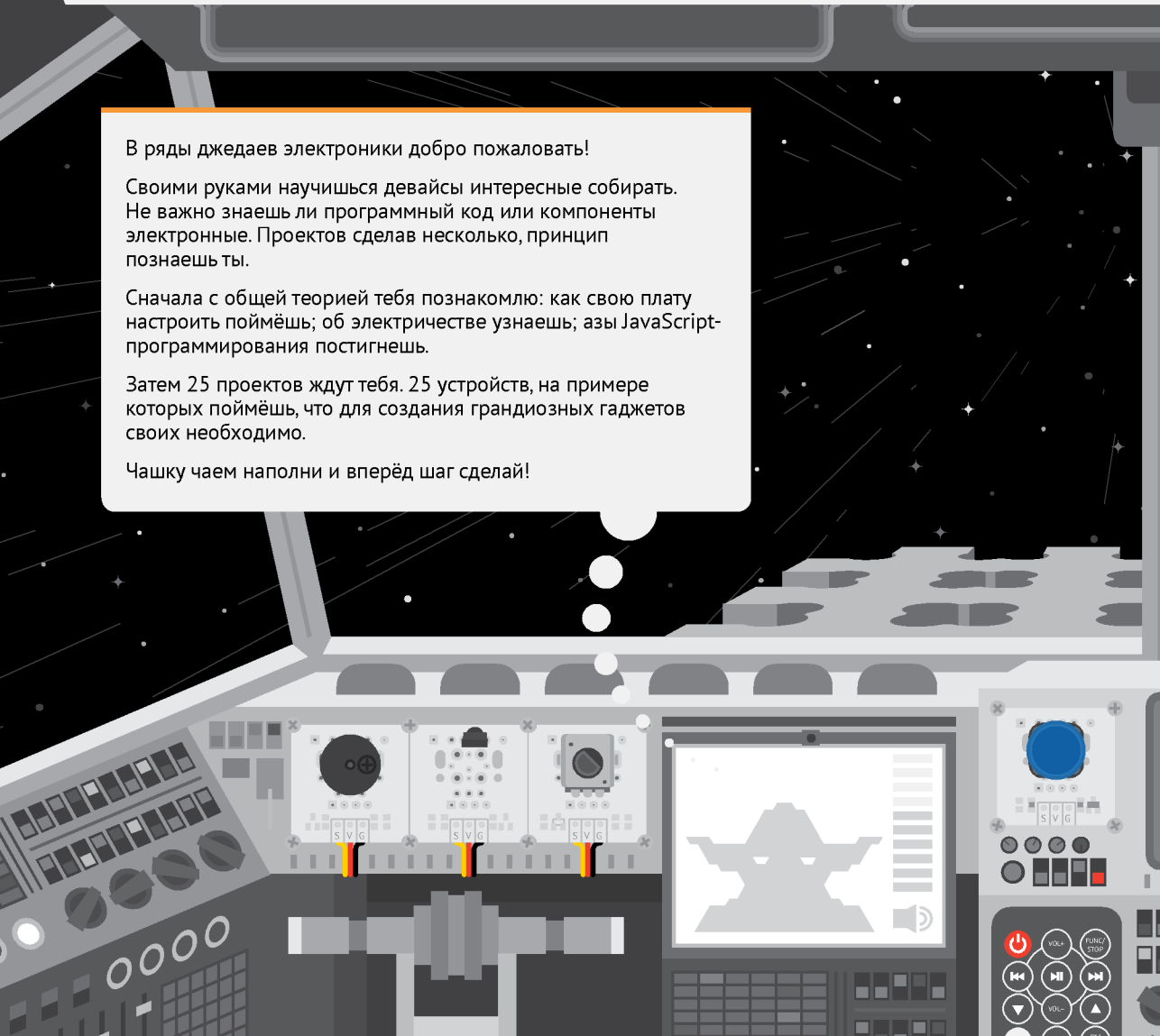
йодо

25 ПРОЕКТОВ НА ISKRA JS

Амперка

йодо



В ряды джедаев электроники добро пожаловать!

Своими руками научишься девайсы интересные собирать. Не важно знаешь ли программный код или компоненты электронные. Проектов сделав несколько, принцип познаешь ты.

Сначала с общей теорией тебя познакомлю: как свою плату настроить поймёшь; об электричестве узнаешь; азы JavaScript-программирования постигнешь.

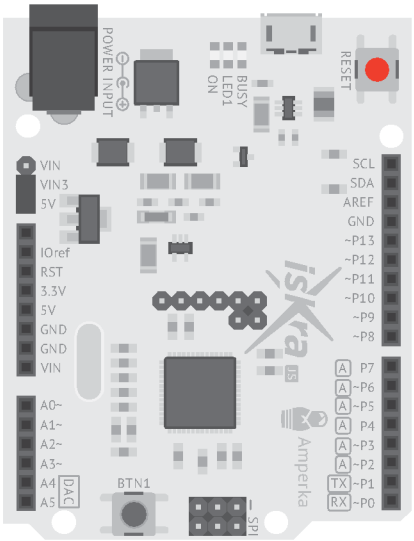
Затем 25 проектов ждут тебя. 25 устройств, на примере которых поймёшь, что для создания грандиозных гаджетов своих необходимо.

Чашку чаем наполни и вперёд шаг сделай!

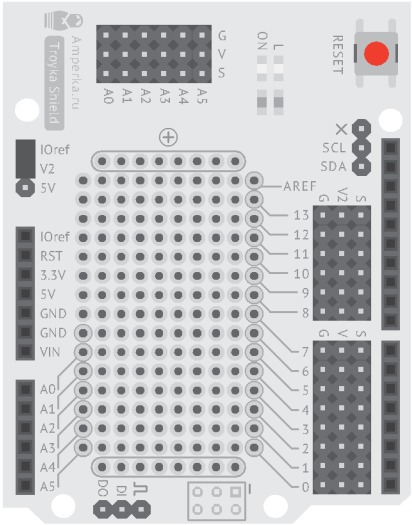
СОДЕРЖАНИЕ

ЭЛЕМЕНТЫ В НАБОРЕ

В набор входит 2 платы, 9 модулей, 8 шлейфов для их подключения, ИК-пульт управления, #структор из 22 элементов, USB-кабель и инструкция.



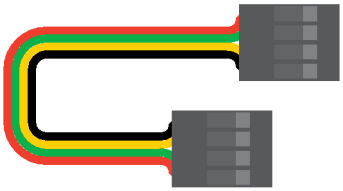
Iskra JS
Мозг твоего устройства



Troyka Shield
Плата расширения для Iskra JS, «хаб» для подключения модулей

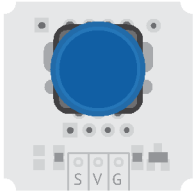
Кабель micro-USB
Соединяет Iskra JS с компьютером

Трёхпроводной шлейф (8 шт)
Соединяет модуль и плату

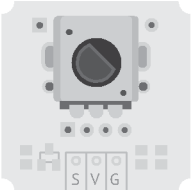


Четырёхпроводной шлейф
Соединяет дальномер и плату

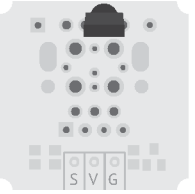
#Структор
Каркас твоего устройства



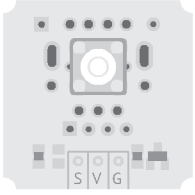
Кнопка
Сообщает о нажатии



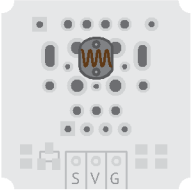
Потенциометр
Сообщает о повороте ручки



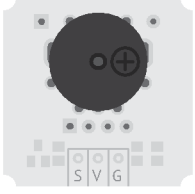
ИК-приёмник
Принимает сигналы
с пульта управления



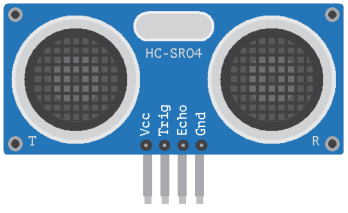
Светодиод
Светит и мигает



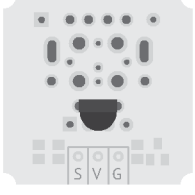
Датчик освещенности
Измеряет яркость света



Зуммер
Пищит, издаёт звуки



Ультразвуковой дальномер
Измеряет расстояние



Термометр
Измеряет температуру воздуха

Сервопривод
Поворачивается
к заданному углу

Пульт
Посылает инфракрасные сигналы

УСТРОЙСТВО ISKRA JS

Iskra JS — это маленький компьютер, мозг твоего устройства. Он умеет измерять напряжение, умеет его выдавать. Этого достаточно, чтобы взаимодействовать с внешним миром: считывать всевозможные сенсоры, выдавать команды на реле, моторы, светодиоды, дисплеи.

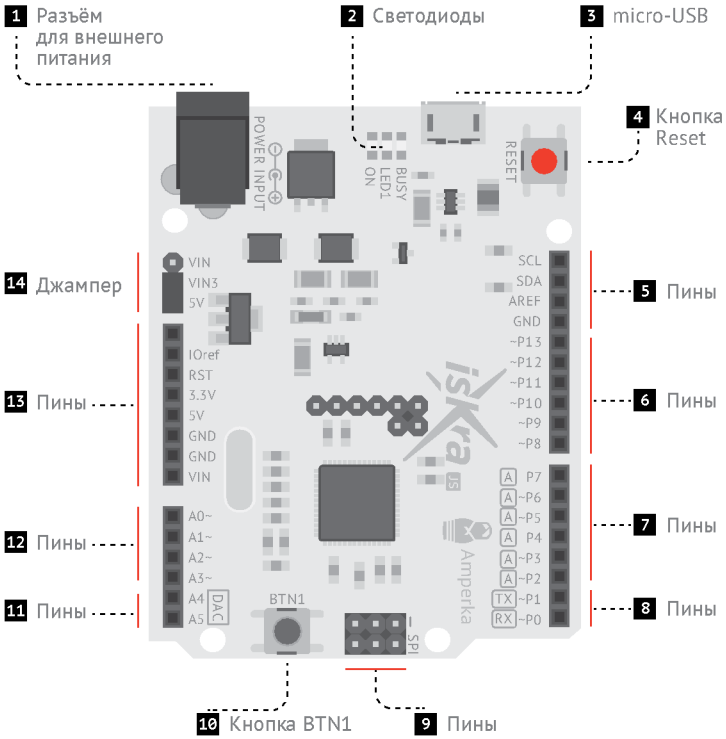
Чтобы получилось законченное устройство, нужно описать его поведение. Это поведение ты можешь запрограммировать на языке JavaScript, загрузить программу в плату и таким образом получить уникальный самостоятельный гаджет!

1
Разъём для питания платы от розетки или аккумулятора.

2
Светодиоды:
• ON — горит, когда на плату поступает питание;
• LED1 — можешь использовать по собственному усмотрению;
• BUSY — включается, когда микроконтроллер производит вычисления или передачу данных.

3
Разъём micro-USB – через него загружают программу, передают данные обратно на компьютер и питают плату.

4
Кнопка RESET — перезагружает плату, последняя сохранённая программа начинает исполняться сначала.



5 6 7 8 9 11 12 13

Колодки с контактами вдоль длинных сторон — это входы и выходы платы. Их называют *пинами*.

6 7 8 11 12

Пины P0…P13 и A0…A5 используют для взаимодействия с модулями, платами расширения и другими устройствами. Их называют *портами GPIO* (General Purpose Input and Output) или просто *портами*.

Порты GPIO в режиме входа могут определять есть ли на них напряжение 3,3 вольта или его нет. В режиме выхода порты могут либо выдавать 3,3 вольта, либо выдавать ноль.

7 11 12

Пины A0…A5 и те, что отмечены A умеют измерять точную величину входного напряжения. К ним подключают аналоговые сенсоры, о которых ты вскоре узнаешь.

6 7 8 12

Те, что отмечены ~ (тильда) умеют переключать выходное напряжение между 0 и 3,3 вольтами тысячи раз в секунду. Это используют, чтобы влиять на яркость светодиодов, скорость моторов, мощность магнитов и т. п.

5 6 9 11

Пины DAC, RX, TX, SPI, SDA, SCL обладают дополнительными функциями, которые нужны при работе с некоторыми модулями.

10

Кнопка BTN1 — можешь использовать по собственному усмотрению.

13

Пины питания:

- 3.3V и IORef выдают ровные и родные для Iskra JS 3,3 вольта.
- 5V выдаёт ровные 5 вольт для модулей, которым не хватает штатных 3,3 вольт.
- VIN выдаёт входное напряжение как есть. То, что приходит через разъём USB или разъём внешнего питания.
- GND — земля платы, точка отсчёта напряжения, 0 вольт.

14

Джампер, который можно перекидывать между двумя положениями:

- VIN3+5V — обычный режим;
- VIN3+VIN — питание от низковольтного аккумулятора.

ТРЮК

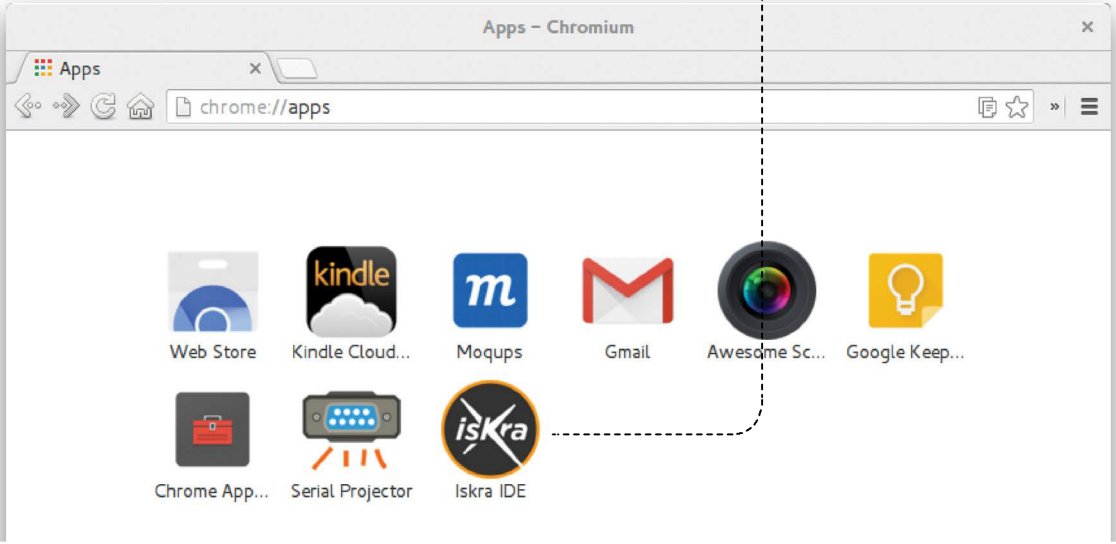
Если зажать BTN1 на момент нажатия RESET, плата после перезагрузки не начнёт исполнять программу, а перейдёт в режим ожидания перепрошивки: светодиоды LED1 и BUSY будут попеременно мигать. Если после этого нажать BTN1 ещё раз, плата перейдёт в обычный режим, но не будет исполнять программу, сохранённую в флеш-памяти. Это полезно, если ты написал такой алгоритм, который «вешает» плату.

УСТАНОВКА IDE

1 Если у тебя ещё не установлен браузер Google Chrome, установи его: google.com/chrome. Не беспокойся, если пользуешься другим браузером: привычки менять не придётся. Google Chrome нужен лишь как платформа для среды программирования.

2 В один клик установи среду программирования IDE для работы с Iskra JS с сайта js.amperka.ru.

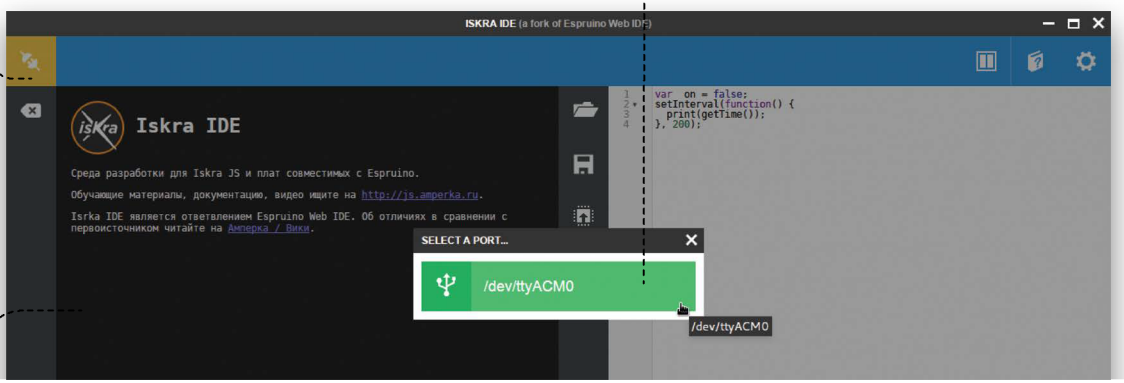
3 Теперь запусти среду. Перейди на вкладку приложений в Google Chrome (chrome://apps) и кликни по иконке приложения.



ПОДСКАЗКА

Правым кликом по иконке приложения в Chrome ты можешь вынести ярлык среды программирования на рабочий стол, чтобы в дальнейшем запускать её напрямую, без запуска Google Chrome.

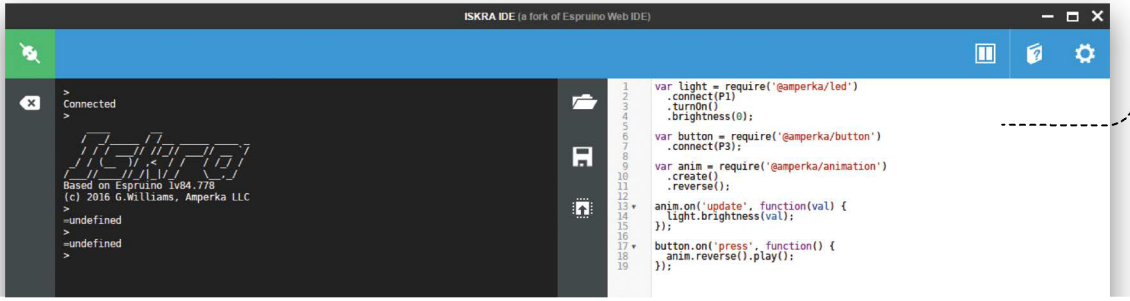
4 Подключи Iskra JS через кабель micro-USB к своему компьютеру, нажми на кнопку соединения и выбери порт.
— *COMx* на Windows
— */dev/cu.usbmodemXXXX* на Mac OS
— */dev/ttyACMx* на Linux
Всё, мы готовы к работе.



Левая панель — это окно *консоли*. Сюда ты можешь на лету посылать инструкции JavaScript, плата их будет исполнять и посылать результат обратно на компьютер.

Правая панель — это окно *программы*. Здесь ты работаешь над кодом. Клик по кнопке заставит среду загрузить код программы в плату, а та начнёт его исполнять.

5 Можно начинать программировать!



НЕМНОГО О JAVASCRIPT

JavaScript — популярный язык программирования. Он простой, выразительный и гибкий. JavaScript обычно используют для создания веб-приложений, но его также может исполнять и твоя Iskra JS.

Теперь попробуй! В окне консоли введи `"Hello!"` (не забудь кавычки) и нажми *Enter*, чтобы отправить строку на плату:

```
>"Hello!"
```

Мгновение спустя, ты увидишь ответ платы:

```
="Hello!"
```

Мы поздоровались. Это означает, что всё настроено правильно и мы готовы к чему-то более серьёзному.

АРИФМЕТИКА

JavaScript знает об арифметике. Введи выражение и нажми *Enter*:

```
>15 + 13 - 3
=25
```

Whoa! Плата получила твою инструкцию, исполнила её и вернула результат «25» обратно. Давай ещё:

```
>5 + (4 - 1) * 3 / 2
=9.5
```

ПЕРЕМЕННЫЕ

Чуть усложним. Добавим *переменных*. Переменная — это значение, которому мы дали имя. Чтобы создать новую переменную, используй слово `var` (от англ. Variable).

```
>var x = 3
=3
>var y = 4
=4
>x * x + y * y + 5
=30
>x = y + 1
=5
>x * x + y * y + 5
=46
```

Переменные могут хранить не только числа, но и строки, логические значения, функции, составные объекты. Обо всём этом ты узнаешь в своё время.

ФУНКЦИИ

А сейчас давай наконец попробуем сделать что-то специфичное для платы:

```
>getTime()
=1425.66402724404
```

Обрати внимание, что буква T — заглавная, а остальные — строчные. Регистр важен.

Мы вызвали встроенную *функцию* с именем `getTime`. Она возвращает время в секундах, прошедшее с момента включения платы. Попробуй вызвать её несколько раз и посмотри, как меняется результат.

ОБЪЕКТЫ И МЕТОДЫ

Теперь попробуем порулить электричеством!

```
>LED1.write(1)
=undefined
```

А сейчас переведи свой взгляд на плату: светодиод LED1 горит! Давай выключим:

```
>LED1.write(0)
=undefined
```

Теперь попробуй сам помигать светодиодом.

Разберёмся, что здесь происходит. Мы обращаемся к встроенной переменной-объекту `LED1` и вызываем *метод* `write` с *параметром* `1`... Обо всём по порядку.

Метод — это функция, которая работает с объектом, на котором её вызвали.

В скобках мы передаём методу параметры. У `LED1.write` параметр один — значение, которое определяет нужно ли выключить светодиод (ноль) или включить (не ноль).

Читай инструкцию целиком так: «Эй, светодиод1, запиши ка мне единичку».

И наконец, ответ — `undefinded`. В JavaScript это означает «пусто», «ничего», «дырка от бублика». Функция `getTime` возвращала нам время, а `LED1.write` не возвращает ничего, поэтому мы видим `undefinded`.

ПОПРОБУЕМ ЕЩЁ?

`BTN1` — встроенный объект, который олицетворяет кнопку на Iskra JS. Метод `read` считывает текущее состояние кнопки. Параметры ему не нужны.

```
>BTN1.read()
=false
```

Метод вернул `false`. Это логическое значение, которое означает «ложь», «нет», «ноль»: кнопка не нажата.

Теперь зажми кнопку и выполни инструкцию ещё раз:

```
>BTN1.read()
=true
```

О! Теперь мы видим `true`. Это «истинна», «да», «единица»: кнопка зажата.

ЕЩЁ БОЛЬШЕ ФУНКЦИЙ

Встроенных функций и методов много. У них разные параметры. Как ими пользоваться, что они делают, ты можешь узнать на js.amperka.ru

ОБ ЭЛЕКТРИЧЕСТВЕ

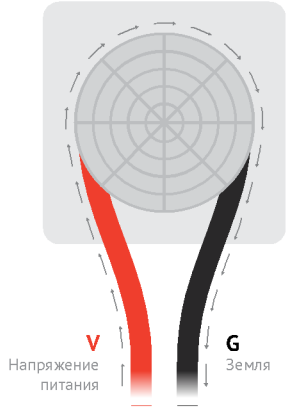
Любому электронному компоненту для работы нужно питание.
Питание подводят по двум проводам.
Помнишь, как выглядит розетка?

По одному проводу электричество втекает, по другому — вытекает. Тот провод, по которому ток втекает (● красный провод), называют *плюсом* или *+V* (англ. Voltage), или *Vcc*, или просто *питанием*, или же сколькими-то вольтами (например 3,3 вольта).

Провод, по которому ток вытекает (● чёрный провод), называют *минусом* или *землёй* (англ. Ground, GND), или *нулём вольт*.

> Не путай *землю* устройства и планету Земля. В слаботочных устройствах эти понятия никак не связаны.

Электричество передают не только по проводам, но и по дорожкам на плате, через разъёмы и даже по воздуху. Поэтому вместо слова «провод» также употребляют слово «линия»: линия питания, линия земли.

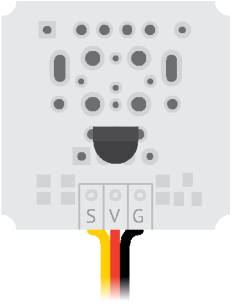


СИГНАЛЫ

Просто включить устройство — хорошо, но как с ним можно общаться? Как электронные устройства передают друг другу *информацию*? Люди могут использовать для этого жесты, письменность, голос. Электронные устройства используют *напряжение*.

В большинстве модулей ты увидишь третий провод — *сигнальную линию* (● жёлтый провод). По ней передаётся информация, которая закодирована *величиной* напряжения.

Преобразовывать информацию в напряжение можно по-разному. Существует два основных вида сигналов: *аналоговые* и *цифровые*.



Термометр и трехпроводной шлейф

● G – земля (чёрный провод)
● V – питание (красный провод)
● S – сигнал (жёлтый провод)

АНАЛОГОВЫЕ СИГНАЛЫ

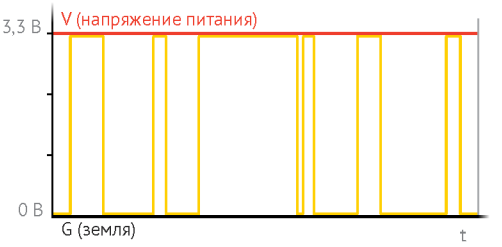
В аналоговом сигнале величина напряжения в вольтах и определяет числовое значение, которое передаётся.

Микроконтроллер может считать напряжение на сигнальной линии и с помощью нехитрой арифметики вычислить значение. Например, температуру с термометра в градусах Цельсия.

При помощи аналоговых сигналов можно общаться с потенциометром, датчиком освещённости, термометром.

Аналоговые модули можно подключать к портам A0…A5 или P2…P7.

ЦИФРОВЫЕ СИГНАЛЫ



Здесь величина напряжения в любой момент времени равна либо 0 вольт, либо напряжению питания. А передаваемое значение определяется *последовательностью* и *длительностью* напряжений.

Напряжение питания на сигнальной линии при этом называют *логической единицей*, а ноль вольт — *логическим нулём*.

Микроконтроллер может измерить время и темп импульсов, чтобы восстановить значение.

При помощи аналоговых сигналов можно общаться с сервоприводом, ИК-приёмником, светодиодом, зуммером, кнопкой, дальномером.

Цифровые модули можно подключать к любым портам Iskra JS.

ПЛЮСЫ-МИНУСЫ

+ Просто создавать, просто считывать, просто декодировать.

– Сложно обеспечить высокую точность и широкий диапазон передаваемых значений.

– При передаче по длинным проводам сигнал набирает помехи из радиоволн, показания искажаются.

ПЛЮСЫ-МИНУСЫ

– Сложно кодировать и декодировать.

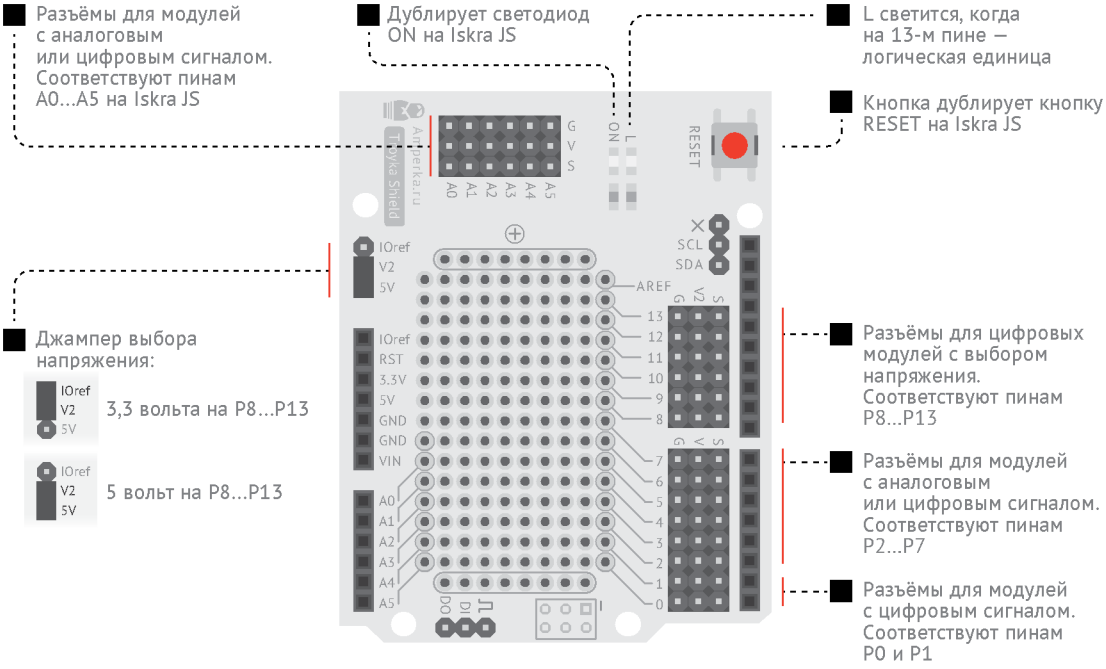
+ Значения передаются в первозданном виде: с произвольной точностью и в произвольном диапазоне.

+ Приёмник округляет напряжение до логической единицы и нуля, поэтому помехи в длинных проводах — не проблема.

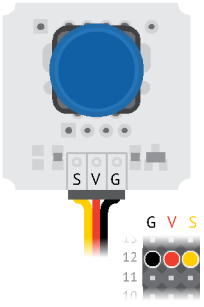
ПЛАТА TROYKA SHIELD

На Iskra JS — 20 портов GPIO, а пинов питания всего несколько. Когда ты захочешь подключить десяток-другой сенсоров и модулей, придётся думать о том, как разветвить линии GND и 3.3V: питание ведь нужно каждому модулю. Чтобы об этом не думать, есть *Troyka Shield*.

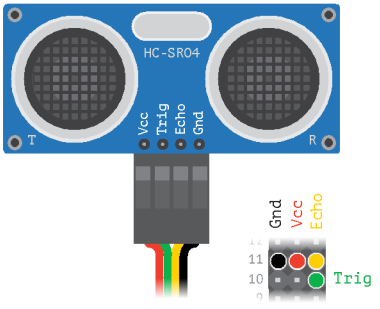
Это плата расширения для Iskra JS, которая каждый пин P0…P13 и A0…A5 превращает в тройной разъём «земля–питание–сигнал». Теперь достаточно соединить разъём на Troyka Shield и модуль трёхпроводным шлейфом, и он готов к работе.



Кнопка



Ультразвуковой дальномер



НАПРЯЖЕНИЕ ПИТАНИЯ

По линиям, отмеченным *V* Troyka Shield подводит родные для Iskra JS 3,3 вольта. Но некоторым модулям не достаточно этого напряжения для работы. Поэтому на пинах P8…P13 предусмотрено альтернативное напряжение *V2*.

В наших проектах мы будем использовать ультразвуковой дальномер и сервопривод, поэтому установи джампер в положение V2+5V.

TROYKA-МОДУЛИ

Troyka-модули легко подключаются к Troyka Shield через трёхпроводные шлейфы. Достаточно соединить соответствующие линии между модулем и платой:

- G – земля (чёрный провод)
- V – питание (красный провод)
- S – сигнал (жёлтый провод)

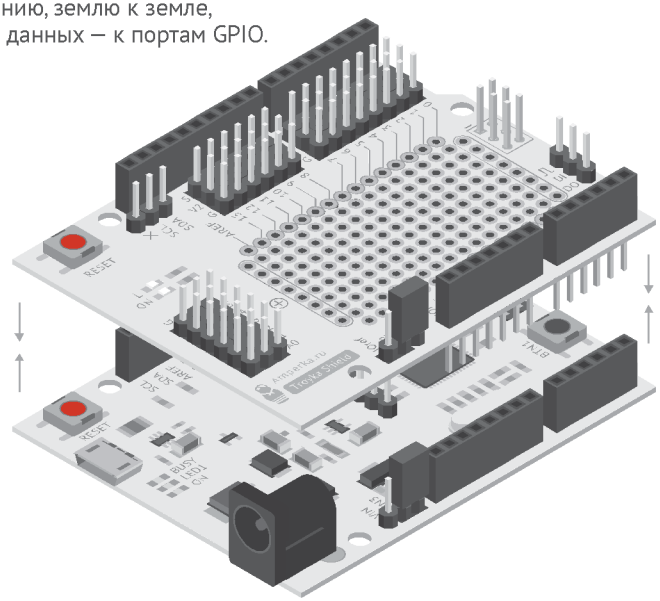
ДРУГИЕ МОДУЛИ

Существует множество сторонних модулей, у которых пины выведены не тройками, а как-то иначе. Такие модули также можно подключить к Troyka Shield. Главное — следи за тем, чтобы подключить питание к питанию, землю к земле, линии данных — к портам GPIO.

КАК ПОДКЛЮЧАТЬ

Вставь Troyka Shield в пины Iskra JS сверху. Ты получишь единое устройство.

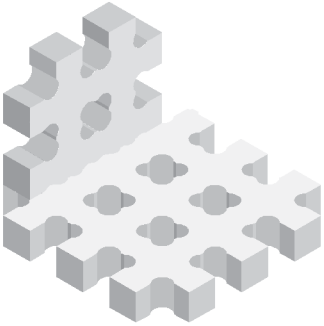
Для соединения с компьютером, как и раньше, используй USB-порт на Iskra JS. А для подключения модулей используй разъёмы на Troyka Shield.



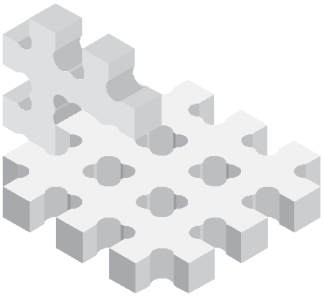
#СТРУКТОР

Модули, висящие на проводах — не лучший дизайн устройства. Используй детали #структора, чтобы создать скелет для своего устройства. А чтобы закрепить электронные модули, сенсоры, приводы, используй детали-переходники.

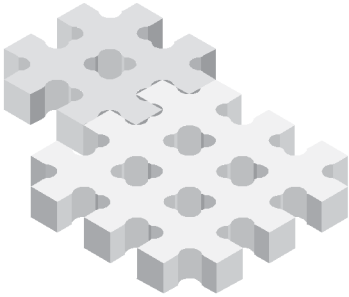
#Структор сделан из вспененного ПВХ и из него легко можно построить практически любой каркас.



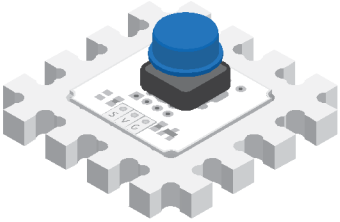
Крепление под уголом



Перпендикулярное крепление



Параллельное крепление



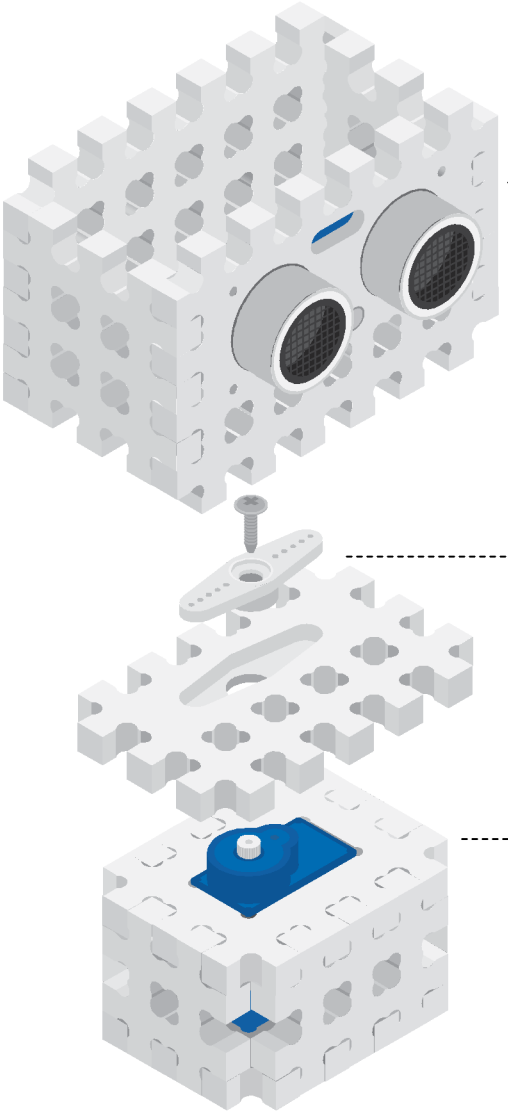
В наборе есть специальные детали для крепления дальномера, сервопривода и остальных модулей.

АЛЬТЕРНАТИВЫ

Также ты можешь сделать корпус своего устройства из обыденных вещей: пищевых контейнеров, старых игрушек, шкатулок, картонных коробок.

Ещё один вариант — напечатать нужные детали на 3D-принтере или вырезать лазером в мастерской.

Существуют и готовые корпуса-полуфабрикаты, в которых остаётся только насверлить отверстий для разъёмов и кнопок. Включай фантазию — вариантов масса!



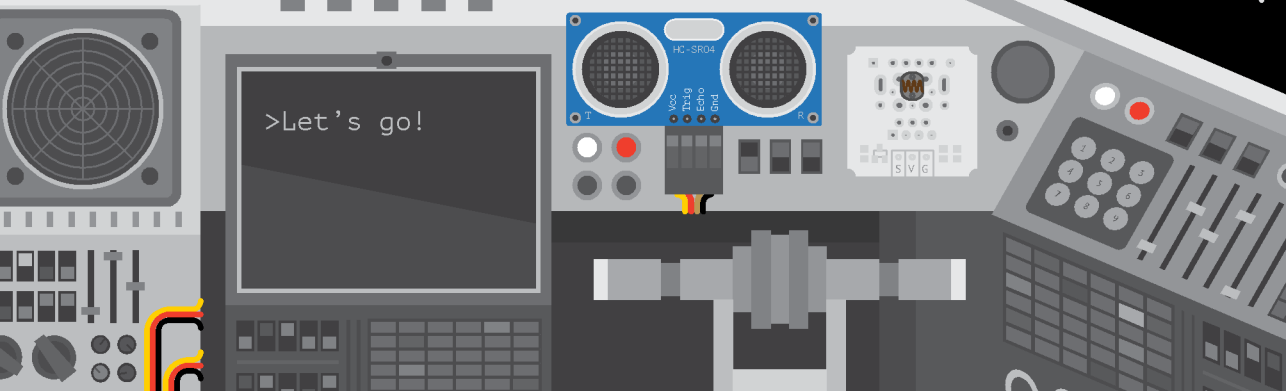
Пример того, как можно собрать с помощью #структора проект №25 «Настольный радар»:

1 Зафиксируй дальномер с помощью специальной детали #структора.

2 В специальной выемке зафиксируй одну из качелек сервопривода и закрепи на валу винтом. Это позволит сервоприводу вращать дальномер и «сканировать» пространство вокруг.

3 Для фиксации сервопривода используй детали с прямоугольными выемками.

ПРОЕКТЫ



ЛАМПА

№1

Светодиод сделаем, который всегда горит. Микроконтроллер использовать, чтобы просто свет включить — это перебор. Но ведь учишься ты!

В разъём 1 на плате Troyka Shield модуль со светодиодом подключи. Troyka Shield на Iskra JS поставь. Среду программирования запусти. В *правой панели* IDE код целиком напиши и для запуска кликни.

```
1  var myCoolLamp = require('@amperka/led').connect(P1);
2  myCoolLamp.turnOn();
```

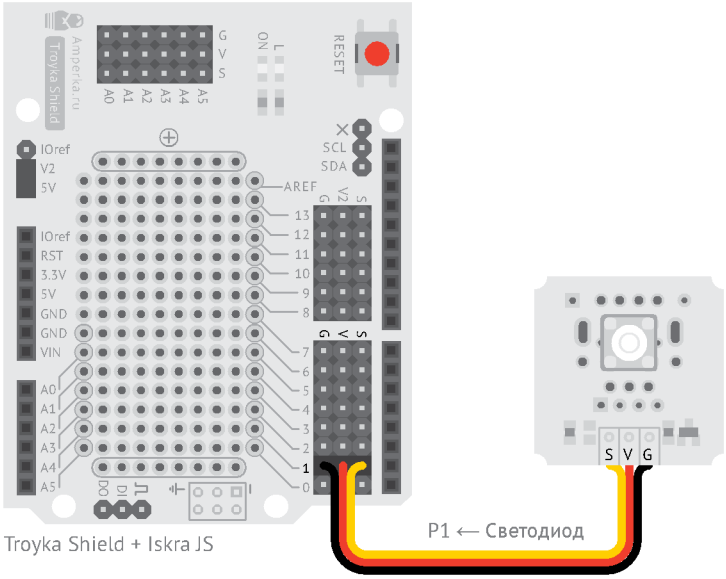
1 Создаём переменную с именем `myCoolLamp` и говорим, что это светодиод, который подключен к пину P1

`require` означает, что мы хотим использовать объект. `'@amperka/led'` — что это объект-светодиод. `connect(P1)` — что он подключен к первому пину.

Мы сохраняем объект-светодиод в переменной, которой дали имя `myCoolLamp`. Придумай другое имя, если хочешь.

2 У объекта-светодиода вызываем метод `turnOn`, чтобы включить его.

У каждого модуля есть свой набор методов. Подробнее о всех методах смотри в справочнике на последней странице.



Troyka Shield + Iskra JS

ЗАДАНИЕ

Через другие порты светодиод включить попробуй.

МАЯЧОК

№2

Мигающий светодиод сделаем. Можешь показывать им, что живо устройство или действия ждёт.

```
1  var led = require('@amperka/led')
2    .connect(P1);
3
4  led.blink(0.1, 0.9);
```

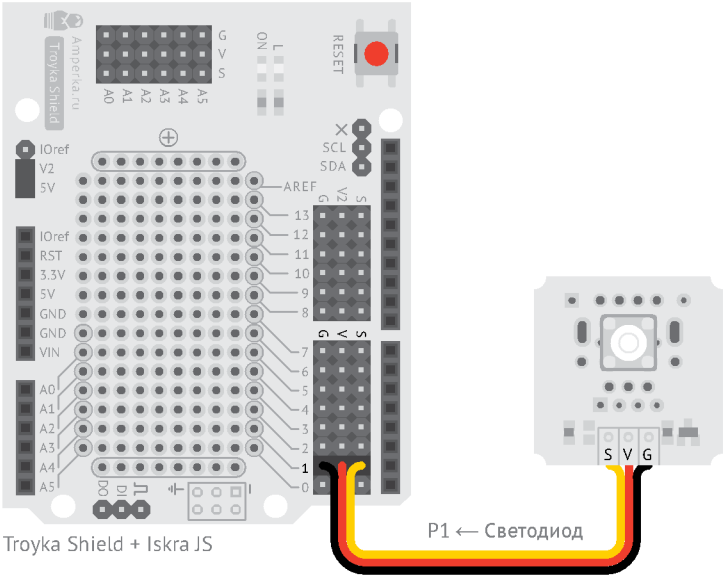
1 Для стройности выражения можно разбивать на несколько строк.

Чтобы код было легче читать, отбивай дополнительные строки двумя пробелами.

Главное — не забывай ставить «;» (точку с запятой) в конце.

2 Заставим светодиод мигать! Пусть он светится одну десятую секунды, а затем гаснет на девять десятых. И так по кругу. Именно это делает метод `blink`.

Если не передать в `blink` второй параметр, светодиод моргнёт всего один раз на время, заданное первым параметром.



Troyka Shield + Iskra JS

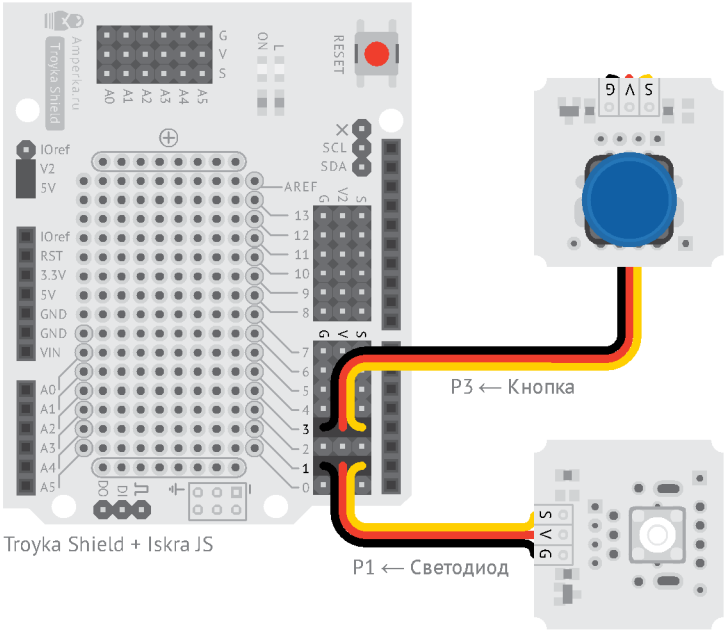
ЗАДАНИЕ

На 4 мигания в секунду темп измени.

КНОПОЧНЫЙ ВЫКЛЮЧАТЕЛЬ

№3

Света переключатель давай сделаем.
Клик — свет включится, клик — выключится.



1 Создаём переменную с именем `button` и говорим, что это объект-кнопка (`'@amperka/button'`) подключённая (`connect`) к пину P3.

2 Слово `function` говорит о том, что мы заводим новую *функцию*.

Функция — это блок кода, у которого есть имя. Выражения внутри неё выполняются, когда кто-нибудь эту функцию *вызывает*. Функции заводят, чтобы использовать один и тот же код снова и снова.

После слова `function` следует желаемое имя. Мы назвали её `myCoolButtonHandler`.

В круглых скобках перечисляются *параметры*. У нас их нет, поэтому там пусто, но скобки обязательны.

И наконец, в фигурных скобках следует *тело* функции, т.е. инструкции, которые будут исполняться при вызове.

```
var led = require('@amperka/led')
  .connect(P1);

var button = require('@amperka/button')
  .connect(P3);

function myCoolButtonHandler() {
  led.toggle();
}

button.on('press', myCoolButtonHandler);
```

3 Метод `toggle` переключает «включённость» светодиода:
- если он был выключен — включает;
- если включён — выключает.

4 Некоторые объекты генерируют события. В определённые моменты они как бы говорят «Эй, у меня случилось что-то важное».

Мы можем подписаться на событие, чтобы всякий раз, когда оно происходит, вызывать одну из наших функций.

У кнопок есть событие `press`. Оно происходит в момент нажатия кнопки, один раз на каждое нажатие.

Подпишемся на нажатие. Для подписки на события у объектов есть метод `on`. Есть он и у объекта-кнопки. Первым параметром передаём имя события (`press`), вторым — функцию, которую мы хотим исполнять при наступлении события. Мы используем функцию `myCoolButtonHandler`, которую создали ранее.

Все события, которые есть у кнопок, ищи в приложении на последней странице.

ЗАДАНИЕ

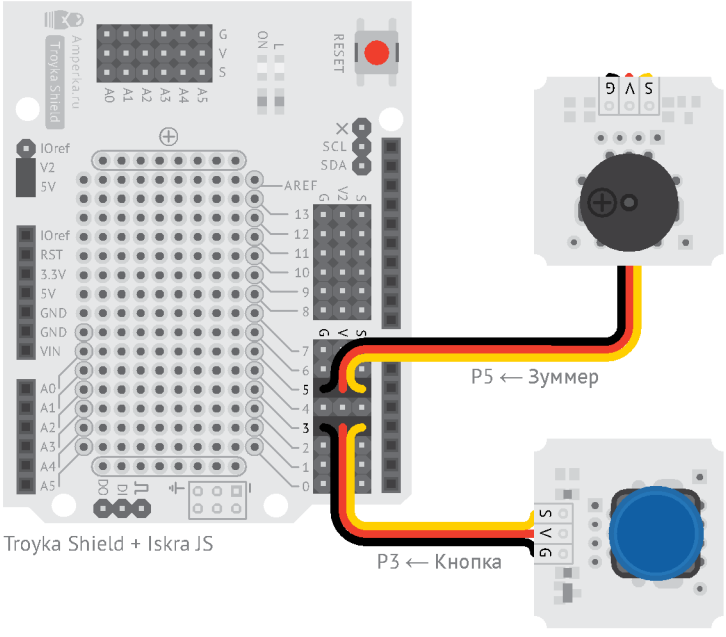
Так сделай, чтобы нажатие на кнопку включало на 1 секунду светодиод, а после сам бы выключался он.

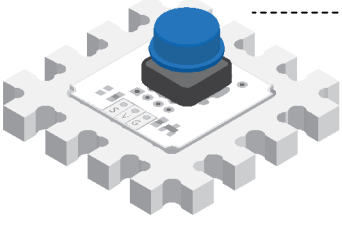
Предыдущий проект внимательнее перечитай, если решение придумать не можешь.

ТЕЛЕГРАФ

№4

Звуковых сообщений простых передатчик сделаем мы. Телеграмму поможет передать он.





Используй детали #структора, чтобы зафиксировать модули.

1 Создаём объект-пьезоизлучатель с именем `buzzer`, подключённый к пину P5.

Ты наверняка уже понял принцип создания переменных, которые олицетворяют электронные модули. Рассмотрим подробнее, что при этом происходит.

`require` — встроенная функция, которая подключает *библиотеку*. Библиотека — это программный код, который уже кто-то написал. Библиотеки существуют для удобства: достаточно один раз создать набор полезных функций, объектов и методов, а затем использовать их снова и снова.

В качестве параметра в скобках передают *имя* нужной библиотеки. При загрузке кода в плату среда загружает код запрошенной библиотеки из интернета и приклеивает её содержимое к твоей программе, что даёт доступ к её возможностям.

```
var buzzer = require('@amperka/buzzer')
  .connect(P5);

var button = require('@amperka/button')
  .connect(P3);

button.on('press', function() {
  buzzer.turnOn();
});

button.on('release', function() {
  buzzer.turnOff();
});
```

`connect` — это просто одна из функций библиотеки, которая сделана для создания объекта управления электронным модулем. В качестве параметра она принимает пин, к которому подключён модуль.

Перечень некоторых библиотек и их функций ты можешь узнать из приложения на последней странице.

2 Если одна функция принимает в качестве параметра другую функцию, её можно создавать прямо на месте, без имени. Такие функции называют *анонимными*.

Наша функция вызывает у объекта `buzzer` метод `turnOn`. Тем самым мы включаем пищалку при наступлении события `press`, т.е. при нажатии на кнопку. Модуль будет пищать пока кнопка зажата.

3 Событие `release` у объектов-кнопок наступает один раз на каждое отпускание кнопки. То есть:
- зажали — вызвался `press`;
- отпустили — вызвался `release`.

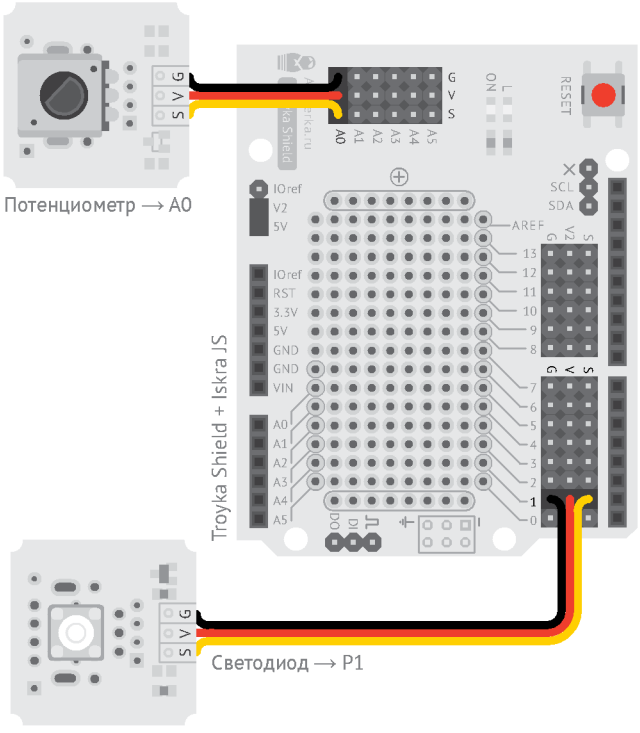
ЗАДАНИЕ

В проект светодиод добавь. Пусть в один момент с зуммером включается он. Сообщение не только звуком, но и светом такой телеграф передаст.

ДИММЕР

№5

Силы света регулятор сделаем мы. Ручку крути, чтобы сторону выбрать свою: тёмную или светлую.



1 `@amperka/pot` — библиотека для работы с модулем-потенциометром.

2 Вызовы методов можно сцеплять: вызывать один за другим через точку. Здесь мы подключаем светодиод и следом сразу же его включаем.

```
var pot = require('@amperka/pot')
  .connect(A0);

var led = require('@amperka/led')
  .connect(P1)
  .turnOn();

function updateBrightness() {
  var val = pot.read();
  led.brightness(val);
}

setInterval(updateBrightness, 10);
```

3 Создаём функцию с именем `updateBrightness`, которая преобразует значение с потенциометра в яркость светодиода.

Помни, содержимое функции срабатывает не сразу, а когда функцию вызывают.

4 Заводим переменную с именем `val` и присваиваем ей значение, которое отдаёт нам метод `read` объекта-потенциометра. Результат этого метода — вещественное число от 0 до 1, которое соответствует углу поворота ручки потенциометра:

- `0.0` — упор слева;
- `0.5` — по центру;
- `1.0` — упор справа и т.д.

5 Установим яркость светодиода. Для этого есть метод `brightness`. Он принимает 1 параметр: значение от 0 до 1, которое определяет яркость. Мы в качестве параметра используем значение переменной `val`, которую только что определили.

6 Чтобы что-то происходило периодически и до бесконечности, в JavaScript есть функция `setInterval`. Первым параметром она принимает функцию, которую нужно вызывать периодически, вторым — интервал в миллисекундах.

Мы используем созданную нами функцию `updateBrightness`, чтобы светодиод менял яркость в зависимости от поворота ручки каждые 10 миллисекунд. 100 раз в секунду — более, чем достаточно, чтобы глаз не заметил задержек.

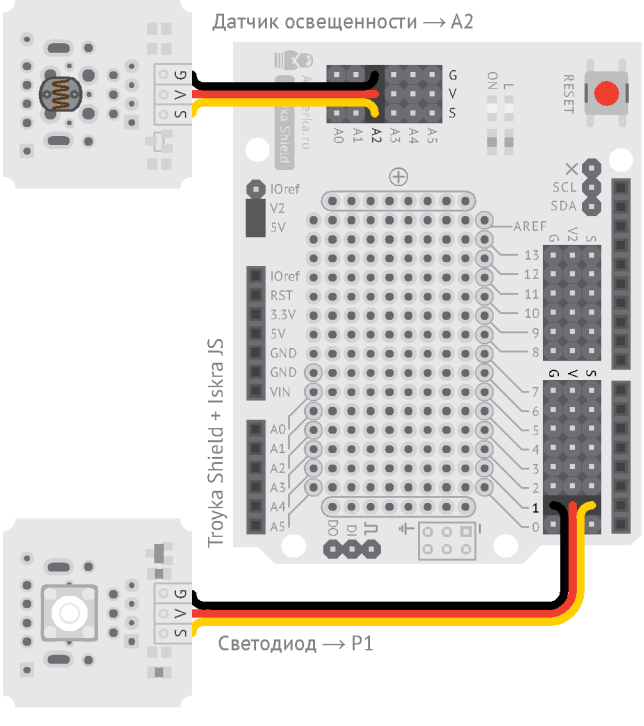
ЗАДАНИЕ

В проект кнопку добавь. Нажатие её пусть то тьму включает, то к работе диммер возвращает.

АВТОМАТИЧЕСКИЙ ДИММЕР

№6

Разум диммеру дадим. Пусть тем ярче свет горит, чем окружающий мир темнее.



1 `@amperka/light-sensor` — модуль для работы с датчиком освещённости.

2 Для компактности на этот раз используем анонимную функцию

3 У сенсора освещённости есть метод `read` для считывания текущего значения. В качестве параметра можно передать желаемые единицы измерения. `lx` означает люксы.

Место	*Освещённость (люкс)*
Ночь без луны	0,0003
Полнолуние	0,27
Жилая комната	50
Очень пасмурный день	100
Светлый офис	400
Облачный день	1000
Ясный летний день в тени	20000
Ясный летний день на солнце	100000

Записываем результат в переменную `luxes`.

```
var led = require('@amperka/led')
  .connect(P1)
  .turnOn();

var sensor = require('@amperka/light-sensor')
  .connect(A2);

setInterval(function() {
  var luxes = sensor.read('lx');
  var level = 1 - luxes / 50;
  led.brightness(level);
}, 10);
```

5 Устанавливаем рассчитанную яркость светодиода. Даже если переданное значение выйдет за границы от 0 до 1, метод `brightness` позаботится о том, чтобы привести значение к допустимому.

4 С помощью арифметического выражения рассчитываем значение для ещё одной переменной, `level`. Формулу мы составили так, чтобы яркость была максимальной при нулевом уровне освещённости сенсора и плавно сходила к нулю в диапазоне до 50 люкс.

Ты можешь использовать все арифметические операторы, которые знаешь из математики.

Оператор	*Действие*
`a + b`	Сложение `a` и `b`
`a - b`	Вычитание `b` из `a`
`a * b`	Умножение `a` на `b`
`a / b`	Деление `a` на `b`
`a % b`	Остаток от деления `a` на `b`
`a + b * c`	Умножение `b` на `c` и затем сложение с `a`
`(a + b) * c`	Сложение `a` и `b`, а затем умножение на `c`
`-a`	Эквивалентно `0 - a`

ЗАДАНИЕ

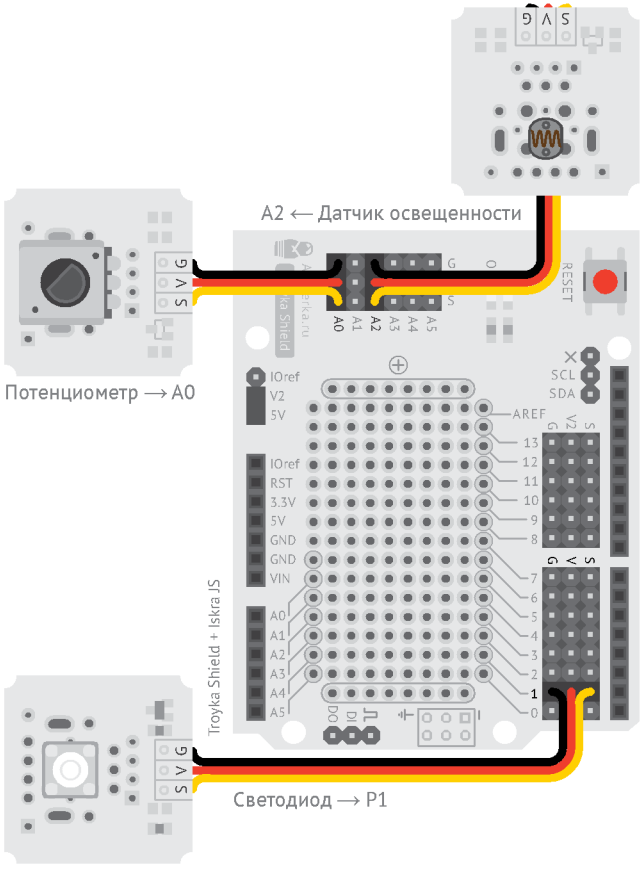
Потенциометр добавь. Пусть яркость светодиода итоговую приглушать можно будет. Без этого ночью джедаю светильник уснуть не даст.

Подробнее о сенсоре освещённости: js.amperka.ru → @amperka/buzzer

УМНОЕ ОСВЕЩЕНИЕ

№7

Энергию сбережём давай. Свет во тьме включаться
и на рассвете гаснуть заставим. Границу света и тьмы
потенциометр задаст.



1 `if` — это *условное* выражение. Их используют, чтобы сделать в коде *ветвление*, когда в зависимости от некоего условия выполняется либо один код, либо другой.

Условие записывается в круглых скобках. Интерпретатор проверяет истинно ли оно или ложно. Если выражение истинно, он выполняет код в фигурных скобках, следующих прямо за условием. Если выражение ложно, код в этих фигурных скобках он пропустит.

Для записи условия ты можешь использовать операторы сравнения, логические операторы «и», «или», «не».

```
var led = require('@amperka/led')
  .connect(P1);

var pot = require('@amperka/pot')
  .connect(A0);

var sensor = require('@amperka/light-sensor')
  .connect(A2);

setInterval(function() {
  var threshold = pot.read() * 100;
  var luxes = sensor.read('lx');
  if (luxes < threshold) {
    led.turnOn();
  } else {
    led.turnOff();
  }
}, 10);
```

2 Включаем светодиод, если `luxes` меньше, чем `threshold`.

3 `else` — это ветвь, которая выполняется, только если условие в `if` было ложным. Ветку `else`, если она вам не нужна, можно к `if` не добавлять. В нашем примере она нужна.

4 Выключаем светодиод, если `luxes` был не меньше, чем `threshold`.

Оператор	*Значение*
`a < b`	`a` меньше `b`?
`a <= b`	`a` меньше или равно `b`?
`a > b`	`a` больше `b`?
`a >= b`	`a` больше или равно `b`?
`a === b`	`a` равно `b`?
`a !== b`	`a` не равно `b`?
`a < b && a >= c`	`a` меньше `b` и `a` больше либо равно `c`
`a < b || a >= c`	`a` меньше `b` или `a` больше либо равно `c`
`a`	`a` истинно или не ноль?
`a || b`	`a` истинно или не ноль и `b` истинно или не ноль
`!a && !b`	`a` ложно или ноль и `b` ложно или ноль

ЗАДАНИЕ

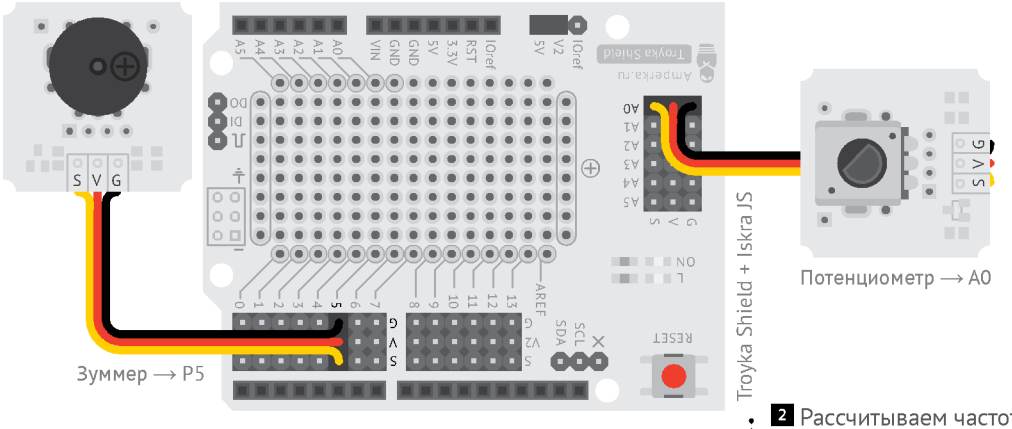
Очень медленно потенциометр поворачивай. Границу света и тьмы поймай. Светодиод хаотично гореть и гаснуть будет. Эффекта сего причина — шум вселенной в сигнале датчика освещённости.

Каждый день на закате и рассвете эффект этот беспокоить будет. Программу так измени, чтобы с нестабильностью навсегда справиться.

ЭЛЕМЕНТАРНЫЙ СИНТЕЗАТОР

№8

Музыкой заняться пора. С космическим звучанием и одной ручкой синтезатор построим.




```
var buzzer = require('@amperka/buzzer')
  .connect(P5)
  .turnOn();

var pot = require('@amperka/pot')
  .connect(A0);

setInterval(function() {
  var freq = 20 + 4000 * pot.read();
  buzzer.frequency(freq);
}, 10);
```

1 Создаём объект-пьезоизлучатель и сразу его включаем. Пусть пищит.

2 Рассчитываем частоту звука в зависимости от поворота ручки потенциометра. Получим значение от 20 до 4020.

3 Меняем частоту звучания, то есть ноту. Метод `frequency` принимает значения в герцах, поэтому мы получим тон от «баса», на который только способна крохотная пищалка, до пронзительного визга.

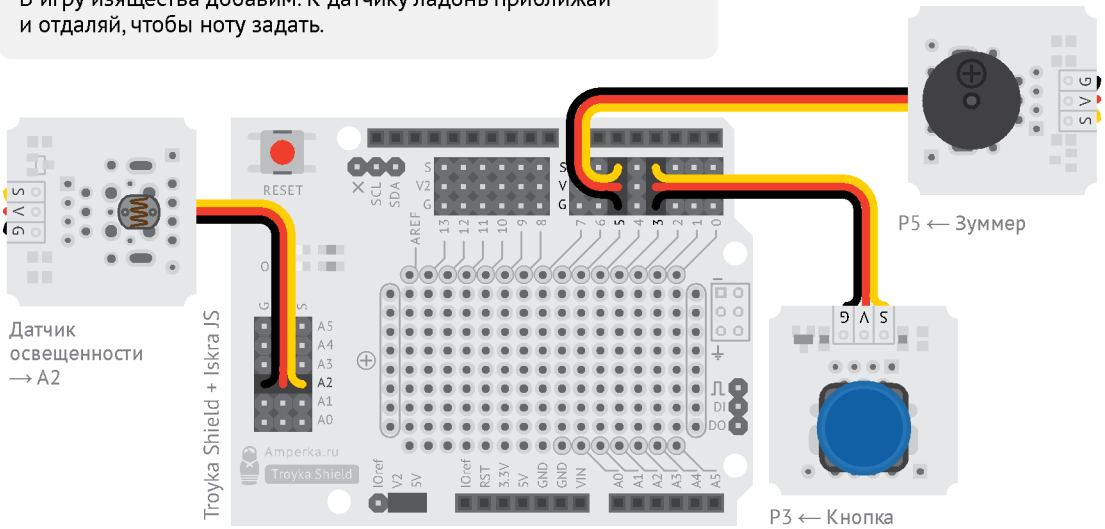
ЗАДАНИЕ

До 20 килогерц Диапазон частот синтезатора измени.
Возможностей зуммера и ушей своих предел узнаешь ты.

ТЕРМЕНВОКС

№9

В игру изящества добавим. К датчику ладонь приближай и отдаляй, чтобы ноту задать.




```
var buzzer = require('@amperka/buzzer')
  .connect(P5)
  .turnOn();

var sensor = require('@amperka/light-sensor')
  .connect(A2);

var button = require('@amperka/button')
  .connect(P3);

button.on('press', function() {
  buzzer.toggle();
});

setInterval(function() {
  buzzer.frequency(20 * sensor.read('lx'));
}, 10);
```

1 Параметры для методов и функций можно рассчитывать на месте, внутри круглых скобок. Заводить отдельную переменную не обязательно. Но если выражение громоздкое, расчёт отдельной строкой сделает программу читабельнее.

ЗАДАНИЕ

Потенциометр добавь, что «октаву» меняет: тот диапазон частот, в котором терменвокс звучит.

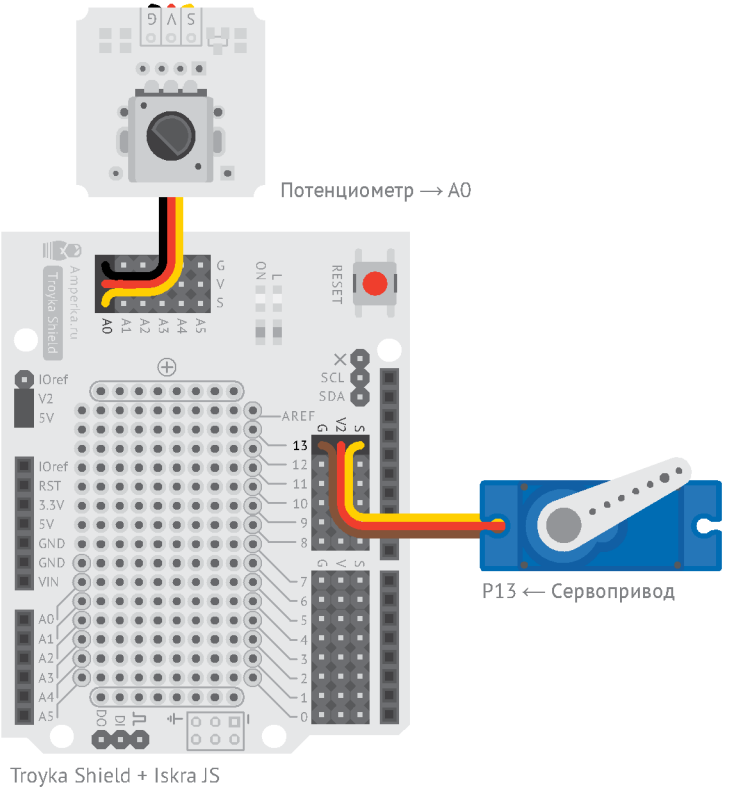
ПАНТОГРАФ

№10

Механикой самое время заняться. Манипулятор соберём, что ручки потенциометра движение повторяет.

Обрати внимание на шлейф сервомотора:
коричневый — земля (G),
красный — питание (V2),
оранжевый — сигнал (S).

Сервоприводу нужно 5 вольт, а не стандартные 3,3 вольта. Поэтому его нужно подключать к одному из пинов P8… P13 и при этом убедиться, что на них подаётся 5 вольт. Для этого джампер на Troyka Shield должен быть установлен в положение V2+5V.



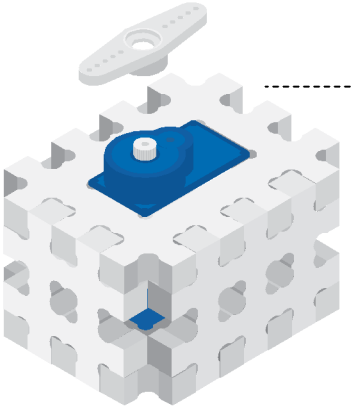
Troyka Shield + Iskra JS

```
var servo = require('@amperka/servo')
  .connect(P13);

var pot = require('@amperka/pot')
  .connect(A0);

setInterval(function() {
  var angle = 180 * pot.read();
  servo.write(angle);
}, 20);
```

1 `@amperka/servo` — библиотека для работы с распространёнными сервомоторами.

2 Рассчитываем угол поворота. Потенциометр выдаёт значение от 0 до 1, а сервопривод ожидает значение в диапазоне от 0 до 180°. Умножив значение с потенциометра на 180, мы получим нужный угол.

3 Используем метод `write` сервопривода для того, чтобы скомандовать ему повернуться к заданному углу. Метод `write` в качестве параметра принимает угол в градусах. Мы используем значение, которое вычислили строкой ранее.



Используй детали #структора, чтобы закрепить сервомотор и видеть поворот его вала.

ЗАДАНИЕ

Кнопкой потенциометр замени. Пусть нажатие её в 30° привод ставит, а отжатие — в 150°.

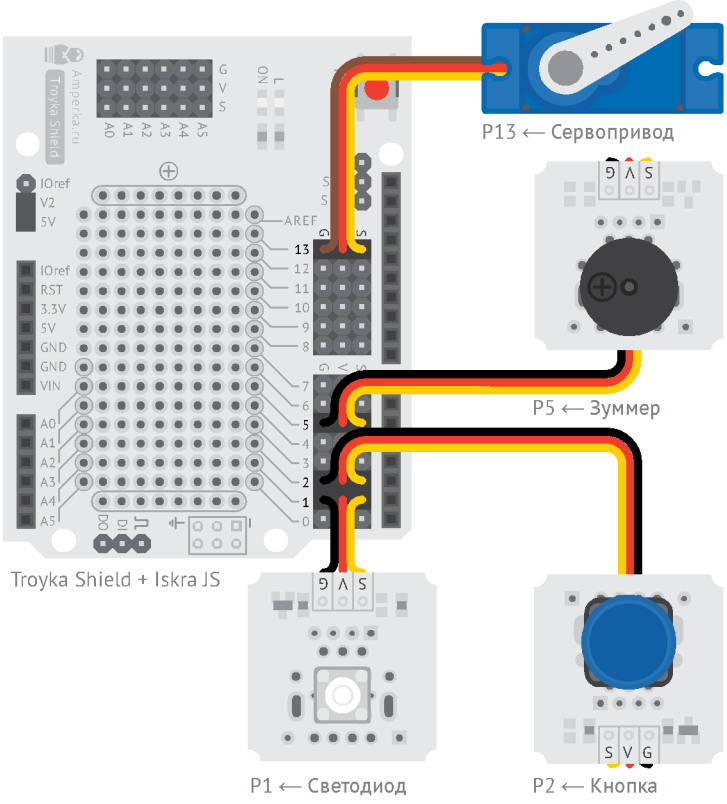
Подробнее о сервоприводе: js.amperka.ru → @amperka/servo

ПЕРЕЕЗД

№11

Кораблей движением управлять нужно нам. Палку-управлялку сделаем, что шлагбаумом дикари зовут, со звуком и светом для надёжности пущей.

Кнопку нажми, чтобы проезд закрыть. Снова нажми, чтобы открыть.



1 Создаём объект-пьезоизлучатель и сразу устанавливаем частоту звучания на `50` Гц. Потому что переезды жужжат, а не пищат.

2 Создаём объект-сервопривод и сразу устанавливаем угол `90°`, чтобы шлагбаум был открыт.

3 Заводим переменную `closed` и присваиваем ей значение «ложь». Мы будем хранить в ней текущее состояние переезда: истина — он закрыт, ложь — он открыт. Такие переменные, которые хранят одно из двух логических значений, называют *булевыми*.

Мы создали переменную на верхнем уровне, а не внутри функции, как делали ранее, чтобы её значение сохранялось на протяжении всей программы, а не перезаписывалось заново при вызове функции.

```
var trigger = require('@amperka/button')
  .connect(P2);

var buzzer = require('@amperka/buzzer')
  .connect(P5)
  .frequency(50);

var light = require('@amperka/led')
  .connect(P1);

var barrier = require('@amperka/servo')
  .connect(P13)
  .write(90);

var closed = false;

trigger.on('press', function() {
  closed = !closed;
  if (closed) {
    buzzer.beep(1, 0.5);
    light.blink(1, 0.5);
    barrier.write(0);
  } else {
    buzzer.turnOff();
    light.turnOff();
    barrier.write(90);
  }
});
```

4 Инвертируем значение `closed` оператором «не» (`!`). Если она была истина – станет ложью, если была ложью – станет истиной.

5 Записываем выражение `if`, где всем условием и является значение переменной `closed`. Условие сработает, если она истина, а ветка `else` – если ложна.

6 Переезд закрыт. Чтобы сигнализировать об этом, мы жужжим динамиком, мигаем светом и опускаем шлагбаум.

7 Переезд открыт. Выключаем динамик и свет, поднимаем шлагбаум.

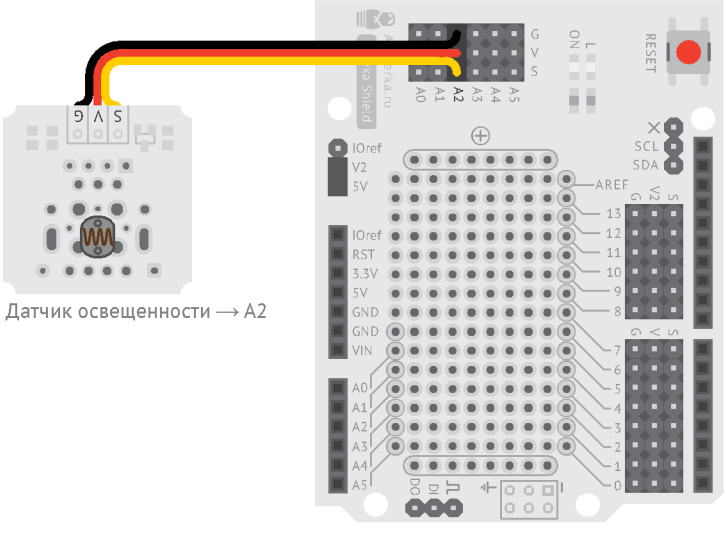
ЗАДАНИЕ

В проект потенциометр добавь. Пусть вращения сектор задаёт он, чтобы без пересборки палки-управлялки положение настроить можно было.

КОНСОЛЬНЫЙ ЛЮКСОМЕТР

№12

Устройство соберём, что на компьютер силу света посылает. За её величиной в виде числа сможем следить мы.



Датчик освещенности ⟶ A2

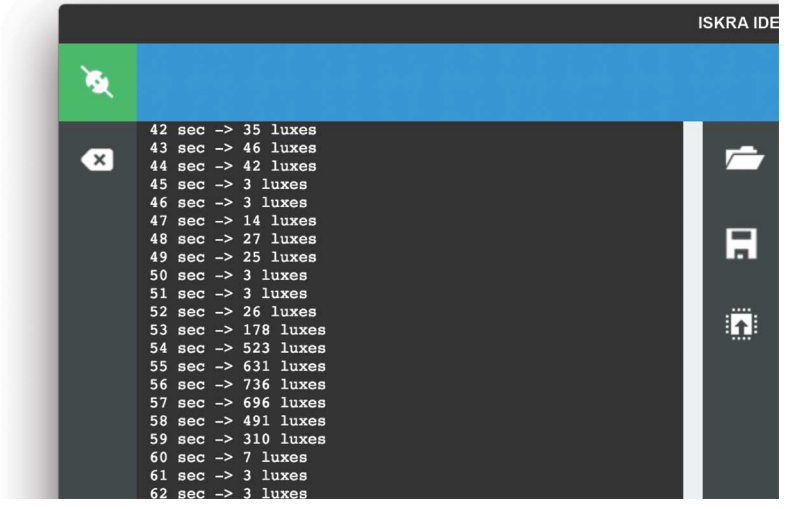
Troyka Shield + Iskra JS

1 Метод чисел `toFixed` округляет их до заданного знака после запятой. В нашем случае оставляем ноль знаков, т.е. отбрасываем дробную часть целиком.

2 Встроенная функция `getTime` возвращает количество секунд, прошедшее с момента включения или перезагрузки платы. Для вывода в консоль мы также округляем это значение до целого.

```
var sensor = require('@amperka/light-sensor')
  .connect(A2);

setInterval(function() {
  var lx = sensor.read('lx').toFixed(0);
  var time = getTime().toFixed(0);
  console.log(time, 'sec', '->',  lx, 'luxes');
}, 1000);
```

За результатами следи в левом окне консоли:



3 Встроенный объект `console` и его метод `log` нужны для того, чтобы передавать текстовые данные с платы на компьютер. Его используют, чтобы вести протокол происходящего или для отладки.

Метод `log` принимает неограниченное количество аргументов. Все они выводятся друг за другом разделённые пробелом. В конце строки `log` добавляет символ переноса строки, т.е. «печатает» *enter*.

ЗАДАНИЕ

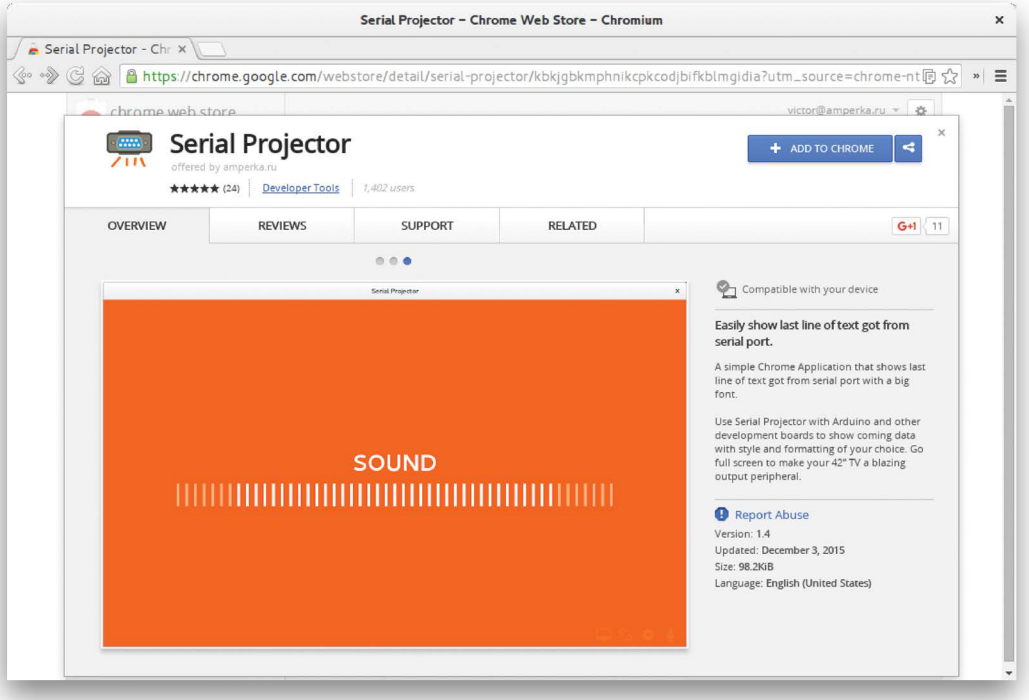
Программу измени так, чтобы точнее данные видел ты: после запятой с двумя знаками.

ЭКРАННЫЙ ЛЮКСОМЕТР

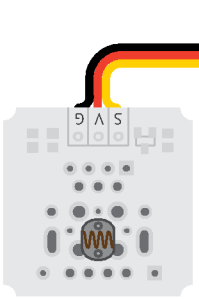
№13

Предыдущее устройство улучшим. Доступным родной язык станет нам. Издалека за показаниями следить возможно будет.

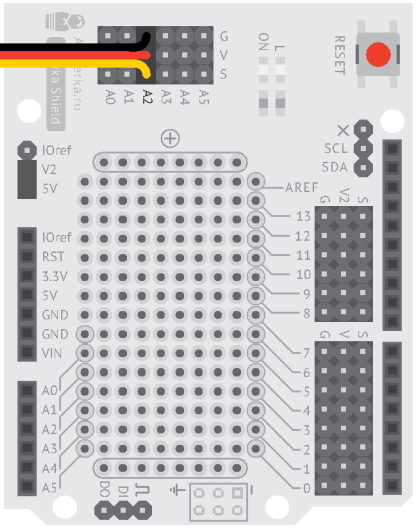
Установи из Chrome Web Store приложение Serial Projector. Набери название это в строке поиска Web Store или перейди по ссылке: amperka.ru/chrome/serial-projector.



Запусти приложение и наблюдай за своими данными.



Датчик освещенности ⟶ A2



Troyka Shield + Iskra JS

Одновременно Iskra JS может общаться только с одной программой. Чтобы Serial Projector подключился к плате, нужно сначала отсоединиться от неё из среды программирования кнопкой.

Используй кнопку, чтобы устанавливать и разрывать соединение в Serial Projector.

```
1  var sensor = require('@amperka/light-sensor')
2    .connect(A2);
3
4  setInterval(function() {
5    var lx = sensor.read('lx').toFixed(0);
6    console.log(lx, 'люкс'); ┆
7  }, 200);
```

1 При выводе в Serial Projector можно использовать кириллицу и другие *unicode-символы*: греческие буквы, пиктограммы и т.п.

ЗАДАНИЕ

С цветом фона и шрифтом эксперимент проведи.
В меню по кнопке настройки эти смени.

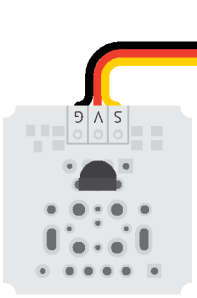
На весь экран кнопкой окно распахни.

HTML-ТЕРМОМЕТР

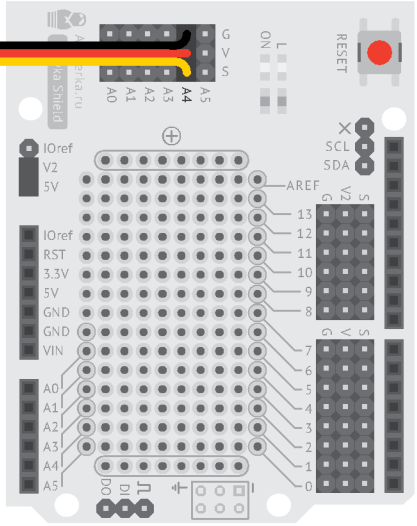
№14

Великий термометр сделаем, что символами крупными температуру выводит.

Serial Projector знакомый используй, чтобы результат наблюдать.



Термометр ⟶ A4



Troyka Shield + Iskra JS

1 `@amperka/thermometer` — модуль для работы с линейным аналоговым термометром.

2 Метод `read` термометра возвращает температуру в указанных единицах. Нам интересны градусы Цельсия, обозначаемые латинской `C`.

```
var thermometer = require('@amperka/thermometer')
  .connect(A4);

setInterval(function() {
  var celsius = thermometer.read('C');
  console.log(
    '<div style="font-size: 0.5em">',
      'Температура',
    '</div>',
    celsius.toFixed(1),
    '°C'
  );
}, 1000);
```

Serial Projector так будет выглядеть твой:

3 Для `console.log` всё равно какой текст мы выводим: простой или с каким-то форматированием. Зато некоторые программы могут по-своему отображать разметку. Так, например, Serial Projector умеет отображать HTML. Благодаря этому мы можем показывать свои данные красиво.

Мы используем тег `div` со встроенным стилем `font-size`, чтобы сделать подпись «`Температура`» на отдельной строке и полуразмерным шрифтом. Чтобы узнать все возможности форматирования, обратись к справочникам по HTML и CSS.

4 Встраиваем в вывод полученное значение температуры с точностью до одного знака после запятой.

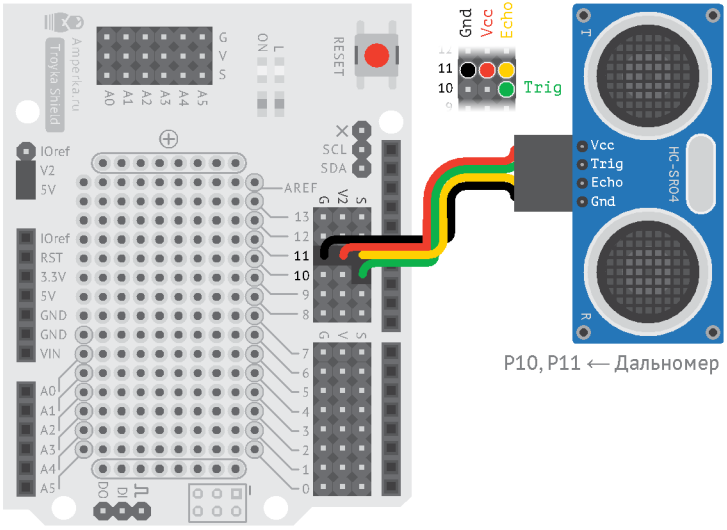
ЗАДАНИЕ

Чтобы единицы измерения менять, кнопку добавь.
В провинциях глубоких градусы Кельвина и Фаренгейта нужны.

УЛЬТРАЗВУКОВАЯ ЛИНЕЙКА

№15

Расстояние измеряющий прибор сделаем мы. В Serial Projector смотри, чтобы точную дистанцию знать.



P10, P11 ⟵ Дальномер

Troyka Shield + Iskra JS

Обрати внимание, что дальномеру для работы нужно 5 вольт. Поэтому подключать его стоит к пинам группы P8…P13 и джампером на Troyka Shield в положении V2+5V.

1 Используем модуль `@amperka/ultrasonic` для работы с ультразвуковым дальномером.

Метод `connect` принимает объект, в котором должны быть определены два пина: `trigPin` и `echoPin`, которые соответствуют одноимённым пинам на плате дальномера.

```
var sonic = require('@amperka/ultrasonic')
  .connect({trigPin: P10, echoPin: P11});

setInterval(function() {
  sonic.ping(function(err, val) {
    if (err) {
      console.log(err.msg);
    } else {
      console.log(val.toFixed(0), 'мм');
    }
  }, 'mm');
}, 100);
```

2 Чтобы замерить расстояние, ультразвуковой дальномер должен сгенерировать звуковой пучок, перейти в режим ожидания и дождаться отражённый сигнал. Это может занять до нескольких миллисекунд. А ведь за это время можно сделать много всего полезного! Поэтому метод чтения дальномера отличается от других сенсоров.

Метод `ping` запускает звуковой пучок и принимает 2 параметра:
- функцию результата, которую нужно вызвать при возврате эха, т.е. когда расстояние измерено;
- единицы измерения результата.

В функцию результата дальномер передаст 2 параметра:
- `err` — ошибка, если она произошла при измерении;
- `val` — измеренное расстояние, если ошибки не было.

3 Мы проверяем, была ли ошибка.

Если она произошла, условие сработает. А в свойстве `msg` объекта-ошибки будет содержаться разъяснение. Например:
- `timeout` — не дождались эха: препятствие либо дальше 4 метров, либо ближе 2 сантиметров;
- `busy` — предыдущее измерение ещё не завершено;
- `wrong connection` — сенсор не подаёт сигналов, скорее всего он подключен не верно.

4 Запрашиваем результат в миллиметрах.

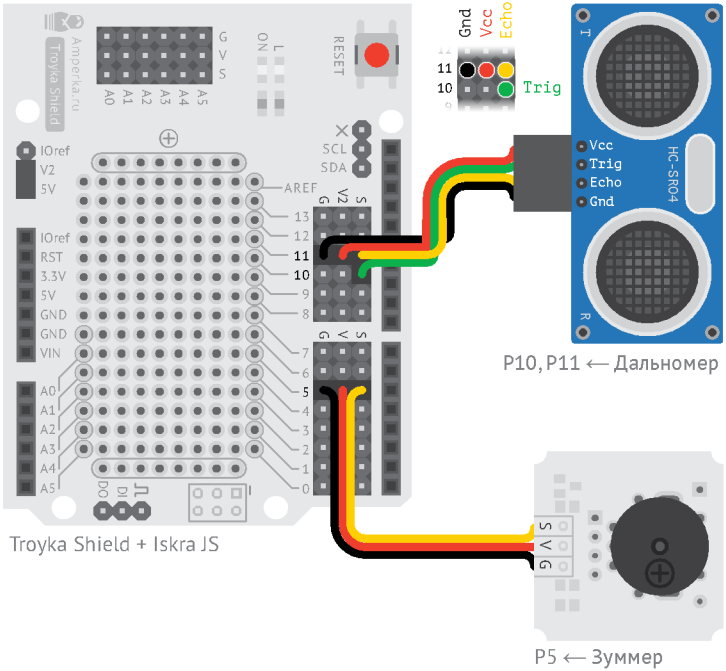
ЗАДАНИЕ

В проект кнопку добавь. На экране до следующего клика показание нажатие её пусть заморозит.

ПАРКТРОНИК

№16

Для манёвров точных парктроник сделай. Непрерывный звук будет, когда препятствие близко. Прерывистый сигнал издаст он, если ещё до столкновения время есть.



Troyka Shield + Iskra JS

1 Расстояние меньше 5 см? Мы в опасной близости от препятствия. Включаем непрерывный сигнал.

```
var sonic = require('@amperka/ultrasonic')
  .connect({trigPin: P10, echoPin: P11});

var buzzer = require('@amperka/buzzer')
  .connect(P5)
  .frequency(440);

setInterval(function() {
  sonic.ping(function(err, val) {
    if (val < 5) {
      buzzer.turnOn();
    } else if (val < 20) {
      buzzer.beep(0.1, 0.1);
    } else if (val < 50) {
      buzzer.beep(0.2, 0.2);
    } else {
      buzzer.turnOff();
    }
  }, 'cm');
}, 100);
```

3 Расстояние 5–20 см — пищим часто.

4 Расстояние 20–50 см — пищим редко.

5 Расстояние более 50 см или не может быть измерено — молчим.

2 Обрати внимание, как можно сцеплять выражения `if / else`, чтобы сделать последовательную проверку условий.

ЗАДАНИЕ

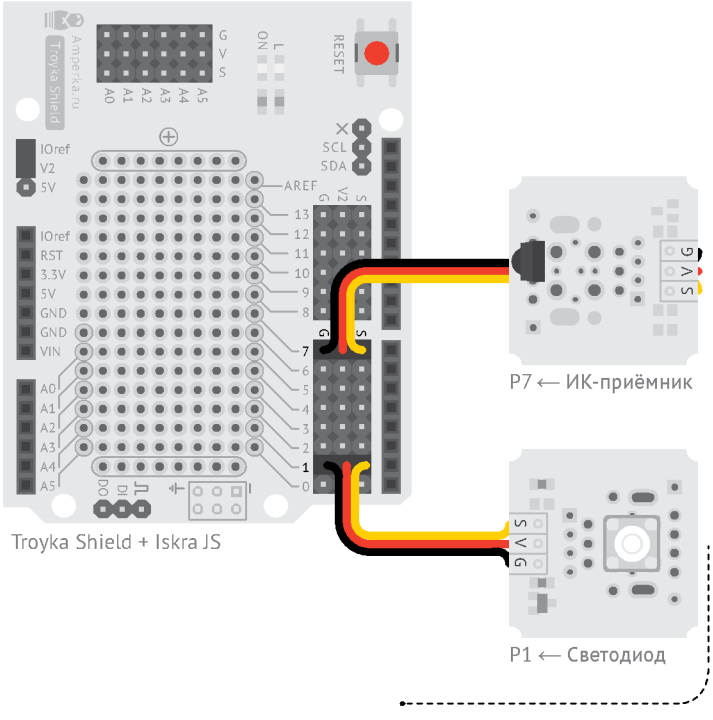
Потенциометром девайс снабди. Пусть расстояние срабатывания настроить можно будет им.

СКАНЕР ИК-ПУЛЬТОВ

№17

Пульта инфракрасный свет научу видеть. Код кнопки нажатой в консоль устройство выведет. Любой пульт подойдёт ему. Приём сигнала светодиод покажет.

[1] `@amperka/ir-receiver` — модуль для работы с инфракрасными приёмниками.



Troyka Shield + Iskra JS

При приёме сигнала светодиод мигает.

Пульт

```
var ir = require('@amperka/ir-receiver')
  .connect(P7);

var light = require('@amperka/led')
  .connect(P1);

ir.on('receive', function(code, repeat) {
  if (!repeat) {
    console.log('********');
  }

  console.log('0x' + code.toString(16));
  light.toggle();
});
```

3 Если мы видим, что кнопка нажата только что, выведем в консоль какой-нибудь маркер для наглядности. Например, несколько звёздочек.

4 Выводим полученный код в консоль. `toString(16)` преобразует целое число в строку с его представлением в шестнадцатеричной системе счисления. Когда речь идёт о кодах, идентификаторах, протоколах, принято использовать шестнадцатеричные числа.

2 Событие `receive` наступает всякий раз, когда приёмник фиксирует пришедший с пульта код нажатия кнопки.

В функцию-обработчик события приёмник передаёт 2 параметра:

- `code` — числовой код нажатой на пульте кнопки;
- `repeat` — если истина, кнопка зажата и мы поймали повторную отправку её кода; если ложь — кнопка нажата только что.

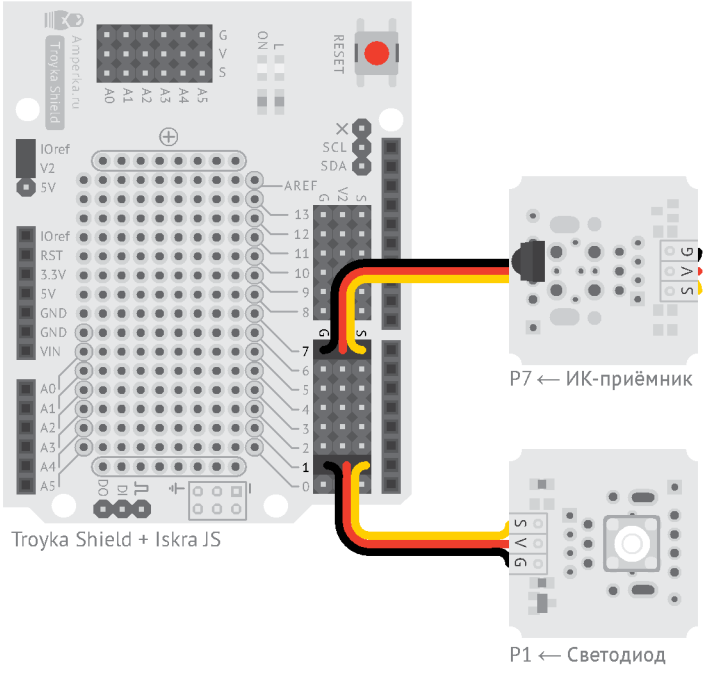
ЗАДАНИЕ

Программу улучши, чтобы всегда светодиод гас после того, как на пульте кнопку отпустили.

ИК-ВЫКЛЮЧАТЕЛЬ СВЕТА

№18

Светом с пульта управлять будем. Кнопкой, которая свет переключает, та станет, что первой после включения платы нажмёшь.



Troyka Shield + Iskra JS

1 Создаём глобальную переменную, в которой будем хранить код кнопки для включения света. Мы хотим, чтобы изначально устройство было не запрограммированно, поэтому `powerCode` выставляем в нулевое значение.

Значение `null` — специальная встроенная константа JavaScript, которая означает «нет значения» или «значение очищено». `null` и число `0` — разные вещи. Они не эквивалентны друг другу.

```
var ir = require('@amperka/ir-receiver')
  .connect(P7);

var light = require('@amperka/led')
  .connect(P1);

var powerCode = null;

ir.on('receive', function(code, repeat) {
  if (repeat) {
    return;
  }

  if (powerCode === null) {
    powerCode = code;
  }

  if (code === powerCode) {
    light.toggle();
  }
});
```

2 Если мы получили повторную отправку кода, она нам не интересна: ведь мы переключаем свет только один раз на нажатие кнопки.

3 Если `powerCode` ещё не назначен, мы находимся в режиме программирования устройства. В качестве кода-переключателя назначим код, который пришёл на приёмник. Следующий раз условие уже не сработает, поэтому устройство запомнит назначенную кнопку до перезагрузки.

4 Если была нажата та кнопка, которую мы назначили на переключение, переключаем светодиод.

ЗАДАНИЕ

Программу измени, чтобы кнопка одна всегда свет включала, а другая — выключала.

ПУЛЬТ КИНОМАНА

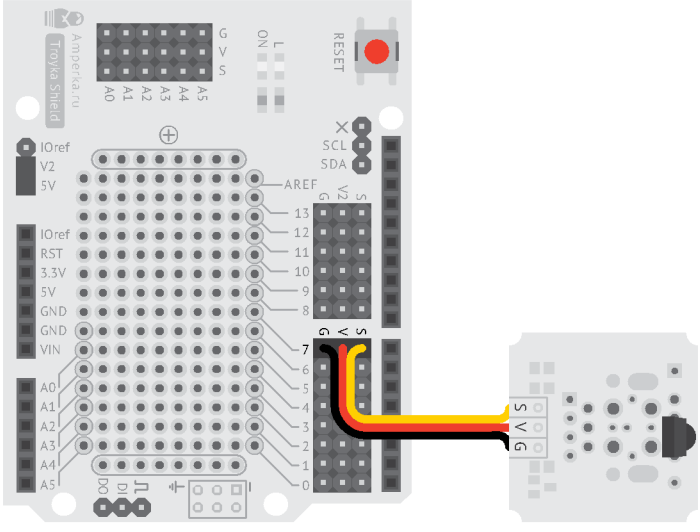
№19

USB-устройство построим, которое клавиатурой притворяется. Видеоплеером VLC с инфракрасного пульта сможешь управлять ты.

После загрузки и сохранения программы в флеш-память Iskra JS нужно отключить её от USB и подключить снова. Это необходимо, чтобы операционная система признала в подключенном устройстве клавиатуру.

1 Подключаем библиотеку `@amperka/usb-keyboard`, которая позволяет действовать, как компьютерная клавиатура.

2 Определяем в виде констант коды клавиш для перемотки и проигрывания. Используй проект №17«Сканер ИК-пультов» для того, чтобы определить свои коды.



Troyka Shield + Iskra JS

P7 ← ИК-приёмник

Пульт

```
var ir = require('@amperka/ir-receiver')
  .connect(P7);

var kb = require('@amperka/usb-keyboard');

var rewindCode = 0xfd20df;
var forwardCode = 0xfd609f;
var playCode = 0xfda05f;

ir.on('receive', function(code, repeat) {
  if (code === playCode) {
    if (!repeat) {
      kb.tap(kb.KEY.SPACE);
    }
  } else if (code === rewindCode) {
    kb.tap([kb.MODIFY.CTRL, kb.KEY.LEFT]);
  } else if (code === forwardCode) {
    kb.tap([kb.MODIFY.CTRL, kb.KEY.RIGHT]);
  }
});
```

3 Метод `tap` клавиатуры эквивалентен короткому клацанию по клавише. Код клавиши метод принимает в качестве параметра. Коды клавиш собраны под понятными именами в объекте KEY модуля-клавиатуры.

Если мы поймали нажатие кнопки паузы/проигрывания и это первое её нажатие, нажимаем пробел. Пробел в большинстве видео-плееров ставит видео на паузу или продолжает проигрывание. Это то, что нам нужно.

4 Если мы поймали сигнал кнопки перемотки, нажимаем контрол + стрелку влево. Нажимаем стрелку даже, если поймали повторный код. Это заставит срабатывать перемотку в видео-плеере снова и снова пока зажата клавиша на пульте.

Нажатие кнопок в комбинации с модификаторами Ctrl, Alt, Shift можно осуществлять, если вызывать `tap` с массивом в качестве первого параметра. В масиве ожидается перечень модификаторов, а затем сама клавиша.

5 Аналогично, при нажатии кнопки промотки вперёд, эмулируем нажатие клавиш контрол + вправо на клавиатуре.

ЗАДАНИЕ

Для Windows Media Player, QuickTime Player или Power Point устройство адаптируй.

Для Window Media Player:
Пауза/проигрыш — Ctrl+P
Перемотка назад — Ctrl+Shift+b
Перемотка вперёд — Ctrl+Shift+f

Для QuickTime Player:
Пауза/проигрыш — пробел
Перемотка назад — стрелка влево
Перемотка вперёд — стрелка вправо

Для Power Point:
Запуск презентации — F5
Следующий слайд — Page Down
Предыдущий слайд — Page Up

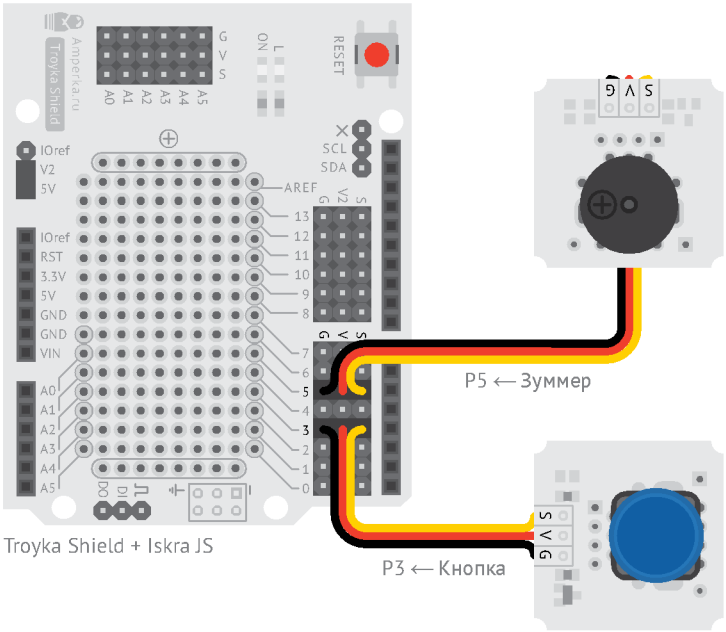
Подробнее об эмуляции клавиатуры: js.amperka.ru → @amperka/usb-keyboard

ГЕНЕРАТОР ПАРОЛЕЙ

№20

Устройство сделаем, что из 16 символов пароли совершенно случайные придумывает.

Кнопку надолго зажми, чтобы новый пароль задумать.
Кратко кнопку нажми, чтоб у курсора пароль текущий ввести.



Troyka Shield + Iskra JS

1 Задаём опцию кнопки `holdTime` в полсекунды, чтобы далее использовать события `click` и `hold`, которые означают короткое и длинное нажатие.

2 Подключаем библиотеку для генерации случайных чисел.

```
var button = require('@amperka/button')
  .connect(P3, {holdTime: 0.5});

var buzzer = require('@amperka/buzzer')
  .connect(P5);

var kb = require('@amperka/usb-keyboard');
var random = require('@amperka/hw-random');
var password = '';

function generatePassword() {
  password = '';
  while (password.length < 16) {
    var code = random.int(33, 126);
    password += String.fromCharCode(code);
  }
}

button.on('hold', function() {
  generatePassword();
  console.log(password);
  buzzer.beep(0.1);
});

button.on('click', function () {
  kb.type(password);
});

generatePassword();
```

3 Заводим переменную, в которой будем хранить текущий пароль.

4 Создаём функцию, которая при вызове придумывает новый пароль.

5 Первым делам очищаем предыдущий пароль, если он был.

6 Пока пароль не достиг 16 символов генерируем случайное число от 33 до 126, переводим этот код в символ по ASCII-таблице и приклеиваем новый символ в конец пароля.

7 При долгом нажатии на кнопку генерируем новый пароль, выводим его в консоль и подаём короткий звуковой сигнал.

8 При коротком нажатии на кнопку «печатаем» текущий пароль на клавиатуре.

Метод `type` объекта-клавиатуры посылает через USB сигналы нажатия кнопок так, будто на клавиатуре печатает человек.

9 При старте платы первым делом генерируем новый пароль.

ЗАДАНИЕ

Потенциометр добавь, чтобы длину пароля любую задать смог ты.

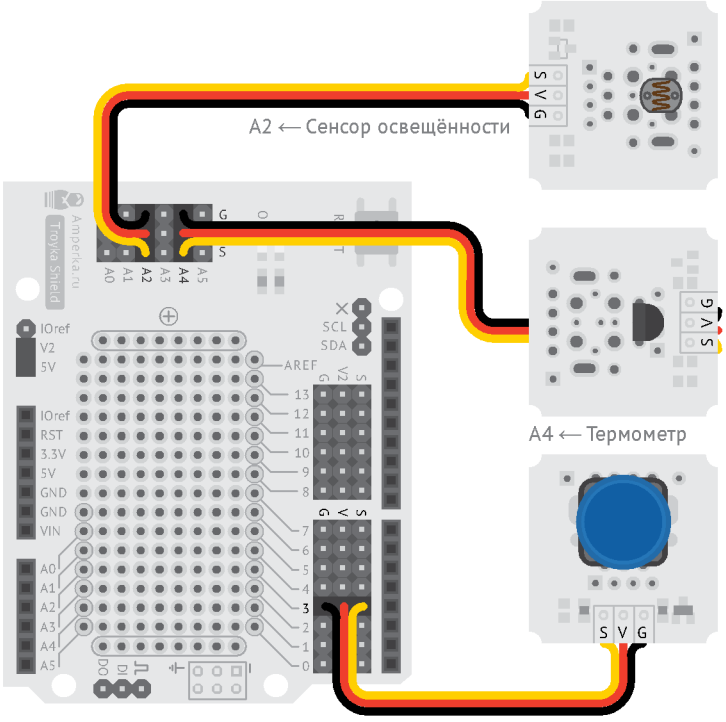
EXCEL-РОБОТ

№21

Клавиатурного робота напишем, который каждые 5 секунд в новую строку Excel освещённость и температуру вбивает.

Запись чтоб начать, Excel запусти и в ячейку A2 курсор поставь. Чтобы запись прекратить, ещё раз кнопку нажми.

На ночь робота работать оставь. По записанным данным график построй. Изменение данных в динамике увидишь ты.



1 Создаём объект из библиотеки `@amperka/timer`. Таймер — это ещё один способ сделать действие, которое повторяется регулярно. Таймер — более мощный аналог `setInterval`. Его можно ставить на паузу, сбрасывать, запускать, подстраивать.

Метод `create` создаёт новый объект-таймер, а в качестве аргумента он принимает интервал в секундах. Мы задали интервал 5 секунд.

2 Метод `isRunning` возвращает истину, если таймер запущен.

```
var lightSensor = require('@amperka/light-sensor')
  .connect(A2);

var thermometer = require('@amperka/thermometer')
  .connect(A4);

var button = require('@amperka/button')
  .connect(P3);

var kb = require('@amperka/usb-keyboard');

var timer = require('@amperka/timer')
  .create(5);

button.on('press', function() {
  if (timer.isRunning()) {
    timer.stop();
  } else {
    timer.tick().run();
  }
});

timer.on('tick', function() {
  var time = getTime();
  var lx = lightSensor.read('lx');
  var c = thermometer.read('C');

  kb.type(time.toFixed(0) + '\t' +
          lx.toFixed(0) + '\t' +
          c.toFixed(0) + '\n');
});
```

4 Метод `tick` приводит к сиюсекундному срабатыванию таймера, а метод `run` — к запуску. Если таймер не был запущен, этой строкой мы заставляем тикнуть его прямо в момент нажатия кнопки, а затем размеренно срабатывать каждые 5 секунд.

5 Событие `tick` происходит всякий раз, когда таймер срабатывает. То есть один раз в установленный интервал.

6 Символ `\t` — это специальное обозначения клавиши «TAB». А символ `\n` — обозначение «Enter».

Мы печатаем время, TAB, освещённость, TAB, температуру, Enter. После такого ввода мы получим заполненную строку и перейдём в начало следующей в любой программе электронных таблиц: MS Excel, LibreOffice Calc или Google Spreadsheet.

3 Метод `stop` останавливает таймер. Если таймер запущен, остановим его.

ЗАДАНИЕ

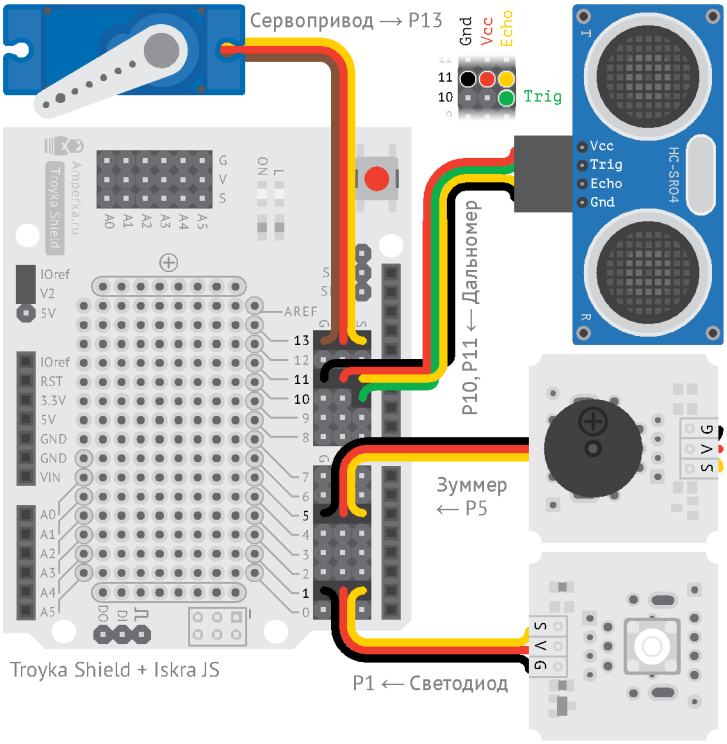
С дальномером устройство сделай, которое в Excel время записывает, когда в дверной проём джедай проходит.

УМНЫЙ ШЛАГБАУМ

№22

Из проекта 11 переезд улучшим, самостоятельным сделаем его.

Сам переезд закроется, если препятствие увидит. Обратно откроется он, как только на 4 секунды пустоту увидит.



1 Создаём объект-гистерезис. *Гистерезис* — это фильтр для сигнала: у него есть аналоговый вход и дискретный выход. Он сравнивает вход с заданными высоким и низким пороговыми значениями и генерирует события `low` или `high` при пересечении входным сигналом этих пороговых значений.

При этом события генерируются не сразу, а через некоторое время, которое называется *лагом*. Лаги используют, чтобы убедиться в том, что входной сигнал устаканился с той или иной стороны от пороговых значений.

В нашем случае пороговые значения — `high` и `low` равны `0.5` метрам, а лаг на превышение высокого порога — 4 секунды.

2 Гистерезису нужны входные данные. Метод `push` добавляет очередное значение, которое рассматривается им как входной сигнал. Мы добавляем новое значение каждые `100` мс.

3 Событие `low` произойдёт как только входной сигнал станет ниже `0.5`. Это означает, что дальномер увидел препятствие.

```
var sonic = require('@amperka/ultrasonic')
  .connect({trigPin: P10, echoPin: P11});

var buzzer = require('@amperka/buzzer')
  .connect(P5)
  .frequency(50);

var light = require('@amperka/led')
  .connect(P1);

var barrier = require('@amperka/servo')
  .connect(P13)
  .write(90);

var hysteresis = require('@amperka/hysteresis')
  .create({high: 0.5, highLag: 4, low: 0.5, lowLag: 0});

setInterval(function() {
  sonic.ping(function(err, val) {
    if (err) return;
    hysteresis.push(val);
  }, 'm');
}, 100);

hysteresis.on('low', function(val) {
  buzzer.beep(1, 0.5);
  light.blink(1, 0.5);
  barrier.write(0);
});

hysteresis.on('high', function(val) {
  buzzer.turnOff();
  light.turnOff();
  barrier.write(90);
});
```

4 При приближении поезда мы закрываем шлагбаум, подаём звуковой и световой сигналы.

5 Событие `high` произойдёт спустя 4 секунды после того, как входной сигнал стал выше `0.5`, и только если за это время он не падал ниже `0.5`. Это означает, что дальномер уже 4 секунды не видел препятствий.

6 Препятствий нет — открываем шлагбаум, выключаем свет и звук.

ЗАДАНИЕ

Умного освещения систему сделай, которая свет выключает-включает лишь после того, как на 10 секунд окружающая освещённость изменится.

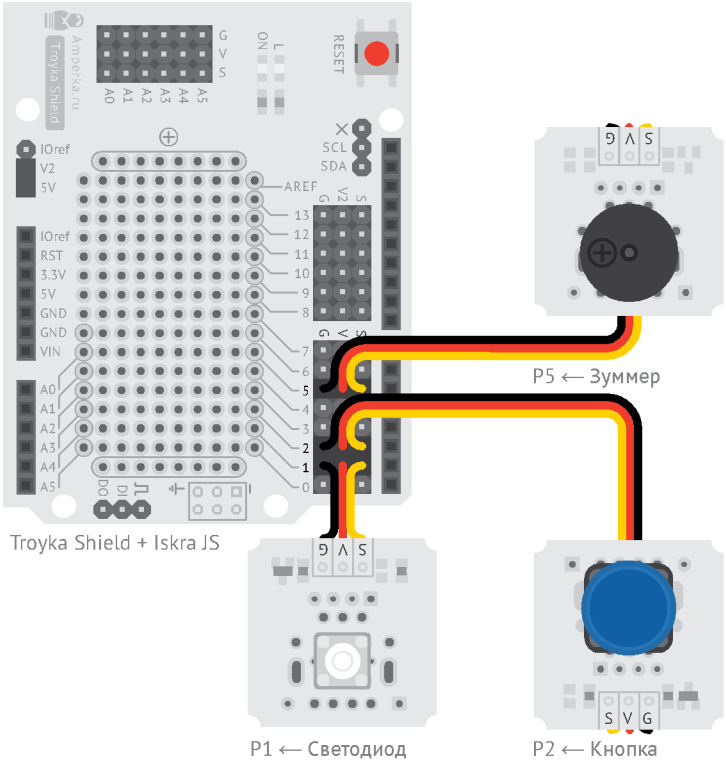
Подробнее о гистерезисе: js.amperka.ru → @amperka/hysteresis

ТРЕВОЖНАЯ КНОПКА

№23

Тёмные силы отпугнуть чтобы, звонкую сирену собери.

Кнопку нажми, чтобы тревогу включить. Ещё раз нажми, чтобы прекратить.



Troyka Shield + Iskra JS

1 Создаём объект-анимацию. Анимация помогает плавно изменять заданные значения.

В качестве настроек мы указываем несколько параметров:
- `from` равный нулю — начальное значение анимации;
- `to` равный единице — конечное значение;
- `loop` равный `true`, что заставит анимацию проигрываться заново, когда она закончится;
- `updateInterval` равный `0.01`, что заставит обновлять значения каждые 0,01 секунды (10 миллисекунд).

```
var button = require('@amperka/button')
  .connect(P2);

var buzzer = require('@amperka/buzzer')
  .connect(P5);

var light = require('@amperka/led')
  .connect(P1);

var animation = require('@amperka/animation')
  .create({
    from: 0,
    to: 1,
    loop: true,
    updateInterval: 0.01
  });

var armed = false;

animation.on('update', function(val) {
  light.brightness(val);
  buzzer.frequency(1000 + 4000 * val);
});

button.on('press', function() {
  armed = !armed;
  buzzer.toggle(armed);
  light.toggle(armed);
  if (armed) {
    animation.play();
  } else {
    animation.stop();
  }
});
```

2 Задаём реакцию на изменение значения анимации. Раз в `updateInterval` анимация генерирует событие `update` и передаёт в функцию обработчик текущее значение `val`.

3 Используем `val` для того, чтобы выставить яркость лампы и тон зуммера.

4 При нажатии кнопки, вызываем метод `play` анимации. Он запустит процесс обновления значений.

5 При повторном нажатии кнопки останавливаем процесс при помощи метода `stop`.

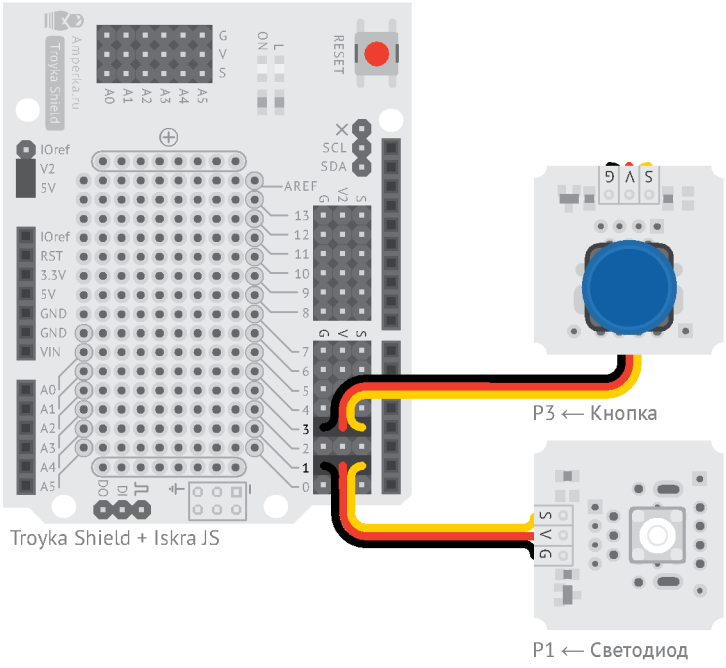
ЗАДАНИЕ

Ультразвуковым дальномером кнопку замени, чтобы сама тревога включилась, когда тёмную силу увидит.

ТЕАТРАЛЬНЫЙ СВЕТ

№24

Для представлений свет сделай, который плавно гаснет и мягко нарастает.



Troyka Shield + Iskra JS

1 Наш светодиод мы включаем, но тут же устанавливаем его яркость в ноль. На старте нам нужен свет именно в таком состоянии: погашенный, но готовый к работе.

2 Создаём анимацию без параметров. По умолчанию анимация длится 1 секунду, продвигает значение от 0 до 1 с интервалом в 0,01 секунды и не повторяется после завершения. Это нам подходит.

```
var light = require('@amperka/led')
  .connect(P1)
  .turnOn()
  .brightness(0);

var button = require('@amperka/button')
  .connect(P3);

var anim = require('@amperka/animation')
  .create()
  .reverse();

anim.on('update', function(val) {
  light.brightness(val);
});

button.on('press', function() {
  anim.reverse().play();
});
```

3 Тут же инвертируем направление проигрывания. Это нужно для того, чтобы повторное инвертирование, которое произойдёт далее, при нажатии кнопки, в самый первый раз привело к проигрыванию в прямом направлении. Минус на минус даёт плюс.

4 В качестве реакции на изменение значения анимации устанавливаем яркость света.

5 При нажатии кнопки меняем направление анимации на противоположное и запускаем проигрыш.

ЗАДАНИЕ

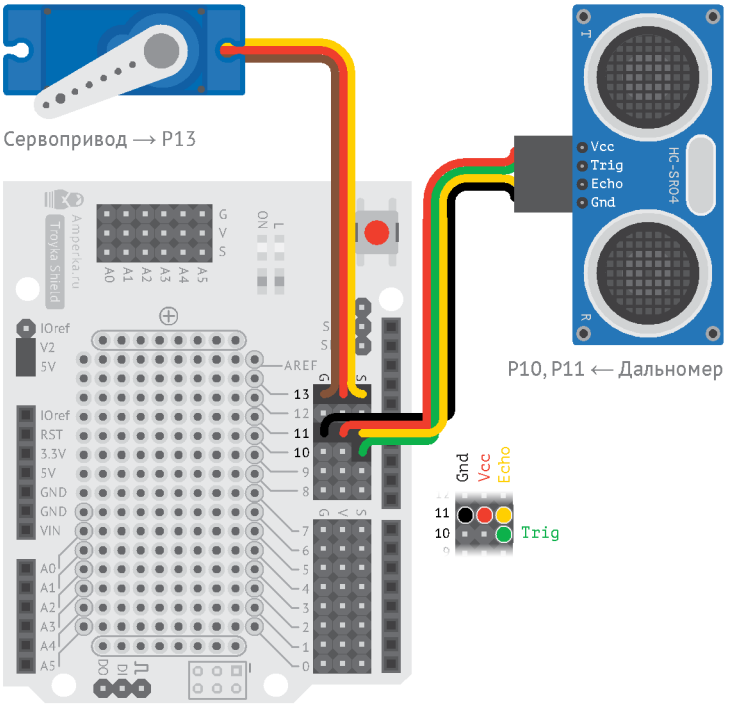
Для ленивых устройство адаптируй. С ИК-пульта возможность светом управлять дай им.

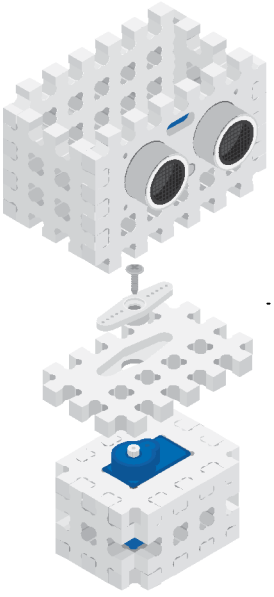
НАСТОЛЬНЫЙ РАДАР

№25

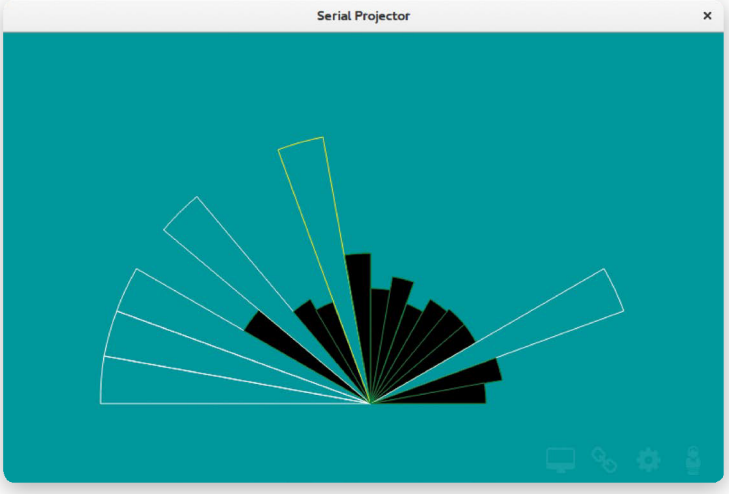
Робота голову соберём. Дальномер крутиться будет, чтоб перед собой пространство понять.

В виде круговой диаграммы через Serial Projector работу наблюдай.





Используй детали #структора чтобы соединить сервопривод и дальномер. Подробнее на стр 17.



Serial Projector так будет выглядеть твой:

```
var ultrasonic = require('@amperka/ultrasonic')
  .connect({trigPin: P10, echoPin: P11});

var servo = require('@amperka/servo')
  .connect(P13);

var canvas = {
  width: 800,
  height: 500,
  radius: 300,
  margin: 150
};

var sectors = {
  count: 18,
  current: 0,
  direction: 1
};

sectors.values = new Array(sectors.count);


function dumpSvg() {
  var svg = [];
  svg.push('<svg width="' + canvas.width + 'px' +
           '" height="' + canvas.height + 'px');
  svg.push('" xmlns="http://www.w3.org/2000/svg">');

  var cx = canvas.width / 2;
  var cy = canvas.height - canvas.margin;
  var astep = Math.PI / sectors.count;

  for (var i = 0; i < sectors.count; ++i) {
    var fill = 'black';
    var stroke = 'green';
    var r = sectors.values[i];

    if (!r || r > canvas.radius) {
      fill = 'none';
      stroke = 'white';
      r = canvas.radius;
    }

    if (i === sectors.current) {
      stroke = 'yellow';
    }

    var a1 = astep * i - Math.PI / 2;
    var a2 = a1 + astep;
```

1 Заводим объект с настройками радара, которые не меняются по ходу исполнения программы.

2 Заводим объект, где будем хранить текущее состояния с данными, которые получаем по ходу сканирования пространства.

3 Создаём функцию, которая переводит содержимое `sectors` в картинку в формате SVG и передаёт её на компьютер, в Serial Projector.

```
    var x1 = cx + r * Math.sin(a1);
    var y1 = cy - r * Math.cos(a1);
    var x2 = cx + r * Math.sin(a2);
    var y2 = cy - r * Math.cos(a2);

    x1 = x1.toFixed(0);
    y1 = y1.toFixed(0);
    x2 = x2.toFixed(0);
    y2 = y2.toFixed(0);

    svg.push('<path d="');
    svg.push('M' + cx + ' ' + cy + ' ');
    svg.push('L' + x1 + ' ' + y1 + ' ');
    svg.push('A' + r  + ' ' + r  + ' 0, 0, 1, ' +
             x2 + ' ' + y2 + ' ');
    svg.push('Z');
    svg.push('" stroke="' + stroke +
             '" fill="' + fill + '" />');
  }

  svg.push('</svg>');

  console.log(svg.join(''));
}

setInterval(function() {
  ultrasonic.ping(function(err, val) {
    sectors.values[sectors.current] = val;
    sectors.current += sectors.direction;
    if (sectors.current === sectors.count - 1 ||
        sectors.current === 0) {
      sectors.direction = -sectors.direction;
    }

    servo.write(sectors.current *
                (180 / sectors.count));
    dumpSvg();
  }, 'mm');
}, 300);
```

4 Заставляем сервопривод вращаться туда-сюда и делать замер на каждом секторе. В конце каждого измерения вызываем `dumpSvg` для отправки диаграммы на компьютер.

Полностью если код не понял, не беспокойся. Занятия и тренировки нужны, чтобы джедаем настоящим стать. На js.amperka.ru совершенствование продолжи своё.

СПРАВОЧНИК ПО ОБЪЕКТАМ

СВЕТОДИОД

```
var led = require('@amperka/led').connect(P1)
```

- `led.turnOn()` — включить
- `led.turnOff()` — выключить
- `led.toggle()` — если выключен — включить, если включен — выключить
- `led.blink(0.2, 0.8)` — мигать: 0,2 секунды гореть, 0,8 секунды не гореть
- `led.blink(0.2)` — мигнуть 1 раз в течение 0,2 секунд
- `led.brightness(0.42)` — установить яркость в 42%

КНОПКА

```
var button = require('@amperka/button').connect(P3)
```

- `button.on('press', function() { … })` — вызвать функцию при нажатии
- `button.on('release', function() { … })` — вызвать функцию при отжатии
- `button.on('click', function() { … })` — вызвать функцию при коротком клике
- `button.on('hold', function() { … })` — вызвать функцию при длительном нажатии

ЗУММЕР

```
var buzzer = require('@amperka/buzzer').connect(P5)
```

- `buzzer.turnOn()` — включить
- `buzzer.turnOff()` — выключить
- `buzzer.toggle()` — если выключен — включить, если включен — выключить
- `buzzer.beep(0.2, 0.8)` — прерывисто пищать: 0,2 секунды пищать, 0,8 секунды молчать
- `buzzer.beep(0.2)` — пискнуть 1 раз в течение 0,2 секунд
- `buzzer.frequency(1234)` — звучать на частоте 1234 герц

ПОТЕНЦИОМЕТР

```
var pot = require('@amperka/pot').connect(A0);
```

- `pot.read()` — считать значение от 0.0 до 1.0

СЕНСОР ОСВЕЩЁННОСТИ

```
var sensor = require('@amperka/light-sensor').connect(A2);
```

- `sensor.read('lx')` — считать значение в люксах

СЕРВОПРИВОД

```
var servo = require('@amperka/servo').connect(P13);
```

• `servo.write(42)` — повернуть в положение 42°

ТЕРМОМЕТР

```
var thermometer = require('@amperka/thermometer').connect(A4);
```

• `thermometer.read('C')` — считать значение в градусах Цельсия

УЛЬТРАЗВУКОВОЙ ДАЛЬНОМЕР

```
var sonic = require('@amperka/ultrasonic')
  .connect({trigPin: P10, echoPin: P11});
```

• `sonic.ping(function(error, distance) { … }, 'cm')` — измерить значение в сантиметрах и вызвать функцию с результатом

ИК-ПРИЁМНИК

```
var ir = require('@amperka/ir-receiver').connect(P7);
```

• `ir.on('receive', function(code, repeat) { … })` — вызвать функцию при нажатии кнопки на ИК-пульте

ЭМУЛЯЦИЯ КЛАВИАТУРЫ

```
var kb = require('@amperka/usb-keyboard');
```

• `kb.tap(kb.KEY.SPACE)` — нажать пробел
• `kb.tap([kb.MODIFY.ALT, kb.KEY.SPACE])` — нажать альт+пробел
• `kb.type('Hello World!')` — набрать «Hello World!»

СЛУЧАЙНЫЕ ЧИСЛА

```
var random = require('@amperka/hw-random');
```

• `random.int(1, 6)` — случайное число: 1, 2, 3, 4, 5 или 6

ВСТРОЕННЫЕ ФУНКЦИИ

• `console.log('Hello!')` — вывести «Hello!» на компьютер
• `setInterval(function() { … }, 350)` — вызывать функцию каждые 350 миллисекунд
• `setTimeout(function() { … }, 350)` — вызвать функцию через 350 миллисекунд
• `(3.1415926).toFixed(3)` — округлить до 3 знаков после запятой: 3,142
• `getTime()` — получить время в секундах с момента старта или перезагрузки платы

СОДЕРЖАНИЕ

ПРОЕКТЫ

В компании Амперка надеемся мы, что понравился наш набор.

Если вопросы есть у тебя, ответят на форуме на них: forum.amperka.ru

За порцией вдохновения к каналу на ю-тьюб обращайся: youtube.com/AmperkaRU

Руководства и инструкции подробные ищи на wiki.amperka.ru

За платами новыми и модулями могучими в магазин специальный иди: amperka.ru

Электронная версия книги: js.amperka.ru/jodo

Амперка

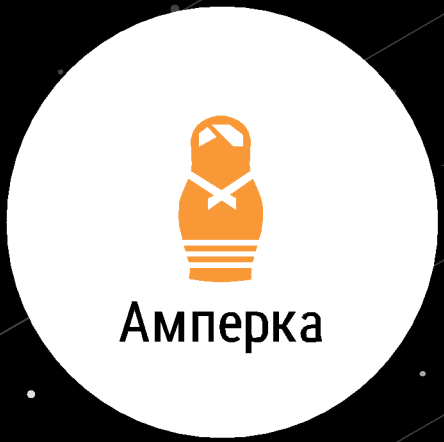
Конструктор Йодо
www.amperka.ru

vk.com/amperkaru

facebook.com/amperka.ru

instagram.com/amperkaru

twitter.com/amperka



СВЕТЛАЯ СТОРОНА СИЛЫ