МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ

ГОУ ВПО ЛНР «ЛУГАНСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ ИМЕНИ ТАРАСА ШЕВЧЕНКО»

Р.Н. Сентяй
В.Н. Шишлакова

Архитектура ЭВМ и микроконтроллеров

Учебно-методические рекомендации по выполнению лабораторных работ
для студентов очной формы обучения
по направлению подготовки
09.03.01 «Информатика и вычислительная техника»

Луганск
КНИТА
2018

УДК 004.2(076.5)
ББК 32.973.26-02р3
С31

Рецензенты:

Комаров Н.В. **–** доцент кафедры микро- и наноэлектроники ГОУ ВПО ЛНР «Луганский государственный университет имени Владимира Даля», кандидат технических наук, доцент;

Остапущенко Д.Л. – и.о. заведующего кафедрой теоретической и прикладной информатики ГОУ ВПО ЛНР «Луганский национальный университет имени Тараса Шевченко», кандидат технических наук;

Мальцев Я.И. **–** доцент кафедры информационных технологий и систем ГОУ ВПО ЛНР «Луганский национальный университет имени Тараса Шевченко», кандидат технических наук, доцент.

С 31 Сентяй Р.Н., Шишлакова В.Н.
Архитектура ЭВМ и микроконтроллеров : учебно-методические рекомендации по выполнению лабораторных работ / Р.Н. Сентяй, В.Н. Шишлакова; ГОУ ВПО ЛНР «Луганский национальный университет имени Тараса Шевченко». – Луганск : Книта, 2018. – 136 с.

Данные методические рекомендации содержат цикл лабораторных работ, позволяющих освоить программирование микроконтроллеров Arduino. Каждая лабораторная работа содержит теоретические сведения, задания для практического выполнения и контрольные вопросы по соответствующей теме, предназначенные для закрепления полученных теоретических и практических знаний.

Учебно-методические рекомендации по выполнению лабораторных работ разработаны для студентов 2 курса очной формы обучения направления подготовки 09.03.01 «Информатика и вычислительная техника» и предназначены для выполнения лабораторных работ в рамках дисциплины «Архитектура ЭВМ и микроконтроллеров».

УДК 004.2(076.5)
ББК 32.973.26-02р3

Рекомендовано Учебно-методическим советом Луганского национального университета имени Тараса Шевченко в качестве учебно-методического пособия для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника»
(протокол № 11 от 12 июня 2018 года)

Введение

Микропроцессоры и микроконтроллеры являются одной из наиболее динамично развивающихся областей современной электронной техники. Впервые появившись в 1971 году, эти электронные приборы быстро прошли путь от простейших устройств, выполняющих несложные вычисления, до современных высокопроизводительных процессоров, реализующих сотни миллионов операций в секунду. Еще более впечатляющим является лавинообразное расширение их областей применения. В настоящее время на производстве и в быту нас окружают десятки микропроцессоров и микроконтроллеров, встроенных в аппаратуру различного назначения. Это не только персональные компьютеры, которые стали обязательным атрибутом, как домашнего интерьера, так и рабочего места специалистов во многих сферах производства и обслуживания. Это и современная бытовая техника, средства проведения досуга, автомобильная электроника и медицинская аппаратура, средства связи, разнообразные приборы и системы, используемые в производственном оборудовании, и многое другое.

Знание основ микропроцессорной техники является необходимым для многих специалистов, как разрабатывающих новые виды устройств на базе современных микропроцессоров и микроконтроллеров, так и использующих в своей профессиональной деятельности аппаратуру, реализованную на их основе.

Микроконтроллеры являются наиболее массовым представителем микропроцессорной техники. Интегрируя на одном кристалле высоко производительный процессор, память и набор периферийных устройств, микроконтроллеры позволяют с минимальными затратами реализовать широкую номенклатуру систем управления различными объектами и процессами. Использование микроконтроллеров в системах управления и обработки информации обеспечивает исключительно высокие показатели эффективности при достаточно низкой стоимости. Микроконтроллерам практически нет альтернативы, когда нужно создать качественные и дешевые системы. Иногда система может состоять только из одного микроконтроллера. Исключение составляет применение программируемых логических интегральных схем (ПЛИС) в области обра-

ботки сигналов в том случае, когда требуется параллельная обработка большого потока входных данных.

До появления Arduino программирование микроконтроллеров сопровождалось сложным и рутинным обучением. С Arduino даже те, кто не имел опыта работы с электронными устройствами, могут проникнуть в ранее загадочный для них мир электроники. Теперь новичкам не нужно тратить много времени на изучение соответствующего материала, они могут быстро разработать прототип, который будет полноценно рабочим.

Учебно-методические рекомендации по выполнению лабораторных работ разработаны для студентов 2 курса очной формы обучения направления подготовки 09.03.01 «Информатика и вычислительная техника» и предназначены для выполнения лабораторных работ в рамках дисциплины «Архитектура ЭВМ и микроконтроллеров».

Данные методические рекомендации содержат цикл лабораторных работ, позволяющих освоить программирование микроконтроллеров в среде Arduino IDE. В процессе выполнения лабораторных работ студенты познакомятся с возможностями платформы Arduino, изучат среду разработки – Arduino IDE, структуру программы – скетч, базовые элементы Processing C/C++, порядок запуска и компиляции готовых скетчей, особенности беспаечной макетной платы, правила её использования, правила подготовки её к работе и электронных компонентов, используемых при работе с платой.

В соответствии с учебным планом дисциплина «Архитектура ЭВМ и микроконтроллеров» изучается в четвертом семестре, всего запланировано 12 лабораторных работ. По окончании изучения дисциплины студенты сдают зачет.

По каждой лабораторной работе оформляется отчет, который предоставляется преподавателю для защиты не позднее следующего лабораторного занятия. К защите лабораторной работы студент должен подготовить ответы на вопросы для самоконтроля.

Лабораторная работа № 1–2.
Подключение светодиода, кнопки и потенциометра к плате Arduino

Цель работы: приобрести практические навыки по подключению кнопки и светодиода к плате Arduino.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

Основные сведения

Arduino – это программно-аппаратная платформа для сверхбыстрого создания электронных устройств, поддерживаемая разработчиками по всему миру и пользующаяся огромной популярностью в России и за рубежом. Платформа позволяет общаться и взаимодействовать с окружающим миром с помощью всевозможных датчиков, сенсоров, моторов и других узлов. Платформа состоит из двух частей: программной и аппаратной.

В качестве программной части выступает кроссплатформенная среда разработки Arduino IDE, которая может запускаться на операционных системах Windows, Linux, Mac OS. С помощью дан-

ной среды можно писать код и программировать платы. В комплекте с программой поставляются многочисленные примеры, библиотеки и удобные утилиты.

В качестве аппаратной части выступают различные платы. На сайте производителя их насчитываются уже более двух десятков разновидностей. Так как платформа распространяется весьма свободно, то в продаже можно найти множество клонов и различных плат расширения.

Рассмотрим основные преимущества данной платформы.

Низкий порог входа для новичков. Например, для того чтобы запрограммировать микроконтроллер фирмы Atmel, требуется заказать микроконтроллер, специализированный программатор, прочитать Data Sheet (техническое описание) объемом 350 страниц, а затем Errata Sheet (список ошибок для данного микроконтроллера). В итоге, чтобы написать простейшую программу, потребуется много времени – не менее недели. На Arduino для аналогичной программы потребуется несколько минут.

Кроссплатформенная среда разработки. В отличие от многих сред программирования, Arduino не ограничивает свободу выбора операционной системы.

Отсутствие необходимости в программаторе. Почти все платы имеют USB разъем. Для плат, в которых не предусмотрены USB, продаются дешевые переходники для подключения USB. Это удобно в том случае, если необходимо навсегда оставить плату Arduino в разработанном устройстве. Без разъема USB она стоит дешевле, а переходником можно воспользоваться один раз и оставить его для других плат.

Наличие большого числа плат. Существует несколько десятков видов оригинальных плат для разных задач, а также их многочисленные клоны.

Переносимость кода. Написав один раз код для платы Arduino UNO, вы можете перенести его на более мощную плату Arduino MEGA или более слабую Arduino NANO. Никаких исправлений в коде делать не придется.

Отсутствие необходимости в пайке. Схемы собираются на беспаечной макетной плате, с помощью специальных проводков.

Открытый исходный код и открытые чертежи (Open Source + Open Hardware). Сообщество разработчиков делится своими достижениями: кодом и чертежами. Если появится желание глубже разобраться в механизмах работы Arduino, то всегда можно заглянуть в схемы и уже написанные программы. Секрета из них никто не делает.

Наличие САПР (систем автоматизированного проектирования), эмуляторов. Они также с открытым исходным кодом, кроссплатформенные. Можно даже на специальных программах проверить, как будет вживую работать плата Arduino с подключенными к ней моторами и датчиками.

Язык программирования C/C++ – один из самых популярных языков программирования. Большинство программистов в мире знают и пользуются этим языком. Зная этот язык, можно с легкостью освоить другие языки. Кроме того, для того чтобы программировать Arduino, совсем не обязательно знать язык в полном объеме – достаточно знать сотую часть всех премудростей. Разумеется, требования к уровню знания языка C/C++ растут по мере усложнения программ.

Наличие большого числа плат-расширений. С ними платы Arduino превращаются в конструктор. Можно добавить сетевую плату Ethernet, плату Bluetooth, GPS, GSM и даже видеоплату VGA. Компания «Мастеркит» разработала очень удобную плату расширения для использования вместе с Arduino. На ней находятся большинство узлов, которые чаще всего используются в Arduino-проектах. Также на плате находится огромное число разъемов для подключения различных датчиков, плат, двигателей, светодиодов, реле.

Платформа постоянно развивается, происходит обновления среды разработки, совершенствование старых плат и появление новых. Вместе с каждой библиотекой поставляется пример ее использования. Например, для написания протокола обмена данными с GSM модулем или со сканером отпечатков пальцев необходимо проверить работоспособность готовой библиотеки или устройства и продолжить работу.

Рассмотрим основные отличия между микропроцессорами и микроконтроллерами. Основная задача микропроцессора – это об-

работка данных. Обрабатываются эти данные по специальной программе, написанной программистами. Микроконтроллер – это тот же микропроцессор, с одним небольшим отличием: он предназначен для управления и контроля какого-то процесса. Например, в компьютере или ноутбуке находится микропроцессор. Он обрабатывает огромные объемы данных. А вот в микроволновой печи, телевизоре или стиральной машинке находятся уже микроконтроллеры. Они управляют работой этих изделий, используя программу, которую написали программисты.

Аппаратная платформа Arduino

Существует множество разных Arduino-совместимых плат. Совместимость означает переносимость кода. Написав код один раз для платы Arduino, можно потом использовать его на других платах, учитывая ограничения конкретной платы. Платы различаются количеством встроенной памяти, частотой процессора (редко), количеством цифровых портов ввода/вывода, аналоговых портов, размерами.

Самые распространенные платы Arduino- Arduino UNO, Arduino Mega, Arduino Nano.

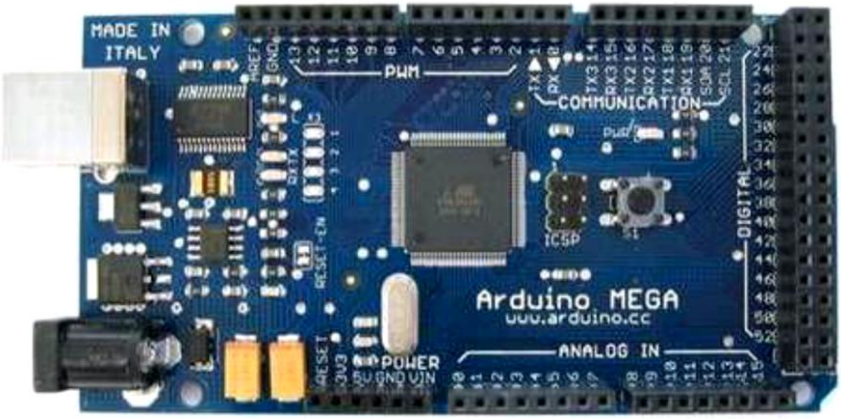

Рис. 1.1 Arduino MEGA

ArduinoMega предназначена для проектов с большими программами и с применением множества датчиков и линий управления. Например, она используется в самодельных 3D принтерах.

Объем программы может превышать в 8 раз максимальный объем программы других плат Arduino.

Рис. 1.2. Arduino UNO

Arduino UNO самая функциональная плата со стандартным набором узлов. Она очень удобна при подключении к ней уже готовых узлов и плат расширения.

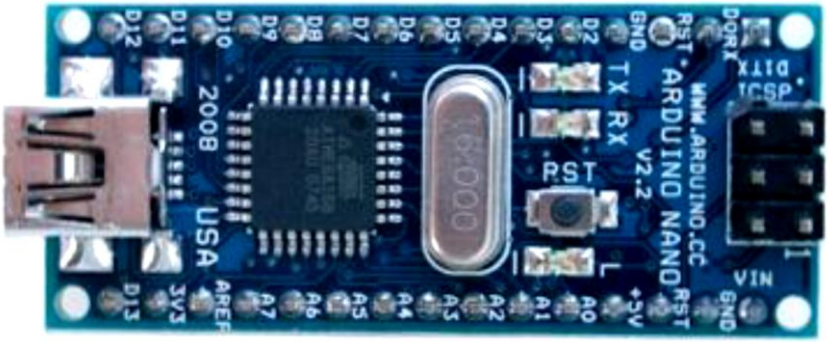

Рис. 1.3. Arduino Nano

Arduino Nano – плата, аналогичная Arduino UNO, но с гораздо меньшим размером и удобными разъемами для макетирования на беспаечных макетных платах.

Arduino Nano – это полнофункциональное миниатюрное устройство, адаптированное для использования с макетными платами.

Все «размеры» и разновидности Arduino-плат абсолютно совместимы друг с другом – если вас заинтересовал проект на Arduino Nano – ничто вам не помешает реализовать его на обычной Arduino UNO или Arduino Mega, причём ни в коде, ни в схеме переделывать ничего не придётся. Можно и наоборот, например, с «меги» на «нано» – лишь бы выводов/памяти хватило (часто в проектах применяются откровенно избыточные платы).

Микроконтроллер – это «мозг» Arduino, он выполняет программу, хранит промежуточные результаты в своей памяти, совершает все логические и арифметические операции, дает команды портам ввода/вывода.

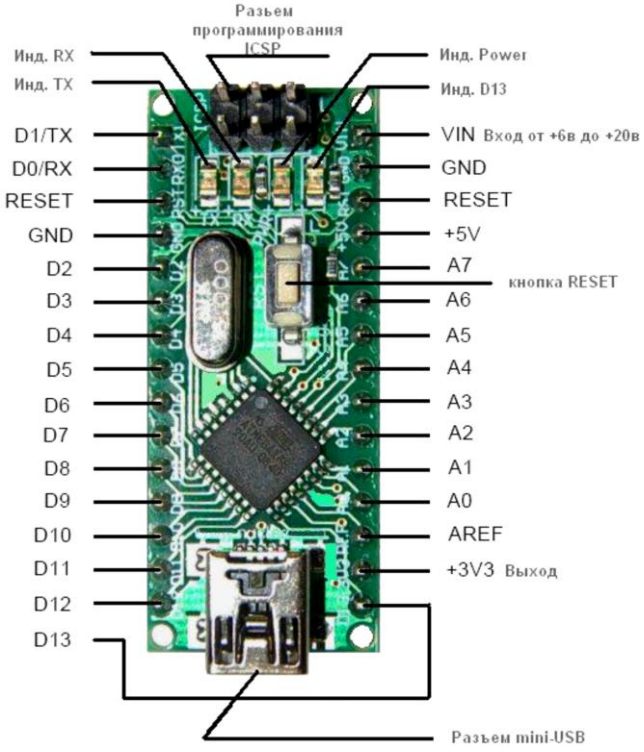

Рис. 1.4. Порты Arduino Nano

Внутри этого мозга есть хранилище, в котором находится программа. Будучи один раз туда записанной, она может храниться там десяток лет. Объем этой программы не может превышать 30 килобайт. Это примерно соответствует 30 тысячам элементарных команд микроконтроллера. Начинающему довольно тяжело написать программу, которая займет всю доступную память. Поэтому места для программы у нас очень много. Программа может перезаписываться не менее 1000 раз.

Порты ввода/вывода – это «руки», «ноги», органы зрения, осязания, обоняния. Через них Arduino общается с окружающим миром: с предметами, людьми, другими платами, выходит в сеть и т.д. С помощью USB порта плата общается с компьютером и через этот же разъем программа попадает из компьютера в микроконтроллер.

На плате расположены еще несколько дополнительных выводов:

– ***Reset –*** (сброс программы). Замкнув эту ножку с ножкой минуса (земли) GND – можно сбросить программу микроконтроллера аналогично кнопке Reset. Нами это особенность не будет использована.

– ***3V3 –*** напряжение на данном выводе +3.3 вольта, генерируемое встроенным регулятором на плате. Максимальное потребление тока 50 миллиампер. От этого вывода будут питаться некоторые модули.

– ***5V –*** напряжение на данном выводе +5 вольт, генерируемое встроенным регулятором на плате.

– ***GND*** – земля, общий, минус- можно называть этот вывод как угодно. Относительно этого вывода подается питание и подключаются остальные выводы. Т.е. этот вывод является вторым выводом для любого другого.

– ***Vin –*** используется для подачи питания от внешнего источника в отсутствии 5 В от разъема USB, например, когда мы хотим запустить нашу плату отдельно от компьютера.

Внешнее питание (не USB) может подаваться через разъем Vin или на разъем XP14 платы расширения. Платформа может работать при внешнем напряжении от 6 до 20 В, но рекомендуется

использовать напряжение в диапазоне 7–12 В для предотвращения перегрева или нестабильной работы.

– ***AREF*** – напряжение, на котором работает один из узлов микроконтроллера – аналогово-цифровой преобразователь, который преобразует напряжение в вольтах в числа. Далее эти числа мы можем использовать в нашей программе. Нам этот вывод не понадобиться, так как мы не собираемся менять это напряжение на другое значение.

– ***Разъем программирования ICSP –*** предназначен для подключения специального программатора, чтобы перепрограммировать Arduino с помощью этого программатора. Либо может быть использован для того чтобы превратить ненадолго саму Arduino в этот самый специализированный программатор. Мы эту возможность не будем использовать.

Типы данных в Arduino IDE

Компьютеры и Arduino в том числе, работают с различными типами данных. В их основе лежит арифметически-логическое устройство (АЛУ), которое выполняет арифметические и логические операции с ячейками памяти: R1+R2, R3*R7, R4&R5 и т.д. Для АЛУ нет разницы, какой тип данных отображать пользователю: текст, целые числа, числа с плавающей запятой или даже часть программного кода.

Команды для этих операций поступают от компилятора, а команды компилятору передаются от пользователя. Именно вы, программист, объясняете компилятору, что это значение – целое (integer), а то значение – число с плавающей запятой (floating point number). После этого компилятор начинает разбираться, что имеется в виду, когда вы сообщаете ему «добавь это целое к числу с плавающей запятой».

Характеристика основных типов данных в Arduino IDE

Оболочка для программирования Arduino по сути представляет из себя язык C++ с поддержкой большого количества библиотек для облегчения процесса написания программ. C++ предлагает широкий выбор различных типов данных.

Ниже представлен список основных типов данных, которые используются в скетчах Arduino. Рядом с каждым типом данных указан его размер.

Обратите внимание, что переменные типа ***signed*** дают возможность оперировать положительными и отрицательными числами, а переменные типа ***unsigned*** допускают только работу с позитивными значениями.

– ***boolean*** (8 бит) – простое логическое true/false (правда/ложь)

– ***byte (***8 бит) – unsigned число в диапазоне от 0-255

– ***char*** (8 бит) – signed число в диапазоне от -128 до 127. В некоторых случаях компилятор будет интерпретировать этот тип данных как символ, что может привести к неожиданным результатам.

– ***unsigned char*** (8 бит) – то же что и «byte»; для ясности кода рекомендуется вместо этого типа данных использовать «byte».

– ***word*** (16 бит) – unsigned число в диапазоне от 0 до 65535

– ***unsigned int*** (16 бит) – то же, что и «word». Рекомендуется заменять типом данных «word» для сокращения кода и ясности

– ***int*** (16 бит) – signed число в диапазоне от -32768 до 32767. Один из самых распространенных типов данных, который очень часто используется для объявления переменных в скетчах-примерах для Arduino, встроенных в Arduino IDE

– ***unsigned long*** (32 бита) – unsigned число в диапазоне от 0 до 4 294 967 295. Чаще всего этот тип данных используется для хранения результатов функции millis( ), которая возвращает количество миллисекунд, на протяжении которого работал ваш код.

– ***long*** (32 бита) – signed число в диапазоне от -2 147 483 648 до 2 147 483 647

– ***float*** (32 бита) – signed число в диапазоне от -3.4028235E38 до 3.4028235E38. Числа с плавающей запятой не характерны для Arduino и компилятору придется прилично попотеть, чтобы их обработать. Так что рекомендуется по возможности их избегать. Более детально мы затронем этот вопрос позже.

Пока не рассматриваются ***arrays (массивы), pointers (указатели) и strings (строки);*** это специальные типы данных, с более сложной концепцией, которую надо рассматривать отдельно.

Переменные в Arduino

Переменные – это способ именовать и хранить числовые значения для последующего использования программой. Переменные – это значения, которые могут последовательно меняться, в отличие от констант, чье значение никогда не меняется. Говоря очень простым языком, переменная – это место, где хранятся какие либо данные.

Их суть заключается в том, что можно однажды определить (задекларировать её), и дальше в программе многократно её использовать, проводя с ней различные манипуляции.

Переменные нужно декларировать (объявлять). Следующий код объявляет переменную inputVariabie, присваивает ей начальное значение, равное нулю, а затем присваивает ей значение, полученное от 2-го аналогового порта:

```
int inputVariabie = 0;
inputVariabie = analogRead(2);
```

Переменные могут быть названы любыми именами, которые не являются ключевыми словами языка программирования Arduino. Все переменные должны быть задекларированы до того, как они могут использоваться. Объявление переменной означает определение типа ее значения int, long, float и т.д. Задание уникального имени переменной, и дополнительно ей можно присвоить начальное значение. Все это следует делать только один раз в программе, но значение может меняться в любое время при использовании арифметических или других разных операций.

Следующий пример показывает, что объявленная переменная inputVariabie имеет тип int, и ее начальное значение равно нулю. Это называется простым присваиванием:

```
int inputVariabie = 0;
```

Переменная может быть объявлена в разных местах программы, и то, где это сделано, определяет, какие части программы могут использовать переменную. Границы переменных в некоторых языках программирования могут называться областями видимости. Переменные могут быть объявлены в начале программы перед **void setup()**, локально внутри функций и иногда в блоке выражений, таком как цикл for. То, где объявлена переменная, опре-

деляет ее границы, т. е. возможность некоторых частей программы ее использовать.

Глобальные переменные таковы, что их могут видеть и использовать любые функции и выражения программы. Такие переменные декларируются в начале программы перед функцией setup(). *Локальные переменные* определяются внутри функций или таких частей, как цикл for. Они видимы и могут использоваться только внутри функции, в которой объявлены. Таким образом, могут существовать несколько переменных с одинаковыми именами в разных частях одной программы, которые содержат разные значения. Уверенность, что только одна функция имеет доступ к ее переменной, упрощает программу и уменьшает потенциальную опасность возникновения ошибок.

Среда разработки Arduino

Среда разработки Arduino представляет собой текстовый редактор программного кода, область сообщений, окно вывода текста (консоль), панель инструментов и несколько меню.

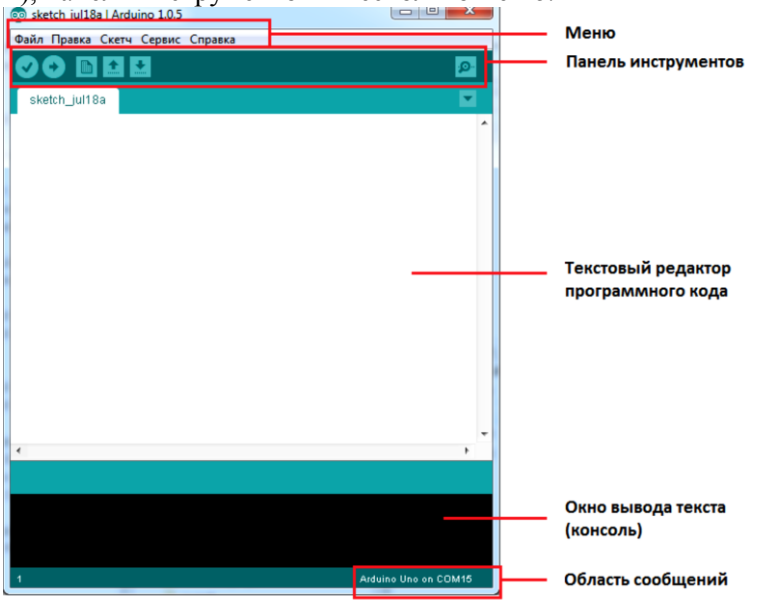

Рис. 1.5. Среда разработки Arduino

Для загрузки программ и связи среда разработки подключается к аппаратной части Arduino.

«Мигание светодиодом»

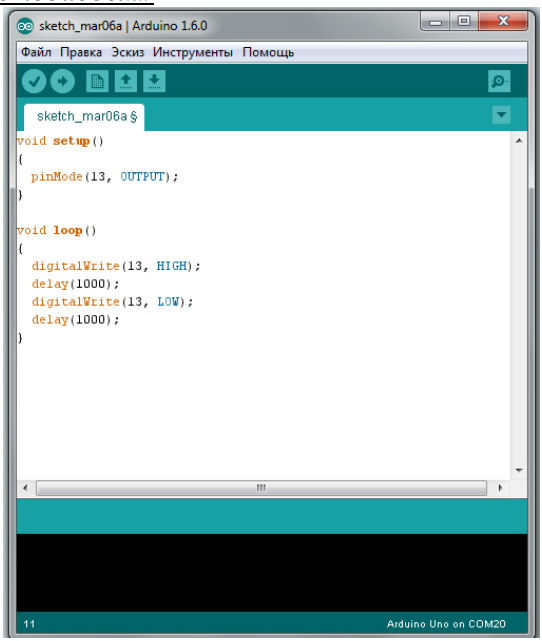

Рис. 1.6. Пример программы «Мигание светодиодом»

Здесь сначала объявили функцию ***setup()***, а за ней функцию ***loop()***. Для программирования микроконтроллера необходимы знание его архитектуры и умение обращаться с регистрами разной периферии, а в arduino это не нужно. Для работы той или иной периферии нужно лишь вызвать ту или иную функцию и передать по надобности какие-либо параметры, а уже в зависимости от периферии функция может вернуть какое-то значение. То есть если бы был графический интерфейс, то все программирование свелось бы к выбору разных галочек и подпунктов меню. Нужно просто запомнить, что любая программа для arduino должна начинаться именно с объявления этих двух функций.

Первая с названием ***setup()*** нужна для начальной конфигурации программы. Данная функция вызывается один раз при первом

старте программы и на протяжении всего периода работы основной программы больше не вызывается. Зачем так сделано? Дело в том что чаще всего при первом старте программы, необходимо проинициализировать периферию. Если это делать в основном цикле, то процессор будет выполнять не нужную работу. Поэтому нужно пока запомнить, что код написанный в теле функции ***setup()*** выполнится один раз. А вот основная программа, которая будет выполнятся на протяжении всей работы микроконтроллера, записывается в тело функции ***loop()***.

Т.е. для начальной настройки и выполнении разово кода программы заполняем тело функции ***setup()***. Для работы основной программы заполняем тело функции ***loop()***.

Для того чтобы помигать светодиодом, нам необходимо посмотреть где он подключен. На плате arduino UNO в примере он подключен к **13** ножке выводов ***DIGITAL***. Сначала в функции **setup()** мы настроим ножку, а затем в функции ***loop()*** будем включать и выключать светодиод. Цифровая ножка микроконтроллера может выполнять всего два действия, либо выводить значение **1** или **0**, либо читать значение переданное на ножку тоже **1** или **0**. В данный момент, так как нам нужно зажигать светодиод, то вывод должен быть настроен как выход.

Задача функции ***pinMode()*** является настройка пина на вход или на выход. Для того чтобы данная функция поняла что ей делать, она принимает два аргумента. Первый аргумент отвечает за номер пина. В нашем случае это ***13***, а второй аргумент (все аргументы передаваемые функциям пишутся через запятую) это направление. В нашем случае это выход ***OUTPUT***. в конце функции как можно заметить стоит точка с запятой. Данная точка с запятой говорит о том что текущая команда закончилась. Следует запомнить как таблицу умножения, что каждая команда *обязательно* должна заканчиваться точкой с запятой. Вот таким образом мы настроили вывод.

Теперь переходим к телу функции ***loop()***. Первая строчка ***digitalWrite(13, HIGH);*** – это функция, которая выводит в вывод микроконтроллера некое значение. Для того чтобы она понимала какое значение и куда ей выводить мы снова передаем два аргумента. Первый это номер вывода, а второй ***HIGH***, то есть ***1***. Как

только эта функция отработает, на выводе **13** появится логическая единица (логическая единица всегда равна напряжению питания МК. Чаще всего это 5 вольт или 3,3 вольта). Следующая строка ***delay(1000);*** **–** это пауза. Данной функции в качестве аргумента передается значение времени в миллисекундах. В нашем случае это ***1000*** мс или ***1*** секунда. Теперь рассмотрим, что произойдёт. Сначала функция ***digitalWrite(13, HIGH);*** выводит на ножку логическую единицу. Затем функция **delay(1000);** останавливает работу программы на ***1*** секунду, а затем выполняется функция ***digitalWrite(13, LOW);***. Как не трудно догадаться она выводит на ножку логический ноль. Затем снова останавливается на **1** секунду. Так как функция ***loop()*** является циклической, то после паузы программа возвращается к началу тела функции ***loop()*** и все начинается с начала. Вот и все, наша первая программа написана.

Если программа не сохранена, то перед загрузкой вам будет предложено ее сохранить. После сохранения, программа загрузится в плату Arduino и вы увидите как замигает светодиод (См. рис. 1.7)

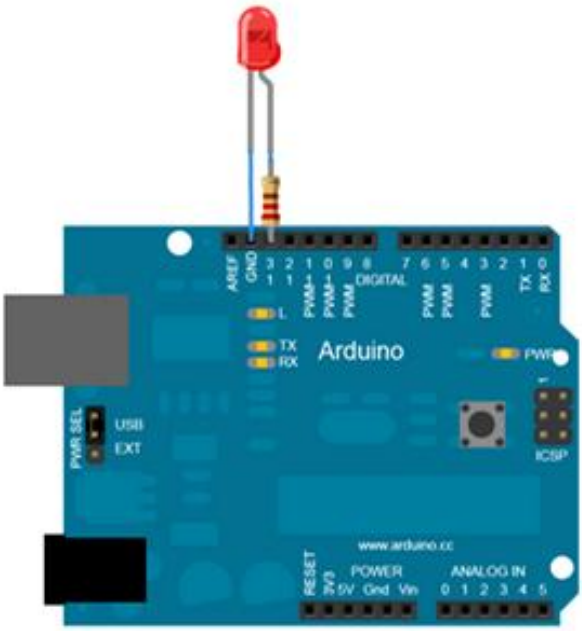

Рис. 1.7. Внешний вид подключения светодиода к плате Arduino

Задание 1. Собрать приведенную ниже схему

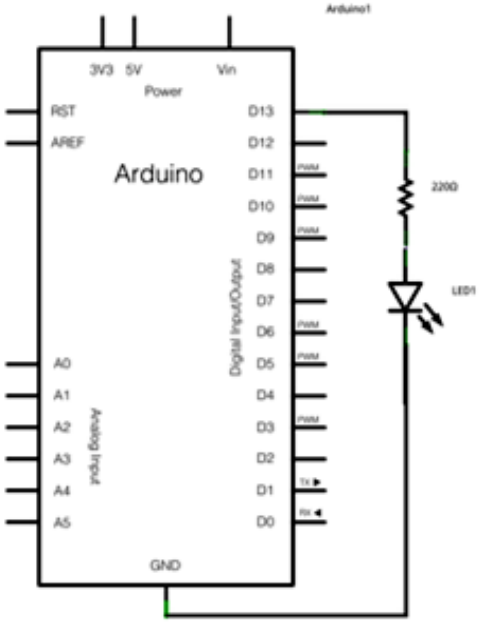

Рис. 1.8. Схема подключения светодиода к плате Arduino

Пример программного кода 1.1:

```
/* Мигание встроенным в плату LED
* Включает и выключает светодиод (LED) подсоединенный
* к выходу 13 платы, с интервалом 1 секунда горит и одна секунда паузы
*
*/
int ledPin = 13;     // LED подсоединен к выводу 13
void setup()
{
   pinMode(ledPin, OUTPUT);     // настраиваем порт 13 на выход
}
void loop()
{
  digitalWrite(ledPin, HIGH);      // включаем LED подаем лог. 1
  delay(1000);     // пауза 1 секунда (1000 мс = 1 секунда) удерживаем лог. 1
  digitalWrite(ledPin, LOW);      // выключаем LED
  delay(1000);     // пауза 1 секунда (1000 мс = 1 секунда) удерживаем лог. 0
}
/* далее программа будет повторяться бесконечно */
```

«Управление светодиодом с помощью кнопки»

Для связи с внешними элементами в контроллере Arduino UNO существуют 14 цифровых выводов. Каждый вывод может быть определен программой как вход или выход. У цифрового выхода есть только два состояния высокое и низкое. Высокое состояние соответствует напряжению на выходе порядка 5В, низкое состояние – 0В. Выход допускает подключение нагрузки с током до 40мА. Когда вывод определен как вход, считав его состояние, можно определить уровень напряжения на входе. При напряжении близком к 5В (реально более 3В) будет считано высокое состояние, соответствующее константе HIGH. При напряжении близком к 0 (менее 1,5В) будет считано низкое состояние, или константа LOW.

Светодиод мы должны подключить к выводу, определив его как выход, а кнопка подключается к выводу с режимом вход. Светодиод подключается через резистор, ограничивающий ток. Вот типичная схема.

Программа должна управлять светодиодом с помощью кнопки: при нажатой кнопке светодиод светится, а при отжатой кнопке светодиод не светится.

Для связи с внешними элементами в контроллере Arduino UNO существуют 14 цифровых выводов. Каждый вывод может быть определен программой как вход или выход. У цифрового выхода есть только два состояния высокое и низкое. Высокое состояние соответствует напряжению на выходе порядка 5В, низкое состояние – 0В. Выход допускает подключение нагрузки с током до 40 мА.

Когда вывод определен как вход, считав его состояние, можно определить уровень напряжения на входе. При напряжении близком к 5В (реально более 3В) будет считано высокое состояние, соответствующее константе HIGH. При напряжении близком к 0 (менее 1,5В) будет считано низкое состояние, или константа LOW. Светодиод подключаем к выводу, определив его как выход, а кнопка подключается к выводу с режимом вход. Светодиод подключается через резистор, ограничивающий ток

Резистор рассчитывается по формуле

$$I=(U_{выхода}–U_{падения\ на\ светодиоде})/R.$$

$U_{выхода}$=5В, $U_{падения\ на\ светодиоде}$ можно принять равным 1,5В. Получается, ч то в нашей схеме ток через светодиод задан на уровне 10мА.

Для простоты соединений будем использовать светодиод, установленный на плате. Тот самый, который мигал в первом тестовом примере. Он подключен к цифровому выводу 13. В этом случае дополнительный светодиод к плате подключать не надо.

Кнопку подключаем к любому другому выводу, например, 2. Аппаратная часть схемы подключения кнопки должна обеспечивать уровни напряжений 0В при нажатой кнопке и 5В при свободной.

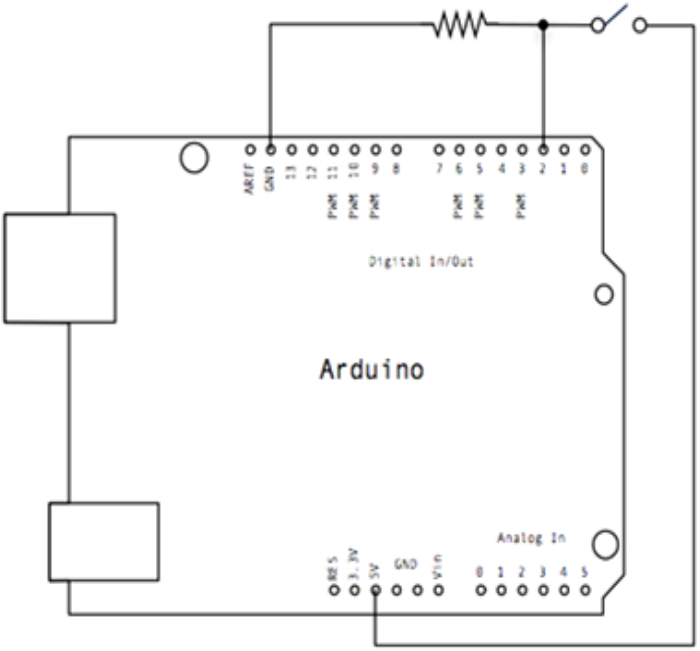

Рис 1.9. Схема подключения кнопки к плате Arduino

При отжатой кнопке резистор формирует на выводе 5В, а при нажатой – вход замыкается на землю. Рекомендации по выбору резистора я напишу в заключительном уроке про кнопки. Сейчас предложу другой вариант. Все выводы платы имеют внутри контроллера резисторы, подключенные к 5В. Их можно программно

включать или отключать от выводов. Сопротивление этих резисторов порядка 20-50 кОм.

В итоге схема подключения будет выглядеть так:

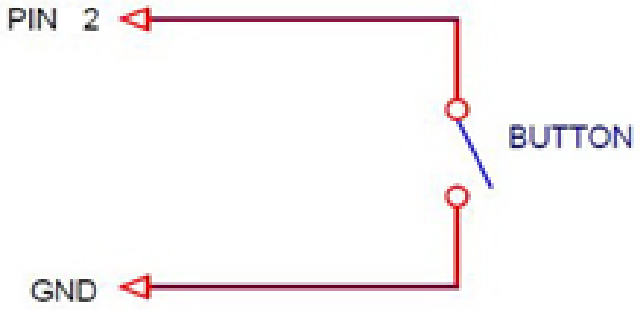

Рис. 1.10. Схема подключения кнопки.

Кнопку можно припаять на проводах к разъему, а можно установить ее на макетную плату без пайки.

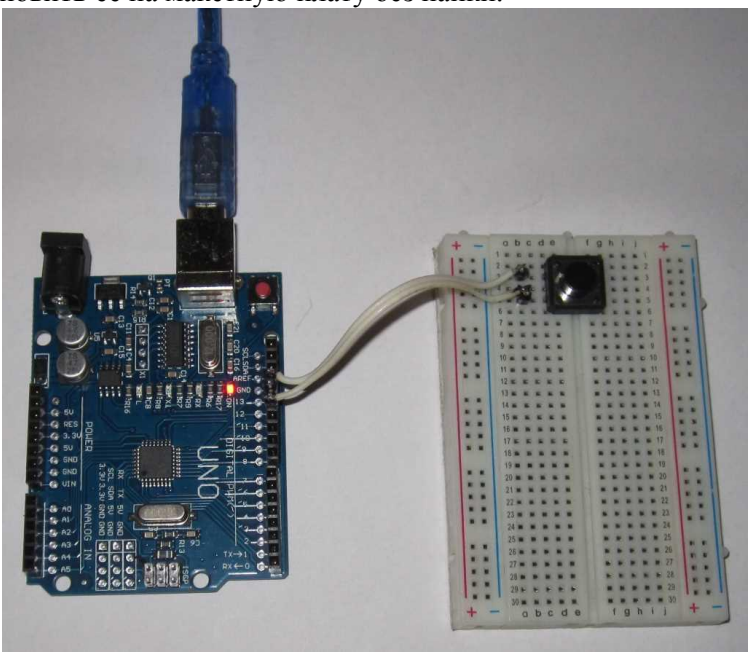

Рис. 1.11. Внешний вид Arduino и беспаечной платы с кнопкой

Функции управления вводом/выводом.

Для работы с цифровыми выводами в системе Arduino есть 3 встроенные функции. Они позволяют установить режим вывода, считать или установить вывод в определенное состояние. Для определения состояния выводов в этих функциях используются константы HIGH и LOW, которые соответствуют высокому и низкому уровню сигнала.

pinMode(pin, mode) – устанавливает режим вывода (вход или выход).

Аргументы: pin – номер вывода; mode – режим вывода.

mode = INPUT	вывод определен как вход, подтягивающий резистор отключен
mode = INPUT_PULLUP	вывод определен как вход, подтягивающий резистор подключен
mode = OUTPUT	вывод определен как выход

Функция не возвращает ничего.

digitalWrite(pin, value) – устанавливает состояние выхода (высокое или низкое).

Аргументы: pin – номер вывода; value – состояние выхода.

value = LOW	устанавливает выход в низкое состояние
value = HIGH	устанавливает выход в высокое состояние

Функция не возвращает ничего.

digitalRead(pin) – считывает состояние входа.

Аргументы: pin – номер вывода.

Возвращает состояние входа:

digitalRead(pin) = LOW	низкий уровень на входе
digitalRead(pin) = HIGH	высокий уровень на входе

Программа управления светодиодом.

Программа в Arduino состоит из двух функций setup() и loop. В setup() мы устанавливаем режимы выводов, а в loop() считываем состояние кнопки в переменную *buttonState* и передаем его на светодиод. По пути инвертируем, т.к. при нажатой кнопке низкое состояние сигнала, а светодиод светится при высоком.

Пример программного кода 1.2:

```
 * Программа зажигает светодиод (вывод 13) при нажатии кнопки
(вывод 12) */
```

```
boolean buttonState;   //- создаем глобальную переменную buttonState
 void setup() {
  pinMode(13, OUTPUT);       //- определяем вывод 13 (светодиод)
 //как выход
  pinMode(2, INPUT_PULLUP); //- определяем вывод 2 (кнопка)
//как вход
}
//  бесконечный цикл
void loop() {
  buttonState = digitalRead(2);  //- считываем состояние 2 входа
 // (кнопки) и записываем в buttonState
  buttonState = ! buttonState;    //- инверсия переменной buttonState
  digitalWrite(13, buttonState);  //- записываем состояние из buttonState
// на выход 13 (светодиод)
}
```

Для хранения промежуточного значения состояния кнопки создаем переменную buttonState с типом boolean. Это логический тип данных. Переменная может принимать одно из двух значений: true (истинно) или false (ложно). В нашем случае – светодиод светится и не светится.

Скопируйте или перепишите код программы в окно Arduino IDE. Загрузите в контроллер и проверьте следующий блок программы

```
buttonState = digitalRead(2);  // считываем состояние 2 входа (кнопки)
// и записываем в buttonState
  buttonState = ! buttonState;    //- инверсия переменной buttonState
  digitalWrite(13, buttonState);  //- записываем состояние из buttonState
//на выход 13 (светодиод)
```

можно записать без использования промежуточной переменной buttonState.

```
digitalWrite(13, ! digitalRead(2) );
```

В качестве аргумента для функции *digitalWrite()* выступает функция *digitalRead()*. Хороший стиль это именно такой вариант. Не требуются дополнительные переменные, меньше текст. Т.е. функцию можно использовать как аргумент другой функции. Функции можно вызывать из функций.

Другой вариант этой же программы, использующий условный оператор *if*:

```
/* Программа зажигает светодиод (вывод 13) при нажатии кнопки
(вывод 2) */
void setup() {
```

```
  pinMode(13, OUTPUT);       //- определяем вывод 13 (светодиод)
//как выход
  pinMode(2, INPUT_PULLUP);  //- определяем вывод 2 (кнопка)
// как вход
}
//  бесконечный цикл
void loop() {
  if ( digitalRead(2) == LOW ) digitalWrite(13, HIGH);
  else digitalWrite(13, LOW);
}
```

В бесконечном цикле проверяется состояние вывода 2 (кнопка), и если оно низкое (LOW), то на выводе 13 (светодиод) формируется высокое состояние (HIGH). В противном случае состояние светодиода низкое (LOW).

Директива ***#define***.

Во всех примерах для функций ввода/вывода мы указывали аргумент pin, определяющий номер вывода, в виде конкретного числа – константы. Мы помнили, что константа 2 это номер вывода кнопки, а 13 – номер вывода светодиода. Гораздо удобнее работать с символьными именами. Для этого в языке С существует директива, связывающая идентификаторы с константами, выражениями.

Директива #define определяет идентификатор и последовательность символов, которая подставляется вместо идентификатора, каждый раз, когда он встречается в тексте программы.

В общем виде она выглядит так:

#define имя последовательность_символов

Если в наших программах мы напишем:

```
#define LED_PIN 13   //- номер вывода светодиода равен 13
```

то каждый раз, когда в программе встретится имя LED_PIN, при трансляции вместо него будет подставлены символы 13. Функция включения светодиода выглядит так:

```
digitalWrite(LED_PIN, HIGH);
```

Окончательный вариант программы с использованием #define:

```
/* Программа зажигает светодиод (вывод 13) при нажатии кнопки
(вывод 2) */
#define LED_PIN 13    //- номер вывода светодиода равен 13
#define BUTTON_PIN 2  //- номер вывода кнопки равен 2
void setup() {
```

```
  pinMode(LED_PIN, OUTPUT);    //- определяем вывод 13 (светодиод)
// как выход
  pinMode(BUTTON_PIN, INPUT_PULLUP);  //- определяем вывод 2
// (кнопка) как вход
}
//  бесконечный цикл
void loop() {
  digitalWrite(LED_PIN, ! digitalRead(BUTTON_PIN) );
}
```

Обратите внимание, что после директивы #define точка с запятой не ставится, потому что это псевдо оператор. Он не совершает никаких действий. Директива задает константы, поэтому принято имена для нее писать в верхнем регистре с разделителем – нижнее подчеркивание.

Аналоговые входы платы Arduino.

Плата Arduino UNO содержит 6 аналоговых входов предназначенных для измерения напряжения сигналов. Правильнее сказать, что 6 выводов платы могут работать в режиме, как дискретных выводов, так и аналоговых входов.

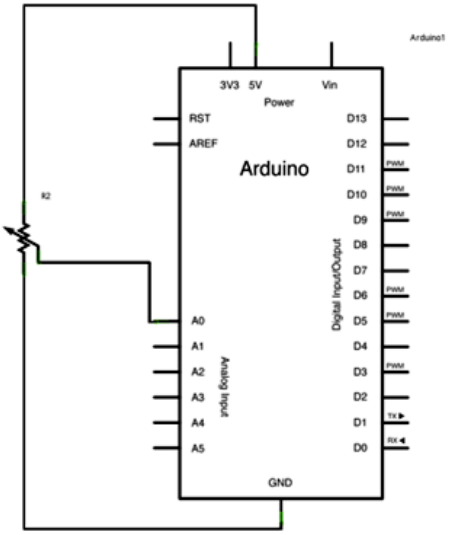

Рис. 1.12. Схема подключения переменного резистора к аналоговому входу платы Arduino

Эти выводы имеют номера от 14 до 19. Изначально они настроены как аналоговые входы, и обращение к ним можно производить через имена A0-A5. В любой момент их можно настроить на режим дискретных выходов.

```
pinMode(A3, OUTPUT);  // установка режима дискретного вывода для A3
digitalWrite(A3, LOW);   // установка низкого состояния на выходе A3
```

Чтобы вернуть в режим аналогового входа:

```
pinMode(A3, INPUT);   // установка режима аналогового входа для A3
```

Аналоговые входы и подтягивающие резисторы.

К выводам аналоговых входов, так же как и к дискретным выводам, подключены подтягивающие резисторы. Включение этих резисторов производится командой

```
digitalWrite(A3, HIGH); // включить подтягивающий резистор к входу A3
```

Команду необходимо применять к выводу настроенному в режиме входа.

Резистор может оказать влияние на уровень входного аналогового сигнала. Ток от источника питания 5В, через подтягивающий резистор, вызовет падение напряжения на внутреннем сопротивлении источника сигнала. Так что лучше резистор отключать.

Аналого-цифровой преобразователь платы Arduino.

Собственно измерение напряжение на входах производится аналого-цифровым преобразователем (АЦП) с коммутатором на 6 каналов. АЦП имеет разрешение 10 бит, что соответствует коду на выходе преобразователя 0 1023. Погрешность измерения не более 2 единиц младшего разряда.

Для сохранения максимальной точности (10 разрядов) необходимо, чтобы внутреннее сопротивление источника сигнала не превышало 10 кОм. Это требование особенно важно при использовании резисторных делителей, подключенных к аналоговым входам платы. Сопротивление резисторов делителей не может быть слишком большим.

Программные функции аналогового ввода

Функция **analogRead()**

```
int analogRead(port)
```

считывает значение напряжения на указанном аналоговом входе. Входное напряжение диапазона от 0 до уровня источника

опорного напряжения (часто 5В) преобразовывает в код от 0 до 1023.

При опорном напряжении равном 5В разрешающая способность составляет 5В / 1024 = 4,88 мВ. Занимает на преобразование время примерно 100 мкс.

```
int inputCod;   // код входного напряжения
float inputVoltage;    // входное напряжение в В
inputCod=analogRead(A0);    // чтение напряжения на входе А0
inputVoltage=((float)inputCod * 5./1024. );// пересчет кода в напряжение (В)
```

Если аналоговый вход не подключен, то значения возвращаемые функцией analogRead() могут принимать случайные значения.

Пример программного кода 1.3:

```
int analogPin = 0;     // номер порта к которому подключен потенциометр
int val = 0;           // переменная для хранения считываемого значения
 void setup()
{
  Serial.begin(9600)     //  установка связи по serial
}
Void loop()
{
  val = analogRead(analogPin);     // считываем значение
  Serial.println(val);             // выводим полученное значение
}
```

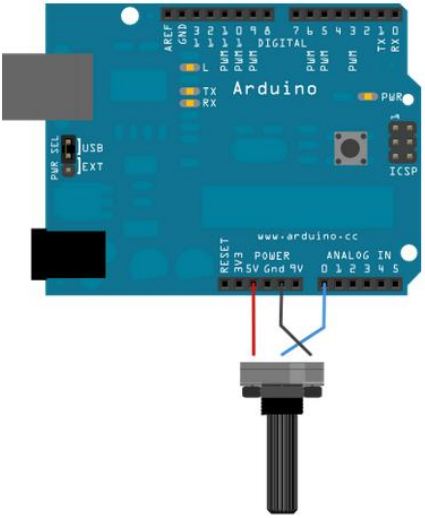

Рис 1.13. Пример подключения переменного резистора к аналоговому входу платы Arduino

Функция **analogWrite()**

analogWrite(pin, value)

выдает аналоговую величину (ШИМ-волну) на порт вход/выхода. Функция может быть полезна для управления яркостью подключенного светодиода или скоростью электродвигателя. После вызова analogWrite() на выходе будет генерироваться постоянная прямоугольная волна с заданной шириной импульса до следующего вызова analogWrite (или вызова digitalWrite или digitalRead на том же порту вход/выхода). Частота ШИМ сигнала приблизительно 490 Hz.

На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11, на плате Arduino Mega порты с 2 по 13.

Период ШИМ сигнала на портах вход/выхода 5 и 6 будет несколько длиннее. Это связано с тем, что таймер для данных выходов также задействован функциями millis() и delay(). Данный эффект более заметен при установке коротких периодов ШИМ сигнала (0-10).

Пример программного кода 1.4. Задание яркости светодиода пропорционально значению, снимаемому с фоторезистора:

```
int ledPin = 9;    // Светодиод подключен к выходу 9
int analogPin = 3; // фоторезистор подключен к выходу 3
int val = 0;       // переменная для хранения значения
void setup()
{
   pinMode(ledPin, OUTPUT);      // установка порта на выход
}
void loop()
{
   val = analogRead(analogPin);  // считываем значение с порта,
//подключенному к потенциометру
   analogWrite(ledPin, val / 4); // analogRead возвращает значения от 0
// до 1023, analogWrite должно быть в диапазоне от 0 до 255


}
```

Задания и порядок выполнения работы

1. Сгенерировать сигнал SOS с помощью светодиода, пользуясь азбукой Морзе:

Рис. 1.14. Пример ускоренного радиообмена с использованием азбуки Морзе

2. Сгенерировать сигнал вашего имени с помощью светодиода, пользуясь алфавитом азбуки Морзе:

А • —	Л • — • •	Ц — • — •
Б — • • •	М — —	Ч — — — •
В • — —	Н — •	Ш — — — —
Г — — •	О — — —	Щ — — • —
Д — • •	П • — — •	Ъ • — — • — •
Е •	Р • — •	Ы — • — —
Ж • • • —	С • • •	Ь — • • —
З — — • •	Т —	Э • • — • •
И • •	У • • —	Ю • • — —
Й • — — —	Ф • • — •	Я • — • —
К — • —	Х • • • •	

Рис. 1.15. Алфавит азбуки Морзе

3. С помощью трёх светодиодов сымитировать работу светофора
4. Выполнить включение светодиода с задержкой после нажатия кнопки

5. Включать светодиод при нажатии на кнопку, при последующем нажатии включить следующий светодиод, а предыдущий потушить (в схеме использовать не менее пяти светодиодов).
6. Собрать схему и написать программу для индикатора уровня сигнала, аналогового входа, индикатор состоит из пяти светодиодов
7. Собрать схему и написать программу для управления потенциометром загорания одного из пяти светодиодов в зависимости от величины входного сигнала

Контрольные вопросы

1. Для чего предназначена программно-аппаратная платформа Arduino?
2. Что входит в состав программной части платформы Arduino?
3. Что входит в состав аппаратной части платформы Arduino?
4. Перечислите основные преимущества платформы Arduino.
5. Опишите интерфейс среды разработки Arduino.
6. Опишите панель Меню.
7. Аналоговые порты ввода и вывода.
8. Какие циклы применяются при программировании для Arduino?
9. Поясните синтаксис цикла *for* и *while*.
10. Поясните порядок запуска готовых скетчей.
11. Что такое компиляция программы?
12. Для чего применяется резистивный ограничитель тока? Формула расчёта?
13. Какие электронные компоненты используются при работе с платой расширения?
14. От чего зависит процесс прошивки программы в Arduino?
15. Для чего нужна директива *#define*?

Лабораторная работа № 3–4.
Управление трехцветным RGB-светодиодом по сигналу с компьютера

Цель работы: приобрести практические навыки по управлению различными устройствами по сигналу с компьютера.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

Управление RGB светодиодом на Arduino

Для отображения всей палитры оттенков вполне достаточно три цвета, используя RGB синтез (Red – красный, Green – зеленый, Blue – синий). RGB палитра используется не только в графических редакторах, но и в сайтостроении. Смешивая красный, зеленый и синий цвет в разной пропорции можно получить практически любой цвет.

RGB светодиоды объединяют три кристалла разных цветов в одном корпусе. Использование RGB светодиодов и RGB LED ленты позволяет создать осветительный прибор или освещение инте-

рьера с любым оттенком цвета. Преимущества RGB светодиодов в простоте конструкции и высоком КПД светоотдачи.

RGB LED имеет 4 вывода – один общий (анод или катод имеет самый длинный вывод) и три цветовых вывода. К каждому цветовому выходу следует подключать резистор. Кроме того, RGB LED может сразу монтироваться на плате и иметь встроенные резисторы.

Аналоговые выходы на Arduino используют «широтно-импульсную модуляцию» для получения различной силы тока. Мы можем подавать на все три цветовых входа на светодиоде различное значение ШИМ-сигнала в диапазоне от 0 до 255, что позволит нам получить на RGB LED Arduino практически любой оттенок света.

Поскольку многоцветный светодиод состоит из трех обычных, мы будем подключать их отдельно. Каждый светодиод соединяется со своим выводом и имеет свой отдельный резистор. Используем RGB-светодиод с общим катодом, так что провод к земле будет только один.

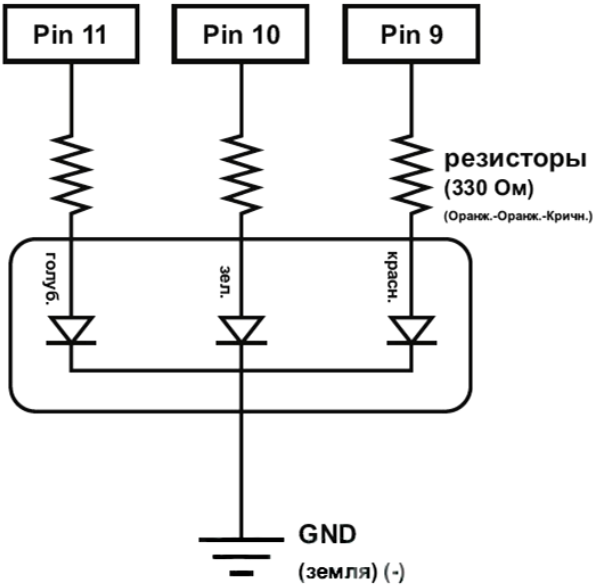

Рис. 3.1. Принципиальная схема подключения RGB модуля к плате Arduino

Подключение оборудования.

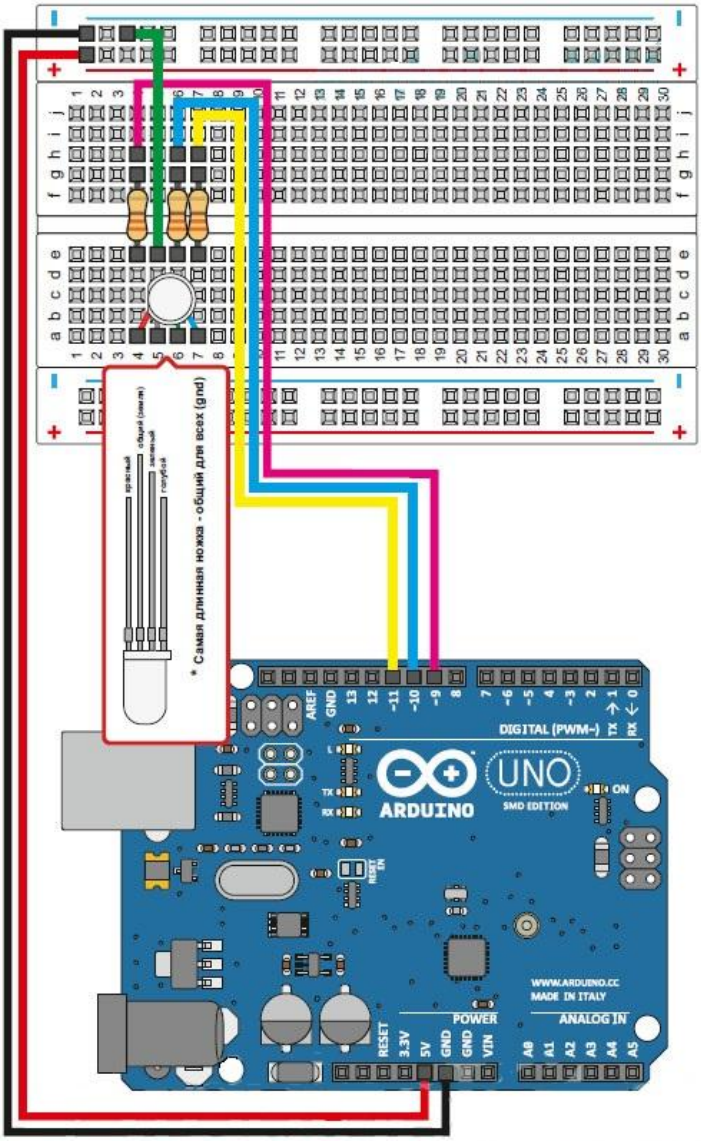

Рис. 3.2. Внешний вид подключения RGB светодиода к плате Arduino

На кромке светодиода есть небольшой скос, это ключ, он указывает на ножку красного светодиода, дальше идет общая, дальше зеленый и синий.

1. Подключите ногу красного светодиода к резистору 330 Ом. Подключите другой конец резистора в порт Arduino pin9.
2. Подключите Общий вывод к земле GND .
3. Подключите ногу зеленого к резистору 330 Ом.
4. Подключите другой конец резистора в порт Arduino pin10.
5. Подключите ногу синего к резистору 330 Ом.
6. Подключите другой конец резистора в порт Arduino pin11.

Рисунок 3.3 показывает внешний вид макетной платы с собранной схемой, и плату Arduino с проводами идущими от макетной платы.

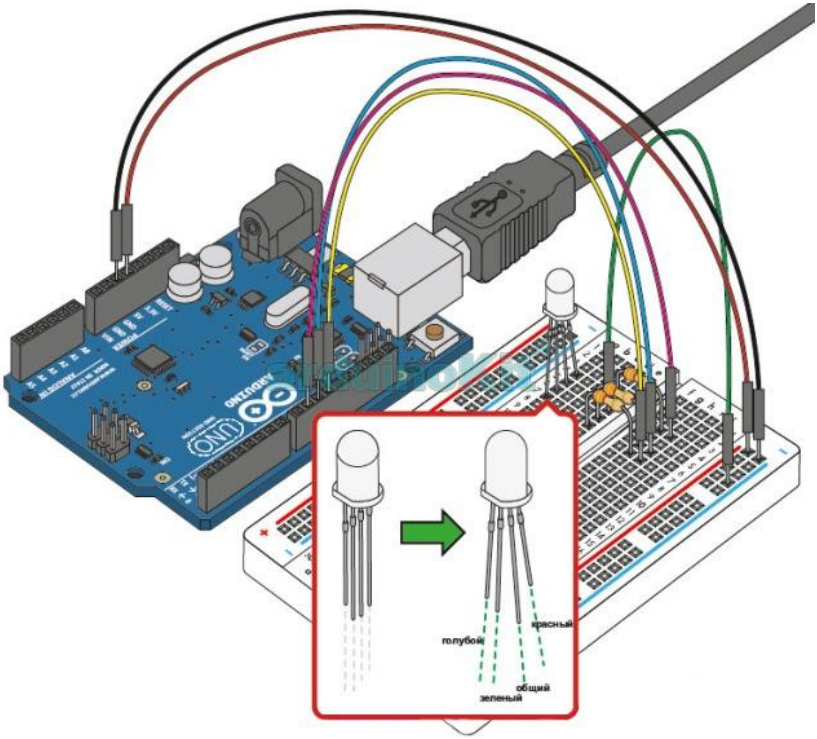

Рис 3.3. Внешний вид макетной платы с собранной схемой, и платы Arduino с проводами, идущими от макетной платы

Остается загрузить программу в Arduino через USB шнур.

Пример программного кода 3.1 Программа демонстрации возможностей RGB LED:

```
// Сначала мы присвоим нужным портам имена, чтобы легче
// читать и понимать нашу программу.
// Поставленное слово «const» перед переменной
// указывает, на то что эта переменная имеет «постоянное» значение,
// которое никогда не будет меняются. ( Вы должны это помнить, иначе
// Arduino выдаст вам дружеское предупреждение, если вы случайно
// попытаетесь изменить это значение.
const int red_pin = 9;
const int green_pin = 10;
const int blue_pin = 11;
// Следующая переменная контролирует, на сколько быстро
// выполняется цикл
// по смене цветов (в качестве эксперимента попробуйте поменять ее
// значение)
int display_time = 500;
void setup() {
// Далее мы конфигурируем порты Arduino которые будут управлять
// светодиодом:
  pinMode(red_pin, OUTPUT);
  pinMode(green_pin, OUTPUT);
  pinMode(blue_pin, OUTPUT);
}
void loop() {
// В этом примере, мы начнем писать свои собственные функции.
// Функции разбивают программу на маленькие кусочки, это упрощает
// понимание программы и поэтому отпадает необходимость каждый
// раз вносить изменения в setup() или loop().
// Мы покажем вам два способа для запуска RGB LED.
// Первый способ состоит из включения и выключения красного,
// зеленого и синего в различных сочетаниях (например красный – синий,
// или зеленый синий, зеленый – синий и т.п.). Такие сочетания, в общей
// сложности, дают вам восемь цветов (если считать «черный» в
// качестве цвета).
// Мы создали функцию с именем mainColors (), которая проходит через
// все восемь цветов. Здесь мы только вызываем нашу функцию,
// как бы говорим – запустить. Сам же код функции находится ниже.
  mainColors();
// Эта функция включает и выключает отдельно каждый светодиод либо
// Вкл., либо Выкл. Если вы хотите получить больше чем восемь цветов,
//  вам придется изменять еще и яркость каждого светодиода. Для
//  этого можно воспользоваться функцию analogWrite(), – градация
// яркости от 0 до 255. Далее идет функция с именем showSpectrum (),
```

```
//  которая плавно проследует через каждый из 8 цветов.
// Опять же здесь мы только называем ее, сам же код находится ниже.
  showSpectrum();
}
// Если захочется использовать функцию mainColors() в своих собственных
// программах, просто скопируйте ее код и добавьте в свой скетч.
void mainColors()
{
// Выкл. – все светодиоды выключены
  digitalWrite(red_pin, LOW);
  digitalWrite(green_pin, LOW);
  digitalWrite(blue_pin, LOW);
  delay(1000);
// Включаем КРАСНЫЙ – RED_PIN, HIGH
  digitalWrite(red_pin, HIGH);
  digitalWrite(green_pin, LOW);
  digitalWrite(blue_pin, LOW);
  delay(1000);
// Включаем ЗЕЛЕНЫЙ – GREEN_PIN, HIGH.
  digitalWrite(red_pin, LOW);
  digitalWrite(green_pin, HIGH);
  digitalWrite(blue_pin, LOW);
  delay(1000);
// Включаем СИНИЙ – BLUE_PIN, HIGH.
  digitalWrite(red_pin, LOW);
  digitalWrite(green_pin, LOW);
  digitalWrite(blue_pin, HIGH);
  delay(1000);
// Включаем ЖЕЛТЫЙ – RED и BLUE HIGH.
  digitalWrite(red_pin, HIGH);
  digitalWrite(green_pin, HIGH);
  digitalWrite(blue_pin, LOW);
  delay(1000);
// БИРЮЗОВЫЙ – GREEN и BLUE
  digitalWrite(red_pin, LOW);
  digitalWrite(green_pin, HIGH);
  digitalWrite(blue_pin, HIGH);
  delay(1000);
//РОЗОВЫЙ – RED и BLUE
  digitalWrite(red_pin, HIGH);
  digitalWrite(green_pin, LOW);
  digitalWrite(blue_pin, HIGH);
  delay(1000);
//БЕЛЫЙ – все цвета HIGH (Вкл.)
  digitalWrite(red_pin, HIGH);
```

```
  digitalWrite(green_pin, HIGH);
  digitalWrite(blue_pin, HIGH);
  delay(1000);
}
// Ниже приведен код еще двух функций:
// ShowSpectrum () и showRGB().
// ShowRGB () выводит один цвет на светодиод. Когда вам необходим
// определённый цвет, вызываете showRGB() с нужным вам цветом.
// ShowSpectrum () выводит все цвета по очереди – радуга.
// На самом деле showSpectrum() вызывает снова и снова
// функцию showRGB (), для того чтобы получилась радуга.

// Мы будем часто разбивать задачи на отдельные функции, как
// в этом примере. Это упрощает написание программ, и можно
// легко перенести функцию в свою новую программу.

// showSpectrum()
// Эта функция перебирает все возможные цвета, путем увеличения
// переменной от 0 до 768 (общее количество цветов), и неоднократно
// вставляет полученное значение в showRGB (), чтобы отобразить от-
дельные цвета.

// В этой функции, мы используем петлю "loop()" в которой выполняется
// цикл из трех последовательных шагов:
// Инициализация переменной «int x», в следующем шаге увеличиваем эту
// переменную на единицу, и в следующем шаге вызываем функцию
//showRGB(),  вставляя в нее новое значение переменной, ждем некоторое
//время, примерно 10мс.,  и выполняем петлю заново. Таким образом наша
// функция showRGB() получает новое значение при каждом новом проходе
// петли.

// Если еще подробнее – Каждый «for() loop» цикл имеет
// три инструкции через точку с запятой:

// 1. Что-то делать, прежде чем начать
// 2. Тест – выполняется ли условие. Петля «loop()» будет выполняться,
пока условие истинно.
// 3. Что-то сделать после каждого цикла – в нашем случае увеличить
переменную.

// Условие для for(), чуть ниже:
// 1. x = 0; Перед началом, убедиться, что x = 0.
// 2. x <768; Далее условие: проверить, x меньше 768, если да,
// то выполнить следующую инструкцию.
// 3. x + + т.е увеличить переменную x на единицу.
```

```
// Вы можете использовать и такой вид записи – «x = x + 1»

// Каждый раз, когда вы идете через петлю, значение переменной будет
// увеличиваться  пока не дойдет до 768, а это значит, что условие стало
// ложным, и значит пора выходить из петли. И переходить к следующей
// инструкции.


void showSpectrum()
{
  int x;
  for (x = 0; x < 768; x++)
  {
    showRGB(x); // Вызов RGBspectrum() сновым значением x
    delay(10); // Пауза 10 ms (1/100 секунды)
  }
}

// ShowRGB()
// Эта функция переводит число от 0 до 767 в определенный цвет.
// Если плавно пройтись по этому числовому ряду, то светодиод будет
// плавно менять цвет через весь цветовой спектр.

// Базовые цифры:
// 0 = чистый красный
// 255 = чистый зеленый
// 511 = чистый синий
// 767 = чистый красный (опять же)

//Числа между вышеуказанных создадут промежуточные цвета.
// Например, 640 находится на полпути между 512 (чистый синий)
// и 767 (Чистый красный). Это даст смесь синего и красного,
// в результате чего получится фиолетовый.

// Если посчитать в верх от 0 до 767 и передать это число функции,
// то светодиод будет плавно переливаться всеми цветами радуги.
// Начиная и заканчивая красным.

void showRGB(int color)
{
  int redIntensity;
  int greenIntensity;
  int blueIntensity;

//Здесь мы будем использовать условие «if/else» (если/тогда), для
```

```
// того чтобы определить в какую из трех зон (R,G,B)
// попадет значение переменной x
//Каждая из этих зон охватывает значение от 0 до 255, потому что
// только такие значения понимает функция analogWrite().
// В каждой из этих зон, мы будем вычислять яркость для
// каждого из светодиодов – красного, зеленого и синего.
  if (color <= 255) // зона 1
  {
    redIntensity = 255 – color; // красный меняется из включенного на
// выключенный
    greenIntensity = color; // зеленый меняется из выключенного на
// включенный
    blueIntensity = 0; // голубой всегда выключен
  }
  else if (color <= 511) // зона 2
  {
    redIntensity = 0; // красный всегда выключен
    greenIntensity = 255 – (color – 256); // зеленый от включенного к
// выключенному
    blueIntensity = (color – 256); // голубой от выключенного к включенному
  }
else // color >= 512 // зона 3
  {
    redIntensity = (color – 512); // красный выкл. К вкл.
    greenIntensity = 0; // зеленый всегда выкл.
    blueIntensity = 255 – (color – 512); // голубой от вкл. К выкл.
  }
// Теперь, когда параметры значения яркости были установлены,
// функциям передаются значение этих переменных
  analogWrite(red_pin, redIntensity);
  analogWrite(blue_pin, blueIntensity);
  analogWrite(green_pin, greenIntensity);
}
```

Последовательная передача данных

Arduino/Freeduino имеет встроенный контроллер для последовательной передачи данных, который может использоваться как для связи между Arduino/Freeduino устройствами, так и для связи с компьютером. На компьютере соответствующее соединение представлено либо обычным COM-портом (в случае Arduino Single-Sided Serial Board), либо USB COM-портом, который появляется в системе после установки необходимого драйвера.

Связь происходит по цифровым портам 0 и 1, и поэтому Вы не сможете использовать их для цифрового ввода/вывода если используете функции последовательной передачи данных.

Serial.begin(long);

Serial.begin(long); – устанавливает скорость передачи информации COM-порта битах в секунду для последовательной передачи данных. Для того чтобы поддерживать связь с компьютером, используйте одну из этих нормированных скоростей: 300, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, или 115200. Также Вы можете определить другие скорости при связи с другим микроконтроллером по портам 0 и 1. *Скорость_передачи* – скорость потока данных в битах в секунду.

```
Serial.begin(9600); //устанавливаем скорость 9600 бит/сек
```

Serial.available(void);

Serial.available(void); – принимаемые по последовательному порту байты попадают в буфер микроконтроллера, откуда программа может их считать. Функция возвращает количество накопленных в буфере байт. Последовательный буфер может хранить до 128 байт.

Возвращает значение типа *uint8_t (typedef uint8_t byte;)* – количество байт, доступных для чтения, в последовательном буфере, или 0, если ничего не доступно.

```
if (Serial.available() > 0) { // Если в буфере есть данные
   // здесь должен быть прием и обработка данных
 }
```

Serial.read(void);

Serial.read(void); – **с**читывает следующий байт из буфера последовательного порта. Возвращает первый доступный байт входящих данных с последовательного порта, или -1 если нет входящих данных.

```
incomingByte = Serial.read(); // читаем байт
```

Serial.write(uint8_t c)

Serial.write(uint8_t c) – Записывает данные в последовательный порт. Данные посылаются как байт или последовательность

байт; для отправки символьной информации следует использовать функцию print().

```
Serial.write(val);
Serial.write(str);
Serial.write(buf, len);
```

Здесь ***val*** – переменная для передачи, как единственный байт; ***str*** – строка для передачи, как последовательность байт; ***buf*** – массив для передачи, как последовательность байт; ***len*** – длина массива

Serial.flush(void)

Serial.flush(void) – очищает входной буфер последовательного порта. Находящиеся в буфере данные теряются, и дальнейшие вызовы *Serial.read()* или *Serial.available()* будут иметь смысл для данных, полученных после вызова *Serial.flush()*.

```
Serial.flush(); // Очищаем буфер – начинаем прием данных «с
// чистого листа»
```

Serial.print()

Serial.print() – вывод данных на последовательный порт.

Функции print наследуются классом HardwareSerial от класса Print (\hardware\cores\arduino\ Print.h)

Функция имеет несколько форм вызова в зависимости от типа и формата выводимых данных.

```
int b = 79;
Serial.print(b, DEC); //выдаст в порт строку «79».
```

```
int b = 79;
Serial.print(b, HEX); //выдаст в порт строку «4F»
```

```
int b = 79;
Serial.print(b, OCT); //выдаст в порт строку «117»
```

```
int b = 79;
Serial.print(b, BIN); //выдаст в порт строку «1001111»
```

```
int b = 79;
Serial.print(b, BYTE); //выведет число 79 (один байт). В мониторе
//последовательного порта получим символ «O» – его код равен 79
```

```
char bytes[3] = {79, 80, 81};    //массив из 5 байт со значениями 79,80,81
Serial.print("Here our bytes:"); //выводит строку «Here our bytes:»
Serial.print(bytes);   //выводит 3 символа с кодами 79,80,81
// – это символы «OPQ»
```

```
char b = 79;
Serial.print(b); //выдаст в порт символ «O»
```

```
int b = 79;
Serial.print(b); //выдаст в порт строку «79»
```

Для того, чтобы обмен данными происходил корректно, необходимо соблюсти общий протокол и скорость обмена данными. Это происходит при инициализации функции **Serial.begin(9600)/**

Serial.println()

Функция ***Serial.println*** аналогична функции *Serial.print*, и имеет такие же варианты вызова. Единственное отличие заключается в том, что после данных дополнительно выводятся два символа – символ возврата каретки (ASCII 13, или '\r') и символ новой линии (ASCII 10, или '\n'):

```
void Print::println(void)
{
  print('\r');
  print('\n');
}
```

Например, примеры программного кода 3.2 и 3.3 выведут в порт одно и то же:

Пример программного кода 3.2:

```
int b = 79
  Serial.print(b, DEC); //выдаст в порт строку «79»
  Serial.print("\r\n"); //выведет символы "\r\n" – перевод строки
  Serial.print(b, HEX); //выдаст в порт строку «4F»
  Serial.print("\r\n"); //выведет символы "\r\n" – перевод строки
```

Пример программного кода 3.3:

```
int b = 79;
  Serial.println(b, DEC); //выдаст в порт строку «79\r\n»
  Serial.println(b, HEX); //выдаст в порт строку «4F\r\n»
```

В мониторе последовательного порта получим:

Рис. 3.4. Отображение в мониторе последовательного порта результата приведенного выше программного кода

Serial monitor

Для начала попробуем управлять режимами работы светодиода с помощью ***Serial monitor.***

Соберем простенькую схему:

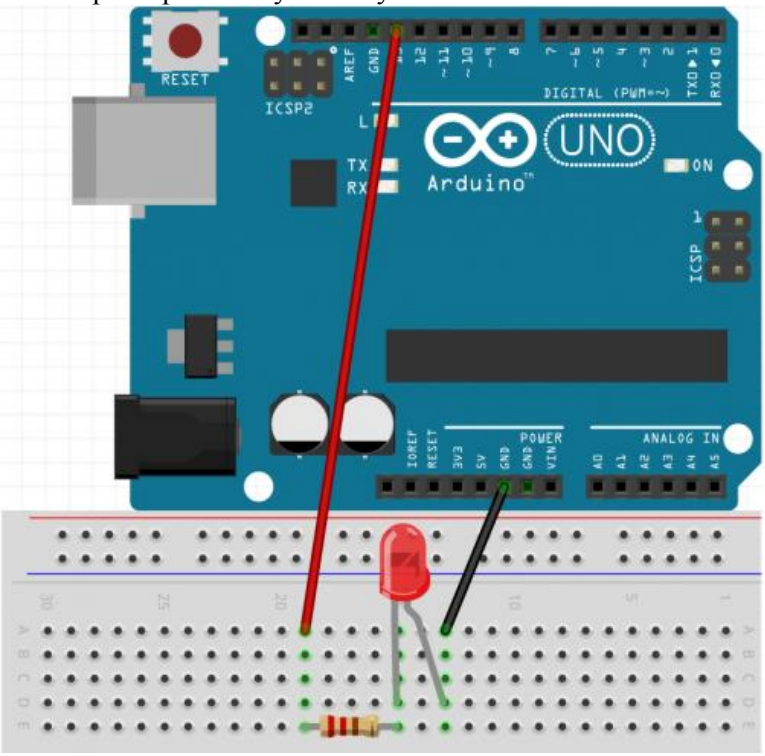

Рис. 3.5 Внешний вид подключения светодиода к плате Arduino

Пример программного кода 3.4:

```
#define LED 13

void setup() {
  Serial.begin(9600); //Установим соединение с компьютером.

  //Выведем в Serial два сообщения
  //Кстати, на русском сообщение будет отображаться некорректно.
  Serial.println("Write: on – for turn led on, off – for turn led off, blink – for blink");
  Serial.println("Wait for commands");
```

```
  pinMode(LED, OUTPUT);
}

void loop() {

  if(Serial.available() > 0) //Проверяем, пришла ли команда.
  {
   char cmd = Serial.read(); //Объявим переменную, в которую будем
//запоминать команду.

   //Команду можно ввести и капсом.
   //Для этого опишем возможные варианты пришедших команд.

   if (cmd == 'o' || cmd == 'O')      //Если пришла команда ON
    {
    digitalWrite(LED, HIGH);          //Включаем светодиод
    }
    if (cmd == 'f' || cmd == 'F')     //Если пришла команда OFF
    {
    digitalWrite(LED, LOW);           //Выключаем светодиод
    }
    if( cmd == 'b' || cmd == 'B')     //Если пришла команда Blink
    {
      for( int i = 0; i < 10; i++)    //Мигаем светодиодом
      {
      digitalWrite(LED, HIGH);
      delay(600);
      digitalWrite(LED, LOW);
      delay(600);
      }
    }
  }
}
```

Serial.print – с помощью этой команды мы можем выводить сообщения или какие-либо значения в ***Serial monitor***. Еще есть команда ***Serial.println***. Она так же выводит сообщение, но с новой строки.

Сообщение должно выводиться в виде: *Serial.println(“Текст сообщения”)*;

Serial.available() > 0. В этом куске кода мы проверяем, есть ли какие-то команды, а если есть, приступаем к определению, что за команда пришла и выполняем эту команду.

Переменная типа ***char*** может хранить один алфавитно-цифровой символ (литеру). При объявление литеры используются одиночные кавычки: 'O'

Serial.read – команда считывает команды, вводимые в *Serial monitor*.

Потенциометр и Serial monitor

Потенциометр или переменный резистор – это делитель из двух резисторов в одном корпусе. Сопротивление меняется поворотом ручки.

Мы соберем схему, которая будет в зависимости от поворота потенциометра отправлять в Serial monitor тестовые сообщения. Можно на основе скетча, вместо тестовых сообщений, выполнять нужные нам действия.

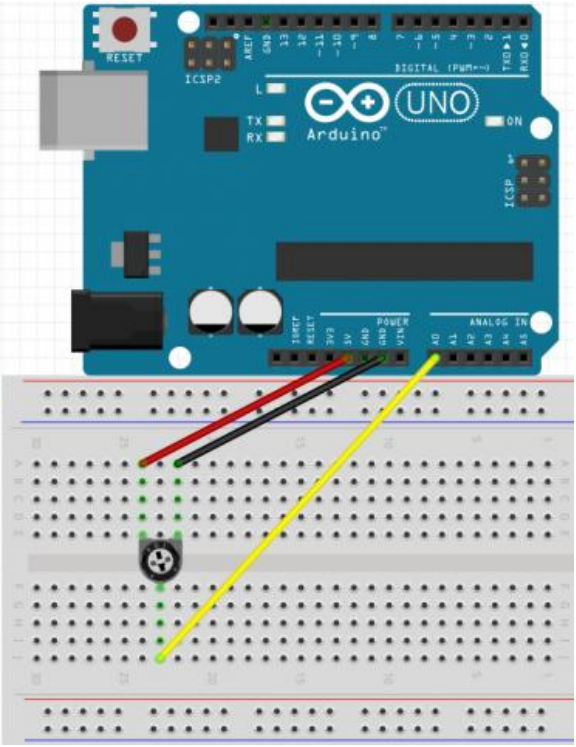

Рис. 3.6 Внешний вид подключения потенциометра к плате Arduino

Пример программного кода 3.5:

```
#define POT A0

void setup()
{
  pinMode(POT, INPUT);
  Serial.begin(9600);                //Установим связь с компьютером.
  Serial.println("Enter command V."); //Это сообщение просит
// ввести команду.

}

void loop()
{
  int pot = analogRead(POT);
  int val = analogRead(POT);
  val = map(val, 0, 1023, 0, 3);
// 250 градусов возможного поворота потенциометра разобьем
//на 4 части.
// По запросу, будем выводить на экран значение, считываемое с  по-
тенциометра.
 if(Serial.available() > 0)
  {
   char cmd = Serial.read();
   if( cmd == 'v' || cmd == 'V')
   {
   Serial.println(" Potentiometr's value: ");
   Serial.println(pot);
   delay(1000);
   }

//Здесь, вместо выведения сообщений в сериал может быть любой код.
//Получается, что поворачивая потенциометр мы меняем исполняемую
//в данный  момент команду.
   if(val == 0)
   {
    Serial.println("Command 0");
   }
   else if(val == 1)
   {
    Serial.println("Command 1");
   }
   else if(val == 2)
   {
    Serial.println("Command 2");
```

```
   }
    else if(val == 3)
   {
    Serial.println("Command 3");
   }
  }
}
```

Map – пропорционально переносит значение из текущего диапазона значений в новый диапазон, заданный параметрами.

В общем виде map можно записать так: *map (значение, изМеньш, изБольш, вМеньш, вБольш)*, где *Значение* – это значение, которое мы переносим, *изМеньш ,изБольш* – это крайние значения текущего диапазона значений, *вМеньш, вБольш* – это крайние значения нового диапазона значений.

Считываем последовательную строку ввода из последовательного монитора порта, ищем три разделенных запятыми целые числа с новой строкой в конце. Значения должны быть между 0 и 255. Программа использует эти значения для установки цвета светодиода RGB, прикрепленного к контактам 9-11.

Пример программного кода 3.6:

```
String inString = "";    // переменная для ввода потока данных
int currentColor = 0;
int red, green, blue = 0;
void setup() {
  // Открываем порт для обмена данными
  Serial.begin(9600);
  // Выводим надпись "\n\nString toInt() RGB:"
  Serial.println("\n\nString toInt() RGB:");
  Serial.println();
  // 9-й, 10-й, 11-й порты настраиваем на выход
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
}
void loop() {
  int inChar;// Создаём переменную для хранения входящего байта
//Считываем входное значение с порта и записываем в inChar
  if (Serial.available() > 0) {
    inChar = Serial.read();
  }
  if (isDigit(inChar)) {
    // конвертируем входящий байт в символьную переменную
```

```
    // и добавляем его в поток inString:
    inString += (char)inChar;
  }
  // если пришел символ "," то хранящееся значение в
  // inString присваивается текущему цвету
  // счетчик цвета увеличиваем на единицу
  if (inChar == ',') {
    switch (currentColor) {
      case 0:    // 0 = красный
        red = inString.toInt();
        // очищаем поток для записи новых данных
        inString = "";
        break;
      case 1:    // 1 = зелёный
        green = inString.toInt();
        // очищаем поток для записи новых данных
        inString = "";
        break;
    }
    currentColor++;
  }
  // если пришел символ перевода строки
  // то хранящееся значение в inString присваиваем синему цвету
  if (inChar == '\n') {
    blue = inString.toInt();
    // устанавливаем на выводах 9,10,11 шим сигнал
    // соответствующий величине хранящейся в переменных
    // red, green, blue. При значении 0 светодиод не горит
    // при значении 255 светодиод светит максимально ярко
    analogWrite(11,  red);
    analogWrite(9, green);
    analogWrite(10, blue);
    // выводим в монитор порта цвет и его текущее значение
    Serial.print("Red: ");
    Serial.print(red);
    Serial.print(", Green: ");
    Serial.print(green);
    Serial.print(", Blue: ");
    Serial.println(blue);
    // очищаем string
    inString = "";
    // сбрасываем счётчик
    currentColor = 0;
  }
}
```

Задания и порядок выполнения работы

1. Управлять яркостью 2-х трехцветных светодиодов (цвета должны переливаться независимо друг от друга)
2. Управлять яркостью 2-х трехцветных светодиодов по сигналу с компьютера

Контрольные вопросы

1. Введите понятие функции? Тело функции?
2. Какова структура программы для платформы Arduino?
3. На каком языке пишутся программы для Arduino?
4. Какие типы данных могут использоваться в программе?
5. Перечислите существующие операторы сравнения.
6. Назовите назначение функции Serial.begin(9600).
7. Назовите назначение функции Serial.print().
8. Назовите назначение функции Serial.println().
9. Поясните программный код изменения цвета светодиода.
10. Назовите назначение функции loop().
11. Назовите назначение функции millis().
12. Назовите назначение функции delay().
13. Назовите назначение функции setup().
14. Назовите назначение функции pinMode().
15. Назовите назначение функции digitalWrite().
16. Назовите назначение функции digitalRead().
17. Назовите назначение функции inString.toInt().

Лабораторная работа № 5.
Подключение джойстика к плате Arduino

Цель работы: приобрести практические навыки по подключению джойстика к плате Arduino.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

Принцип действия аналогового джойстика

Джойстик – удобное и лёгкое в использовании устройство для передачи информации. Видов джойстиков по количеству степеней свободы, принципу считывания показаний и используемым технологиям существует большое количество. Джойстики чаще всего используются для управления движением каких-либо механизмов, управляемых моделей, роботов.

Аналоговый джойстик, который мы сегодня рассмотрим, представляет собой ручку, закреплённую на шаровом шарнире с двумя взаимно перпендикулярными осями. При наклоне ручки, ось вращает подвижный контакт потенциометра, благодаря чему изме-

няется напряжение на его выходе. Сам геймпад оснащен пружиной, благодаря которой плавно возвращается в первоначальное центральное состояние после отпускания его с какой-либо позиции.

Рис. 5.1. Аналоговый джойстик

Устройство позволяет более плавно отследить степень отклонения от центральной (нулевой) точки. Также аналоговый джойстик имеет тактовую кнопку, которая срабатывает при вертикальном надавливании на ручку.

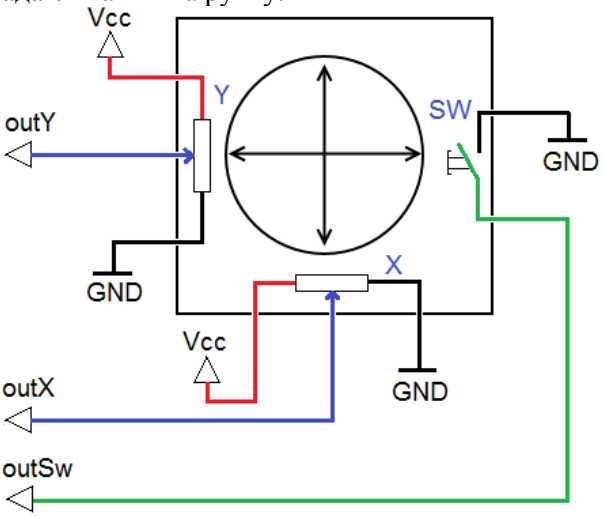

Рис. 5.2. Принципиальная схема аналогового джойстика

Подключим джойстик по приведённой на рис. 5.3. схеме. Аналоговые выходы X и Y джойстика подключим к аналоговым входам A1 и A2 Arduino, выход кнопки SW – к цифровому входу 2. Питание джойстика осуществляется напряжением +5 В.

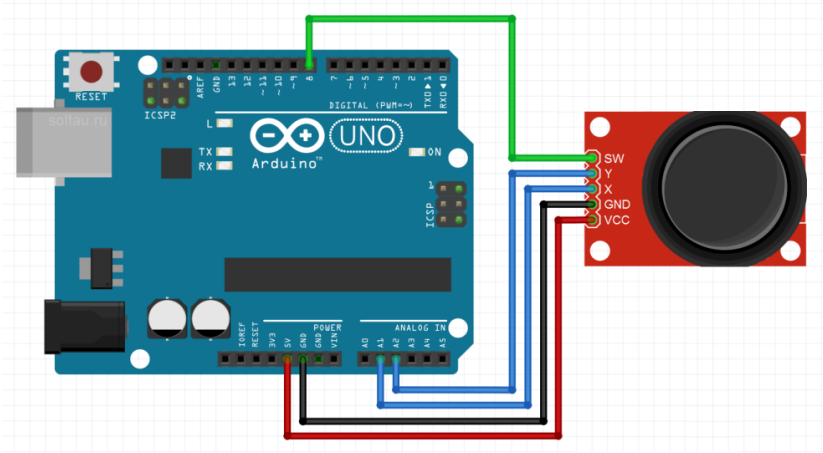

Рис. 5.3. Схема подключения аналогового джойстика к Arduino

Для того чтобы наглядно увидеть, как работает джойстик, напишем такой скетч:

Пример программного кода 5.1:

```
const int switchPin = 2;  // кнопка джойстика
const int pinX = A1; // Ось X джойстика
const int pinY = A2; // Ось Y джойстика
const int ledPin = 13;

void setup() {
  pinMode(ledPin, INPUT);
  pinMode(pinX, INPUT);
  pinMode(pinY, INPUT);
  pinMode(switchPin, INPUT);
  digitalWrite(switchPin, HIGH); // включаем встроенный
//подтягивающий резистор
  Serial.begin(9600);
}

void loop() {
  int ledState = digitalRead(switchPin); // считываем состояние кнопки
  digitalWrite(ledPin, ledState); // вкл./выкл. светодиод по нажатию
```

```
кнопки
  int X = analogRead(pinX); // считываем аналоговое значение оси X
  int Y = analogRead(pinY); // считываем аналоговое значение оси Y
  Serial.print(X); // Выводим значение в Serial Monitor
  Serial.print("\t"); // табуляция
  Serial.println(Y);
}
```

На мониторе порта увидим следующее:

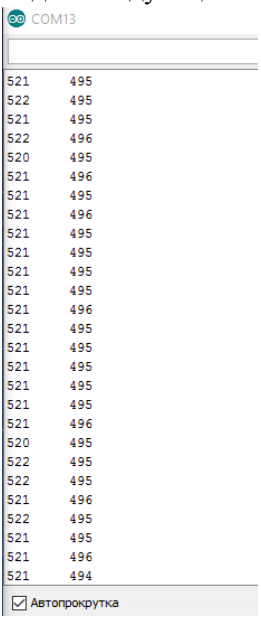

Рис. 5.4. Значения X и Y с аналогового джойстика, выведенные в монитор последовательного порта

Объявим пины, зададим им режимы работы. Обратите внимание, в процедуре setup() мы подали на вход switchPin высокий уровень. Этим мы включили встроенный подтягивающий резистор на этом порту. Если его не включить, то, когда кнопка джойстика не нажата, 8-ой порт Arduino будет висеть в воздухе и ловить наводки. Это повлечёт за собой нежелательные хаотичные ложные срабатывания.

В процедуре loop() мы постоянно опрашиваем состояние кнопки и отображаем его с помощью светодиода на выходе 13. Из-

за того, что вход switchPin подтянут к питанию, светодиод постоянно горит, а при нажатии кнопки гаснет, а не наоборот.

Далее мы считываем показания двух потенциометров джойстика – выхода осей X и Y. Arduino имеет 10-разрядные АЦП, поэтому значения, снимаемые с джойстика, лежат в диапазоне от 0 до 1023. В среднем положении джойстика, как видно на иллюстрации, снимаются значения в районе 500 – примерно середина диапазона.

Управление яркостью и цветом светодиода с помощью аналогового джойстика и Arduino

Обычно джойстик используют для управления электродвигателями. Но почему бы не использовать его, например, для управления яркостью светодиода? Давайте подключим по приведённой схеме RGB светодиод (или три обычных светодиода) к цифровым портам 9, 10 и 11 Arduino, не забывая, конечно, о резисторах.

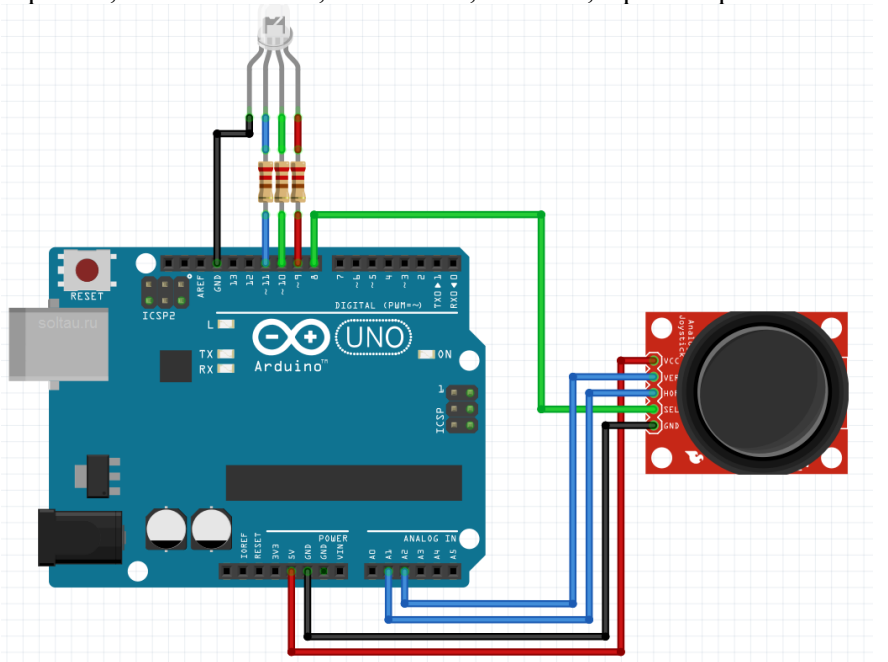

Рис. 5.5. Подключение RGB светодиода и джойстика к Arduino

Будем менять яркость соответствующих цветов при изменении положения джойстика по осям, как показано на рисунке.

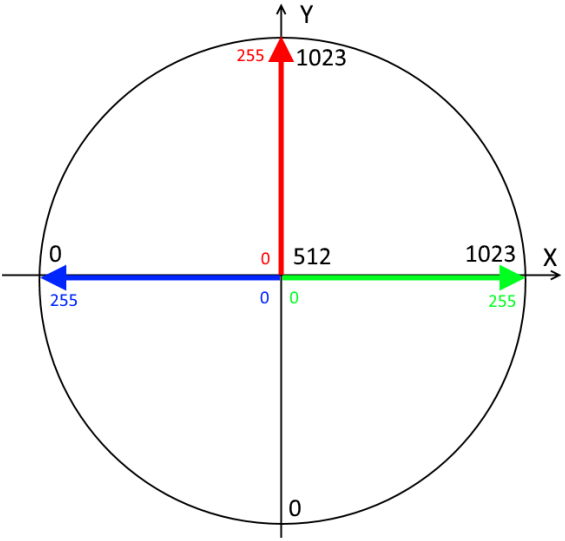

Рис. 5.6. Диаграмма распределения яркости красного, синего и зелёного каналов светодиода в зависимости от положения ручки джойстика

Из-за того, что джойстик может быть не точно отцентрирован производителем и иметь середину шкалы не на отметке 512, а варьироваться в диапазоне примерно от 490 до 525, то светодиод может слегка светиться даже когда джойстик находится в нейтральном положении. Если вы хотите, чтобы он был полностью выключен, то внесите в программу соответствующие поправки.

Ориентируясь на приведённую диаграмму, напишем скетч управления Arduino яркостью RGB светодиода с помощью джойстика.

Пример программного кода 5.2. Изменение яркости красного, синего и зелёного каналов светодиода в зависимости от положения ручки джойстика и нажатия кнопки:

```
const int pinRed    = 9;
const int pinGreen  = 10;
const int pinBlue   = 11;
const int swPin = 2;
const int pinX      = A1; // X
```

```
const int pinY      = A2; // Y
const int ledPin    = 13;
boolean ledOn = false;  // текущее состояние кнопки
boolean prevSw = false; // предыдущее состояние кнопки

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(pinRed, OUTPUT);
  pinMode(pinGreen, OUTPUT);
  pinMode(pinBlue, OUTPUT);
  pinMode(pinX, INPUT);
  pinMode(pinY, INPUT);
  pinMode(swPin, INPUT);
  digitalWrite(swPin, HIGH); // включаем встроенный подтягивающий
// резистор
}

void loop() {
  if (isLedOn()) freeMode(); // если нажата кнопка и горит светодиод
//на пине 13, включаем режим "фонарик"
  else discoMode(); // иначе включаем "цветомузыку"
}

boolean isLedOn() { // ОПРЕДЕЛЯЕМ НАЖАТИЕ КНОПКИ
  if (digitalRead(swPin) == HIGH && prevSw == LOW) {
    ledOn = !ledOn;
    prevSw = HIGH;
  }
  else  prevSw = digitalRead(swPin);
  digitalWrite(ledPin, ledOn); // включаем светодиод на пине 13
  return ledOn;
}

void freeMode() { // РЕЖИМ "ФОНАРИК"
  int X = analogRead(pinX); // считываем положение джойстика
  int Y = analogRead(pinY);
  int RED = map(Y, 512, 1023, 0, 255); // маппинг значений
  int GREEN = map(X, 512, 1023, 0, 255);
  int BLUE = map(X, 511, 0, 0, 255);
  analogWrite(pinRed, RED);     // включение каналов R,G,B
  analogWrite(pinGreen, GREEN);
  analogWrite(pinBlue, BLUE);
}

void discoMode() { // РЕЖИМ "ЦВЕТОМУЗЫКА"
```

```
  for (int i=0; i <= 255; i++) {
    if (isLedOn()) { break; } // при нажатии кнопки выходим из цикла
    analogWrite(pinRed, i); // работает канал RED
    analogWrite(pinGreen, 0);
    analogWrite(pinBlue, 0);
    delay(5);
  }
  for (int i=0; i <= 255; i++) {
    if (isLedOn()) { break; } // при нажатии кнопки выходим из цикла
    analogWrite(pinRed, 0);
    analogWrite(pinGreen, 0);
    analogWrite(pinBlue, i);  // работает канал BLUE
    delay(5);
  }
  for (int i=0; i <= 255; i++) {
    if (isLedOn()) { break; } // при нажатии кнопки выходим из цикла
    analogWrite(pinRed, 0);
    analogWrite(pinGreen, i); // работает канал GREEN
    analogWrite(pinBlue, 0);
    delay(5);
  }
}
```

Сначала объявим соответствие пинов и две переменные – ***ledOn*** и ***prevSw*** – для работы с кнопкой. В процедуре setup() назначим пинам функции и подключим к пину кнопки подтягивающий резистор командой ***digitalWrite(swPin, HIGH)***.

В цикле ***loop()*** определяем нажатие кнопки джойстика. При нажатии на кнопку переключаем режимы работы между режимом «фонарика» и режимом «цветомузыки».

В режиме ***freeMode()*** управляем яркостью светодиодов с помощью наклона джойстика в разные стороны: чем сильнее наклон по оси, тем ярче светит соответствующий цвет. Причём преобразование значений берёт на себя функция *map (значение, отНижнего, отВерхнего, кНижнему, кВерхнему)*.

Функция ***map()*** очень полезна и удобна в применении. Она переносит измеренные значения (отНижнего, отВерхнего) по осям джойстика в желаемый диапазон яркости (кНижнему, кВерхнему). Можно то же самое сделать обычными арифметическими действиями, но запись с помощью функции ***map()*** существенно короче.

В режиме ***discoMode()*** три цвета попеременно набирают яркость и гаснут. Чтобы можно было выйти из цикла при нажатии кнопки, каждую итерацию проверяем, не была ли нажата кнопка.

В результате получился фонарик из трёхцветного RGB светодиода, яркость свечения каждого цвета которого задаётся с помощью джойстика. А при нажатии на кнопку происходит включение режима «цветомузыка».

Задания и порядок выполнения работы

1. Управлять яркостью 4-х светодиодов при помощи джойстика
2. Управление яркостью 4-х светодиодов осуществлять только при нажатии кнопки

Контрольные вопросы

1. Какие операторы анализа условий применяются в программировании для Arduino?
2. Поясните синтаксис оператора if – else.
3. Введите понятие цикла.
4. Назовите назначение функции analogWrite().
5. Назовите назначение функции analogRead().
6. Назовите назначение функции map(Value, 0, 1023, 0, 255).

Лабораторная работа № 6.
Гашение дребезга контактов программным способом

Цель работы: приобрести практические навыки по устранению дребезга контактов при подключении кнопок к микроконтроллерам.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

«Дребезг» контактов

Кнопка – очень простое и полезное изобретение, служащее для лучшего взаимодействия человека и техники. Но, как и всё в природе, она не идеальна. Проявляется это в том, что при нажатии на кнопку и при её отпускании возникает т.н. «дребезг» («bounce» по-английски). Это многократное переключение состояния кнопки за короткий промежуток времени (порядка нескольких миллисекунд), прежде чем она примет установившееся состояние. Это нежелательное явление возникает в момент переключения кнопки из-за упругости материалов кнопки или из-за возникающих при электрическом контакте микроискр.

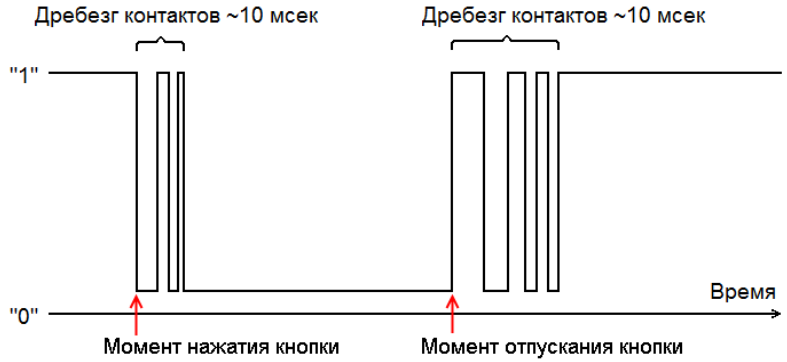

Рис. 6.1. Схематическое изображение дребезга при нажатии кнопки

«Дребезг» контактов – это явление, свойственное механическим переключателям, кнопкам, тумблерам и реле. Из-за того, что контакты обычно делают из металлов и сплавов, которые обладают упругостью, при физическом замыкании они не сразу устанавливают надёжное соединение.

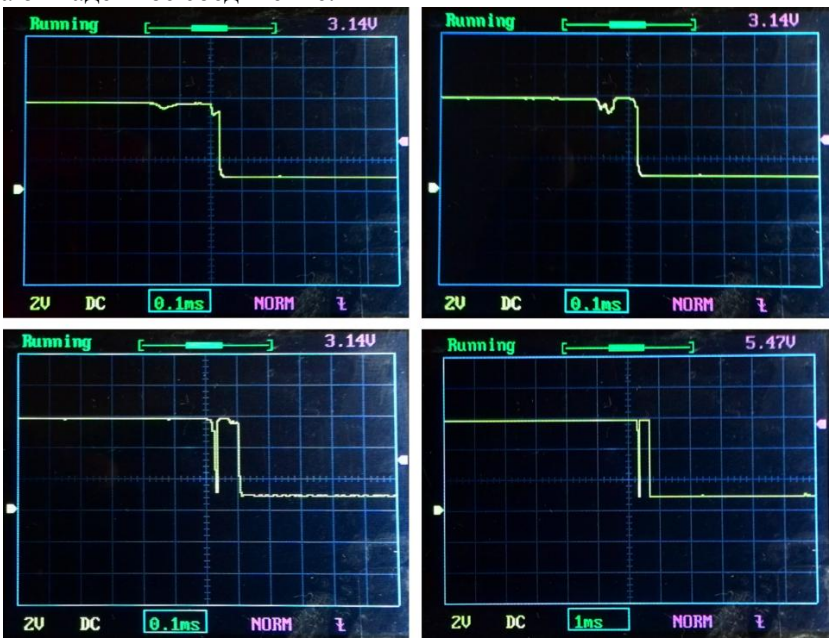

Рис. 6.2. Эффект дребезга контактов на осциллограммах

В течение короткого промежутка времени контакты несколько раз смыкаются и отталкиваются друг от друга. В результате этого электрический ток принимает установившееся значение не моментально, а после череды нарастаний и спадов.

Длительность этого переходного эффекта зависит от материала контактов, от их размера и конструкции. На иллюстрации показана типичная осциллограмма при замыкании контактов тактовой кнопки. Видно, что время от момента переключения до установившегося состояния составляет несколько миллисекунд. Это и называется «дребезгом».

Этот эффект не заметен в электрических цепях управления освещением, двигателями или другими *инерционными* датчиками и приборами.

Но в цепях, где идёт быстрое считывание и обработка информации (где частоты того же порядка, что и импульсы «дребезга», или выше), это является проблемой. В частности, Arduino UNO, который работает на частоте 16 МГц, отлично ловит «дребезг» контактов, принимая последовательность единиц и нулей вместо единичного переключения от 0 к 1.

Подключим к Arduino тактовую кнопку по схеме с подтягивающим резистором. Будем по нажатию кнопки зажигать светодиод и оставлять включённым до повторного нажатия кнопки. Для наглядности подключим к цифровому выводу 13 внешний светодиод, хотя можно обойтись и встроенным.

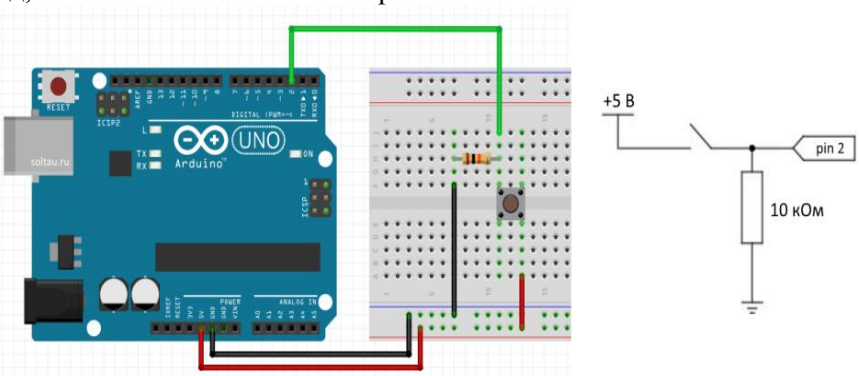

Рис. 6.3. Схема подключения кнопки к Arduino для демонстрации подавления эффекта «дребезга» контактов

Алгоритм подавления «дребезга» контактов

Чтобы реализовать задачу подавления дребезга контактов, можно:

- запоминать предыдущее состояние кнопки;
- сравнивать с текущим состоянием;
- если состояние изменилось, то меняем состояние светодиода.

Напишем такой скетч и загрузим в память Arduino.

Пример программного кода 6.1. Включение светодиода по нажатию кнопки и выключение при повторном нажатии.:

```
int switchPin = 2; // вывод считывания кнопки
int ledPin = 13; // вывод светодиода
boolean lastButton = false; // предыдущее состояние кнопки
boolean ledOn = false; // включён или выключен светодиод

void setup() {
  pinMode(switchPin, INPUT); // состояние кнопки считываем (in)
  pinMode(ledPin, OUTPUT); // светодиод запитываем (out)
}

void loop() {
  int pressed = digitalRead(switchPin); /* состояние кнопки:
  HIGH, true – нажата, LOW, false – нет */
  if (pressed == true && lastButton == false) { /* если кнопка сейчас
// нажата, а до этого была не нажата */
    ledOn = !ledOn; // меняем состояние светодиода
    lastButton = true; // запоминаем новое состояние кнопки
  }
  else {
    lastButton = digitalRead(switchPin); // считываем состояние кнопки
  }
  digitalWrite(ledPin, ledOn); // зажигаем или гасим светодиод
}
```

Подавление дребезга контактов с помощью задержки

Постараемся исправить ситуацию. Мы знаем, что дребезг контактов проявляет себя в течение нескольких миллисекунд после замыкания контактов. Давайте после изменения состояния кнопки выжидать, скажем, 5мсек. Это время для человека является практически мгновением, и нажатие кнопки человеком обычно происходит значительно дольше – несколько десятков миллисекунд. А

Arduino прекрасно работает с такими короткими промежутками времени, и эти 5мсек позволят ему отсечь дребезг контактов от нажатия кнопки.

Пример программного кода 6.2. Подавление дребезга контактов с помощью задержки:

```
int switchPin = 2; // пин кнопки
int ledPin = 13; // пин светодиода
boolean lastButton = false; // предыдущее состояние кнопки
boolean currentButton = false; // текущее состояние кнопки
boolean ledOn = false; // состояние светодиода

void setup() {
  pinMode (switchPin, INPUT);
  pinMode (ledPin, OUTPUT);
}

void loop() {
  currentButton = debounce (lastButton); // получаем состояние кнопки
//без дребезга
  if (lastButton == false && currentButton == true) { // если кнопка была
// нажата дольше 5 мсек,
    ledOn = !ledOn; // то меняем состояние светодиода
  }
  lastButton = currentButton; // обнуляем состояние нажатия кнопки
  digitalWrite (ledPin, ledOn); // зажигаем/гасим светодиод
}

// Процедура определения нажатия кнопки без дребезга:
boolean debounce(boolean last) {
  boolean current = digitalRead(switchPin); // считываем текущее
состояние кнопки
  if (last != current) { // если состояние изменилось
    delay(5); // делаем задержку на 5 мсек, пока уляжется дребезг
    current = digitalRead(switchPin); // и считываем снова
  }
  return current; // возвращаем текущее состояние кнопки
}
```

В данном скетче мы объявим процедуру debounce() («bounce» по-английски – это как раз «дребезг», приставка «de» означает обратный процесс), на вход которой мы подаём предыдущее состояние кнопки. Если нажатие кнопки длится более 5мсек, значит это действительно нажатие. Определив нажатие, мы меняем состояние светодиода.

Загрузим скетч в плату Arduino. Кнопка срабатывает без сбоев, при нажатии светодиод меняет состояние, как мы и хотели.

Библиотеки для подавления дребезга контактов

Аналогичная функциональность обеспечивается специальными библиотеками, например, библиотекой **Bounce2**. Для установки библиотеки помещаем её в директорию /libraries/ среды разработки Arduino и перезапускаем IDE.

Библиотека **Bounce2** содержит следующие функции:

Таблица 6.1. Функции библиотеки Bounce2

Название	Назначение
Bounce()	инициализация объекта «Bounce»;
void interval (мсек)	устанавливает время задержки в миллисекундах;
void attach (номерПина)	задаёт вывод, к которому подключена кнопка;
int update()	обновляет объект и возвращает true, если состояние пина изменилось, и false в противном случае;
int read()	считывает новое состояние пина.

Перепишем наш скетч с использованием библиотеки. Можно также запоминать и сравнивать прошлое состояние кнопки с текущим, но давайте упростим алгоритм.

Пример программного кода 6.3:

```
#include <Bounce2.h> // подключаем библиотеку

const int switchPin = 2; // пин кнопки
const int ledPin = 13; // пин светодиода
int cnt = 0; // счётчик нажатий

Bounce b = Bounce(); // инициализируем объект Bounce

void setup() {
  pinMode(switchPin, INPUT);
  digitalWrite(switchPin, HIGH); // включаем подтягивающий резистор
  pinMode(ledPin, OUTPUT);
  b.attach(switchPin); // объект Bounce будет слушать кнопку на пине
//switchPin
  b.interval(5); // устанавливаем время задержки в [мс]
}

void loop() {
  if (b.update() && b.read() == 0) { // если зарегистрировано событие и
//кнопка нажата,
    cnt += 1; // инкрементируем счётчик нажатий
```

```
    if (cnt %2 == 0) digitalWrite(ledPin, LOW); // если нажатий чётное
  //число, гасим светодиод
    else digitalWrite(ledPin, HIGH); // иначе – зажигаем светодиод
   }
 }
```

При нажатии кнопки будем считать нажатия, и каждое нечётное нажатие будем включать светодиод, каждое чётное – выключать. Такой скетч смотрится лаконично, его легко прочитать и легко применить.

Задания и порядок выполнения работы

1. Написать программу, опрашивающую две кнопки и зажигающую два светодиода без использования специальной библиотеки.
2. Написать программу, опрашивающую две кнопки и зажигающую два светодиода с использованием специальной библиотеки

Контрольные вопросы

1. Для чего используются макетные платы?
2. Что такое дребезг контактов?
3. Какой алгоритм подавления дребезга контактов?
4. Как подавить дребезг контактов?
5. Для чего производят экранирование макетной платы?
6. Каковы особенности подготовки макетной платы к работе?

Лабораторная работа № 7.
Подключение матрицы кнопок к микроконтроллеру

Цель работы: приобрести практические навыки по устранению дребезга контактов при подключении матричных клавиатур к микроконтроллерам.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

В предыдущих лабораторных работах мы подключали кнопки к контроллеру, используя для каждой отдельный вывод. Если необходимо использовать в системе большое число кнопок, такое решение может оказаться неприемлемым. Просто не хватит выводов микроконтроллера.

Существует другой способ подключения кнопок к плате Arduino – объединение кнопок в матрицу.

При необходимости использования в устройстве клавиатуры с большим количеством кнопок, например в кодовом замке, очень часто применяют матричную клавиатуру. Если подключить 12 кнопок обычным способом потребуется 12 выводов микроконтрол-

лера плюс общий провод, в матрице же используется всего один порт контроллера, что способствует экономии выводов контроллера. Кнопки в такой клавиатуре подключаются к общим столбцам и к общим строкам, линии порта микроконтроллера разделяются на ввод PB7-PB4 и вывод PB3-PB0. В каждый момент времени сигнал низкого уровня (логический ноль) подается только на одну строку кнопок, на остальные должна подаваться логическая единица. Это исключит неоднозначность определения номера нажатой кнопки. Двоичные сигналы, присутствующие при этом на столбцах клавиатуры, считываются через порт ввода микроконтроллера.

Подключение матрицы кнопок к микроконтроллеру.

Вот пример схемы такой матрицы. Для подключения 16 кнопок требуется только 8 выводов микроконтроллера.

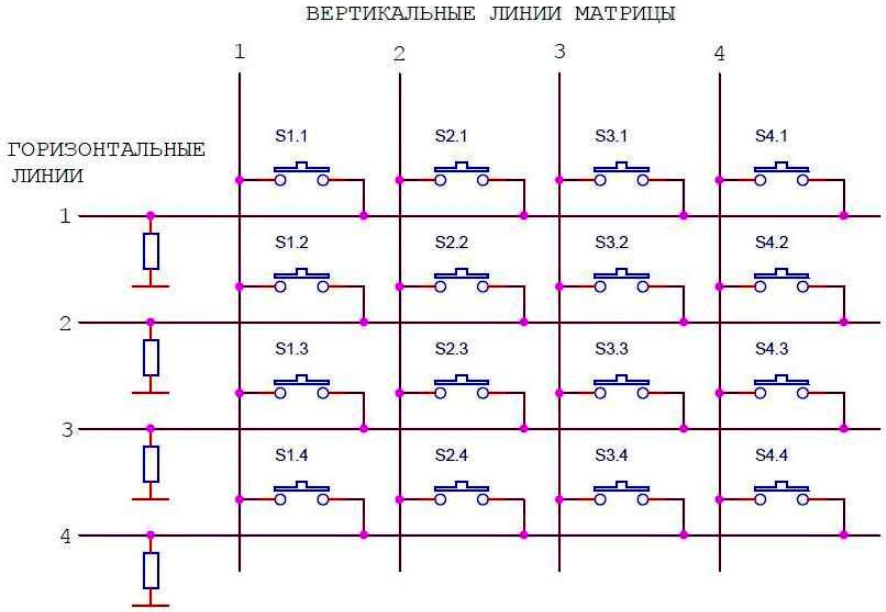

Рис. 7.1. Электрическая принципиальная схема матричной клавиатуры 4x4

Кнопки подключены к вертикальным и горизонтальным линиям матрицы. Состояние кнопок для каждой вертикальной линии

проверяется отдельно. На вертикальную линию подается сигнал высокого уровня (5В) и считываются состояния горизонтальных линий. Высокий уровень в горизонтальной линии покажет, что соответствующая кнопка нажата. Далее проверяются остальные вертикальные линии.

Резисторы обеспечивают нулевое напряжение на входах микроконтроллера при разомкнутых кнопках.

Такая схема часто используется для подключения матрицы кнопок к плате Arduino в практических приложениях, не смотря на то, что эта схема работает некорректно при одновременном нажатии двух и более кнопок из одной горизонтальной линии. В этом случае результат будет непредсказуемым. Через горизонтальную линию и две нажатые кнопки замкнутся вертикальные линии с высоким и низким уровнями сигнала. Какая вертикальная линия перетянет, такой уровень и будет на горизонтальной линии.

Произойдет замыкание выводов микроконтроллера с разными уровнями сигналов. В лучшем случае повысится потребляемый ток, и микроконтроллер будет нагреваться.

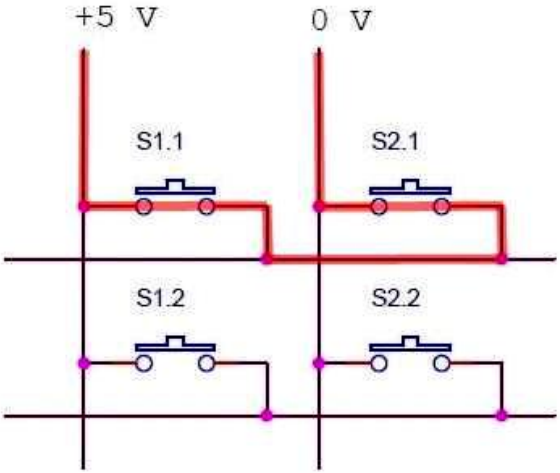

Рис. 7.2. Схематическое представление замыкания выводов микроконтроллера с разными уровнями сигналов при одновременном нажатии кнопок S1.1 и S2.1

Для устранения этой проблемы достаточно подключить вертикальные линии через диоды. Тогда замыкание выводов микроконтроллера невозможно при любом количестве одновременно нажатых кнопок. Также схема с диодами позволит корректно определять состояние матрицы с двумя одновременно нажатыми кнопками. При нажатых трех соседних кнопках произойдет замыкание четвертой.

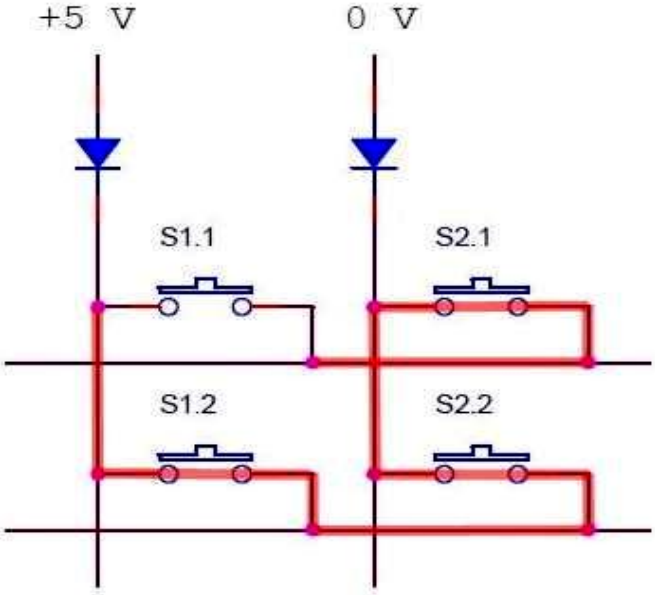

Рис. 7.3. Схематическое представление устранения замыкания выводов микроконтроллера при использовании диодов

Подключение матрицы кнопок к плате Arduino.

Подключим матрицу кнопок 3x4 к плате Arduino по схеме, приведенной ниже.

Нажатие каждой кнопки желательно сопровождать коротким звуковым сигналом. Для этого подключим к плате звуковой пьезоизлучатель.

Для устранения дребезга кнопок используется способ ожидания стабильного состояния контактов, описанный в предыдущей

лабораторной. В параллельном процессе должен регулярно вызываться метод scanState().

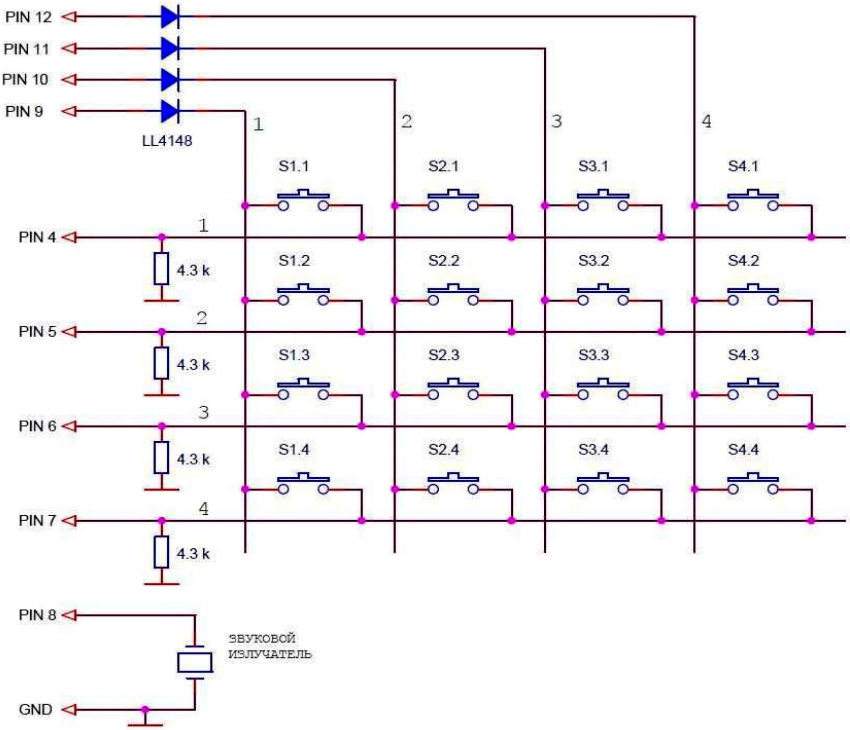

Рис. 7.4. Схема подключения матричной клавиатуры 4x4 и звукового излучателя к плате Arduino

В результате формируются признаки массивов *flagPress[4][4]* и *flagClick[4][4]*:

- при нажатой кнопке flagPress= true;
- при отжатой кнопке flagPress= false;
- при нажатии на кнопку flagClick= true.

Объект типа MatrixKeys при создании имеет параметры:

- номера выводов подключения вертикальных линий матрицы 1, 2, 3 ,4;
- номера выводов подключения горизонтальных линий матрицы 1, 2, 3 ,4;

– число подтверждений состояния контактов.

Пример создания объекта:

```
// создаем объект матрица кнопок keys
// подключаем вертикальные линии к выводам 9, 10, 11, 12
// подключаем горизонтальные линии к выводам 4, 5, 6, 7
// число подтверждений состояния контактов 6
MatrixKeys  keys(9, 10, 11, 12, 4, 5, 6, 7, 6);
```

Конкретные выводы матрицы можно отключить, задав для них в конструкторе номера равные 255. Это может потребоваться при подключении матрицы меньшей размерности.

Для генерации звука по нажатию кнопок можно использовать стандартную функцию tone().

Tone() – стандартная функция генерации звука, генерирует на заданном выводе сигнал прямоугольной формы.

void tone(pin, frequency)

void tone(pin, frequency, duration)

где *pin* – номер вывода; *frequency* – частота сигнала в Гц; *duration* – длительность сигнала в миллисекундах.

Если длительность сигнала не задана третьим аргументом, то сигнал вырабатывается до тех пор пока не будет вызвана функция noTone().

Необходимо помнить, что для генерации сигнала функция tone() использует Таймер 2 платы Arduino. Поэтому, если этот таймер уже используется в программе, например для формирования прерывания, то функция tone() приведет к конфликту обращения к таймеру 2.

Библиотека сканирования матричной клавиатуры 4 х 4.

В отличие от известной библиотеки Keypad эта библиотека:

– значительно компактнее и быстрее;
– ориентирована на использование в параллельном процессе;
– надежно обрабатывает дребезг контактов, позволяет задавать число подтверждений состояния кнопок.

Напишем простую программу для проверки и демонстрации библиотеки MatrixKeys.h. Программа передает на компьютер по последовательному порту состояние матрицы кнопок:

– нажатая кнопка отображается как «*»;
– отжатая – «.»;

– момент нажатия кнопки (клик) отображается символом «=» в течение 0,5 секунд.

Пример программного кода 7.1. Матрица кнопок:

```
// матрица кнопок
#include <MatrixKeys.h>
#include <MsTimer2.h>
// создаем объект матрица кнопок keys
// подключаем вертикальные линии к выводам 9, 10, 11, 12
// подключаем горизонтальные линии к выводам 4, 5, 6, 7
// число подтверждений состояния контактов 6
MatrixKeys  keys(9, 10, 11, 12, 4, 5, 6, 7, 6);
void setup() {
  Serial.begin(9600); // инициализируем порт, скорость 9600
  MsTimer2::set(2, timerInterrupt); // задаем период прерывания по
//таймеру 2 мс
  MsTimer2::start();              // разрешаем прерывание по таймеру
}
void loop() {
  // перебор строк
  for (int i = 0; i < 4; i++) {

    // перебор столбцов
    for (int j = 0; j < 4; j++) {
          if     (keys.flagClick[j][i]     ==     true)     {     Serial.print("=");
keys.flagClick[j][i]=false; tone(8,1000,50); }
      else { if (keys.flagPress[j][i] == true) Serial.print("*"); else Seri-
al.print("."); }
    }
     Serial.println(" ");
  }
  Serial.println(" ");
  delay(500);
}
//--------------------------------------- обработчик прерывания 2 мс
void  timerInterrupt() {
  keys.scanState(); // сканирование матрицы
}
```

Загрузите программу в плату. Не забудьте установить библиотеки MatrixKeys.h и MsTimer2.h. Откройте монитор последовательного порта. На экране, каждые 0,5 секунды, будут пробегать блоки состояния матрицы.

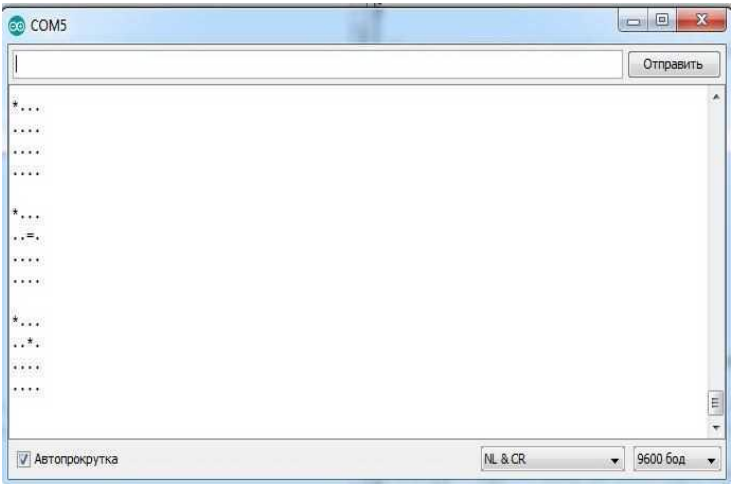

Рис. 7.5. Вывод блоков состояния матрицы в зависимости от нажатия кнопок в монитор последовательного порта

Нажимая на кнопки матрицы можно проверить работу программы.

Если необходимо передавать код нажатой клавиши, то можно преобразовать состояние массива признаков кликов keys.flagClick[4][4] в коды кнопок codKeys[4][4].

Пример программного кода 7.2:

```
// матрица кнопок
#include <MatrixKeys.h>
#include <MsTimer2.h>
// массив кодов кнопок
const char codKeys[4][4] =
{ {'1', '4', '7', '*'},
  {'2', '5', '8', '0' },
  {'3', '6', '9', '#'},
  {' ', ' ', ' ', ' '}
};
// создаем объект матрица кнопок keys
// подключаем вертикальные линии к выводам 9, 10, 11, 12
// подключаем горизонтальные линии к выводам 4, 5, 6, 7
// число подтверждений состояния контактов 6
MatrixKeys  keys(9, 10, 11, 12, 4, 5, 6, 7, 6);
void setup() {
  Serial.begin(9600); // инициализируем порт, скорость 9600
```

```
  MsTimer2::set(2, timerInterrupt); // задаем период прерывания по таймеру 2 мс
  MsTimer2::start();              // разрешаем прерывание по таймеру
}
void loop() {
  // вычисление кода нажатой кнопки
  // перебор столбцов
  for (int i = 0; i < 4; i++) {
    // перебор строк
    for (int j = 0; j < 4; j++) {
      if (keys.flagClick[i][j] == true) {
        keys.flagClick[i][j]=0;
        Serial.println(codKeys[i][j]);
        }
    }
  }
}
//------------------------------------ обработчик прерывания 2 мс
void  timerInterrupt() {
  keys.scanState(); // сканирование матрицы

}
```

Если загрузить программу в плату и открыть монитор последовательного порта, то при каждом нажатии кнопки в окне монитора будет отображаться символ, соответствующий кнопке.

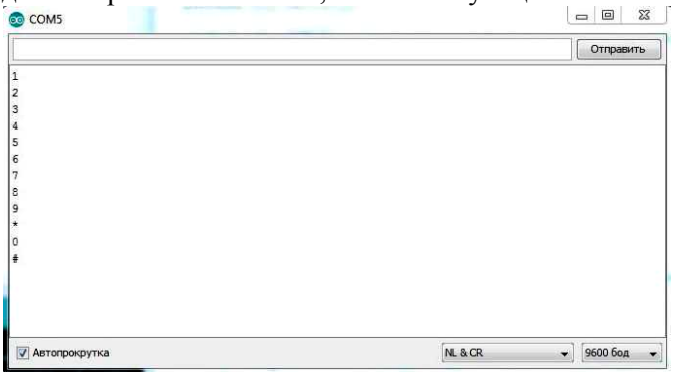

Рис. 7.6. Символы, соответствующие кнопке

При необходимости Вы легко сможете переделать библиотеку на матрицу кнопок других размерностей.

Задания и порядок выполнения работы

1. Задать произвольные символы и цифры, которые будут выводиться при нажатии кнопок матричной клавиатуры

2. Задать произвольные фразы, которые будут выводиться при нажатии кнопок матричной клавиатуры (например: фамилию, имя, отчество и т.д.)

Контрольные вопросы

1. Как подключается матричная клавиатура к микроконтроллерам?
2. Какие особенности подключения матричной клавиатуры к микроконтроллерам?
3. Какие пассивные элементы используются при подключении матричной клавиатуры к микроконтроллерам и для чего?
4. Опишите особенности используемых библиотек MatrixKeys и MsTimer2
5. Какой максимальный размер матричной клавиатуры можно реализовать при использовании библиотек MatrixKeys и MsTimer2
6. Поясните программный код для матричной клавиатуры.

Лабораторная работа № 8.
Подключение аналоговых термодатчиков к микроконтроллеру. Рабочий проект термометра

Цель работы: приобрести практические навыки по подключению аналоговых термодатчиков к микроконтроллеру.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

Интегральные датчики температуры с аналоговым выходом по напряжению.

Температурные датчики предназначены для измерения температуры объекта или вещества с помощью свойств и характеристик измеряемой среды. Все датчики работают по-разному. По принципу измерения эти устройства можно разделить на несколько групп:

– Термопары;
– Термисторы;
– Пьезоэлектрические датчики;

– Полупроводниковые датчики;

– Цифровые датчики;

– Аналоговые датчики.

По области применения можно выделить датчики температуры воздуха, жидкости и другие. Они могут быть как наружные, так и внутренние.

Любой температурный датчик можно описать набором характеристик и параметров, которые позволяют сравнивать их между собой и выбирать подходящий под конкретную задачу вариант.

Основными характеристиками являются:

– Функция преобразования, т.е. зависимость выходной величины от измеряемого значения. Для датчиков температуры этот параметр измеряется в Ом/С или мВ/К.

– Диапазон измеряемых температур.

– Метрологические параметры – к ним относятся различные виды погрешностей.

– Срок службы.

– Время отклика.

– Надежность – рассматриваются механическая устойчивость и метрологическая стойкость.

– Эксплуатационные параметры – габариты, масса, потребляемая мощность, стойкость к агрессивному воздействию среды, стойкость к перегрузкам и другие.

– Линейность выходных значений.

При всем разнообразии этих устройств, им присущи следующие общие качества:

– напряжение на выходе линейно пропорционально температуре;

– датчики имеют калиброванный масштабный коэффициент зависимости выходного напряжения от температуры, дополнительной калибровки не требуют.

Попросту говоря для измерения температуры с помощью датчиков такого типа необходимо измерить напряжение на выходе и через масштабный коэффициент пересчитать его в температуру.

Существует множество термодатчиков, которые можно отнести к этой категории. Следует выделить следующие типы датчиков температуры: ***LM35; TMP35; TMP36; TMP37***. Это самые распространенные, достаточно точные, недорогие устройства.

Подключение термодатчиков к микроконтроллеру.

Удобнее всего использовать датчики в корпусе TO-92.

Рис. 8.1 Датчик TMP36 в корпусе TO-92

Схема подключения для датчиков TMP36 выглядит так:

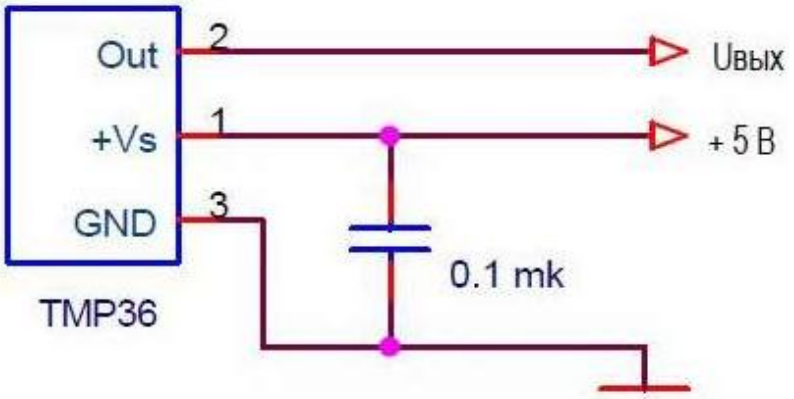

Рис. 8.2. Схема подключения датчика TMP36 *в корпусе TO-92*

По этой схеме будут работать также датчики LM35, TMP35, TMP37.

Основные параметры, различия датчиков.

Принципиальные отличия перечисленных датчиков друг от друга заключается в том, что они имеют различные диапазоны измерения температуры. Так TMP36 – единственный из перечисленных термодатчиков, способен измерять отрицательную температуру.

Речь идет о термодатчиках, подключенных по приведенной выше схеме. Например, есть схема включения LM35, позволяющая измерять отрицательные температуры. Но она сложнее в реализации и требует дополнительного питания. Лучше для отрицательных температур использовать TMP36.

Основные параметры термодатчиков LM35, TMP35, TMP36, TMP37 для этой схемы сведены в таблицу.

Таблица 8.1. Основные параметры термодатчиков LM35, TMP35, TMP36, TMP37

Тип	Диапазон измерения температуры, °C	Смещение напряжения на выходе, мВ	Масштабный коэффициент, мВ/°C	Напряжение на выходе при +25 °C, мВ
LM35, LM35A	0 … + 150	0	10	250
LM35C, LM35CA	0 … + 110	0	10	250
LM35D	0 … + 100	0	10	250
TMP35	+ 10 … + 125	0	10	250
TMP36	- 40 … + 125	500	10	750
TMP37	+ 5 … + 100	0	20	500

У всех термодатчиков напряжение на выходе может быть только положительным, но за счет смещения TMP36 способен измерять отрицательную температуру. Нулевое напряжение на его выходе соответствует температуре – 40°C, а при выходном напряжении 0,5 В температура будет равна 0°C. TMP36 являются самыми удобными из аналоговых интегральных датчиков температуры и используются достаточно широко.

Схема термометра на базе платы Arduino UNO R3 выглядит так:

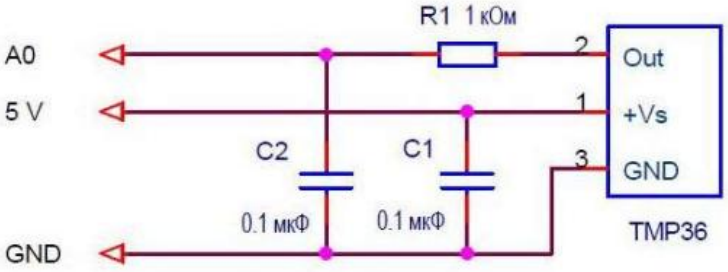

Рис. 8.3. Схема термометра на базе платы Arduino UNO R3

Датчик температуры подключен к аналоговому входу A0. Конденсатор C1 – сглаживающий питание датчика, R1 и C2 – простейший аналоговый фильтр. Если термодатчик установлен вблизи микроконтроллера, то фильтр можно исключить из схемы.

TMP35, TMP36, TMP37 допускают работу на нагрузку емкостью до 10 нФ, а LM35 – не более 50 пкФ. Поэтому, если датчик соединяется с контроллером длинной линией имеющей значительную емкость, то резистор R1 должен быть установлен со стороны датчика, а конденсатор C2 – со стороны контроллера. Сглаживающий конденсатор C1 всегда устанавливается рядом с термодатчиком.

Вычисление температуры.

Принцип простой. Для вычисления температуры датчиков LM35, TMP35, TMP37 необходимо:

– Считать код АЦП.

– Вычислить напряжение на выходе датчика как

$U_{вых} = N * U_{ион} / 1024$,

где $U_{вых}$ – напряжение на выходе термодатчика; N – код АЦП; $U_{ион}$ – напряжение источника опорного напряжения (для нашей схемы 5 В); 1024 – максимальное число градаций АЦП (10 разрядов).

– Разделить напряжение на выходе датчика на масштабный коэффициент.

Для датчика TMP36 необходимо перед делением на масштабный коэффициент вычесть напряжение смещения (0,5 В).

Формулы вычисления температуры для разных датчиков при опорном напряжении 5В выглядят так:

Таблица 8.2. Формулы вычисления температуры для разных датчиков при опорном напряжении 5В

Тип датчика	Формула вычисления температуры T (°C), при опорном напряжении 5 В, из кода АЦП – N.
LM35, TMP35	T = (N * 5 / 1024) / 0.01
TMP36	T = (N * 5 / 1024 – 0.5) / 0.01
TMP37	T = (N * 5 / 1024) / 0.02

Программа Arduino термометра.

Программа должна выполнять следующие функции:

– считывать значения кодов АЦП;

– усреднять их (цифровая фильтрация) для повышения помехозащищенности;

– вычислять температуру из кода АЦП;

– передавать раз в секунду значение температуры на компьютер в символьном формате.

Пример программного кода 8.1:

```
void setup() {
   Serial.begin(9600);
}

void loop() {
  //считываем аналоговое значение с вывода платы A0
  float sensorValue = analogRead(A4);// объявим переменную с плавающей
// запятой
  sensorValue = sensorValue*5000;//преобразовываем считанный код
  sensorValue = sensorValue/1024; // в температуру
  sensorValue = sensorValue/10;
  delay(1000);
  // выводим на компьютер значение измеренной температуры
  Serial.println(sensorValue,2);


}
```

Загружаем, проверяем. Запускаем монитор последовательного порта и проверяем данные на компьютере.

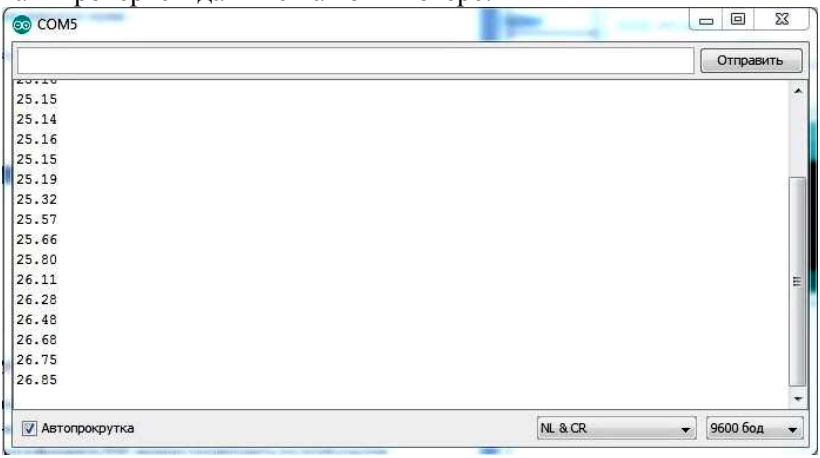

Рис. 8.4. Результаты выполнения программы

Разрешающая способность и точность термометра.

Разрешающая способность АЦП в нашей схеме 5В/1024=4,88мВ.

Разрешающая способность термометра:

– при масштабном коэффициенте 10 мВ/°С (датчики LM35, TMP35, TMP36) равна менее 0,5 °С;

– при масштабном коэффициенте 20 мВ/°С (датчик TMP37) равна менее 0,25 °С.

Вполне приличные параметры.

Что касается погрешности измерения – несколько хуже.

Погрешность измерения самих датчиков составляет:

– не более 0,5 °С для LM35;

– не более 1 °С для TMP35, TMP36, TMP37.

Погрешность измерения АЦП платы Arduino.

В нашем устройстве мы использовали опорное напряжение 5В, т.е. напряжение источника питания. В платах Arduino UNO R3 напряжение 5В формируется на линейном стабилизаторе NCP1117ST50. Стабильность выходного напряжения этой микросхемы довольно высокая – 1%.

Т.е. общая погрешность измерения термометра составляет не более 2%.

Программа верхнего уровня для термометра.

Смотреть на бегущие строки чисел в окне монитора Arduino IDE быстро надоедает. Для удобства пользователя была создана программа ***Thermometer.exe***, которая

– отображает текущее значение температуры;

– регистрирует изменение температуры с дискретностью 1 сек;

– выводит информацию об изменении температуры в графическом виде.

Программа работает под управлением операционных систем Windows 95, 98, XP, 7.

Подключение термометра к компьютеру.

Обмен данными между компьютером и контроллером осуществляется через COM-порт. Порт может быть реальным или виртуальным.

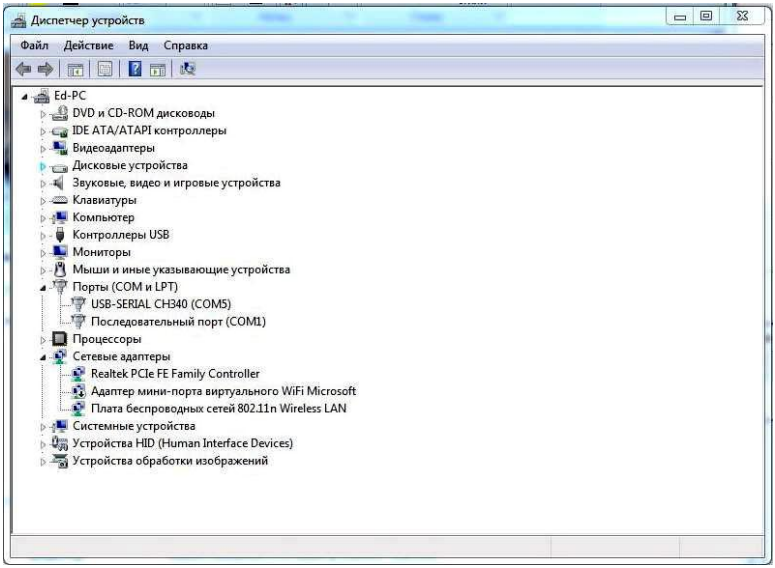

Рис. 8.5. Определение номеров COM-портов в диспетчере устройств

Удобнее всего использовать виртуальный порт, который создает драйвер платы Arduino. Порт появляется при подключении платы к компьютеру. Запускать Arduino IDE не надо. Номер порта можно посмотреть: Панель управления→Система→Диспетчер устройств→Порты (COM и LPT)

Если на компьютере есть штатный COM-порт (интерфейс RS232), то никаких драйверов устанавливать не надо. Для подключения контролера в этом случае необходимо использовать преобразователь уровней RS232 – TTL, микросхемы ADM232, SP232, MAX232 и им подобные.

Вариантов подключения много. Главное, чтобы на компьютере был сформирован COM-порт, виртуальный или реальный.

Первый запуск программы.

Перед запуском программы виртуальный COM-порт должен быть уже создан на компьютере. А так как порт создается при подключении к разъему платы Arduino, то это означает, что сначала необходимо подключить плату к компьютеру.

Дальше запускаете программу Thermometer.exe. В файле конфигурации программы записан какой-то COM-порт. Программа попытается открыть его при запуске. Если не получится, то она выдаст сообщение с номером ошибочного порта. Нажмите OK и откроется окно программы. Вместо температуры будут прочерки. Нет данных.

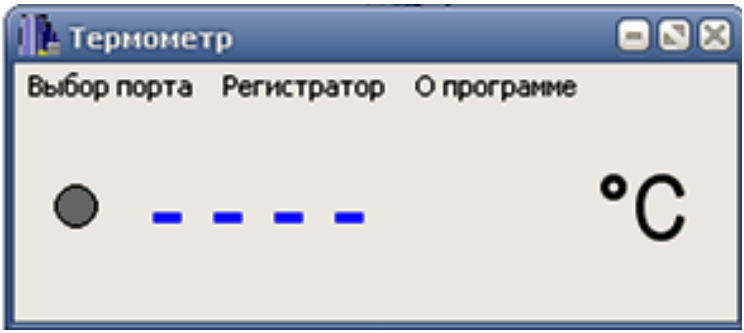

Рис. 8.6. Стартовое окно программы Thermometer.exe

Выберите в меню (сверху) режим выбор порта. Откроется окно выбора:

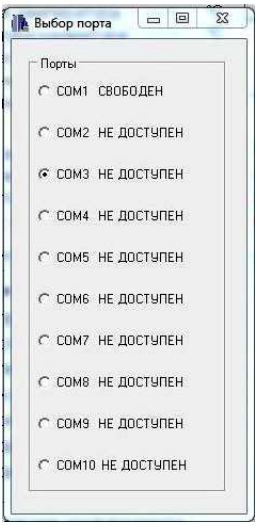

Рис. 8.7. Окно выбора порта

Задайте номер порта для платы. У каждого порта написано его состояние. Естественно, необходимо выбирать из портов с надписью «свободен».

Закройте окно. Выбранный COM-порт сохранится в файле конфигурации, и всегда будет вызываться при запуске программы. Задавать порт при каждом запуске программы не надо.

Если плата включена, программа загружена, все работает правильно, то раз в секунду должен мигать кружок-светодиод перед значением температуры. Он мигает при поступлении новых данных.

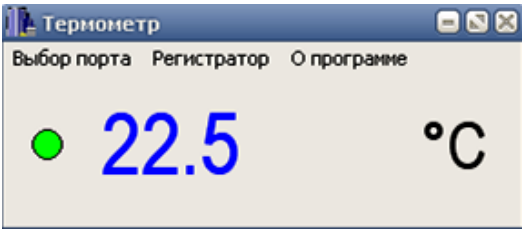

Рис. 8.8. Результат работы программы Thermometer.exe

В программе существует регистратор, который позволяет наблюдать динамику изменения температуры. Регистратор включается автоматически при запуске программы. Он записывает значения температуры с дискретностью по времени 1 секунда. Максимальное время регистрации 30 000 сек или 8,3 часов.

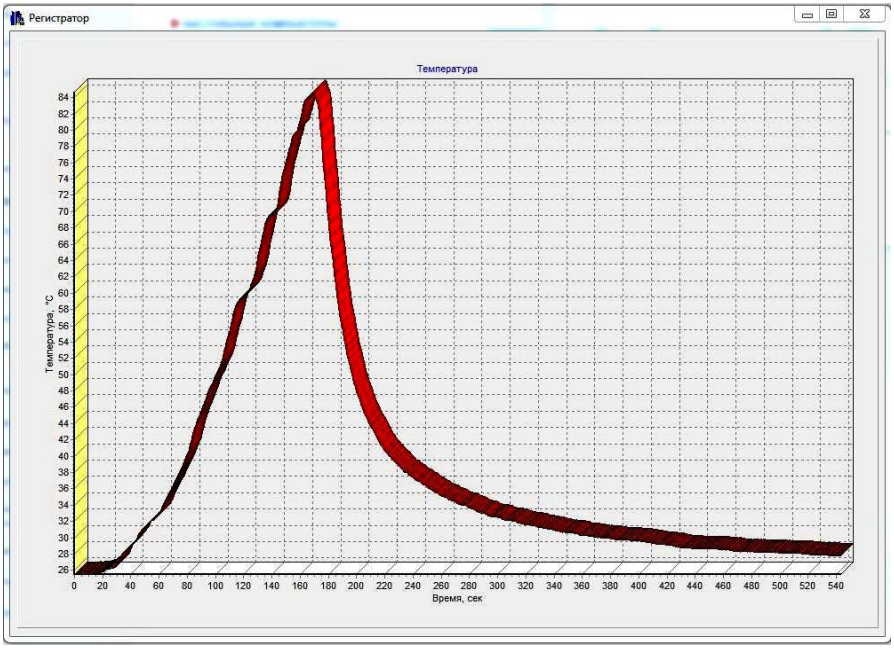

Рис. 8.9. Результаты записи регистратора

Чтобы посмотреть результаты записи необходимо нажать закладку меню ”Регистратор”.

Задания и порядок выполнения работы

Доработать программный код, предложенный в примере и обеспечить включение светодиода при достижении определенной температуры (например 28°С).

Контрольные вопросы

1. Как подключаются температурные датчики к микроконтроллеру?
2. Какой диапазон измерения у датчика LM35?
3. Поясните программный код датчика температуры.
4. Какая погрешность измерения температуры у рассмотренного температурного датчика?
5. По какому протоколу происходит обмен данными между компьютером и микропроцессором?

Лабораторная работа № 9.
Подключение цифрового термодатчика DH11 к микроконтроллеру. Рабочий проект термометра с семисегментной индикацией

Цель работы: приобрести практические навыки по подключению цифровых термодатчиков к микроконтроллеру используя вывод данных на семи сегментный индикатор.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

Термистор – это термический резистор, сопротивление которого изменяется с температурой, т.е. увеличение температуры приводит к падению его сопротивления. По сути, термистор – это термометр сопротивления, изготовленный на основе смешанных оксидов переходных металлов. Относится к измерительной технике и может быть использован для автоматического измерения температуры в различных средах.

Емкостной датчик влажности – это конденсатор с переменной емкостью, который содержит токопроводящие обкладки из медной фольги на текстолите. Этот конденсатор заключен в герме-

тичный чехол, поверх которого расположен влагопоглощающий слой. При попадании частиц воды на этот слой, меняется его диэлектрическая проницаемость, что приводит к изменению емкости конденсатора.

Датчики DHT11 и DHT22 не обладают высоким быстродействием и точностью, но зато просты, недороги и отлично подходят для обучения. Они выполнены из двух частей – емкостного датчика влажности и термистора. Чип, находящийся внутри, выполняет аналого-цифровое преобразование и выдает цифровой сигнал, который можно считать с помощью любого микроконтроллера.

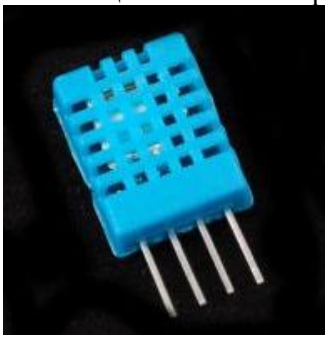

Рис. 9.1. Внешний вид сенсора DHT11

Спецификации:

– определение влажности в диапазоне 20-80%
– определение температуры от 0°C до +50°C
– частота опроса 1 раз в секунду

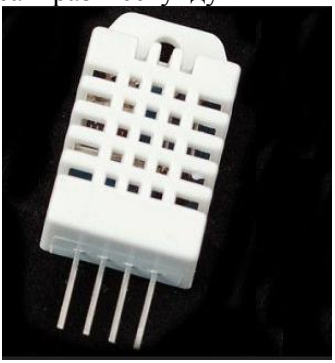

Рис. 9.2. Внешний вид сенсора DHT22

Спецификации:

– определение влажности в диапазоне 0-100%

– определение температуры от -40°C до +125°C

– частота опроса 1 раз в 2 секунды

Две версии сенсоров DHT похожи друг на друга и имеют одинаковые расположения и назначения выводов. Их отличия в характеристиках. Характеристики датчика DHT22 лучше по сравнению с DHT11, и поэтому он чуть-чуть дороже. Снимать показания чаще, чем раз в 1-2 секунды не получится, но, возможно, для вашего проекта более высокое быстродействие и не требуется.

Подключение сенсоров DHT к микроконтроллеру

Датчики DHT имеют стандартные выводы и их просто подключать.

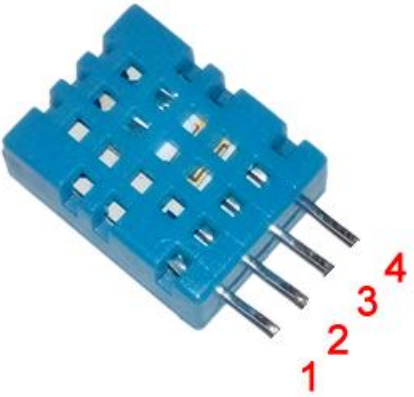

Рис. 9.3. Выводы датчика DHT

Датчики DHT имеют 4 вывода:

– **Vcc** – питание.

– **Data** – вывод данных

– **NC** – не используется.

– **GND** – земля.

Между выводами питания и вывода данных нужно разместить резистор номиналом 10 кОм.

Датчик DHT часто продается в виде готового модуля. В этом случае он имеет три вывода и подключается без резистора, т.к. резистор уже есть на плате.

Загрузите программу в контроллер и проверьте правильность работы при помощи Сервис→Монитор порта, не забудьте установить библиотеку «iarduino_DHT»

Пример программного кода 9.1:

```
#include <iarduino_DHT.h> // подключаем библиотеку для работы
//с датчиком DHT
iarduino_DHT sensor(14); // объявляем переменную для работы
//с датчиком DHT,
//указывая номер цифрового вывода к которому подключён
//датчик (сейчас 14pin(A0))
// Количество опрашиваемых датчиков ограничено количеством свободных выводов и самих датчиков.
// Библиотека iarduino_DHT.h сама определяет тип датчика
// (DHT11, DHT21, DHT22)
// Если подключено два датчика (например, один к выводу 2, а второй
// к выводу 5),
// то в начале скетча, после подключения библиотеки, нужно
//объявить две переменные,
// для работы с датчиками (по одной на каждый датчик).
// #include <iarduino_DHT.h>
// iarduino_DHT sensor1(2);
// iarduino_DHT sensor2(5);
// sensor1.read(); // чтение показаний первого датчика
// sensor1.hum // показания влажности первого датчика
// sensor1.tem // показания температуры первого датчика
// sensor2.read(); // чтение показаний второго датчика
// sensor2.hum // показания влажности второго датчика
// sensor2.tem // показания температуры второго датчика
void setup() {
Serial.begin(9600); // открываем последовательный порт на
// скорости 9600 бод
}
void loop() {
// Задержка 1 секунда между измерениями
delay(1000);
sensor.read(); // чтение показаний с датчика
//Считываем влажность
float h = sensor.hum; // показания влажности датчика
// Считываем температуру
float t = sensor.tem; // показания температуры датчика
// Проверка удачно прошло ли считывание.
```

```
if (isnan(h) || isnan(t)) {
Serial.println("ERROR"); //Не удается считать показания
return;
}
Serial.println((String)"Vlagnost:"+ h +" %\t"+"Temperatura: "+ t +" *C ");
}
```

Вы должны увидеть температуру и влажность. Изменения можно увидеть, например, выдыхая на датчик (как для затуманивания окна). Дыхание увеличивает влажность.

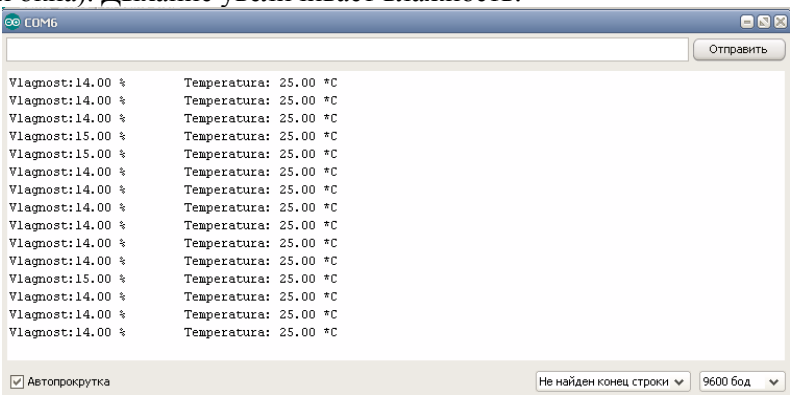

Рис. 9.4. Результат выполнения программного кода 9.1

Семисегментный индикатор

Это наиболее распространенный элемент, позволяющий наглядно отобразить цифровую информацию (а иногда и буквенную). Типовой светодиодный или жидкокристаллический элемент выглядит следующим образом.

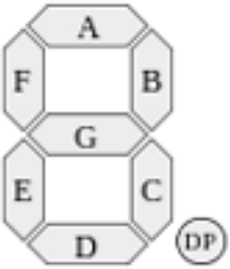

Рис. 9.5. Светодиодный или жидкокристаллический элемент

Семисегментный индикатор – это просто набор обычных светодиодов в одном корпусе. Просто они выложены восьмёркой и имеют форму палочки-сегмента.

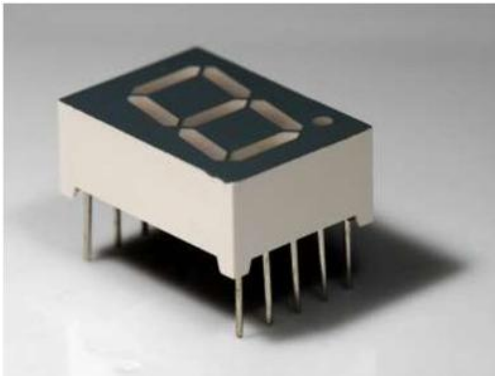

Рис. 9.6. Семисегментный индикатор

Светодиодов на этот раз 8 (семь полосок и один кружочек) и они расположены не друг за другом, а в определённом порядке, которые позволяют вам выводить цифры от 0 до 9.

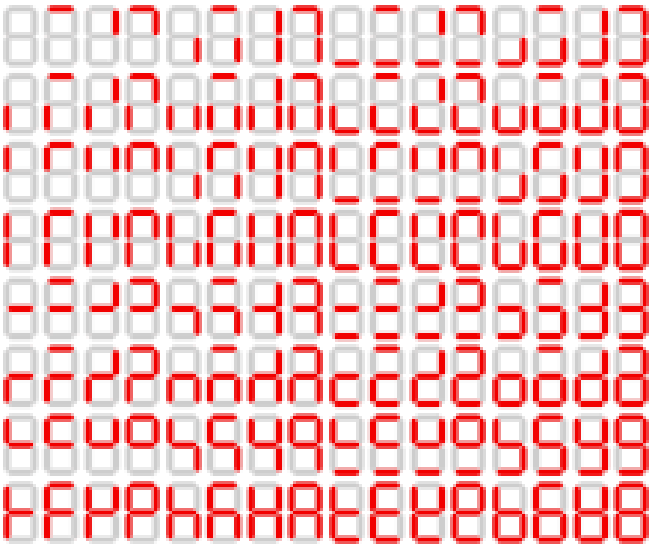

Рис. 9.7. Комбинации свечения

С помощью семи сегментов можно показать пользователю 128 (27) различных комбинаций свечения.

Важная отличительная черта – у индикатора имеются общие ножки для катода (ножки 3 и 8). Всего их две и они равноценны. Это удобно, вам не нужно будет от каждого катода вести отдельный провод на землю. Достаточно выбрать один из общих катодов и от неё соединиться с GND. Аноды у всех отдельные.

Также при желании вы можете установить несколько таких индикаторов подряд для вывода больших двухзначных, трёхзначных и т.д. чисел. Но существуют готовые компактные наборы для этих целей.

На семисегментный индикатор распространяются те же правила, что и на стандартные светодиоды – у каждого должен быть свой резистор.

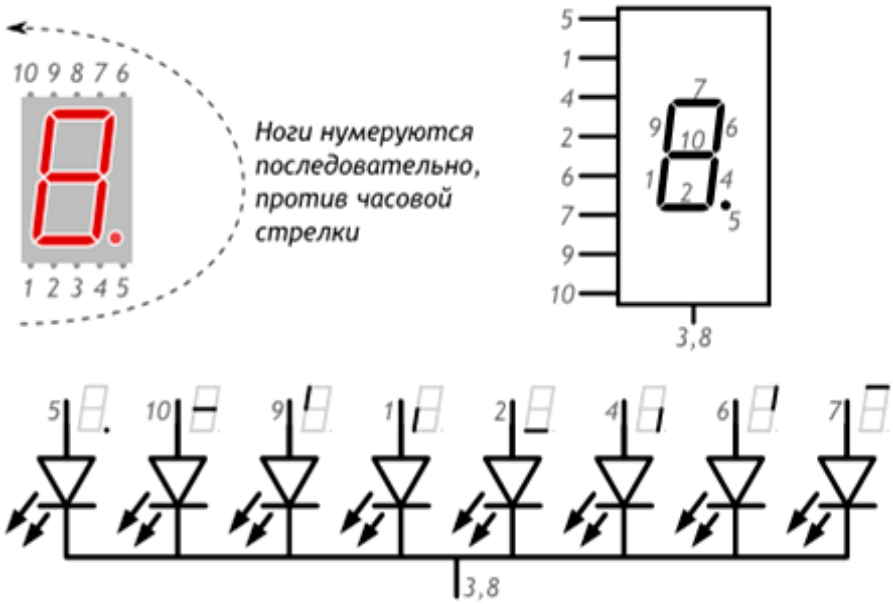

Рис. 9.8. Схематическое изображение семисегментного индикатора с нумерацией выводов.

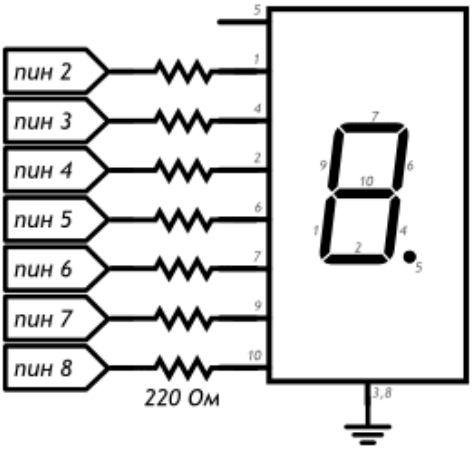

Рис. 9.9. Принципиальная схема подключения семисегментного индикатора к выводам микроконтроллера.

Матрица 4-разрядная из семисегментных индикаторов состоит из четырех семисегментных индикаторов и предназначена для одновременного вывода на матрицу 4 цифр, также есть возможность вывода десятичной точки.

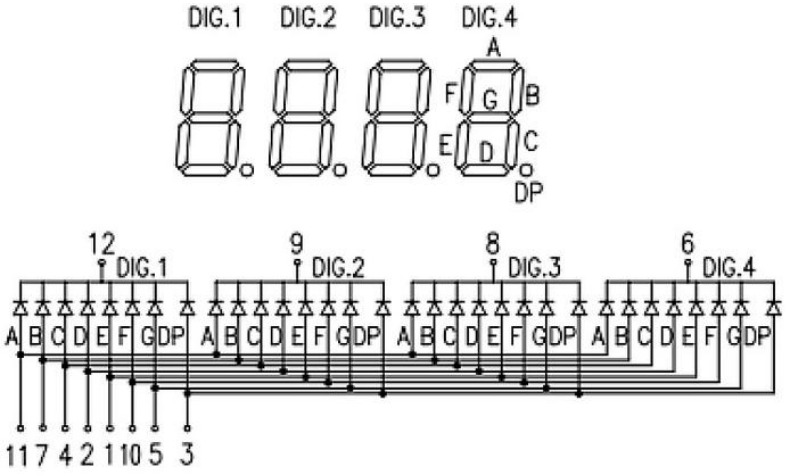

Рис. 9.10. Схема расположения выводов четырёхразрядного семисегментного индикатора.

Для вывода цифры необходимо зажечь нужные светодиоды на контактах A–G и DP и выбрать нужную матрицу подачей LOW на вывод 6, 8, 9 или 12.

Подключим контакты матрицы к плате Arduino и будем выводить цифры на различные разряды матрицы. Для подключения понадобятся 12 выводов Arduino. При подключении контактов используются ограничительные резисторы 510 Ом.

В примере, рассмотренном ниже, выведем на 3-х разрядную матрицу температуру с датчика DH11.

Для проверки программы воспользуемся отладочной платой на которой сигнальный вывод датчика DH11 подключен к выводу A0(14), катоды 3-х разрядной матрицы подключены к выводам платы Arduino (2,3,4), светодиоды на контактах A–G подключены к выводам платы Arduino (5,6,7,8,9,10,11,12), катоды 2-х разрядной матрицы подключены к выводам платы Arduino (13,19), сигнальные светодиоды подключены к выводам платы Arduino (A1(15),A2(16),A3(17),A4(18)).

Пример программного кода 9.2:

```
#include <iarduino_DHT.h> // подключаем библиотеку для работы
// с датчиком DHT
iarduino_DHT sensor(14); //  объявляем  переменную для работы
//с датчиком DHT,
//указывая номер цифрового вывода к которому подключён
//датчик (сейчас 14pin(A0))
// список выводов Arduino для подключения к разрядам a-g
// семисегментного индикатора
int pins[8]={5,6,7,8,9,10,11,12};
// значения для вывода цифр 0-9
byte numbers[10] = { B11111100, B01100000, B11011010,
B11110010, B01100110, B10110110,
B10111110, B11100000, B11111110,
B11110110};
//переменные для отсчета времени
unsigned long currentTime1;
unsigned long loopTime1;
// переменная для хранения и обработки текущего значения
int number=0;
int number1=0;
int number2=0;
// семисегментного индикатора
```

```
int pindigits[3]={4,3,2};
// переменная для хранения текущего разряда
int digit=0;
// для отмеривания 100 мс
unsigned long millis1=0;
void setup()
{

// Сконфигурировать контакты как выходы
for(int i=0;i<8;i++)
pinMode(pins[i],OUTPUT);
for(int i=0;i<3;i++)
{pinMode(pindigits[i],OUTPUT);
digitalWrite(pindigits[i],HIGH);
}
}
void loop()
{
  currentTime1 = millis();
  if(currentTime1 >= (loopTime1 + 2000)){
 switch(sensor.read()){    // читаем показания датчика
    case DHT_OK:              Serial.println((String)
    "СЕНСОР В КОМНАТЕ: " + sensor.hum + "% – " + sensor.tem +
"*C");  break;
    case DHT_ERROR_CHECKSUM:   Serial.println(
    "СЕНСОР В КОМНАТЕ: НЕ РАВЕНСТВО КС");                  break;
    case DHT_ERROR_DATA:       Serial.println(
    "СЕНСОР В КОМНАТЕ: ОТВЕТ НЕ СООТВЕТСТВ. СЕНСОРАМ
'DHT'"); break;
    case DHT_ERROR_NO_REPLY:   Serial.println(
    "СЕНСОР В КОМНАТЕ: НЕТ ОТВЕТА");                       break;
    default:                Serial.println(
    "СЕНСОР В КОМНАТЕ: ERROR");                        break;
  }
  loopTime1 = currentTime1;
 }
number=sensor.tem*10; //записываем текущее значение температуры
// в переменную number
number1=number;
for(int i=0;i<3;i++)
{
number2=number1%10;
number1=number1/10;
showNumber(number2,i);
digitalWrite(pindigits[i],HIGH);
```

```
delay(1);
for(int j=0;j<3;j++){
digitalWrite(pindigits[j],LOW);
}
}
}
// функция вывода цифры на семисегментный индикатор
void showNumber(int num,int dig)
{
for(int i=0;i<8;i++)
{
if(bitRead(numbers[num],7-i)==HIGH) // зажечь сегмент
digitalWrite(pins[i],HIGH);
else // потушить сегмент
digitalWrite(pins[i],LOW);
}
if(dig==1) // десятичная точка для второго разряда
digitalWrite(pins[7],HIGH);
}
```

Задания и порядок выполнения работы

1. Доработать программный код предложенный в примере 9.1 и обеспечить включение светодиодов, при достижении определенной температуры и влажности (например 28°С и 50% влажности) светодиоды подключены к выводам А4 и А5 платы Arduino и при достижении определенной влажности (например 40%) светодиоды подключены к выводам А2 и А3 платы.

2. Доработать программный код предложенный в примере 9.2 и обеспечить отображение на семисегментных индикаторах отображение температуры и влажности, а также включение светодиодов при достижении определенной температуры и влажности (например 28°С и 50% влажности) светодиоды подключены к выводам А4 и А5 платы Arduino и при достижении определенной влажности (например 40%) светодиоды подключены к выводам А2 и А3 платы.

Контрольные вопросы

1. Как подключается датчик DH11 к микроконтроллеру?
2. Какой диапазон измерения у датчика DH11?
3. Поясните программный код датчика температуры и влажности.
4. Как преобразовать полученный код в метеостанцию?
5. Какая погрешность измерения температуры и влажности у рассмотренного датчика DH11?
6. По какому протоколу происходит обмен данными между датчиком DH11 и микропроцессором?
7. Какие особенности отображения информации при использовании семи сегментных индикаторов?

Лабораторная работа № 10.
Управление портами в AVR. Регистры DDRX И PORTX. Представление чисел. Побитные операции. Функции задержки. Безусловный переход в программе

Цель работы: приобрести практические навыки по управлению регистрами DDRX И PORTX микроконтроллера и использовать их для вывода данных на семи сегментный индикатор.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

Порты микроконтроллера – это устройства ввода/вывода, позволяющие микроконтроллеру передавать или принимать данные. Стандартный порт микроконтроллера AVR имеет восемь разрядов данных, которые могут передаваться или приниматься параллельно. Каждому разряду (или биту) соответствует вывод (ножка) микроконтроллера. Ножки микроконтроллера также называют *пинами*. Для обозначения портов используются латинские буквы A, B, C и т.д. Количество портов ввода/вывода варьируется в зависимости от модели микроконтроллера.

Любой порт микроконтроллера можно сконфигурировать как вход или как выход. Для того чтобы это сделать, следует записать в соответствующий порту регистр **DDRx** необходимое значение. Кроме того, как вход или выход можно сконфигурировать отдельно любой вывод (пин) порта. В любом случае, хотите вы сконфигурировать весь порт или отдельный вывод, вам необходимо будет работать с регистрами DDRx**.**

DDRx – регистр направления передачи данных. Этот регистр определяет, является тот или иной вывод порта входом или выходом. Если некоторый разряд регистра DDRx содержит логическую единицу, то соответствующий вывод порта сконфигурирован как выход, в противном случае – как вход. Буква x в данном случае должна обозначать имя порта, с которым вы работаете. Таким образом, для порта A это будет регистр DDRA, для порта B – регистр **DDRB** и т. д.

Используя AVR GCC, записать в необходимый регистр то или иное значение можно одним из следующих способов.

Для всего порта сразу.

DDRD = 0xff;

Все выводы порта D будут сконфигурированы как выходы.

0xff – шестнадцатеричное представление числа ff, где 0x является префиксом, используемым для записи шестнадцатеричных чисел. В десятичном представлении это будет число 255, а в двоичном виде оно будет выглядеть как 11111111. То есть во всех битах регистра **DDRD** будут записаны логические единицы.

В AVR GCC для представления двоичных чисел используется префикс 0b. Таким образом, число 11111111 должно представляться в программе как 0b11111111. Мы можем записать предыдущую команду в более читабельном виде.

DDRD = 0b11111111;

Хотя такая запись и выглядит более наглядной, при конфигурировании портов принято использовать шестнадцатеричное представление чисел.

Для того чтобы сконфигурировать все выводы порта D как входы, следует записать во все биты регистра DDRD логические нули.

DDRD = 0x00;

В регистр DDRD можно записать и другие числа. Например:

DDRD = 0xb3;

0xb3 – шестнадцатеричное представление числа 179. В двоичном виде оно будет выглядеть как 10110011. То есть часть выводов порта D будет сконфигурирована как выходы, а часть – как входы.

PD0 – 1 (выход)
PD1 – 1 (выход)
PD2 – 0 (вход)
PD3 – 0 (вход)
PD4 – 1 (выход)
PD5 – 1 (выход)
PD6 – 0 (вход)
PD7 – 1 (выход)

Каждый бит регистров **DDRx** может быть установлен отдельно. Например, чтобы сконфигурировать отдельно вывод PD2 как выход, нам необходимо в соответствующий бит регистра **DDRD** записать 1. Для этого применяют следующую конструкцию.

DDRD |= 1<<2;

1<<2 – осуществляет сдвиг единички влево на 2 бита, то есть справа добавляются два нулевых бита и получается 100, а знак «|», стоящий перед знаком присваивания «=», осуществляет операцию побитного логического сложения.

При логическом сложении 0+0=0, 0+1=1, 1+1=1. Операцию логического сложения по-другому называют операцией ИЛИ (английское название OR).

Таким образом, к битам, хранящимся в регистре **DDRD**, прибавляется двоичное 100, представленное в 8-битном регистре микроконтроллера как 00000100, и результат записывается обратно в регистр **DDRD**.

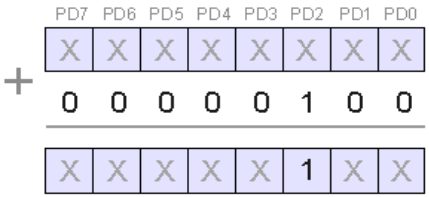

Рис. 10.1. Регистр DDRD. Вывод PD2 как выход.

Чтобы сконфигурировать отдельно вывод PD2 как вход, нам необходимо в соответствующий бит регистра DDRD записать 0. Для этого применяют следующую конструкцию.

DDRD &= ~(1<<2);

В данном случае результат сдвига единицы на две позиции влево инвертируется с помощью операции побитного инвертирования, обозначаемой значком «**~**».

При инверсии мы получаем вместо нулей единички, а вместо единичек – нули. Эта логическая операция иначе называется операцией **НЕ** (английское название NOT).

Таким образом, при побитном инвертировании 00000100 мы получаем 11111011.

Получившееся число с помощью операции побитного логического умножения **&** умножается на число, хранящееся в регистре **DDRD**, и результат записывается в регистр **DDRD.**

При логическом умножении 0*0=0, 0*1=0, 1*1=1. Операцию логического умножения иначе называют операцией **И** (английское название AND).

То есть сдвинутая нами влево на две позиции единичка превращается при инвертировании в ноль и умножается на соответствующий бит, хранящийся в регистре **DDRD**. При умножении на ноль мы получаем ноль. Таким образом, бит **PD2** становится равным нулю.

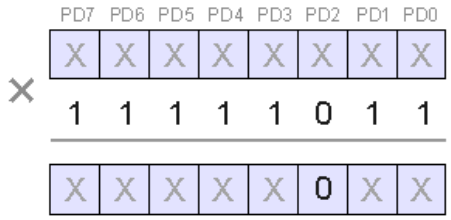

Рис. 10.2. Регистр DDRD. Вывод PD2 как вход.

Кроме логических операций *И, ИЛИ, НЕ* существует также операция «*исключающее ИЛИ*» (английское название XOR). Она обозначается значком ^.

При *исключающем ИЛИ* значение бита, к которому «прибавляется» единичка, изменяется на противоположное.

Например, 110011 ^ 11010 = 101001

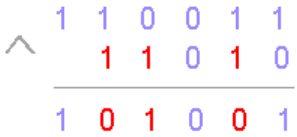

Рис. 10.3. Пример исключающего ИЛИ

Следует добавить, что работа с числами в 8-битном микроконтроллере проходит с использованием 8-битных регистров. Перед вычислениями аргумент помещается в один из специальных регистров, с которыми напрямую может работать арифметико-логическое устройство (АЛУ). Например, перед выполнением команды **DDRD &= ~(1<<2)** аргумент помещается во вспомогательный регистр микроконтроллера. Содержимое такого регистра будет выглядеть как **11111011**.

После этого осуществляется операция побитного умножения, что дает во втором бите регистра DDRD значение 0.

После того как направление передачи данных у порта сконфигурировано, можно присвоить порту значение, которое будет храниться в соответствующем регистре **PORTx.**

PORTx – регистр порта, где x обозначает имя порта.

Если вывод сконфигурирован как выход, то единичка в соответствующем бите регистра **PORTx** формирует на выводе сигнал высокого уровня, а ноль – сигнал низкого уровня.

Если же вывод сконфигурирован как вход, то единичка в соответствующем бите регистра **PORTx** подключает к выводу внутренний подтягивающий pull-up резистор, который обеспечивает высокий уровень на входе при отсутствии внешнего сигнала.

Установить «1» на всех выводах порта D можно следующим образом:

PORTD = 0xff;

А установить «0» на всех выводах порта D можно так:

PORTD = 0x00;

К каждому биту регистров **PORTx** можно обращаться и по отдельности так же, как в случае с регистрами **DDRx.**

Например, команда

PORTD |= 1<<3;

установит «1» (сигнал высокого уровня) на выводе **PD3**.

Команда

PORTD &= ~(1<<4);

установит «0» (сигнал низкого уровня) на выводе **PD4**.

В AVR GCC сдвиг можно осуществлять и с помощью функции **_BV(),** которая выполняет поразрядный сдвиг и вставляет результат в компилируемый код.

В случае использования функции **_BV()** две предыдущие команды будут выглядеть следующим образом.

PORTD |= _BV(PD3); // установить «1» на линии 3 порта D

PORTD &= ~_BV(PD4); // установить «0» на линии 4 порта D

В микроконтроллерах AVR каждому параллельному порту ввода/вывода поставлен в соответствие также регистр **PINx.**

PINx является регистром выводов порта и в отличие от регистров **DDRx** и **PORTx** доступен только для чтения. **PINx** позволяет считывать входные данные порта на внутреннюю шину микроконтроллера.

Пример программного кода 10.1:Отображение температуры на семисегментном индикаторе с использованием регистра PORTx

```
#include <iarduino_DHT.h>   // подключаем библиотеку для работы
// с датчиком DHT
iarduino_DHT sensor(14); // объявляем  переменную для работы
//с датчиком DHT,
//указывая номер цифрового вывода к которому подключён
//датчик (сейчас 14pin)
int b0, b1, b2=0;
unsigned long currentTime1;
unsigned long loopTime1;
byte toScreen_D[]={
  0b11100000,
  0b11000000,
  0b01100000,
  0b11100000,
  0b11000000,
  0b10100000,
  0b10100000,
  0b11100000,
  0b11100000,
  0b11100000,
  0b00000000,
```

```
  0b00000000
};
byte toScreen_B[]={
  0b00111,
  0b00000,
  0b01011,
  0b01001,
  0b01100,
  0b01101,
  0b01111,
  0b00000,
  0b01111,
  0b01101,
  0b10000,
  0b00000
};
byte toScreen_H[]={
  0b10111,
  0b10000,
  0b11011,
  0b11001,
  0b11100,
  0b11101,
  0b11111,
  0b10000,
  0b11111,
  0b11101,
  0b10000,
  0b10000
};
void setup(){
  Serial.begin(9600);       // открываем последовательный порт на
// скорости 9600 бод
  //delay(1000);

 pinMode(2, OUTPUT);
 pinMode(3, OUTPUT);
 pinMode(4, OUTPUT);
 pinMode(5, OUTPUT);
 pinMode(6, OUTPUT);
 pinMode(7, OUTPUT);
 pinMode(8, OUTPUT);
 pinMode(9, OUTPUT);
 pinMode(10, OUTPUT);
 pinMode(11, OUTPUT);
```

```
 pinMode(12, OUTPUT);
 pinMode(13, OUTPUT);
 pinMode(15, OUTPUT);
 pinMode(16, OUTPUT);
 pinMode(17, OUTPUT);
 pinMode(18, OUTPUT);
 pinMode(19, OUTPUT);
 currentTime1 = millis();      // считываем время, прошедшее с момента
// запуска программы
  loopTime1 = currentTime1;
 sensor.read();
}

void loop(){
  currentTime1 = millis();
  if(currentTime1 >= (loopTime1 + 2000)){
 switch(sensor.read()){    // читаем показания датчика
    case DHT_OK:              Serial.println((String)
    "СЕНСОР В КОМНАТЕ: " + sensor.hum + "% – " + sensor.tem +
"*C");  break;
    case DHT_ERROR_CHECKSUM:   Serial.println(
    "СЕНСОР В КОМНАТЕ: НЕ РАВЕНСТВО КС");                  break;
    case DHT_ERROR_DATA:      Serial.println(
    "СЕНСОР В КОМНАТЕ: ОТВЕТ НЕ СООТВЕТСТВ. СЕНСОРАМ
'DHT'"); break;
    case DHT_ERROR_NO_REPLY:   Serial.println(
    "СЕНСОР В КОМНАТЕ: НЕТ ОТВЕТА");                     break;
    default:                  Serial.println(
    "СЕНСОР В КОМНАТЕ: ERROR");                        break;
  }
  loopTime1 = currentTime1;
 }

//  Количество опрашиваемых датчиков ограничено количеством
// свободных выводов и самих датчиков.
//  Библиотека iarduino_DHT.h сама определяет тип датчика
// (DHT11, DHT21, DHT22)
//  Если подключено два датчика (например, один к выводу 2, а второй к
выводу 5), то в начале скетча, после подключения библиотеки,
//  нужно объявить две переменные, для работы с датчиками
//  (по одной на каждый датчик).
//  #include <iarduino_DHT.h>
//  iarduino_DHT    sensor1(2);
//  iarduino_DHT    sensor2(5);
//
```

```
//  sensor1.read(); // чтение    показаний   первого датчика
//  sensor1.hum     // показания влажности   первого датчика
//  sensor1.tem     // показания температуры первого датчика
//  sensor2.read(); // чтение    показаний   второго датчика
//  sensor2.hum     // показания влажности   второго датчика
//  sensor2.tem     // показания температуры второго датчика


float t = sensor.tem;
  int mainPartt = (int)t,
  fractial2=0;
  t-=mainPartt;//0.xx
  t*=100; //xx
  fractial2 = (int)t;
  b2=fractial2/=10;
  b1=mainPartt%10;
  mainPartt = mainPartt-b1;
  b0=mainPartt/10;
 Serial.println((String) "term    " + b0 + " " + b1 + "*T");


  for (int i =1; i<=3; i++){
  switch (i) {
    case 1:
     PORTD = PORTD | toScreen_D[b0],BIN;
     PORTB = PORTB | toScreen_B[b0],BIN;
     digitalWrite(2, HIGH);
      delay(5);
        PORTD =toScreen_D[11],BIN;
        PORTB = toScreen_B[11],BIN;
   //выполняется когда  var равно 1
      break;
      case 2:
    PORTD = PORTD | toScreen_D[b1],BIN;
     PORTB = PORTB | toScreen_H[b1],BIN;
     digitalWrite(3, HIGH);
     delay(5);
      PORTD =toScreen_D[11],BIN;
        PORTB = toScreen_B[11],BIN;
   //выполняется когда  var равно 2
      break;
       case 3:
       PORTD = PORTD | toScreen_D[b2],BIN;
     PORTB = PORTB | toScreen_B[b2],BIN;
      digitalWrite(4, HIGH);
```

```
    delay(5);
   PORTD =toScreen_D[11],BIN;
   PORTB = toScreen_B[11],BIN;
//выполняется когда  var равно 3
    break;
  }
}
}
```

Задания и порядок выполнения работы

Доработать программный код предложенный в примере и обеспечить отображение на семисегментных индикаторах отображение температуры и влажности и включение светодиодов, при достижении определенной температуры и влажности (например 28°С и 50% влажности) светодиоды подключены к выводам А4 и А3 платы Arduino и при достижении определенной влажности (например 40%) светодиоды подключены к выводам А1 и А2 платы. Для активации 2-х разрядного семисегментного индикатора используйте выводы 13 и 19 платы Arduino.

Контрольные вопросы

1. Что подразумевается под понятием «порты микроконтроллера»?
2. Что такое регистр порта?
3. Какие регистры портов вы знаете?
4. Что необходимо для того чтобы сконфигурировать все выводы порта?
5. Что произойдет вследствие выполнения команды PORTD=0xff;
6. Что произойдет вследствие выполнения команды PORTD=0x00;
7. Что произойдет вследствие выполнения команды PORTD|=1<<3;
8. Что произойдет вследствие выполнения команды PORTD&=~(1<<4);
9. Поясните программный код приведенный в примерах.

Лабораторная работа № 11.
Управление семисегментным дисплеем по SPI

Цель работы: приобрести практические навыки по управлению семисегментным дисплеем по SPI-порту микроконтроллера и использовать их для вывода данных на семисегментный индикатор.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

Шина SPI

SPI (Serial Peripheral Interface) – это последовательный синхронный стандарт передачи данных, который предназначен для общения контроллера с периферией. Большинство современных микроконтроллеров поддерживают SPI, поскольку она очень проста и удобна в использовании. У Arduino есть библиотека для работы с SPI, которая состоит всего из шести простых функций.

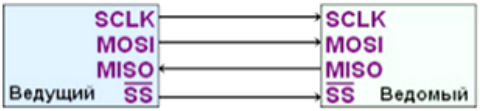

Рис. 11.1. Шина SPI

Шина SPI состоит из 4х проводов: MOSI, MISO, CS и SCLK:

– ***MOSI (Master Out Slave In)*** или просто **SI** – передача данных от ведущего к ведомому.

– ***MISO (Master In Slave Out)*** или просто **SO** – передача данных от ведомого к ведущему.

– ***CS (Chip Select)*** или ***SS (Slave Select)*** – выбор ведомого устройства.

– ***SCLK (Serial CLocK)*** или просто ***SCK*** – передача тактового сигнала от ведущего к ведомому.

Передача данных от ведущего производится так: ведущий выставляет низкий уровень на проводе CS ведомого устройства, с которым он собирается общаться, после чего по MOSI передаются биты. По окончании передачи ведущий «отпускает» провод CS – возвращает его на высокий уровень.

Чтобы подключить несколько устройств к шине SPI, для каждого устройства заводится свой провод CS, и ведущий, поочередная «дергая» ими, общается с тем или иным устройством. Подробнее подключение нескольких устройств мы рассмотрим в следующей статье, а пока двинемся дальше.

Сдвиговый регистр 74PC595

Мы будем использовать в качестве ведомого устройства сдвиговый регистр 74HC595, к которому подключим 7-сегментный дисплей. Полное его название – «8-bit serial-in, serial or parallel-out shift register with output latches» – «восьмибитный регистр с последовательным вводом и последовательным или параллельным выводом и с выходными защелками».

Принцип действия такого регистра заключается в том, что он последовательно получает биты, по проводу DS, а при выставлении низкого уровня на проводе STCP «защелкивает» полученные биты так, что на его параллельных выводах аккуратно формируется весь полученный байт. Выход Q7S «повторяет» биты на входе DS, что делает возможным работу регистра как бы в режиме bypass, что пригождается в случае последовательного соединения нескольких регистров (это мы будем использовать в следующий раз).

Одно из устройств на шине является ведущим (чаще всего, сам контроллер), а остальные – ведомыми. Данные передаются по

шине от ведущего к одному выбранному ведомому или от ведомого к ведущему синхронно по тактирующему сигналу ведущего.

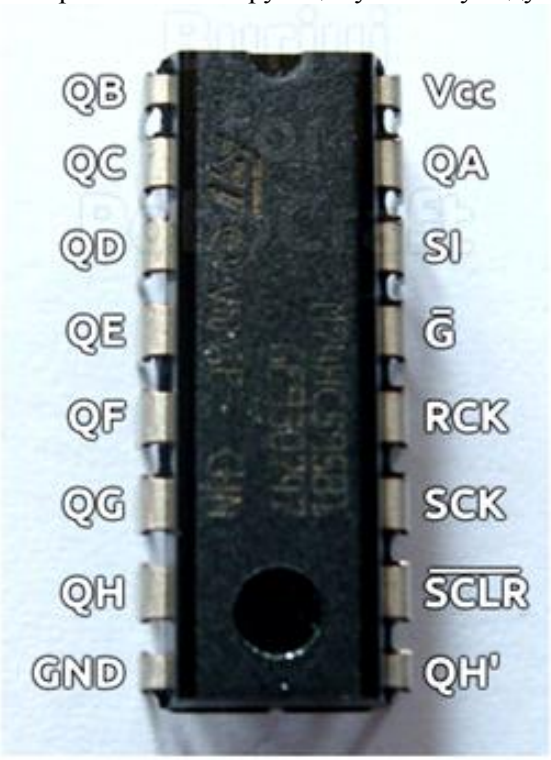

Рис 11.2. Сдвиговый регистр 74HC595

Здесь:

– **Vcc –** питание, от 2 до 6 В;

– **GND** – земля;

– **QA-QH** – эти выводы соответствуют битам, записанными по SPI;

– **SI** – вход ведомого, MOSI (SPI);

– **G** – Output Enabe; когда на этом выводе низкий уровень, выводы включены (подключены к «защёлкам»), когда высокий – выводы переходят в состояние Hi-Z;

– **RCK** – защёлка, SS (SPI); при установке низкого уровня выводы регистра защёлкиваются;

– **SCK** – тактовый вход, SCLK (SPI);

– **SCLR** – Shift Register Clear Input; если на этом выводе низкий уровень, очищает все триггеры по фронту тактового сигнала на SCLK. С нашей точки зрения это банальный RESET: прижал к земле – сбросил все биты регистра;

– **QH'** – на этом выводе будет появляться старший переданный бит.

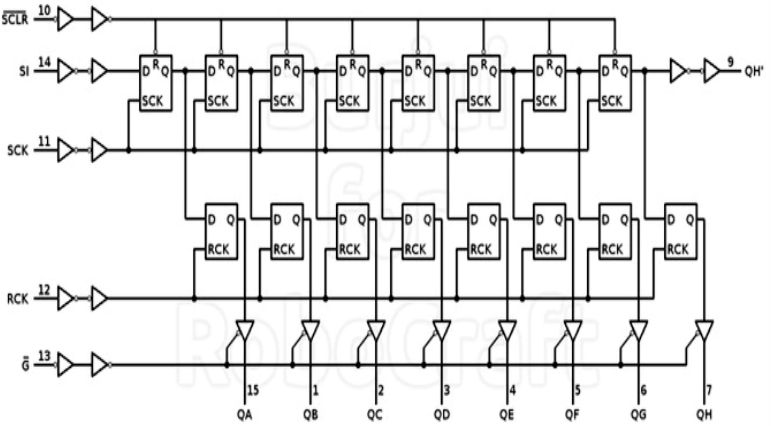

Рис. 11.3. Схема сдвигового регистра 74HC595

Подключим вывод микросхемы 747HC595 №11 (SH_CP) – тактовая шина (clock) к 10 контакту Arduino (не принципиально), 12 (ST_CP) – защёлка (latch) к 9 контакту Arduino (не принципиально), 14 (DS) – данные (data) к 8 контакту Arduino (не принципиально) и вгрузим пример. Разряды (0-3) семисегментного индикатора подключены к выводам 2,2,4,5 платы Arduino.

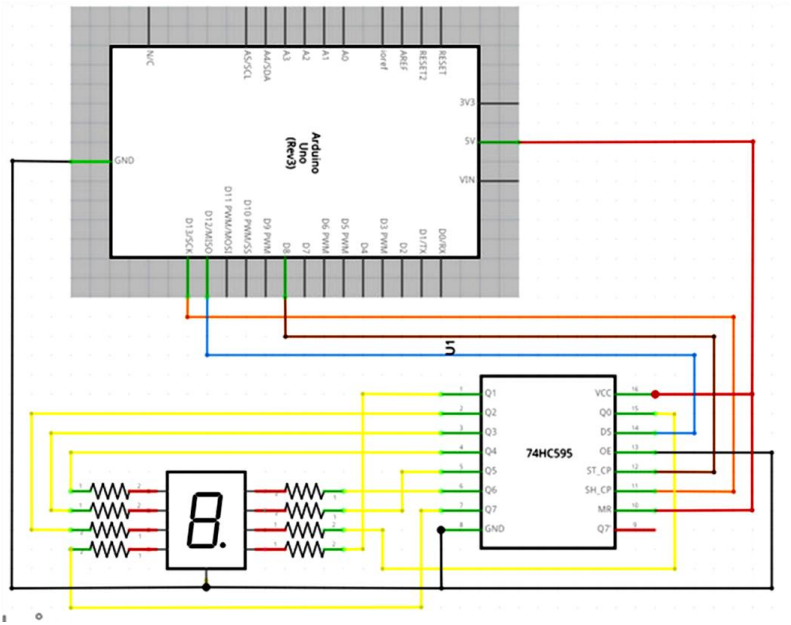

Рис. 11.4. Схема подключения сдвигового регистра 74HC595 и семисегментного индикатора к плате Arduino

Преимущества использования сдвигового регистра 74HC595:

– не требует никакой обвязки кроме конденсатора по питанию;

– работает через широко распространенный интерфейс SPI;

– для самого простого включения достаточно двух выходов микроконтроллера;

– возможность практически неограниченного расширения количества выходов без увеличения занятых выходов микроконтроллера;

– частота работы до 100 МГц;

– напряжение питания от 2 В до 6 В;

– дешевый;

– выпускается как в планарных корпусах (74HC595D удобен для производства), так и в DIP16 (74HC595N удобен для радиолюбителей и макетирования).

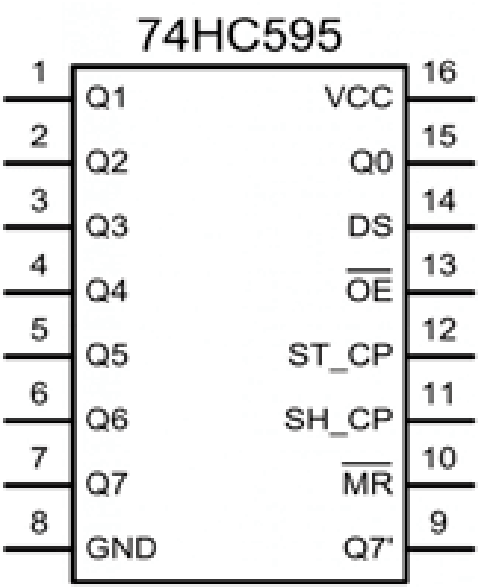

Рис. 11.5. Микросхема 74HC595

Пример программного кода 11.1 Использование сдвигового регистра 74HC595 для отображения чисел на семисегментном индикаторе

```
#define CLOCK 10 //SH_CP тактовый выход на 10-ом выводе
#define DATA 9  //DS выход даннах на 9-ом вывода
#define LATCH 8  //ST_CP разрешение отображения на 8-ом выводе

void setup() {

 //настраиваем контакты на выход
 pinMode(CLOCK, OUTPUT);
 pinMode(DATA, OUTPUT);
 pinMode(LATCH, OUTPUT);
  pinMode(2, OUTPUT); //разряд 0
 pinMode(3, OUTPUT); //разряд 1
 pinMode(4, OUTPUT); //разряд 2
 pinMode(5, OUTPUT); //разряд 3
 //отключаем LATCH (чтобы регистр не ждал данных)
 digitalWrite(LATCH, HIGH);
}

void loop() {
```

```
 //В цикле последовательно отображаем цифры от нуля до 9
//на четырёх разрядах семи сегментного индикатора
for (int i =1; i<=4; i++){
  switch (i) {
    case 1:
      digitalWrite(2, HIGH);
      taim();
  digitalWrite(2, LOW);//выполняется, когда i равно 1
       break;
       case 2:
     digitalWrite(3, HIGH);
     taim();
  digitalWrite(3, LOW); //выполняется когда  i равно 2
       break;
        case 3:
       digitalWrite(4, HIGH);
       taim();
  digitalWrite(4, LOW);//выполняется, когда i равно 3
       break;
    case 4:
     digitalWrite(5, HIGH);
     taim();
  digitalWrite(5, LOW); //выполняется когда  i равно 4
  //i=0;
       break;
    //default:
      // выполняется, если не выбрана ни одна альтернатива
      // default необязателен
  }
}
}
void taim()
{
 digitalWrite(LATCH, LOW);
//отображаем цифру 0
 shiftOut(DATA, CLOCK, LSBFIRST, 0b11111100);
 delay (500);
 digitalWrite(LATCH, HIGH);
 delay (1);
 digitalWrite(LATCH, LOW);
//отображаем цифру 1
 shiftOut(DATA, CLOCK, LSBFIRST, 0b01100000);
 delay (500);
 digitalWrite(LATCH, HIGH);
  delay (1);
```

```
 digitalWrite(LATCH, LOW);
//отображаем цифру 2
  shiftOut(DATA, CLOCK, LSBFIRST, 0b11011010);
 delay (500);
 digitalWrite(LATCH, HIGH);
  delay (1);
 digitalWrite(LATCH, LOW);
//отображаем цифру 3
 shiftOut(DATA, CLOCK, LSBFIRST, 0b11110010);
 delay (500);
 digitalWrite(LATCH, HIGH);
  delay (1);
 digitalWrite(LATCH, LOW);
//отображаем цифру 4
 shiftOut(DATA, CLOCK, LSBFIRST, 0b01100110);
 delay (500);
 digitalWrite(LATCH, HIGH);
  delay (1);
 digitalWrite(LATCH, LOW);
//отображаем цифру 5
 shiftOut(DATA, CLOCK, LSBFIRST, 0b10110110);
 delay (500);
 digitalWrite(LATCH, HIGH);
  delay (1);
 digitalWrite(LATCH, LOW);
//отображаем цифру 6
 shiftOut(DATA, CLOCK, LSBFIRST, 0b10111110);
 delay (500);
 digitalWrite(LATCH, HIGH);
 delay (1);
 digitalWrite(LATCH, LOW);
//отображаем цифру 7
 shiftOut(DATA, CLOCK, LSBFIRST, 0b11100000);
 delay (500);
 digitalWrite(LATCH, HIGH);
  delay (1);
 digitalWrite(LATCH, LOW);
//отображаем цифру 8
 shiftOut(DATA, CLOCK, LSBFIRST, 0b11111110);
 delay (500);
 digitalWrite(LATCH, HIGH);
  delay (1);
 digitalWrite(LATCH, LOW);
//отображаем цифру 9
 shiftOut(DATA, CLOCK, LSBFIRST, 0b11110110);
```

```
 delay (500);
digitalWrite(LATCH, HIGH);
  delay (1);
digitalWrite(LATCH, LOW);
//отображаем точку
 shiftOut(DATA, CLOCK, LSBFIRST, 0b00000001);
 delay (500);
 digitalWrite(LATCH, HIGH);
 delay (1);
 digitalWrite(LATCH, LOW);
//гасим все сегменты
 shiftOut(DATA, CLOCK, LSBFIRST, 0b00000000);
 delay (500);
 digitalWrite(LATCH, HIGH);
 delay (1);
}
```

Задания и порядок выполнения работы

Доработать программный код предложенный в примере и обеспечить отображение на семисегментных индикаторах отображение напряжения считываемое с аналогового порта A0.

Контрольные вопросы

1. Какой принцип работы сдвигового регистра 747HC595?
2. Как подключить сдвиговый регистр 74HC595 к Arduino?
3. Опишите код программы для работы с семи сегментным дисплеем при использовании сдвигового регистра 747HC595.
4. Как осуществляется передача данных по шине SPI?
5. Что необходимо для того чтобы подключить несколько регистров 747HC595 к шине SPI.

Лабораторная работа № 12.
Отображение данных на LCD

Цель работы: приобрести практические навыки по управлению LCD дисплеем и использовать его для вывода данных.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программ указанных в примерах.
4. Выполнить задания (порядок выполнения заданий аналогичен указанному порядку в примерах).
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

Жидкокристаллические индикаторы (ЖКИ, англ. LCD) являются удобным и недорогим средством для отображения данных ваших проектов. Символьный индикатор WH1602 позволяет выводить на экран 2 строки по 16 символов (размером 5×7 или 5×10 и дополнительная строка под курсор). Управляет работой дисплея контроллер.

На рис. 12.1 показан ЖКИ Winstar с контроллером HD44780.

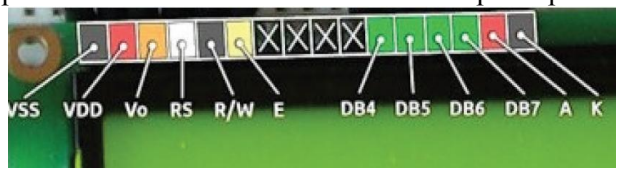

Рис 12.1. Назначение выводов ЖКИ WH1602 на основе контроллера HD44780

Назначение выводов контроллера:

– **DB0–DB7** – отвечают за входящие/исходящие данные;

– **RS** – высокий уровень означает, что сигнал на выходах DB0–DB7 является данными, низкий – командой;

– **R/W** – определяет направление данных (чтение/запись). Так как операция чтения данных из индикатора обычно бывает невостребованной, то можно установить постоянно на этом входе низкий уровень;

– **E** – импульс длительностью не менее 500 мс на этом выводе определяет сигнал для чтения/записи данных с выводов DB0–DB7, RS и W/R;

– **V0** – используется для задания контраста изображения;

– **A, K** – питание подсветки (анод и катод), если она имеется;

– **VSS** – земля;

– **VDD** – питание ЖК-индикатора.

Для управления ЖК-индикатором необходимо 6 или 10 выводов Arduino, в зависимости от того, выбран 4- или 8-битный режим обмена данными. Для сокращения требуемого числа выводов микроконтроллера можно работать в 4-битном режиме. В этом случае на выводах DB4–DB7 индикатора сначала будут передаваться старшие четыре бита данных/команды, затем – младшие четыре бита. Выводы DB0–DB3 останутся незадействованными.

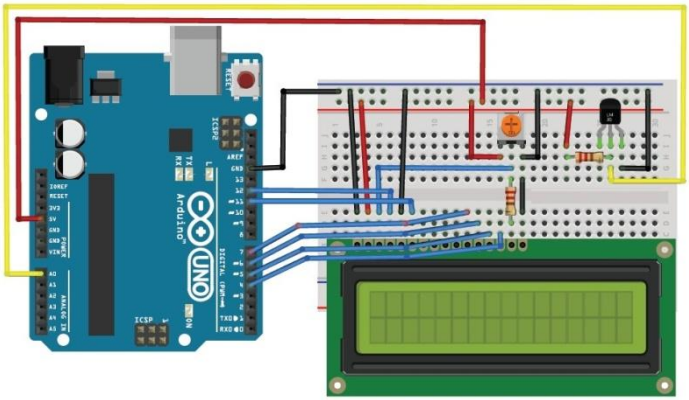

Рис 12.2. Схема подключения датчика температуры и ЖКИ WH1602 к Arduino

Приступим к написанию скетча. Функционал Arduino может быть расширен за счет использования библиотек. Библиотеки Arduino предоставляют дополнительную функциональность для использования в скетчах и сильно упрощают процесс написания программ. Ряд основных библиотек устанавливается вместе со средой Arduino IDE, а дополнительные, которых очень много, вы можете установить сами.

При работе Arduino с ЖКИ-дисплеями на контроллере HD44780 будем использовать библиотеку **LiquidCrystal**. Для подключения библиотеки в начале скетча вставляем строку

```
#include
```

Затем создаем переменную типа LiquidCrystal:

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

где 12, 11, 5, 4, 3, 2 – номера контактов RS, E, D4, D5, D6, D7.

В setup() запускаем функцию lcd.begin(), определяющую размерность индикатора, для установки курсора в определенную позицию – lcd.setCursor(), для вывода информации на экран дисплея – lcd.print().

Пример программного кода 12.1.Отображение изменения режимов работы при помощи ЖКИ кнопок и светодиода

```
#include <LiquidCrystal.h> //Подключим библиотеку для дисплея.
//Покажем, к каким выводам мы подключили дисплей
LiquidCrystal lcd(12, 11, 5, 4, 3 ,2);
#define BUT2 A2 //к аналоговому выводу A2 подключена кнопка2
#define BUT1 A1 //к аналоговому выводу A1 подключена кнопка1
#define LED A0 //к аналоговому выводу A0 подключен светодиод
int buttonState1 = 0;
int buttonState2 = 0;
unsigned long currentTime;
unsigned long loopTime;
int ledState = LOW;
void setup()
{
 lcd.begin(16, 2); //Сообщаем кол-во строк и столбцов.
 lcd.print ("Command's name:"); //В первую строку выводим сообщение.
 pinMode(LED, OUTPUT);
 pinMode(BUT1, INPUT);
 pinMode(BUT2, INPUT);
 currentTime = millis(); // считываем время, прошедшее с момента запуска программы
 loopTime = currentTime;
```

```
}
void loop() {
 currentTime = millis();
 lcd.setCursor(0, 1); //Ставим курсор на 1 строку, колонку 0.
 buttonState1 = digitalRead(BUT1);
 buttonState2 = digitalRead(BUT2);
  //Выводим сообщения параллельно с выполнением задач.
  if( buttonState1 == HIGH)// если нажата 1-я кнопка то
  {                        //то выводится надпись "Led on    " и
    lcd.print ("Led on   ");
    digitalWrite(LED, HIGH); //загорается светодиод
  }
  else if( buttonState2 == HIGH) // если нажата 2-я кнопка то
  {                        //то выводится надпись "Led off" и
    lcd.print ("Led off");
    digitalWrite(LED, LOW); //тухнет светодиод
  }
  else if( buttonState1 == LOW && buttonState2 == LOW)
  {  //если кнопки не нажаты то светодиод мигает и выводится надпись
    lcd.print ("Blink   "); //и выводится надпись  "Blink   "
   if(currentTime >= (loopTime + 500)){
     ledState = !ledState;
     loopTime = currentTime;
    }
    digitalWrite(LED, ledState);
  }
}
```

Порядок подключения:

1. Подключаем ЖКИ к отладочной плате.

2. Загружаем в плату Arduino скетч из пример программного кода 12.1.

3. Смотрим на экране дисплея изменение надпеси режимов при нажатии кнопок и изменение яркости свечения светодиода .

Отображение данных на LCD по протоколу I2C

Стандартная схема подключения не всегда удобна, так как занимает как минимум 6 цифровых выходов. Рассмотрим способ, как это можно обойти и использовать только два аналоговых выхода.

Нам понадобится дополнительный модуль 1602 LCD конвертор в IIC/I2C. Он припаивается к нашему дисплею как видно на рисунке ниже:

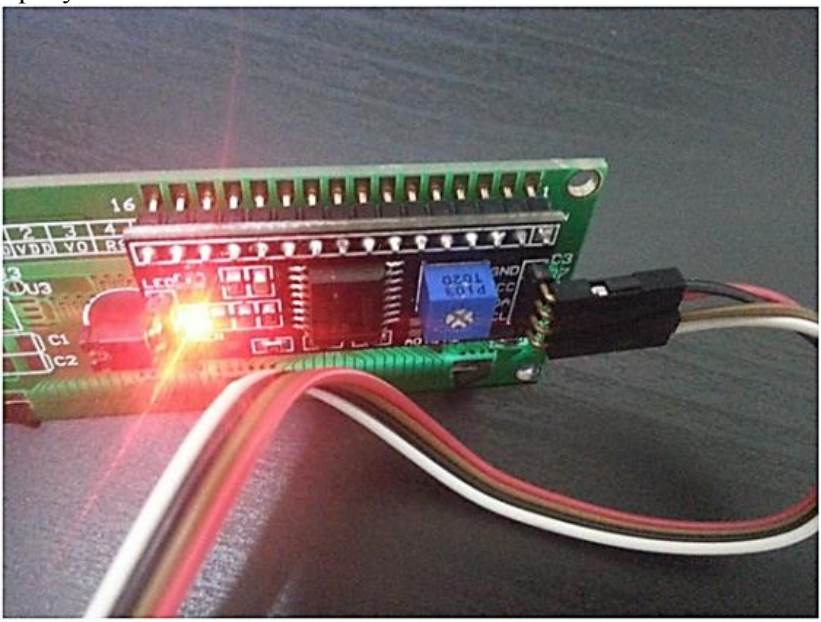

Рис. 12.3. Внешний вид ЖКИ WH1602 с модулем I2C

И подключается к отладочной плате Arduino следующим образом:

Таблица 12.1. Нумерация выводов шины I2C для плат Arduino Mega и Arduino Uno

Arduino Mega	***Arduino Uno***	***LCD i2c***
GND	GND	GND
5V	5V	VCC
20 (SDA)	A4	SDA
21(SCL)	A5	SCL

Этот способ подключения работает только со специально библиотекой, скачать библиотеку можно по ссылке (LiquidCrystal_I2C1602V1 Библиотека). Установите данную библиотеку.

На дисплее LCD 1602 русский шрифт не работает, но можно вывести кириллицу на Arduino с помощью собственных символов. Рассмотрим, как сделать любые символы и буквы на кириллице на LCD дисплее Arduino. Для этого нам потребуется использовать в скетче переменную byte и функцию lcd.createChar(). Сразу отметим, что объем памяти на Arduino для знаков ограничен всего восемью символами.

Выводим свой символ на LCD 1602

Вывести свой символ или кириллическую букву на дисплей поможет таблица знакогенератора (CGROM). Такой вид памяти в Arduino, как CGRAM, может хранить собственные символы, но размер памяти ограничен и может вместить лишь 8 собственных символов. Один из нестандартных символов, который пригодится для создания домашней метеостанции – знак градуса. Давайте нарисуем символ.

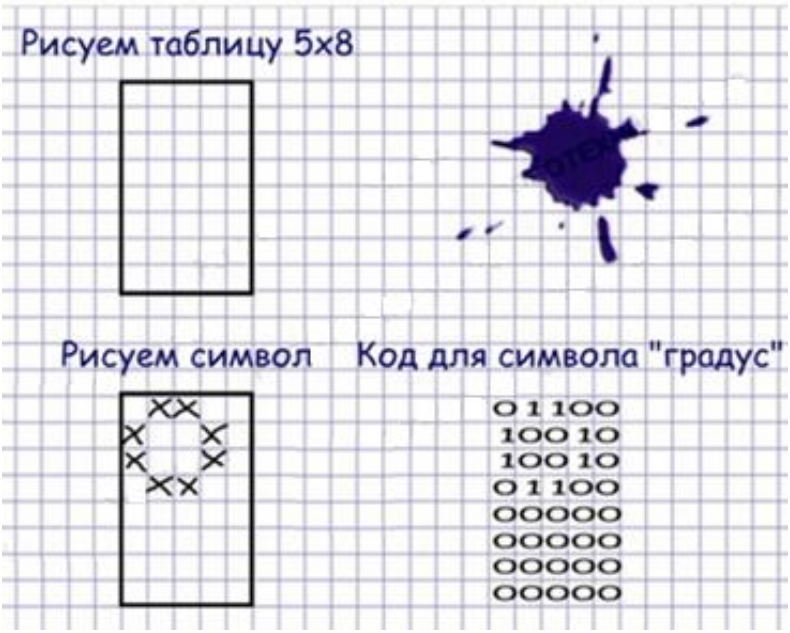

Рис. 12.4. Создание своего символа для LCD дисплея 1602

Для начала возьмите листок бумаги и нарисуйте на нем таблицу, где будет 5 столбцов и 8 строчек. Далее заштрихуйте в таблице клеточки (смотри рисунок выше), которые должны высвечиваться на дисплее. Дело в том, что каждый символ на дисплее состоит из пикселей (5 пикселей в ширину и 8 пикселей в высоту). Далее представим наш символ в виде массива данных, состоящего из восьми элементов – восьми строк.

Например русская буква «П»:

```
uint8_t bukva_t[8] = {
  B11111,
  B10001,
  B10001,
  B00001,
  B10001,
  B10001,
  B10001
      };
```

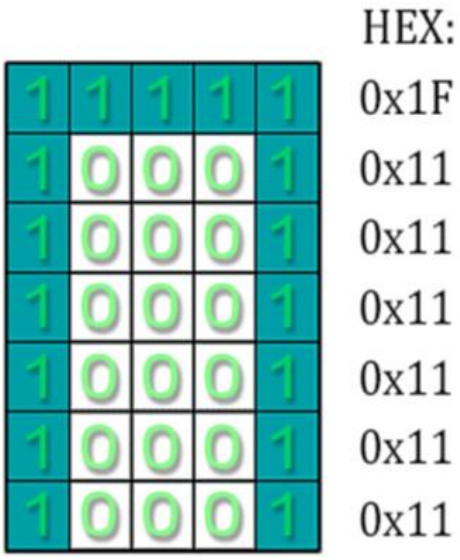

Рис. 12.5. Отображение созданного символа на дисплее ЖКИ

Подключение LCD дисплея по I2C производится четырьмя проводами – 2 провода данных и 2 провода питания. Если вы подключаете дисплей Ардиуно УНО, используйте следующую

схему – вывод SDA подключается к порту A4, вывод SCL – к порту A5 и два провода питания – GND и 5V. Если LCD 1602 подключается по шине I2C к Arduino MEGA, то на плате имеются соответствующие порты – SDA и SCL.

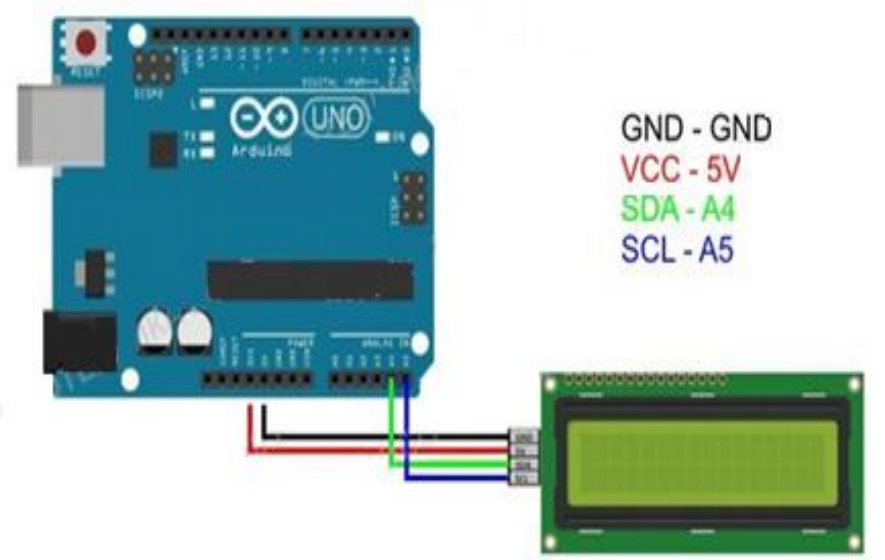

Рис. 12.6. Подключение LCD 1602 к Arduino через I2C

Дисплей поддерживает только 8 новых символов или русских букв (пронумерованных от 0 до 7) размером 5 на 8 пикселей. Букву на кириллице, как и символ, можно задать массивом из восьми байт, характеризующий соответствующую строку. Поскольку можно добавить только до 8 новых букв и символов, то используйте для написания фраз латинские и кириллические буквы, как на примере ниже.

После подключения монитора к Arduino загрузите пример приведенный ниже:

Пример программного кода 12.2 Вывод русских букв и произвольных символов на ЖКИ по шине I2C:

```
//Подключаем библиотеки
#include <OneWire.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#if defined(ARDUINO)&&ARDUINO>=100
```

```
#define printByte(args) write(args);
#else
#define printByte(args) write(args);
#endif
//Создаём символ русской буквы "т"
uint8_t bukva_t[8]={
  B00000,
  B00000,
  B11111,
  B00100,
  B00100,
  B00100,
  B00100,
};
//Устанавливаем адрес 0x27 и колличество символов – 16
//и колличество строк -2
LiquidCrystal_I2C lcd(0x27,16,2);
void setup() {
 lcd.init(); //инициализация LCD
 lcd.backlight(); //включаем подсветку
 lcd.clear();  //очищаем экран
 lcd.createChar(0, bukva_t); //создаем символ и записываем его в
 //нулевую ячейку памяти LCD (всего их восемь)
 lcd.setCursor(0, 0); //устанавливаем курсор на нулевую ячейку
// нулевой строки
 lcd.print("Рос"); //выводим на экран надпись                Рос
 lcd.printByte(0); //выводим на экран русскую букву      т
 lcd.print("ok"); //выводим на экран                          ok
}

void loop() {
  //Выводим счетчик секунд в нулевой ячейке 1-й строки
 lcd.setCursor(0, 1);
 lcd.print(millis()/1000);
 delay(100);
}
```

Задания и порядок выполнения работы

1. Доработать программный код предложенный в примере и обеспечить отображение на LCD индикаторе отображение напряжения считываемое с аналогового порта A3

2. Напишите свою фамилию, имя и отчество на ЖК-дисплее латинскими буквами.

3. Придумайте свой собственный символ и выведите его на ЖК-дисплей.

4. Напишите свою фамилию, имя и отчество на ЖК-дисплее русскими буквами.

Контрольные вопросы

1. Поясните правила подключения двустрочного символьного жидкокристаллического индикатора.
2. Опишите код программы для работы с жидкокристаллическим дисплеем.
3. Как использовать русские символы на ЖК-дисплее?
4. Какие команды используются для вывода символов на ЖК-дисплей?
5. Какие библиотеки используются при работе с ЖК-дисплеем?

Заключение

Развитие электронной промышленности и компьютеростроения осуществляется такими быстрыми темпами, что буквально через пару лет сегодняшнее «чудо техники» становится морально устаревшим. Однако принципы устройства компьютера остаются неизменными еще с того момента, как знаменитый математик Джон фон Нейман в 1945 году подготовил доклад об устройстве и функционировании универсальных вычислительных устройств, то есть компьютеров. Микроконтроллеры развиваются невероятными темпами и их можно встретить в огромном количестве современных промышленных и бытовых приборах: станках, автомобилях, телефонах, телевизорах, холодильниках, стиральных машинах и даже кофеварках.

Ни одна другая машина в истории не привнесла в наш мир столь быстрых и глубоких изменений. Благодаря компьютерам стали возможными такие знаменательные достижения, как посадка аппаратов на поверхность Луны и исследование планет Солнечной системы. Компьютеры создают тысячи удобств и услуг в нашей повседневной жизни. Они управляют анестезионной аппаратурой в операционных, помогают детям учиться в школах, «изобретают» видеотрюки для кинематографа. Компьютеры взяли на себя функции пишущих машинок в редакциях газет и счетных аппаратов в банках. Они улучшают качество телевизионного изображения, управляют телефонными станциями и определяют цену покупок в кассе универсального магазина. Иными словами, они столь прочно вошли в современную жизнь, что обойтись без них практически невозможно.

Arduino – аппаратная вычислительная платформа, основными компонентами которой являются плата ввода-вывода и среда разработки. Arduino может использоваться как для создания автономных интерактивных объектов, так и подключаться к программному обеспечению, выполняемому на компьютере. Arduino как и относится к одноплатным компьютерам. Arduino часто используется

как мозг робота. Преимущество плат Arduino перед аналогичными платформами – относительно невысокая цена и практически массовое распространение среди любителей и профессионалов робототехники и электротехники.

Именно поэтому важно знать основные принципы работы и устройства компьютера, а также его основных составных, сфера применения которых безгранична.

Рекомендованная литература

1. Банци М. Arduino для начинающих волшебников / Массимо Банци – М.: Рид Групп, 2012. – 128 с.
2. Белов А.В. Программирование ARDUINO: создаем практические устройства + виртуальный диск / А.В. Белов. – С-П.: Наука и Техника, 2018. – 270 с.
3. Блум Д. Изучаем Arduino®: инструменты и методы технического волшебства / Джереми Блум – С-П.: БХВ-Петербург, 2017. – 336 с.
4. Бокселл Дж. Изучаем Arduino: 65 проектов своими руками / Джон Бокселл – С-П.: Питер, 2017. – 396 с.
5. Момот М.В. Мобильные роботы на базе Arduino: руководство для начинающих конструкторов / М.В. Момот. – С-П.: БХВ-Петербург, 2017. – 288 с.
6. Монк С. Программируем Arduino: профессиональная работа со скетчами / Саймон Монк – С-П.: Питер, 2017. – 272 с.
7. Петин В.А., Биняковский А.А Практическая энциклопедия Arduino / В.А. Петин, А.А Биняковский – М.: ДМК Пресс, 2017. – 151 с.
8. Петин В.А. Проекты с использованием контроллера Arduino / Виктор Петин – С-П.: БХВ-Петербург, 2014. – 398 с.
9. Платт Ч. Электроника для начинающих / Чарльз Платт – 2-е изд. – С-П.: БХВ-Петербург, 2017. – 395 с.
10. Ревич Ю.В. Азбука электроники. Изучаем Arduino / Юрий Ревич. – М.: АСТ, 2017. – 223 с.
11. Ревич Ю.В. Занимательная электроника / Юрий Ревич. – 4-е изд., перераб. и доп. – С-П.: БХВ-Петербург, 2017. – 639 с.

12. Рюмик С.М. 1000 и одна микроконтроллерная схема / С.М. Рюмик. – М.: ДМК-Пресс, 2016. – 356 с.

13. Соммер У. Программирование микроконтроллерных плат Arduino/Freeduino / Улли Соммер – 2-е изд. – С-П.: БХВ-Петербург, 2016. – 254 с.

14. Танака К. Занимательная электроника. Электронные схемы / Кэнъити Танака – М.: ДМК Пресс, 2015. – 184 с.

15. Ярнольд С. Arduino для начинающих: самый простой пошаговый самоучитель / Стюарт Ярнольд – М.: Э, 2017. – 253 с.

Учебное издание

СЕНТЯЙ Роман Николаевич
ШИШЛАКОВА Виктория Николаевна

АРХИТЕКТУРА ЭВМ И МИКРОКОНТРОЛЛЕРОВ

Учебно-методические рекомендации
по выполнению лабораторных работ
для студентов очной формы обучения
по направлению подготовки
09.03.01 «Информатика и вычислительная техника»

В авторской редакции
Редактор – Сентяй Р.Н.
Дизайн обложки – Швыров В.В.
Корректор – Шишлакова В.Н.
Верстка – Шишлакова В.Н.

Подписано в печать 10.09.2018. Бумага офсетная.
Гарнитура Times New Roman.
Печать ризографическая. Формат 60x84/16. Усл. печ. л. 7,9
Тираж 50 экз. Заказ. № 95.

Издатель ГОУ ВПО ЛНР
«Луганский национальный университет
имени Тараса Шевченко»
«Книта»
ул. Оборонная, 2, г. Луганск, ЛНР, 91011. Т/ф: (0642)58-03-20
e-mail: knitaizd@mail.ru