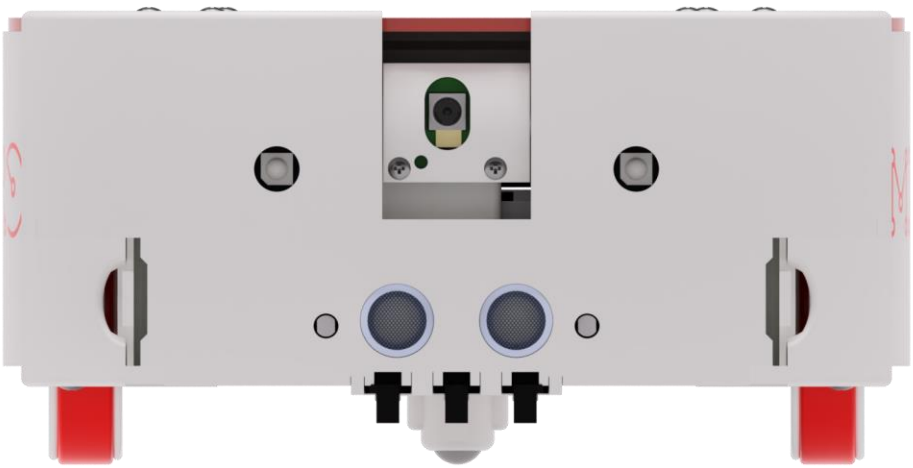
MIRO

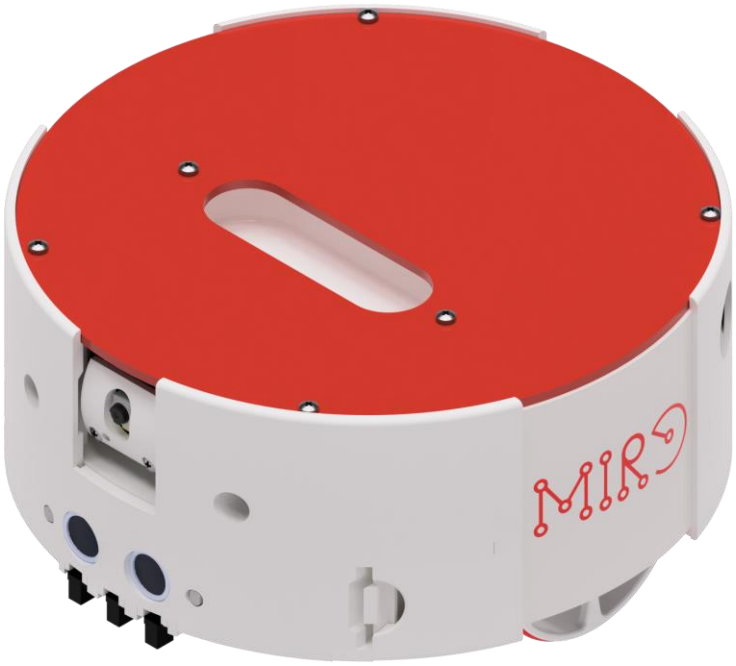
EDUCATION BOOK

LEVEL 1



УРОК 1. ЗНАКОМСТВО С РОБОТОМ MIRO

Давайте познакомимся с тем роботом с которым мы будем работать на протяжении всего курса! Знакомьтесь, его зовут MIRO («МАЙРО»)!



Он предназначен для того чтобы научить вас базовым практическим навыкам программирования микропроцессорных устройств, а также сборки и программированию простого колесного робота MIRO для выполнения ряда типовых задач мобильной робототехники.

У робота MIRO есть ряд ключевых особенностей.

Все аппаратные и программные решения робота MIRO распространяются по свободной лицензии CC Attribution 4.0 International (CC BY 4.0). Поэтому любой желающий может скачать проектные файлы и изготовить своего робота!

В производстве робота используется лишь три технологические операции:

раскрой листового материала на лазерном или фрезерном станке;

3D-печать;

пайка.

При этом, документация робота включает подробную технологическую карту производства деталей.

Робот включает доступные, недорогие серийные комплектующие (электроника и двигатели).

Робот MIRO построен по модульному принципу. Он предполагает установку различных модулей в передней и задней части робота с коммутацией электроники модулей с центральной вычислительной системой. Модули имеют безвинтовое крепление и фиксируются посредством клипс в корпусе робота. Разработчики робота надеются, что другие авторы, экспериментаторы, мейкеры будут разрабатывать модули для робота и публиковать свои решения под свободной лицензией.

Сборка робота проста и не должна вызвать затруднений даже у неопытного робототехника.

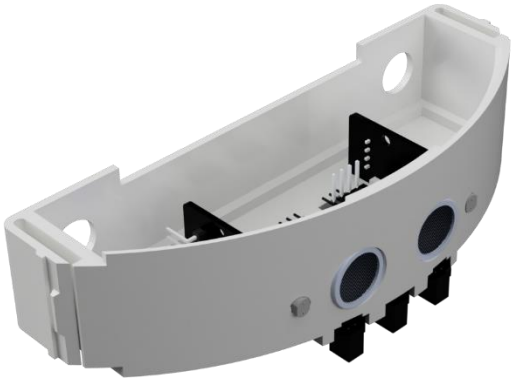
В роботе используется связка из микрокотроллерной платы RobotDyn Arduino+WiFi и одноплатного компьютера RASPBERRY PI ZERO W. Последнее обстоятельство позволяет реализовывать на базе робота достаточно серьезные алгоритмы.

Робот оснащен камерой RASPBERRY PI CAMERA V1.3, подключенной к плате RASPBERRY PI, что позволяет при разработке и обучении использовать пространство возможностей, связанное с техническим зрением.

В базовой комплектации робот MIRO оснащен передним модулем SENS1, включающим:

- три цифровых датчика линии;
- ультразвуковой датчик HC-SR04;
- два фоторезистивных датчика освещенности.

Схема подключения всех устройств базовой комплектации робота с модулем SENS1 представлена в Приложении 1.



Многие современные образовательные программы на рынке в области IT часто позиционируются как доступные для освоения даже абсолютно не подготовленными людьми. Однако, из нашего опыта, почти всегда – это не совсем правда. В рамках нашей образовательной программы приводится достаточно подробный разбор примеров программ, даются подробные комментарии. Тем не менее, авторы надеются, что вы уже немного знакомы с языком программирования C/C++. Это существенно облегчит понимание материала и выполнение практических заданий. Если же Вы еще не знакомы с этим прекрасным языком программирования, то мы рекомендуем Вам пройти один из множества бесплатных курсов программирования, доступных в сети Интернет (например: https://stepik.org курс «Введение в программирование (C++)).

УРОК 2. ЗНАКОМСТВО С ПЛАТФОРМОЙ ARDUINO

В роботе MIRO используется связка из микрокотроллерной платы RobotDyn Arduino+WiFi и одноплатного компьютера RASPBERRY PI. К рассмотрению платформы RASPBERRY PI мы перейдем существенно позже, а сейчас давайте познакомимся с платформой ARDUINO.

ARDUINO — аппаратная вычислительная платформа, основными компонентам которой являются простая плата ввода/вывода и среда разработки на языке Wiring (C++).

Аппаратная часть

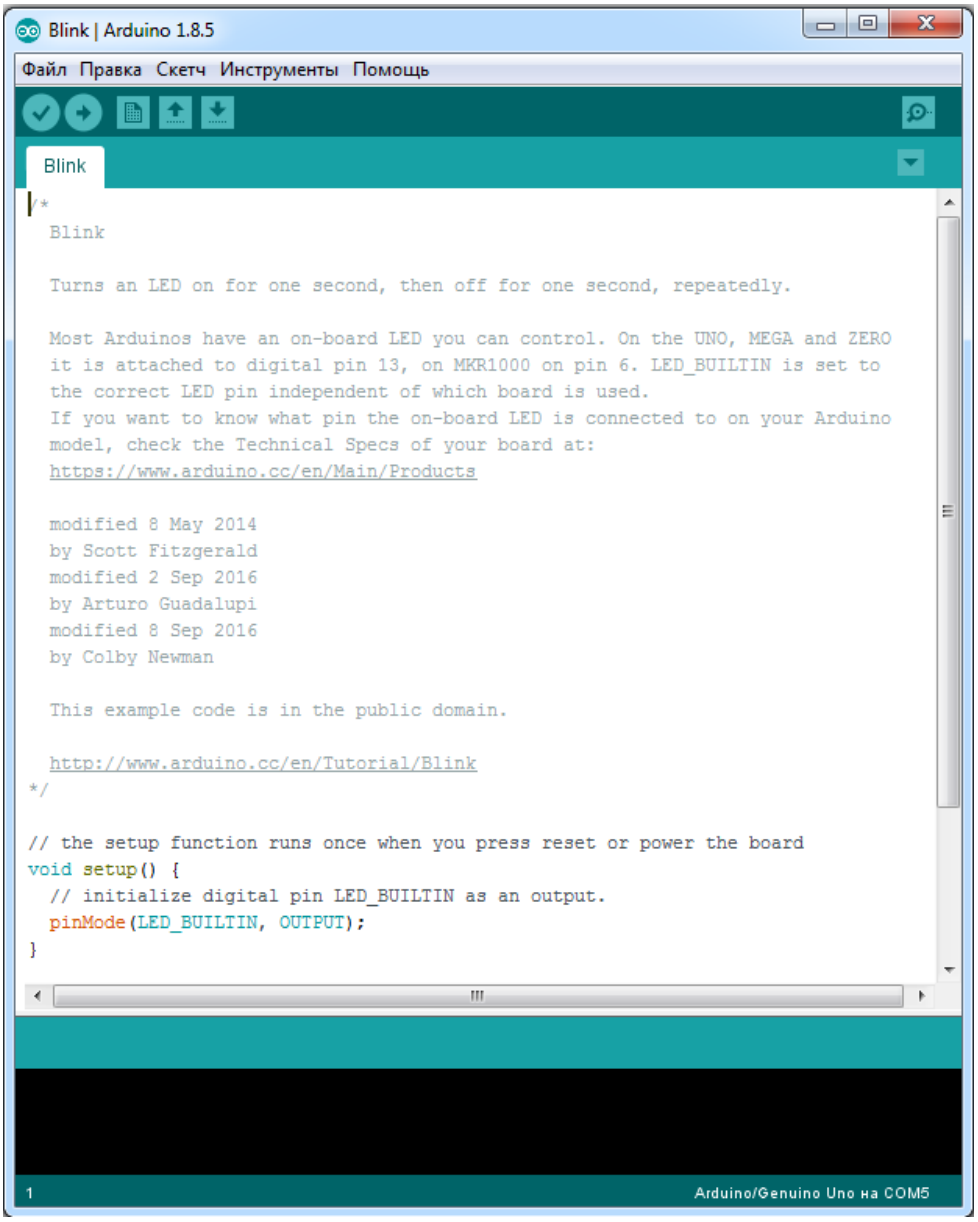
Плата ARDUINO состоит из микроконтроллера ATMEL AVR (ATmega328 и ATmega168 в новых версиях и ATmega8 в старых) и элементной обвязки для программирования и интеграции с другими схемами. На каждой плате обязательно присутствуют линейный стабилизатор напряжения 5В и 16 МГц кварцевый генератор (в некоторых версиях керамический резонатор). В микроконтроллер предварительно прошит загрузчик, поэтому внешний программатор не нужен.

На концептуальном уровне, все платы программируются через RS-232 (последовательное соединение), но реализация этого способа отличается от версии к версии. Плата в роботе MIRO программируются через USB, что осуществляется благодаря микросхеме конвертера USB-to-serial вроде FTDI FT232.

Платы ARDUINO позволяют использовать большую часть I/O выводов микроконтроллера во внешних схемах. Например, в плате робота доступно 14 цифровых вводов/выводов (уровни «LOW» - 0В и «HIGH» - 5В), 6 из которых могут выдавать ШИМ сигнал, и 6 аналоговых входов (0-5В). Эти выводы доступны в верхней части платы через 0,1 дюймовые разъёмы типа «мама». На рынке доступны несколько внешних плат расширения, известных как «shields».

Программное обеспечение

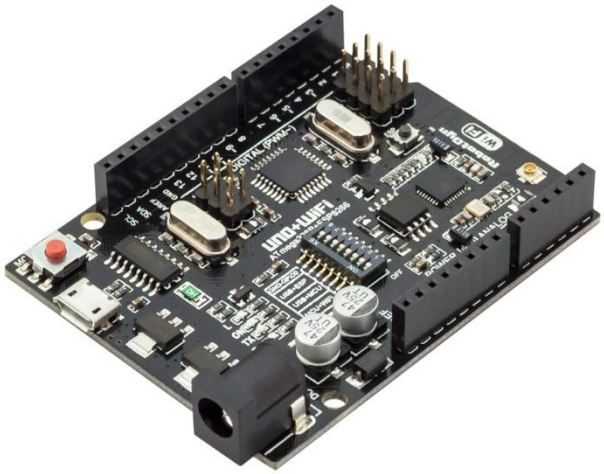
Интегрированная среда разработки ARDUINO IDE — это кроссплатформенное приложение на Java, включающее в себя редактор кода, компилятор и модуль передачи прошивки в плату.



Среда разработки спроектирована для программирования новичками, не знакомыми близко с разработкой программного обеспечения. Язык программирования аналогичен используемому в проекте Wiring. На официальном сайте пишут: "*...is programmed using the ARDUINO programming language (based on Wiring)*". Фактически — нет никакого особого языка программирования и программы пишутся на C/C++, а компилируются и собираются с помощью широко известного компилятора avr-gcc (в версии для Windows — WinAVR). Все особенности сводятся к тому, что имеется набор библиотек, включающий в себя некоторые функции (вроде pinMode) и объекты (вроде Serial), а при компиляции Вашей программы среда разработки создает временный .cpp файл, в который кроме Вашего кода включается еще несколько строчек, и полученный результат передается компилятору, а затем линковщику с нужными параметрами.

Клоны и разновидности

Название «ARDUINO» (и производные от него) является торговой маркой для официального продукта и не может использоваться для производных работ без разрешения. В официальном документе, об использовании названия ARDUINO, подчеркивается, что проект открыт для всех желающих работать над официальным продуктом. В базовой комплектации робота MIRO используется модифицированная плата ARDUINO UNO от российского производителя RobotDyn, оснащенная, кроме всего прочего еще и модулем беспроводной связи WiFi ESP8266.



Начало работы

С чего и как начать работу с MIRO, как подготовить среду разработки достаточно подробно описано в документе MIRO BEGINNRES GUIDE. Если вы еще не ознакомились с ним – самое время это сделать.

Сразу отметим одну замечательную особенность робота MIRO, с которой вам отныне придётся жить – робота вы можете программировать и отлаживать без всяких проводов! Как именно – читайте в MIRO BEGINNRES GUIDE

В уроке использованы материалы:
http://robocraft.ru/blog/arduino/14.html
http://robocraft.ru/blog/arduino/98.html
http://robocraft.ru/blog/arduino/714.html

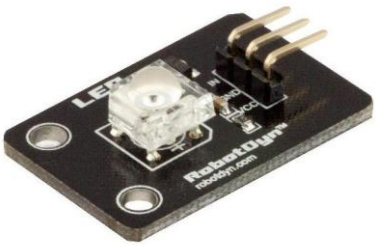
УРОК 3. СВЕТОДТОДНОЕ ОСВЕЩЕНИЕ РОБОТА

Описание урока.

Знакомимся с элементом подсветки робота MIRO. И попробуем написать первую программу для робота.

Информация по уроку

На сегодняшнем уроке мы будем программировать светодиодную подсветку робота. На роботе MIRO установлены два сверхярких светодиодных модуля RobotDyn Piranha.



Ниже представлен код примера для этого урока с пояснениями. Загрузите его в микроконтроллерную плату вашего робота, изучите принцип работы этой простой программы.

Код программы

```
#define LED_L 9 //номер пина, который управляет левым светодиодом подсветки
#define LED_R 10 //номер пина, который управляет правым светодиодом подсветки

void setup()
{
  // настраиваем пины №10 и №9 в режим выхода,
  pinMode(LED_L, OUTPUT);
  pinMode(LED_R, OUTPUT);
}

void loop()
{
  // подаём на пины LED_L и LED_R «высокий сигнал» (англ. «HIGH»), т.е.
  // выдаём 5 вольт. Через светодиоды побежит ток.
  // Это заставит их светиться.
  digitalWrite(LED_L, HIGH);
  digitalWrite(LED_R, HIGH);

  // задерживаем (англ. «delay») микроконтроллер в этом
  // состоянии на 100 миллисекунд
  delay(100);

  // подаём на пины LED_L и LED_R «низкий сигнал» (англ. «LOW»), т.е.
  // выдаём 0 вольт.
  // Врезультате светодиоды погаснут
  digitalWrite(LED_L, LOW);
```

```
  digitalWrite(LED_R, LOW);

  // замираем в этом состоянии на 900 миллисекунд
  delay(900);

  // после «размораживания» loop сразу же начнёт исполняться
  // вновь, и со стороны это будет выглядеть так, будто
  // светодиоды мигают раз в 100 мс + 900 мс = 1000 мс = 1 сек
}
```

Пояснения к коду

Процедура `setup` выполняется один раз при запуске микроконтроллера. Обычно она используется для конфигурации портов микроконтроллера и других настроек.

После выполнения `setup` запускается процедура `loop`, которая выполняется в бесконечном цикле. Именно этим мы пользуемся в данном примере, чтобы маячок мигал постоянно

Процедуры `setup` и `loop` должны присутствовать в любой программе (скетче), даже если вам не нужно ничего выполнять в них — пусть они будут пустые, просто не пишите ничего между фигурными скобками. Например:

```
void setup()
{
}
```

Запомните, что каждой открывающей фигурной скобке `{` всегда соответствует закрывающая `}`. Они обозначают границы некого логически завершенного фрагмента кода. Следите за вложенностью фигурных скобок. Для этого удобно после каждой открывающей скобки увеличивать отступ на каждой новой строке на один символ табуляции (клавиша Tab). Кроме того, в среде ARDUINO IDE есть встроенная функция автоформатирования кода, доступная из контекстного меню по щелчку правой кнопкой мыши – пользуйтесь этой функцией регулярно, чтобы привести свой код к подобающему виду.

Обращайте внимание на ; в концах строк. Не стирайте их там, где они есть, и не добавляйте лишних. Вскоре вы будете понимать, где они нужны, а где нет.

Функция `digitalWrite(pin,value)` не возвращает никакого значения и принимает два параметра:

`pin` — номер цифрового порта, на который мы отправляем сигнал

`value` — значение, которое мы отправляем на порт. Для цифровых портов значением может быть `HIGH` (высокое, единица) или `LOW` (низкое, ноль)

Если в качестве второго параметра вы передадите функции `digitalWrite` значение, отличное от `HIGH`, `LOW`, `1` или `0`, компилятор может не выдать ошибку, но считать, что передано `HIGH`. Будьте внимательны!

Обратите внимание, что использованные нами константы: `INPUT`, `OUTPUT`, `LOW`, `HIGH`, пишутся заглавными буквами, иначе компилятор их не распознает и выдаст ошибку. Когда ключевое слово распознано, оно подсвечивается синим цветом в ARDUINO IDE

Вопросы для проверки знаний

1. Зачем нужна встроенная функция `pinMode`? Какие параметры она принимает?
2. Зачем нужна встроенная функция `digitalWrite`? Какие параметры она принимает?
3. С помощью какой встроенной функции можно заставить микроконтроллер ничего не делать?
4. В каких единицах задается длительность паузы для этой функции?

Практические задания

1. Измените код примера так, чтобы светодиоды светились полсекунды, а пауза между вспышками была равна одной секунде.
2. Измените код примера так, чтобы светодиоды включались на три секунды после запуска устройства, а затем мигали в стандартном режиме.
3. Измените код примера так, чтобы светодиоды моргали попеременно.

В уроке использованы материалы:
http://wiki.amperka.ru/конспект-arduino:маячок

УРОК 4. АКУСТИЧЕСКИЙ ПЬЕЗОИЗЛУЧАТЕЛЬ РОБОТА

Описание урока.

Продолжаем знакомство с аппаратно-программным комплексом робота MIRO. Рассмотрим на этом уроке другое простое устройство - пьезодинамик (пьезоизлучатель звука), и напишем программу для работы с ним.

Информация по уроку

Самым простым и дешевым вариантом генерации звука является использование пьезоизлучателя.

Пьезокерамические излучатели (пьезоизлучатели) — электроакустические устройства воспроизведения звука, использующие пьезоэлектрический эффект - возникновение поляризации диэлектрика под действием механических напряжений (прямой пьезоэлектрический эффект). Существует и обратный пьезоэлектрический эффект — возникновение механических деформаций под действием электрического поля. Прямой пьезоэффект можно встретить, например, в пьезозажигалках, для получения высокого напряжения на разряднике. Обратный пьезоэлектрический эффект используется в пьезоизлучателях (эффективны на высоких частотах и имеют небольшие габариты). Пьезоизлучатели широко используются в различных электронных устройствах — часах-будильниках, телефонных аппаратах, электронных игрушках, бытовой технике.

Пьезокерамический излучатель состоит из металлической пластины, на которую нанесён слой пьезоэлектрической керамики, имеющий на внешней стороне токопроводящее напыление. Пластина и напыление являются двумя контактами.

Пьезоизлучатель также может использоваться в качестве пьезоэлектрического микрофона или датчика (прямой пьезоэффект).

В роботе MIRO пьезоизлучатель стандартно подключается к пину 8 платы ARDUINO.

Загрузите код примера в робота и изучите принцип его работы.

Код программы

```
// даём имя для пина с пьезоизлучателем (англ. buzzer)
```

```
#define BUZZER_PIN 8

void setup()
{
  // пин с пьезоизлучателем — выход...
  pinMode(BUZZER_PIN, OUTPUT);
  delay(500);
}

// объявляем переменные, которые будет выступать регуляторами частоты
// звучания
byte val = 0;
int frequency = 0;

void loop()
{
  // на каждом шаге цикла увеличиваем значение переменной, чтобы получить
  // эффект бегающей частоты.
  val = val + 1;

  // рассчитываем частоту звучания пищалки в герцах (ноту),
  // используя функцию проекции (англ. map). Она отображает
  // значение из одного диапазона на другой, строя пропорцию.
  // В нашем случае [0; 255] -> [100; 1000]. Так мы получим
  // частоту от 100 до 1000 Гц.
  frequency = map(val, 0, 255, 100, 1000);

  // заставляем пин с пищалкой «вибрировать», т.е. звучать
  // (англ. tone) на заданной частоте 15 миллисекунд. При
  // следующих проходах loop, tone будет вызван снова и снова,
  // и на деле мы услышим непрерывный звук
  tone(BUZZER_PIN, frequency, 15);
  delay(10);
}
```

Пояснения к коду

Переменным принято давать названия, начинающиеся со строчной буквы.

Чтобы использовать переменную, необходимо ее объявить, что мы и делаем инструкцией:

```
int val;
```

Для объявления переменной необходимо указать ее тип, здесь — `int` (от англ. *integer*) — целочисленное значение в диапазоне от -32 768 до 32 767. Переменные одного типа можно объявить в одной инструкции, перечислив их через запятую, что мы и сделали.

Обратите внимание, как мы записали значение в переменную `val`: мы просто поместили его в переменную `val` с помощью оператора присваивания `=`, который записывает то, что находится справа от него в ту переменную, которая стоит слева.

Функция `map(value, fromLOW, fromHIGH, toLOW, toHIGH)` возвращает целочисленное значение из интервала `[toLOW, toHIGH]`, которое является пропорциональным отображением содержимого `value` из интервала `[fromLOW, fromHIGH]`.

Верхние границы `map` не обязательно должны быть больше нижних и могут быть отрицательными. К примеру, значение из интервала `[1, 10]` можно отобразить в интервал `[10,-5]`.

Если при вычислении значения `map` образуется дробное значение, оно будет отброшено, а не округлено.

Функция `map` не будет отбрасывать значения за пределами указанных диапазонов, а также масштабирует их по заданному правилу.

Если вам нужно ограничить множество допустимых значений, используйте функцию `constrain(value, from, to)`, которая вернет:

- `value`, если это значение попадает в диапазон `[from, to]`;
- `from`, если `value` меньше него;
- `to`, если `value` больше него.

Функция `tone(pin, frequency, duration)` заставляет пьезодинамик, подключенный к порту `pin`, издавать звук высотой `frequency` герц на протяжении `duration` миллисекунд.

Параметр `duration` не является обязательным. Если его не передать, звук включится навсегда. Чтобы его выключить, вам понадобится функция `noTone(pin)`. Ей нужно передать номер порта с динамиком, который нужно выключить.

Одновременно можно управлять только одним пьезоизлучетелем. Если во время звучания вызвать `tone` для другого порта, ничего не произойдет.

Вызов `tone` для уже звучащего порта обновит частоту и длительность звучания.

Обратите внимание

Полярность пьезодинамика роли не играет: вы можете подключать любую из его ножек к земле, любую к порту микроконтроллера, но если есть разметка полярности, например возле одной ножки расположен плюс то следует эту ножку подключать к порту, не к земле.

Вопросы для проверки знаний

1. Что будет, если стереть из программы строчку `pinMode(BUZZER_PIN, OUTPUT);`?
2. Каков будет результат вызова `map(30,0,90,90,-90)`?
3. Как будет работать вызов `tone` без указания длительности звучания?
4. Можно ли устроить полифоническое звучание с помощью функции `tone`?

Практические задания

1. Пропищите азбукой Морзе позывной SOS: три точки, три тире, три точки.
2. Измените код программы так, чтобы звук раздавался не непрерывно, а 10 раз в секунду с различимыми паузами.

3. Изучите работу стандартного примера из библиотеки ARDUINO – Melody (Sketchbook — Examples –Digital – Melody). Измените код примера так, чтобы мелодия исполнялась не из памяти микроконтроллера, а из терминала (используйте для генерации мелодии какую-нибудь терминальную программу для ПК, например, Putty).

В уроке использованы материалы:
http://wiki.amperka.ru/конспект-arduino:терменвокс
https://ru.wikipedia.org/wiki/Пьезоэлектрический_эффект

УРОК 5. УЛЬТРАЗВУКОВОЙ ДАЛЬНОМЕР РОБОТА. ПОДКЛЮЧЕНИЕ И РАБОТА.

Описание урока

Изучаем ультразвуковой (УЗ) дальномер, входящий в базовый комплект робота MIRO с модулем SENS1. Пишем программы для определения расстояния до препятствия с использованием ультразвукового дальномера.

Информационный блок

УЗ дальномер – это устройство для измерения расстояния до объекта. Существуют также инфракрасные, лазерные и другие виды дальномеров. Их отличия заключаются в принципиально разных подходах к измерению расстояния, а именно типу и природе используемого сигнала. В УЗ дальномере используется ультразвуковой сигнал.

Дальномер состоит из двух компонентов:

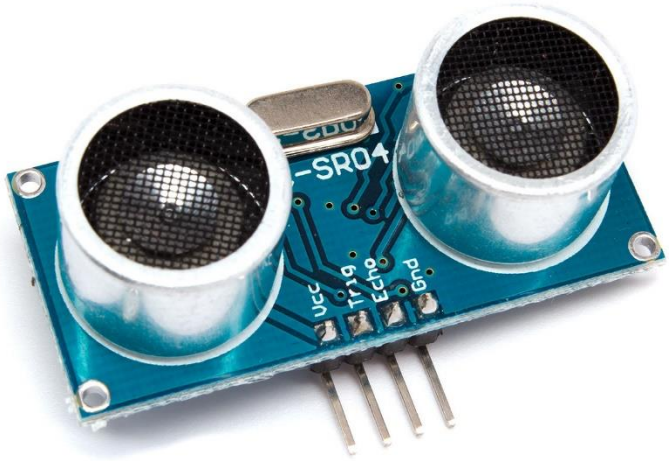
- УЗ излучатель – испускает ультразвуковой сигнал;
- УЗ приемник – принимает ультразвуковой сигнал.

В инфракрасном дальномере так же есть два устройства приемник и излучатель сигнала. Отличие лишь, в том, что в качестве сигнала используется инфракрасное излучение, а регистрирует его инфракрасный диод или транзистор.

В лазерном дальномер ситуация похожа, только используется сигналы длинных волн.

Особенность всех трех видов сигналов в том, что они отражается от объектов. Зная временной интервал между испускаемым сигналом и его возвратом на приемник, а также скорость распространения этого сигнала в среде, можно определить расстояние до объекта.

Очевидны и источники ошибок измерения каждым типом датчиков. Так, показания УЗ дальномера могут существенно изменятся при изменении скорости распространения УЗ волн в среде. А эта скорость в свою очередь имеет сложную нелинейную зависимость от других параметров, таких как влажность, температура, атфосферное давление воздуха. Кроме того, УЗ дальномеры как правило имеют достаточно широкую диаграмму направленности – в случае используемого в роботе недорого дальномер HC-SR04 – около 15 градусов. В ряде случаев, последнее обстоятельство может оказаться полезным, но в ряде случаев требуется более узкая диаграмма направленности.



У дальномера есть 4 контакта для подключения:

1. Vcc - питание +5В;
2. Trig - линия управления (триггер) излучения;
3. Echo – линия регистрации сигнала;
4. Gnd– минус.

Код программы

```
//получение данных от дальномера
//дальномер имеет два выхода излучатель и приемник, соответственно линии
//trig, echo

#define TRIG 13
#define ECHO 12

//переменные для хранения расстояния и времени распространения
long duration, distance;

void setup()
{
  Serial.begin(115200); //инициализация последовательного порта
  pinMode(TRIG,OUTPUT); //настройка пина излучателя на выход
}
void loop()
{
  digitalWrite(TRIG,LOW);
  delayMicroseconds(4);

  digitalWrite(TRIG,HIGH);
  delayMicroseconds(20);
  digitalWrite(TRIG,LOW);
  duration = pulseIn(ECHO, HIGH);
  //пересчет значений расстояния в сантиметры
  distance = duration/58.0;
  Serial.print("Distance: ");
  Serial.println(distance);
  delay(1000);
}
```

Пояснения к коду

Для того, чтобы получить данные о расстоянии необходимо придерживаться следующего плана действий.

Для начала необходимо подключить дальномер согласно контактам, которые были рассмотрены ранее. И реализовать код, в котором на первоначальном этапе объявляем две строки `TRIG` и `ECHO`. Данные строки будут хранить номера пинов подключенных к плате:

```
#define TRIG 13
#define ECHO 12
```

После этого объявления объявляем две глобальные переменные типа `long`. Данный тип данных хранит целые значения в расширенном диапазоне.

```
long duration, distance;
```

Каждое из которых необходимо для хранения значений:

`distance` – хранит вычисленное расстояние;

`duration` – хранит значение длительности сигнала.

На следующем этапе добавляем код в процедуру `setup()` и выполняем первоначальную настройки платы для работы.

- Настраиваем работу последовательного порта, для вывода данных о расстоянии:

```
Serial.begin(115200);
```

- Настраиваем пин излучателя на выход:

```
pinMode(TRIG, OUTPUT);
```

- На последнем этапе настраиваем процедуру `loop()`
- Для работы дальномера необходимо для начала сбросить сигнал на контакте `TRIG`:

```
digitalWrite(TRIG,LOW);
delayMicroseconds(4);
```

После этого алгоритм работы с ультразвуковым дальномером следующий:

- Посылаем сигнал от источника

```
digitalWrite(TRIG,HIGH);
delayMicroseconds(20);
```

- Устанавливаем низкий сигнал на источнике

```
digitalWrite(TRIG,LOW);
```

- С помощью функции `pulseIn()` считываем время, которое прошел сигнал.

```
duration = pulseIn(ECHO, HIGH);
```

Функция `pulseIn()` считывает длину сигнала на заданном порту (`HIGH` или `LOW`). Например, если задано считывание `HIGH` функцией `pulseIn()`, функция ожидает пока на заданном порту не появиться `HIGH`. Когда `HIGH` получен, включается таймер, который будет остановлен, когда на порту вход/выхода будет `LOW`. Функция `pulseIn()` возвращает длину сигнала в микросекундах. Функция возвращает `0`, если в течение заданного времени (таймаута) не был зафиксирован сигнал на порту.

Полученное значение делим на коэффициент полученный из паспорта к датчику расстояния «58.0». Для пересчета расстояния в сантиметры:

```
distance = duration/58.0;
```

В конце выводим данные на экран

```
Serial.println("distance: ");
Serial.println(distance);
delay(1000); // делаем паузу, чтобы увидеть результат
```

Обратите внимание

Согласно техническому паспорту на ультразвуковой дальномер HC-SR04, минимальное расстояние, на котором дальномеры считывают расстояние 2 см, а максимальное 4 метра.

Вопросы для проверки знаний

1. Назовите ключевое отличие ультразвукового и лазерного дальномеров.
2. Что будет, если увеличить длину импульса излучателя?
3. Объясните, откуда взялась цифра «58», на которую мы делим время распространения `duration` чтобы получить расстояние в сантиметрах? От чего зависит эта цифра?

Практическое задание

1. Выведете значение расстояния в дюймах, метрах.
2. Напишите программу, вычисляющую скорость сближения препятствия и робота.

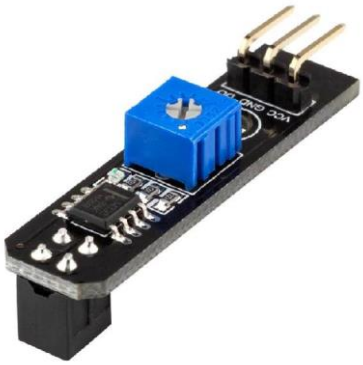
УРОК 6. ДАТЧИКИ ЛИНИИ С ЦИФРОВЫМ ВЫХОДОМ

Описание урока

На этом уроке познакомимся с инфракрасными датчиками линии с цифровым выходом, и узнаем для чего они нужны роботу.

Информационный блок

Модуль SENS1 содержит три датчика линии (контраста) RobotDyn Sens-LineTrack с цифровым выходом. Датчик освещает поверхность направленным инфракрасным (ИК) светодиодом (длинна световой волны 940нм). Световой поток отражается от поверхности и попадает на кремниевый NPN фото-транзистор, где преобразуется в электрический сигнал. Так как используется фото-транзистор NPN типа (обратной проводимости), с нагрузкой на коллекторе, то уровень электрического сигнала обратно-пропорционален отраженному от поверхности свету и прямо пропорционален удалению от отражающей поверхности. На выходе датчика стоит компаратор, который приводит уровень сигнала к значению логического «0) или «1». Порог срабатывания компаратора можно задавать при помощи переменного резистора, размещенного на датчике.



Датчик линии, как и большинство датчиков, подключаемых к плате ARDUINO использует 3 контакта для подключения:

1. VCC - питание 5В;
2. GND - (заземление) минус питания;
3. DO - сигнал – так как датчик цифровой, сигнал будет цифровой (0 или 1).

Для работы с датчиком на этом уроке мы будем использовать пин A0. Хоть он и аналоговый он позволит нам считывать цифровые данные. Просто он будет интерпретировать значения от 0 до 1023, как 0 и 1.

Итак, давайте, напишем простой код, для чтения данных с центрального (middle) датчика линии.

Код программы

```
/*С помощью директивы указываем номер пина, подключенного к сигнальной линии датчика*/
#define LINE_C A1

//объявляем переменную для хранения значения от датчика
unsigned int line_c;

void setup()
{
   //инициализация работы последовательного порта
   //настройка скорости его работы
   Serial.begin(115200);
   pinMode(LINE_C, INPUT);
}

void loop()
{
   //считываем значение из пина
   line_c = digitalRead(LINE_C);

   //выводим строку
   Serial.print("Center sensor line: ");

   //вывод значения переменной в последовательный порт
   Serial.println(line_c);

   //Устанавливаем значение задержки микроконтроллера
   delay(1000);
}
```

Вопросы для проверки знаний

1. Назовите значения напряжения логических уровней «0» и «1».

Практическое задание

1. Подключите левый и правый датчики линии и выполните вывод данных с трех датчиков.

В уроке использованы материалы:
https://wiki.iarduino.ru/page/datchik-linii-analogovyy/

УРОК 7. ДАТЧИК ОСВЕЩЕННОСТИ РОБОТА

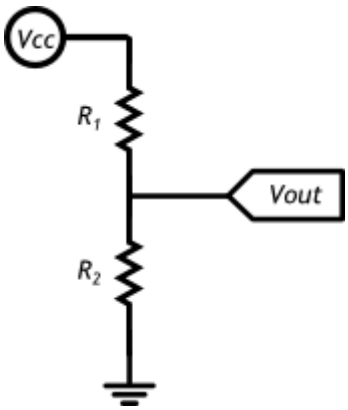
Описание урока

На этом уроке познакомимся с фоторезистивным датчиком освещенности, изучим его устройство и принципы работы. Напишем несколько программ, которые выполняют настройку робота на прием и обработку сигнала от датчика.

Информационный блок

Фоторезистор — полупроводниковый прибор, изменяющий величину своего сопротивления при облучении светом. В полной темноте он имеет максимальное сопротивление в сотни кОм, а по мере роста освещённости сопротивление уменьшается до десятков кОм.

На его основе очень просто собрать схему, которая бы позволяла регистрировать уровень освещенности. Для этого достаточно собрать элементарный делитель напряжения, где одним из резисторов будет фоторезистор, а вторым — резистор соразмерного номинала (например - 10 кОм). Делитель напряжения, как вы наверное уже знаете, представляет из себя последовательно подключённые резисторы, которые делят поступающее на них напряжение в определённой пропорции.



Сила тока, протекающая через резисторы одинакова, т.к. они соединены последовательно, и по закону Ома может быть рассчитана как:

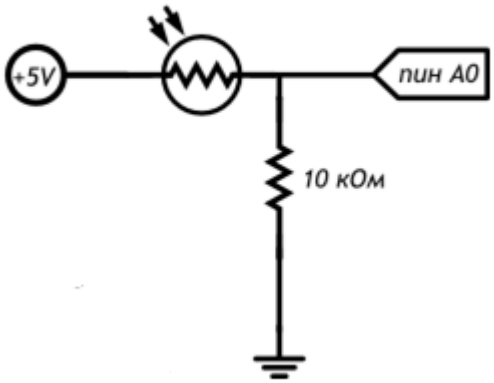
$$I = \frac{V_{CC}}{R_1 + R_2}$$

По тому же закону Ома можно вычислить выходное напряжение V_{out}, которое падает на резисторе R_2:

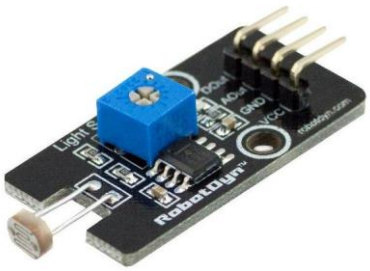
$$V_{out} = U_2 = I \times R_2 = \frac{R_2 \cdot V_{CC}}{R_1 + R_2}$$

Из полученной формулы видно, что чем больше R_2 относительно R_1, тем большее напряжение падает на нём.

Если вместо R_2 использовать не постоянный резистор, а устройство, которое меняет своё сопротивление, в зависимости от регистрируемого параметра внешней среды, выходное напряжение V_{out} будет также зависеть от этого измеряемого значения этого параметра.



И микроконтроллер умеет измерять напряжение. Таким образом, мы можем использовать свойства делителя напряжения для получения показаний от фоторезистора. Именно так устроен используемый в модуле SENS1 робота MIRO фоторезистивный датчик RobotDyn Sens-LightADout.



В данной схеме выходное напряжение делителя напряжения подключается к аналоговому пину ARDUINO A0, а фоторезистор и резистор в 10 кОм делят входное напряжение равное 5 В.

Схема на роботе

В Базовой комплектации робота MIRO, в модуле SENS1 установлены два датчика освещенности на основе фоторезисторов.

Код программы

```
#define LDR_L_PIN A2
#define LDR_R_PIN A3

void setup() //процедура setup
{
  Serial.begin(115200);
  Serial.println();
}

void loop() //процедура loop
{
  //объявляем переменную, которая будет хранить данные считанные
  //из аналогового пина
  int val_L = analogRead(LDR_L_PIN);
  int val_R = analogRead(LDR_R_PIN);
  //записать данные в последовательный порт
  Serial.println("Light sensor values: ");
  Serial.println("LEFT  RIGHT");
  Serial.print(val_L);
  Serial.print("  ");
  Serial.println(val_R);
  Serial.println();

  //выставляем задержку после вывода данных
  delay(1000);
}
```

Пояснение к коду:

Функция `analogRead(LDR_L_PIN)` и `analogRead(LDR_R_PIN)` считывают данные из аналоговых пинов. Числа представлены в формате 10 бит, имеют диапазон значений в интервале от 0 до 1023

Еще раз обратите внимание на использование функции `Serial`, они используются для работы с последовательным портом. И для начала необходимо инициализировать работу с помощью функции `begin`, затем с помощью функции `print` мы выводим данные в монитор порта.

Обратите внимание

Фоторезистор имеет два вывода, но не имеет полярности. Для получения данных с фоторезистора необходимо реализовать делитель напряжения. При увеличении освещенности, сопротивление фоторезистора падает, следовательно, значение на выходе делителя напряжение наоборот увеличивается.

Вопросы для проверки знаний

1. Если мы установим фоторезистор в схеме датчика между аналоговым входом и землей, наше устройство будет работать наоборот: светодиод будет включаться при увеличении количества света. Почему?
2. Какой результат работы устройства мы получим, если свет от светодиода будет падать на фоторезистор?
3. Если мы все же установили фоторезистор так, как сказано в предыдущем вопросе, как нам нужно изменить программу, чтобы устройство работало верно?
4. Допустим, у нас есть код `if (условие) {действие;}`. В каких случаях будет выполнено `действие`?
5. При каких значениях `y` выражение `((x + y) > 0)` будет истинным, если `x > 0`?
6. Обязательно ли указывать, какие инструкции выполнять, если условие в операторе `if` ложно?
7. Чем отличается оператор `==` от оператора `=`?
8. Если мы используем конструкцию `if (условие) действие1; else действие2;`, может ли быть ситуация, когда ни одно из действий не выполнится? Почему?

Практические задания

1. Напишите программы включения подсветки робота при наступлении темноты и отключении при достаточном наружном освещении.

В уроке использованы материалы:
http://wiki.amperka.ru/конспект-arduino:ночной-светильник

УРОК 8. ДВИЖЕНИЕ РОБОТА

Описание урока

Знакомимся с понятием «широтно-импульсная модуляция» сигнала (ШИМ), и реализуем движение робота.

Информация по уроку

Широтно-импульсная модуляция (ШИМ, PWM — Pulse Width Modulation)) импульсный сигнал постоянной частоты и переменной скважности (отношение длительности импульса к периоду его следования). С помощью задания скважности можно менять среднее напряжение на выходе ШИМ. Т.е. хоть мы и работаем с цифровым устройством, которое понимает только 1 и 0 (высокий уровень напряжения +3..+5V (HIGH) и низкий уровень напряжения 0..+2V (LOW)), но мы всё же можем получить напряжение отличное от данных *изменяя скважность импульсов*. Другими словами – получаем аналоговый сигнал цифровыми методами.

На плате RobotDyn UNO+WiFi (ARDUINO) робота MIRO есть несколько цифровых пинов, которые позволяют генерировать ШИМ сигнал. Это пины № 3, 5, 6, 9, 10, 11. Они позволяют задавать значение «заполнения» периода в диапазоне от 0 до 255. Максимальное значение в вольтах для значения 255, соответствует максимальному напряжению порта 5В. Значение 0 соответствует значению выходного напряжения 0В. Исходя из этого можно управлять напряжением путем изменения значения ШИМ-сигнала. Можно определить значение в вольтах для сигнала ШИМ, например, 128, или наоборот, посчитать значение ШИМ по значению напряжения.

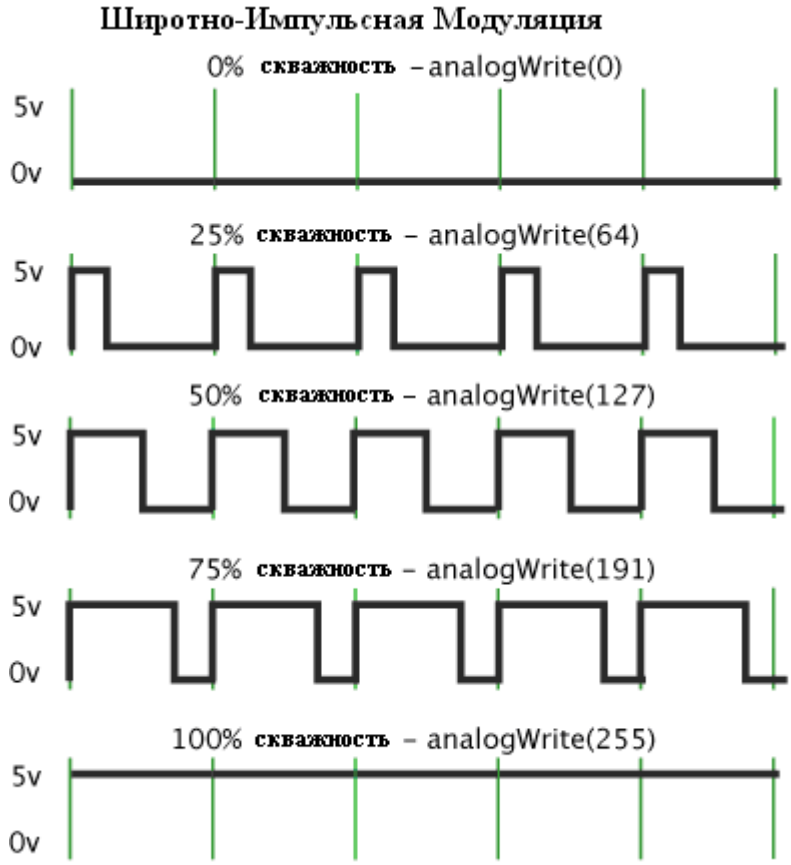
Для расчета необходимо использовать следующие формулы:

$$U = (S \times 5)/255,$$

$$S = (U \times 255)/5,$$

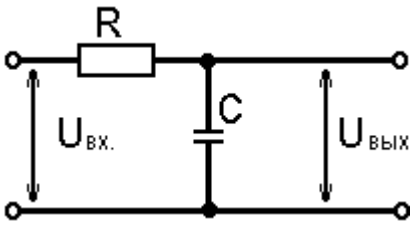
где S – это значение ШИМ, а U – это значение напряжения.

Для формирования ШИМ сигнала на выводе платы ARDUINO используется функция `analogWrite()`, которая принимает два параметра. Первый – это номер пина. Второй – это значение ШИМ (от 0 до 255). Ниже представлен вид сигнала на выходе при разных параметрах функции `analogWrite()`.



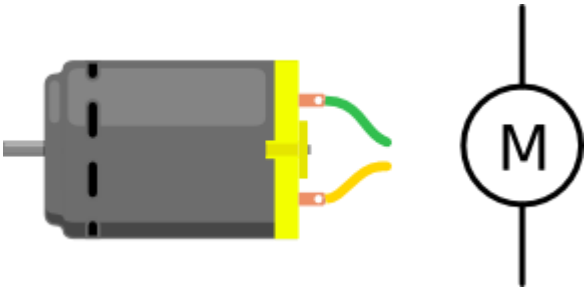
0V, 1.25V, 2.5V, 3.75V и 5V соответственно.

Разумеется, как видно из того же рисунка, на выходе наблюдается всё тот же цифровой сигнал, но если подключить простую интегрирующую цепочку, то на выходе мы получим уже вполне аналоговый сигнал.



Пример работы ШИМ можно посмотреть в тестовом скетче Fading (File -> SketchBook -> Examples -> Analog -> Fading). В том примере с помощью ШИМ управляют яркостью светодиода, подключенного к плате ARDUINO, а «интегрирующую» роль выполняет инерционность наших глаз и самого светодиода.

Электродвигатель – это устройство, которое переводит электрическую энергию в механическую энергию вращения. Мощность, количество оборотов в минуту, напряжение и тип питания являются основными показателями электродвигателей. Также, большое значение имеют массогабаритные и энергетические показатели.



Самый простой вид электродвигателя — коллекторный двигатель постоянного тока. Именно такие двигатели, объединенные с редукторами (говорят: «мотор-редукторы») установлены в роботе MIRO. При подаче напряжения в одном полярности вал крутится по часовой стрелке, в обратной полярности — против часовой.

В отличие от рассмотренных ранее устройств робота, электродвигатели потребляют гораздо больше энергии и требуют специальных средств для управления. В роботе MIRO используется специальная плата, с распаянными на ней двумя драйверами для управления двигателями постоянного тока. Максимальный ток, который может обеспечить каждый драйвер составляет 800 мА. Это достаточно мало, но это в 16 раз больше предельного тока, который может обеспечить вывод платы RobotDyn UNO+WiFi (ARDUINO). Моторы, установленные в роботе потребляют от 200 до 600 мА – в зависимости от характера движения (нагрузки при движении).

Обмотки электродвигателя в условиях вращающегося ротора, при управлении сигналом ШИМ также выступают интегратором, фильтруя импульсную составляющую напряжения. Именно поэтому для управления коллекторными электродвигателями (и не только) широко применяется ШИМ-сигналы.

Обратите внимание

Плата с драйверами мотор-редукторов робота MIRO подключаются к пинам № 4, 5, 6, 7. Выводы 4 и 7 нужны для управления направлением движения робота, а 5 и 6 - для управления скоростью движения робота.

Для управления каждым двигателем используется два сигнала. Один цифровой, который задает направление вращения мотора, второй ШИМ, который управляет скоростью вращения.

При низких значениях длины импульса до «50» двигатели не проявляют реакции. Ток через обмотки идет, но этот ток не является пусковым – его недостаточно, чтобы создать необходимое магнитное поле и провернуть ротор двигателя.

Код программы 1

```
// даём разумное имя для пинов 4 и 5 - линии управления одним из
// двигателей робота
// Так нам не нужно постоянно вспоминать куда он подключён
#define PWM_L_PIN 5
#define DIR_L_PIN 4

void setup()
{
  // настраиваем пины в режим выхода,
  // как и раньше
  pinMode(PWM_L_PIN, OUTPUT);
  pinMode(DIR_L_PIN, OUTPUT);
}

void loop()
{
  // задаем направление вращения
  digitalWrite(DIR_L_PIN, LOW);
  // выдаём неполное напряжение на двигатель
  // (он же ШИМ-сигнал, он же PWM-сигнал).
  // Микроконтроллер переводит число от 0 до 255 к напряжению
  // от 0 до 5 В. Например, 85 — это 1/3 от 255,
  // т.е. 1/3 от 5 В, т.е. 1,66 В.
  analogWrite(PWM_L_PIN, 85);
  // держим такую скорость 3 секунды
  delay(3000);

  // выдаём 170, т.е. 2/3 от 255, или иными словами — 3,33 В.
  // Больше напряжение — выше скорость!
  analogWrite(PWM_L_PIN, 170);
  delay(3000);

  // все 5 В — полный вперед!
  analogWrite(PWM_L_PIN, 255);
  // ждём ещё немного перед тем, как начать всё заново
  delay(3000);
}
```

Пояснения к коду 1

Идентификаторы переменных, констант, функций (в этом примере идентификатор `PWM_A_PIN`) являются одним словом (т.е. нельзя создать идентификатор `PWM_A PIN`).

Идентификаторы могут состоять из латинских букв, цифр и символов подчеркивания _. При этом идентификатор не может начинаться с цифры.

```
PRINT       // верно
PRINT_3D    // верно
MY_PRINT_3D // верно
_PRINT_3D   // верно
```

```
3D_PRINT    // ошибка
ПЕЧАТЬ_3Д   // ошибка
PRINT:3D    // ошибка
```

Регистр букв в идентификаторе имеет значение. Т.е. `PWM_A_PIN`, `PWM_A_pin` и `PWM_A_Pin` с точки зрения компилятора — различные идентификаторы.

Идентификаторы, создаваемые пользователем, не должны совпадать с предопределенными идентификаторами и стандартными конструкциями языка; если среда разработки подсветила введенный идентификатор каким-либо цветом, замените его на другой.

Директива `#define` просто говорит компилятору заменить все вхождения заданного идентификатора на значение, заданное после пробела (здесь `9`), эти директивы помещают в начало кода. В конце данной директивы точка с запятой `;` не допустима.

Названия идентификаторов всегда нужно делать осмысленными, чтобы при возвращении к ранее написанному коду вам было ясно, зачем нужен каждый из них.

Также полезно снабжать код программы комментариями: в примерах мы видим однострочные комментарии, которые начинаются с двух прямых слэшей `//` и многострочные, заключённые между `/* */`.

```
// однострочный комментарий следует после двойного слеша до конца строки
/* многострочный комментарий
   помещается между парой слеш-звездочка и звездочка-слеш */
```

Комментарии игнорируются компилятором, зато полезны людям при чтении давно написанного, а особенно чужого, кода.

Теперь давайте напишем программу вращения двигателей в противофазе, в разных направлениях с одинаковой скоростью. В таком режиме работы двигателей робот будет разворачиваться на месте.

Код программы 2

```
// даём разумное имя для пинов 4, 5, 6, 7 - линии управления
// двигателями робота
// Так нам не нужно постоянно вспоминать куда они подключёны
#define PWM_L_PIN 5
#define DIR_L_PIN 4
#define PWM_R_PIN 6
#define DIR_R_PIN 7

#define LIN_SPEED 120

void setup()
{
  // настраиваем пины в режим выхода,
  // как и раньше
  pinMode(PWM_L_PIN, OUTPUT);
  pinMode(DIR_L_PIN, OUTPUT);
  pinMode(PWM_R_PIN, OUTPUT);
  pinMode(DIR_R_PIN, OUTPUT);
```

```
}

void loop()
{
  digitalWrite(DIR_L_PIN, HIGH);
  digitalWrite(DIR_R_PIN, LOW);
  analogWrite(PWM_L_PIN, 255 - LIN_SPEED);
  analogWrite(PWM_R_PIN, LIN_SPEED);

  delay(2500);

  digitalWrite(DIR_L_PIN, LOW);
  digitalWrite(DIR_R_PIN, HIGH);
  analogWrite(PWM_L_PIN, LIN_SPEED);
  analogWrite(PWM_R_PIN, 255 - LIN_SPEED);

  delay(2500);
}
```

Пояснения к коду 2

Обратите внимание на то, как в коде программы задается скорость вращения колес в противофазе. Понятно, что скорость вращения зависит от напражения на контактах двигателя. Рассмотрим этот момент подробнее. Для правого двигателя все просто: на линию `DIR_R_PIN` подается низкий уровень (`digitalWrite(DIR_R_PIN, LOW)`), а на линию ШИМ – `LIN_SPEED` (`analogWrite(PWM_R_PIN, LIN_SPEED)`). Но чтобы левый двигатель вращался в обратном направлении с той же скоростью, необходимо, чтобы модуль разности потенциалов на его клеммах был равен модулю разности потенциалов на клеммах правого двигателя, но знак разности отличался. Для этого, нужно подать на цифровую линию управления правым двигателем высокий уровень (`digitalWrite(DIR_L_PIN, HIGH)`), а на ШИМ-линию – разность между высоким уровнем (соответствует значению 255 для ШИМ) и `LIN_SPEED` (`analogWrite(PWM_L_PIN, 255-LIN_SPEED)`). Возможно, формальный расчет внесет больше понимания.

```
    Ur = Vin * (PWM_R_PIN - DIR_R_PIN)/255 = 7,4 * (120 – 0)/255 = 3,48В
    Ul = Vin * (PWM_L_PIN – DIR_L_PIN)/255 = 7,4 * ((255-120) - 255)/255 = 7,4
* (0 – 120)/255 = -3,48В.
    Где:
          Ur – напряжение на правом двигателе;
          Ul – напряжение на левом двигателе;
          Vin = 7,4В – напряжение питания драйвера двигателей (фактически –
напряжение питания аккумулятора робота).
```

Очевидно, что если подать на `PWM_L_PIN` значение просто `LIN_SPEED` (а не `(255-LIN_SPEED)`), то на левом двигателе получиться совсем другая разность потенциалов, модуль которой не равен модулю разности потенциалов на правом двигателе:

```
    Ul = Vin * (A_IB – A_IA)/255 = 7,4 * (255 - 120)) = 7,4 * (0 – 120) =
3,91В.
```

Стоит также учитывать, что направление вращения обоих двигателей и каждого по отдельности может отличаться от ожидаемого – в зависимости от того как подключены выводы драйвера к клеммам двигателей. Если какой-то из двигателей вращаются не в ту сторону, следует произвести перекоммутацию проводов, идущих от драйвера к двигателю (поменять местами в пределах соответствующей пары).

Вопросы для проверки знаний

1. Почему при максимальных значениях длины импульса ШИМ сигнала, робот едет вперед медленнее, чем назад.
2. Добавлением каких строк кода, обеспечит остановку робота.
3. Вычислите значение напряжения для значений ШИМ в функции `analogWrite()`, равных 34, 128, 6, 182, 201.

Практическое задание

1. Напишите программу, для движения робота налево, направо, назад.
2. Напишите программу движения робота по квадрату или окружности.
3. Запрограммировать робота таким образом, чтобы он издавал мелодию и в такт мелодии мигал светодиодом.
4. Запрограммировать робота таким образом, чтобы он стал полицейской машинкой, другими словами ему необходимо мигать светодиодом поочередно, и издавать звук сирены, и ездить в разные стороны.

В уроке использованы материалы:
http://robocraft.ru/blog/arduino/34.html

УРОК 9. Память EEPROM.

Описание урока

На уроке мы кратко рассмотрим устройство памяти в ARDUINO и реализуем работу с одной энергонезависимой памятью на ARDUINO.

Информационный блок

Все микроконтроллеры AVR, на основе которых создана ARDUINO, имеют три вида памяти:

- Flash-память (ROM) – в ней хранится программа, выполняемая ядром и константы;
- ОЗУ (RAM) – статическая оперативная память, используется для размещения переменных, массивов и т.п.;
- EEPROM (ЭСППЗУ) – используется для хранения пользовательских данных.

Flash и EEPROM являются так называемой энергонезависимой памятью (volatile memory). Это означает, что данные в них сохраняются и не изменяются даже при пропадании питания.

Оперативная память (Random Access Memory, RAM, память с произвольным доступом) или оперативное запоминающее устройство (ОЗУ) - энергозависимая часть системы компьютерной памяти, в которой во время работы компьютера хранится выполняемый машинный код (программы), а также входные, выходные и промежуточные данные, обрабатываемые ядром.

Содержащиеся в современной полупроводниковой оперативной памяти данные доступны и сохраняются только тогда, когда на модули памяти подаётся напряжение. Выключение питания оперативной памяти, даже кратковременное, приводит к искажению либо полному разрушению хранимой информации.

В общем случае, от объёма оперативной памяти зависит количество задач, которые одновременно может выполнять вычислительная система.

Flash-память — разновидность полупроводниковой технологии электрически перепрограммируемой памяти. Это же слово используется в электронной схемотехнике для обозначения технологически законченных решений постоянных запоминающих устройств в виде микросхем на базе этой полупроводниковой технологии. В быту это словосочетание закрепилось за широким классом твердотельных устройств хранения информации.

Благодаря компактности, дешевизне, механической прочности, большому объёму, скорости работы и низкому энергопотреблению, Flash-память широко используется в цифровых портативных устройствах и носителях информации. Серьёзным недостатком данной технологии является ограниченный срок эксплуатации носителей, а также чувствительность к электростатическому разряду.

Познакомимся с памятью EEPROM, и напишем программы по работе с этим типом памяти.

Информационный блок

EEPROM — (Electrically Erasable Programmable Read-Only Memory) электрически стираемое перепрограммируемое ПЗУ, ЭСППЗУ). Память такого типа может стираться и заполняться данными несколько десятков тысяч раз. Используется в твердотельных накопителях.

Микроконтроллеры ATmega328, работающие в установленной в роботе плате RobotDyn UNO+Wi-Fi имеют на борту 512 байт EEPROM – энергонезависимой памяти, в которой можно сохранять какие-либо данные, которые будут доступны после отключения питания.

Для работы с данной памятью в составе ARDUINO IDE уже есть удобная библиотека EEPROM (\hardware\libraries\EEPROM\).

Библиотека содержит всего две ключевые функции – чтения и записи данных.

Код программы (чтение данных)

```
#include <EEPROM.h>

// начальный адрес памяти EEPROM
int address = 0;
byte value;

void setup()
{
  Serial.begin(115200);
}

void loop()
{
  // считываем значение по текущему адресу EEPROM
  value = EEPROM.read(address);

  Serial.print(address);
  Serial.print("\t");
  Serial.print(value, DEC);
  Serial.println();

  // устанавливаем следующую ячейку памяти
  address = address + 1;

  // EEPROM содержит всего 512 байт: от 0 до 511, поэтому
  // если адрес достиг 512, то снова переходим на 0
  if (address == 512)
    address = 0;

  delay(500);
}
```

Пояснение к коду

Метод класса `EEPROM.read (address)` считывает байт из энергозависимой памяти, хранящийся по адресу `address`. Если байт не перезаписывался прежде никем, то вернется значение 255 (`0xFF`).

Переменная `address` хранит номер ячейки памяти для чтения (от 0 до 511 – целый тип данных)

Код программы (записи данных)

```
/*
 * EEPROM Write
 *
 * Сохраняет в энергонезависимой памяти EEPROM значения,
 * считанные с аналогового входа analog input 0.
 * Данные значения останутся в памяти и после отключения питания
 * от платы и в будущем могут быть доступны для
 * другого скетча.
*/

#include <EEPROM.h>

// текущее значение адреса EEPROM
int address = 0;

void setup()
{
}

void loop()
{
  // деление на 4 необходимо, чтобы перевести значение от
  // 0-1023 к одному байту, т.к. EEPROM может хранить только
  // значения от 0 до 255.
  int value = analogRead(0) / 4;

  // записываем значение в энергонезависимую память
  EEPROM.write(address, value);

  // устанавливаем следующую ячейку памяти.
  // т.к. EEPROM содержит всего 512 ячеек - при достижении
  // конца памяти - возвращаемся на начало :)
  address = address + 1;
  if (address == 512)
    address = 0;

  delay(100);
}
```

Обратите внимание

Документация (технической паспорт (Datasheet)) на микроконтроллер ATmega328 говорит, что возможное количество циклов перезаписи данных в памяти не менее 100000 раз (Write/Erase Cycles). Это очень много, но все же, это обстоятельство следует учитывать при использовании данной памяти.

Так же документация указывает, что время, требуемое для завершения цикла записи составляет 3.3 мс. Если в это время попытаться что-либо считать/записать в EEPROM, то такая попытка окончится неудачей. Однако, данная задержка уже учитывается библиотекой EEPROM, поэтому в дополнительном вызове `delay()` нет необходимости.

Практическое задание

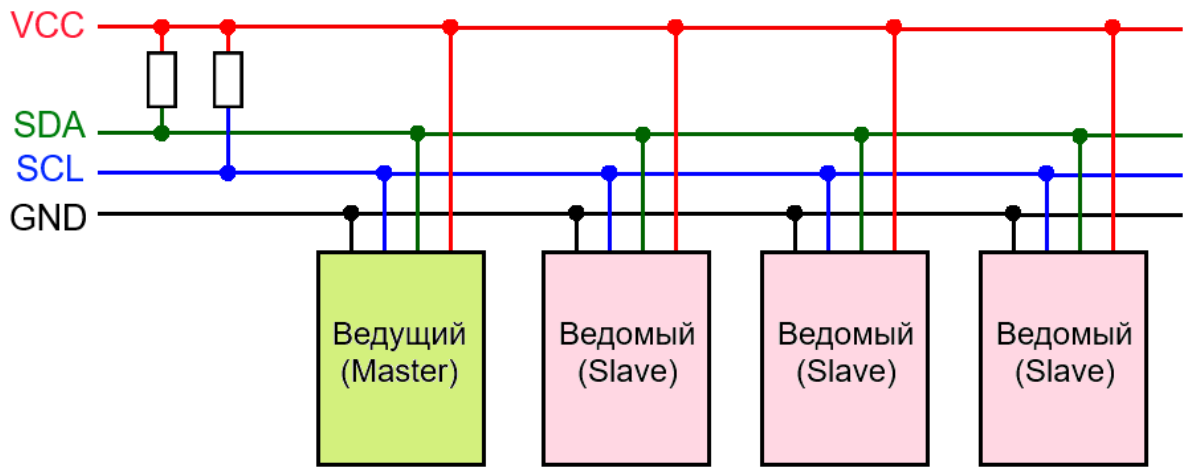
1. Разберитесь как работает программа EEPROM Clear из стандартного примера библиотеки EEPROM ARDUINO IDE.
2. Разработайте программное обеспечение робота (скорее всего это будут две программы), которое однократно записывает в память EEPROM основную информацию о вашем роботе: Торговое название (MIRO (Mobile Inteligent Robot)), Идентификатор (ID – придумайте самостоятельно), Серийный номер (SN придумайте самостоятельно), Тип установленного переднего модуля (FModule: SENS1), Тип установленного заднего модуля (BModule: default), Тип источника питания (Battery: Li-Ion 7,4V) и пр. (на ваше усмотрение). А затем считывает и выводит эту информацию в форматированном виде (каждый параметр – в отдельной строке) в терминал после включения питания или сброса.

В уроке использованы материалы:
http://robocraft.ru/blog/arduino/82.html

УРОК 10. ИНТЕРФЕЙС I2C (TWI)

Последовательный протокол обмена данными IIC (также называемый I2C – Inter-Integrated Circuits, межмикросхемное соединение) использует для передачи данных две двунаправленные линии связи, которые называются шина последовательных данных SDA (Serial Data) и шина тактирования SCL (Serial Clock). Также имеются две линии для питания. Шины SDA и SCL подтягиваются к шине питания через резисторы. В роботе MIRO интерфейс I2C применяется прежде всего для связи двух микропроцессорных плат: RobotDyn UNO+Wi-Fi (ARDUINO с интегрированным модулем WiFi на базе ESP8266) и RASPBERRY PI Pi Zero W.

В сети есть хотя бы одно ведущее устройство (Master), которое инициализирует передачу данных и генерирует сигналы синхронизации. В сети также есть ведомые устройства (Slave), которые передают данные по запросу ведущего. У каждого ведомого устройства есть уникальный адрес, по которому ведущий и обращается к нему. Адрес устройства указывается в техническом паспорте (Datasheet), либо может задаваться разработчиком этого устройства непосредственно в его управляющей программе. К одной шине I2C может быть подключено до 127 устройств, в том числе несколько ведущих. К шине можно подключать устройства в процессе работы, т.е. она поддерживает так называемое «горячее подключение».

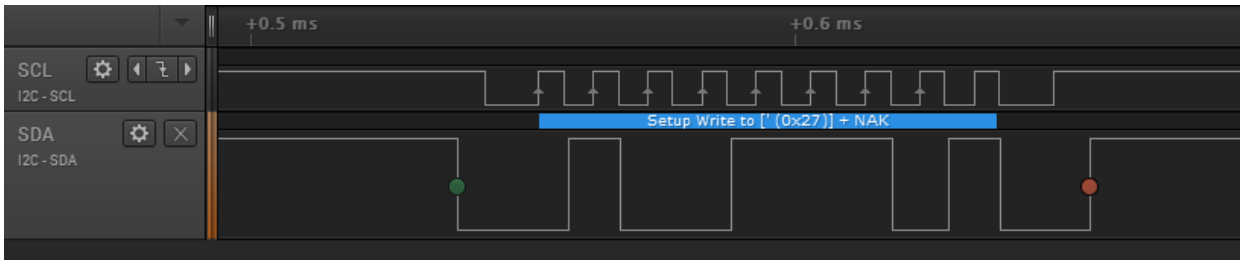


Давайте рассмотрим временную диаграмму обмена по протоколу I2C. Есть несколько различающихся вариантов, рассмотрим один из распространённых.

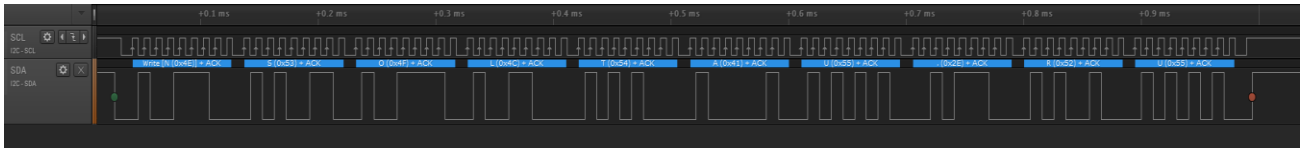
Мастер инициирует обмен. Для этого он начинает генерировать тактовые импульсы и посылает их по линии SCL пачкой из 9-ти штук. Одновременно на линии данных SDA он выставляет адрес устройства, с которым необходимо установить связь, которые тактируются первыми 7-ми тактовыми импульсами (отсюда ограничение на диапазон адресов: 2^7 = 128 минус нулевой адрес). Следующий бит посылки – это код операции (чтение или запись) и ещё один бит – бит подтверждения (ACK), что ведомое устройство приняло запрос. Если

бит подтверждения не пришёл, на этом обмен заканчивается. Или мастер продолжает посылать повторные запросы.

Это проиллюстрировано на рисунке ниже. Задача такая: подключиться к ведомому устройству с адресом 0x27 и передать ему строку "MIROROBOT.RU". В первом случае, для примера, отключим ведомое устройство от шины. Видно, что мастер пытается установить связь с устройством с адресом 0x27, но не получает подтверждения (NAK). Обмен заканчивается.



Теперь подключим к шине I2C ведомое устройство и повторим операцию. Ситуация изменилась. На первый пакет с адресом пришло подтверждение (ACK) от ведомого. Обмен продолжился. Информация передаётся также 9-битовыми посылками, но теперь 8 битов занимают данные и 1 бит – бит подтверждения получения ведомым каждого байта данных. Если в какой-то момент связь оборвётся и бит подтверждения не придёт, мастер прекратит передачу.



Микроконтроллеры ATmega имеют аппаратную поддержку интерфейса I2C (TWI). Линии интерфейса SDA и SCL у МК ATmega328, сидят на ножках с номерами 27 (PC4) и 28 (PC5), соответственно.

На платах ARDUINO, линия данных — SDA выведена на аналоговый пин 4 (A4), а линия тактирования — SCL выведена на аналоговый пин 5 (A5).

Для работы с протоколом I2C, у ARDUINO есть штатная библиотека Wire, которая позволяет взаимодействовать с I2C/TWI-устройствами, как в режиме ведущего, так и в режиме ведомого.

Библиотека Wire наследуется от Stream, что делает её использование схожим с другими библиотеками чтения/записи (read/write libraries). В связи с этим, методы `send()` и `receive()` были заменены на `read()` и `write()`.

Рассмотрим методы библиотеки.

Методы `void begin()`, `void begin(uint8_t address)`, `void begin(int address)` производит инициализацию библиотеки Wire и интерфейса I2C (TWI) микроконтроллера и подключение к шине в качестве ведущего или ведомого. Как правило, вызывается только один раз. Параметр address - 7-битный адрес

устройства (если работаем в режиме ведомого). Если не указано, то контроллер подключается к шине в роли ведущего (master).

Методо `uint8_t requestFrom(uint8_t address, uint8_t quantity)` используется мастером для запроса байта от ведомого устройства. Байты могут быть получены с помощью методов `available()` и `read()`. Параметр `address` все также задает 7-битный адрес устройства для запроса байтов данных, а параметр `quantity` - количество запрашиваемых байт.

Метод `void beginTransmission(uint8_t address)` инициирует начало передачи I2C для ведомого устройства с заданным адресом. Затем, нужно вызвать метод `write()` для добавления последовательности байт в очередь предназначенных для передачи, и выполнить саму передачу данных методом `endTransmission()`.

Метод `uint8_t endTransmission(void)` завершает передачу данных для ведомого устройства, которое было начато методом `beginTransmission()` и, фактически, осуществляет передачу байт, которые были поставлены в очередь методом `write()`. Метод `endTransmission` возвращает байт статуса передачи: 0 – успешная передача, 1 – данных слишком много и они не помещаются в буфер передачи, 2 - получен NACK на передачу адреса, 3 - получен NACK на передачу данных, 4 - другая ошибка.

Методы `size_t write(uint8_t data)` и `size_t write(const uint8_t *data, size_t quantity)` записывает данные от ведомого устройства в ответ на запрос ведущего, или записывает очередь байт для передачи от ведущего к ведомому устройству (в промежутках между вызовами `beginTransmission()` и `endTransmission()`). Параметры: `data` - байт или указатель на массив байт для передачи, `quantity` - число байт для передачи. Методы `write()` возвращают число переданных байт.

Метод `int available(void)` возвращает количество байт, доступных для получения. Этот метод должно быть вызван на стороне ведущего, после вызова `requestFrom()` или на стороне ведомого внутри обработчика `onReceive()`. Метод `available()` наследуется от класса `Stream` и возвращает число байт, доступное для чтения.

Методо `int read(void)` считывает байт, который был передан от ведомого устройства к ведущему, после вызова `requestFrom()` или был передан от ведущего к ведомому. Метод `read()` также наследуется от класса `Stream` и возвращает следующий полученный байт.

Метод `void onReceive( void (*function)(int) )` регистрирует функцию, которая вызывается, когда ведомое устройство получает данные от мастера. Параметры: `function` – указатель на функцию, которая вызывается, когда ведомый получает данные; обработчик должен принимать один параметр — `int` (число байт, считанных от ведущего) и ничего не возвращать.
Пример объявления функции обработчика:

```
void MyHandler (int numBytes);
```

Метод `void onRequest( void (*function)(void) )` регистрирует функцию, которая вызывается, когда ведущее устройство запрашивает данные от ведомого устройства.
Параметры: `function` – указатель на функцию, которая будет вызываться (функция не должна иметь параметров и не должна что-либо возвращать).

Пример объявления функции обработчика:

```
void MyHandler();
```

Код программы 1

```
#include <Wire.h>

byte val = 0;

void setup()
{
   Wire.begin();                    // подключение к шине i2c в качестве
ведущего
}

void loop()
{
   Wire.beginTransmission(44);      // передача для устройства #44 (0x2c)

   Wire.write(val);                 // отправка байта val
   Wire.endTransmission();          // передача данных

   val++;                           // инкремент значения
   if(val == 64)                    // если достигли 64 (max)
   {
     val = 0;                       // начинаем с минимального значения
   }
   delay(500);
}
```

Код программы 2

```
#include <Wire.h>

void setup()
{
   Wire.begin();         // подключение к шине i2c
   Serial.begin(115200);  // запуск последовательного порта
}

void loop()
{
   Wire.requestFrom(2, 6);     // запрос 6 байт от ведомого с адресом #2

   while(Wire.available())     // пока есть, что читать
   {
     char c = Wire.read();     // получаем байт (как символ)
     Serial.print(c);          // выводим в последовательный порт
   }
   delay(500);
}
```

В уроке использованы материалы:
https://soltau.ru/index.php/arduino/item/371-interfejs-i2c-i-arduino
http://robocraft.ru/blog/arduino/786.html