ПРИДНЕСТРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Т.Г.ШЕВЧЕНКО
Инженерно-технический институт
Факультет среднего профессионального образования
Кафедра «Интегрированных компьютерных технологий
и систем»

Лабораторные работы по микропроцессорным системам

Методические указания

Тирасполь, 2016

УДК 681.32 (072.8)
ББК з 971.322.5 р 30
Л 12

Составители: Е.Н.Аксёнов, преподаватель
А.В.Деткова, ст. преподаватель

Рецензенты:
С.А.Устименко, декан ФСПО, доцент
(Приднестровский государственный университет)
О.М. Фурдуй, зав. кафедрой ИКТиС, доцент
(Приднестровский государственный университет)

Лабораторные работы по микропроцессорным системам: Методические указания/ сост.: Е.Н. Аксёнов, А.В. Деткова – Тирасполь, 2016. – 90 с.

Методические указания содержат цикл рационально составленных лабораторных работ, позволяющих освоить программирование микроконтроллеров Arduino. Лабораторные работы позволяют изучить основные этапы разработки и проектирования автоматических устройств на базе микроконтроллеров.

Предназначены для выполнения лабораторных работ по дисциплине «Микропроцессорные системы», а так же по дисциплине «Теоретические основы контроля и анализа функционирования систем автоматического управления» и подготовлены в соответствии с учебными планами ФСПО ИТИ ПГУ и рекомендованы студентам данной специальности.

Рекомендовано Научно-методическим Советом ПГУ им. Т.Г. Шевченко.

СОДЕРЖАНИЕ

Введение

Применение микропроцессорных систем практически во всех электрических устройствах - важнейшая черта технической инфраструктуры современного общества. Электроэнергетика, промышленность, транспорт, системы связи существенно зависят от компьютерных систем управления. Микропроцессорные системы встраиваются в измерительные приборы, электрические аппараты, осветительные установки и другие устройства.

Целью изучения дисциплины является формирование у студентов знания общей методологии, а также конкретных методов проектирования основных разновидностей современных микропроцессорных средств.

Методические указания по дисциплине "Микропроцессорные системы" содержат цикл рационально составленных лабораторных работ, позволяющих освоить программирование микроконтроллеров Arduino. 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.

До появления Ардуино 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. Это мощный шаг вперед, в то время как некоторые довольно популярные устройства представляют собой "черные ящики", закрытые и защищенные патентами.

Лабораторная работа №1

Тема: Изучение возможностей платформы Arduino

Цель: изучить программную и аппаратную часть платформы Arduino, возможности и достоинства данной платформы, разновидности плат Arduino

1. 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 данной 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 насчитывается 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 Ардуино 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 Ардуино 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 беспаячной макетной плате, с помощью специальных проводков.

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

– **Наличие САПР (систем автоматизированного проектирования), эмуляторов.** Также с открытым исходным кодом, кроссплатформенные. Можно даже на специальных программах проверить как будет вживую работать плата Ардуино 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 Ардуино, 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 Ваших 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 Ардуино. На ней находятся большинство узлов, которые чаще всего используются в Ардуино-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 обработка 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.

2. 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.

Все «размеры» и разновидности ардуино-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).

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

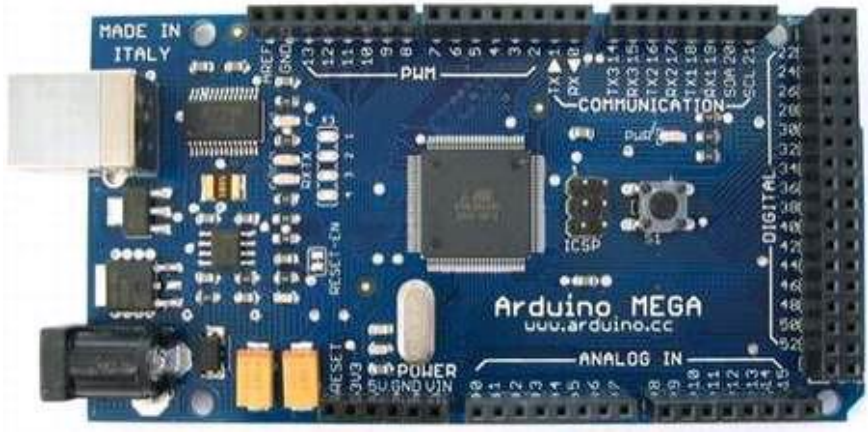

Рисунок 1- Arduino MEGA

ArduinoMega (рис.1) 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 Ардуино.

Рисунок 2 - Arduino UNO

Arduino UNO (рис.2) самая функциональная плата со стандартным набором узлов. Очень удобна при подключении к ней уже готовых узлов и плат расширения.

Arduino Nano (рис.3) – плата, аналогичная Arduino UNO, но с гораздо меньшим размером и удобными разъемами для макетирования на беспаечных макетных платах.

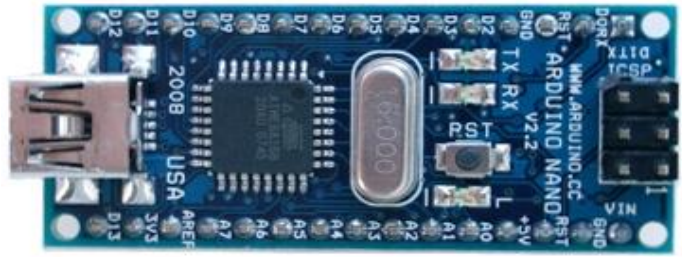

Рисунок 3-Arduino Nano

Рассмотрим описание Arduino Nano.

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

Рассмотрим плату Arduino Nano (рис. 4), построенную на базе микроконтроллера (МК) ATmega 328. Основными элементами являются: 14 цифровых портов ввода/вывода (от D0 до D13), 8 аналоговых входов (от A0 до A7), кварцевый резонатор (генератор тактов) частотой 16 мегагерц (слева от кнопки сброса), разъем mini-USB для подключения к компьютеру, несколько выводов питания, кнопка перезагрузки и сам МК.

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

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.

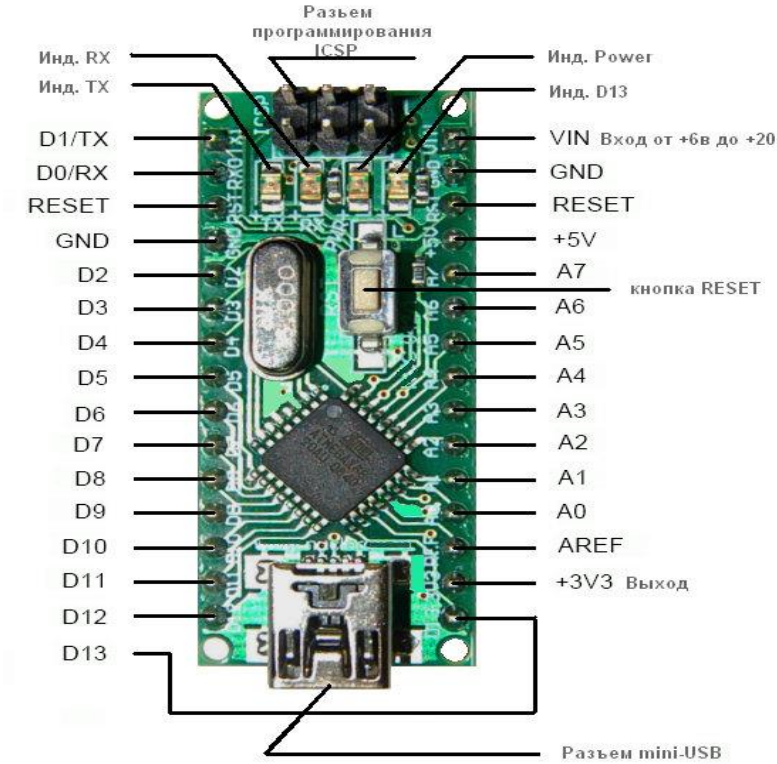

Рисунок 4 - Arduino Nano

Кварцевый резонатор заставляет работать МК на определенной частоте, без него микроконтроллер просто не сможет работать. По сути, это «сердце» Ардуино, бьющееся 16 миллионов раз в секунду. И за каждый удар такого сердца микроконтроллер выполняет одну команду из его программы. Максимальная частота, на которой может работать микроконтроллер- 20 мегагерц. Т.е. 20 миллионов команд в одну секунду. Однако следует обратить внимание, что это не те команды, которые мы будем писать для программы. Одна написанная нами команда может превратиться в несколько десятков или даже несколько тысяч команд самого микроконтроллера. Но они всё равно будут очень быстро выполняться. Как правило, микроконтроллеры на платах Ардуино работают с частотами 16 или 8 мегагерц. Чем больше частота-тем

больше потребление энергии микроконтроллером. К тому же не все микроконтроллеры для Ардуино могут работать на частотах выше 16 мегагерц.

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 Ардуино с помощью этого программатора. Либо может быть использован для того чтобы превратить ненадолго саму Ардуино в этот самый специализированный программатор. Мы эту возможность не будем использовать.

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. Какие платы Ардуино самые распространенные? Поясните их назначение.
6. Перечислите основные элементы платы Arduino Nano (рис.4).

Лабораторная работа №2

Тема: Среда разработки Arduino IDE

Цель: изучить среду разработки для платформы Arduino - Arduino IDE, правила подключения и установки Arduino Nano

1. Установка среды разработки

Рассмотрим среду разработки для платформы Arduino - Arduino IDE.

Свежую версию всегда можно найти на официальном сайте www.arduino.cc. Базовые среды Arduino 1.6.x, 1.5.x, 1.0.x. Рекомендуем скачать все три - например, 1.0.6, 1.5.8, 1.6.5. в виде zip архивов и разархивировать их в три разные папки с соответствующим названием. Внешне они мало чем отличаются, но внутреннее отличие весьма существенно. Запускается среда путем запуска файла Arduino.exe в соответствующей папке.

Среда разработки Arduino (рис.5) представляет собой текстовый редактор программного кода, область сообщений, окно вывода текста (консоль), панель инструментов и несколько меню. Для загрузки программ и связи среда разработки подключается к аппаратной части Arduino.

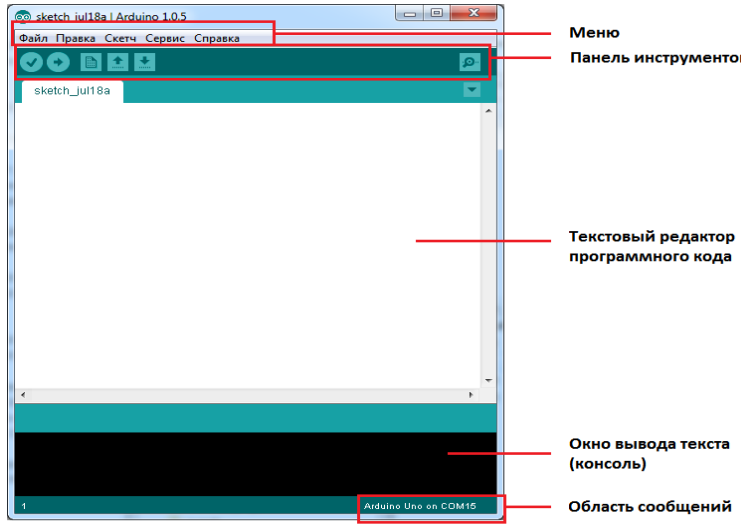

Рисунок 5 - Среда разработки

Рассмотрим рабочую панель. Меню "Файл" (рис.6). Все пункты меню очевидны. В подпункте «Образцы» - хранятся готовые скетчи примеры. По умолчанию Arduino IDE сохраняет каждый скетч в отдельную папку. Имя папки совпадает с именем, указанным для скетча при сохранении. Изменить рабочую директорию для папок со скетчами можно в пункте меню "Настройки".

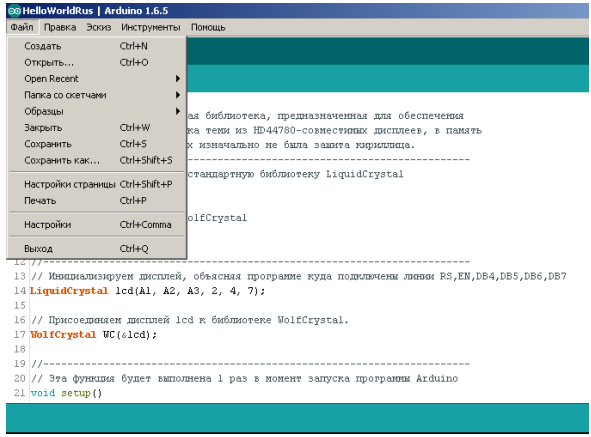

Рисунок 6 - Меню «Файл»

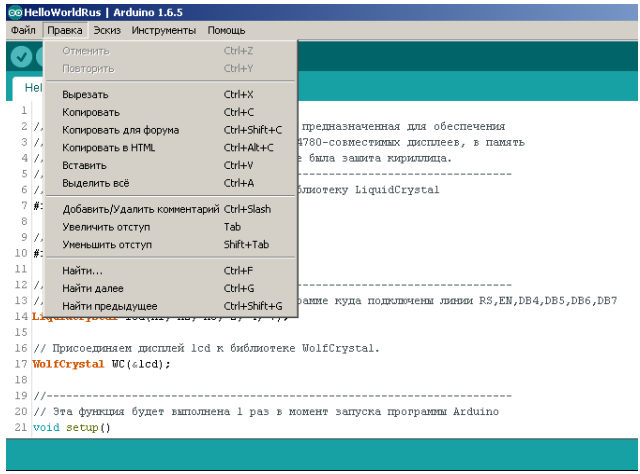

Рисунок 7 - Меню «Правка»

В меню "Правка" расположены команды для работы с кодом программы (рис.7). Некоторые команды удобны наличием комбинаций для быстрого доступа посредством клавиатуры. Удобными функциями являются возможность копирования для форумов в HTML формате.

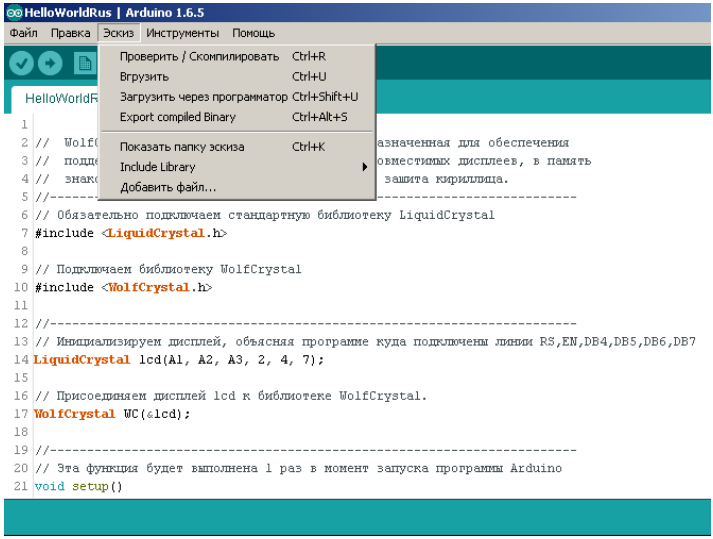

Рисунок 8 - Меню «Эскиз»

Рассмотрим меню "Эскиз" (рис.8). В данном меню продублирована команда из панели управления "Проверить / Скомпилировать", выполнение которой приведет к проверке кода на ошибки, и в случае их отсутствия - к компиляции кода.

Пункт меню "Показать папку эскиза" откроет рабочую директорию Arduino IDE, указанную в настройках.

"Добавить файл..." позволяет открыть текстовый файл (или скетч) в отдельной вкладке.

Include Library позволяет установить в среду Ардуино IDE новую, еще не установленную библиотеку. Но лучше этим пунктом не пользоваться - не всегда он работает корректно. Лучше напрямую разархивировать папку с библиотекой в папку «Мои документы/Arduino/Libraries». Папка с библиотекой не должна содержать в своем названии тире, пробелов, русских букв. Если содержит- просто их уберите или переименуйте ее. Еще не должно

быть такого, что папка с библиотекой содержит только одну подпапку а в ней файлы. В этом случае по пути «Мои документы/Arduino/Libraries» должна лежать эта подпапка с файлами. После этого среду Ардуино нужно перезапустить. Библиотека должна появиться в меню «Файл»-> «Образцы».

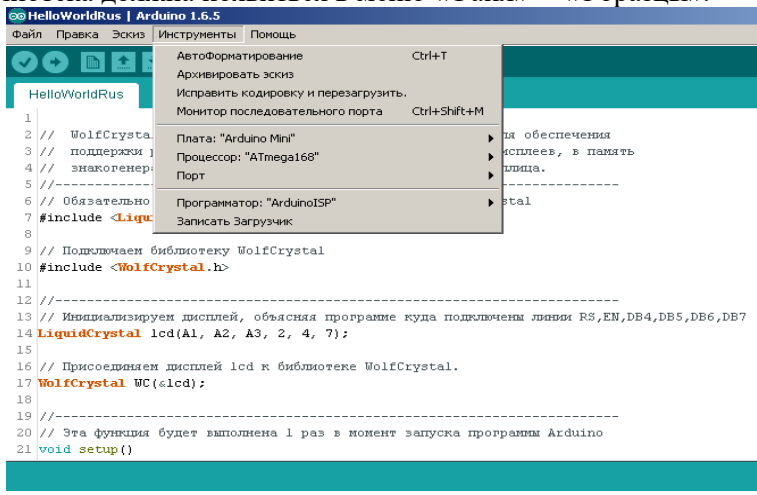

Рисунок 9 - Меню «Инструменты»

В меню "Инструменты" необходимо указать модель вашей Arduino платы (рис.9), а также COM порт, к которому она подключена. Вариантов выбора COM порта не много, поэтому его легко угадать.

Удобной функцией является автоформатирование, которая позволяет исправить разметку скетча и привести его в удобочитаемый вид. Особенно актуально при копировании сторонних программ.

Пункт меню "Монитор последовательного порта" вызывает окно для обмена сообщениями с Arduino через COM порт.

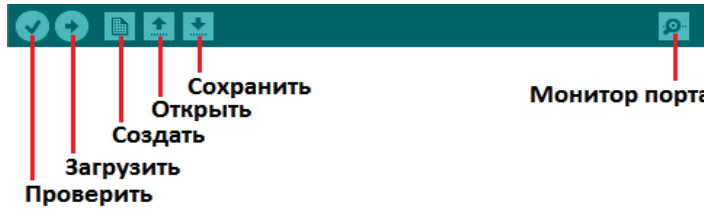

Рисунок 10 - Панель управления

Команды панели управления дублируют некоторые пункты меню (рис.10).

3. Подключение и установка Arduino Nano

После извлечения из архивов среды подключаем плату Arduino Nano к компьютеру. Если драйвера не установились автоматически, то вы увидите следующее окно:

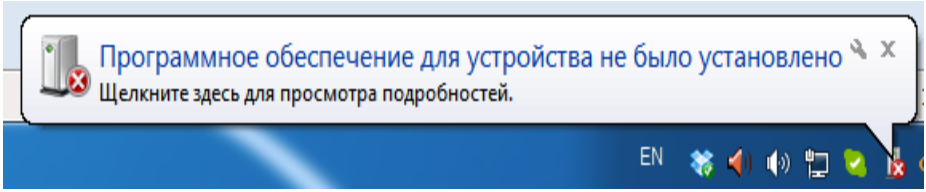

Рисунок 11- Драйвера не установлены

Открываем Мой Компьютер/Свойства/Диспетчер устройств (рис 12).

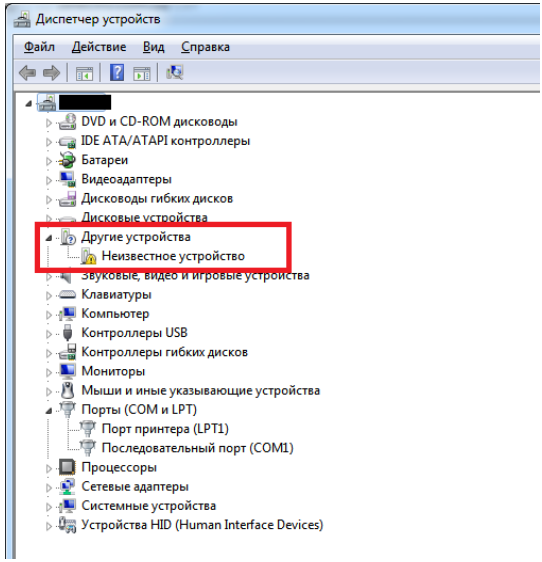

Рисунок 12 - Диспетчер устройств

Два раза нажимаем по "Неизвестному устройству". Далее нажимаем кнопку "Обновить драйвер"(рис.13).

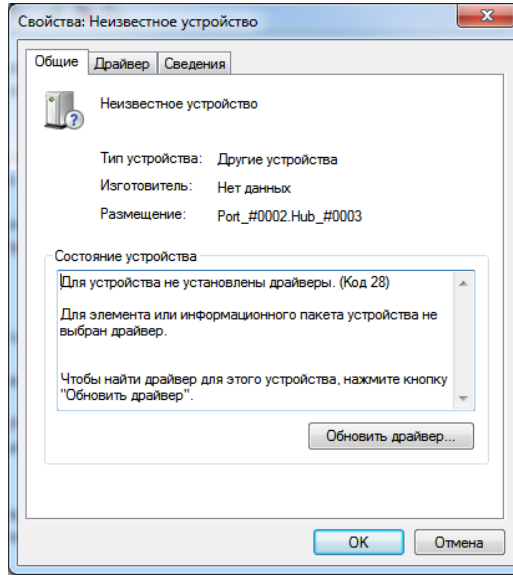

Рисунок 13 - Обновление драйвера

Выбираем пункт "Выполнить поиск драйверов на этом компьютере" (рис.14).

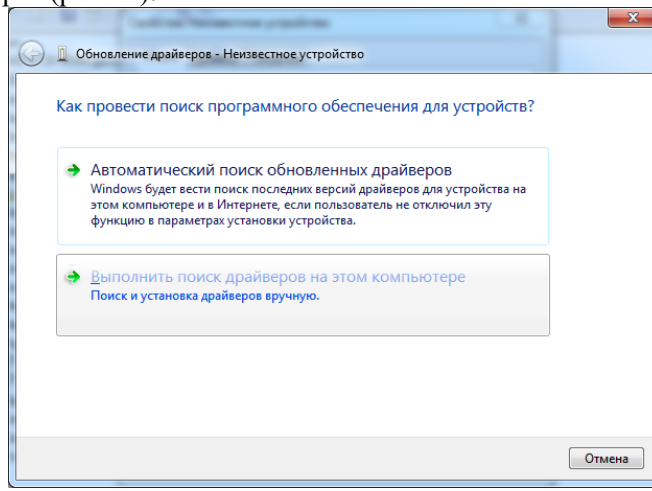

Рисунок 14 - Установка драйверов вручную

Указываем директорию, куда вы установили Arduino IDE, а точнее её подпапку "drivers" (рис.15).

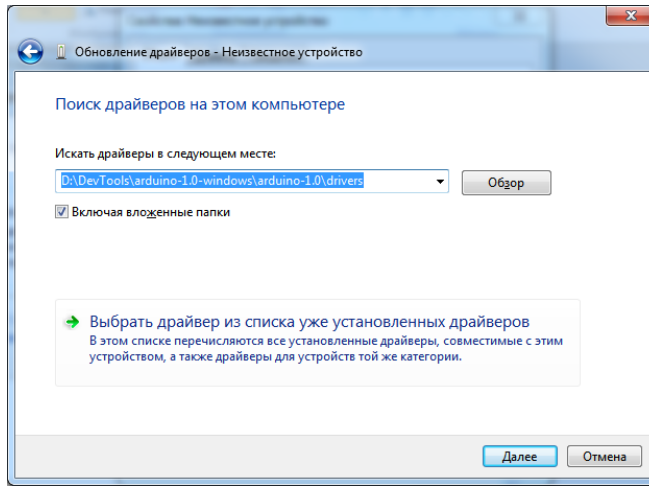

Рисунок 15 - Путь к папке с драйверами

Причем желательно указать самую старую версию IDE. Галочку "Включая вложенные подпапки" оставляем.

Если выскочит предупреждение, все равно соглашаемся на установку драйверов.

Ожидаем некоторое время. Драйвера установились и наша плата готова к работе. Остаётся только запустить Arduino IDE, выбрать тип платы и потом COM порт.

Контрольные вопросы:

1. Опишите интерфейс среды разработки Arduino.
2. Опишите панель Меню.
3. Какова последовательность шагов при подключении и установке Arduino Nano?

Лабораторная работа №3

Тема: Изучение структуры программы для платформы Arduino

Цель: изучить структуру программы - скетч, базовые элементы Processing C/C++, типы данных

1. Основные сведения

Скетч - это программа, написанная для платформы Arduino и имеющая определенную структуру.

Функция – это блок кода, имеющий строго заданное имя с возможностью вызова её по этому имени. При этом функция выполняет какое-то законченное действие. Тело функции ограничивается операторами "{" (открывающая фигурная скобка) и "}" (закрывающая фигурная скобка).

Скетч обязательно содержит 2 функции (рис.16): функцию setup() и функцию loop().

Прошивка Arduino при включении или сразу же после успешного программирования вызывает функцию setup(). Функция setup() вызывается лишь раз, при каждом запуске платы. Это место идеально подходит для инициализации (задания начальных значений) переменных, установки режимов пинов (ввод/вывод), задания соответствия подключенных датчиков/сервоприводов/прочего с пинами.

После выполнения функции setup идет циклический вызов функции loop() (т.е. сразу после выхода из функции setup, выполняется функция loop(), после выхода из неё, она же вызывается снова. Процесс продолжается пока питание не будет отключено.

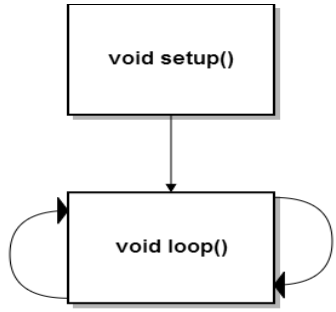

Рисунок 16 - Структура программы

2. Базовые элементы Processing C/C++

Программы, а точнее как их принято называть - "скетчи", для Arduino пишутся на языке, основанном на C/C++.

Рассмотрим такие элементы базового синтаксиса, как ";"(точка с запятой), "{}"(фигурные скобки), "//"(однострочный комментарий), "/* */" (многострочный комментарий), директивы препроцессора #define и #include.

Комментарии в программе бывают двух типов: однострочные и многострочные. Комментарии игнорируются компилятором и не влияют на работу программы, а используются лишь в качестве информации нам об участках кода.

Говорят, что хороший код не требует комментариев. Но всегда есть исключения, поэтому рекомендуется все неочевидные участки комментировать. Не нужно комментировать очевидные вещи. Например, комментарий к строке a=2; //«переменной a присваиваем значение 2» является очевидным. Помните, то, что может быть понятно сейчас, вероятно будет очень сложно разобрать спустя время или неочевидно для другого человека.

Также использование комментариев может быть очень удобным на этапе отладки. Чтобы временно исключить из компиляции участок кода - просто закомментируйте его.

Пример однострочного комментария:

```
//Это однострочный комментарий
```

Пример многострочного комментария:

```
/*
Это
многострочный
комментарий
*/
```

Оператор точка с запятой ";" в языке C++ обозначает завершение любой команды.

Пример:

```
a = b + 1;
```

Забыв указать ";" в конце команды, вы получите ошибку на этапе компиляции.

Фигурные скобки "{ }" используются для выделения блока кода. Открывающая фигурная скобка должна обязательно

завершаться закрывающей. В противном случае возникнет ошибка компиляции. Хорошей практикой является печатание закрывающей скобки сразу же после добавления открывающей, и только потом уже добавление блока кода внутрь. Среда Arduino IDE подсвечивает пару скобок, при выборе одной из них.

Директива #include используется для добавления библиотек в ваш скетч.

Например:

#include <RTC.h>

Эта строка в начале скетча будет указывать на то, что Вы хотите использовать библиотеку (набор готовых программ) RTC.h

Конструкция #define называется макроопределением. Она говорит компилятору о том, что всякий раз, когда он видит указанное имя, он подменит на этом месте указанное значение.

#define LED_PIN 13

Теперь, когда в тексте программы встретиться надпись LED_PIN, компилятор подменит ее на 13. Это очень удобно, когда приходится писать программу и все время помнить что 13 - это порт, к которому подключен светодиод. Проще один раз указать такую подмену и писать вместо числа 13 строку LED_PIN, осознавая что к этому порту подключен светодиод. Если вдруг мы захотим пере подключить светодиод к 9 порту, то нужно только в одном месте программы 13 заменить на 9. Использование понятных имён вместо магических чисел — это один из признаков профессионализма. Это делает код более понятным и простым для изменений.

Макроопределения пишутся большими буквами с подчеркиванием вместо пробела.

Обратите внимание: макроопределения #define не завершаются точкой с запятой в конце строки.

3. Типы данных

Переменные - именованные значения, которые могут изменяться при исполнении программы микроконтроллером. Другими словами - это место, где может храниться число. Можно посмотреть что там за число, поместить туда новое число во время работы программы. Каждое место для хранения должно как-то

называться. Название этого места - имя переменной. Называть можно как угодно английскими буквами, цифрами и нижними подчеркиваниями. Например:

a
mmm
chislo1
otvet_1
OtvetZadachi

Но лучше начинать строчными буквами, а каждое новое слово писать слитно, с заглавной буквы и при этом использовать понятные, лаконичные имена из которых чётко понятно, зачем нужна конкретная переменная в программе.

Например:
otvetZadachi1
countDelayBeforeStart

Размер места, где хранится число, описывается отдельно и указывается один раз – в начале программы, когда мы объявляем эту переменную, т.е. говорим компилятору что хотим создать место с таким размером и с таким именем. Одновременно мы можем сказать компилятору, что хотим, чтобы он сразу же в нее поместил такое число.

Пример:

int otvetZadachi1;

Мы просто зарезервировали место для целого числа от -32768 до 32767 и назвали его otvetZadachi1. Если мы туда раньше ничего не помещали, то, как правило, там хранится число 0, но может храниться абсолютно любое число. Поэтому, перед тем, как просмотреть какое там число хранится, нужно туда поместить что-то определенное.

int otvetZadachi1=2;

Мы зарезервировали место для целого числа от -32768 до 32767 и назвали его otvetZadachi1. При этом поместили туда число 2.

Рассомтрим основные типы данных.

- **boolean**

Логический тип данных. Может принимать два значения: true(истина, «1») и false(ложь, «0»). Булевы переменные занимают один байт памяти. В языке C++ работает следующее правило: за

"истину" считается любое значение отличное от нуля, ноль же трактуется как "ложь".

Пример:

boolean a = true;

Создали булеву переменную с именем a. В переменную сразу же занесли значение «истина».

- **char**

Тип данных, занимающий 1 байт памяти, используемый для хранения символов или целых чисел от -128 до 127. Символы записываются в одинарных кавычках. Вместо символа автоматически подставляется его код из специальной таблицы ASCII

Пример:

char c = 'B';

Создали переменную размерностью 1 байт с именем c. В переменную сразу же занесли значение кода буквы B. Код B =66. Значит в переменной с хранится число 66.

- **int**

Целочисленный тип int - это основной тип данных для хранения чисел.

В Arduino переменные типа int хранят 16-битные (2-байтовые) значения. Такая размерность дает диапазон от -32768 до 32767.

Пример:

int ledPin = 13;

Создали переменную размерностью 2 байт с именем ledPin. В переменную сразу же занесли число 13.

- **long**

Переменные типа long обладают расширенным размером для хранения чисел и имеют размерность 32 бита (4 байта), что позволяет им хранить целые числа в диапазоне от -2 147 483 648 до 2 147 483 647.

Пример

long speedOfLight = 186000L;

Создали переменную размерностью 4 байта с именем speedOfLight. В переменную сразу же занесли число 186000.

По умолчанию, целочисленные константы интерпретируются как целые числа типа int с соответствующими предельными

значениями. Чтобы задать целочисленной константе другой тип, запишите после нее:

'u' или 'U'', чтобы привести константу к беззнаковому типу данных. Например: 33u

'l' или 'L', чтобы привести константу к типу данных long. Например: 186000L

'ul' или 'UL', чтобы привести константу к типу unsigned long. Например: 32767ul

Перед числами также могут указываться другие модификаторы. 0x говорит о том, что число после этого модификатора записано в шестнадцатеричной системе счисления.

0x0F - число, соответствующее десятичному числу 15.

Модификатор B говорит о том, что число после этого модификатора записано в двоичной системе счисления.

B101 - число, соответствующее десятичному числу 5.

Переменные типа unsigned char, unsigned int, unsigned long (беззнаковые) схожи с обычными типами переменных. Отличие состоит в том, что вместо отрицательных чисел они могут хранить только положительные целые значения в удобном диапазоне от 0 до 255 unsigned char, от 0 до 65535 unsigned int, от 0 до 4 294 967 295 unsigned long.

Для хранения вещественных чисел используются типы float и double.

Тип **float** занимает 4 байта памяти и используется для хранения чисел с плавающей точкой в диапазоне от -3.4028235E+38 до 3.4028235E+38.

Таким образом обеспечивается точность в 6-7 десятичных цифр (общее количество, а не цифры после запятой).

Пример:

```
float x = 1.1111;
float y = 2.0;
```

Будьте внимательны, работая с вещественными числами. Они не являются точными, это значит, что поделив 6.0 на 2.0 вы не обязательно получите точно 3.0. Можете получить 2.999999989. Поэтому сравнивайте результаты, учитывая погрешность. Отличительной особенностью записи этих чисел является наличие точки.

Тип **double** на arduino платах на основе контроллеров Atmega (Uno, Mega...) ничем не отличается от типа float.

Добавление модификатора const к объявлению переменной делают эту переменную константой. Если Вы попытаетесь ее случайно поменять во время работы программы- компилятор предупредит о ошибке.

const unsigned int ledPin=13;

Теперь ledPin является константой, менять ее нельзя. Действительно, зачем ее менять, если во время работы светодиод мы не собираемся переключать с 13 порта на какой-то другой порт.

Контрольные вопросы:

1. Как называется программа, написанная для платформы Arduino?
2. Введите понятие функции? Тело функции?
3. Какова структура программы для платформы Arduino?
4. На каком языке пишутся программы для Arduino?
5. Какие типы данных могут использоваться в программе?

Лабораторная работа №4

Тема: Операторы анализа условий для платформы Arduino. Порты ввода-вывода

Цель: изучить условия и операторы сравнения, циклы, аналоговые порты ввода и вывода, их назначение

1. Условия и операторы сравнения

Существуют следующие операторы сравнения:

> - больше;

< - меньше;

== - равно;

!= - не равно;

>= - больше, или равно;

<= - меньше или равно.

Для задания логики нашему приложению применяются условные конструкции, такие как **if, if..else**.

Оператор if ("если" с англ.) имеет следующую структуру:

```
if (логическое выражение)
{
//блок кода
}
```

Что же означает данная конструкция?

Конструкция if осуществляет выполнение блока кода, расположенного в следующих за ним операторных скобках {}, при истинности логического условия. В случае, когда условие принимает значение "ложь" - код расположенный в операторных скобках игнорируется. Рассмотрим более наглядный пример.

```
if (a>5)
  {
   b=3;
   c=b+1;
  }
```

Если число в переменной с именем a больше 5, то выполнится код b=3; c=b+1; а значит, в переменную b попадет число 3 и в переменную c попадет число b и еще добавится единичка, т.е. будет 4.

Если после if мы не укажем скобок, то это означает, что к оператору if попадает под действие только одна команда.

```
if (a>5) b=3;
c=b+1;
```

Такая запись означает, что b=3; выполнится только если a больше 5. А вот c=b+1; выполнится всегда, независимо от результатов сравнения переменной a с числом 5.

Опционально после if может следовать оператор else ("иначе" с англ.). Конструкция с else имеет следующий вид:

```
if (логическое выражение)
{
//блок кода
}
else
{
//блок кода
}
```

В случае ложности логического выражения, будет выполнен блок кода, идущий в операторных скобках после else.

```
if (a>5)
  {
   b=3;
   c=b+1;
  }
   else
    {
     b=4;
     c=b+2;
    }
```

Если число в переменной с именем a больше 5, то выполнится код b=3; c=b+1; а значит, в переменную b попадет число 3 и в переменную c попадет число b и еще добавится единичка, т.е. будет 4. Если же число в переменной с именем a **не** больше 5, то выполнится код b=4; c=b+2; а значит, в переменную b попадет число 4 и в переменную c попадет число b и еще добавится два, т.е. будет 6.

2. Циклы

Цикл — многократное прохождение по одному и тому же коду программы. Циклы необходимы программисту для многократного выполнения одного и того же кода, пока истинно какое-то условие. Если условие всегда истинно, то такой цикл называется бесконечным, у такого цикла нет точки выхода. Итерацией цикла называется один проход этого цикла.

При использовании цикла for (рис.17) необходимо задать три параметра (в круглых скобках через точку с запятой).

```
for(i=0; i<5; i++)
    {
       // Тело цикла
        …     }
```

Первый параметр – начальное значение переменной, задается в виде присваивания переменной значения, в нашем случае i=0.

Второй параметр – конечное значение переменной, задается в виде условия на значение переменной. Цикл будет исполняться, пока условие истинно, в нашем случае условие i<5 означает, что переменная i будет принимать значения до 5, не включая число 5.

Третий параметр – шаг изменения переменной. Запись i++ означает, что переменная i будет увеличиваться на 1 с каждым новым исполнением цикла, запись i-- – уменьшаться.

Рисунок 17 - Цикл for

Как будет выполняться этот цикл? Сначала переменной i присвоится значение 0. Потом проверится условие i<5. В нашем случае 0<5 и условие будет истинно. Значит, последовательно выполнятся все команды внутри этого цикла. После окончания выполнения тела цикла произойдет выполнение i++. Т.е. i станет равным 1. Опять проверится условие i<5. И так далее. Всего тело цикла выполнится 5 раз. При этом i будет принимать значения 0,1,2,3,4. Как только закончится пятое выполнение тела цикла i станет равной 5. Условие i<5 станет ложным и тело цикла выполняться уже не будет. Цикл будет окончен и начнут выполняться команды, следующие за ним.

Когда мы не знаем, сколько итераций должен произвести цикл, нам понадобится цикл while. Синтаксис цикла while выглядит следующим образом:

```
while (условие )
      {
        // Тело цикла.
       …
       }
```

Тут тело цикла будет выполняться пока условие будет истинно.

```
while (i<128)
      {
        i=i*2;
       }
```

3. Порты ввода-вывода

Каждый из 14 цифровых портов ввода/вывода (D0-D13) может быть портом ввода или портом вывода. Что это означает? Когда порт является портом вывода, то мы можем подавать на него напряжение или убирать его. Когда мы подаем на него напряжение, то на этом порте появляется напряжение чуть меньше 5 вольт. Причем максимально мы можем забрать ток в 40 миллиампер. Это не очень много- лампочкам и двигателям не хватит этого тока, но для светодиодов или датчиков этого тока вполне хватает. Когда мы убираем с порта напряжение, на порте оно пропадает и становится

близким к 0 вольт. В случае с подачей напряжения, мы можем говорить о высоком уровне (HIGH, или «1») на порте вывода. Если напряжение убрано – это низкий уровень напряжения(LOW, или «0»).

В случае, если порт является портом ввода, то мы можем определить есть ли там напряжение, получая в результате (HIGH, или «1») или его там нет (LOW, или «0»).

Настроить порт на ввод или вывод мы можем с помощью функции **pinMode ( ).** Функция – специальный кусочек программы, который выполняет какое-то законченное действие. Например, **pinMode ( )** настраивает порт на вход или выход.

Чтобы сконфигурировать порт номер 13 (D13) как порт вывода, в программе необходимо написать (вызвать функцию)

pinMode (13, OUTPUT);

Чтобы сконфигурировать порт номер 13 (D13) как порт ввода, необходимо вызвать функцию

pinMode (13, INPUT);

Когда порт настроен на вывод, мы можем подавать или убирать напряжение. Чтобы подать на вывод номер 13(D13) напряжение 5 вольт, нужно вызвать функцию

digitalWrite (13, HIGH);

Или, что то же самое,

digitalWrite (13, 1);

Чтобы убрать на выводе номер 13 напряжение, нужно вызвать функцию

digitalWrite (13, LOW);

или

digitalWrite (13, 0);

Как только микроконтроллер дойдет до выполнения этой строчки, почти мгновенно напряжение сразу же пропадет. Обратите

внимание, что микроконтроллер выполняет все функции и команды последовательно. Поэтому сначала всегда нужно настроить порт на ввод или вывод, а только потом подавать напряжение, убирать или проверять на нем напряжение. Если пытаться убрать напряжение с порта, на котором и так оно уже убрано, то ничего страшного не произойдет. Ничего страшного не произойдет, если подать напряжение на порт с уже поданным напряжением.

Вместо цифры 13 может быть любой порт от 0 до 13. Также могут быть использованы аналоговые входы от A0 до A5- вместо числа пишем, например, A3. A6 и A7 не могут использоваться в качестве цифровых входов или выходов.

Но почему же мы заговорили о числе 13? Да просто на плате Ардуино уже установлен светодиод, подключенный к порту 13. И если мы подаем туда напряжение- то светодиод загорится. Уберем напряжение- светодиод погаснет. Очень удобный способ проверить есть ли там напряжение.

По умолчанию все порты Arduino сконфигурированы как порты ввода. Это значит, что если вы хотите считать данные с порта номер 13, нет нужды сначала вызывать функцию

pinMode (13, INPUT);

и настраивать нужный порт на ввод.

Чтобы считать напряжение с порта вывода номер 13, нужно вызвать функцию

val=digitalRead (13);

которая возвращает число 0 или 1, в зависимости от входного напряжения, поданного в порт 13. Это число заносится в переменную с именем «val». Причем, если напряжение меньше 1 вольта- то функция вернет число 0. Если напряжение больше 2,5 вольт, то вернет число 1. Промежуточное значение от 1 до 2,5 вольт может выдать неопределенные значения, зависящие от многих условий. Поэтому для получения точных значений следует избегать подачи напряжение между 1 и 2,5 вольтами на вход.

Внимание! На вход нельзя подавать напряжение больше 5 вольт. Это приведет к выходу из строя платы Ардуино.

Обратите внимание, что в конце каждой функции стоит точка с запятой. Это признак окончания команды или функции. Если пропустить этот знак препинания, то компьютер не поймет написанную программу и выдаст ошибку.

3.1 Аналоговые порты ввода

Итак, мы уже научились подавать и считывать напряжение, равное 0 или 5 В. Но иногда требуется получить промежуточные значения напряжения, например, чтобы плавно менять обороты мотора или считывать значение с аналогового датчика освещенности, расстояния и т. д.

Внутри Ардуино находится аналогово-цифровой преобразователь (АЦП). У Ардуино 8 входов(A0-A7), к которым может подключаться этот единственный АЦП. Он измеряет напряжение на подключенном входе и переводит его в число. Напряжение может быть от 0 до 5 вольт. При этом АЦП переводит 0 вольт в число 0, а 5 вольт в число 1023. Все промежуточные значения соответствуют промежуточным числам. Например, 2,5 вольта будет соответствовать числу 511. Таким образом, весь диапазон в 5 вольт делится на 1024 кусочка. Каждый кусочек будет соответствовать 5/1024=4,88 милливольтам. Если мы получили число 300, то мы его умножаем на 4,88 и получаем 1,46 вольта. Это означает, что АЦП измерил на входе напряжение в 1,46 вольта.

Число с АЦП может быть получено с помощью вызова функции

val = analogRead(A0);

При этом в переменную «val» будет помещено число, которое вернул нам АЦП после измерения напряжения на аналоговом входе A0. Число может быть любым от 0 до 1023, в зависимости от напряжения на входе A0.

Если последовательно читать напряжение с разных входов, то необходимо делать задержку перед каждым чтением в 10-40мкс для обеспечения корректности полученных данных.

3.2 Аналоговые порты вывода

Напрямую получать напряжение с портов Ардуино не получится. Для этого нужен узел, который называется цифро-аналоговый преобразователь (ЦАП). Он превращает записанное в него число в напряжение по тому же принципу, что и АЦП.

Узел ЦАП есть только в Arduino Due, но и его тяжело назвать полноценным - выдает напряжение только от 0,6 вольт до 2,4 вольт. Т.е. от 0 вольт он не работает. Остальные Ардуино не могут похвастаться наличием даже такого ЦАПа. Но не беда - есть много способов обойти это препятствие. Один из способов - это широтно-импульсная модуляция (ШИМ, PWM) - веселая штука, и особенно забавно с ее помощью управлять сервомоторами и трехцветным светодиодом. Это позволит управлять его цветом и яркостью.

ШИМ - это управление средним значением напряжения на нагрузке путём изменения скважности импульсов, подаваемых на нагрузку. В ШИМ частота импульсного (прямоугольного) сигнала постоянная, а скважность (отношение периода следования импульса к его длительности) переменная. С помощью изменения скважности импульсов можно менять среднее напряжение на выходе ШИМ.

Теперь разберемся, что это значит. Пусть есть обычный такой прямоугольный сигнал:

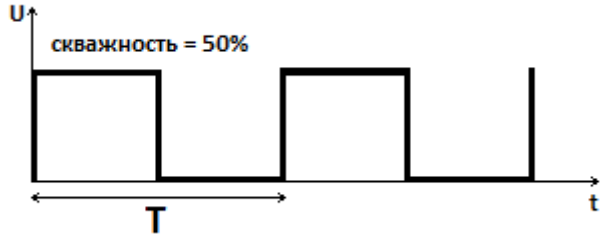

Рисунок 18 - ШИМ заполнением 50 процентов

Он имеет фиксированную частоту и заполнение 50% (рис.18). Это означает, что половину периода напряжение максимально, а другую половину оно равно нулю. Проинтегрировав этот сигнал за период, мы увидим, что его энергия равна половине максимальной. Это будет эквивалентно тому, как если бы мы все время подавали половину напряжения.

Так как напряжение пропадает и появляется на очень короткое время, то человеческий глаз не заметит мерцаний подключенного светодиода, а просто усреднит его яркость.

Если у нас максимальное напряжение равно 5 вольт, то напряжение, получаемое на выходе ШИМ равно заполнение умножить на 5 вольт и делить на 100%.

Arduino позволяет записать на ШИМ - выход значение от 0 до 255, а это значит, что мы можем получить напряжение с шагом примерно 20 милливольт – от 0 до 5 вольт (рис.19).

Частота ШИМ на Arduino - примерно 490 Гц. На плате Arduino Nano выводы, которые могут быть использованы для ШИМ - 3,5,6, 9, 10 и 11. На плате к этому может быть подсказка - шелкографией перед номерами ШИМ - выводов есть тильда или диез.

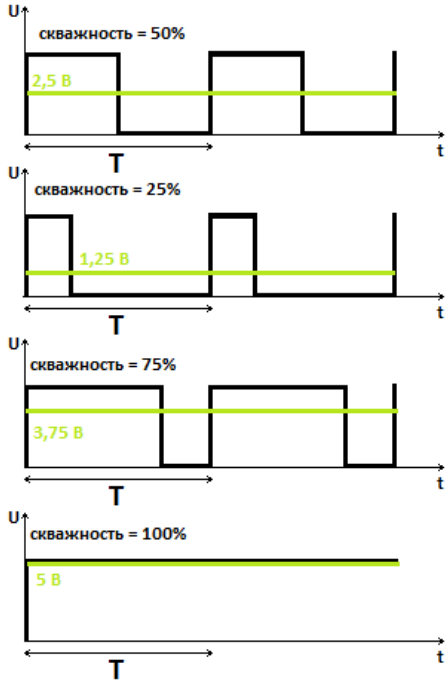

Рисунок 19 - ШИМ с разным заполнением

Для управления ШИМ на Arduino используется одна единственная функция

analogWrite(3, 255);

где 3 - номер вывода(3,5,6, 9, 10 и 11), а 255 значение для ШИМ - любое от 0 до 255. При этом порт можно предварительно не настраивать на выход.

На сконфигурированном однажды выходе будет постоянно присутствовать сигнал ШИМ частотой 490 Гц с заданными параметрами до следующего вызова функции analogWrite() (или digitalRead(), или digitalWrite()) для этого же вывода микроконтроллера.

Одновременно можно формировать ШИМ для всех указанных выходов.

Жаль, конечно, что нельзя управлять 13 выводом, к которому уже подключен светодиод. Так бы удалось управлять его яркостью от выключенного состояния при значении ШИМ 0 до максимальной яркости при значении ШИМ 255.

Контрольные вопросы:

1. Перечислите существующие операторы сравнения.
2. Какие операторы анализа условий применяются в программировании для Arduino?
3. Поясните синтаксис оператора if – else.
4. Введите понятие цикла.
5. Какие циклы применяются при программировании для Arduino?
6. Поясните синтаксис цикла for и while.
7. Аналоговые порты ввода и вывода.

Лабораторная работа №5

Тема: Запуск готовых скетчей. Blink

Цель: изучить порядок запуска и компиляции готовых скетчей, изменение скетча

1. Порядок запуска готовых скетчей

Попробуем загрузить скетч из образцов, разобрать его и запустить.

Для начала нужно подключить Arduino Nano к компьютеру. К плате расширения пока подключать Ардуино не обязательно. На Ардуино должен загореться светодиод питания. После этого запускаем arduino.exe (рис.20).

Выбираем «Файл->Образцы->Basics->Blink»

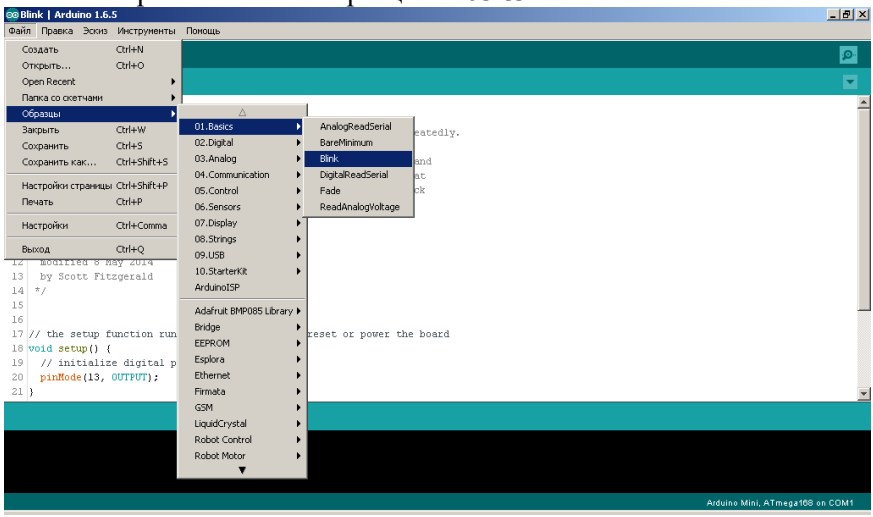

Рисунок 20 - Выбор примера

Откроется скетч в новом окне.

Приведем полностью его текст:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
```

```
  Most Arduinos have an on-board LED you can control. On the Uno and
  Leonardo, it is attached to digital pin 13. If you're unsure what
  pin the on-board LED is connected to on your Arduino model, check
  the documentation at http://www.arduino.cc
  This example code is in the public domain.
  modified 8 May 2014
  by Scott Fitzgerald
 */
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

В начале многострочный комментарий, который содержит поясняющую информацию. Далее идет однострочный комментарий.

Потом начинается функция setup(). В теле этой функции выполняется одна единственная команда:

pinMode(13, OUTPUT);

которая настраивает 13 порт на выход. Не забываем, что к 13 порту на плате Arduino Nano подключен светодиод.

После выполнения этой функции начинает выполняться функция loop(), которая содержит в своем теле всего 4 команды:

digitalWrite(13, HIGH); подает напряжение на 13 порт;

delay(1000); организует паузу в 1000миллисекунд=1 секунду;

digitalWrite(13, LOW); убирает напряжение с порта 13;

delay(1000); организует паузу в 1000миллисекунд=1 секунду.

После того, как отрабатывает последняя команда, функция loop() начинает свою работу заново.

Получается, что данная программа включает светодиод, через 1 секунду его выключает, ждет секунду и всё повторяется заново. Вот простая «Мигалочка». Теперь попробуем откомпилировать программу и залить ее в Ардуино. Для этого нажимаем на панели управления кнопочку «Вгрузить» (рис.21).

Происходит компиляция программы.

Компиляция - это процесс преобразования программного кода с языка верхнего уровня, в бинарный код, который будет выполнять микроконтроллер. Если скетч написан без синтаксических ошибок, он успешно будет скомпилирован и залит на arduino. Если возникла проблема загрузки на плату, проверьте еще раз подключена ли плата, установлены ли на нее драйвера в диспетчере устройств и попробуйте выбрать другой COM порт. Перед заливкой нового скетча старый скетч внутри микроконтроллера очищается автоматически.

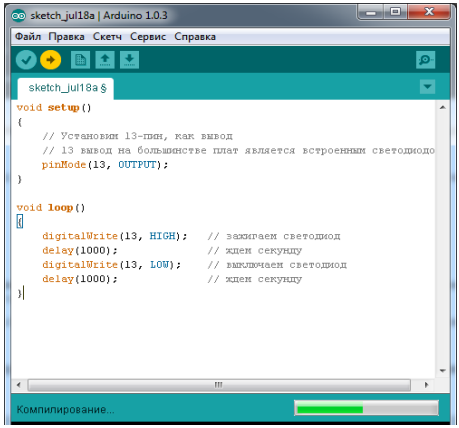

Рисунок 21 - Компилируем и вгружаем

Если загрузка прошла удачно, Ардуино будет подмигивать Вам светодиодиком раз в секунду (рис.22).

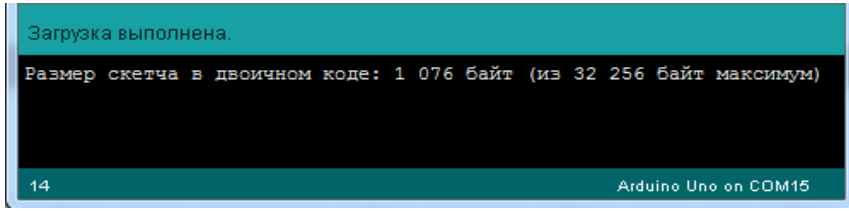

Рисунок 22 - Удачная загрузка

Измените скетч таким образом, чтобы светодиод мигал в 2 раза чаще.

Контрольные вопросы:

1. Поясните порядок запуска готовых скетчей.
2. Изучите листинг программы и поясните каждую строку скетча.
3. Что такое компиляция программы?

Лабораторная работа №6

Тема: Беспаечная макетная плата. Электронные компоненты, используемые при работе с платой

Цель: изучить особенности беспаечной макетной платы, правила её использования, правила подготовки её к работе; изучить описание платы расширения NN500 и электронных компонентов, используемых при работе с платой

1. Особенности беспаечных макетных плат

Для налаживания и тестирования самодельных электронных устройств используются макетные платы. Применение макетной платы позволяет проверить, наладить и протестировать схему ещё до того, как устройство будет собрано на готовой печатной плате. Это позволяет избежать ошибок при конструировании, а также быстро внести изменения в разрабатываемую схему и тут же проверить результат. Макетная плата экономит много времени и является очень полезной.

Прогресс и развитие электроники также затронул и макетные платы. В настоящее время можно без особых проблем приобрести беспаечную макетную плату. Главное достоинство беспаечной монтажной платы – это отсутствие процесса пайки при макетировании схемы. Это обстоятельство значительно сокращает процесс макетирования и отладки устройств. Собрать схему на беспаечной монтажной плате можно буквально за пару минут.

Беспаечная макетная плата состоит из пластмассового основания, в котором имеется набор токопроводящих контактных разъёмов. Этих контактных разъёмов очень много. В зависимости от конструкции макетной платы контактные разъёмы объединяются в строки, например, по 5. В результате образуется пятиконтактный разъём. Каждый из разъёмов позволяет подключать к нему выводы электронных компонентов или токопроводящих проводников диаметром, как правило, не более 0,7 мм.

На рисунке 23 показана беспаечная макетная плата **EIC-402** для монтажа без пайки на 840 точек. Таким образом, данная макетная плата содержит 840 контактных разъёмов.

Основа макетной платы – ABS пластик. Контактные разъёмы выполнены из фосфористой бронзы и покрыты никелем. Благодаря

этому, контактные разъёмы (точки) рассчитаны на 50 000 циклов подключения/отключения. Контактные разъёмы позволяют подключать выводы радиодеталей и проводники диаметром от 0,4 до 0,7 мм.

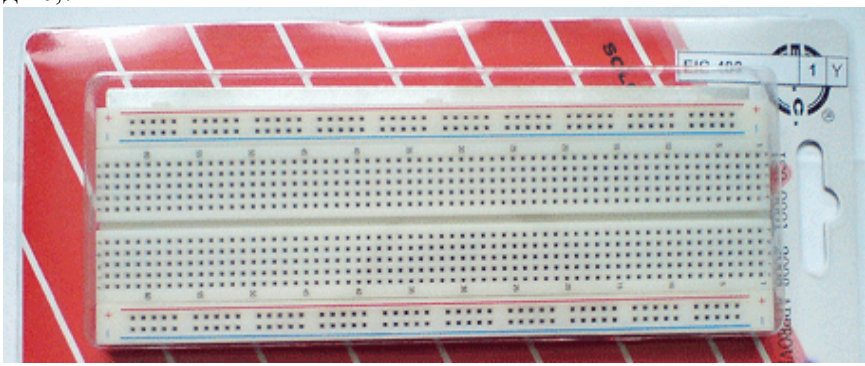

Рисунок 23 - Беспаечная макетная плата EIC-402

На рисунке 24 показана отладочная плата для микроконтроллеров, собранная на беспаечной макетной плате.

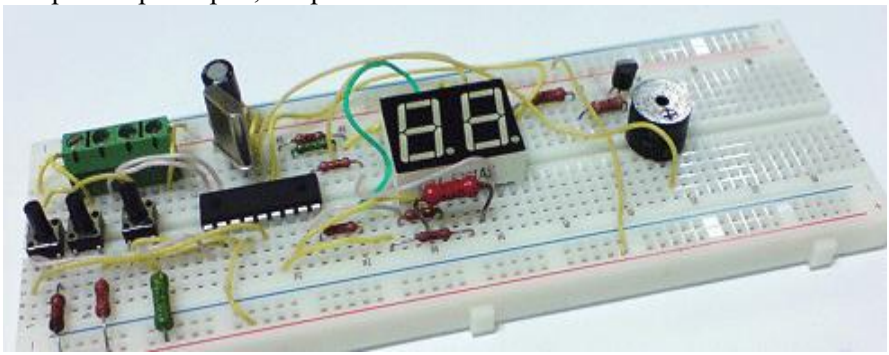

Рисунок 24 - Отладочная плата для микроконтроллеров

Как видим (рис.25), беспаечная макетная плата позволяет устанавливать резисторы, конденсаторы, микросхемы, светодиоды и индикаторы.

С помощью беспаечной макетной платы изучение электроники превращается в увлекательный процесс. Принципиальные схемы собираются на плате без лишнего труда. В зависимости от сложности беспаечной макетной платы она может комплектоваться

набором соединительных проводников (проводов-джамперов), дополнительных разъёмов и пр.

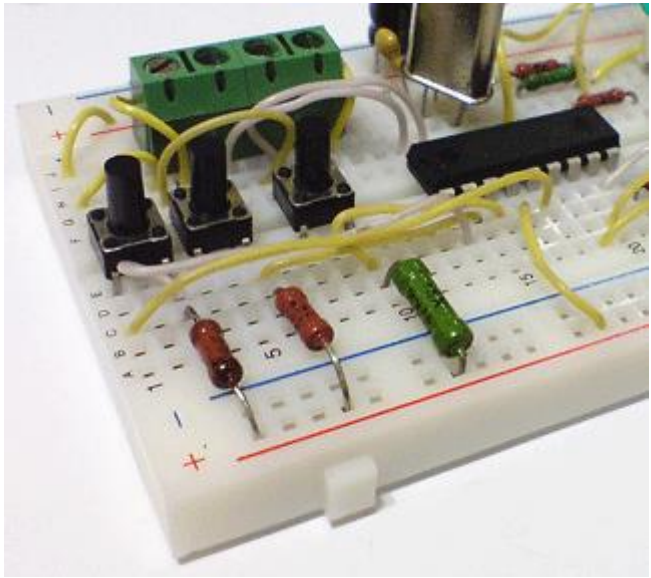

Рисунок 25 – Собранная макетная плата

Несмотря на все дополнительные элементы основным показателем качества беспаечной макетной платы всё же является качество контактных разъёмов и их количество. Тут всё понятно, чем больше контактных точек (разъёмов), тем более сложную схему можно смонтировать на такой плате. Качество разъёмов также важно, ведь от частого использования разъёмы могут потерять свои упругие свойства, а это в будущем приведёт к плохому качеству контакта.

Правила использования беспаечных макетных плат:

– Поскольку разъёмы макетной платы позволяют подключать проводники диаметром не более 0,4-0,7 мм, то попытки «затолкнуть» толстые выводы деталей могут привести лишь к порче контакта. В таком случае к выводам радиоэлементов, имеющим достаточно большой диаметр, например, как у мощных диодов, лучше припаять или намотать провод меньшего диаметра и уже тогда подключать элемент к макетной плате.

– Если планируется макетирование достаточно сложной схемы с большим количеством элементов, то площади беспаечной макетной платы может и не хватить. В таком случае схему лучше разделить на блоки, каждый из которых нужно собрать на отдельной макетной плате и затем соединить блоки в единое устройство с помощью соединительных проводников. Понятно, что в таком случае понадобится дополнительная макетная плата.

– Как правило, макетная плата с набором соединительных проводников разной длины (проводов-джамперов) стоит дороже обычных беспаечных плат, которые такими проводниками не комплектуются. Но это не беда. В качестве соединительных проводников можно использовать и обычный провод в изоляции.

Например, для таких целей весьма распространённый и доступный по цене провод **КСВВ 4х0,4**, который используется для монтажа охранно-пожарной сигнализации. Этот провод имеет 4 жилы, каждая из которых покрыта изоляцией. Диаметр самой медной жилы без учёта изоляции составляет 0,4 мм. Изоляция с такого провода легко снимается кусачками, а медный провод не покрыт лаковым покрытием.

–Из одного метра такого кабеля можно наделать множество соединительных проводников разной длины. Кстати, на фотографиях макетной платы, показанных выше, для соединения радиодеталей использовался как раз провод КСВВ.

–Макетную плату следует оберегать от пыли. Если плата долгое время не используется, то на её поверхности оседает пыль, которая забивает контактные разъёмы. В дальнейшем это приведёт к плохому контакту и плату придётся чистить.

–**Беспаечные макетные платы не предназначены для работы с напряжением 220 вольт!** Также стоит понимать, что макетирование и проверка работы сильноточных схем на беспаечной макетной плате может привести к перегреву контактных разъёмов.

Экранирование макетной платы

Обилие соединительных проводников и сама конструкция макетной платы при работе собранного устройства провоцирует так называемые «паразитные связи». По-простому их называют «наводками» или помехами. Эти помехи отрицательно влияют на работу схемы, собранной на плате. Чтобы избежать этого общий

провод (GND) схемы электрически соединяют с металлической подложкой. Сама подложка закрепляется на нижней части беспаечной макетной платы.

2. Подготовка беспаечной макетной платы к работе

Перед тем, как начать макетировать схему на новой беспаечной макетной плате не лишним будет «прозвонить» контактные разъёмы мультиметром. Это нужно для того, чтобы узнать, какие точки-разъёмы соединены между собой.

Дело в том, что точки (разъёмы) на макетной плате соединены на макетной плате особым образом (рис.26). Так, например, беспаечная макетная плата EIC-402 имеет 4 независимые контактные зоны. Две по краям – это шины питания (плюсовая «+» и минусовая «-»), они маркированы красной и синей линией вдоль контактных точек. Все точки шины электрически соединены между собой и, по сути представляют собой один проводник, но с кучей точек-разъёмов.

Рисунок 26 - Соединение точек на макетной плате

Центральная область разделена на две части. Посередине эти две части разделяет своеобразная канавка. В каждой части 64 строки по 5 точек-разъёмов в каждой. Эти 5 точек-разъёмов в строке электрически соединены между собой. Таким образом, если

установить, например, микросхему в корпусе DIP-8 или DIP-18 по центру макетной платы, то к каждому её выводу можно подключить либо 4 вывода радиоэлементов, либо 4 соединительных проводника-джампера.

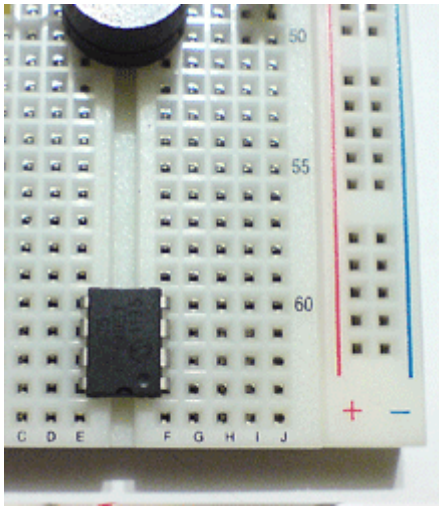

Рисунок 27– Пример использования макетной платы

Макетную плату для монтажа без пайки удобно использовать для быстрой сборки измерительных схем. Пример использования макетной платы на рисунке 27.

3. Описание платы расширения NN500

Для стандартизации лабораторных работ на беспаечной макетной плате можно собрать следующую схему платы расширения NN500 (рис.28).

Плата расширения еще называется Шилд (от англ. Shield). Подключая к ней Arduino Nano можно значительно расширить функционал Ардуино платы и легко подсоединить к ней множество датчиков и исполнительных устройств. Всё это поможет обучиться основам программирования Ардуино, быстро попробовать возникающие идеи и в конечном итоге создать множество

полезных устройств. Среди них: электронные часы, метеостанция, система управления теплым полом.

Шилд обладает множеством разъемов, к которым можно подключить большое число модулей, датчиков и различных исполнительных устройств.

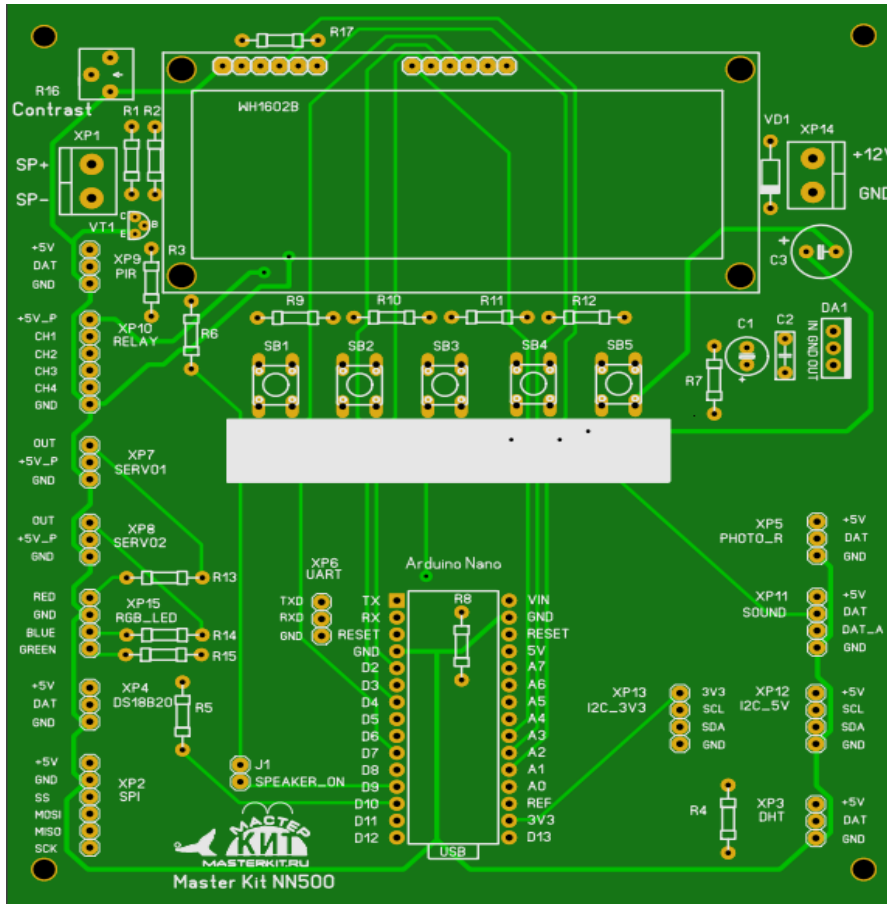

Рисунок 28 - Плата расширения NN500

Сердцем шилда является плата Arduino Nano. На плате расширения уже установлен двухстрочный жидкокристаллический дисплей, клавиатура с пятью кнопками, усилитель звука для подключения к плате мощного динамика и дополнительный

источник питания чтобы питать потребляющие много энергии узлы- блок реле и две сервомашинки. Заметьте, что без подключения внешнего источника питания к разъему XP14 блок реле и сервомашинки работать не будут. Вся остальная плата может обходиться только одним питанием по USB. Для этого USB шнур не обязательно подключать к компьютеру- можно просто подключить его к зарядному устройству.

Разъем питания защищен от неправильного подключения питания, поэтому переполюсовка плате не страшна.

Принципиальная схема платы расширения приведена на рисунке 29.

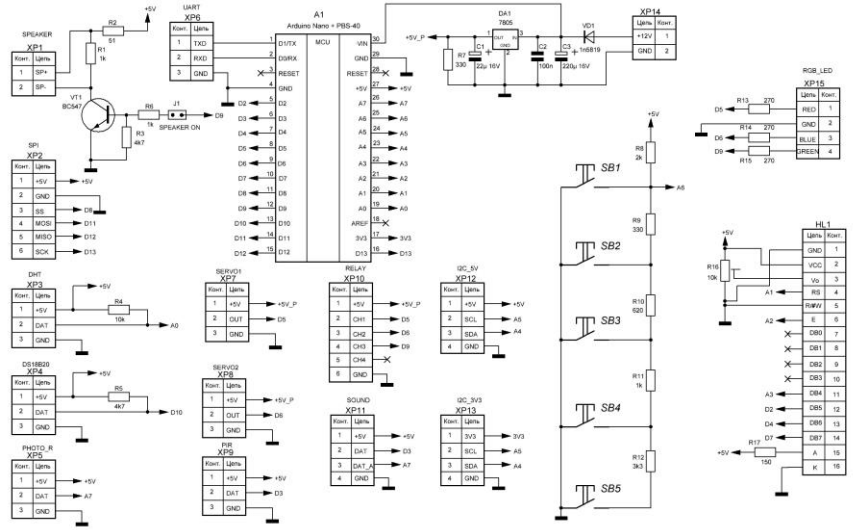

Рисунок 29 - Схема платы расширения NN500

На транзисторе VT1 построен усилитель тока, который может быть использован для подключения динамика (рис.30) или светодиода (рис.31). Резистор R6 ограничивает базовый ток транзистора, а резистор R3 обеспечивает надежное запирание транзистора, если соответствующий управляющий цифровой порт D9 не настроен на выход. Резистор R1 является нагрузкой этого транзистора и одновременно используется в качестве разрядного резистора при подключении к разъему XP1 пьезокерамического излучателя. Резистор R2 ограничивает ток через транзистор до

безопасного уровня. Перемычка J1 используется для отключения этого узла от порта D9. Если не отключать порт и использовать его в качестве линии ввода, то транзистор может помешать выдавать датчику правильные уровни напряжения.

Рисунок 30- Динамик

Резисторы R4 и R5 используются в качестве подтяжки линии данных к 5 вольтам. Это нужно для стабильной работы датчика температуры DS18B20 и датчика влажности DHT11. Резисторы R13, R14, R15 являются токоограничительными для подключения RGB светодиода с общим катодом к разъему XP15.

Рисунок 31- RGB светодиод

Самый длинный вывод - это общий катод, а все остальные - это аноды, каждый отвечает за свой цвет: (смотрим на рисунок 31) самый нижний - красный, второй сверху - зеленый, самый верхний - синий.

На микросхеме DA1 собран стабилизатор напряжения, преобразующий входное напряжение 7,5-12 вольт в стабильное 5

вольт. Диод VD1 служит защитой от переполюсовки внешнего источника питания. Конденсаторы C1, C2, C3 необходимы для стабильной работы стабилизатора.

На кнопках SB1-SB5 и резисторах R8-R12 собрана хитрая аналоговая клавиатура. Такое решение позволяет сэкономить множество линий платы Ардуино и задействовать один аналоговый вход A6. Рассмотрим теорию работы такой клавиатуры.

Резистивный делитель напряжения представляет собой два резистора, включенных последовательно друг другу и параллельно источнику питания.

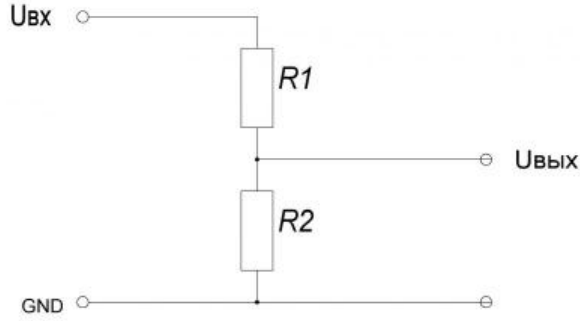

Рисунок 32 - Делитель напряжения

Данный делитель рассчитывается по формуле

$$Uвых=Uвх*R2/(R2+R1)$$

Пример расчета:

Примем Uвх = 5в , R1 = 1кОм, R2 = 1кОм.

5*1/(1+1)=2,5вольта

На выходе делителя 2,5 вольта. Хотя на входе - 5 вольт.

Резисторы делителя называют плечами. Верхнее плечо подключено к Uвх, нижнее - к GND. Т.е. R1 – верхнее плечо, R2 – нижнее плечо.

Если номиналы обоих резисторов равны напряжение будет поделено пополам. Важно знать, что общее сопротивление делителя должно быть значительно меньше сопротивления нагрузки подключенной к нему (примерно в 100 раз). Как раз с нагрузкой порта Arduino настроенного на вход проблем никаких нет- это десятки мегаом. Но общее сопротивление делителя не

должно быть и слишком низким. В этом случае мы получим нагрев делителя и расход тока впустую. Общее сопротивление не должно быть ниже 4,7кОм. В общем, идеальный диапазон сопротивлений от 4,7кОм до 50кОм

Теперь рассмотрим ЦАП (Цифро-аналоговый преобразователь) на основе делителя напряжения. Устройство преобразует цифровой сигнал в аналоговый.

На данной схеме изображен простой двух канальный резистивный ЦАП.

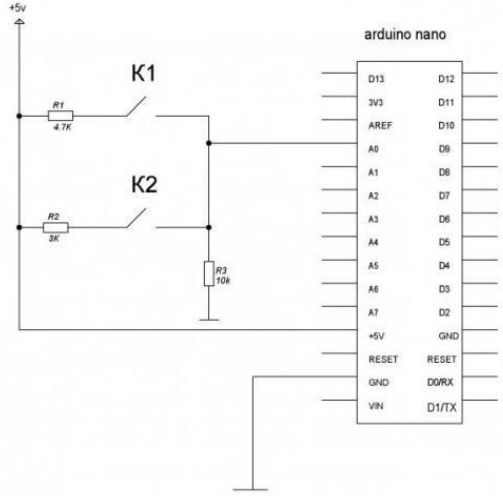

Рисунок 33- Подключение аналоговой клавиатуры

Рассмотрим некоторые важные моменты.

Резистор R3 является подтягивающим и его номинал не стоит опускать ниже 4,7кОм. В то же время R3 является нижним плечом делителя напряжения.

Когда кнопки не нажаты на аналоговом порте АЦП – 0 вольт. Нажатая кнопка К1 образует делитель напряжения с номиналами плеч R1=4,7кОм, R2=10кОм. Считаем по формуле.

5*10/(4.7+10) = 3.4вольта

Т.е. при нажатой кнопке К1 на аналоговом порте 3,4 вольта.

Когда нажата кнопка К2 номиналы делителя будут R1=3кОм , R2=10кОм.

5*10/(3+10) = 3.8вольт.

Т.е. при нажатой кнопке К2 на аналоговом порте 3,8 вольта.

Но что произойдет, если обе кнопки будут нажаты одновременно?

В этом случае верхнее плечо делителя будет состоять из двух резисторов R1 и R2 включенных параллельно. Общий номинал параллельно включенных резисторов считаем по формуле

R=R1*R2/(R1+R2)

Rобщ = (4,7*3)/(4,7+3) = 1,83кОм

Считаем выходное напряжение делителя

5*10/(1.83+10) = 4,22 вольта.

Т.е., если обе кнопки будут нажаты одновременно на аналоговом порте 4,22 вольта.

Эти разные значения напряжения программа легко может измерить и решить о том, какая кнопка или кнопки были нажаты. Такой тип клавиатуры часто используется внутри телевизоров.

Количество каналов ЦАПа может быть сколь угодно большим (и соответственно количество кнопок). Нужно только точно рассчитывать номиналы резисторов. И уделять особое внимание соотношениям номиналов между собой.

К Ардуино на плате расширения подключен также двухстрочный жидкокристаллический индикатор (ЖКИ). ЖК дисплеи (символьные) очень популярны и используются практически во многих проектах с микроконтроллерами (рис.34). С помощью них можно выводить текст, значения и псевдографику. Он позволяет выводить сообщения в 2 строки по 16 символов. Внутри дисплея встроен специальный контроллер, который формирует изображение. Нам остается ему только сказать какие символы куда выводить. Для экономии портов он подключен по 4-битной шине данных, но это уменьшает скорость передачи в 2раза.

Рисунок 34- Двухстрочный ЖКИ

Контрастность регулируется подстроечным резистором R16. Прокрутите его так, чтобы стали видны символы, выводимые на дисплей.

Резистор R17 ограничивает ток подсветки дисплея.

К разъему XP2 может подключаться любой модуль со SPI интерфейсом. Например, модуль реального времени, часы, индикатор, SD карта.

К разъему XP3 подключается модуль измерения влажности DHT11 MP590 производства «Мастеркит» (рис.35).

Рисунок 35- MP590

К разъему XP4 подключается модуль измерения температуры DS18B20. Например, MP18B20 производства «Мастеркит» (рис36).

Рисунок 36- MP18B20

К разъему XP5 подключается модуль фотодатчика. Например, MP 594 производства «Мастеркит» (рис.37).

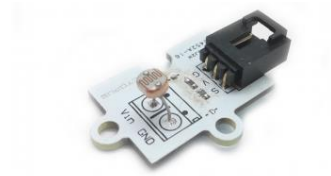

Рисунок 37- MP594

К разъему XP6 подключается модуль с UART интерфейсом-USB кабель, GPS модуль, GSM модуль, BT модуль.

К разъемам XP7 и XP8 подключаются сервомашинки.

К разъему XP9 подключается модуль датчика движения. Например, MP559 производства «Мастеркит» (рис.38).

Рисунок 38- MP559

К разъему XP10 подключается блок реле. Например, MP4411 производства «Мастеркит» (рис.39).

Рисунок 39- MP4411

К разъему XP11 подключается модуль датчика звука. Например, MP546 производства «Мастеркит» (рис.40).

Рисунок 40 - MP546

К разъему XP12 подключается модуль с шиной I2C и 5-ти вольтовым питанием. Например, модуль реального времени MP1095 производства «Мастеркит» (рис.41).

Рисунок 41 - MP1095

К разъему XP13 подключается модуль с шиной I2C и 3-х вольтовым питанием. Например, модуль барометра MP591 производства «Мастеркит» (рис.42).

Рисунок 42 - MP591

Контрольные вопросы:

1. Для чего используются макетные платы?
2. Правила использования беспаечной макетной платы.
3. Для чего призводят экранирование макетной платы?
4. Каковы особенности подготовки беспаечной макетной платы к работе?
5. Опишите плату расширения NN500.
6. Поясните принципиальную схему платы расширения NN500.
7. Для чего применяется резистивный делитель напряжения? Формула расчёта?
8. Какие электронные компоненты используются при работе с платой?

Лабораторная работа №7

Тема: Сборка платы расширения NN500. Подключение двустрочного символьного жидкокристаллического дисплея

Цель: изучить особенности сборки платы расширения NN500, порядок сборки, возможные неполадки, подключение двустрочного символьного жидкокристаллического дисплея и его скетч.

1. Порядок сборки платы расширения NN500

Плату расширения NN500 необходимо предварительно собрать.

Для работы потребуется:

- паяльник мощностью не более 40Вт;
- припой марки ПОС-61М или его аналог и жидкий неактивный флюс для радиомонтажных работ (например, 30% раствор канифоли в этиловом спирте, ЛТИ-120).

Порядок сборки:

1. Отформуйте выводы компонентов и установите их в соответствии с маркировкой на печатной плате.
2. Установите компоненты на печатные платы. При установке электролитических конденсаторов, диодов и микросхем требуется соблюдать полярность и соответствие маркировки на платах.
3. Все компоненты устанавливаются на печатную плату со стороны маркировки.
4. Обрежьте выводы.
5. Припаяйте все компоненты.
6. Запаяйте на плату специальные разъемы для индикатора и Arduino Nano.
7. Внимание! Время пайки одного контакта не более 3 секунд. Это предотвратит отслаивание токопроводящих дорожек и перегрев элементов.
8. Очистите платы от остатков флюса с помощью отрезка бинта и изопропилового спирта (Осторожно! Яд!) или, в крайнем случае, медицинского этилового спирта (обладает большей гигроскопичностью и возможен белесый налет на плате).
9. Проверьте правильность монтажа.

10. Установите джампер J1 для использования с платой внешнего динамика.
11. Убедитесь, что все детали установлены на нужные позиции печатной платы. Проверьте соответствие «ключей» индикатора и платы Arduino Nano.

Рекомендуем во время работы проветривать помещение и мыть руки после работы.

Если собранное устройство не работает:

1. Проверьте правильность монтажа:
   – Убедитесь, что все детали установлены на нужные позиции печатной платы.
   – Проверьте соответствие «ключей» диодов, микросхем, транзисторов, индикатора и Arduino Nano.
2. Проверьте правильность пайки:
   – Убедитесь, что все точки пайки надёжно припаяны.
   – Убедитесь, что в процессе пайки не возникло паразитных перемычек между токоведущими дорожками, при обнаружении, аккуратно удалите их паяльником.
3. Проверьте правильность подключения питания и шнур USB.

2. Подключение двустрочного символьного жидкокристаллического дисплея и его скетч

На рисунке 43 показан жидкокристаллический индикатор (далее ЖКИ, LCD-англ.) WH1602B-TMI-CT фирмы Winstar. LCD дисплеи от компании Winstar уже на протяжении нескольких лет являются неотъемлемой частью современной электронной продукции, и не только на российском рынке. Они дешевы и очень распространены.

Рисунок 43 – ЖКИ

У дисплея 16 выводных линий, из которых 11 – линии управления, расположеных в ряд с шагом 2,54мм. Два вывода(15 и

16) используются для подсветки. Дисплеи линейки WH построены на базе специализированного контроллера LCD-модулей HD44780, который как раз и разрабатывался для управления знакосинтезирующими ЖК-панелями. Всю сложную работу по формированию и хранению изображения выполняет этот самый контроллер. Нам же, остается, только подключившись к дисплею, передавать ему команды- какие символы отображать. Сами символы уже «зашиты» внутрь контроллера дисплея и рисовать нам их не нужно. Просто объяснить дисплею, какой именно символ следует отображать.

ЖКИ позволяет выводить 2 строки текста по 16 символов в каждой строке. Отображаемые символы-белые на синем фоне. Дисплей поддерживает отображение символов кириллицы. Однако, сама кириллица не поддерживается средой Arduino IDE. Поэтому, для отображения русских букв нам придется пойти на некоторые ухищрения.

Дисплей может работать в 2 режимах: в режиме 8-битной передачи данных, когда данные передаются группами по 8 бит (при этом обеспечивается максимальная скорость взаимодействия с дисплеем), и в режиме 4-битной передачи, когда 8-битные данные разбиваются на две группы по четыре разряда и последовательно передаются по четырем старшим линиям данных. В последнем случае используется всего лишь 6 линий для подключения к микроконтроллеру. Нас как раз и устраивает такой вариант экономии портов ввода-вывода Arduino.

Так как на плате расширения NN500 предусмотрено подключение дисплея, идущего в комплекте, то нам достаточно просто подключить его и заняться программой.

Рассмотрим скетч HelloWord из набора скетчей-примеров для платы расширения NN500.

Подключаем плату расширения с установленной Arduino Nano к компьютеру. Если не были установлены драйвера для Arduino Nano, то устанавливаем их, как уже было описано выше. Запускаем Arduino IDE. В меню «Инструменты» выбираем плату «Arduino Nano» и порт, соответствующей нашей плате. Распаковываем содержимое файла со скетчами для платы расширения NN500 на любой диск. В меню «Файл» выбираем подпункт «Открыть…» и

открываем скетч HelloWord. В новом окне появится программный код скетча.

Подробно рассмотрим код программы.

Для того, чтобы легко работать с дисплеем нам нужно подключить готовую библиотеку дисплея к нашей программе:

```
// Обязательно подключаем стандартную библиотеку LiquidCrystal
#include <LiquidCrystal.h>
```

Так как в языке С++, на котором написаны многие библиотеки применяется объектно-ориентированный подход, то нам придется сталкиваться с понятием объекта определенного типа.

Вот и сейчас мы создадим в программе объект индикатора

```
// Создаем объект дисплей lcd, объясняя программе, куда подключены линии RS,EN,DB4,DB5,DB6,DB7
LiquidCrystal lcd(A1, A2, A3, 2, 4, 7);
```

Одновременно при создании объекта мы указали программе, к каким портам ввода-вывода подключены соответствующие линии индикатора. Например, линия RS подключена к порту с именем A1. Линия EN индикатора к порту Arduino с именем A2. Для данной платы расширения NN500 подключение неизменно. Но если Вы захотите отдельно подключить индикатор к другим портам ввода-вывода платы Arduino, эту строчку придется переделать.

Далее нам необходимо настроить индикатор, или инициализировать. Делать это нужно 1 раз перед его использованием. Поэтому в функции setup(), которая выполняется в самом начале программы один раз, пишем строку:

```
// Инициализируем LCD как обычно -16 символов и 2 строки
lcd.begin(16, 2);
```

Т.е. мы указали программе, что дисплей у нас состоит из 2 строк по 16 символов в каждой. Верхняя строка нумеруется 0 (в программировании нумерация всегда начинается не с единицы, а с нуля.) Нижняя строка нумеруется единичкой. Первый символ слева каждой строки имеет нулевой номер, а самый правый-пятнадцатый. Всего шестнадцать.

После инициализации мы можем печатать с того места, где находится курсор на дисплее. А он находится на верхней– нулевой строке и первом слева символе- нулевом. Т.е. с координатами (0,0).

Для того, чтобы что-то написать на дисплее, нам нужно просто указать что мы будем писать.

```
// И напишем на дисплее Hello!
lcd.print("Hello!");
```

Сейчас мы вывел на дисплей символы, которые указали в кавычках.

На дисплее будет красоваться надпись Hello!

Просто подождем пару секунд, чтобы насладиться надписью и нашим успехом:

```
// Выдержим паузу в 2000 миллисекунд= 2 секунды
delay(2000);
```

Во время выполнения этой функции программа «зависает» ровно на 2 секунды. Потом продолжает работу, как ни в чем не бывало.

На этом заканчивается выполнение функции setup()

После этого начнет выполняться функция loop() , которая будет бесконечно повторяться, пока мы не отключим питание от платы.

Далее:

```
// Заводим переменную с именем randomNum
long randomNum;
```

В эту переменную мы будем помещать случайное число.

```
// Записываем в эту переменную случайное число от 0 до (256-1) включительно
randomNum = random(256);
```

Функция random() создает случайное число от 0 до того числа, которое мы указали в скобках. Правда, не включая само число. Полученное число записываем в переменную randomNum.

Далее очистим дисплей:

```
// Очищаем дисплей от всех надписей
lcd.clear();
```

Все, что было написано на дисплее, сотрется и курсор установится опять в позицию с координатами (0,0). Но мы можем и принудительно выставить курсор на нужную нам позицию, чтобы текст печатался с нужного места:

```
// Переведем курсор на седьмой слева символ (нумерация начинается с 0) и нижнюю строку (нумерация начинается с 0)
lcd.setCursor(6, 1);
```

Сейчас мы выведем наше случайное число, которое содержится в переменной randomNum.

```
// И напишем чему равна наша переменная со случайным числом в этот раз
lcd.print(randomNum);
```

Подождем одну секундочку:

```
// Выдержим паузу в 1000 миллисекунд= 1 секунду
delay(1000);
```

На этом месте заканчивается выполнение функции loop(). И сразу же она начнется сначала: опять создаем случайное число, очищаем экран, двигаем курсор и печатаем число, ждем секунду.

Нажмем на кнопку панели управления «Вгрузить». Через несколько секунд скетч в виде программы окажется внутри платы. На экране мы увидим надпись Hello!, через 2 секунды она пропадет и на второй строке с периодом в 1 секунду будут появляться случайные числа.

Вывод русских символов на дисплее

Если на дисплей требуется выводить кириллицу, надо решить две проблемы:

1. Arduino IDE по причине своей Java-природы транслирует русские буквы в кодировку UTF-8 в процессе компиляции скетча, в то время как дисплей понимает только однобайтовые символы и понятие не имеет, что такое схемы кодирования юникода.

2. Китайские товарищи добавили в знакогенератор дисплея только набор дополнительных кириллических символов, никак не позаботившись о соблюдении вообще какой-либо кодировки. То есть, символ "Б" есть, а вот букву "А" надо использовать из английского набора. Наверное, они рассуждали так: "напихаем побольше разных символов, а эти славяне со своей письменностью сами разберутся".

Самое простое решение - добавить библиотеку, которая будет решать обе проблемы хитрым образом. Но разработчики Arduino пока не решали эту проблему.

Есть и другой путь- дисплею можно «нарисовать» 8 любых символов и дополнительно использовать английские большие символы. В этом случае, можно писать по-русски даже на дисплеях без поддержки кириллицы. Именно это осуществляет библиотека WolfCrystal.

К сожалению, в разных версиях Arduino IDE много внутренних отличий. И данная библиотека не работает на версиях Arduino IDE 1.6.x. Для того, чтобы использовать эту библиотеку, понадобиться Arduino IDE 1.5.x.

Распакуйте содержимое архива libraries.zip в папку «Мои документы»-> «Arduino»-> «libraries». Перезапустите Arduino IDE и откройте скетч HelloWorldRus.

Не забудьте подключить плату и выбрать ее тип и порт.

В первых строках делаем уже знакомые нам вещи:

```
// Обязательно подключаем стандартную библиотеку LiquidCrystal
#include <LiquidCrystal.h>
```

Подключаем новую библиотеку:

```
// Подключаем библиотеку WolfCrystal
#include <WolfCrystal.h>
```

Создаем объект дисплея:

```
LiquidCrystal lcd(A1, A2, A3, 2, 4, 7);
```

И тут же создаем объект библиотеки WolfCrystal, подсовывая ему в качестве исходных данных ссылку на уже созданный объект дисплея:

```
// Присоединяем дисплей lcd к библиотеке WolfCrystal.
WolfCrystal WC(&lcd);
```

Всю основную работу на этот раз выполним внутри функции setup():

```
  // Инициализируем LCD как обычно -16 символов и 2 строки
  lcd.begin(16, 2);
  // Курсор находится на первой строке (верхней) и первом слева символе
```

Далее пробуем напечатать русский текст. Для этого передадим текст сначала библиотеке WolfCrystal, а она уже вернет функции lcd.print нужный нам текст:

```
  // Для того, чтобы использовать русский, применяем вставку
  lcd.print( WC.GS("Привет, мир!") );
```

Напечатанный текст будет немного отличаться от написанного: вместо строчных букв будут прописные. Ведь библиотека старается выкрутиться, организовав надпись из очень узкого набора символов. Но зато- кириллица!!!

Далее переведем курсор на вторую строку и напечатаем там текст латиницей:

```
// Переведем курсор на первый слева символ (нумерация начинается с 0) и вторую строку (нумерация начинается с 0)
lcd.setCursor(0, 1);
  // И напишем латиницей
lcd.print("rulez!");
```

На этом функция setup() заканчивается. Осталась функция loop(). Но она пуста. Это означает, что в программе будет бесконечно выполняться пустая функция loop(). А значит, ничего не будет происходить после однократного выполнения функции setup().

После того, как нажмем кнопку «Вгрузить», на дисплее отобразиться 2 надписи. Одна на кириллице, вторая на латинице.

Контрольные вопросы:

1. Поясните порядок сборки платы расширения NN500.
2. Каковы причины возможных неполадок платы?
3. Поясните правила подключения двустрочного символьного жидкокристаллического индикатора.
4. Опишите код программы - скетч для работы с дисплеем.
5. Как использовать русские символы на ЖК-дисплее?

Лабораторная работа №8

Тема: Использование аналоговой клавиатуры

Цель: изучение программной поддержки аналоговой клавиатуры

1. Программная поддержка аналоговой клавиатуры

Ранее мы рассмотрели принцип работы аналоговой клавиатуры, которая реализована в плате расширения NN500. Сейчас рассмотрим программную поддержку данной клавиатуры.

Открываем скетч Keyboard.

Так как в этом скетче используем дисплей, то будем использовать уже знакомые нам элементы программы.

```
// Обязательно подключаем стандартную библиотеку LiquidCrystal
#include <LiquidCrystal.h>
```

После этого для клавиатуры обязательно указать некоторые калибровочные значения. Они уже тщательно рассчитаны и подобраны, поэтому остается их просто копировать для этой платы:

```
// Определяем сколько кнопок у нас подключено
#define NUM_KEYS 5
// Для каждой кнопки заносим калибровочные значения(выведены экспериментально)
int  adcKeyVal[NUM_KEYS] = {30, 150, 360, 535, 760};
```

Возвращаемся к созданию объекта дисплея:

```
// Инициализируем дисплей, объясняя программе, куда подключены линии RS,EN,DB4,DB5,DB6,DB7
LiquidCrystal lcd(A1, A2, A3, 2, 4, 7);
```

В функции setup() настраиваем дисплей:

```
// Инициализируем LCD как обычно -16 символов и 2 строки
  lcd.begin(16, 2);
  // Курсор находится на первой строке (верхней) и первом слева символе
```

Далее немного красоты:

```
  // И напишем на дисплее Keyboard
  lcd.print("Keyboard");
   // Выдержим паузу в 2000 миллисекунд= 2 секунды
  delay(2000);
```

Следующий код уже входит в функцию loop() и будет бесконечно повторяться:

```
// Заводим переменную с именем key
int key;
```

В самом низу, ниже функции loop мы описали свою функцию, которую назвали get_key(). Эта функция возвращает число, которое равно номеру нажатой кнопки. Если кнопки не нажаты - возвращается число 0. При нажатии левой кнопки - вернется число 1. Правой - число 5.

В заведенную переменную key помещаем число, соответствующее нажатой кнопке:

```
// Записываем в эту переменную номер нажатой кнопки, вызывая на исполнение нижеописанную функцию get_key() :
key = get_key();
```

Очищаем дисплей:

```
// Очищаем дисплей от всех надписей
lcd.clear();
// Курсор находится на первой строке (верхней) и первом слева символе
```

Выводим номер нажатой кнопки:

```
// И напишем какую кнопку нажали, 0 - ни одна кнопка не нажата
lcd.print(key);
```

Ждем 0,1 секунду. Потом функция loop() начинает выполняться сначала.

Код функции get_key() мы расписывать не будем. Как начинающего в программировании, возможно, он Вас совсем запутает. Эта функция будет выполнена, только когда ее вызвали из программы. Функция читает значение с АЦП, куда подключена аналоговая клавиатура и сравнивает с калибровочными значениями, определяя номер нажатой кнопки.

Для использования в проекте клавиатуры просто скопируйте эту функцию вместе с калибровочными настройками.

Контрольные вопросы:

1. Поясните программный код для аналоговой клавиатуры.
2. Назовите назначение функции get_key().
3. Перечислите особенности данного скетча.

Лабораторная работа №9

Тема: Изменение цвета RGB светодиода

Цель: изучение программы изменения цвета RGB светодиода

1. Изменение цвета RGB светодиода

Сделаем так, чтобы светодиод переливался разными цветами. Пусть один цвет плавно гаснет, в то время как другой разгорается. Поочередно будем менять пару цветов, и цвет будет переходить по кругу из красного в зеленый, из зеленого в синий, из синего в красный.

Мы уже знаем, что такое ШИМ и как использовать аналоговый вывод. Также мы уже разобрали пример RGB светодиода. Подключите его к плате расширения NN500 к разъему XP15. В Arduino IDE откройте скетч RGBLed.

Вначале заводим переменные, указывая к каким портам ввода-вывода подключены цвета светодиода. Для NN500 - это определенные конструктивом значения.

```
//обзываем выводы соответственно цвету
int redPin = 5;
int greenPin = 9;
int bluePin = 6;
```

Функция setup() на этот раз пустая.

В функции loop() создаем три цикла. В каждом цикле работаем только с двумя цветами. Третий цвет не горит.

Так, в первом цикле for мы увеличиваем яркость зеленого от 0 до максимума 255. При этом уменьшаем яркость красного от максимума до 0.

Во втором цикле уже зеленый уменьшаем, увеличиваем синий. А в третьем уменьшаем синий, увеличиваем красный.

Итак, первый цикл:

```
  for(int value = 0 ; value <= 255; value ++) {
    //красный не горит
    analogWrite(redPin, 0);
    //яркость зеленого уменьшается от максимума к 0
    analogWrite(greenPin, 255-value);
    //яркость синего увеличивается
    analogWrite(bluePin, value);
```

```
    // Выдержим паузу в 30 миллисекунд
    delay(30);
  }
```

Переменная value каждый проход тела цикла будет увеличиваться от 0 до 255. Когда пройдет 256 проходов цикла, цикл перестанет выполняться и начнется выполнение следующего цикла.

В теле этого цикла мы выключаем красный, выдавая на вывод красного светодиода ШИМ равный нулю:

```
analogWrite(redPin, 0);
```

К зеленому светодиоду мы подаем ШИМ 255-value=0 при первом проходе:

```
analogWrite(greenPin, 255-value);
```

В результате зеленый светодиод будет ярко светится. При последующих проходах значение value будет расти и, соответственно значение ШИМ будет уменьшаться 254, 253 … 0. А значит и яркость зеленого светодиода будет падать.

Значение ШИМ синего светодиода, напротив, будет расти 0,1,2…255.

```
analogWrite(bluePin, value);
```

После этого зафиксируем текущее состояние на 30 миллисекунд и тело цикла начнет выполняться сначала.

```
    // Выдержим паузу в 30 миллисекунд
    delay(30);
```

Аналогично работают второй и третий циклы, только цвета светодиодов другие.

Контрольные вопросы:

1. Поясните программный код изменения цвета светодиода.
2. Назовите назначение функции loop().
3. Перечислите особенности данного скетча.

Лабораторная работа №10

Тема: Программные часы

Цель: выполнение программного проекта - часы

1. Программные часы.

Arduino позволяет сделать множество простых и интересных проектов. Один из таких полезных проектов-обыкновенные часы. Вся особенность этой программы в том, что мы не будем использовать электронные модули часов и сами попытаемся прописать всю логику работы часов.

В проекте будет использоваться индикатор, клавиатура и NN500 с установленной Arduino Nano.

Открываем скетч Clock.

Вставляем уже известный нам код - подключение библиотеки дисплея и калибровочные значения для клавиатуры:

```
// Обязательно подключаем стандартную библиотеку LiquidCrystal
#include <LiquidCrystal.h>
// Определяем сколько кнопок у нас подключено
#define NUM_KEYS 5
// Для каждой кнопки заносим калибровочные значения(выведены экспериментально)
int adcKeyVal[NUM_KEYS] = {30, 150, 360, 535, 760};
```

Заводим переменные, которые нам понадобятся для программы. В первых двух переменных мы будем хранить число миллисекунд с момента запуска программы на Arduino. В остальных переменных будем хранить минуты, часы и секунды:

```
// Заводим переменные для хранения числа миллисекунд с момента наступления разных событий
unsigned long curMillis , lastTick;
// Заводим переменные, в которых будут храниться секунды, минуты и часы
int sec, mins, hour;
```

Далее создаем объект дисплея:

```
// Инициализируем дисплей, объясняя программе куда подключены линии RS,EN,DB4,DB5,DB6,DB7
LiquidCrystal lcd(A1, A2, A3, 2, 4, 7);
```

В функции setup() настраиваем дисплей:

```
// Инициализируем LCD как обычно -16 символов и 2 строки
lcd.begin(16, 2);
```

Основной код будем выполнять в функции loop():

```
// Записываем в переменную cur_millis  число миллисекунд, прошедших с момента запуска программы Arduino
curMillis  = millis();
```

Функция millis() возвращает число миллисекунд, прошедших с момента запуска программы на Arduino. Это число записываем в переменную curMillis.

Дальше сравниваем число пройденных миллисекунд с прошлым запомненным значением. Так как в момент включения прошлое запомненное значение lastTick еще равно 0, то, как только их разница будет больше 1000 миллисекунд=1секунде, это будет означать, что у нас наступила новая секунда.

```
// Если с прошлым полученным значением миллисекунд разница теперешнего более 1 секунды, то, значит, наступила новая секунда
if (curMillis - lastTick >= 1000) {
```

Ну а раз наступила новая секунда, то нам нужно увеличить число секунд и скорректировать минуты и часы. Но прежде мы запишем в прошлое запомненное значение текущее значение пройденных миллисекунд:

```
// Текущее значение прошедших миллисекунд копируется в переменную с прошлым значением
lastTick = curMillis;
```

В момент включения curMillis и lastTick равны 0. Но переменная curMillis каждый раз обновляется при выполнении прошлой строки

```
curMillis  = millis();
```

Предположим, что curMillis в какой-то момент после старта программы стала равна 1001 миллисекунде. Так как

```
1001 - 0 >= 1000
1001>=1000
```

То условие в if становится истинным. И это означает, что пошла новая секунда. Теперь в lastTick вместо 0 будет 1001 миллисекунда и следующая секунда будет, когда в curMillis окажется 2001 миллисекунда.

В блоке команд if выполняем следующие команды:

```
// Текущее значение прошедших миллисекунд копируется в переменную с прошлым значением
lastTick = curMillis;
// Увеличиваем число секунд на 1
sec++;
```

Запись «переменная++» означает, что мы увеличиваем переменную на единицу. «Переменная—«, соответственно, уменьшаем на единицу.

Так как мы увеличили число секунд на 1, то у нас может получиться число 60. Такого числа в секундах быть не может, а значит, мы должны подкорректировать минуты и секунды:

```
// Если у нас стало после этого 60 и более секунд- то это следующая минута
if (sec >= 60) {
// Добавляем минуту
mins++;
// Обнуляем секунды
sec = 0;
}
```

Та же ситуация с минутами. Корректируем их:

```
// Если у нас стало после этого 60 и более минут- то это следующий час
if (mins >=60) {
// Добавляем 1 час
hour++;
// Обнуляем минуты
mins = 0;
}
```

Та же ситуация с часами. Корректируем их:

```
// Если у нас стало после этого 24 и более часов- то это следующий день
if (hour >=24) {
// Обнуляем часы
hour=0;
// Обнуляем минуты
mins = 0;
}
```

Выведем надпись:

```
// Устанавливаем курсор в левую верхнюю позицию
lcd.setCursor(0,0);
// Выводим надпись
lcd.print("Clock v1.0");
```

// Устанавливаем курсор на вторую слева позицию на нижнюю строку

```
lcd.setCursor(1,1);
```

А теперь мы выводим часы, минуты и секунды через двоеточие. Но при этом применим небольшую хитрость. Мы все привыкли, что часы выводятся в формате 02:07:59. Только вот lcd.print нам напишет число часов «2» вместо «02». Поэтому незначащий ноль слева нам придется самим пририсовать:

```
// Если часов меньше 10, то перед цифрой добавляем нолик
if (hour < 10) {lcd.print("0");}
// Выводим число часов
lcd.print(hour);
// Выводим двоеточие
lcd.print(":");
// Если минут меньше 10, то перед цифрой добавляем нолик
if (mins < 10) {lcd.print("0");}
// Выводим число минут
lcd.print(mins);
// Выводим двоеточие
lcd.print(":");
// Если секунд меньше 10, то перед цифрой добавляем нолик
if (sec < 10) {lcd.print("0");}
// Выводим число секунд
lcd.print(sec);
```

На этом действия в блоке if наступления новой секунды окончены.

Нам осталось корректировать часы с кнопок. Если нажали первую кнопку, то увеличиваем часы:

// Если вызванная функция get_key() вернула нам число 1(была нажата кнопка 1)

```
if (get_key()==1) {
// Увеличиваем часы на 1
hour++;
```

```
 // Пауза перед следующим увеличением в 0,3 секунды
delay(300);
}
```

Если нажали вторую кнопку, то увеличиваем минуты:

```
// Если вызванная функция get_key() вернула нам число 2(была нажата кнопка 2)
if (get_key()==2) {
// Увеличиваем минуты на 1
mins++;
// Обнуляем секунды
sec=0;
// Пауза перед следующим увеличением в 0,3 секунды
delay(300);
}
```

Все эти действия будут постоянно повторяться в функции loop().

Контрольные вопросы:

1. Поясните код программных часов.
2. Назовите назначение функции loop(), millis().
3. Перечислите особенности данного скетча.

Лабораторная работа №11

Тема: Датчик влажности и температуры. Барометр. Метеостанция

Цель: изучение программы датчика влажности и температуры, разработка проекта метеостанции

1. Датчик влажности и температуры

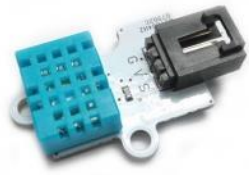

Рисунок 44 - Модуль MP590 с датчиком DHT11

DHT11 недорогой цифровой датчик температуры и влажности (рис.44). Он использует емкостной датчик влажности и терморезистор для измерения температуры окружающего воздуха, данные выдает в цифровой форме по шине типа 1-wire. В использовании он довольно прост.

Для преобразования данных внутри датчика используется микроконтроллер. В процессе производства датчики калибруются и калибровочная константа записывается вместе с программой в память микроконтроллера. Однопроводной последовательный интерфейс дает возможность быстрой интеграции в устройство. Его небольшие размеры, низкое энергопотребление и до-20-метров передачи сигнала, что делает его привлекательным выбором для различных приложений.

Датчик измеряет следующие параметры:

Влажность: 20-90%±5%

Температура: 0-50°C ±2°C

Разрешение влажности в 1 процент и температуры в 1 градус. Датчик достаточно медленный, и отклик на изменение реальных условий у него может достигать 30 секунд.

Питание DHT11 составляет 3-5.5V DC. После подачи питания на датчик, необходимо выдержать паузу длительностью не менее 1 секунды перед началом считывания данных. Весь обмен данными

выполняется по одному проводу (шине). На шине может присутствовать только один датчик. Для получения высокого уровня используется подтягивающий резистор (5-10 кОм- уже установлен на плате NN500), т.е в пассивном состоянии на шине высокий уровень. Чтобы не вникать в тонкости работы этого датчика, для Arduino разработана библиотека DHT. Всё что нужно, это подключить датчик к разъему XP3 и установить библиотеку, распаковав содержимое архива libraries.zip в папку «Мои документы»-> «Arduino»-> «libraries» перед запуском Arduino IDE.

Откройте скетч примера DHT11.

Вначале подключаем библиотеку индикатора и библиотеку датчика:

```
// Обязательно подключаем стандартную библиотеку LiquidCrystal
#include <LiquidCrystal.h>
// Подключаем библиотеку датчика DHT
#include "DHT.h"
```

Затем указываем определенным образом тип датчика и порт подключения линии данных этого датчика:

```
// Указываем к какому порту подключен датчик
#define DHTPIN A0
// Указываем тип датчика
#define DHTTYPE DHT11
```

После этого создаем объекты индикатора и датчика:

```
// Создаем программный объект датчика DHT
DHT dht(DHTPIN, DHTTYPE);
// Создаем программный объект дисплей lcd, объясняя программе куда подключены линии RS,EN,DB4,DB5,DB6,DB7
LiquidCrystal lcd(A1, A2, A3, 2, 4, 7);
```

В функции setup() инициализируем индикатор и датчик DHT:

```
  // Инициализируем LCD как обычно -16 символов и 2 строки
  lcd.begin(16, 2);
  // Специальная функция инициализации датчика DHT
  dht.begin();
```

В функции loop() заводим переменные humDht и tDht, куда сразу же записываем считанные с датчика значения влажности и температуры, соответственно:

int humDht = dht.readHumidity(); // Считываем в переменную humDht значение влажности с датчика DHT

int tDht = dht.readTemperature(); // Считываем в переменную tDht значение температуры с датчика DHT

Дальше просто выводим на дисплей текст и значение влажности из переменной humDht:

```
// Выводим на дисплей текст
  lcd.print("H:");
// Выводим на дисплей значение переменной humDht
  lcd.print(humDht);
```

Дальше просто выводим на дисплей текст и значение температуры из переменной tDht:

```
// Выводим на дисплей текст
  lcd.print("% T:");
// Выводим на дисплей значение переменной tDht
  lcd.print(tDht);
// Выводим на дисплей текст
  lcd.print("*C");
```

Чтение влажности и температуры начинается заново при каждом запуске функции loop().

Как видим, с датчиком очень легко работать.

2. Барометр

Рисунок 45 - Модуль MP591 с датчиком BMP085

BMP085 – датчик абсолютного атмосферного давления (рис.45). Область применения: измерение давления для барометров, метеостанций и приборов, перемещающихся в атмосфере.

Был разработан фирмой Bosch и общается по шине I2C. Помимо давления он так же выдает температуру с точностью до десятой доли градуса.

Характеристики BMP085:

– Пределы измерения абсолютного давления 30-110кПа (-500…9000 метров над уровнем моря)

– Питание 1.8 – 3.6 вольт

– Размер корпуса: 5.0X5.0 мм.

– Разрешение: 1Па, 0.1 С

Датчик откалиброван. Имеет термостабилизацию. Логика работы сходна с прочими цифровыми датчиками. С памяти датчика считываются калибровочные коэффициенты, показания АЦП сенсора давления и температуры, после чего по специальной формуле вычисляются значение давления и температуры. Всю эту процедуру за нас выполняет библиотека Adafruit_BMP085. Нам остается только получить конечные значения давления и температуры.

Для увеличения эффективности, упрощения схемотехнических решений, Philips разработала простую двунаправленную двухпроводную шину для так называемого "межмикросхемного" (inter-IC) управления.

Шина получила название - InterIC, или IIC (I2C) шина.

I2C шина является одной из модификаций последовательных протоколов обмена данных. В стандартном режиме обеспечивается передача последовательных 8-битных данных со скоростью до 100 кбит/с, и до 400 кбит/с в "быстром" режиме. Для осуществления процесса обмена информацией по I2C шине, используется всего два сигнала: линия данных SDA и линия синхронизации SCL.

Простая двухпроводная последовательная шина I2C минимизирует количество соединения между микросхемами. Микросхемы и узлы имеют меньше контактов, и требуется меньше дорожек. Как результат - печатные платы становятся более простыми и технологичными при изготовлении.

Каждое устройство распознается по уникальному адресу и может работать как передатчик или приёмник, в зависимости от назначения устройства.

Нам остается подключить модуль к разъему XP13, установить библиотеку(если еще этого не успели сделать), распаковав содержимое архива libraries.zip в папку «Мои документы»-> «Arduino»-> «libraries» перед запуском Arduino IDE.

Откройте скетч DHT11_BMP085.

Он сделан на основе предыдущего скетча, поэтому рассмотрим только добавления, касающиеся нового датчика.

Вначале добавим две связанные с ним библиотеки:

```
// Подключаем библиотеку последовательной шины I2C
#include <Wire.h>
// Подключаем библиотеку датчика давления BMP085
#include <Adafruit_BMP085.h>
```

Затем создаем объект датчика:

```
// Создаем программный объект датчика BMP
Adafruit_BMP085 bmp;
```

В функции setup() добавим инициализацию барометра:

```
  // Специальная функция инициализации датчика BMP085
  bmp.begin();
```

В функции loop() заводим переменные tBmp и pBmp. Считываем с датчика температуру и давление и заносим соответственно, в эти переменные. Считанное давление умножаем предварительно на 0,0075 для перевода его в миллиметры ртутного столба.

```
// Считываем в переменную tBMP значение температуры с датчика BMP
  float tBmp=bmp.readTemperature();
// Считываем в переменную pBMP значение атмосферного давления с датчика BMP и одновременно, умножая на 0.0075, переводим его из паскалей в мм рт.ст.
  int pBmp=bmp.readPressure()*0.0075;
```

Ну и добавляем вывод на дисплей полученной температуры и давления от датчика BMP085:

```
// Переведем курсор на первый слева символ (нумерация начинается с 0) и вторую строку (нумерация начинается с 0)
  lcd.setCursor(0, 1);
// Выводим на дисплей текст
  lcd.print("T:");
// Выводим на дисплей значение переменной tBmp с одним знаком после запятой
  lcd.print(tBmp,1);
// Выводим на дисплей текст
  lcd.print(" P:");
// Выводим на дисплей значение переменной pBmp
```

lcd.print(pBmp);

Чтение влажности- температуры с DHT11 и температуры-давления с BMP085 начинается заново при каждом запуске функции loop().

3. Датчик температуры DS18B20

Рисунок 46 - Модуль MP18B20 с защищенным датчиком DS18B20

DS18B20 производится фирмой «Dallas Semiconductor» и позволяет измерять температуру в диапазоне -55...+125С с точностью 0.5 градуса (рис.46). Датчик откалиброван на заводе, поэтому показания его очень точны. Время преобразования температуры занимает около секунды.

Внутри датчика – сложная схема с сенсором, АЦП, ПЗУ, регистрами хранения и системой последовательного вывода.

Датчик температуры DS18B20 работает по протоколу передачи данных 1-Wire® и позволяет подключить несколько датчиков на одну шину.

Шина 1-Wire привлекательна, в первую очередь, тем, что использует только одну линию связи. Для работы по этому интерфейсу должно быть одно ведущее устройство и одно или несколько ведомых. У каждого 1-Wire устройства есть 64-битный уникальный код. Используя который, ведущий определяет, с каким из устройств на линии он будет работать.

Итак, чтобы работать с любым устройством на шине 1-Wire, нам нужно инициализировать его, послать ему команду и принять какие-то данные.

Датчики подключаются двумя вариантами:

- внешнее подключение (по трем проводам);
- паразитное подключение (по двум проводам).

Мы можем использовать любой вариант подключения датчика к разъему XP4. Необходимо также установить библиотеку(если еще этого не успели сделать), распаковав содержимое архива libraries.zip в папку «Мои документы»-> «Arduino»-> «libraries» перед запуском Arduino IDE.

Откройте скетч DHT11_BMP085_DS18B20.

Он сделан на основе предыдущего скетча, поэтому рассмотрим только добавления, касающиеся нового датчика.

Вначале добавим две связанные с ним библиотеки:

```
// Подключаем библиотеку шины 1-Wire
#include <OneWire.h>
// Подключаем библиотеку датчика DS18B20
#include <DallasTemperature.h>
```

Укажем, к какому порту ввода-вывода подключен датчик:

```
// Указываем, к какому порту подключен датчик DS18B20
#define ONE_WIRE_BUS 10
```

Добавляем еще два программных объекта:

```
// Создаем программный объект шины 1-Wire
OneWire oneWire(ONE_WIRE_BUS);
```

// Создаем программный объект датчика DS18B20 на основе объекта датчика 1-Wire.

```
DallasTemperature sensors(&oneWire);
```

В функцию setup() добавляем инициализацию датчика:

```
  // Специальная функция инициализации датчика DS18B20
  sensors.begin();
```

В функции loop() заводим переменную tDs. Запускаем специальной командой процедуру измерения температуры. Затем считываем с датчика температуру и заносим в эту переменную:

```
// Отсылаем датчику DS18B20 команду на измерение температуры
  sensors.requestTemperatures();
// Считываем в переменную tDs значение температуры с датчика DS18B20
  float tDs=sensors.getTempCByIndex(0);
```

Ну и добавляем вывод на дисплей полученной температуры от датчика DS18B20:

```
// Выводим на дисплей текст
  lcd.print("% To:");
// Выводим на дисплей значение переменной tDs с одним знаком после запятой
  lcd.print(tDs,1);
```

Заметьте, что мы убрали вывод температуры с датчика влажности ввиду ее низкой точности и оставили две температуры Ti- внутренней температуры с барометра и To –внешней температуры с DS18B20.

Чтение влажности с DHT11, температуры- давления с BMP085 и температуры с DS18B20 начинается заново при каждом запуске функции loop().

4. Проектирование метеостанции

Добавляем RTC и легким движением руки NN500 превращается в метеостанцию.

Рисунок 47 - Модуль RTC MP1095

Часы реального времени (англ. Real Time Clock, RTC) — электронная схема, предназначенная для учёта текущего времени, даты, дня недели и др.), представляет собой систему из автономного источника питания и специального счетчика (рис.47). Чаще всего часы реального времени встречаются в вычислительных машинах, хотя на самом деле RTC присутствует практически во всех электронных устройствах, которые должны хранить время. В отличии от прошлой программной версии часов, эти часы идут автономно от собственного источника питания. Нам остается только к ним обращаться, чтобы узнать сколько сейчас времени.

Модуль использует уже знакомую нам шину I2C. От батарейки может «идти» в течении нескольких лет. Часы идут абсолютно автономно. Мы можем устанавливать время и дату либо считывать их.

Для удобства необходимо использовать библиотеку RTClib.

Подключите модуль RTC (естественно, что остальные модули совсем не следует отключать) к разъему XP12.

Необходимо еще установить библиотеку(если еще этого не успели сделать), распаковав содержимое архива libraries.zip в папку «Мои документы»-> «Arduino»-> «libraries» перед запуском Arduino IDE.

Откройте скетч DHT11_BMP085_DS18B20_RTC.

Он сделан на основе предыдущего скетча, поэтому рассмотрим только добавления, касающиеся RTC.

Вначале добавим связанные с ним библиотеки:

```
// Подключаем библиотеку RTC
#include "RTClib.h"
```

Так как RTC использует шину I2C, то необходимо было бы еще подключить библиотеку Wire, но она у нас и так подключена для барометра.

В этом проекте мы будем использовать клавиатуру, поэтому добавляем уже знакомые строки:

```
// Определяем сколько кнопок у нас подключено
#define NUM_KEYS 5
// Для каждой кнопки заносим калибровочные значения(выведены экспериментально)
int  adcKeyVal[NUM_KEYS] = {30, 150, 360, 535, 760};
```

Не забываем создать объект часов:

```
// Создаем программный объект RTC
RTC_DS1307 RTC;
```

В функцию setup() добавляем инициализацию RTC:

```
// Специальная функция инициализации RTC
  RTC.begin();
```

А вот тут мы сделаем хитрый ход. Постараемся сделать так, чтобы в часы было записано время с компьютера. Делать это нужно командой

```
RTC.adjust(DateTime(__DATE__, __TIME__));
```

Вместо __DATE__ и __TIME Arduino IDE проставит текущее время и дату на момент формирования программы и в таком виде они отправятся внутрь нашей платы. Каждый раз при старте в часы писалось бы одно и то же время- когда была создана программа. Естественно, это очень неудобно. Поэтому мы добавляем

синхронизацию часов с компьютеров, только если зажата в момент старта программы метеостанции левая кнопка на Arduino IDE. Т.е. Когда мы нажимаем «Вгрузить», необходимо зажать левую кнопку на плате NN500. Только тогда время часов и компьютера будет синхронизировано. Добавим проверку нажатия кнопки:

```
// Если вызванная функция get_key() вернула нам число 1(была нажата кнопка 1), записываем в часы текущее время компьютера на момент компиляции программы
if (get_key()==1) RTC.adjust(DateTime(__DATE__, __TIME__));
```

В функции loop() запрашиваем время у RTC:

```
// Запускаем механизм чтения времени с RTC
DateTime now = RTC.now();
```

Время будет помещено в специальную структуру данных. Теперь мы должны с нее отдельно забрать год, месяц, день, часы, минуты и секунды:

```
// Считываем в переменную year значение текущего года и вычитаем 2000, чтобы получить две последние цифры
int year=now.year()-2000;
// Считываем в переменную month значение текущего месяца
int month=now.month();
// Считываем в переменную day значение текущего дня
int day=now.day();
// Считываем в переменную hour значение текущего часа
int hour=now.hour();
// Считываем в переменную minute значение текущей минуты
int minute=now.minute();
// Считываем в переменную second значение текущей секунды
int second=now.second();
```

Теперь нам осталось вывести на дисплей дату и время:

```
// Очищаем дисплей от всех надписей
lcd.clear();
// Курсор находится на первой строке (верхней) и первом слева символе
// Выводим на дисплей значение переменной year
lcd.print(year);
// Выводим на дисплей текст
lcd.print('/');
```

```
// Выводим на дисплей значение переменной month
  lcd.print(month);
// Выводим на дисплей текст
  lcd.print('/');
// Выводим на дисплей значение переменной day
  lcd.print(day);
// Выводим на дисплей текст
  lcd.print(' ');
// Выводим на дисплей значение переменной hour
  lcd.print(hour);
// Выводим на дисплей текст
  lcd.print(':');
// Если число минут меньше 10, то на экране дорисовываем перед минутами нолик, например 09
  if (minute<10) lcd.print('0');
// Выводим на дисплей значение переменной minute
  lcd.print(minute);
// Выводим на дисплей текст
  lcd.print(':');
// Если число секунд меньше 10, то на экране дорисовываем перед секундами нолик, например 00
  if (second<10) lcd.print('0');
// Выводим на дисплей значение переменной second
  lcd.print(second);
```

Видим уже знакомые ухищрения по поводу дорисовывания незначащих нулей слева.

В следующем блоке выводим влажность, две температуры: снаружи- «o» и внутри -«i» и давление «p».

Чтение влажности с DHT11, температуры- давления с BMP085, температуры с DS18B20 и времени с RTC начинается заново при каждом запуске функции loop().

Таким образом, у нас получилась метеостанция. Возможности ее можно значительно расширить, используя различные модули «Мастеркит» и добавляя программный код.

Контрольные вопросы:

1. Поясните программный код датчика влажности и температуры.
2. Поясните программный код барометра.
3. Как преобразовать полученный код в метеостанцию.
4. Перечислите особенности данного скетча.

Лабораторная работа №12

Тема: Возможные проблемы в процессе работы с Ардуино

Цель: изучение возможных проблем в процессе работы с Ардуино и способы их ликвидации

1. Возможные проблемы в процессе работы с Ардуино

- Не прошиваются программы в Ардуино

В процессе передачи программы от компьютера к Ардуино участвует множество компонентов. Если хотя бы один из них сработает некорректно - процесс прошивки будет нарушен. В частности, процесс прошивки зависит от: драйверов Ардуино, выбора модели устройства и порта в среде Ардуино, физического подключения к плате и т.д. Ниже приведено несколько рекомендаций по устранению неполадок того или иного компонента.

Убедитесь, что в меню Инструменты-> Плата выбрана именно ваша модель Ардуино. Затем проверьте, чтобы в меню Инструменты -> Порт был выбран правильный порт (если порт не появляется, попробуйте перезапустить среду IDE при подключенном к компьютеру устройстве).

Драйверы позволяют прикладному ПО на компьютере (т.е. среде Ардуино) общаться с подключенным к нему "железом" (платой Ардуино). Применительно к Ардуино, драйверы отвечают за создание в системе виртуального последовательного порта (или виртуального COM-порта). Самый простой способ проверить, правильно ли установлены драйвера - это подключить Ардуино к ПК и открыть меню Инструменты -> Порт в среде разработки. После подключения Ардуино здесь должны появиться новые пункты меню. При этом абсолютно не важно, какое именно имя порта будет присвоено Ардуино.

Если программа долго открывается или вылетает при запуске, либо меню Инструменты в ней открывается слишком долго, то в Диспетчере устройств попробуйте отключить все последовательные Bluetooth-порты и прочие сетевые COM-порты. Во время запуска или открытия меню Инструменты, среда разработки Ардуино сканирует все последовательные порты вашего компьютера, соответственно, наличие таких сетевых портов

может иногда приводить к «подвисаниям» или «вылетам» программы.

Убедитесь, что у вас не запущены программы-сканеры последовательных портов, такие, как USB Cellular Wifi Dongle (например, от Sprint или Verizon), приложения для синхронизации PDA, драйвера Bluetooth-USB (например, BlueSoleil), виртуальные демоны и пр..

Убедитесь, что причиной блокирования последовательного порта не является брандмауэр (например, ZoneAlarm).

Нужно также закрыть все программы, осуществляющие мониторинг данных, идущих через USB между Ардуино и ПК (например, Processing, PD и др.).

Первым делом убедитесь, что Ардуино включен (горит зеленый светодиод) и соединен с компьютером.

На время прошивки отключите все устройства от цифровых выводов 0 и 1, поскольку эти выводы связаны с интерфейсом подключения Ардуино к компьютеру (после успешной прошивки кода в контроллер их можно снова задействовать).

Попробуйте прошить Ардуино, отключив от него все устройства (кроме USB-кабеля, разумеется).

Убедитесь в том, что плата не касается металлических предметов, проводящих ток.

Попробуйте другой USB-кабель, иногда они тоже выходят из строя.

Если программы по-прежнему не прошиваются, задайте вопрос на форуме. В своем сообщении укажите, пожалуйста, следующую информацию:

Версию вашей операционной системы, тип вашего Ардуино. Получалось ли у вас до этого успешно прошить плату. Если да, то опишите, что вы делали с платой до того, как она перестала работать, какое программное обеспечение в последнее время вы ставили или удаляли со своего компьютера? Сообщения, появляющиеся при попытке загрузить программу в Ардуино. Для получения сообщений зажмите кнопку Shift при нажатии на кнопку Вгрузить на панели инструментов.

- **При компиляции программы возникает ошибка java.lang.StackOverflowError**

Среда разработки Arduino осуществляет предварительную обработку вашего скетча путем манипуляций над кодом с помощью регулярных выражений. Иногда определенные строки текста приводят к сбоям этого процесса. Если вы видите примерно такую ошибку:

java.lang.StackOverflowError
at java.util.Vector.addElement(Unknown Source)
at java.util.Stack.push(Unknown Source)
at
com.oroinc.text.regex.Perl5Matcher._pushState(Perl5Matcher.java)

или:

at com.oroinc.text.regex.Perl5Matcher._match(Perl5Matcher.java)
at com.oroinc.text.regex.Perl5Matcher._match(Perl5Matcher.java)
at com.oroinc.text.regex.Perl5Matcher._match(Perl5Matcher.java)
at com.oroinc.text.regex.Perl5Matcher._match(Perl5Matcher.java)
at com.oroinc.text.regex.Perl5Matcher._match(Perl5Matcher.java)
at com.oroinc.text.regex.Perl5Matcher._match(Perl5Matcher.java)
at com.oroinc.text.regex.Perl5Matcher._match(Perl5Matcher.java)

то это именно такая ситуация. Проверьте код на наличие необычных последовательностей символов, связанных с "двойными кавычками", "одинарными кавычками", обратным слэшем \, комментариями и пр. Например, к таким проблемам может приводить отсутствие кавычек или, к примеру, такая последовательность '\"' (вместо '"').

- Подвисает среда разработки Ардуино (на Windows) при попытке прошить программу

Эта проблема может возникать из-за конфликта Ардуино с процессом Logitech 'LVPrcSrv.exe'. Откройте Диспетчер задач и найдите его в списке процессов. Если он там есть, то следует завершить процесс перед прошивкой программы.

- Вылетает ошибка при запуске arduino.exe на Windows

Если при запуске arduino.exe под Windows возникает ошибка, например такая:

Arduino has encountered a problem and needs to close.

то попробуйте запустить Ардуино с помощью файла run.bat. Пожалуйста, будьте терпеливы, т.к. для запуска среды Ардуино требуется некоторое время.

Если при запуске Ардуино возникает такая ошибка:

Java Virtual Machine Launcher: Could not find the main class. Program will exit.

проверьте, правильно ли вы распаковали содержимое .zip-архива Ардуино - в частности, убедитесь, что папка lib находится непосредственно внутри директории Arduino и содержит файл pde.jar.

- Устройство не отображается в меню Инструменты ->Порт

Убедитесь, что драйвера на микросхему FTDI установлены корректно.

После этого убедитесь, что плата действительно подключена: меню со списком портов обновляется каждый раз при открытии меню Инструменты, соответственно, если вы только что отключили плату, то ее не будет в этом списке.

Проверьте, не запущены ли программы, сканирующие все последовательные порты, например, приложения для синхронизации PDA, драйвера Bluetooth-USB (такие, как BlueSoleil), виртуальные демоны и пр.

На Windows-системах номер COM-порта, присваиваемого Арудино, может оказаться слишком большим. Попробуйте уменьшить номер порта, ассоциированного с микросхемой FTDI.

- При попытке прошить программу возникает ошибка "invalid device signature"

Следующая ошибка:

avrdude: Yikes! Invalid device signature.Double check connections and try again, or use -F to override this check.

может означать одно из двух: либо у вас неправильно выбрана плата в меню Инструменты-> Плата, либо вы используете неправильную версию программы avrdude.

Контрольные вопросы:

1. От чего зависит процесс прошивки программы в Ардуино?
2. Какие действия необходимо выполнить для устранения проблем при запуске скетча?
3. Из-за чего может подвисать среда разработки Ардуино на Windows?
4. Почему устройство не отображается в меню Инструменты ->Порт?

Список используемой литературы:

1. Гусев, В.Г. Электроника и микропроцессорная техника: Учебник / В.Г. Гусев, Ю.М. Гусев. - М.: КноРус, 2013. - 800 с.
2. Брамм, П. Микропроцессор 80386 и его программирование / П. Брамм, Б. Брамм. - М.: Мир, 1990.
3. Гребенко, Ю.А. Микропроцессоры / Ю.А. Гребенко, В.К. Раков. - М.: Издательство МЭИ, 2000.
4. Гуревич, В.И. Микропроцессорные реле защиты. Устройство, проблемы, перспективы / В.И. Гуревич. - М.: Инфра-Инженерия, 2011. - 336 с.
5. Иванников, А.Д. Моделирование микропроцессорных систем / А.Д. Иванников. - М.: Энергоатомиздат, 1990.
6. Калабегов, Б.А. Цифровые устройства и микропроцессорные системы / Б.А. Калабегов. - М.: Горячая линия-Телеком, 2007.-336с.
7. Калабеков, Б.А. Цифровые устройства и микропроцессоры / Б.А. Калабеков. - М.: Радио и связь, 1997.
8. Калашников, В.И. Электроника и микропроцессорная техника: Учебник для студ. учреждений высш. проф. обр. / В.И. Калашников, С.В. Нефедов. - М.: ИЦ Академия, 2012. - 368 с.
9. Кузин, А.В. Микропроцессорная техника: Учебник для студ. сред. проф. образования / А.В. Кузин, М.А. Жаворонков. - М.: ИЦ Академия, 2013. - 304 с.
10. Новиков, Ю.В. Основы микропроцессорной техники: курс лекций / Ю.В. Новиков, П.К. Скоробогатов. - М.: ИНТУИТ.РУ, 2003. - 440 с.
11. Новиков, Ю.В. Основы микропроцессорной техники: Учебное пособие / Ю.В. Новиков, П.К. Скоробогатов. - М.: БИНОМ. ЛЗ, ИНТУИТ.РУ, 2012. - 357 с.
12. Новожилов, О.П. Основы микропроцессорной техники. В 2-х т. Т. 2. Основы микропроцессорной техники: Учебное пособие / О.П. Новожилов. - М.: ИП РадиоСофт, 2011. - 336 с.
13. Шевкопляс, Б.В. Микропроцессорные структуры. Инженерные решения: справочник / Б.В. Шевкопляс. - М.: Радио и связь, 1993.
14. Шонфелдер, Г. Измерительные устройства на базе микропроцессора A Tmega: Пер. с англ. / Г. Шонфелдер, К. Шнайдер. - СПб.: БХВ-Петербург, 2012. - 288 с.

Учебное издание
Лабораторные работы по микропроцессорным системам
Методические указания

Составители:
Евгений Николаевич Аксёнов
Анна Васильевна Деткова

Издаётся в авторской редакции
Уч.-изд. л.6.0. Тираж 10 экз.
Отпечатано на принтере SAMSUNGML 1675

90