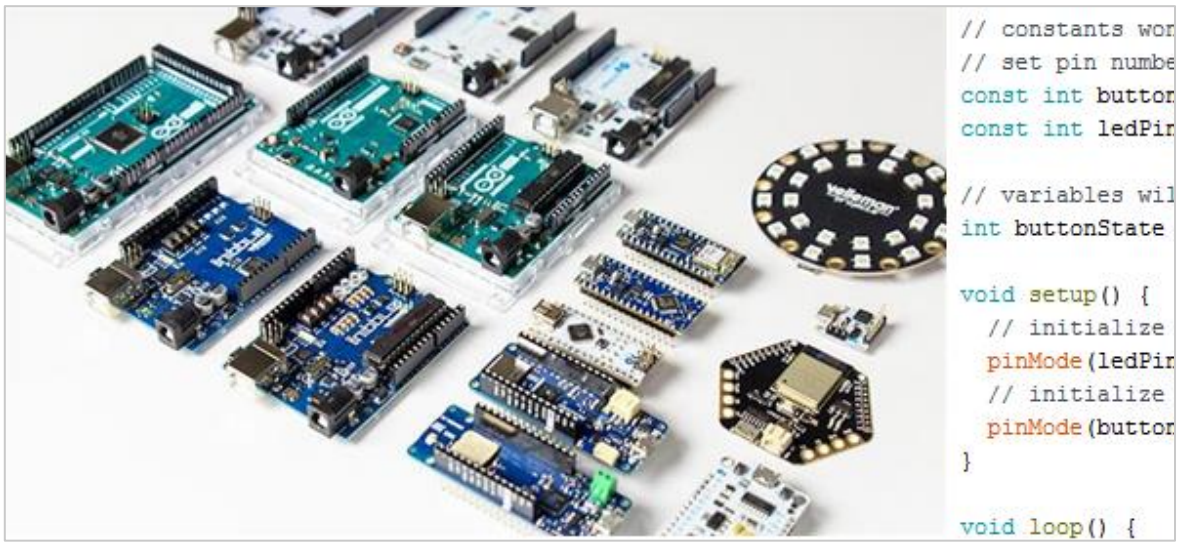
А.А. Лысиков

Изучение принципов работы микроконтроллеров в решениях IoT

Методические указания к лабораторной работе по дисциплине
«Технологии Интернета вещей»
По направлению подготовки
01.04.02 «Прикладная математика и информатика
(Науки о данных)»



Самара
2021

УДК
ББК

Лысиков А.А.

Изучение принципов работы микроконтроллеров в решениях IoT: методические указания к лабораторной работе по дисциплине «Технологии Интернета вещей» [Текст] / А.А. Лысиков – Самара: , 2021. – 27 с.

Методические указания к лабораторной работе предназначены для магистров, обучающихся по направлению 01.04.02 «Прикладная математика и информатика (Науки о данных)».

Содержание

1. Цель и задачи лабораторной работы

Целью лабораторной работы является формирование знаний о принципах работы контроллеров в решениях IoT, получение умений и навыков разработки программ управления датчиками и актуаторами для контроллеров в решениях IoT.

Задачами лабораторной работы являются изучение принципов работы контроллеров в решениях IoT на примере работы контроллера Arduino; получение навыков программирования контроллеров в решениях IoT на примере разработки программы управления датчиком и актуатором для контроллера Arduino.

2 . Порядок выполнения работы

1. Ознакомиться с целью, задачами и описанием лабораторной работы (п.1-3).
2. Ознакомиться с заданием лабораторной работы (п.4).
3. Согласно варианту из таблицы 1, соответствующему номеру студента в списке группы, выполнить задание.
4. Результаты выполнения задания оформить в виде отчета согласно правилам в п.5.
5. Ответить на контрольные вопросы для подготовки к устному отчету по результатам выполненной работы.
6. Отчитаться преподавателю устно о результатах выполненной работы. В ходе устного отчета преподаватель может задать ряд вопросов из списка п.6

3. Принципы функционирования микроконтроллеров IoT

3.1 Контроллеры в решениях IoT

В настоящее время большинство решений IoT на практике реализуются согласно схеме, приведенной на рисунке 1.

Устройства IoT («вещи») представляют из себя датчик или актуатор, совмещенные с контроллером зачастую в одном корпусе. При этом устройство IoT помимо котроллера может содержать одновременно как датчик, так и актуатор.

Датчик – это устройство, обеспечивающее преобразование сведений о внешней среде в машиночитаемые данные. По устройству датчики могут быть как простейшими устройствами измерения температуры, освещенности, влажности, основанными на физическом воздействии природных факторов (влажности, температуры и др.) на электронный компонент (резистор, конденсатор и др.), так и достаточно сложными измерительными приборами.

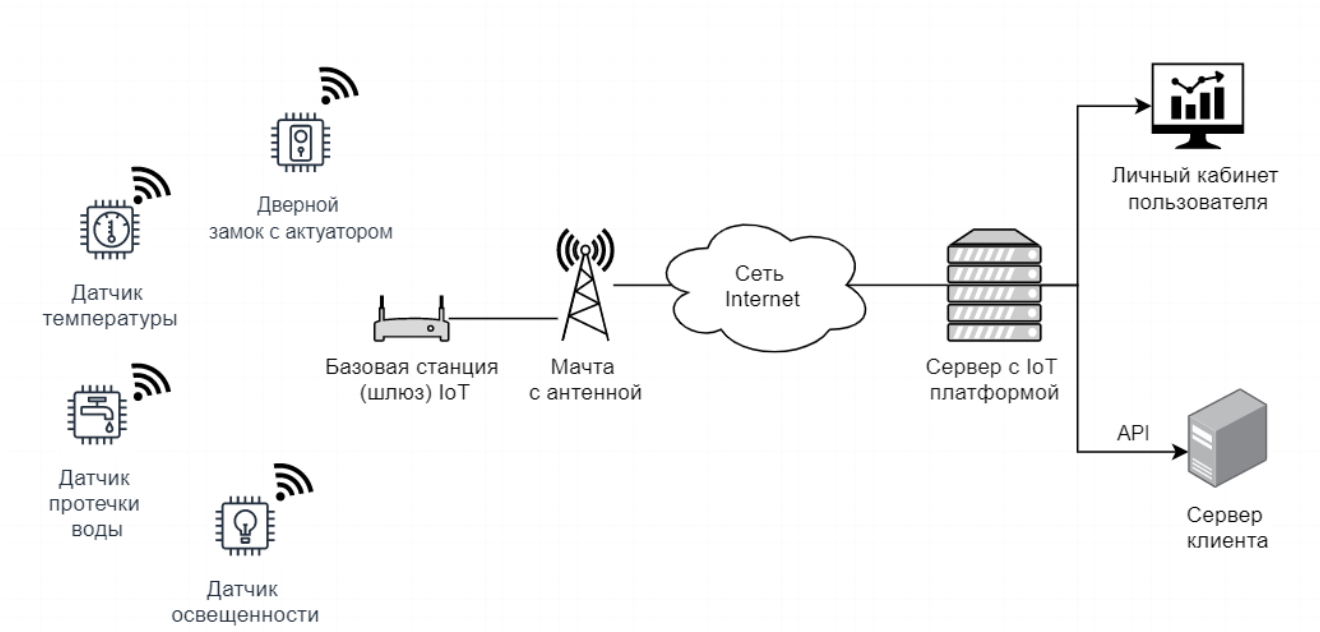


Рис. 1 Схема типового решения IoT

Актуатор – (или исполнительное устройство) это функциональный элемент системы автоматического управления, который воздействует на объект управления, изменяя поток энергии или материалов, которые поступают на объект. Актуатор может быть как простейшим устройством, таким как RGB-светодиод, так и сложным устройством автоматического управления, например, работающим в составе промышленного робота.

Таким образом, датчик – это устройство, которое считывает и отправляет информацию в контроллер, а актуатор — это устройство, которое принимает информацию из контроллера и изменяет свое состояние в зависимости от характера этой информации, выполняя таким образом полезные действия (например, вращает водопроводный кран, толкает каретку оконных штор и т.п.). Информация от датчика (или актуатора) к контроллеру и обратно может передаваться в аналоговом или цифровом виде.

Контроллеры могут подключаться к шлюзу проводным и (преимущественно) беспроводным способом. В настоящее время в решениях IoT могут быть использованы технологии беспроводного подключения, такие как 6LoWPAN, LoRA, ZigBee, WiFi, CIoT (EC-GSM-IoT, LTE Cat 0, LTE Cat 1, LTE Cat M1(eMTC), LTE Cat NB(NB-IoT). Данные стандарты различаются по скорости передачи данных, используемым диапазонам частот (лицензируемым и нелицензируемым), дальности связи, форматом передаваемых сообщений и др.

Существуют и другие схемы решений IoT. Например, шлюз может отсутствовать в схеме, и устройства («вещи») могут напрямую подключаться к сети Интернет. В таком случае, само устройство значительно усложняется, в связи с чем увеличивается его стоимость и стоимость решения в целом.

Внимание данной лабораторной работы сосредоточено в области функционирования контроллеров, датчиков и актуаторов. Следует отметить, что хотя рассматриваемые в данной работе контроллеры могут использоваться в решениях IoT, они зачастую не предназначены для использования в промышленных и коммерческих проектах IoT, так как не соответствуют критически важным для таких решений требованиям: надежности, расширяемости и безопасности. Однако принципы функционирования и разработки программ для любых контроллеров в целом схожи.

В данной лабораторной работе мы изучим принципы работы контроллеров и научимся основам разработки программ управления датчиками и актуаторами для контроллеров на примере работы прибора учета электроэнергии, содержащего контроллер, который считывает данные по току, вычисляет значение количества потребляемой электроэнергии и передает это значение в шлюз и/или облачную платформу IoT. Это одно из наиболее распространенных коммерческих решений, реализуемых современными компаниями на рынке IoT.

3.2 Устройство контроллеров

Микроконтроллер — это микрочип или плата с микрочипом для решения клиентских частей IoT-проектов. Некоторые проекты в IoT проще всего решить на микроконтроллерах. Они поддерживают множество стандартов ввода и вывода, работают с меньшим энергопотреблением и стоят дешевле по сравнению с микрокомпьютерами. Недостатком является меньшая вычислительная мощность и отсутствие операционной системы по умолчанию. Наиболее популярными микроконтроллерами являются Atmel, STM, Arduino, ESP8266, ESP32 и др.

Микрокомпьютер обычно представляет собой систему на чипе, включая классическую архитектуру фон Неймана с центральным процессором, видеокартой, оперативной памятью, модулями подключения к сетям связи и портами ввода-вывода. Современные микрокомпьютеры используют такие операционные системы, как Linux и Windows. Как правило, микрокомпьютеры имеют большую вычислительную мощность, чем микроконтроллеры, видеовыход на HDMI, высокоскоростной Wi-Fi и Bluetooth, подключение к картам флэш-памяти и M.2 и т.д. Недостатком микрокомпьютеров является более высокая цена и более высокое энергопотребление по сравнению с микроконтроллерами. Микрокомпьютеры используются в проектах IoT, если вам необходимо выполнять задачи высокого уровня, включая потоковое видео, сложные вычисления и т.п.

В данной лабораторной работе мы будем использовать модель контроллера Arduino Uno.

Arduino Uno – плата от компании Arduino, построенная на микроконтроллере ATmega 328. Плата имеет 6 аналоговых входов, 14 цифровых

выводов общего назначения (могут являться как входами, так и выходами), кварцевый генератор на 16 МГц, два разъема: силовой и USB, разъем ISCP для внутрисхемного программирования и кнопку горячей перезагрузки устройства. Для стабильной работы плату необходимо подключить к питанию либо через встроенный USB разъем, либо к источнику от 7 до 12В. Через переходник питания плата также может работать от батареи. На рисунке 2 представлены обозначения разъемов Arduino.

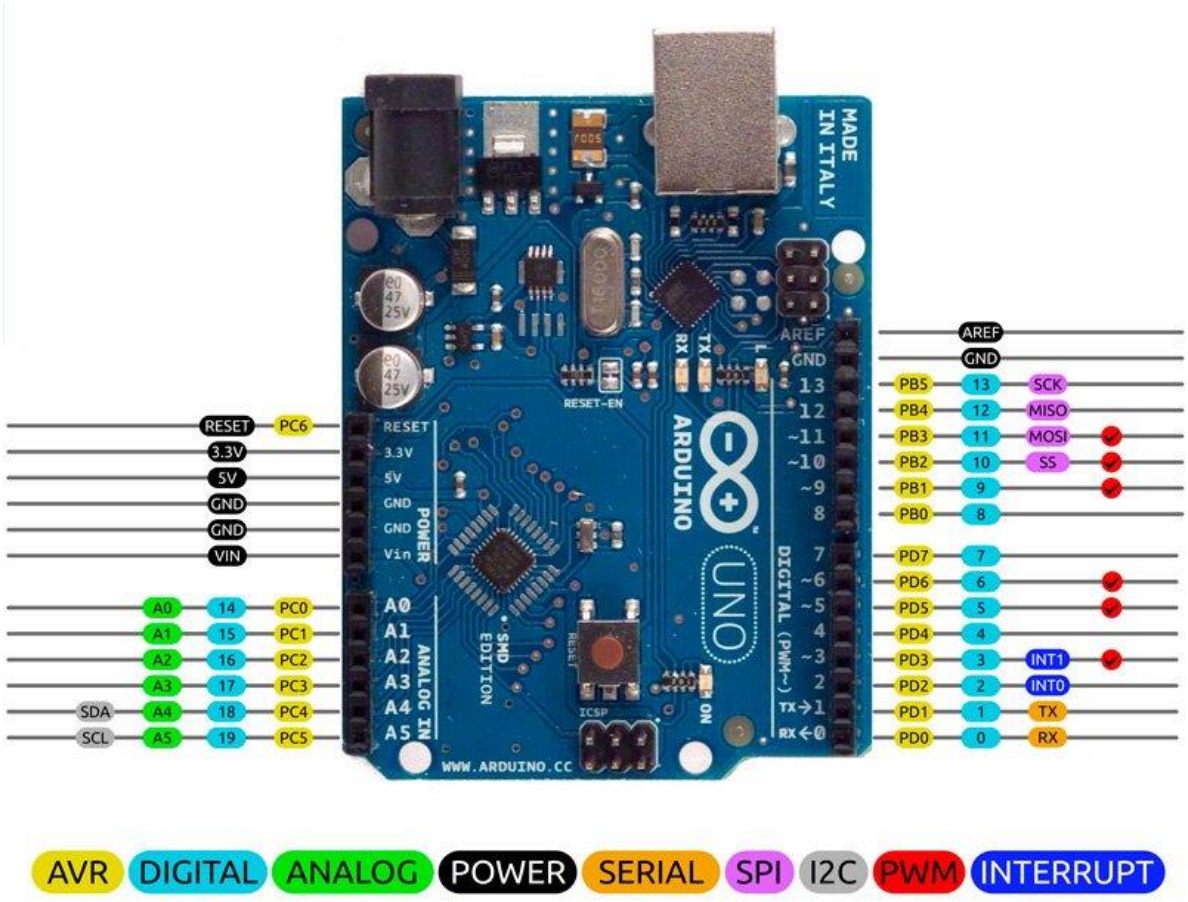


Рис. 2 Плата Arduino с обозначением разъёмов (пинов).

Плата Arduino Uno имеет 3 способа подключения питания: через USB, через внешний разъем питания и через разъем Vin. Плата имеет встроенный стабилизатор, позволяющий не только автоматически выбирать источник питания, но и выравнивать ток до стабильных 5 вольт, необходимых контроллеру для работы. Внешнее питание можно подавать как напрямую от USB порта компьютера, так и от любого AC/DC блока питания через разъем питания или USB.

На плате предусмотрено несколько выводов, позволяющих подключать к питанию датчики, сенсоры и актуаторы. Все эти выводы помечены:

Vin – вход питания, используется для получения питания от внешнего источника. Через данный вывод происходит только подача питания на плату, получить через него питание для внешних устройств невозможно. На вход Vin

рекомендуется подавать напряжение в диапазоне от 7В до 20В, во избежание перегрева и сгорания встроенного стабилизатора.

5V – источник напряжения для питания внешних устройств. При получении питания платой из любых других источников (USB, разъем питания или Vin) на этом контакте вы всегда сможете получить стабильное напряжение 5 вольт. Его можно вывести на макетную плату или подать напрямую на необходимое устройство.

3V3 – источник напряжения 3.3 вольта для питания внешних устройств. Работает по такому-же принципу, что и контакт 5V. Его также можно вывести напряжение на макетную плату, либо подать на необходимый датчик/сенсор напрямую.

GND – контакт для подключения земли. Необходим для создания замкнутой цепи при подключении к контактам Vin, 5V или 3V3. Во всех случаях ножку GND необходимо выводить как минус, иначе цепь не будет замкнута и питание (что внешнее, что внутреннее) не будет подаваться.

Платформа Arduino Uno имеет оснащена микроконтроллером ATmega328, который обладает следующими типами памяти:

- FLASH 32 Кбайт, из которых 0.5 Кбайт используется для хранения загрузчика;

- SRAM (ОЗУ) 2 Кбайт;

- EEPROM –1 Кбайт (доступна с помощью библиотеки EEPROM).

На плате выведены 14 цифровых пинов (контактов), любой из которых может работать как на вывод информации, так и на ввод. Для этого в коде программ применяются специальные функции:

- pinMode(), служит для задания режима работы контакта, будет-ли он работать на выход или на вход. В данной функции задается номер контакта, которым мы в дальнейшем собираемся управлять.

- digitalRead() считывает текущее значение с заданного контакта – его значение может быть HIGH или LOW.

- digitalWrite() передает определенное значение на заданный контакт – оно может быть HIGH или LOW.

Все выводы выдают логическую единицу как напряжение 5В. Каждый вывод платы имеет нагрузочный резистор номиналом 20-50 кОм и может пропускать до 40 мА, но по умолчанию все они отключены. Также, на контактных площадках Arduino Uno выведены специальные интерфейсы подключения различных цифровых устройств:

- Аналоговые входы A0, A1, A2, A3, A4, A5 с разрешением 10 Бит на каждый вход. Данное разрешение говорит о том, что сигнал, приходящий на него, можно оцифровать в диапазоне от 0 до 1024 условных значений. Считывать значения с данных контактов можно функцией analogRead(), а передавать значения – функцией analogWrite(). Значение напряжения на выходах будет находиться в

диапазоне от 0 до 5 вольт, однако при помощи функции analogReference() можно изменять верхний предел.

- Последовательный интерфейс UART: контакты 0 (RX) и 1 (TX). Данные выводы используются для обмена данными по протоколу UART. Контакт RX используется для получения данных, а контакт TX – для их отправки. Эти выводы подключены к соответствующим контактам последовательной шины схемы ATmega8U2 USB-to-TTL, выступающей в данном контексте в роли программатора.

- Внешнее прерывание: контакты 2 и 3. Данные контакты могут конфигурироваться на вызов различных прерываний, когда программа останавливает выполнение основного кода и производит выполнение кода прерывания. Вызов прерывания может быть задан по-разному: на младшем значении, на переднем или заднем фронте, при изменении значения.

- ШИМ: контакты 3, 5, 6, 9, 10, 11. Любой из этих контактов может генерировать сигнал широтно-импульсной модуляции (ШИМ, PWM) с разрешением 8 Бит. Для этого в коде программы используется функция analogWrite().

- SPI интерфейс: контакты 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). С помощью данных контактов происходит подключение периферии, работающей через интерфейс SPI. Для работы с данным интерфейсом в среде Arduino IDE предусмотрена отдельная библиотека с одноименным названием.

- I2C интерфейс: контакты 4 (SDA) и 5 (SCL). При помощи данных контактов к Arduino можно подключать внешние цифровые устройства, умеющие взаимодействовать по протоколу I2C. Для реализации интерфейса в среде Arduino IDE присутствует библиотека Wire.

- Встроенный светодиод: контакт 13. Для проверки вашего кода по ходу его написания, самый удобный способ индикации – встроенный светодиод. Подав значение HIGH на 13 контакт, он загорается на плате красным цветом, тем самым показывая, что условие вашей программы выполнилось (или наоборот, что-то пошло не так). 13 контакт удобно использовать в коде программы для проверки ошибок и отладки. Последовательно к 13-ому контакту подключен резистор на 220 Ом, поэтому не стоит использовать его для вывода питания ваших устройств.

- Дополнительные контакты: AREF и RESET. AREF отвечает за определение опорного напряжения аналоговых входов платформы. Используется только с функцией analogReference(). RESET необходим для аппаратной перезагрузки микроконтроллера. При подаче на него сигнала низкого уровня (LOW) происходит перезагрузка устройства. Данный контакт обычно соединен с аппаратной кнопкой перезагрузки, установленной на плате.

Для осуществления связи с внешними устройствами (компьютером и другими микроконтроллерами) на плате существует несколько дополнительных устройств. На контактах 0 (RX) и 1 (TX) контроллер ATmega328 поддерживает

UART – последовательный интерфейс передачи данных. ATmega8U2, выполняющий на плате роль программатора, транслирует этот интерфейс через USB, позволяя платформе общаться с компьютером через стандартный COM-порт. Прошивка, установленная в контроллер ATmega8U2, имеет стандартные драйверы USB-COM, поэтому для подключения не потребуется никаких дополнительных драйверов.

3.3 Интерфейсы передачи данных

Для передачи данных с контроллера в шлюз или между контроллерами могут использоваться проводные и беспроводные технологии. Для чего к соответствующим интерфейсам платы контроллера могут быть подключены медные проводники или радио модули.

Во многих решениях IoT предпочтение отдается беспроводному подключению ввиду удобства и экономичности (во многих проектах конечные устройства могут размещаться в труднодоступных местах – подвалах, за трубами и т.п., куда сложно протянуть кабели). В настоящее время в большинство контроллеров могут быть встроены или подключаться в качестве плат расширения различные модули беспроводной связи, начиная с простейших одночастотных RF-модулей и заканчивая продвинутыми Bluetooth и Wi-Fi модулями.

Передача данных с контроллера как правило производится через специальные интерфейсы, такие как I2C, UART, SPI. В данной лабораторной работе мы будем использовать интерфейс I2C.

I2C – последовательная асимметричная шина для связи между интегральными схемами внутри электронных приборов. Разработана фирмой Philips Semiconductors в начале 1980-х как простая 8-битная шина внутренней связи для создания управляющей электроники. Была рассчитана на частоту 100 кГц.

I2C использует две двунаправленные линии связи (SDA и SCL). Шина представляет собой два проводника, а для управления интерфейсом достаточно одного микроконтроллера. Шина позволяет производить отключение микросхем в процессе работы. Специальный встроенный фильтр способен справляться со всплесками, гарантируя сохранность обрабатываемой информации.

Недостатками I2C является ограниченная емкость шины (что приводит к ограничению по количеству подключаемых устройств, сложность программирования и трудности с определением неисправности в ситуации с состоянием низкого уровня.

Шина I2C стандартизована в 1992 году, в первой версии к стандартному режиму 100 кбит/с добавлен скоростной режим 400 кбит/с (Fast-mode, Fm); за счёт 10-битной адресации становится возможным подключение на одну шину

более 1000 устройств, количество которых ограничивается максимально допустимой ёмкостью шины — 400 пФ.

В стандарте версии 2.0 (1998 год) представлены высокоскоростной режим работы 3,4 Мбит/с (High-speed mode, Hs) и требования пониженного энергопотребления. Незначительно доработан в версии 2.1 (2000 год).

В версии 3 (2007 год) добавлен режим 1 Мбит/с (Fast-mode plus, Fm+) и механизм идентификации устройств (ID).

В версии 4 (2012 год) появился однонаправленный режим 5 Мбит/с (Ultra Fast-mode, UFm) с использованием двухтактной логики без подтягивающих резисторов, добавлена таблица предустановленных идентификаторов.

В версии 5 (2012 год) исправлены ошибки.

В версии 6 (2014 год) пересчитаны графики, определяющие величину подтягивающих резисторов в зависимости от ёмкости шины и рабочего напряжения.

Шина I2C синхронная, состоит из двух линий: данных (SDA) и тактов (SCL) (рис. 3). Есть ведущий (master) и ведомые (slave). Инициатором обмена всегда выступает ведущий, обмен между двумя ведомыми невозможен. Всего на одной двухпроводной шине может быть до 127 устройств. Такты на линии SCL генерирует master. Линией SDA могут управлять как мастер так и ведомый в зависимости от направления передачи. Единицей обмена информации является пакет, обрамленный уникальными условиями на шине, именуемыми стартовым и стоповым условиями. Мастер в начале каждого пакета передает один байт, где указывает адрес ведомого и направление передачи последующих данных. Данные передаются 8-битными словами. После каждого слова передается один бит подтверждения приема приемной стороной.

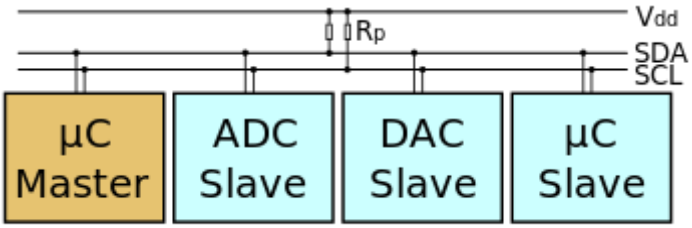


Рис. 3 Пример шины I2C

I²C использует две двунаправленные линии, подтянутые к напряжению питания и управляемые через открытый коллектор или открытый сток — последовательная линия данных (SDA, англ. Serial DAta) и последовательная линия тактирования (SCL, англ. Serial CLock). Стандартные напряжения +5 В или +3,3 В, однако допускаются и другие. Классическая адресация включает 7-битное адресное пространство с 16 зарезервированными адресами. Это означает, что разработчикам доступно до 112 свободных адресов для подключения периферии на одну шину. Основной режим работы — 100 кбит/с; 10 кбит/с в

режиме работы с пониженной скоростью. Также немаловажно, что стандарт допускает приостановку тактирования для работы с медленными устройствами.

Процедура обмена начинается с того, что ведущий формирует состояние СТАРТ: при ВЫСОКОМ уровне на линии SCL он генерирует переход сигнала линии SDA из ВЫСОКОГО состояния в НИЗКОЕ. Этот переход воспринимается всеми устройствами, подключенными к шине, как признак начала процедуры обмена. Генерация синхросигнала — это всегда обязанность ведущего; каждый ведущий генерирует свой собственный сигнал синхронизации при пересылке данных по шине.

При передаче посылок по шине I²C каждый ведущий генерирует свой синхросигнал на линии SCL. После формирования состояния СТАРТ ведущий опускает состояние линии SCL в НИЗКОЕ состояние и выставляет на линию SDA старший бит первого байта сообщения. Количество байт в сообщении не ограничено. Спецификация шины I²C разрешает изменения на линии SDA только при НИЗКОМ уровне сигнала на линии SCL. Данные действительны и должны оставаться стабильными только во время ВЫСОКОГО состояния синхроимпульса. Для подтверждения приёма байта от ведущего-передатчика ведомым-приёмником в спецификации протокола обмена по шине I²C вводится специальный бит подтверждения, выставляемый на шину SDA после приёма 8 бит данных.

Процедура обмена завершается тем, что ведущий формирует состояние СТОП — переход состояния линии SDA из НИЗКОГО состояния в ВЫСОКОЕ при ВЫСОКОМ состоянии линии SCL. Состояния СТАРТ и СТОП всегда вырабатываются ведущим. Считается, что шина занята после фиксации состояния СТАРТ. Шина считается освободившейся через некоторое время после фиксации состояния СТОП.

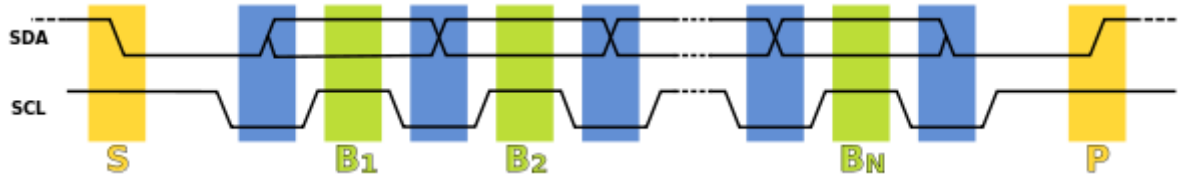


Рис. 4 Таксировка последовательности передачи данных I2C

Таким образом, передача 8 бит данных от передатчика к приёмнику завершаются дополнительным циклом (формированием 9-го тактового импульса линии SCL), при котором приёмник выставляет низкий уровень сигнала на линии SDA, как признак успешного приёма байта. Подтверждение при передаче данных обязательно, кроме случаев окончания передачи ведомой стороной. Соответствующий импульс синхронизации генерируется ведущим. Передатчик отпускает (переводит в ВЫСОКОЕ состояние) линию SDA на время синхроимпульса подтверждения. Приёмник должен удерживать линию SDA в

течение ВЫСОКОГО состояния синхроимпульса подтверждения в стабильном НИЗКОМ состоянии. В том случае, когда ведомый-приёмник не может подтвердить свой адрес (например, когда он выполняет в данный момент какие-либо функции реального времени), линия данных должна быть оставлена в ВЫСОКОМ состоянии. После этого ведущий может выдать состояние СТОП для прерывания пересылки данных. Если в пересылке участвует ведущий-приёмник, то он должен сообщить об окончании передачи ведомому-передатчику путём неподтверждения последнего байта. Ведомый-передатчик должен освободить линию данных для того, чтобы позволить ведущему выдать состояние СТОП или повторить состояние СТАРТ.

Синхронизация выполняется с использованием подключения к линии SCL по правилу монтажного И. Это означает, что ведущий не имеет монопольного права на управление переходом линии SCL из НИЗКОГО состояния в ВЫСОКОЕ. В том случае, когда ведомому необходимо дополнительное время на обработку принятого бита, он имеет возможность удерживать линию SCL в низком состоянии до момента готовности к приёму следующего бита. Таким образом, линия SCL будет находиться в НИЗКОМ состоянии на протяжении самого длинного НИЗКОГО периода синхросигналов.

Устройства с более коротким НИЗКИМ периодом будут входить в состояние ожидания на время, пока не кончится длинный период. Когда у всех задействованных устройств кончится НИЗКИЙ период синхросигнала, линия SCL перейдет в ВЫСОКОЕ состояние. Все устройства начнут проходить ВЫСОКИЙ период своих синхросигналов. Первое устройство, у которого кончится этот период, снова установит линию SCL в НИЗКОЕ состояние. Таким образом, НИЗКИЙ период синхролинии SCL определяется наидлиннейшим периодом синхронизации из всех задействованных устройств, а ВЫСОКИЙ период определяется самым коротким периодом синхронизации устройств.

Механизм синхронизации может быть использован приёмниками как средство управления пересылкой данных на байтовом и битовом уровнях.

На уровне байта, если устройство может принимать байты данных с большой скоростью, но требует определенное время для сохранения принятого байта или подготовки к приёму следующего, то оно может удерживать линию SCL в НИЗКОМ состоянии после приёма и подтверждения байта, переводя таким образом передатчик в состояние ожидания.

На уровне битов устройство, такое, как микроконтроллер без встроенных аппаратных цепей I²C или с ограниченными цепями, может замедлить частоту синхроимпульсов путём продления их НИЗКОГО периода. Таким образом скорость передачи любого ведущего адаптируется к скорости медленного устройства.

Каждое устройство, подключённое к шине, может быть программно адресовано по уникальному адресу. Для выбора приёмника сообщения ведущий

использует уникальную адресную компоненту в формате посылки. При использовании однотипных устройств ИС часто имеют дополнительный селектор адреса, который может быть реализован как в виде дополнительных цифровых входов селектора адреса, так и в виде аналогового входа. При этом адреса таких однотипных устройств оказываются разнесены в адресном пространстве устройств, подключенных к шине. В обычном режиме используется 7-битная адресация.

Процедура адресации на шине I²C заключается в том, что первый байт после сигнала СТАРТ определяет, какой ведомый адресуется ведущим для проведения цикла обмена. Исключение составляет адрес «Общего вызова», который адресует все устройства на шине. Когда используется этот адрес, все устройства в теории должны послать сигнал подтверждения. Однако устройства, которые могут обрабатывать «общий вызов», на практике встречаются редко.

Первые семь битов первых двух байтов образуют адрес ведомого. Восьмой, младший бит, определяет направление пересылки данных. «Ноль» означает, что ведущий будет передавать информацию выбранному ведомому. «Единица» означает, что ведущий будет получать информацию от ведомого.

После того, как адрес послан, каждое устройство в системе сравнивает первые семь бит после сигнала СТАРТ со своим адресом. При совпадении устройство полагает себя выбранным как ведомый-приёмник или как ведомый-передатчик, в зависимости от бита направления.

Адрес ведомого может состоять из фиксированной и программируемой части. Часто случается, что в системе имеется несколько однотипных устройств (к примеру, ИМС памяти, или драйверов светодиодных индикаторов), поэтому при помощи программируемой части адреса становится возможным подключить к шине максимально возможное количество таких устройств. Количество программируемых битов в адресе зависит от количества свободных выводов микросхемы. Иногда используется один вывод с аналоговой установкой программируемого диапазона адресов[2]. При этом в зависимости от потенциала на этом адресном выводе ИМС, возможно смещение адресного пространства драйвера так, чтобы однотипные ИМС не конфликтовали между собой на общей шине.

Все специализированные ИМС, поддерживающие работу в стандарте шины I²C, имеют набор фиксированных адресов, перечень которых указан производителем в описаниях контроллеров.
Комбинация бит 11110XX адреса зарезервирована для 10-битной адресации.

Как следует из спецификации шины, допускаются как простые форматы обмена, так и комбинированные, когда в промежутке от состояния СТАРТ до состояния СТОП ведущий и ведомый могут выступать и как приёмник, и как передатчик данных. Комбинированные форматы могут быть использованы, например, для управления последовательной памятью.

Во время первого байта данных можно передавать адрес в памяти, который записывается во внутренний регистр-защёлку. После повторения сигнала СТАРТа и адреса ведомого выдаются данные из памяти. Все решения об авто-инкременте или декременте адреса, к которому произошёл предыдущий доступ, принимаются конструктором конкретного устройства. Поэтому в любом случае лучший способ избежать неконтролируемой ситуации на шине перед использованием новой (или ранее не используемой) ИМС — следует тщательно изучить паспорт изделия или справочное руководство.

В любом случае по спецификации шины все разрабатываемые устройства должны сбрасывать логику шины при получении сигнала СТАРТ или повторный СТАРТ и подготавливаться к приёму адреса. Тем не менее, основные проблемы с использованием I²C шины возникают именно из-за того, что разработчики, «начинающие» работать с I²C шиной, не учитывают того факта, что ведущий (часто — микропроцессор) не имеет монопольного права ни на одну из линий шины.

3.4 Разработка программ для контроллеров

Обычно программировать микроконтроллеры можно с помощью языков программирования высокого уровня, таких как C, C ++, Python и т.д. В данной лабораторной работе используются модели контроллера Arduino, поэтому для разработки программ мы будет использовать специальный язык Arduino C.

Язык программирования Arduino C представляет собой язык C++ с фреймворком Wiring [3,4], Он имеет некоторые отличия по части написания кода, который компилируется и собирается с помощью avr-gcc, с особенностями, облегчающими написание работающей программы — имеется набор библиотек, включающий в себя функции и объекты. При компиляции программы IDE создает временный файл с расширением *.cpp.

Программы, написанные программистом Arduino, называются наброски или скетчи (транслитерация от англ. sketch) и сохраняются в файлах с расширением *.ino. Эти файлы перед компиляцией обрабатываются препроцессором Arduino. Также существует возможность создавать и подключать к проекту стандартные файлы C++.

Программист должен написать две обязательные для Arduino функции setup() и loop(). Первая вызывается однократно при старте, вторая выполняется в бесконечном цикле.

В текст своей программы (скетча) программист не обязан вставлять заголовочные файлы используемых стандартных библиотек. Эти заголовочные файлы добавит препроцессор Arduino в соответствии с конфигурацией проекта. Однако пользовательские библиотеки нужно указывать.

Менеджер проекта Arduino IDE имеет нестандартный механизм добавления библиотек. Библиотеки в виде исходных текстов на стандартном C++

добавляются в специальную папку в рабочем каталоге IDE. При этом название библиотеки добавляется в список библиотек в меню IDE. Программист отмечает нужные библиотеки, и они вносятся в список компиляции.

Arduino IDE не предлагает никаких настроек компилятора и минимизирует другие настройки, что упрощает начало работы для новичков и уменьшает риск возникновения проблем; но присутствуют директивы препроцессора, такие как #define, #include, и много других.

На рисунке 3 представлен полный текст простейшей программы (скетча) мигания светодиодом, подключенного к 13 выводу (пину) Arduino, с периодом 2 секунды (полпериода, то есть 1 секунду светодиод горит, полпериода — не горит).

Все используемые в этом примере функции являются библиотечными. В комплекте Arduino IDE имеется множество встроенных примеров программ. Существует перевод документации по Arduino на русский язык[1][6].

```
void setup () {
  pinMode (13, OUTPUT); // Назначение порта 13 в качестве выходного порта
}

void loop () {
  digitalWrite (13, HIGH); // Установка порта 13 в состояние "1", светодиод загорается
  delay (1000); // Задержка на 1000 миллисекунд
  digitalWrite (13, LOW); // Установка порта 13 в состояние "0", светодиод гаснет
  delay (1000); // Задержка на 1000 миллисекунд
}
```

Рис. 5 Пример программы Arduino для управления актуатором (светодиодом)

3.5 Среда моделирования Tinkercad

Для эмуляции контроллера Arduino и симуляции его работы с внешними устройствами мы будем использовать среду моделирования Tinkercad Circuits Arduino.

Tinkercad Circuits Arduino – это бесплатный, простой и одновременно мощный онлайн-эмулятор Arduino, предоставляющий удобную среду для написания различных проектов**.**

Для начала работы необходимо получить аккаунт Autocad. Регистрация в Tinkercad бесплатная. Зайдите на сайт https://www.tinkercad.com/ [2] и выполните простые шаги регистрации.

После регистрации зайдите в свою учетную запись и в меню слева выберите вкладку «Цепи» («Circuits») (рис. 6). После этого нажмите кнопку «Создать цепь». Вы перейдете на главный экран Tinkercad Circuits Arduino (рис. 7). Для моделирования контроллера Arduino Uno кликните по соответствующему элементу из списка в меню справа (рис. 7) и переместив курсор в рабочую область кликните по ней левой клавишей мыши.

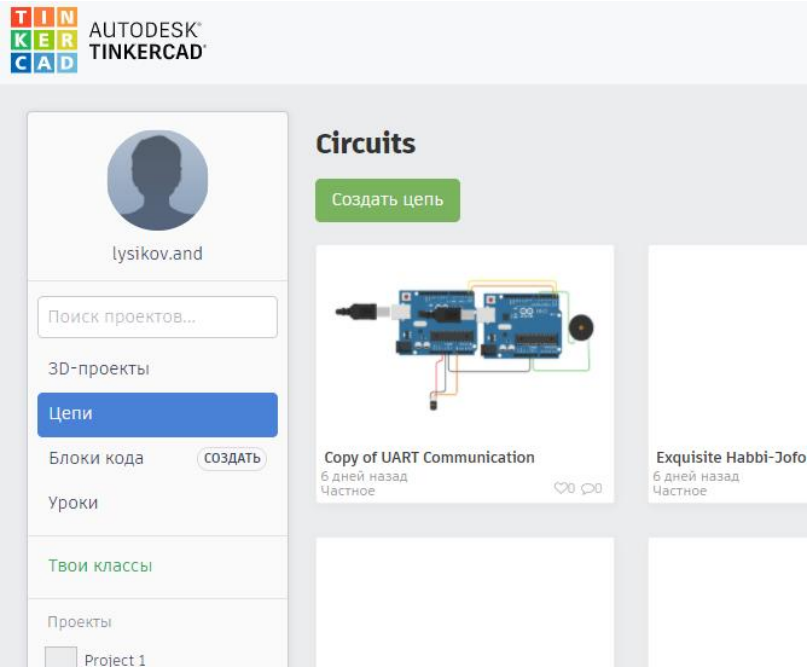


Рис. 6 Меню главного экрана Tinkercad

Для создания цепи выберите необходимые элементы и выполните действия, аналогичные добавлению контроллера. После этого вы можете соединить элементы с помощью проводников, подкачаю проводники к ножкам электронных компонентов и контактам контроллеров либо соедините их через макетную плату.

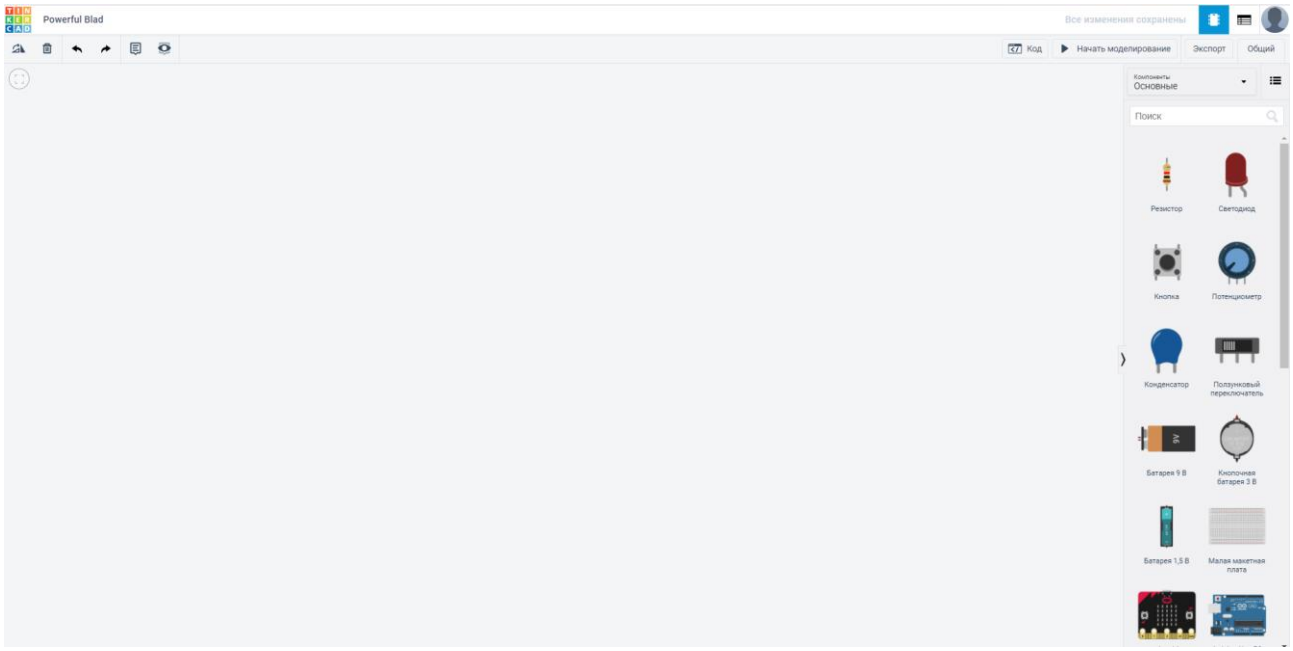


Рис. 7 Главный экран Tinkercad Circuits Arduino

Для написания кода кликните по кнопке «Код». Справа на главном экране откроется панель редактирования кода (рис. 8). Если в вашем проекте несколько контроллеров, то выбор контроллера для программирования осуществляется

нажатием по соответствующей кнопке в меню справа (на рис. 8 выделено в красный овал). Так, на рис. 8 приведен код для 2-го котроллера в проекте. Для запуска написанной программы нужно нажать кнопку «Начать моделирование».

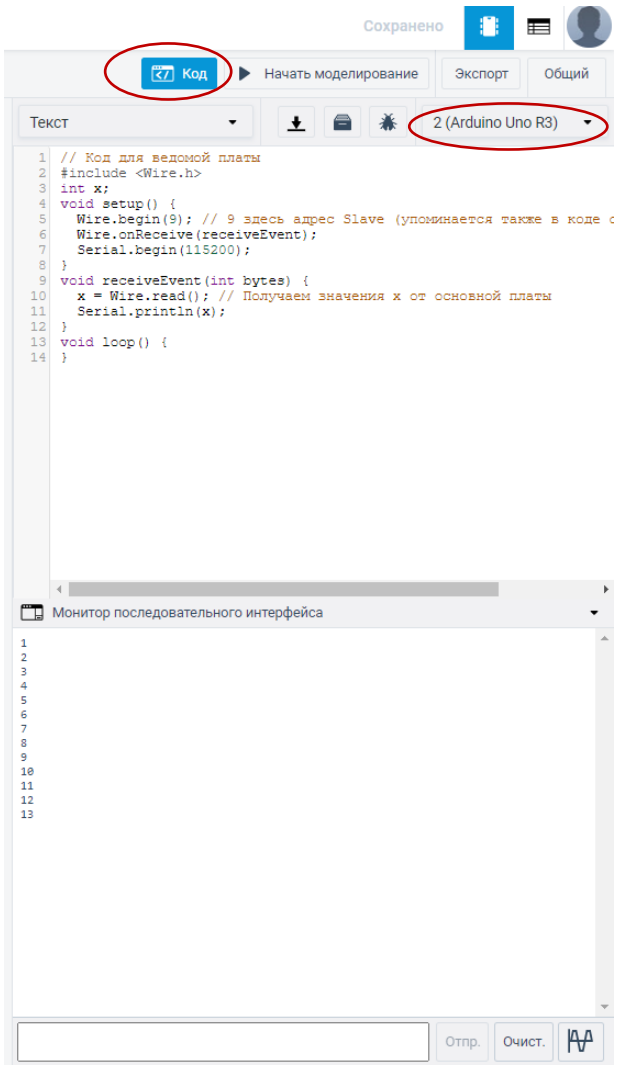


Рис. 8 Редактор кода и монитор вывода данных

Если в вашем коде предусмотрен вывод данных в последовательный интерфейс контроллера, то результат можно наблюдать на панели справа снизу «Монитор последовательного интерфейса» (рис. 8). Если по умолчанию он закрыт, то кликните по соответствующей кнопке в нижней части панели справа. Монитор последовательного интерфейса удобно использовать для проверки работоспособности вашего кода, добавив в код соответствующие функции. Также здесь отображаются ошибки, возникающие при компиляции кода.

Проекты в Tinkercad сохраняются автоматически. Для изменения масштаба отображения компонентов проекта наведите курсор мыши на рабочую область и крутите колесиком мыши вперед или назад для нужного масштаба. Для перемещения схемы нажмите и удерживая левую клавишу мыши перемещайте курсор на нужное положение в рабочей области.

4. Задание

4.1 Модель счетчика электроэнергии

Счетчик электроэнергии мы будем моделировать с помощью контроллера Arduino, который будет считывать параметры тока, получаемого путем прохождения сигнала от генератора через набор электронных компонентов, которые моделируют нагрузку электроприборов. Для моделирования экрана счетчика мы будем использовать модель ЖК дисплея. Для удобства соединения компонентов используем макетную плату, доступную в меню компонентов главного экрана справа.

Создайте новый проект в Tinkercad Circuits Arduino. Соберите счетчик электроэнергии по предложенной схеме на рисунке 9. Параметры электронных компонентов и генератора сигнала представлены в таблице 1. В данной модели датчиком является сам контроллер, который вычисляет потребляемую электроэнергию, актуатором – ЖК дисплей.

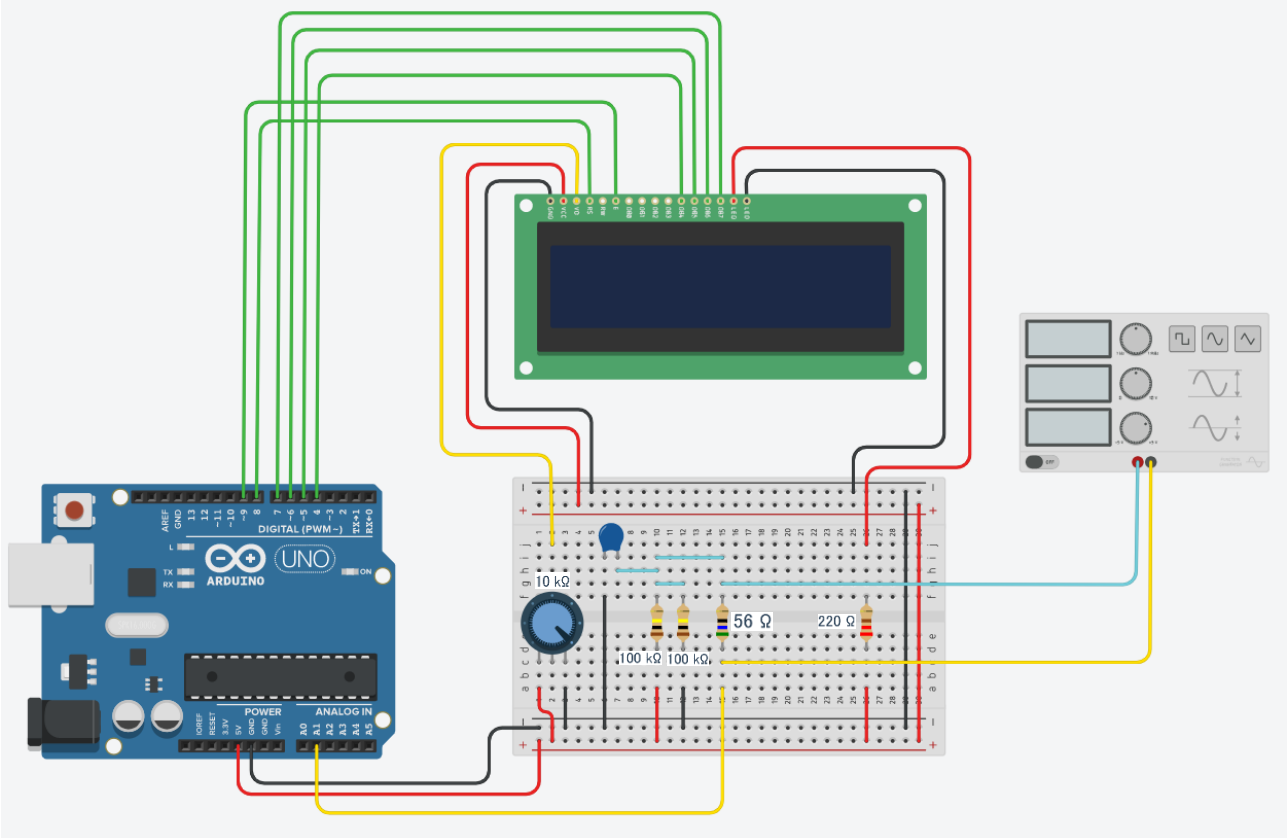


Рис. 9 Модель счетчика электроэнергии с контроллером Arduino

Таблица 1. Перечень компонентов для модели счетчика электроэнергии

Компонент	Количество	Параметр	Значение параметра
ЖК-дисплей (16 x 2)	1	-	-
Генератор сигнала	1	Частота	15,1 Гц
		Напряжение	10 В
		Форма сигнала	синусоидальная
		Сдвиг постоянной составляющей	-5В
Резистор	1	Сопротивление	
Резистор	2	Сопротивление	100 кОм
Резистор	1	Сопротивление	200 Ом
Резистор	1	Сопротивление	56 Ом
Конденсатор	1	Емкость	10 мкФ
Потенциометр	1	Сопротивление	10 кОм

4.2 Программа управления датчиком и актуаторами

Для работы модели счетчика электроэнергии необходимо написать программу управления для контроллера. Напишите предложенный программный код в контроллер, моделирующий счетчик электроэнергии. Проверьте работоспособность кода.

```
#include <Wire.h>
#include <LiquidCrystal.h>
#include <IRremote.h>
#include <EEPROM.h>
#include <LiquidCrystal.h>

int x = 0;
int currentPin = 1;             //Assign CT input to pin 1
double kilos = 0;
int peakPower = 0;
unsigned long startMillis;
unsigned long endMillis;
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);  // Assign LCD screen pins, as per LCD shield
requirements
```

```
void setup()
{
  Wire.begin();
  lcd.begin(16,2);              // columns, rows.  use 16,2 for a 16x2 LCD, etc.
  lcd.clear();
  lcd.setCursor(0,0);           // set cursor to column 0, row 0 (the first row)
  lcd.print("Arduino");
  lcd.setCursor(0,1);           // set cursor to column 0, row 1 (the second row)
  lcd.print("Energy Meter");
  startMillis = millis();
}

void loop()
{
  int current = 0;
  int maxCurrent = 0;
  int minCurrent = 1000;
  for (int i=0 ; i<=50; i++)  //Monitors and logs the current input for 200 cycles to
determine max and min current
  {
    current = analogRead(currentPin);    //Reads current input and records maximum
and minimum current
    if(current >= maxCurrent)
      maxCurrent = current;
    else if(current <= minCurrent)
      minCurrent = current;
  }
  if (maxCurrent <= 517)
  {
    maxCurrent = 516;
  }
  double RMSCurrent = ((maxCurrent - 516)*0.707)/11.8337;    //Calculates RMS
current based on maximum value
  int RMSPower = 220*RMSCurrent;    //Calculates RMS Power Assuming Voltage
220VAC, change to 110VAC accordingly
  if (RMSPower > peakPower)
  {
    peakPower = RMSPower;
  }
  endMillis = millis();
```

```
  unsigned long time = endMillis - startMillis;
  kilos = kilos + ((double)RMSPower * ((double)time/60/60/1000000));    //Calculate
kilowatt hours used
  startMillis = millis();
  delay (2000);
  lcd.clear();
  lcd.setCursor(0,0);          // Displays all current data
  lcd.print(RMSCurrent);
  lcd.print("A");
  lcd.setCursor(10,0);
  lcd.print(RMSPower);
  lcd.print("W");
  lcd.setCursor(0,1);
  lcd.print(kilos);
  lcd.print("kWh");
  lcd.setCursor(10,1);
  lcd.print(peakPower);
  lcd.print("W");
}
```

4.3 Модель подключения контроллера IoT к шлюзу IoT

На рисунке 10 представлена модель подключения двух контроллеров Arduino по шине I2C. Используя данный пример, подключите второй контроллер Arduino который будет моделировать шлюз IoT к модели контроллера счетчика электроэнергии из п 4.1

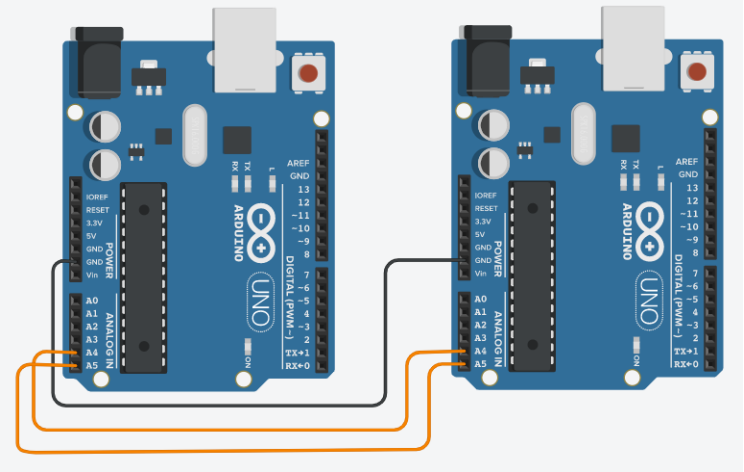


Рис. 10 Модель подключения двух плат Arduino по интерфейсу I2C

4.4 Программа коммуникации между двумя контроллерами

Используя примеры программного кода управления передачей данных между двумя контролерами по шине I2C, приведенные в п.п. 4.3.1 и 4.3.2, напишите аналогичные программы для контроллеров счетчика электроэнергии и второго контроллера (п.п.4.1 и 4.2.). При этом программу для контроллера счетчика электроэнергии вам нужно просто дополнить необходимыми функциями аналогично примеру в п. 4.3.1.

Таким образом, контроллер счетчика электроэнергии должен передавать значения потребляемой электроэнергии на второй контроллер, где они должны будут выводиться в последовательный интерфейс второго контроллера.

Подумайте, как реализовать передачу значений потребляемой электроэнергии в виде отображаемых на ЖК дисплее десятичных дробей, ведь переменная, используемая в примерах, позволяет хранить только целые числа (с учетом типов данных, поддерживаемых для передачи по интерфейсу I2C)?

Все параметры выбираются согласно таблице 2, где номер варианта равен номеру студента в списке.

4.4.1 Ведущая плата

```
// Код для Основной платы
#include <Wire.h>                  // библиотека I2C

int x;

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

void loop()
{
 x++;
  Wire.beginTransmission(9);       // Цифра 9 - адрес ведомой платы
  Wire.write(x);                   // Передает значение x
  Wire.endTransmission();
  Serial.println(x);
  delay(1000);
}
```

4.4.2 Ведомая плата

```
// Код для ведомой платы
#include <Wire.h>
int x;
void setup() {
  Wire.begin(9); // 9 здесь адрес Slave (упоминается также в коде основной платы)
  Wire.onReceive(receiveEvent);
  Serial.begin(115200);
}
void receiveEvent(int bytes) {
  x = Wire.read(); // Получаем значения x от основной платы
  Serial.println(x);
}
void loop() {
}
```

Таблица 2. Варианты задания

Параметр / Номер варианта (номер студента в списке)	Напряжение (В)	Частота переменного тока (Гц)	Количество циклов регистрации максимального и минимального значения тока	Задержка вывода значений на дисплей счетчика электроэнергии (мкс)	Скорость передачи данных (Кбит/с)	Адрес ведомой платы контроллера	Задержка отправки данных (мкс)	Задержка приема данных (мкс)
1	220	50	50	1000	10	1	1000	1000
2	230	50	100	1010	100	2	1010	1010
3	240	50	150	1020	400	3	1020	1020
4	100	60	200	1030	1000	4	1030	1030
5	110	60	250	1040	3400	5	1040	1040
6	115	60	300	1050	5000	6	1050	1050
7	120	60	350	1060	100	7	1060	1060
8	127	60	400	1070	10	8	1070	1070
9	220	60	450	1080	400	9	1080	1080
10	230	60	500	1090	5000	10	1090	1090
11	240	60	550	1100	3400	11	1100	1100
12	100	50	600	1110	100	12	1110	1110
13	110	50	650	1120	5000	13	1120	1120
14	115	50	700	1130	10	14	1130	1130
15	127	50	750	1140	1000	15	1140	1140
16	220	50	800	1150	100	16	1150	1150

Параметр / Номер варианта (номер студента в списке)	Напряжение (В)	Частота переменного тока (Гц)	Количество циклов регистрации максимального и минимального значения тока	Задержка вывода значений на дисплей счетчика электроэнергии (мкс)	Скорость передачи данных (Кбит/с)	Адрес ведомой платы контроллера	Задержка отправки данных (мкс)	Задержка приема данных (мкс)
17	230	50	850	1160	400	17	1160	1160
18	240	50	900	1170	3400	18	1170	1170
19	100	60	950	1180	1000	19	1180	1180
20	110	60	1000	1190	10	20	1190	1190
21	115	60	1050	1200	100	21	1200	1200
22	120	60	1100	1210	400	22	1210	1210
23	127	60	1150	1220	1000	23	1220	1220
24	220	60	1200	1230	3400	24	1230	1230
25	127	50	1250	1240	5000	25	1240	1240

5 Правила оформления отчета

Отчет по выполненной лабораторной работе должен содержать:

1. Титульный лист с указанием названия лабораторной работы, ФИО студента, дату выполнения.

2. Кратко описать суть проделанной работы: что было задано, что конкретно было выполнено.

3. Программные коды ведущей и ведомой платы Arduino.

4. Краткое описание использованных в программах библиотек, назначение использованных функций, переменных и т.д.(для ответа на вопрос пользуйтесь списком приведенных источников в п.8).

5. Ссылку на готовый проект в Tinkercad (для получения ссылки в главном окне Tinkercad Circuits Arduino нажмите кнопку «Общий» в меню справа сверху (см. рис. 7 или 8)).

6 Контрольные вопросы

1. Какие цели и задачи ставились в данной лабораторной работе?
2. Охарактеризуйте применение контроллеров в решениях IoT.
3. Поясните, что такое датчик и что такое актуатор. В чем различие между этими устройствами?
4. Как выглядит схема типового решение IoT?
5. Какие технологии подключения «вещей» вы знаете? В чем основное различие между стандартами подключения?
6. На какой части решений IoT сосредоточена данная лабораторная работа?
7. Приведите примеры современных наиболее распространенных решений IoT.
8. Охарактеризуйте устройство современных контроллеров IoT.
9. Расскажите, какой контроллер использовался в данной работе и его устройство. Какие входы/выходы имеет этот контроллер?
10. Расскажите об основных интерфейсах передачи данных, которые есть у рассматриваемого в данной работе контроллера?
11. Опишите принципы работы интерфейса I2C. Какие стандартные скорости передачи данных он поддерживает и за чет чего реализуется определенная скорость передачи данных?
12. Как подключить два контролера для взаимодействия по интерфейсу I2C?
13. С помощью каких языков программирования можно программировать современные контроллеры IoT?
14. Опишите язык, используемый для программирования контроллеров в данной работе.
15. Расскажите о программе для ведущей платы контроллера.
16. Расскажите о программе для ведомой платы контроллера.

17. Чем различались варианты задания?
18. Какое устройство IoT моделировалось с помощью ведомой платы контроллера Arduino?

7 Порядок устного отчета о результатах выполненной работы преподавателю

Для устного отчета нужно:

1. Выполнить работу по предложенному выше заданию согласно варианту.

2. По результатам работы оформить отчет согласно правилам представленным выше.

3. Изучить вопросы в п.6 и постараться дать краткий устный ответ на все предложенные вопросы (для ответа достаточно использовать материалы данных методических указаний).

8 Учебная литература и дополнительные источники

1. https://arduinomaster.ru/
2. https://www.tinkercad.com/
3. https://www.arduino.cc/reference/en/
4. https://www.arduino.cc/
5. Росляков А.В. Интернет вещей / А.В. Росляков, С.В. Ваняшин, А.Ю. Гребешков, учебное пособие, 2015 г. 200 с.
6. https://play.google.com/store/apps/details?id=com.arduino_hb.Arduino_HandBook_FREE