В панели справа появится множество вариантов выбора плат. Выберем самый первый шаблон «макетная плата» с макетной платой и перетащим его в рабочую область:

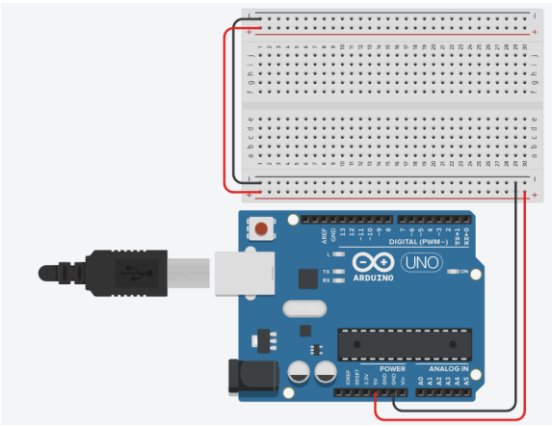


Макетная плата призвана упростить сборку электронных схем. Без нее нам бы пришлось все необходимые компоненты объединять с помощью проводов, что было бы очень неудобно и непрактично.

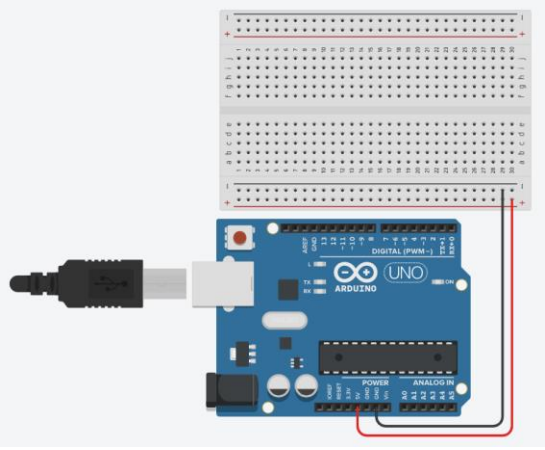
Устройство макетной платы поясняется на следующем рисунке:



На данный момент рабочая область выглядит так:



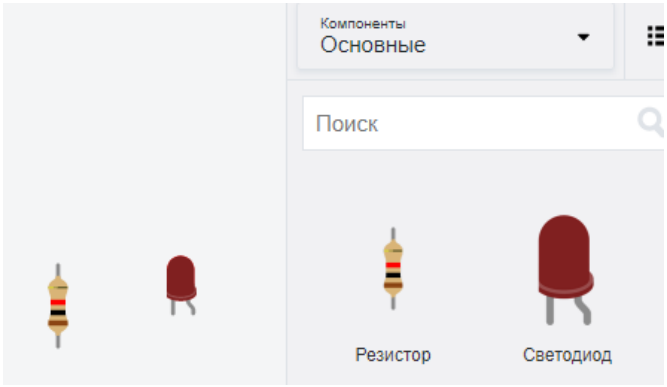
К двум нижним горизонтальным линиям макетной платы подключены порты заземления GND и питания 5V. Это для нас удобно, а вот два провода слева нам ни к чему. Удалим их, после чего схема будет выглядеть так:



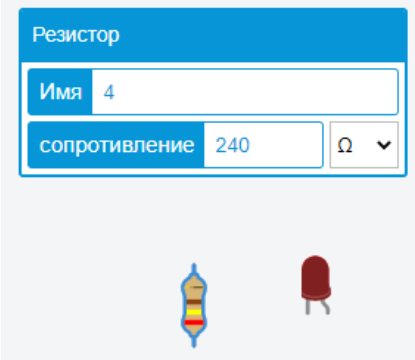
Подготовка завершена, приступим к сборке схемы устройства.

Для начала нам понадобится лампочка, которая будет мигать в режиме сигнализации. Она называется светодиод. В меню выбора компонентов кликнем «Основные» и найдем иконку «Светодиод». Перетащим ее в рабочую область.

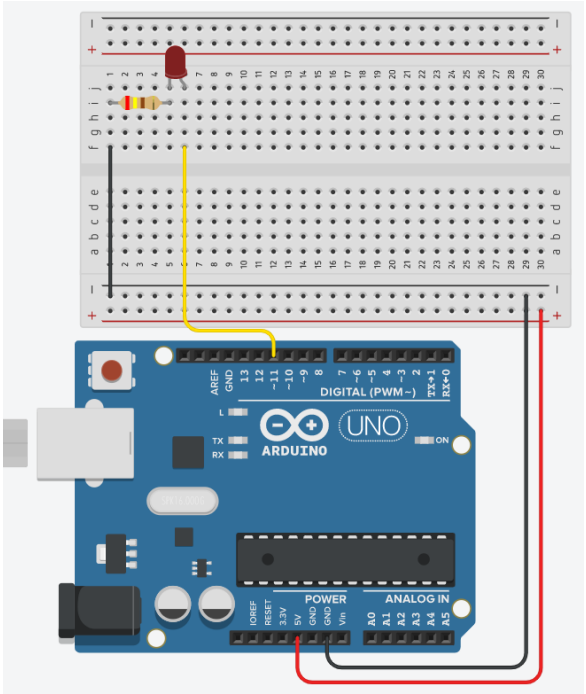
Кроме того, нам понадобится резистор, который будет ограничивать ток на светодиоде, иначе он сгорит. Находим его рядом со светодиодом и тоже перетаскиваем в рабочую область.



Теперь кликнем по резистору и зададим ему номинал в 240 Ом:

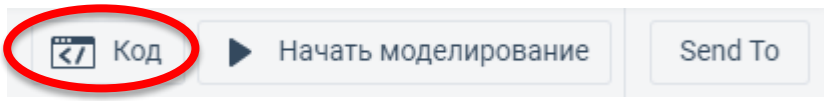


Теперь подключим светодиод к плате Arduino Uno:

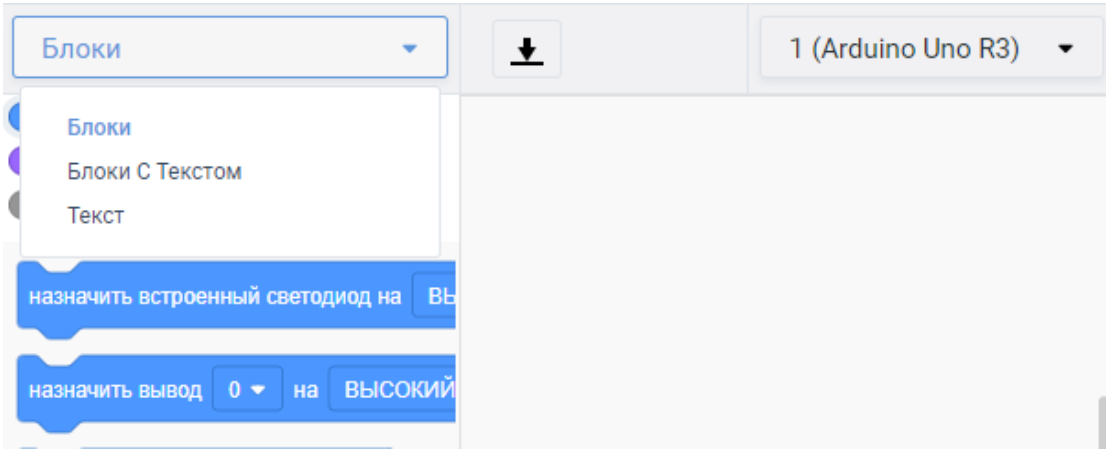


Светодиод, как и обычный диод, может пропускать ток только в одном направлении, а именно, со стороны изогнутой ножки, поэтому ее мы подключили к порту 11, который будет управлять включением и выключением светодиода. Другую ножку мы подключили через резистор к порту GND, чтобы светодиод не сгорел от слишком высокого напряжения и тока.

Теперь приступим к написанию кода. Кликнем по кнопке «Код» в правой верхней части окна:



По умолчанию откроется режим визуального программирования, но нам нужно что-то посерьезнее. Кликнем по галочке рядом с надписью «Блоки» и выберем режим «Текст».



Теперь в правой части появился шаблон программы:

```
1  // C++ code
2  //
3  void setup()
4  {
5    pinMode(LED_BUILTIN, OUTPUT);
6  }
7
8  void loop()
9  {
10   digitalWrite(LED_BUILTIN, HIGH);
11   delay(1000); // Wait for 1000 millisecond(s)
12   digitalWrite(LED_BUILTIN, LOW);
13   delay(1000); // Wait for 1000 millisecond(s)
14 }
```

Здесь расположен код программы на языке C++. Давайте попробуем разобраться в нем.

Строки, начинающиеся с двух косых черт «//» называются комментариями. Они играют роль примечаний для программиста и призваны пояснять назначение того или иного участка кода. При выполнении программы эти строки будут игнорироваться.

Далее мы видим функцию «setup». Функция, это некая подпрограмма, отрывок кода, за которым закреплено определенное имя. Программа в нужный

момент может обратиться к функции по имени, в результате чего будет выполнен участок кода, закрепленный за ней.

Функция «setup» нужна для того, чтобы при включении платы единожды выполнить все необходимые настройки.

Далее расположена функция «loop», в переводе «петля» или «цикл». Код внутри нее будет выполняться снова и снова до тех пор, пока плата не будет выключена. В ней размещается основной рабочий код.

Зажигаем светодиод

Напишем программу, которая будет зажигать и гасить светодиод раз в секунду. Для этого удалим лишние комментарии и все, что содержится внутри блоков (обозначаются фигурными скобками {}) функций «setup» и «loop»:

```
1
2
3  void setup()
4  {
5
6  }
7
8  void loop()
9  {
10
11 }
```

Сперва в начале программы создадим константы, хранящие информацию о том, какие порты использует наша плата:

```
1  #define LED_RED 11
```

В данном примере мы создали константу с именем LED_RED и присвоили ей значение «11». Это означает, что далее в программе мы сможем использовать имя LED_RED, тем самым указав, что хотим обратиться к порту 11 нашей платы.

Константы – это данные, которые не могут меняться. Константы удобно использовать вместо простого указания номера порта, так как, если мы на каком-то этапе захотим подключить светодиод к порту с другим номером, нам не понадобится изменять десятки строк кода, а всего лишь изменить значение константы в начале программы.

Далее необходимо произвести действия по настройке платы внутри функции «setup». Действие всего одно – установить 11-й порт в режим выхода, что позволит нам включать и выключать напряжение на нем. Делается это с помощью функции pinMode:

```
3  void setup()
4  {
5    // Устанавливаем порт, к которому подключен светодиод в
6    // режим выхода:
7    pinMode(LED_RED, OUTPUT);
8  }
```

В данном примере мы использовали стандартную функцию pinMode(). Это функция хранится в отдельном файле, о котором знает среда разработки, поэтому при обращении к этой функции программа без труда ее находит.

В скобках функции указываются аргументы – значения или переменные, которые обрабатываются функцией в процессе работы, либо влияют на то, что именно сделает функция.

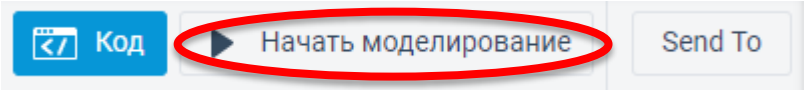
Функции pinMode() необходимо в качестве первого аргумента передать номер настраиваемого порта, а в качестве второго – указать название нужного режима. Порт мы указали LED_RED, что значит «11», а режим – выходной (OUTPUT).

Теперь перейдем внутрь функции «loop», где разместим код для управления светодиодом. Чтобы зажечь светодиод, воспользуемся функцией digitalWrite():

```
10  void loop()
11  {
12    // Подаем напряжение на светодиод:
13    digitalWrite(LED_RED, HIGH);
14  }
```

Здесь мы указали, что нужно подать напряжение на 11 порт, к которому подключен красный светодиод. Стандартная константа HIGH означает 1, а значит, на выходе порта 11 будет максимальное напряжение.

Теперь в верхней правой части экрана нажмем на кнопку «Начать моделирование», после чего светодиод загорится:



Чтобы остановить симуляцию, нажмите «Остановить моделирование».

Теперь заставим светодиод мигать. Для этого нужно:

1. включить светодиод
2. подождать какое-то время
3. выключить светодиод
4. снова подождать
5. вернуться к пункту 1.

Чтобы заставить плату ничего не делать какое-то время, есть функция delay(<время в милисекундах>). Чтобы выключить напряжение на светодиоде, укажем в качестве аргумента функции digitalWrite() стандартную константу LOW.

Полный код программы:

```
#define LED_RED 11

void setup()
{
  // Устанавливаем порт, к которому подключен светодиод в
  // режим выхода:
  pinMode(LED_RED, OUTPUT);
}

void loop()
{
  // Подаем напряжение на светодиод:
  digitalWrite(LED_RED, HIGH);
  // Ждем 1 секунду:
  delay(1000);
  // Гасим светодиод и снова ждем:
  digitalWrite(LED_RED, LOW);
  delay(1000);
}
```

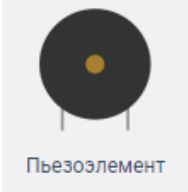
Попробуйте запустить моделирование и убедитесь, что светодиод мигает.

Работа с зуммером

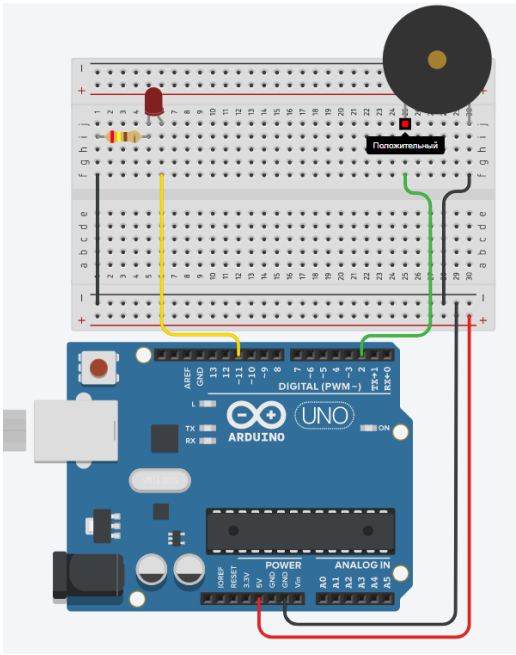
Зуммер – это устройство, которое может издавать звук при подаче на него напряжения. Очень широко распространены пьезоэлектрические зуммеры, которые работают за счет того, что деталь из особого пьезоэлектрического материала внутри них изменяет свои размеры под действием электрического поля, и, как следствие, вибрирует, издавая звук.

Кликнем по кнопке «Код», чтобы убрать окно с написанной нами программой.

Пролистаем список компонентов вниз и найдем там пьезоэлемент:



Подключим положительный контакт пьезоэлемента к порту 2, а отрицательный контакт – к линии заземления.



В начале программы добавим еще одну константу для обозначения номера порта, к которому подключен зуммер:

```
#define LED_RED 11
#define ZUMMER  2
```

Выставим порт зуммера в режим выхода:

```
void setup()
{
  // Устанавливаем порт, к которому подключены светодиод и
  // зуммер в режим выхода:
  pinMode(LED_RED, OUTPUT);
  pinMode(ZUMMER, OUTPUT);
}
```

Чтобы заставить пьезоэлемент издать звук, воспользуемся функцией tone(<номер порта>, <частота в герцах>, <длительность в миллисекундах>):

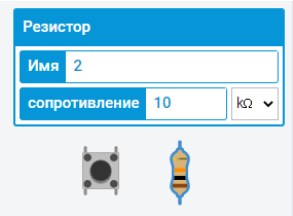
```
void loop()
{
  // Подаем напряжение на светодиод:
  digitalWrite(LED_RED, HIGH);
  // Включаем зуммер на частоте 4000 Гц на 500 милисекунд:
  tone(ZUMMER, 4000, 500);
  // Ждем 1 секунду:
  delay(1000);
  // Гасим светодиод и снова ждем:
  digitalWrite(LED_RED, LOW);
  delay(1000);
}
```

При запуске моделирования убедитесь, что зуммер издает звук.

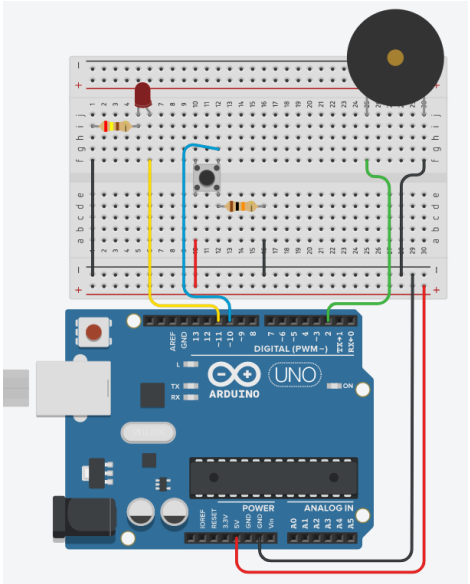
Работа с тактовой кнопкой

Мы почти закончили создавать модель нашего «умного» брелока. Он уже непрерывно подает световой и звуковой сигналы. Тем не менее, неплохо было бы иметь кнопку, которая отключит сигнализацию. Добавим ее в схему.

Найдите среди компонентов тактовую кнопку и резистор. Номинал резистора установим в 10 кОм:



Модернизируем схему нашего устройства следующим образом:



Левый нижний контакт подключаем к линии питания 5 В, правый нижний – к линии заземления через резистор 10 кОм, правый верхний – к порту 10 платы.

Если кнопка не нажата, на порте 10 не будет напряжения. В противном случае, на нем появится напряжение 5 В.

Теперь модернизируем программу. Объявим еще одну константу, обозначающую уже порт, к которому подключена кнопка:

```
#define LED_RED 11
#define ZUMMER  2
#define BUTTON_OFF 10
```

С этого порта нам придется считывать информацию о напряжении, поэтому его необходимо выставить в режим входа, указав стандартную константу INPUT в качестве второго аргумента функции pinMode():

```
void setup()
{
  // Устанавливаем порт, к которому подключены светодиод и
  // зуммер в режим выхода:
  pinMode(LED_RED, OUTPUT);
  pinMode(ZUMMER, OUTPUT);
  // Настраиваем пин кнопки в режиме входа, чтобы
  // мы смогли распознать нажатие:
  pinMode(BUTTON_OFF, INPUT);
}
```

Перейдем внутрь функции «loop». Чтобы распознать нажатие кнопки, нам понадобится переменная. Рассмотрим это понятие, которое является краеугольным камнем программирования.

Переменная – это ячейка памяти вычислительного устройства, за которым закреплено определенное имя. С помощью переменных вычислительные устройства могут хранить информацию до тех пор, пока она не понадобится.

Для лучшего понимания, что такое переменная, можно сравнить ее с ящиком шкафа, на котором есть наклейка с названием вещи, которая хранится в этом ящике. Когда нам понадобится определенная вещь, мы сможем быстро найти её, не перебирая все ящики.



Создадим переменную в начале функции «loop», в которой будет храниться информация о состоянии кнопки (нажата или не нажата) и запишем в нее напряжение, которое приходит с кнопки, с помощью функции digitalRead(<порт>):

```
16  void loop()
17  {
18    // Распознаем нажатие кнопки. Если кнопка нажата,
19    // в button_state запишется HIGH, иначе - LOW:
20    int button_off_state = digitalRead(BUTTON_OFF);
21
22    // Подаем напряжение на светодиод:
23    digitalWrite(LED_RED, HIGH);
```

Чтобы создать переменную, мы сперва указали тип данных int, которые в ней будут храниться. Это означает, что переменная будет хранить целое число. Далее указано имя переменной button_off_state. С помощью оператора присваивания «=» мы сразу записали результат работы функции digitalRead(), которая возвращает напряжение, приходящее от кнопки. Если кнопка нажата, придет значение HIGH (или 1), иначе – LOW (или 0).

Далее нужно модернизировать программу так, чтобы наше устройство могло сделать определенный выбор, в зависимости от того, нажата кнопка или нет. Пусть брелок по умолчанию будет в режиме сигнализации и лишь при нажатии кнопки выйдет из него, включив светодиод. Реализовать это нам поможет управляющая конструкция под названием оператор ветвления:

```
if (условие)  // если условие верно, выполняем
{             // набор инструкций 1
  набор инструкций 1
}
else // иначе, выполняем набор инструкций 2
{
  набор инструкций 2
}
```

Чтобы программа работала по-разному в зависимости от определенных условий, используется конструкция, представленная выше. Если условие в скобках будет выполняться, выполнится набор инструкций из блока оператора if. Если же условие не выполниться, набор инструкций 1 будет пропущен, и выполнение перейдет к блоку инструкций оператора else.

Стоит упомянуть, что оператор else не является обязательным. Можно оставить только блок оператора if. В этом случае, при невыполнении условия, код внутри блока будет просто пропущен, и программа ничего не сделает.

Кроме всего прочего, нужно сказать, каким образом определяется верность или неверность условий. Если условие верное, оно примет значение «1». Если условие не выполнится – «0».

Теперь, зная, как работает оператор ветвления, модернизируем нашу программу:

```
void loop()
{
  // Распознаем нажатие кнопки. Если кнопка нажата,
  // в button_state запишется HIGH, иначе - LOW:
  int button_off_state = digitalRead(BUTTON_OFF);

   // Если кнопка нажата, выключаем сигнализацию:
  if (button_off_state == HIGH)
  {
    while (1)
    {
      digitalWrite(LED_RED, HIGH);
    }
  }
  else
  {
    // Подаем напряжение на светодиод:
    digitalWrite(LED_RED, HIGH);
    // Включаем зуммер на частоте 4000 Гц на 500 милисекунд:
    tone(ZUMMER, 4000, 500);
    // Ждем 1 секунду:
    delay(1000);
    // Гасим светодиод и снова ждем:
    digitalWrite(LED_RED, LOW);
    delay(1000);
  }
}
```

В скобках оператора if расположено условие. Оператор «==» является условным оператором, который проверяет, равно ли значение слева значению справа. Если равенство выполняется, будет произведено действие внутри блока if.

Если кнопка не нажата, значит в переменно button_off_state хранится значение LOW (или 0), следовательно, неравенство не выполняется. В этом случае выполнение перейдет в блок «else», и брелок будет в режиме сигнализации, мигать светодиодом и издавать короткое пищание.

Блок if сработает в том случае, если кнопка была нажата. В переменной button_off_state при этом будет храниться значение HIGH (или 1). А что же находится внутри блока if? Ответ прост, это цикл, подобный нашей функции «loop». Разберемся, как он работает.

```
while (условие)
{
  набор инструкций
}
```

Цикл – это управляющая конструкция, которая множество раз выполняет однотипные действия.

Цикл while, подобно оператору ветвления if, имеет условие. Пока это условие выполняются, будет выполняться набор инструкций внутри блока while. Если же условие в какой-то момент не выполнится, цикл прервется.

В нашем случае в качестве условия стоит «1». Это означает, что цикл не прервется никогда, т.к. это условие будет всегда истинным, следовательно, устройство войдет в режим сигнализации только при перезапуске.

Итоговый код программы

```
#define LED_RED 11
#define ZUMMER  2
#define BUTTON_OFF 10

void setup()
{
  // Устанавливаем порт, к которому подключены светодиод и
  // зуммер в режим выхода:
  pinMode(LED_RED, OUTPUT);
  pinMode(ZUMMER, OUTPUT);
  // Настраиваем пин кнопки в реживм входа, чтобы
  // мы смогли распознать нажатие:
  pinMode(BUTTON_OFF, INPUT);
}

void loop()
{
  // Распознаем нажатие кнопки. Если кнопка нажата,
  // в button_state запишется HIGH, иначе - LOW:
  int button_off_state = digitalRead(BUTTON_OFF);

   // Если кнопка нажата, выключаем сигнализацию:
  if (button_off_state == HIGH)
  {
    while (1)
    {
      digitalWrite(LED_RED, HIGH);
    }
  }
  else
  {
    // Подаем напряжение на светодиод:
    digitalWrite(LED_RED, HIGH);
    // Включаем зуммер на частоте 4000 Гц на 500 милисекунд:
    tone(ZUMMER, 4000, 500);
    // Ждем 1 секунду:
    delay(1000);
    // Гасим светодиод и снова ждем:
    digitalWrite(LED_RED, LOW);
    delay(1000);
  }
}
```

Тестируем устройство

Попробуйте запустить моделирование – устройство начнет мигать светодиодом и издавать писк. При нажатии на кнопку, мигание и писк прекратятся.

Так как в нашей программе имеются задержки, кнопка может не сработать при кратковременном нажатии, поэтому ее стоит удерживать какое-то время.

Модернизация устройства

Задание 1

Модернизируйте устройство, добавив в схему второй светодиод. Мигание светодиодов в режиме сигнализации должно быть попеременным. Пусть загорается первый светодиод, а через секунду гаснет, после чего загорается второй светодиод.

При выходе из режима сигнализации оба светодиода должны гореть непрерывно.

Подсказка: добавьте еще один светодиод и резистор номиналом 240 Ом. Подключите к плате новый светодиод, подобно первому. Для модернизации прошивки используйте функцию digitalWrite() и константы LOW и HIGH.

Задание 2

Модернизируйте устройство, добавив в схему вторую кнопку. При нажатии кнопки 1, устройство должно выйти из режима сигнализации, а при нажатии на кнопку 2 снова войти в него.

Подсказка: добавьте в схему еще одну кнопку и резистор номиналом 10 кОм. Для модернизации прошивки измените условие и блок цикла while.