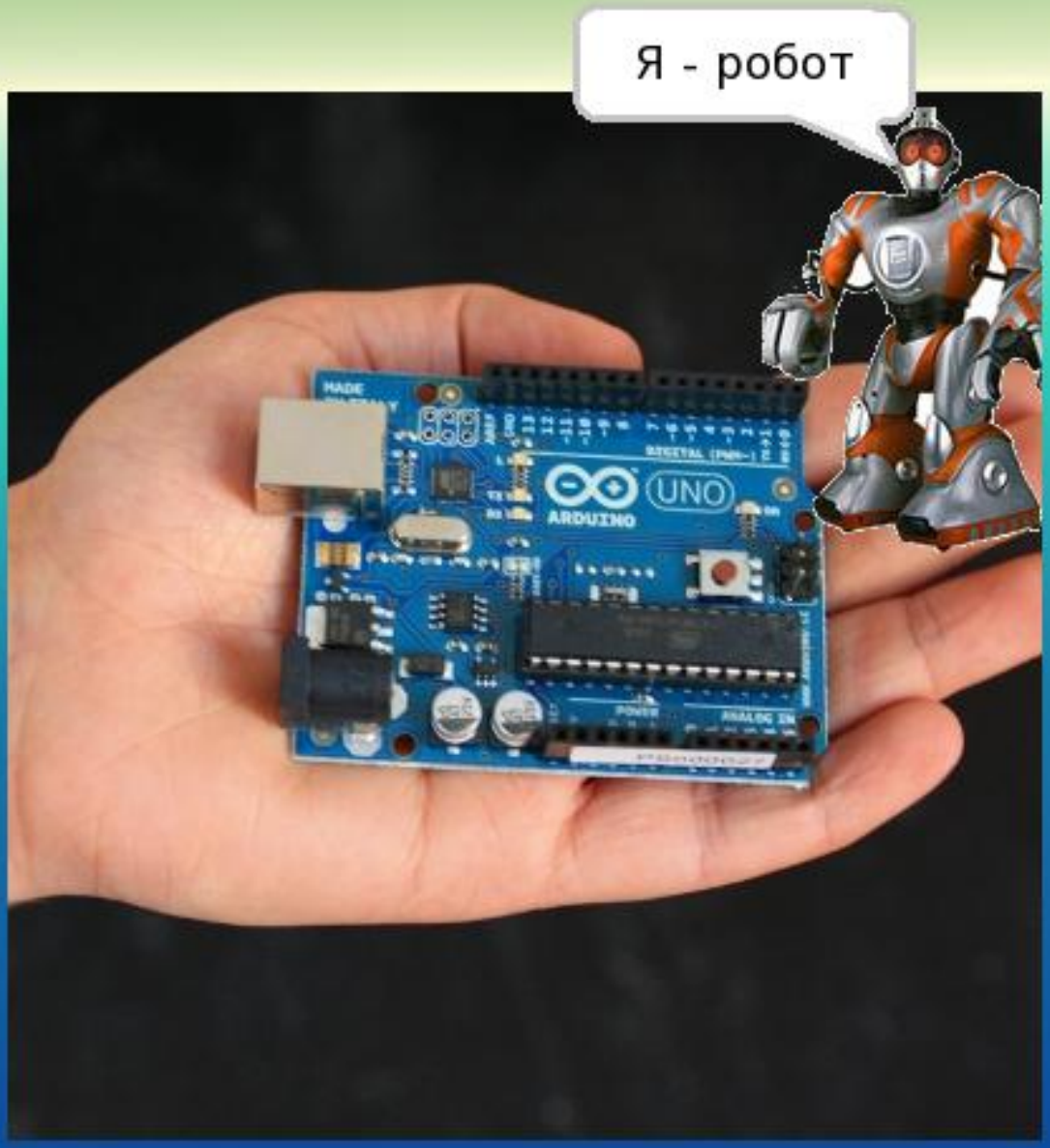
В.Н. Гололобов

С чего начинаются роботы?

О проекте Arduino для школьников(и не только)



Москва 2011

Моим близким – жене Тамаре и детям Анне и Денису, посвящается.

Несколько слов о книге

Есть такой открытый проект, который называется Arduino. Основа этого проекта – базовый аппаратный модуль и программа, в которой можно написать код для контроллера на специализированном языке, и которая позволяет этот модуль подключить и запрограммировать.

Модуль легко соединяется с разными исполняющими устройствами, позволяя создавать и роботов, и устройства автоматики, и приборы.

С момента появления проекта Arduino у него появилось множество почитателей – достаточно ввести в поисковую строку слово arduino, как вы обнаружите сотни сайтов, посвящённых этой теме, сотни проектов, основанных на Arduino.

На английском языке издано несколько книг. И эта должна восполнить пробел в части книг на русском языке.

Хотя книга рассчитана на школьников, она может быть интересна радиолюбителям, и, если не книга, то сам проект может быть интересен преподавателям, и не только работающим в школе, но и в других учебных заведениях, где изучают программирование и работу с микроконтроллерами.

Так что же эта книга? Она в основном описывает ряд программ, которые предназначены для работы с модулем Arduino, как сама программа Arduino, как S4A, как VirtualBreadBoard…

Если все эти программы почти обычным образом устанавливаются в Windows, то в Linux, а они работают и в этой операционной системе, есть особенности, которые описаны в этой книге. Повышенное внимание к Linux в последнее время делает актуальным подобное описание.

Помимо этого в книге рассказано о средах разработки AVR-контроллеров общего назначения, которые поддерживают работу с модулем Arduino – AVR Studio, WinAVR, FlowCode. В основном касательно настройки для работы с Arduino.

Но почему о программах, если речь идёт о роботах?

Суть любого робота – это аппаратные средства с процессорной базой и программа (или набор программ). Поэтому программирование неотъемлемый элемент процесса создания даже самого простого робота.

Научившись программировать модуль Arduino, а программа приходит с огромным набором примеров, касающихся всех областей применения модуля, вы будете готовы создавать интересные и полезные электронные устройства, к которым относятся и роботы. Начните с простых проектов, а остальное в ваших руках.

Возможно школьники, познакомившись с увлекательнейшим проектом Arduino, выберут роботостроение своей будущей профессией. Но даже если нет, то, уверен, через много лет, вспоминая свои эксперименты в этой области, они будут рады, что не прошли мимо, не пожалели времени на освоение основ – им будет, что вспомнить.

Радиолюбители давно и успешно осваивают работу с микроконтроллерами. Они с упоением спорят, какой язык программирования лучше. Возможно, проект Arduino позволит им сделать окончательный выбор? Тем более, что модуль Arduino может работать как программатор для программирования других микроконтроллеров.

Словом, всё интересное и полезное, что есть в проекте Arduino, можно узнать, только работая с ним в компании таких же увлечённых людей.

Оглавление

Глава 1. Паровозик из Ромашково, начало 5
Глава 2. Установка программы Arduino в ALTLinux 13
Глава 3. Введение в работу с программой Arduino 24
Глава 4. Введение в язык программирования Arduino 33
Глава 5. Arduino, визуальное программирование 43
Глава 6. Введение в язык программирования Scratch 53
Глава 7. Отладка программы на виртуальной плате 60
Глава 8. Немного больше о программе VirtualBreadboard 72
Глава 9. Паровозик из Ромашково, продолжение 90
Глава 10. С чего начинаются роботы? 106
Приложение А. О языке программирования Arduino 117
Приложение Б. Работа с модулем Arduino в других средах разработки 157

Глава 1. Паровозик из Ромашково, начало

«Читай книги... читай руководство, а мне не интересно. Мне интересно сразу что-то сделать. Вот, возьму и сделаю... ага, подходит к семафору паровоз, даёт тайный сигнал, и семафор открывается…».

Не думаю, что так. Перед тем как сделать устройство, перед тем, как «заливать» программу в модуль Arduino, нужно написать код. Запускаем программу Arduino... Ах, да. Это у меня программа установлена, а у вас, возможно нет. Тогда так.

Можно в любом поисковике, Yandex, Rambler, Google, в любом, ввести в окно поиска слова arduino download, и вы наверняка найдёте ссылки на сайт проекта.

Рис. 1.1. Основная страница проекта Arduino

Кнопка «Download» и приведёт вас на страницу загрузки. Если вы используете операционную систему Windows, есть версия для Windows; если Linux, есть и для некоторых дистрибутивов Linux; и есть, наконец, для Mac OS X.

Не знаю, как с Mac OS, но с Windows всё просто — щёлкнул по ссылке Windows, дождался конца загрузки, распаковал файл, а он в самом удобном варианте упаковки, который может распаковываться любым свободным архиватором, распаковал и всё. Или почти всё. В отличие от других программ, Arduino вместе с папкой можно сразу перенести в корневую директорию диска C, а для удобства, открыть папку с программой, выделить файл arduino.exe, щёлкнув правой клавишей, и из выпадающего меню выбрать пункт «Создать ярлык». После создания ярлыка его мышкой можно перетащить на рабочий стол и запускать программу обычным образом — двойной

щелчок мышки по ярлыку.

Фу, вот и установили программу. Запускаем её... Ах, да. Я-то уже купил модуль Arduino, а вы, возможно, нет. Как я это сделал?

Я искал самый дешёвый модуль. Модулей Arduino, поскольку проект открытый, много, и под разными именами, скажем, arduino, freeduino, craftduino и т.д., и цены у них разные. Самый дешёвый модуль в Москве я нашёл в Интернет-магазине CarMonitor.ru. Не слишком дорогие модули (и интересный сайт) в Интернет-магазине RoboCraft.ru, или в duino.ru:

Рис. 1.2. Разновидности одного из клонов модуля Arduino

Модули отличаются своим видом друг от друга, удобством использования, но все позволяют подключить их к компьютеру, запустив программу arduino написать код, «прошить» полученную программу в модуль и проверить работу программы в живом виде. И все модули имеют возможность подключать к ним платы расширения, либо готовые, купленные в магазине, либо изготовленные самостоятельно. Кстати, проект открытый, то есть, вы не только можете бесплатно скачать программу и использовать её (и, к слову, переделать по своему вкусу, если вы умеете это делать), вы можете самостоятельно изготовить базовый модуль — есть его схема, есть прошивка, есть всё для самостоятельного изготовления.

Кроме базовых модулей, как показано выше, многие Интернет-магазины предлагают множество дополнений: макетные платы, материал для изготовления механических частей, моторы и сервоприводы, и разного рода датчики — и фото, и тензо, и пьезо, и т.п. Это целый мир устройств, из которых можно создавать и роботов, и устройства автоматики, забавные и полезные игрушки, и «взрослые» полезные устройства.

Всё. Всё я, кажется, сказал, можно начинать. Или не всё?

Да, ещё подключение. О том, как подключить модуль Arduino в операционной системе Windows, можно прочитать, например, на сайте RoboCraft.ru:

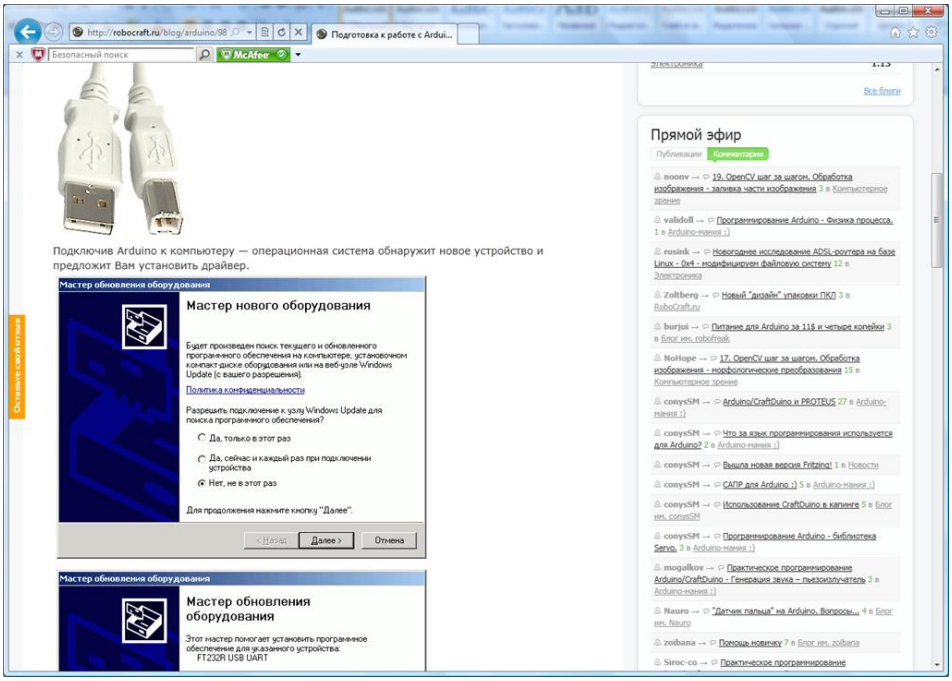


Рис. 1.3. Подключение модуля Arduino в Windows

Моя операционная система Windows Vista сама находит нужный ей драйвер и устанавливает драйвер для преобразователя USB в COM-порт, и добавляет COM5 в список устройств системы.

В программе Arduino я указываю последовательный порт COM5 и модуль Nano, поскольку мой модуль CarDuino именно таков.

Теперь можно написать (конечно, я её «срисую») какую-нибудь программу и попробовать её отправить в модуль Arduino. Мне понравилось, как в книге «Arduino programming notebook» самую простую программу назвали «Hello World» мира микроконтроллеров.

Рис. 1.4. Первая программа для модуля Arduino

Теперь, наверное, всё. Или нет?

Правильно. Я хотел ещё рассказать, как установить программу arduino в Linux.

Самая простая установка программы arduino в дистрибутиве Fedora (из тех дистрибутивов, что есть на моём компьютере, среди которых Fedora 14). Многие, пользуясь Linux, предпочитают работать в терминале. Я не из их числа, и без жестокой необходимости этого не делаю. Так для установки программ я использую приложение под названием Yumex. Его можно найти в разделе «Приложения-Система-Дополнение к Yum». Запустив приложение, дождавшись, когда оно закончит поиск всего ему нужного, следует выделить доступные программы (Available) и в окне поиска ввести arduino. Теперь достаточно нажать клавишу **Enter** на клавиатуре, чтобы нашлась программа. Отметив её (все три блока с именем arduino), нажимаем кнопку **Применить** и программа устанавливается на компьютер, как, впрочем, и любые другие программы, а их в Linux великое множество.

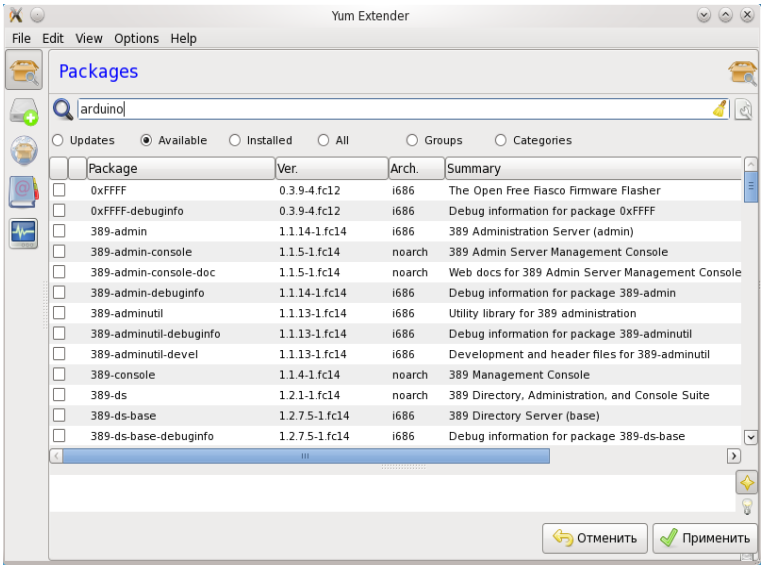


Рис. 1.5. Поиск и установка программы Arduino в дистрибутиве Fedora 14

Несколько сложнее устанавливается программа в openSuse 11.3.

В операционной системе openSUSE есть пользовательский интерфейс для установки программ. Его легко найти, если запустить управление системой от имени администратора.

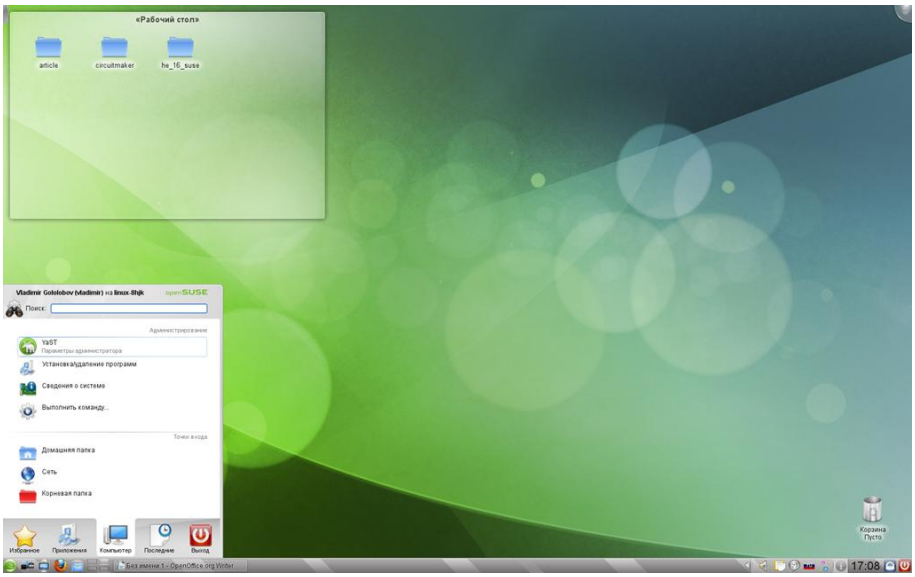


Рис. 1.6. Программа настройки системы YaST в openSUSE

Перед тем, как доступ будет предоставлен, потребуется пароль администратора компьютера.

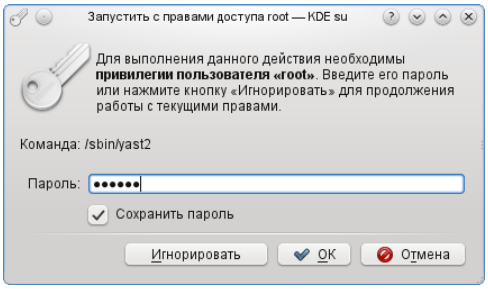


Рис. 1.7. Ввод пароля root

И, если пароль правильный, то в открывающемся окне можно найти, например, средство поиска нужных программ (пакетов).

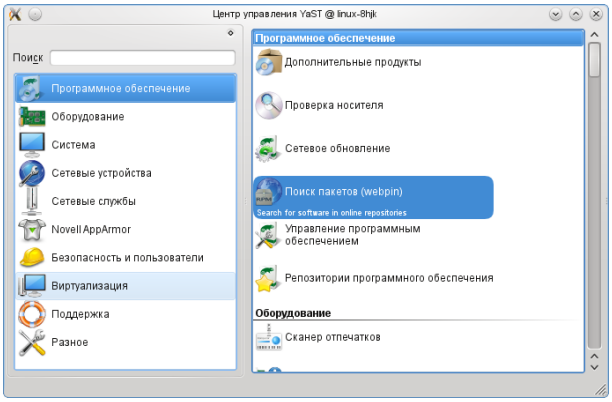


Рис. 1.8. Меню центра управления системой с выделенным разделом поиска

Имя, если вы его знаете, программы вводится в окно поиска.

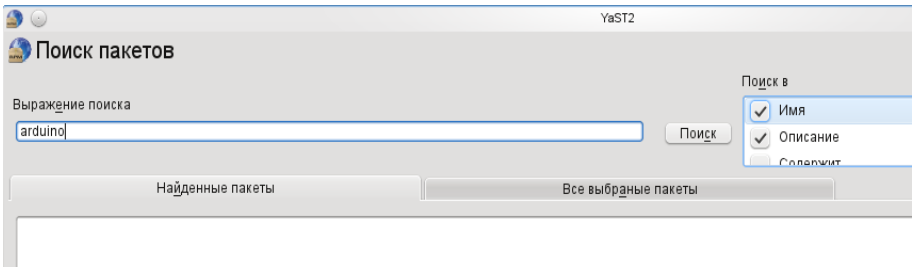


Рис. 1.9. Раздел поиска с окном поиска программ в составе openSuse

К сожалению, если запустить поиск, то ничего не будет найдено. Но это не значит, что программы arduino для openSUSE нет. Обратимся к сайту, где можно найти по адресу

http://arduino.cc/playground/Linux/OpenSUSE

не только нужную программу, но и подробную инструкцию по установке программы.

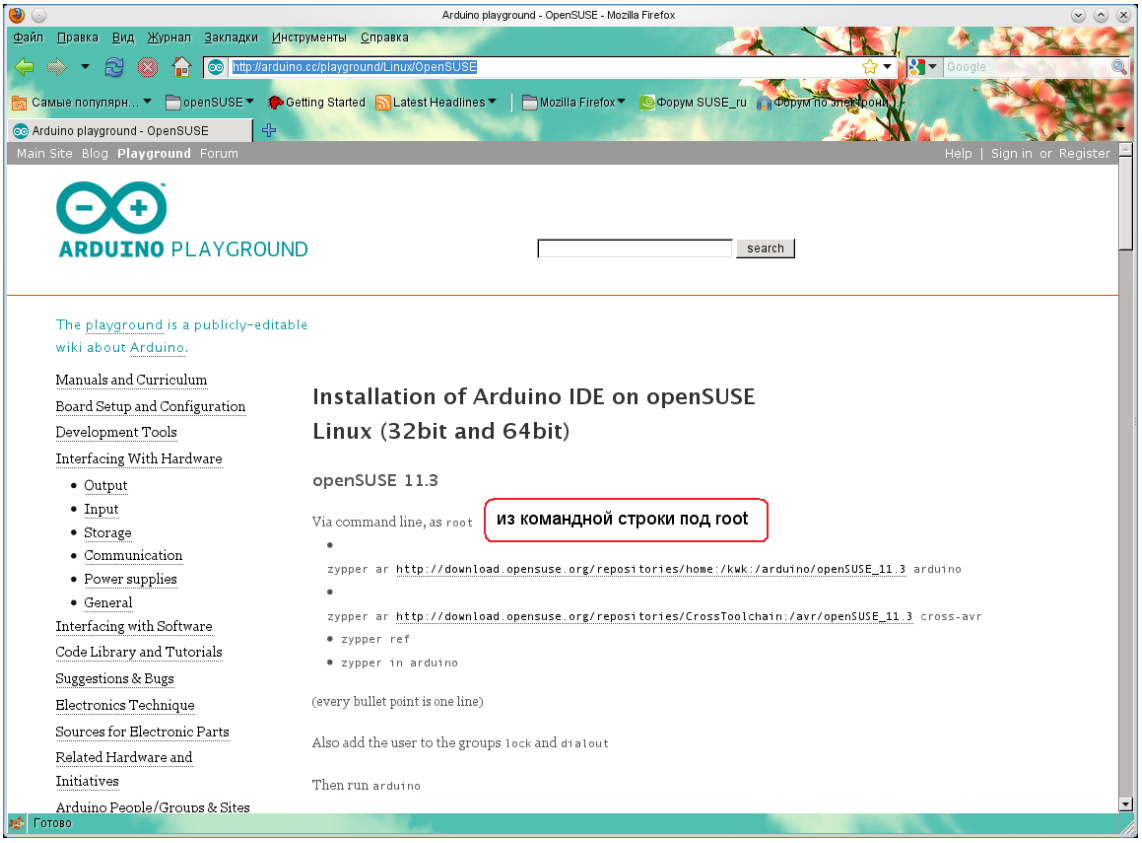


Рис. 1.10. Инструкция по установке программы на сайте Arudino

Я перевёл первый шаг установки. Что он означает?

В любом дистрибутиве Linux есть терминал. И есть те, кто предпочитает работать, вводя все команды с клавиатуры, в терминале. Иногда только так и можно (но не часто) выполнить что-то нужное. Как в данном случае. Если в разделе «Избранное» основного меню у вас нет терминала, то его можно найти на закладке «Приложения» основного меню в разделе «Система». Достаточно щёлкнуть мышкой по программе терминала, чтобы увидеть окно терминала. Теперь можно в открытом web-браузере скопировать полностью нужную строку команды, выделить, прочеркнув мышкой, и нажать правую клавишу мышки для получения выпадающего меню, из которого выбрать пункт «копировать». Скопировав строку, её можно вставить в окно терминала (перейти в окно терминала, нажать правую клавишу и выбрать команду «вставить»).

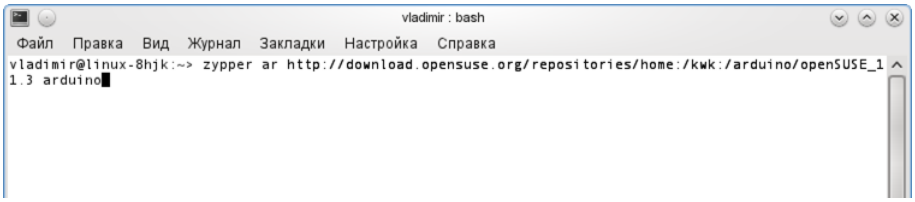


Рис. 1.11. Ввод первой команды инструкции в окно терминала

Все строки инструкции следует копировать полностью. После вставки команды можно нажать клавишу «Enter» и останется только ответить на ряд вопросов, чтобы программа была установлена. Конечно, следует повторить все команды, записанные в инструкции.

После установки программы, как это и написано в конце инструкции, следует пользователя, то есть, вас, включить в две группы: lock и dialout. В этом опять поможет Yast.

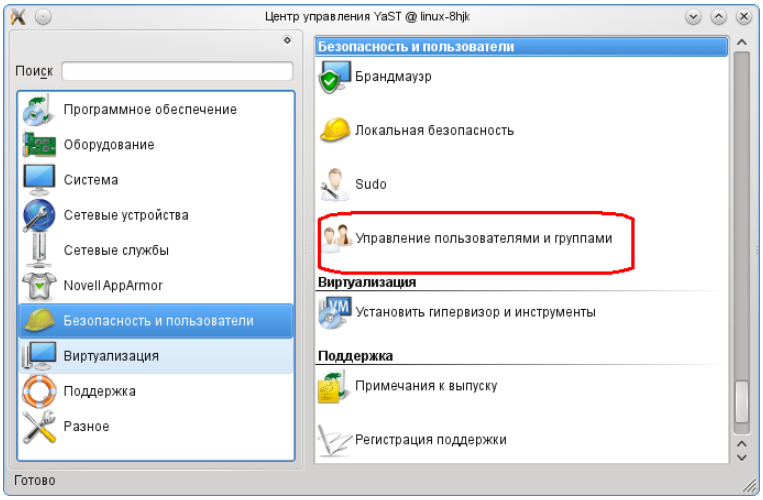


Рис. 1.12. Вход в раздел управления группами и пользователями

При входе в этот раздел, открывается окно, где есть все пользователи компьютера.

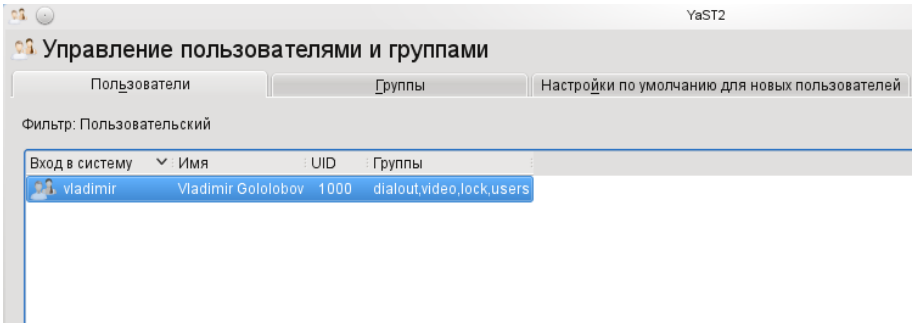


Рис. 1.13. Окно закладки «Пользователи» с перечнем всех пользователей компьютера

Нажав кнопку «Редактировать», мы попадаем в новое окно, где на вкладке «Подробности», есть возможность установить галочки рядом с нужными группами.

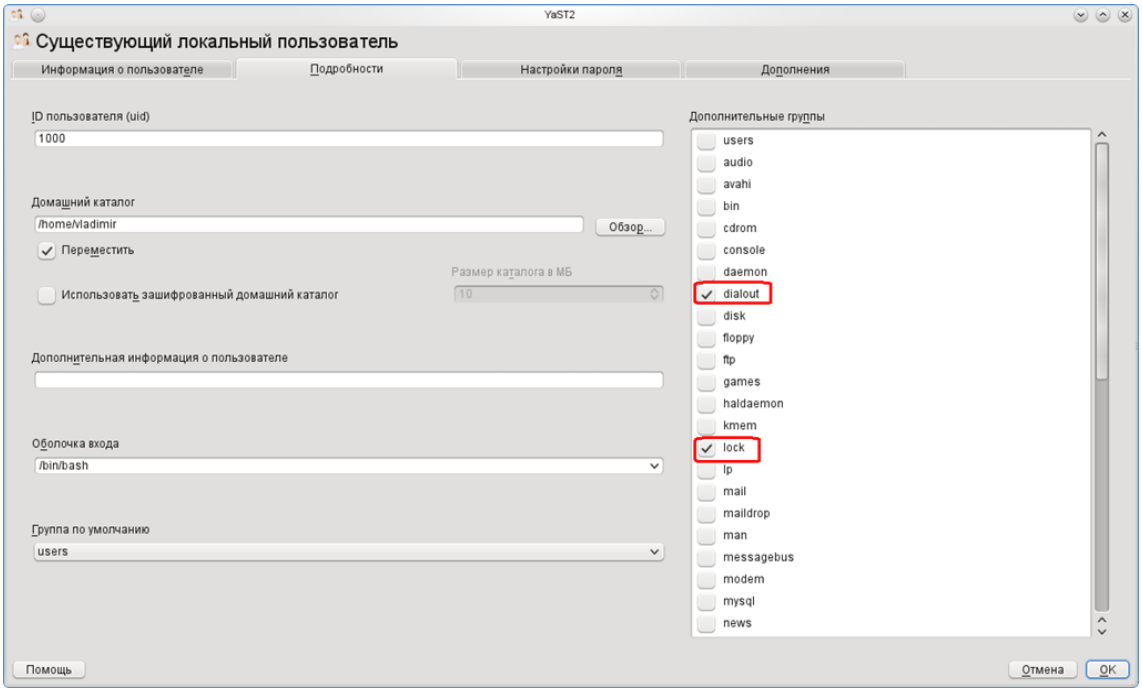


Рис. 1.14. Закладка «Подробности» для добавления пользователя в группы

Теперь кнопка «ОК» выводит нас (постепенно) из программы административного управления

системой. Останется только перезагрузить компьютер и, возможно, добавить программу Arduino в основное меню.

Осталось установить программу на последнем из обитающих на моём компьютере дистрибутивов ALTLinux 5.1. Он, как и остальные, имеет удобную программу для установки других программ, но…

Глава 2. Установка программы Arduino в ALTLinux

К сожалению, в ALTLinux нет готового пакета, который можно скачать, как в Fedora. И на сайте проекта Arduino нет описания, как установить программу в дистрибутиве ALTLinux. По советам на форуме ALTLinux можно скачать и установить программу Kontrollerlab, но будет ли она работать с модулем Arduino? Это вопрос. Тем не менее, можно, используя менеджер установки пакетов, загрузить необходимые файлы (если java установлена, как рекомендуют на сайте Arduino): avr-gcc, avr-gcc++ (Система-Приложения-Менеджер пакетов). Используя кнопку «Искать», в окне поиска ввести avr-gcc, найденные приложения отметить для установки, щёлкнуть по приложению правой клавишей мышки, выбрать из выпадающего меню раздел «Отметить для установки», и, отметив все приложения, нажать кнопку «Применить».

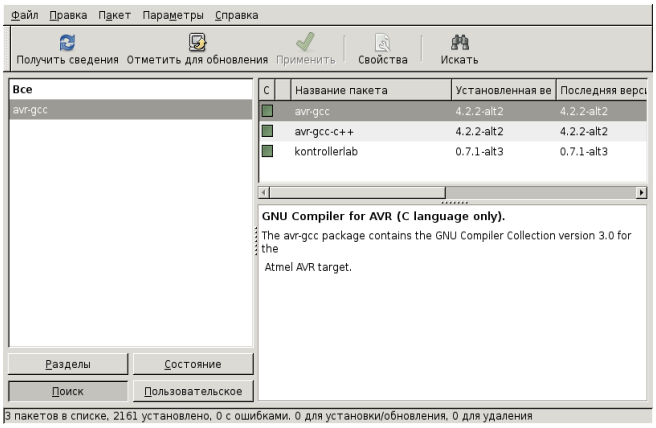


Рис. 2.1. Программа установки программ в ALTLinux

На сайте Arduino в разделе Download, после перехода в этот раздел, можно скачать пакет для Linux.

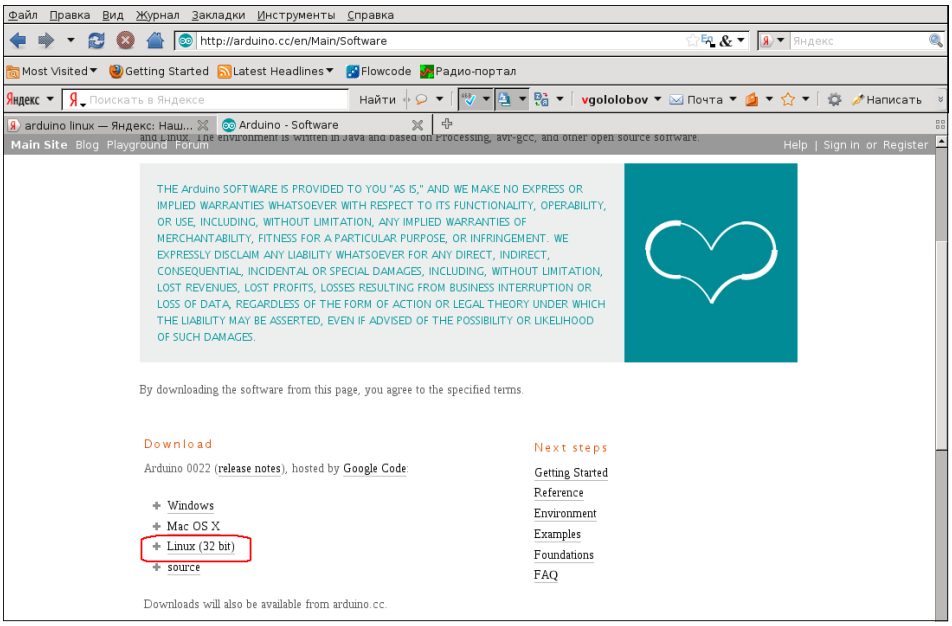


Рис. 2.2. Пакет Arduino для ALTLinux

Для распаковки пакета используется Менеджер пакетов (в разделе «Система» основного меню). Можно перетащить скачанный пакет Arduino из папки «Загрузки» на рабочий стол, открыть его в Менеджере пакетов (если щёлкнуть по загруженному файлу правой клавишей мышки, то можно использовать раздел «Открыть с помощью Менеджера архивов»).

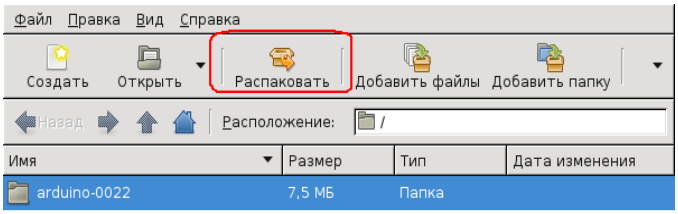


Рис. 2.3. Распаковка пакета в менеджере архивов

Затем, используя кнопку «Распаковать», распаковать полученный пакет. Имя распакованного пакета (arduino-0022 в данном случае) лучше изменить на arduino. Следующий шаг – перенести этот пакет в раздел /usr/share). Но, чтобы это сделать, нужно иметь права администратора системы. И проще всего воспользоваться теми возможностями, которые есть в дистрибутиве. Если в основном меню зайти в раздел «Система» на закладке «Приложения», то можно обнаружить подраздел «Дополнительные приложения». В этом подразделе есть ряд полезных программ и, в частности, «Менеджер файлов (в режиме администратора)». Запустив менеджер файлов, можно легко перенести в нужное место папку с программой.

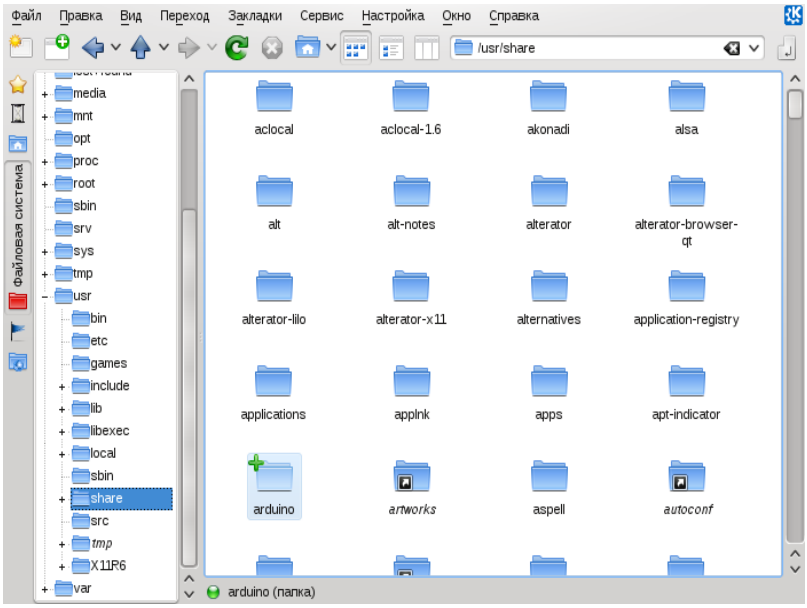


Рис. 2.4. Менеджер файлов в ALTLinux

Хорошо бы программу запустить... Но, конечно, вы не обнаружите программу в основном меню. В Windows для этой цели используется командная строка, в Linux терминал. В окно терминала вводим команду: /usr/share/arduino/arduino, и получаем...

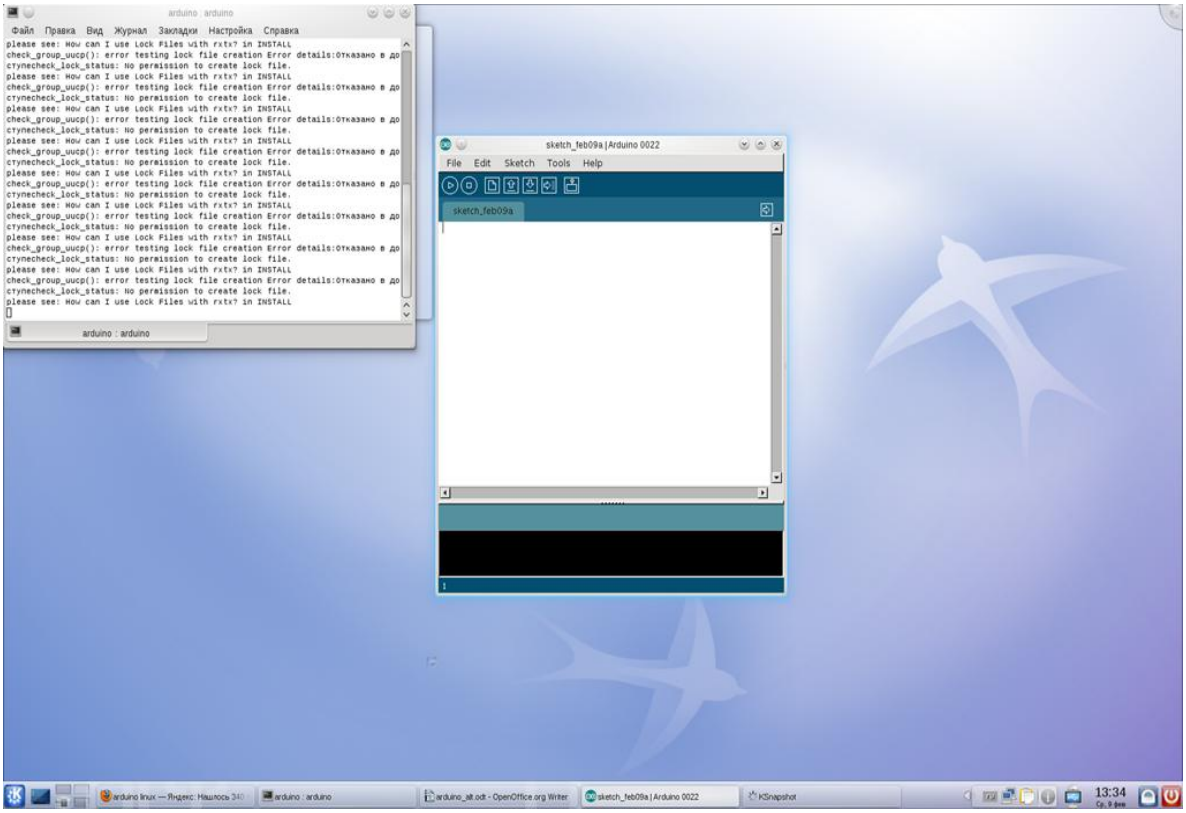


Рис. 2.5. Первый запуск программы в терминале

В использовании терминала есть и ряд преимуществ — на рисунке видно, что запуск программы сопровождается рядом ошибок, отображаемых при запуске в окне терминала. Ошибки могут быть вызваны тем, что программа требует доступа к системным ресурсам, а вам, как рядовому пользователю, доступ к этим ресурсам запрещен.

Если вы помните, в дистрибутиве openSUSE нам понадобилось добавить пользователя в несколько групп. Проверим, вызваны ли ошибки тем, что мы не сделали этого в ALTLinux?

Полностью повторить сделанные в openSuse операции не получится из-за отсутствия программы добавления пользователя в разные группы. Но это не значит, что этого нельзя сделать — Linux, если знать, как это сделать, позволяет сделать всё, что захочется. Воспользуемся вновь терминалом для добавления пользователя (следуя рекомендациям для openSUSE и тому, что пишут на форуме) в группы: uucp, lock и dialout. Эти операции может выполнить только администратор, поэтому нужную команду мы начинаем с дополнительной команды sudo, что означает, выполнить команду от имени администратора.

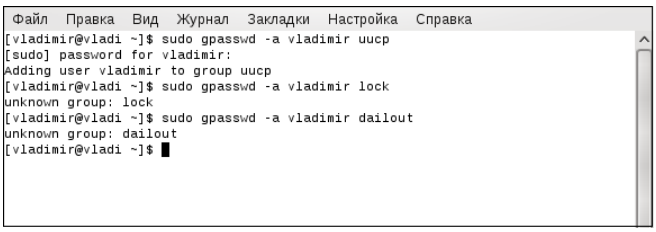


Рис. 2.6. Добавление пользователя в нужные группы

Скоро сказка сказывается, да не скоро дело делается. Ладно, при вводе команды для включения пользователя в группу dialout я сделал опечатку. Исправив её, я получил доступ в группу. Но! Группы lock в ALTLinux нет вообще! Как попасть в группу, если её нет, а программа arduino продолжает жаловаться.

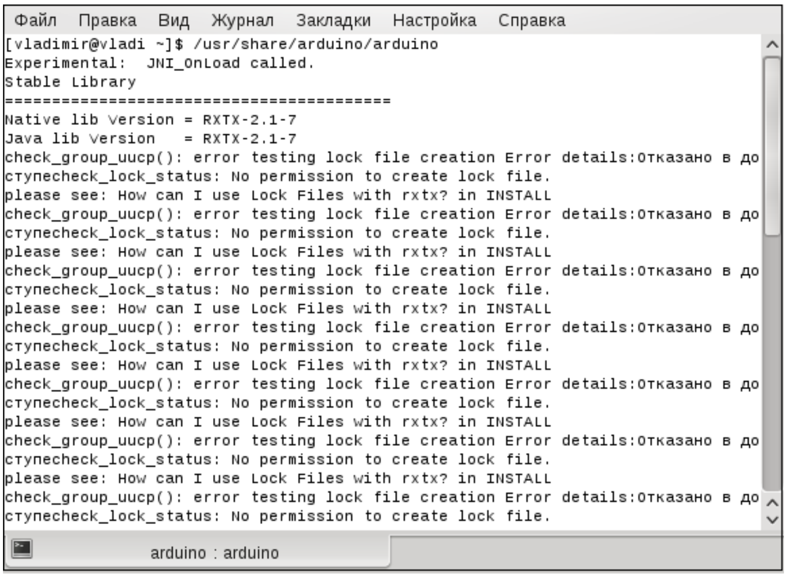


Рис. 2.7. Ошибки при запуске программы

Я не знаю, прав ли (или неправ), но я поступаю следующим образом: я добавляю себя в группу root (администраторов), используя ту же команду, что и раньше. Затем, используя файловый менеджер с правами администратора, нахожу директорию /var, где есть поддиректория lock.

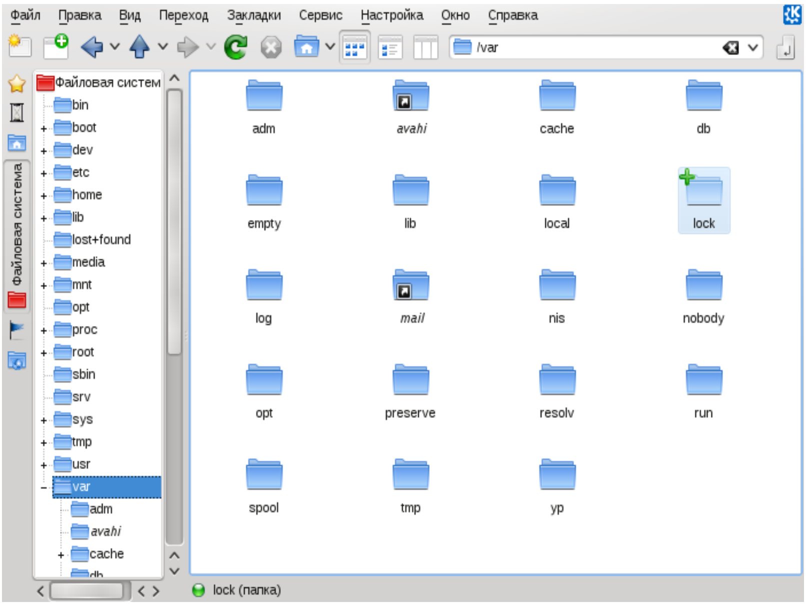


Рис. 2.8. Расположение папки lock в файловой системе

Щёлкнув правой клавишей мышки по этой папке, я открываю из выпадающего меню пункт «Свойства», открываю закладку «Права» и меняю для группы root права:

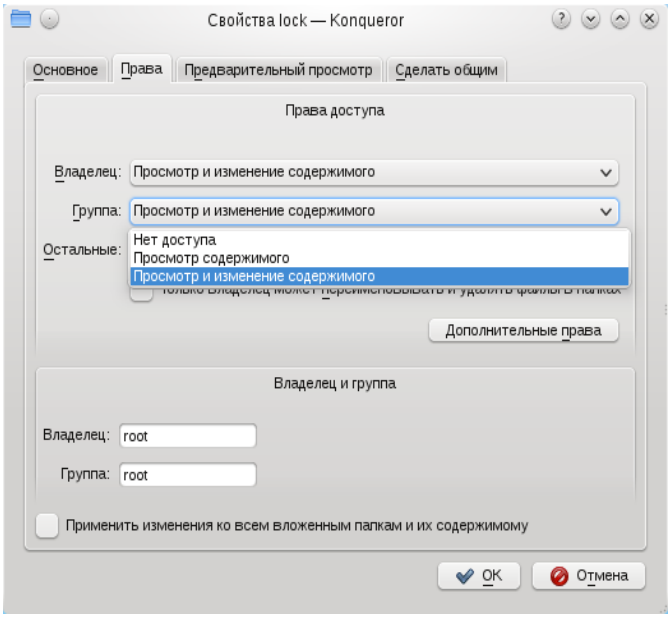


Рис. 2.9. Изменение прав группы администраторов

Теперь программа не жалуется, а прав ли я, покажет время.

Но беда не приходит одна. Вот её напарница: запускаем программу, заходим в основное меню: File-Examples-1.Basics-Blink и…

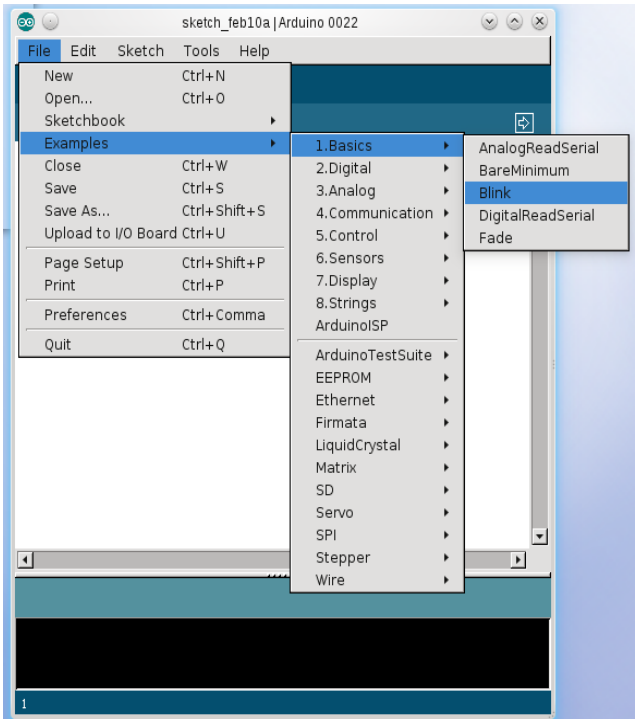


Рис. 2.10. Список примеров в пакете Arduino

…и получаем готовую программу, первая же попытка откомпилировать которую (основное меню «Sketch-Verify/Compile» или кнопка на инструментальной панели с иконкой «Play») приводит только к появлению длинного списка ошибок.

Об истоках проблемы я был, думаю, предупреждён — при скачивании программы на сайте Arduino я видел рекомендации, касающиеся компилятора, а именно: версия avr-gcc должна быть не ниже 4.3.2 (со всеми надлежащими утилитами). В ALTLinux в настоящее время версия 4.2.2. Если честно, то на этом месте следует остановиться, чтобы подождать выхода надлежащей версии.

Однако мне хочется сейчас проверить подключение модуля, работает оно или нет?

Рис. 2.11. Появление ошибок при компиляции программы

Вчитываясь в первую ошибку, я понимаю, что компилятор не находит файл заголовка stdlib.h, и ошибка возникает при обращении к файлу WProgram.h из состава пакета arduino-0022. Этому горю можно помочь. В директории /usr/include есть папка с именем avr.

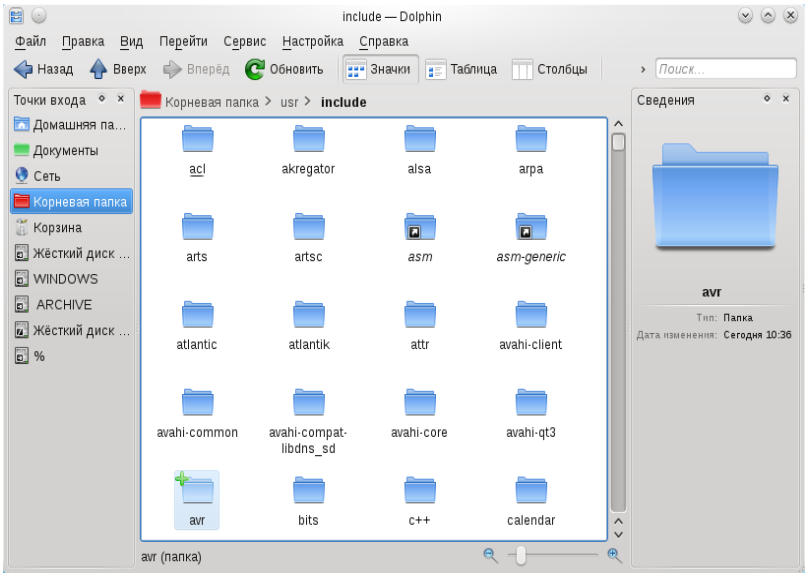


Рис. 2.12. Местоположение папки avr в файловой системе

Откроем её, выделим всё и добавим туда, где обездоленный файл WProgram.h ожидает нашей помощи.

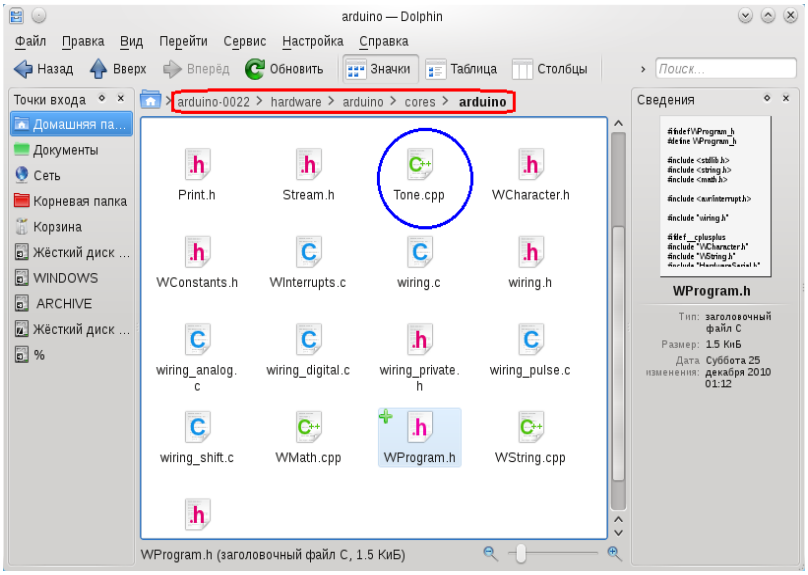


Рис. 2.13. Расположение файла в пакете Arduino

При следующей попытке скомпилировать программу из примеров, приведённых в Arduino, появляется новый набор ошибок. Он связан с тем, что компилятор не понимает синтаксис одного из файлов.

Откроем этот файл Tone.cpp (на рисунке выше он обведён кружком). Как все файлы на языке Си, этот открывается в обычном редакторе текста. Найдём строку 232, как указано в сообщении. В редакторе можно воспользоваться поиском, искать следует сочетание 0b (жалуется компилятор, похоже, на это сочетание).

Рис. 2.14. Сообщение об ошибке при компиляции файла

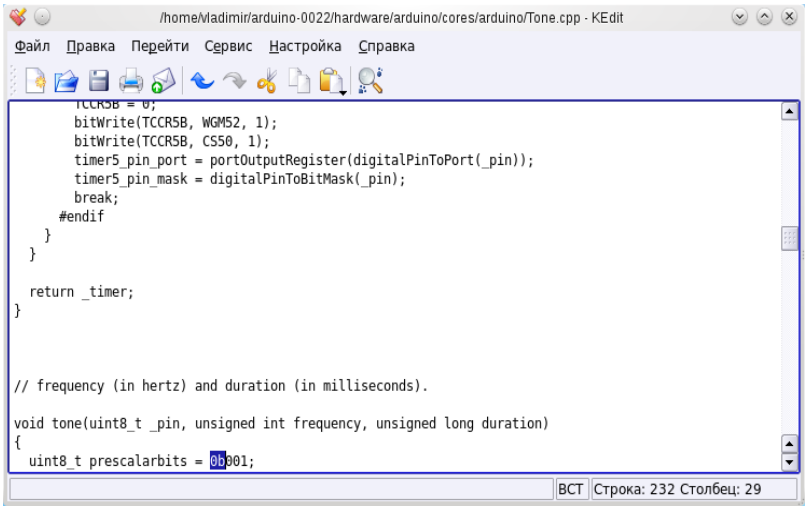


Рис. 2.15. Место «ошибки», вызывающее сбой при компиляции

Константа в строке 232 двоичная. Но написание, видимо, не нравится компилятору. Попробуем заменить это написание, то есть, 0b001 на шестнадцатеричное 0x01. Аналогично, используя поиск, заменим все двоичные числа, начинающиеся с 0b (ноль б), на шестнадцатеричные числа, как показано ниже.

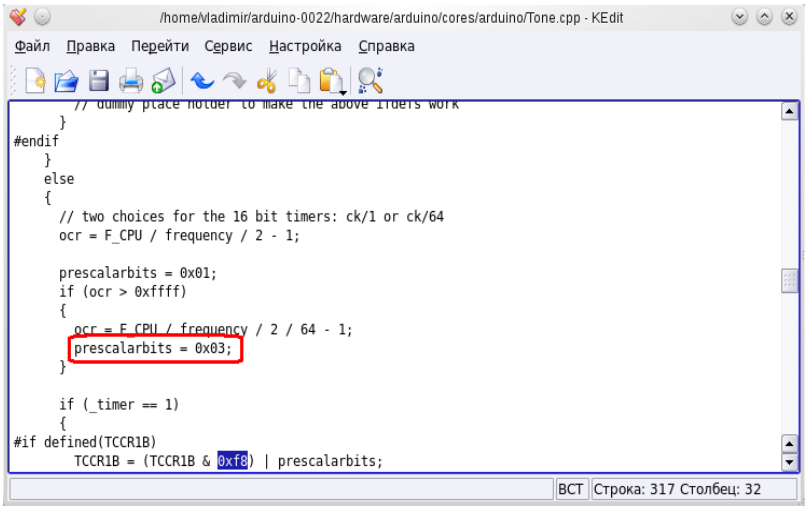
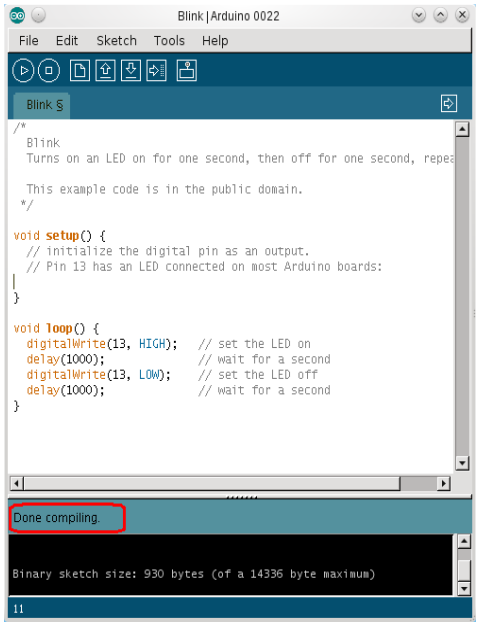


Рис. 2.16. Изменение синтаксиса

Сохранив файл, попытаемся вновь скомпилировать программу примера Blink.



Как видно из рисунка, компиляция прошла успешно. Осталось подключить модуль к USB порту, кстати, до подключения, если зайти в основном меню в Tools-Serial Port, то мы увидим только существующий COM-порт (в Linux это ttyS0). А после подключения появляется ttyUSB0.

Рис. 2.17. Сообщение об удачном завершении компиляции

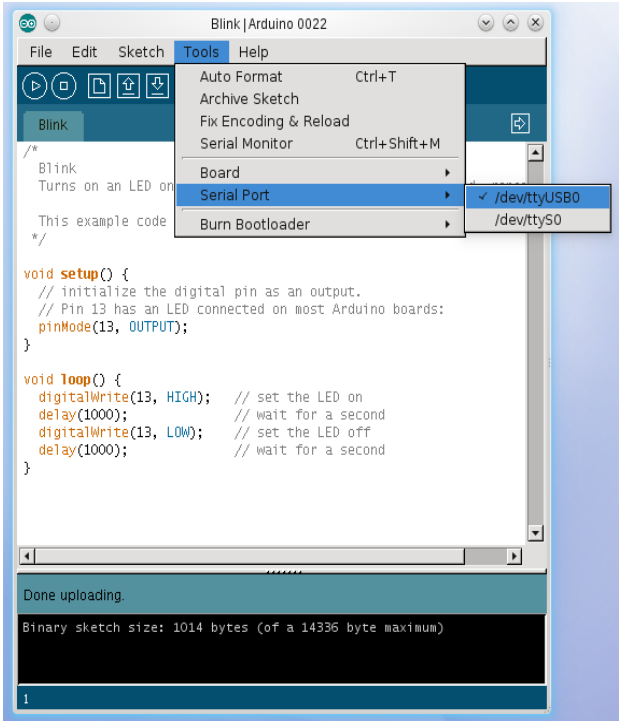


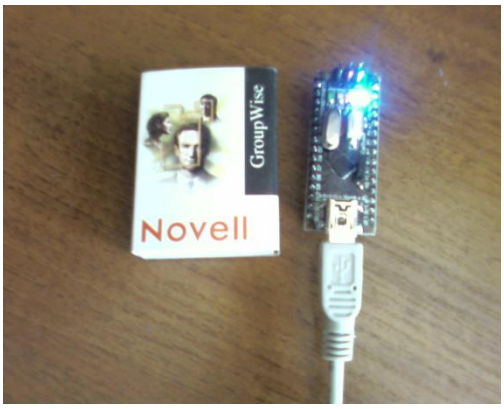
Рис. 2.18. Порт в Linux, к которому подключается модуль Arduino

И, когда мы нажимаем на кнопку инструментального меню, выполняющую команду загрузки программы, то в нижней части окна программы получаем сообщение, что загрузка выполнена (Done uploading).

Что ж, мы и в дистрибутиве ALTLinux проверили подключение модуля. В других операционных системах мы это тоже сделали. Осталось понять, а что мы сделали?

Если мы откомпилировали и загрузили программу в модуль Arduino, то, согласно программе, если мы подключим к выводу 13 светодиод с последовательно включённым резистором, то он будет мигать. Не знаю все ли, но, например, CraftDuino или мой CarDuino, для проведения

эксперимента уже имеют такой резистор и светодиод. И после загрузки программы светодиод начинает мигать.



Чтобы убедиться, мигать светодиод может и без нашего вмешательства, можно изменить программу:

Рис. 2.19. Работа программы Blink

```
int ledPin = 13;
void setup()
{
    pinMode (ledPin, OUTPUT);
    }

    void loop()
    {
    digitalWrite (ledPin, LOW);
    delay (1000);
    digitalWrite (ledPin, LOW);
    delay (1000);
}
```

Теперь светодиод, после компиляции и загрузки, не мигает. Как и заказано. А, значит, мы действительно скомпилировали и загрузили программу.

На сайте, о котором уже упоминалось, RoboCraft.ru, в статье о первой программе предлагается ещё более интересная проверка — отклик модуля на состояние светодиода по COM-порту.

```
int ledPin = 13;

void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);     // инициализация работы с COM-портом
}

void loop()
{
  digitalWrite(ledPin, HIGH);
  Serial.print("H");       // светодиод горит - пишем H
  delay(1000);
  digitalWrite(ledPin, LOW);
  Serial.println("L");     // светодиод погасили - пишем L
  delay(1000);
}
```

Загрузив программу в модуль, открыв терминал, можно увидеть работу программы. Помимо мигания светодиода в терминальном окне отображается состояние светодиода, включён ли он — приходит символ H, или выключен — символ L.

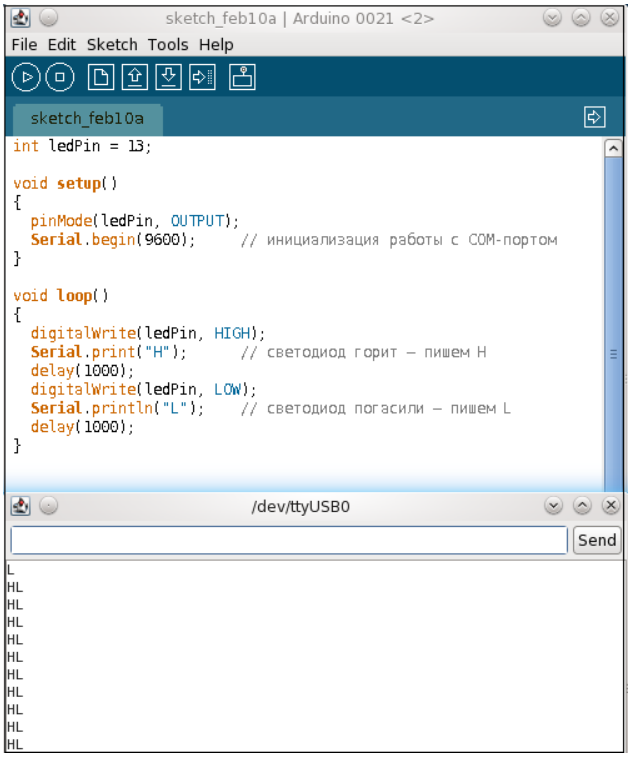


Рис. 2.20. Работа модифицированной программы

И, главное, если отключить модуль при работающей программе, символы перестают приходить. Значит, символы отправляет модуль!

Глава 3. Введение в работу с программой Arduino

Оставим на некоторое время несуразности, связанные с установкой программы и первым с ней знакомством. Если мы хотим собирать роботов, учить их ходить, начать следует с того, чтобы самим научиться неторопливо, шаг за шагом, продвигаться вперёд.

Разберём, что предоставляет в наше распоряжение программа Arduino.

Большую часть окна программы занимает, в общем-то, поле обычного текстового редактора. Хотя не совсем обычного. Это окно специализированного текстового редактора.

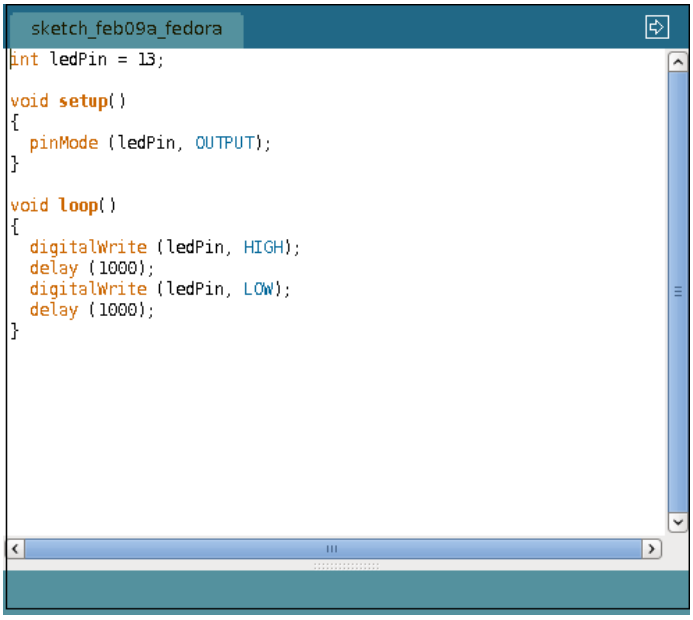


Рис. 3.1. Окно редактора текста в программе Arduino

Специализация редактора выражается и в том, что разным цветом выделяются служебные слова, и в том, что при написании программы автоматически выполняются отступы, формирующие более удобный текст для чтения. При компиляции, если возникает ошибка, строка, где компилятор «споткнулся», выделяется. Текущий текст программы имеет закладку, на которой имя файла. При создании нового файла он автоматически получает имя с текущей датой. Если файлов в проекте несколько, то закладок тоже будет несколько.

Над окном редактора располагаются основное меню и инструментальная панель. Основное меню, как и редактор, мало чем отличается от меню любой программы.

Рис. 3.2. Основное меню программы

Раздел *File* предполагает работу с файлами, *Edit* — команды редактирования. Но отличия, конечно, есть. Так пункт *Sketch* – специализированный раздел работы с проектами, а *Tools* касается разных приложений к аппаратным средствам. Что такое *Help*, думаю, не нуждается в объяснении.

Рассмотрим все разделы основного меню последовательно.

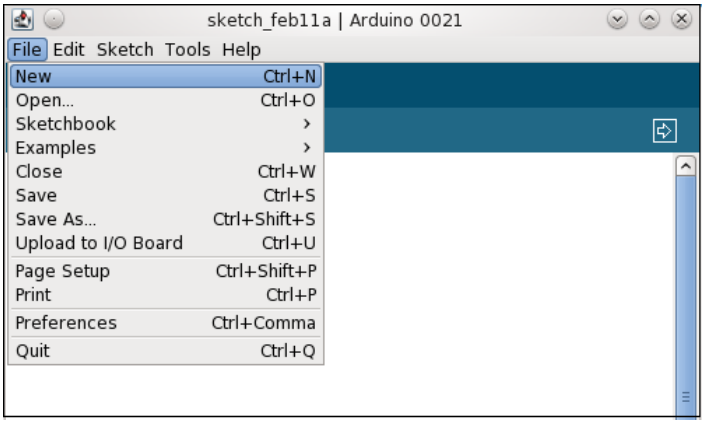
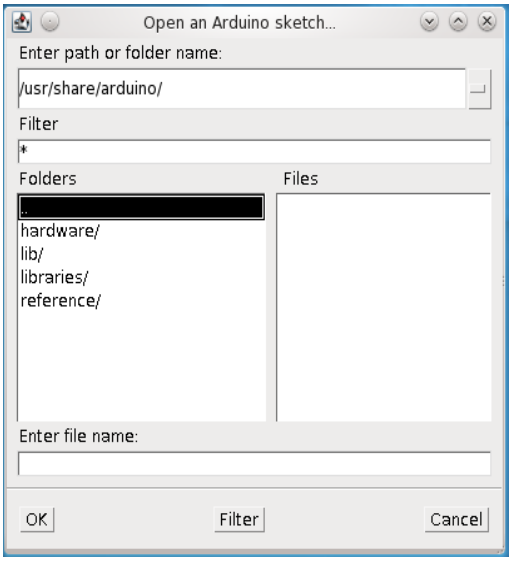


Рис. 3.3. Содержание раздела File основного меню

Пункт *New* выпадающего меню раздела *File* создаёт новое окно с чистым листом в редакторе. Пункт *Open...* открывает диалоговое окно проводника, где можно найти нужный файл программы. Многие программы используют системный менеджер файлов, но в данном случае проводник «принадлежит» программе.



По умолчанию открывается та папка, где заданы файлы после установки программы. Для перемещения по директориям файловой системы можно либо использовать мышку: двойной щелчок по верхней строке перемещает вас в родительскую директорию, - либо использовать клавиатуру, то есть, клавишу «Enter» для входа, клавиши стрелок курсорных клавиш для перемещений вверх и вниз. После выбора нужного файла, его имя появляется в окне «*Enter file name:*» и после нажатия на кнопку «ОК» открывается в окне редактора.

Рис. 3.4. Файловый менеджер программы Arduino

Следующий пункт раздела *File* — это *Sketchbook*. При установке программы (и первого запуска) в домашней папке появляется место, где можно хранить все свои проекты. Называется оно sketchbook. Если вы не сохраняли свои файлы, этот пункт пуст. Но, как только вы сохранили хотя бы свой первый файл, то, едва курсор мышки выделяет этот пункт, вы видите свои файлы.

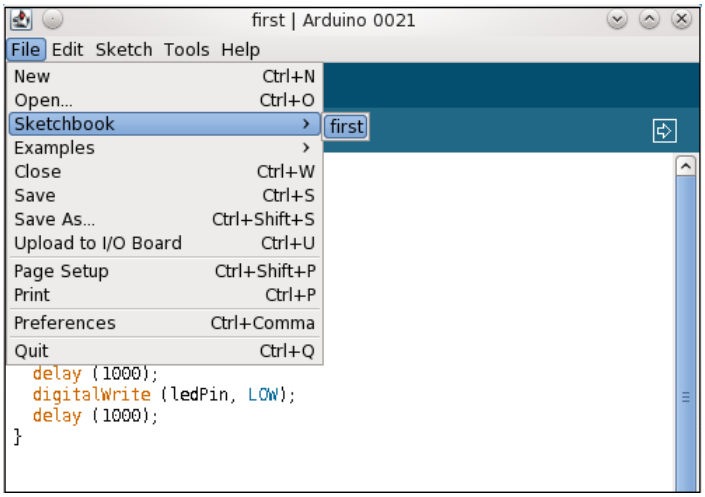


Рис. 3.5. Книга хранения программ

Следующий пункт, *Examples*, отрывает примеры, которые вы получили вместе с программой. Набор примеров и их классификация зависят от версии программы. Так версия 21 показывает такие примеры:

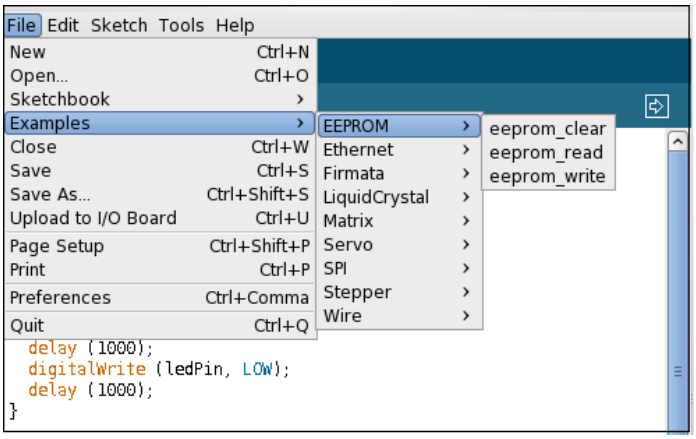


Рис. 3.6. Набор примеров, полученных вместе с пакетом Arduino

Я планирую рассказать о работе программы в разных операционных системах, так что, переходя к другим версиям, при случае, я покажу другой набор примеров.

Следующий пункт *Close* закрывает программу, если вы сохранили файл, и выводит диалог, напоминающий вам, что вы ещё не сохранили файл, и помогающий вам либо вернуться в редактор, либо сохранить файл.

Пункт *Save* служит для сохранения файла. А пункт *Save As...* позволяет сохранить открытый, например, из раздела примеров файл под другим именем и/или в другом месте. Команда открывает менеджер файлов, о котором мы говорили выше, и вы вольны выбрать нужное место и имя, но следует избегать, как и при установке программы, директорий и имён, написанных не латиницей.

Следующий пункт *Upload to I/O Board* — это команда загрузки откомпилированного файла в модуль Arduino.

Page Setup задаёт размеры и ориентацию страницы. А команда *Print* открывает диалоговое окно печати, где можно выбрать принтер и установки печати.

Пункт *Preferences* открывает диалог предустановок программы.

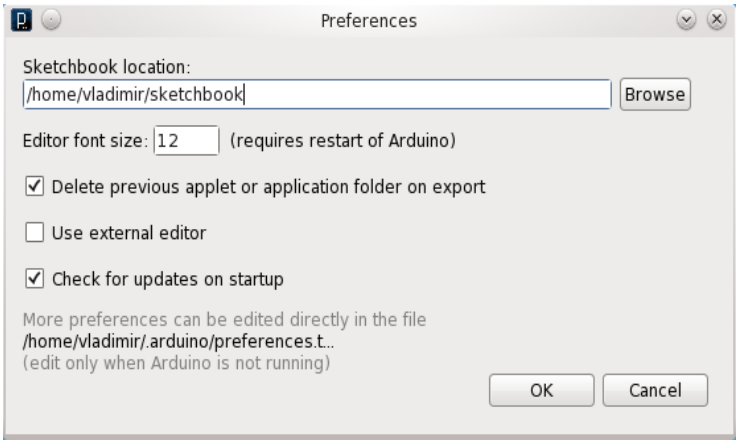


Рис. 3.7. Установки программы по умолчанию

В предустановках можно задать место расположения вашей папки с проектами (sketchbook), изменить размер шрифта и т.д. Если вы хотите использовать вместо встроенного редактора иной, скажем, более привычный для вас или более удобный по вашему мнению, вы можете установить флажок рядом с *Use external editor*. И, обратите внимание, есть указание, где можно найти файл настроек. Это обычный текстовый файл, но имеющий очень много данных для настройки программы.

Quit — выход из программы.

Следующий раздел основного меню относится к возможностям редактирования.

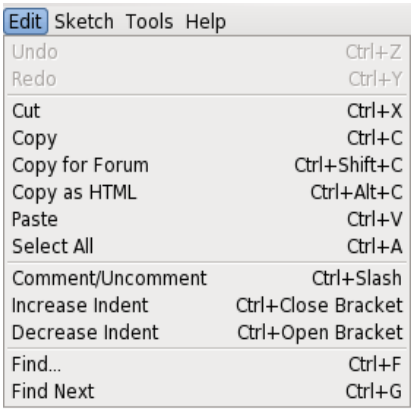


Рис. 3.8. Раздел редактирования основного меню

Как во многих программах меню контекстно-чувствительно. Первые два пункта *Undo*, отменить последнюю операцию, и *Redo*, вернуть отменённую операцию, не активны. До тех пор, пока вы не выполнили никаких операций.

Команда *Cut* вырезает выделенный текст (для выделения можно нажать левую клавишу мышки и «отчеркнуть» текст, а можно установить курсор в нужное место, нажать клавишу «Shift» на клавиатуре и использовать курсорные клавиши).

Команда *Copy* копирует текст в буфер обмена.

Команда *Copy for Forum* копирует в буфер обмена код для форума Arduino.

Copy as HTML копирует текст в формате, удобном для размещения на сайте.

Команда *Paste* вставляет текст из буфера обмена в место, указанное курсором.

Select All — удобная команда для копирования всего текста в окне редактора, например, тогда, когда вы хотите добавить выделенный код в свою программу.

Comment/Uncomment — тоже удобное добавление к командам редактирования, когда вам при

отладке программы нужно убрать на время из кода строку и вставить вновь. Вместо удаления строка будет закомментирована (или символы комментария будут удалены).

Следующие две команды *Increase Indent* и *Decrease Indent* помогают форматировать текст, сдвигая его вправо или влево.

Команда *Find...* открывает диалоговое окно для ввода искомого слова или замещения одного слова, например, имени переменной, другим. С помощью команды *Find Next* можно искать следующее появление искомого слова, если одно из них уже найдено.

Раздел основного меню *Sketch*, понятно, отсутствует в других программах. Этот раздел специфический, относится к работе программы.

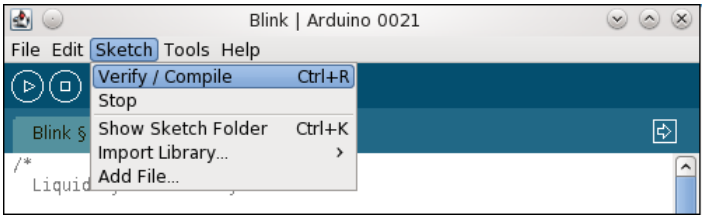


Рис. 3.9. Содержание раздела Sketch

Команда *Verify/Compile* позволяет вам скомпилировать код, который вы создали в редакторе. Команда *Stop* останавливает компиляцию.

Команда *Show Sketch Folder* открывает в системном проводнике то место, из которого был открыт файл или в котором он был сохранён.

Import Library — импортирует библиотеку.

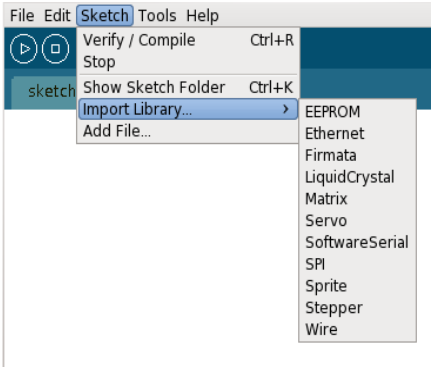


Рис. 3.10. Подраздел импорта библиотек

Add File... добавляет файл к проекту. Команда открывает встроенный проводник для выбора нужного файла.

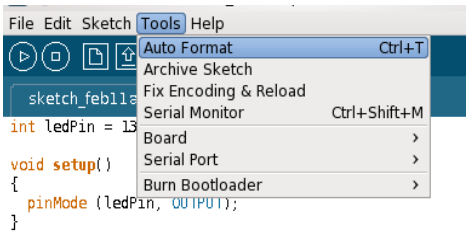


Рис. 3.11. Содержание раздела инструментов программы

Раздел *Tools* – тоже особенность программы.

Некоторые пункты этого меню открывают подменю, они отмечены стрелочками.

Команда *Auto Format* позволяет вам включить автоматическое форматирование вводимого текста программы (или выключить). Это относится к отступам, расположению скобок и т.п.

Команда *Archive Sketch* архивирует проекты.

Fix Encoding & Reload позволяет отменить все исправления и перезагрузить файл проекта.

При работе с последовательным портом очень полезно подключить терминал, чтобы видеть, как модуль Arduid работал бы, будучи подключён к компьютеру по COM-порту. Для этой цели служит встроенная терминальная программа, запускаемая командой *Serial Monitor*. Ранее об этой программе уже упоминалось.

Вот, как работает эта связка в дистрибутиве Fedora 14, а программу можно найти на сайте http://RoboCraft.ru

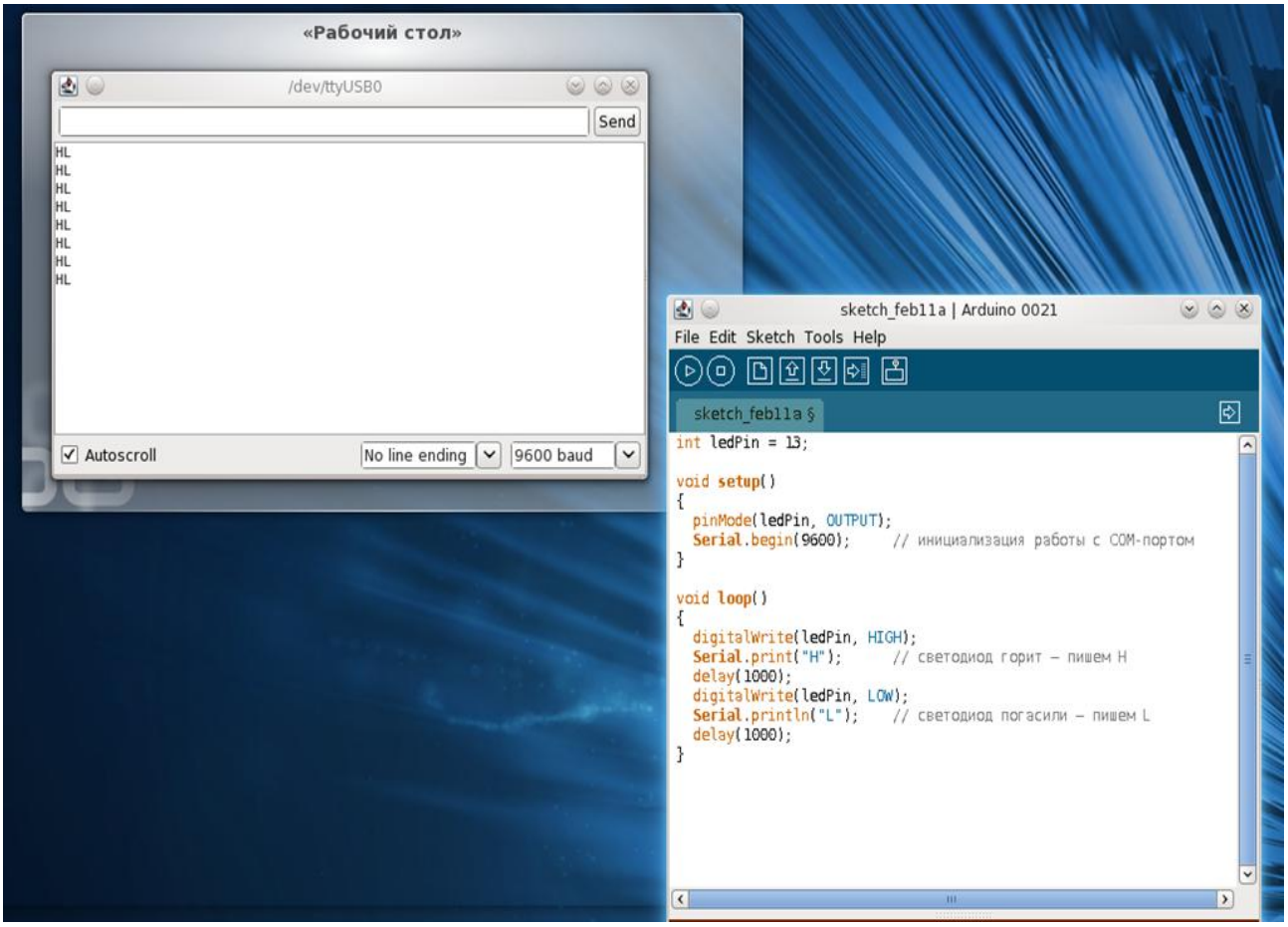


Рис. 3.12. Работа программы, использующей терминал

Программу я просто скопировал из Web-браузера, зайдя на страницу «Первая программа» сайта.

Пункт *Board* служит для выбора вашей модели модуля Arduino из списка возможных версий.

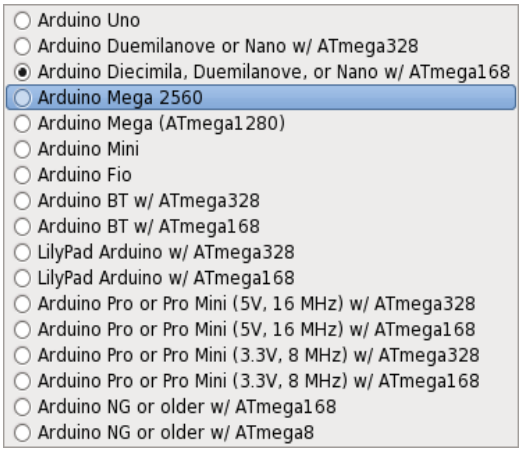


Рис. 3.13. Список модулей, с которыми работает программа Arduino

А следующий пункт *Serial Port* позволяет выбрать порт, к которому подключён модуль. Есть модели, которые подключались к COM-порту. До тех пор, пока модуль не подключён к USB-порту в Linux, вы видите только COM-порт, иначе, подключив модуль и запустив программу, вы увидите и USB-порт.

Пункт *Burn Bootloader* относится к выбору загрузчика вашего модуля Arduino и без предварительного выяснения, чем вам грозит выполнение этой команды, её, пожалуй, лучше оставить в покое (до того момента, когда вы станете с модулем и программой «на коротке»).

Раздел помощи в программе Arduin достаточно хорошо проработан.

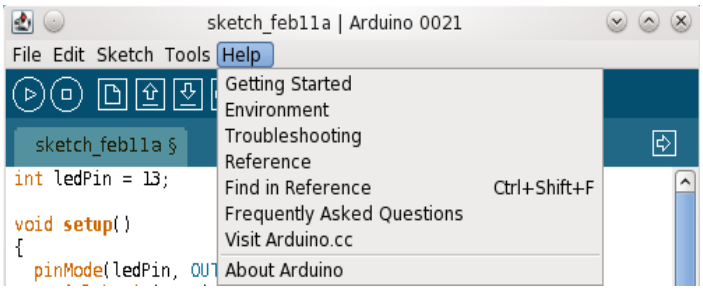


Рис. 3.14. Раздел помощи в программе Arduino

Первый пункт, *Getting Started*, открывает в Интернет-проводнике инструкции по быстрому началу работы с программой.

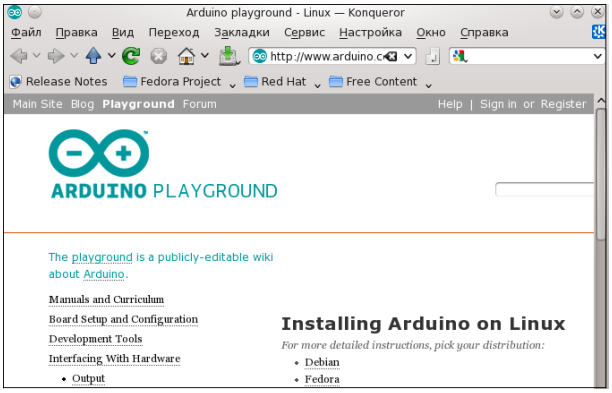


Рис. 3.15. Первый пункт раздела помощи

Я использую графическую оболочку KDE 4, в которой по умолчанию универсальный проводник Konqueror. Он и открывает страницу на сайте Arduino.

Он же открывает следующую страницу по команде *Environment*, но уже на компьютере из места установки программы. Здесь можно найти описание того, чему посвящена эта глава, но, пока, на

английском языке.

Troubleshooting — страница, загружаемая с компьютера о тех проблемах, которые могут возникнуть при работе с программой и модулем Arduino.

Reference — краткий справочник по языковым конструкциям, которые вам предстоит использовать.

Find in Reference — помогает отыскать в этом справочнике нужное место. Выделите, например, функцию и выполните команду. Откроется проводник с нужным описанием этой функции.

Frequently Asked Questions — часто задаваемые вопросы, тоже страница, расположенная на компьютере, но из опыта работы сайта проекта.

Предпоследний пункт откроет в Web-браузере главную страницу сайта Arduino, а последний выведет на экран информацию о программе.

Ниже основного меню расположена инструментальная панель программы. В основном кнопки панели повторяют наиболее часто применяемые команды, а иконки на кнопках хорошо описывают эти команды. Более того, при наведении курсора мышки на кнопку вы можете видеть в правой части панели назначение этой кнопки, как на рисунке ниже.

Рис. 3.16. Инструментальная панель программы Arduino

Слева направо назначение кнопок инструментальной панели:

```
Проверить и компилировать.
Остановить.
Новая страница.
Открыть.
Сохранить.
Загрузить в модуль.
Открыть монитор.
```

Ниже окна редактирования находится строка состояния, отображающая текущее состояние работы программы.

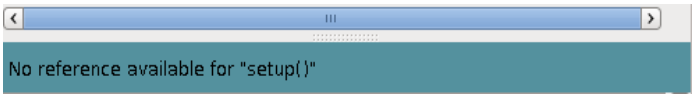


Рис. 3.17. Строка состояния

А под ней окно, в которое программа выводит сообщения, например, об ошибках.

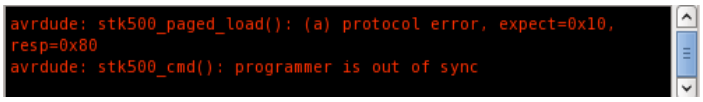
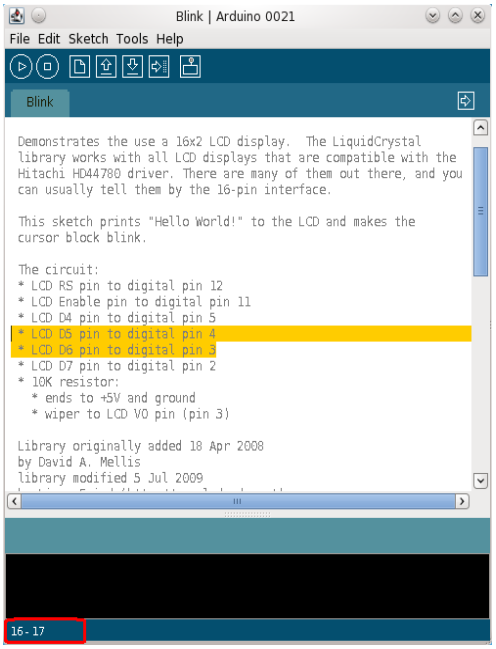


Рис. 3.18. Окно сообщений при работе программы

В самой нижней части можно увидеть, на какой строке текста находится курсор. Это весьма привлекательно, если вы отыскиваете ошибки, которые, как правило, имеют указание на строку текста программы.



Вот, пожалуй, всё, что можно сказать пока о программе. Осталось добавить, что, щёлкнув в окне редактора правой клавишей мышки, вы увидите выпадающее меню, тоже контекстно-чувствительное, повторяющее основные команды редактирования.

Рис. 3.19. Панель, указывающая номер строки в тексте программы

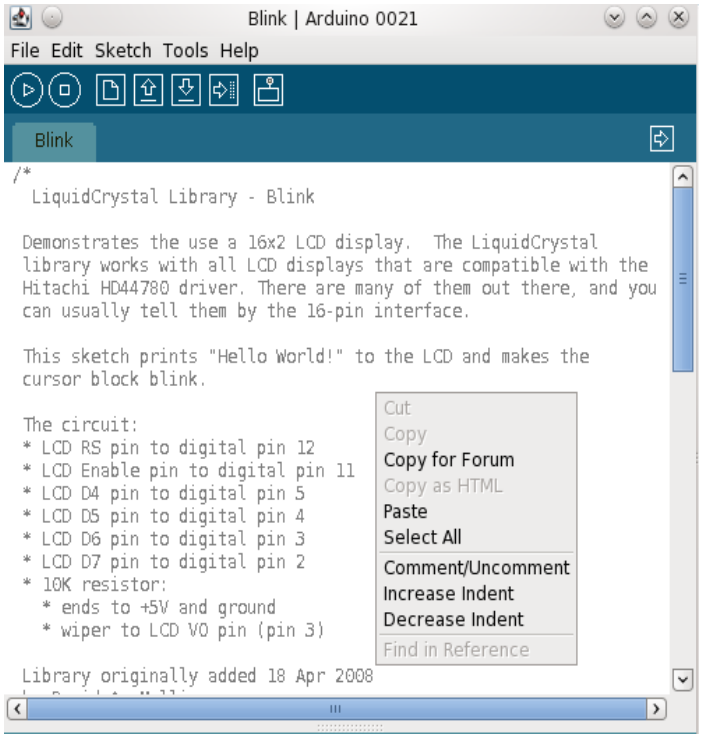


Рис. 3.20. Выпадающее меню редактора текста

Глава 4. Введение в язык программирования Arduino

Основа языка программирования модуля Arduino — это язык Си (скорее Си++). Ещё точнее, этот диалект языка называется Processing/Wiring. Хорошее обозрение языка вы найдёте в приложении. А мне хочется больше рассказать не о языке, а о программировании.

Программа — это некий набор команд, которые понимает процессор, процессор вашего компьютера или процессор микроконтроллера модуля Arduino, не суть важно. Процессор читает команды и выполняет их. Любые команды, которые понимает процессор — это двоичные числа. Это только двоичные числа и ничто иное. Выполняя арифметические операции, для которых процессор некогда и предназначался, процессор оперирует с числами. Двоичными числами. И получается, что и команды, и то, к чему они относятся, это только двоичные числа. Вот так. Но как же процессор разбирается в этой «куче» двоичных чисел?

Во-первых, все эти двоичные числа записываются в последовательные ячейки оперативной памяти, имеющие адреса. Когда вы загружаете программу, и она начинает работать, процессор получает первый адрес программы, где обязательно должна быть записана команда. Те команды, которые требуют от процессора операций с числами, имеют «опознавательные знаки», например, что в следующих двух ячейках памяти два числа, которые нужно сложить. А счётчик, назовём его счётчиком команд, где записан адрес следующей команды, в данном случае увеличивает адрес так, что в программе по этому адресу будет следующая команда. При неправильной работе программы или сбоях процессор может ошибиться, и тогда, прочитав вместо команды число, процессор делает совсем не то, что должен делать, а программа «зависает».

Таким образом, любая программа — это последовательность двоичных чисел. А программирование — это умение правильно записывать правильные последовательности двоичных чисел. Достаточно давно для записи программ стали использовать специальные средства, которые называются языками программирования.

Однако любая программа в первую очередь требует от вас ясного понимания того, что должна делать программа, и для чего она нужна. Чем яснее вы это понимаете, тем легче создать программу. Небольшие программы, хотя трудно сказать, какие программы небольшие, а какие нет, можно рассматривать целиком. Более сложные программы лучше разбить на части, которые можно рассматривать как самостоятельные программы. Так их лучше создать, легче отладить и проверить.

Я не готов спорить, но считаю, что программу удобнее начинать с описания на обычном языке. И в этом смысле я считаю, что программирование не следует путать с написанием кода программы. Когда программа описана обычными словами, вам легче определить, например, какой язык программирования выбрать для создания кода программы.

Ближе всего к записи программы с помощью двоичных чисел, язык ассемблер. Для него характерно соответствие команд языка двоичным командам, понятным процессору. Но кодирование программ на ассемблере требует больших усилий и ближе к искусству, чем к формальным операциям. Более универсальны и легче в применении языки высокого уровня, как Бэйсик или Си. И давно для записи программ в общем виде используют графический язык, а в последнее время появились и «переводчики» с этого языка на язык процессоров.

Кроме языков программирования общего применения, всегда существовала некоторая специализация языков программирования, и существовали специализированные языки. К последним я бы отнёс и язык программирования модуля Arduino.

Всё, что нужно сказать модулю, чтобы он сделал что-то нужное нам, организовано в удобный набор команд. Но вначале о том, что нам нужно от Arduino?

Модуль можно использовать в разных качествах — это и сердце (или голова) робота, это и основа прибора, это и удобный конструктор для освоения работы с микроконтроллерами и т.д.

Выше мы уже использовали простые программы для проверки подключения модуля к компьютеру. Кому-то они могут показаться слишком простыми, а поэтому не интересными, но любые сложные программы состоят из более простых фрагментов, похожих на те, с которыми мы уже знакомились.

Давайте посмотрим, о чём нам может рассказать самая простая программа «Помигать светодиодом».

```
int ledPin = 13;

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

void loop()
{
  digitalWrite (ledPin, HIGH);
  delay (1000);
  digitalWrite (ledPin, LOW);
  delay (1000);
}
```

Вначале вспомним, что такое светодиод. В сущности это обычный диод, у которого, благодаря его конструкции, при протекании тока в прямом направлении начинает светиться переход. То есть, чтобы светодиод светился, нужно чтобы через него протекал ток, а, значит, к светодиоду следует приложить напряжение. А чтобы ток не превысил допустимого значения, последовательно со светодиодом следует включить резистор, который называют токоограничительным (см. Приложение А, цифровой выход). Напряжение к светодиоду прикладывает микроконтроллер, составляющий основу модуля Arduino. У микроконтроллера, кроме процессора, выполняющего наши команды, есть один или несколько портов ввода-вывода. Не вдаваясь в рассмотрение конкретного устройства порта, скажем так — когда вывод порта работает на выход, его можно представить как выход цифровой микросхемы с двумя состояниями, включено и выключено (есть напряжение на выходе, нет напряжения на выходе).

Но этот же вывод порта может работать и как вход. В этом случае его можно представить, например, как вход цифровой микросхемы – на вход подаётся логический уровень, высокий или низкий (см. Приложение А, цифровой ввод).

Как мы мигаем светодиодом:

```
Включить выходной вывод порта.
Выключить вывод порта.
```

Но процессор работает очень быстро. Мы не успеем заметить мигания. Чтобы заметить это мигание, нам нужно добавить паузы. То есть:

```
Включить выходной вывод порта.
Пауза 1 секунда.
Выключить вывод порта.
Пауза 1 секунда.
```

Это наша программа. Процессор прочитает первую команду и включит вывод, светодиод загорится. Затем процессор сделает паузу в работе и выключить вывод, светодиод погаснет. Но он только один раз мигнул.

Повторение какого-либо процесса или набора команд называется в программировании циклом. Используются разные виды циклов. Есть цикл, который выполняется заданное число раз. Это цикл for. Есть циклы, которые выполняются до тех пор, пока не будет выполнено некоторое условие, которое является частью языковой конструкции цикла. А если условие не будет выполнено никогда, то цикл выполняется бесконечное число раз. Это бесконечный цикл.

Я не думаю, что микроконтроллеры используются с программами того вида, который приведён выше. То есть, один раз выполнено несколько команд и больше контроллер не работает. Как правило, он работает постоянно, как только на него подаётся питающее напряжение. А, значит, микроконтроллер должен работать в бесконечном цикле.

Именно об этом говорит функция void loop(), loop — это петля, замкнутый цикл. Условия прекращения работы цикла нет, а, следовательно, нет условия его завершения.

Кроме того, мы должны сообщить модулю Arduino, какой вывод порта и как мы хотим использовать, для выхода (OUTPUT) или для входа (INPUT). Этой цели служит функция void setup(), которая для языка Arduino является обязательной, даже если она не используется, и команда pinMode(), для задания режима работы вывода.

```
void setup()
{
  pinMode (ledPin, OUTPUT);
}
```

И ещё, языковая конструкция использует переменные для определения номера вывода:

```
int ledPin = 13;
```

Использование переменных удобно. Решив, что вы будете использовать не вывод 13, а 12, вы внесёте изменение только в одной строке. Особенно сильно это сказывается в больших программах. Имя переменной можно выбирать по своему усмотрению, но, как правило, оно должно быть только символьным, и часто количество символов ограничивается. Если вы неверно зададите имя переменной, думаю, компилятор вас поправит.

Функция digitalWrite (ledPin, HIGH) устанавливает заданный вывод в состояние с высоким уровнем, то есть включает вывод.

A delay (1000), как вы уже поняли, означает паузу в 1000 миллисекунд или 1 секунду.

Осталось понять, что означают такие приставки, как int, void. Любые значения, любые переменные размещаются в памяти, как и команды программы. В ячейки памяти записываются числа зачастую из 8 битов. Это байт. Но байт — это числа от 0 до 255. Для записи больших чисел нужно два байта или больше, то есть, две или больше ячеек памяти. Чтобы процессору было ясно, как отыскать число, разные типы чисел имеют разные названия. Так число по имени byte, займёт одну ячейку, int (integer, целое) больше. Кроме того, функции, используемые в языках программирования, тоже возвращают числа. Чтобы определить, какой тип числа должна вернуть функция, перед функцией записывают этот тип возвращаемого числа. Но некоторые функции могут не возвращать числа, такие функции предваряют записью void (см. Приложение А, переменные).

Вот, сколько интересного может рассказать даже самая простая программа.

Обо всём этом вы, надеюсь, прочитаете в приложении. А сейчас проделаем простые эксперименты, используя только то, что мы уже знаем из возможностей языка. Первое, заменим переменную типа int, которая занимает много места в памяти, на byte — одно место, одна ячейка памяти. Посмотрим, что у нас получится.

```
byte ledPin = 13;

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

void loop()
{
  digitalWrite (ledPin, HIGH);
```

```
  delay (1000);
  digitalWrite (ledPin, LOW);
  delay (1000);
}
```

После компиляции и загрузки программы в модуль мы не заметим изменений в работе программы. Хорошо. Тогда изменим программу так, чтобы заметить изменения в её работе.

Для этого мы заменим число в функции delay (1000) переменной, назвав её my_del. Эта переменная должна быть целым числом, то есть, int.

```
int my_del = 5000;
delay(my_del);
```

Не забывайте заканчивать каждую команду точкой с запятой. Внесите изменения в программу, скомпилируйте её и загрузите в модуль. Затем поменяйте переменную и повторите компиляцию и загрузку:

```
byte my_del = 5000;
```

Разница, уверен, получится ощутимая.

Проделаем ещё один эксперимент с изменением длительности пауз. Уменьшение длительности пауз выполним, скажем, пять раз. Сделаем паузу в 2 секунды, а затем будем увеличивать тоже пять раз. И вновь сделаем паузу в 2 секунды. Цикл, выполняемый заданное количество раз, называется циклом for и записывается он так:

```
for (int i = 0; i<5; i++)
{
что-то, что выполняется в цикле for
}
```

Для выполнения цикла ему нужна переменная, у нас это i, переменной нужно задать начальное значение, которое мы ей и присвоили. Затем следует условие завершения работы цикла, у нас i меньше 5. А запись i++ — это характерная для языка Си запись увеличения переменной на единицу. Фигурные скобки ограничивают набор команд, подлежащих выполнению в цикле for. В других языках программирования могут быть другие ограничители для выделения блока кода функции.

Внутри цикла мы выполняем то же, что и раньше, с небольшими изменениями:

```
for (int i = 0; i<5; i++)
{
      digitalWrite (ledPin, HIGH);
      delay (my_del);
      digitalWrite (ledPin, LOW);
      delay (my_del);
      my_del = my_del – 100;
}
```

Об изменении записи паузы мы говорили выше, а изменение самой паузы достигается уменьшением переменной на 100.

Для второго цикла мы запишем этот же блок кода, но переменную длительности паузы будем увеличивать на 100.

```
for (int i = 0; i<5; i++)
{
      digitalWrite (ledPin, HIGH);
      delay (my_del);
      digitalWrite (ledPin, LOW);
      delay (my_del);
      my_del += 100;
}
```

Вы заметили, что запись уменьшения паузы и её увеличения выглядят по-разному. Это тоже особенность языка Си. Хотя для ясности следовало повторить эту запись, изменив только знак минус на плюс. Итак, мы получаем такую программу:

```
int ledPin = 13;
int my_del = 1000;


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

void loop()
{
    for (int i = 0; i<5; i++)
    {
          digitalWrite (ledPin, HIGH);
          delay (my_del);
          digitalWrite (ledPin, LOW);
          delay (my_del);
          my_del -= 100;
    }

  delay (2000);
    for (int i = 0; i<5; i++)
    {
          digitalWrite (ledPin, HIGH);
          delay (my_del);
          digitalWrite (ledPin, LOW);
          delay (my_del);
          my_del += 100;
    }

  delay (2000);
}
```

Скопируем код нашей программы в программу Arduin, скомпилируем её и загрузим в модуль. Изменение длительности пауз заметно. И будет ещё заметнее, попробуйте, если цикл for выполнить, скажем, раз 8.

То, что мы сейчас сделали, делают и профессиональные программисты — имея готовую программу, её легко можно модифицировать под свои нужды или желания. Поэтому все свои программы они хранят. Что я советую делать и вам.

Что мы упустили в своём эксперименте? Мы не прокомментировали нашу работу. Для добавления комментария используется либо двойная «прямая» косая черта, либо одиночная, но со звёздочками (см. Приложение А). Я советую вам это сделать самостоятельно, поскольку вернувшись к программе через некоторое время, вы легче в ней разберётесь, если будут пояснения, что вы делаете в том или ином месте программы. И ещё советую в папке с каждой программой хранить её описание на обычном языке, выполненное в любом текстовом редакторе.

Самая простая программа «помигать светодиодом» может послужить ещё для десятка экспериментов (даже с одним светодиодом). Мне кажется эта часть работы, придумывать, что ещё можно сделать интересного, самая интересная. Если вы обратитесь к приложению, где описан язык программирования, к разделу «управление программой», то можно заменить цикл for на другой вид цикла. И попробовать, как работают другие виды цикла.

Хотя процессор микроконтроллера, как любой другой, может производить вычисления (для того его и придумывали), и это используется, например, в приборах, всё-таки наиболее характерной операцией для микроконтроллера будет установка выхода порта в высокое или низкое состояние, то есть, «помигать светодиодом», как реакция на внешние события.

О внешних событиях микроконтроллер узнаёт, в основном, по состоянию входов. Настроив выводы порта на цифровой вход, мы можем следить за ним. Если исходное состояние входа — высокий уровень, а событие вызывает переход входа в низкое состояние, то мы можем что-то сделать, реагируя на это событие.

Самый простой пример — на входе кнопка. Когда кнопка не нажата, вход в высоком состоянии. Если нажать кнопку, то вход переходит в низкое состояние, а мы можем «зажечь» светодиод на выходе. При следующем нажатии на кнопку светодиод можно погасить.

Это опять пример простой программы. Даже начинающему она может показаться неинтересной. Однако и эта простая программа может найти вполне полезное применение. Приведу только один пример: мы будем после нажатия на кнопку не зажигать светодиод, а помигаем (определённым образом). И светодиод возьмём с инфракрасным излучением. В результате мы получим пульт управления. Вот такая простая программа.

В разных версиях программы есть различия в списке примеров. Но можно обратиться к руководству по языку в приложении, где есть пример и схема программы (в разделе примеров, названном «приложение») для работы с вводом. Я скопирую программу:

```
int ledPin = 13;
int inPin = 2;

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

    void loop()
    {
          if (digitalRead(inPin) == HIGH)
          {
                digitalWrite(ledPin, HIGH);
                delay (1000);
                digitalWrite (ledPin, LOW);
                delay (1000);
          }
}
```

И, как вы видите, совершенно новую программу мы получаем, модифицируя старую. Теперь светодиод будет мигать только тогда, когда нажата кнопка, которая присоединена к выводу 2. Вывод 2 через резистор 10 кОм присоединён к общему проводу (земле, GND). Кнопка одним концом присоединена к питающему напряжению +5В, а другим концом к выводу 2.

В программе мы встречаем новую языковую конструкцию if из раздела управления программой. Читается она так: если выполняется условие (заключённое в скобках), то выполняется блок программы, заключённый в фигурные скобки. Обратите внимание, что в условии (digitalRead(inPin) == HIGH) равенство входа высокому состоянию выполнено с помощью двух знаков равенства! Очень часто в спешке об этом забывается, и условие получается неверным.

Программу можно скопировать и загрузить в модуль Arduino. Однако, чтобы проверить работу программы, понадобиться внести некоторые изменения в конструкцию модуля. Впрочем, это зависит от разновидности модуля. Оригинальный модуль имеет розетки для соединения с платами расширения. В этом случае можно вставить подходящие одножильные провода в нужные места разъёма. Мой модуль имеет ножевые контакты для соединения с платами расширения. Я могу либо поискать подходящий разъём, либо, что дешевле, использовать подходящую панельку для микросхемы в корпусе DIP.

Второй вопрос — как найти у модуля те выводы, которые используются в программе?

С этим вопросом поможет разобраться картинка, которую я взял с сайта: http://robocraft.ru/.

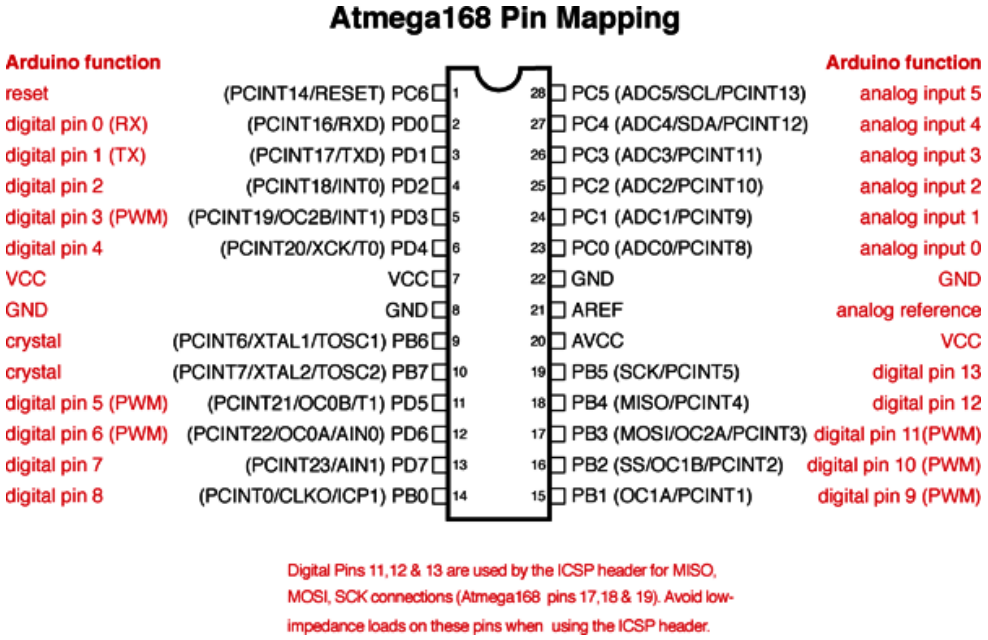


Рис. 4.1. Расположение и назначение выводов контроллера и модуля Arduino

Все выводы моего модуля CraftDuino промаркированы, так что найти нужный вывод не составит труда. Можно подключать кнопку и резистор и проверять работу программы. Кстати, на вышеупомянутом сайте RoboCraft весь процесс отображён на картинках (но программа использует не совсем такие выводы!). Советую посмотреть.

Многие микроконтроллеры в своём составе имеют дополнительные аппаратные устройства. Так Atmega168, на основе которого собран модуль Arduino имеет UART, встроенный блок для связи с другими устройствами с помощью последовательного обмена данными. Например, с компьютером через COM-порт. Или с другим микроконтроллером с помощью его встроенного блока UART. Есть ещё и аналого-цифровой преобразователь. И формирователь широтно-импульсной модуляции.

Использование последнего иллюстрирует программа, которую я тоже скопирую с сайта RoboCraft. Но программу можно взять и из приложения. И, возможно, она есть в примерах программы Arduino.

```
// Fading LED by BARRAGAN <http://people.interaction-ivrea.it/h.barragan>
int value = 0;    // переменная для хранения нужного значения
int ledpin = 9;   // светодиод подключен к digital pin 9

void setup()
{
 // Нет необходимости вызвать функцию pinMode
}
void loop()
{
      for(value = 0 ; value <= 255; value+=5) // постепенно зажигаем светодиод
      {
      analogWrite(ledpin, value); // значение вывода (от 0 до 255)
      delay(30);                  // ждём :)
      }
      for(value = 255; value >=0; value-=5)  // постепенно гасим светодиод
      {
      analogWrite(ledpin, value);
      delay(30);
      }
```

```
}
```

Если в предыдущей программе новой для нас была функция digitalRead(inPin), чтение цифрового ввода, то в этой программе новая для нас функция analogWrite(ledpin, value), хотя параметры этой функции — уже знакомые нам переменные. Об использовании аналогового входа, использовании АЦП (аналого-цифрового преобразователя), мы поговорим позже. А сейчас вернёмся к общим вопросам программирования.

Программирование это то, что доступно всем, но потребуется время, чтобы освоить и программирование, и какой-либо язык программирования. Сегодня есть ряд программ, помогающих освоить именно программирование. И одна из них имеет непосредственное отношение к модулю Arduino. Называется она Scratch for Arduino или сокращённо S4A. Найти и скачать эту программу можно по адресу: http://seaside.citilab.eu/scratch/arduino. Я не знаю, как точно переводится название программы, но «to begin from scratch» переводится, как «начать с нуля».

На сайте проекта S4A есть версии для Windows и Linux, но для последней операционной системы готовая к установке программа в версии дистрибутива Debian. Не хочу сказать, что её нельзя использовать с другими дистрибутивами Linux, но вначале посмотрим, как работать в программе с модулем Arduino в Windows.

После установки программы обычным образом можно настроить интерфейс на русский язык, используя переключатель языков.

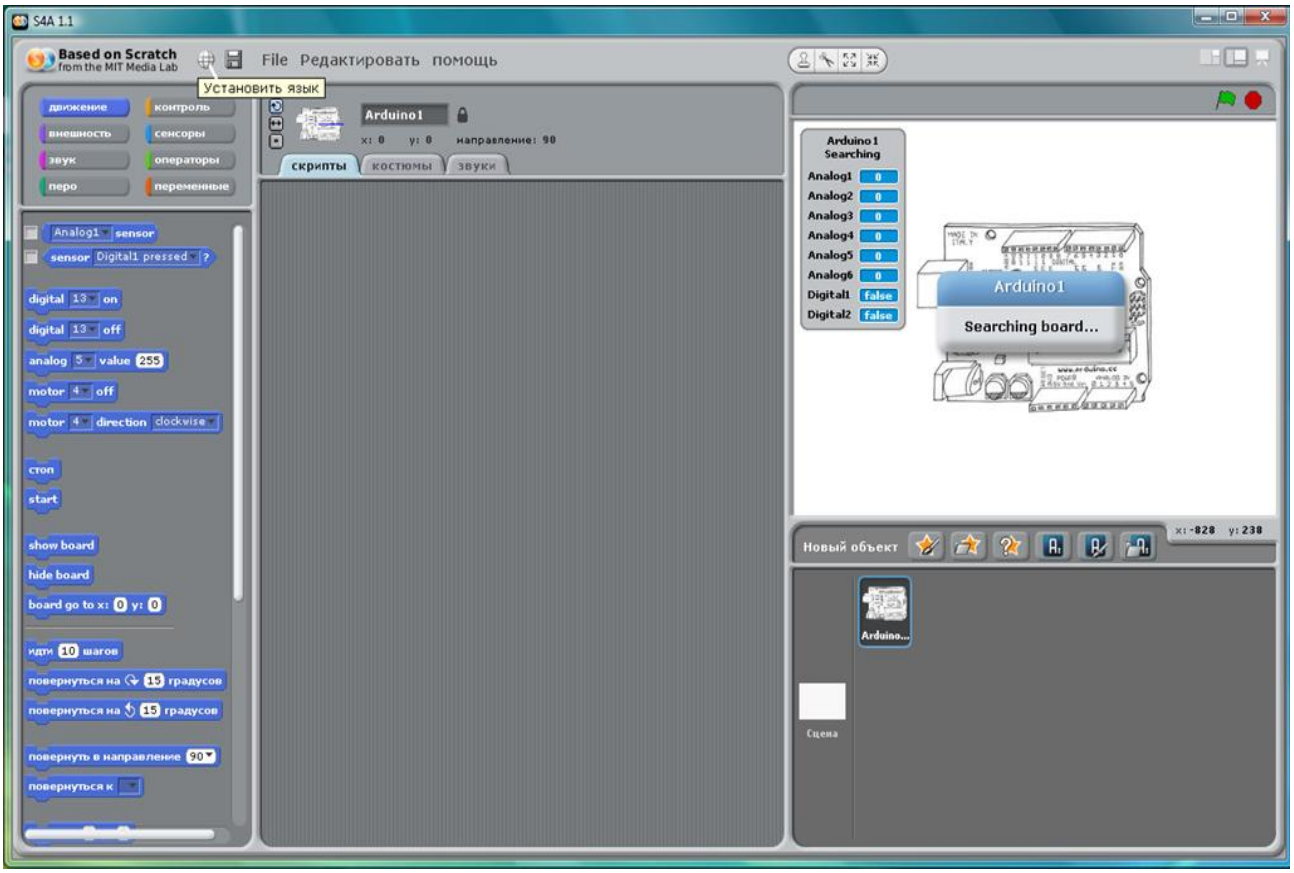


Рис. 4.2. Переключатель языков интерфейса программы

Первый значок инструментальной панели, если его нажать, отображает все возможные языки интерфейса программы. Русский язык можно найти в разделе…

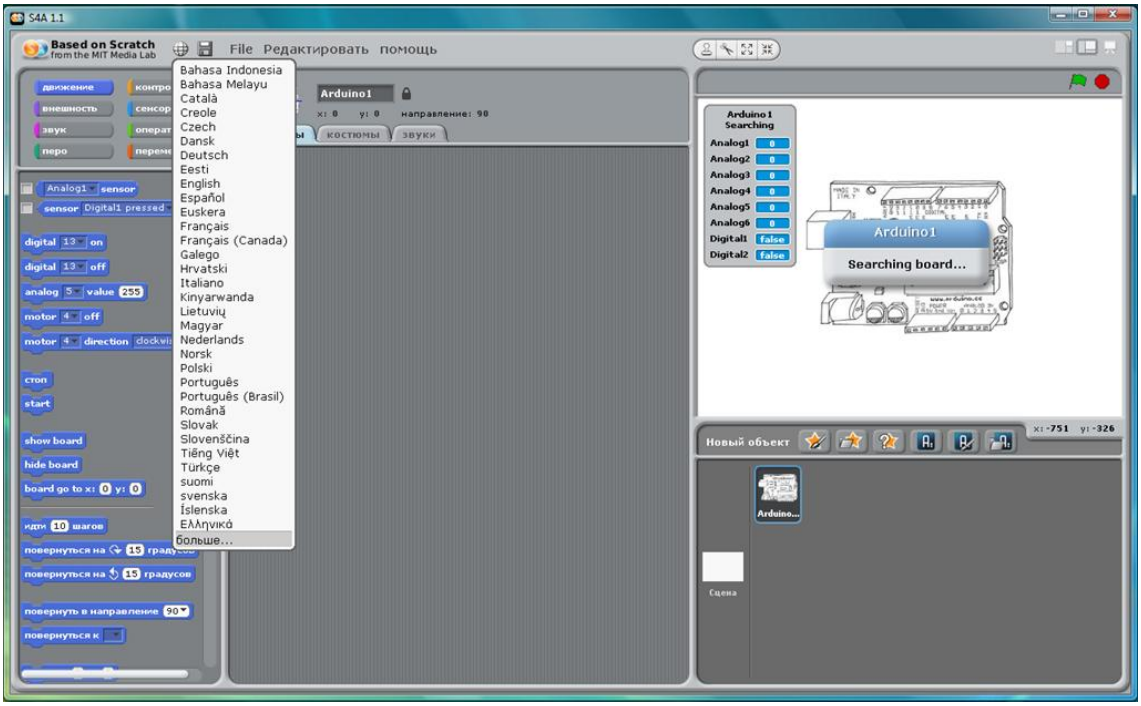


Рис. 4.3. Список языков для использования в интерфейсе программы

… отмеченном, как «больше…».

Если ничего не предпринимать, то надпись в правом окне «Searching board…» остаётся, но модуль не находится. Чтобы модуль Arduino подключить к программе S4A, следует загрузить с сайта проекта ещё кое-что.

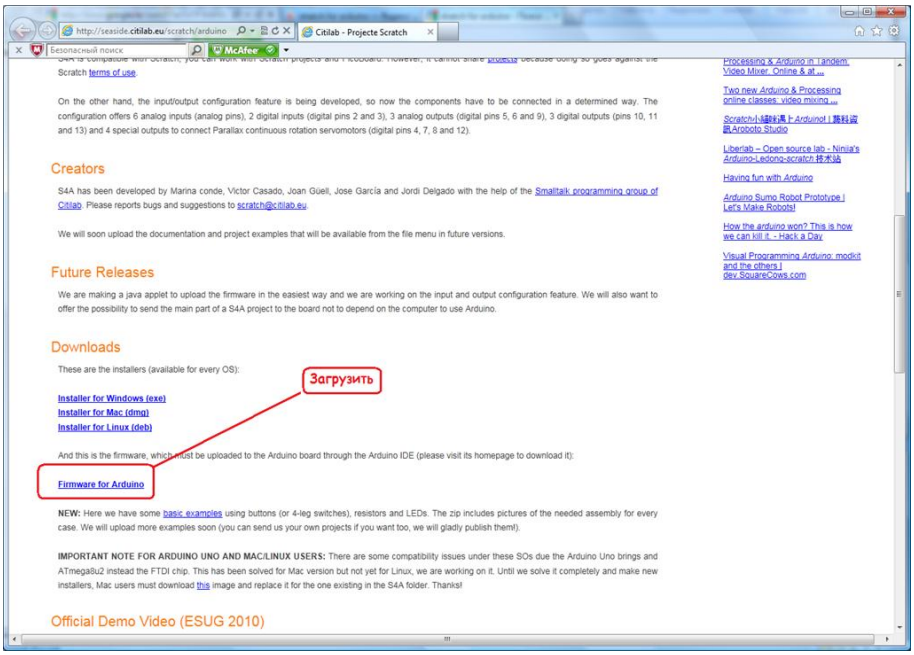


Рис. 4.4. Файл для загрузки в модуль Arduino для S4A

Этот файл не что иное, как программа для Arduino (Sketch). То есть, текстовый файл, который можно скопировать в редактор Arduino, откомпилировать и загрузить в модуль. После выхода из программы Arduino можно запустить программу S4A и теперь модуль находится.

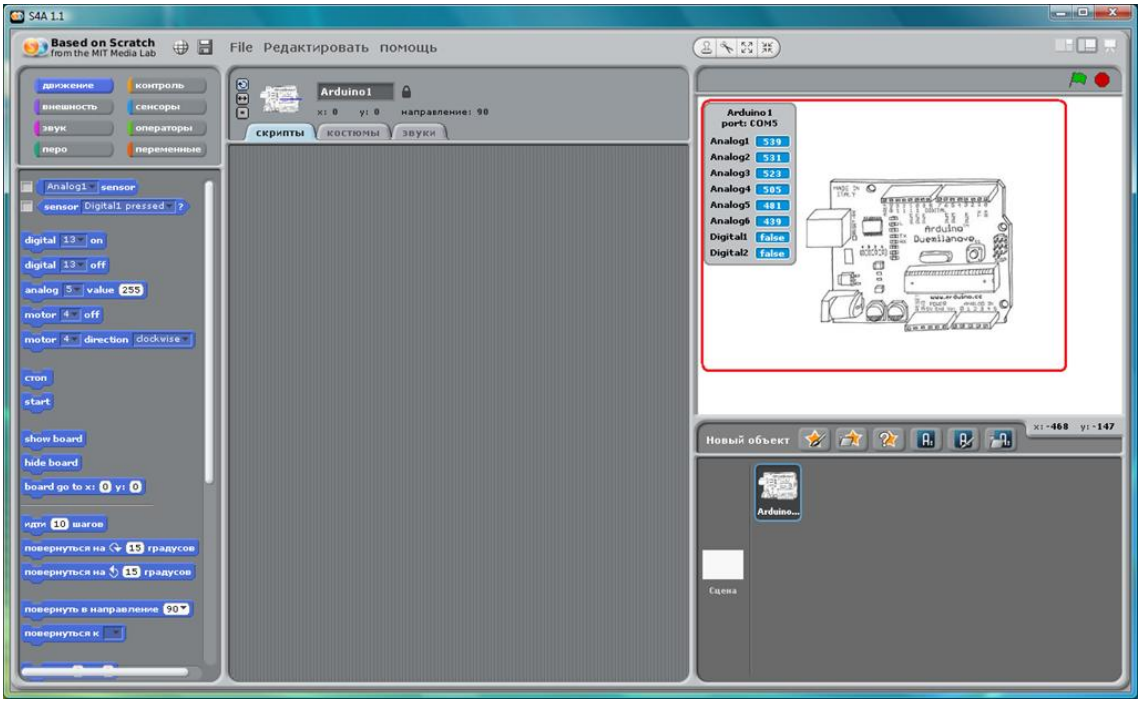


Рис. 4.5. Подключение модуля к программе

Аналоговые входы модуля не подключены, как и цифровые, поэтому значения, отображаемые для модуля, постоянно меняются произвольным образом.

Глава 5. Arduino, визуальное программирование

Возможно, правы разработчики операционных систем, считающие пользователя злейшим врагом и самым опасным вирусом. А, может быть, не правы, создают они свои творения не для себя, а для пользователей. Словом, не знаю. Но, что точно знаю, я хочу видеть работающую программу S4A не только в Windows, но в Linux, и не только в дистрибутиве Debian.

Начинаю я этот процесс с загрузки версии для Debian на сайте разработчиков: http://seaside.citilab.eu/scratch/arduino. Все загружаемые файлы располагаются по завершении загрузки в папке «Загрузка» или «Downloads». Архивированные файлы, предназначенные для Linux, распаковываются менеджером архивов. Скачанный мной в openSUSE файл имеет расширение deb, но, используя Ark, тот самый менеджер архивов, его можно распаковать. В openSUSE с графическим менеджером KDE 4 для этого достаточно щёлкнуть правой клавишей мышки по файлу и выбрать пункт выпадающего меню «Распаковать во вложенную папку». В итоге появляется папка с именем S4A.

Заглянем в неё.

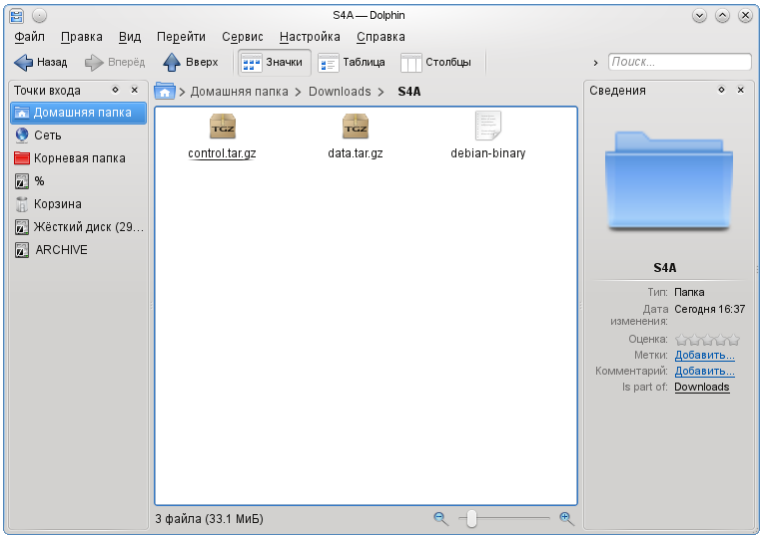


Рис. 5.1. Содержимое скачанной папки S4A

Два файла с расширением tar.gz подлежат дальнейшей разархивации.

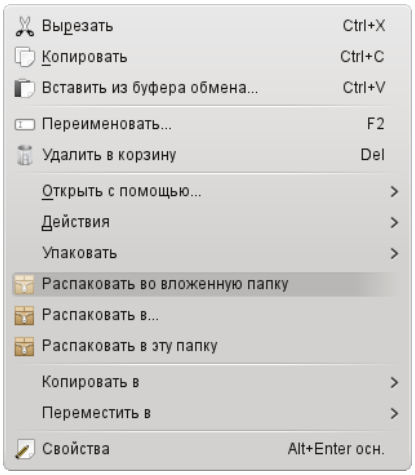


Рис. 5.2. Выпадающее меню работы с архивированными файлами

В результате рядом с архивами появляется ряд файлов и папка, озаглавленная «usr». Из опыта работы с Linux я знаю, что в этой папке могут находиться файлы, которые при установке размещаются по адресу /usr корневой файловой системы. Если открыть эту папку, то,

действительно, в ней можно увидеть еще три папки.

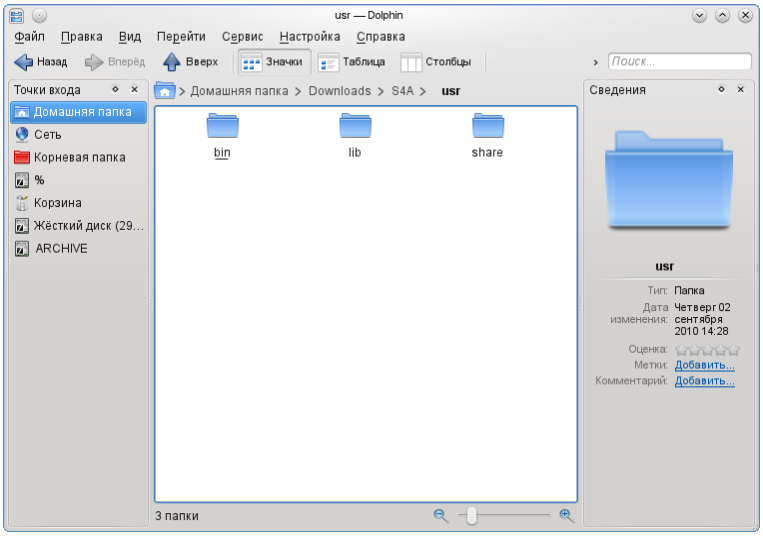


Рис. 5.3. Содержание распакованного файла

Эти три папки соответствуют разделам, которые можно увидеть, если открыть в файловом менеджере раздел «Корневая папка» в директории /usr.

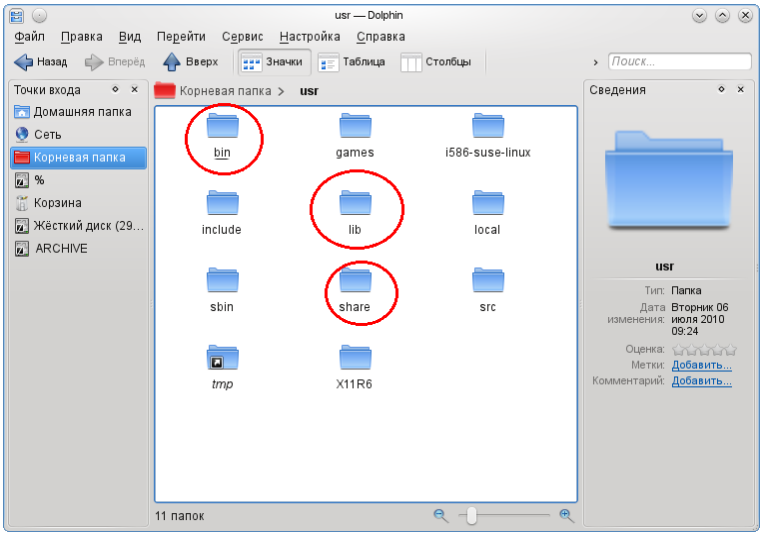


Рис. 5.4. Разделы директории usr файловой системы

Содержимое, скачанное ранее, папок bin, lib и share, как я полагаю, следует разместить в папки, отмеченные выше. Но, конечно, простому пользователю менять что-то в файловой системе никто не позволит. Поэтому в разделе основного меню «Система» находим пункт «Файловый менеджер», открывающий новое подменю, где есть «Менеджер файлов (с правами администратора)». Этот менеджер позволит перенести все нужные файлы в операционную систему. Ничего не выдумывая, открывая параллельно папки в двух проводниках, просто последовательно открывать нужные (они все названы) папки до появления файлов, а файлы копировать.

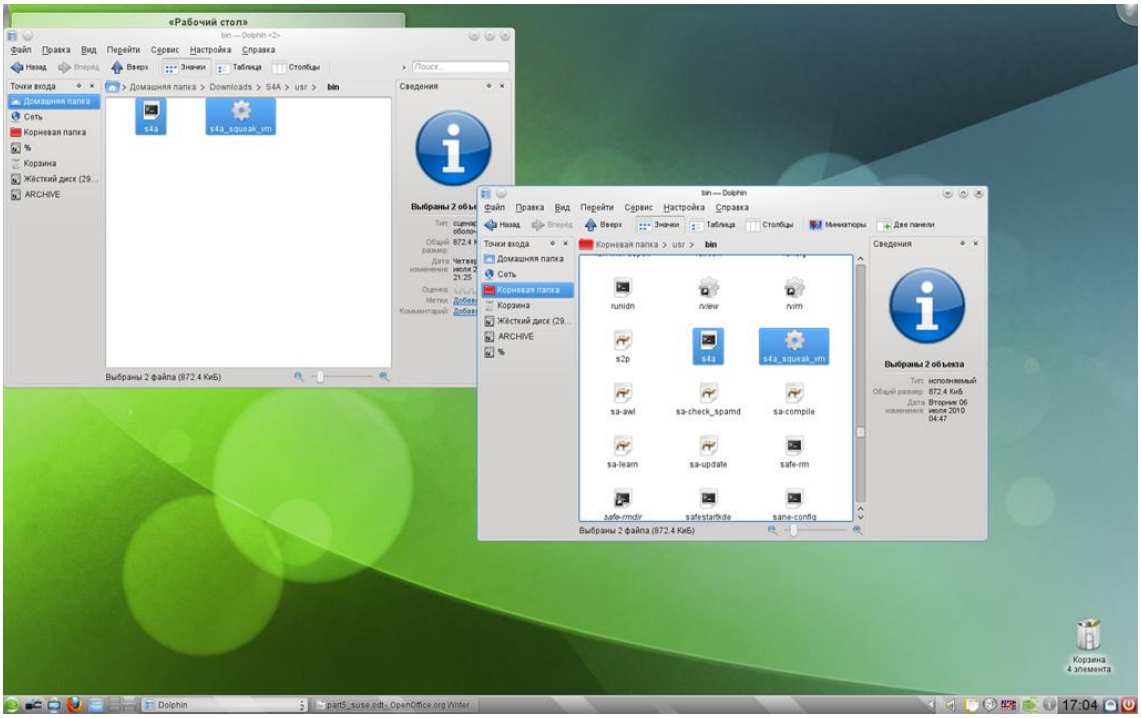


Рис. 5.5. Перенос файлов программы в openSUSE

Особенно внимательно следует отнестись к папке share, поскольку она имеет много вложенных папок, и соответствующие папки следует отыскивать в файловой системе.

Завершив копирование, можно попытаться отыскать программу в основном меню. И, впрямь, на закладке «Приложения» в разделе «Разработка» (у меня ещё один раздел «Другие программы») появляется программа S4A. И её даже можно запустить. Но она после нескольких движений мышкой начинает виснуть...

В терминале, а в openSUSE есть терминал с правами суперпользователя; от имени суперпользователя, предварительно подключив модуль Arduino, запускаем программу. И она работает. Теперь её можно запустить обычным образом.

В других дистрибутивах Linux операции схоже с теми, что описаны выше, отличия не столь значительны. Хотя в Fedora 14 я просто сменил пользователя, войдя в систему под root, что делать, конечно, не следует, но так было проще всё разместить в нужных местах.

Установив программу в Linux, посмотрим, а для чего мы её устанавливали?

Во-первых, программа работает с модулем, показывая, что происходит на аналоговых и цифровых входах модуля. Что уже неплохо. Но не это главное. Главное, во-вторых — программа позволяет собирать программу, а не кодировать на языке Arduino.

Когда программа начинает работать, в левом окне есть ряд элементов, которые можно, подцепив мышкой, перенести в среднее окно — рабочее «сборочное» поле. Перенесём так элемент, который называется Start.

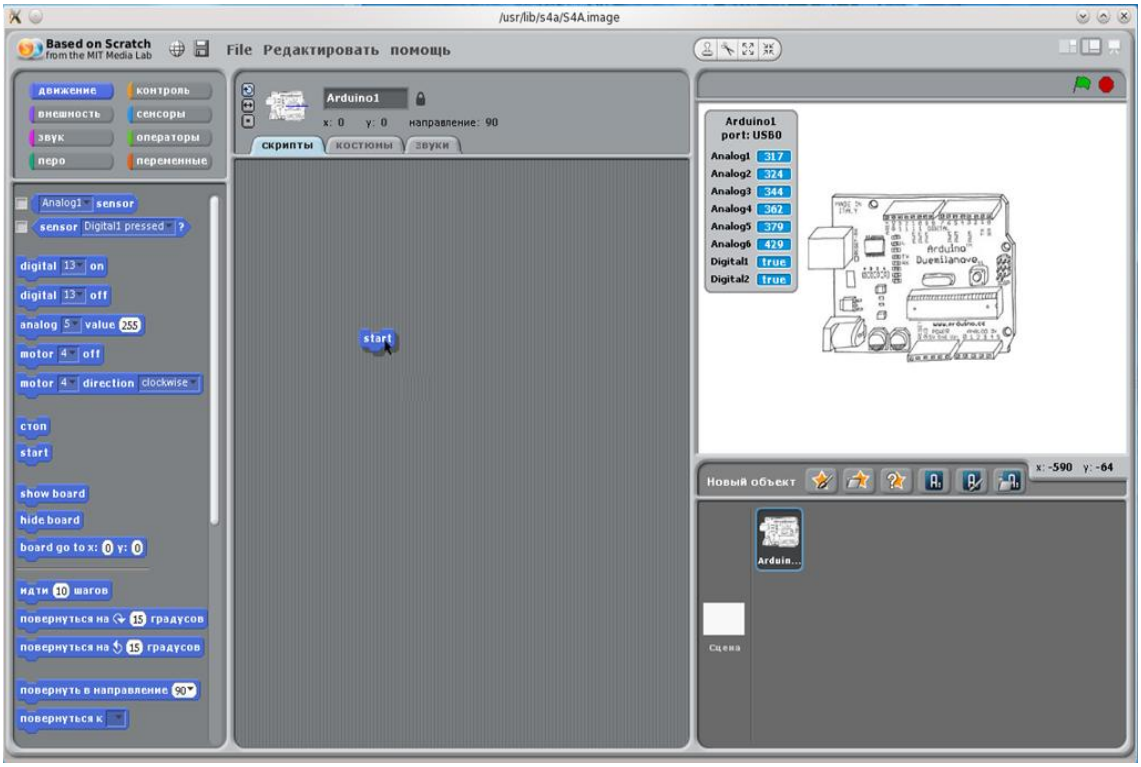


Рис. 5.6. Перенос нужных программных элементов

Теперь, нажав клавишу с надписью «контроль» в окошке чуть выше, получим ряд новых элементов.

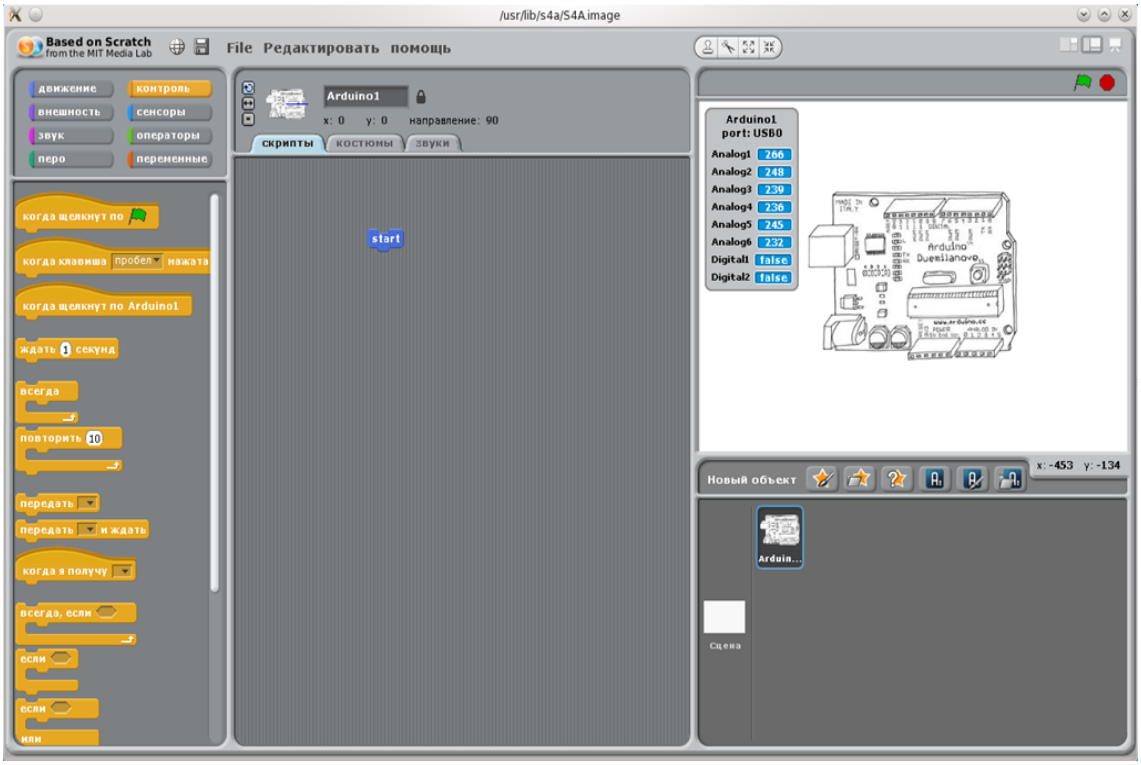


Рис. 5.7. Список элементов в группе «Контроль»

Среди этих элементов выберем элемент «всегда», который перенесём к уже имеющемуся элементу, и добавим так, чтобы верхний вырез вошёл в выступ.

Рис. 5.8. Добавление элементов в программу

Вернёмся к набору элементов, с которого начинали, нажав на клавишу «движение», и выберем элемент «digital 13 on», который перенесём и положим внутрь предыдущего.

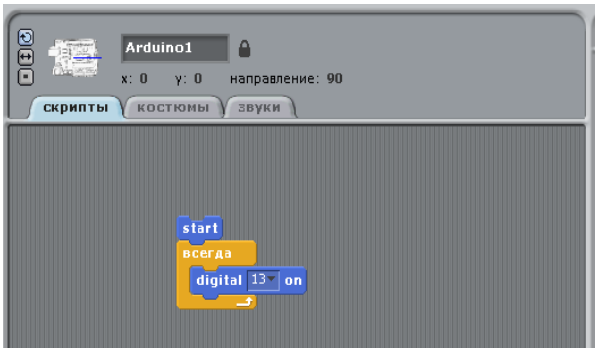


Рис. 5.9. Команда включения цифрового вывода

Из набора элементов «контроль» возьмём элемент «ждать 1 секунду», который вставим внутрь элемента «всегда» под элемент «digital 13 on». Чтобы ускорить этот процесс, вставим элемент ожидания ещё раз, вернёмся к элементам движения и добавим элемент «digital 13 off» между двумя элементами ожидания.

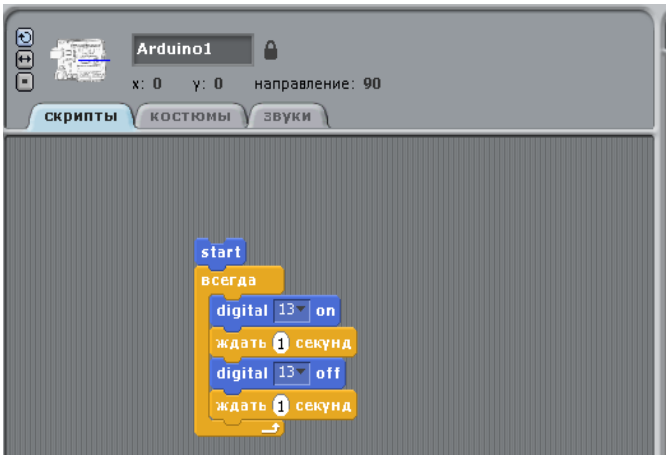


Рис. 5.10. Программа Blink в графическом виде

Вам эта конструкция ничего не напоминает? Когда мы начинали описывать первую программу обычным языком, мы так и записывали её.

Дважды щёлкните по элементу «start» левой клавишей мышки и посмотрите на модуль Arduino — молчавший до сих пор светодиод на выводе 13 исправно мигает раз в секунду.

Мы собрали программу, запустили её и заставили работать модуль согласно этой программе. И мы не написали ни строчки кода. Именно по этой причине я предпочитаю различать программирование и написание программного кода.

Но, может быть, это работает ранее загруженная программа, а не нами собранная?

Остановим работу программы, вновь дважды щёлкнув мышкой по элементу «start». Щёлкнем левой клавишей по единичке элемента «ждать 1 секунду».

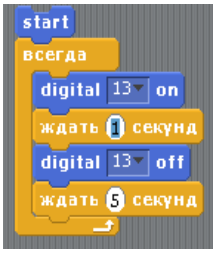


Рис. 5.11. Изменение параметров программных элементов

Впечатаем цифру 5 (как на нижнем элементе). Запустим программу... и убедимся, ничего мы не перепутали, светодиод мигает с интервалом раз в 5 секунд!

Мы не проверяли работу цифрового входа в «живом» виде. Не пора ли это сделать?

Соберём программу в S4A. Первые «кирпичики» те же, что и в предыдущей программе. Далее... нам понадобится выполнить условие: если кнопка нажата, включить светодиод, иначе выключить. Такой элемент есть — это «если... или...». В его верхней части есть «гнездо», куда можно вставить нужное нам условие «цифровой вход...».

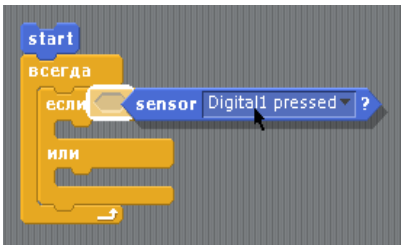


Рис. 5.12. Добавление условия в элемент if ветвления программы

Осталось добавить действия, чтобы получить нужный вид программы.

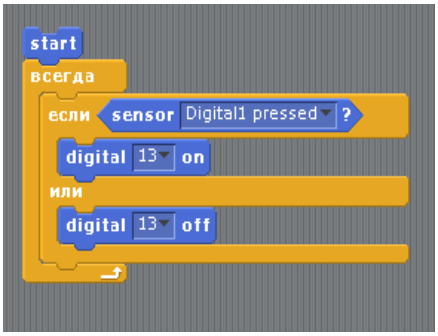
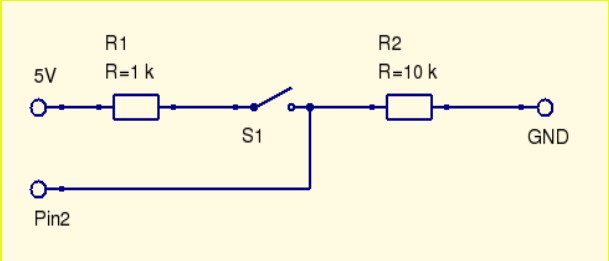


Рис. 5.13. Окончательное формирование программы

Если сравнивать её с программой, написанной на языке Arduino, то можно сказать, что отличия только те, что были внесены сознательно: когда кнопка отпущена, светодиод не горит, когда нажата, светодиод загорается.

Пора перейти к проверке. Но прежде небольшое предупреждение.

На схеме, приведённой в примерах, кнопка соединяется с выводом +5 В. Я бы советовал включить её несколько иначе.



Особенно, если вы проверяете все «на весу». При случайной ошибке может получиться так, как было у меня, из модуля пойдёт дымок, который очень подпортит настроение. А самый правильный путь — использовать макетную плату с переходными разъёмами (для Arduino Nano, думаю, найдётся подходящая панелька под микросхему).

Проверив правильность соединений на макетной плате, подключив к ней модуль Arduino, можно включить его в разъём USB компьютера и запустить программу S4A. Обратите внимание — когда вы между цифровым входом и землёй включили резистор 10 кОм, показания (в правом окошке программы) перестали случайным образом меняться между «false, ложно» и «true, истинно». Запускаем нашу программу двойным щелчком по элементу «start», добавим, зайдя в раздел основного меню «Редактировать», пошаговое выполнение.

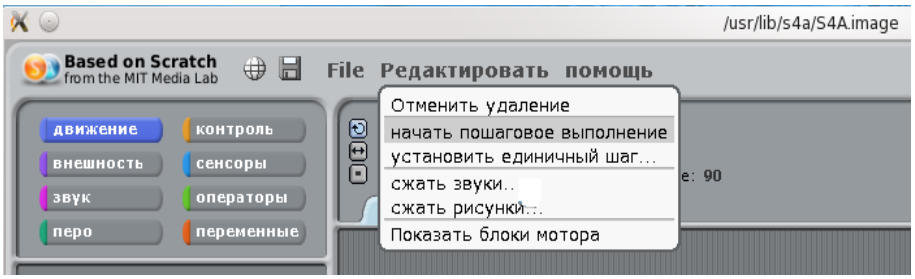


Рис. 5.14. Добавление пошагового выполнения в отладочную процедуру

Можно ещё в пункте «установить единичный шаг...» выбрать скорость выполнения. И теперь, пока кнопка не нажата, мы видим, что светодиод не горит, а программа выполняется только в той части, где это задано.

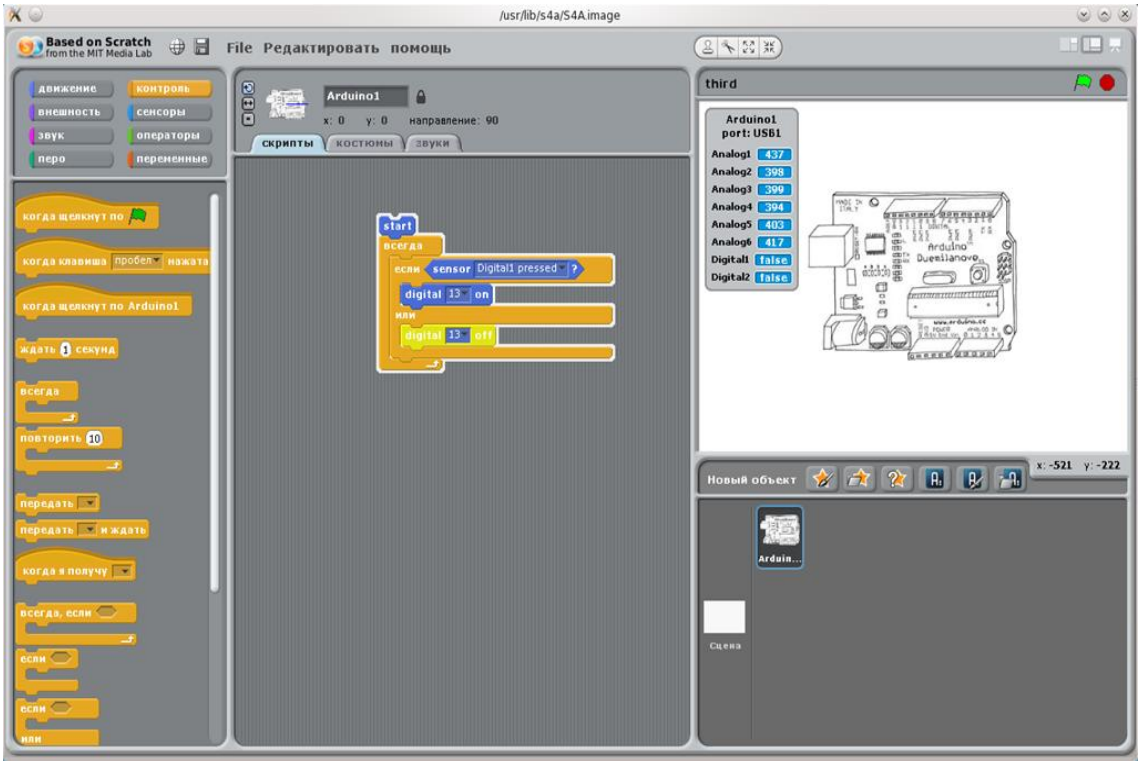


Рис. 5.15. Выполнение программы в режиме отладки

В правом верхнем окошке можно видеть состояние входа Digital1 — false. Вход на земле, на входе низкий логический уровень, а это, с точки зрения программы, состояние «ложно». Нажмём кнопку.

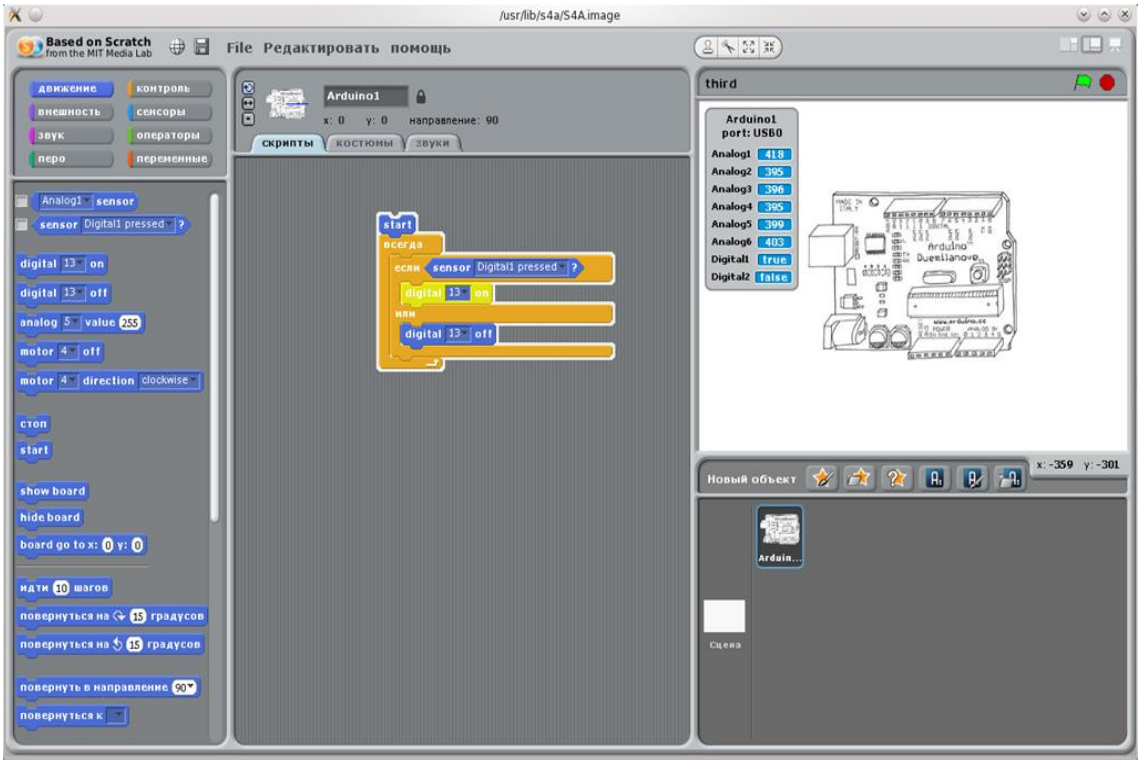


Рис. 5.16. Работа программы при нажатой кнопке

Изменилось состояние входа «true», горит светодиод, и программа входит в ту часть, где условие выполнено.

Если обратить внимание на оранжевые элементы в разделе «контроль», то видно, многие из

них имеют «гнёзда» для вставки условий.

Условия могут быть разными. Выше мы использовали в качестве условия изменение состояния цифрового входа. Но это могут быть и другие условия.

Рис. 5.17. Гнёзда для добавления условий в элементах контроля

И ещё — обратите внимание на чёрные стрелочки «вниз» рядом со многими элементами. Нажимаем эту стрелочку с помощью мышки...

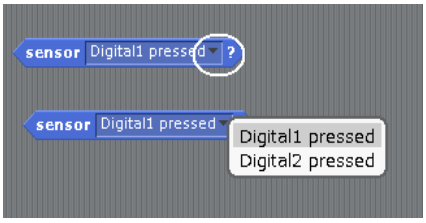


Рис. 5.18. Стрелка, открывающая список возможных сенсоров

... и получаем возможность менять, например, как в этом случае используемый вход. В других случаях меняется, скажем, выходной вывод или аналоговый вход. У нас большой выбор возможностей для экспериментов с модулем Arduino. Впрочем, отчего с модулем? Мы вправе использовать несколько модулей. Достаточно, не скажу, что это единственный способ, перейти на закладку «костюмы», щёлкнуть правой клавишей мышки по существующему «костюму» и выбрать раздел «переключиться к новому объекту».

Появится ещё один модуль Arduino. Если у вас он есть, если вы его подключили к USB порту, то можно, думаю, с ним тоже работать.

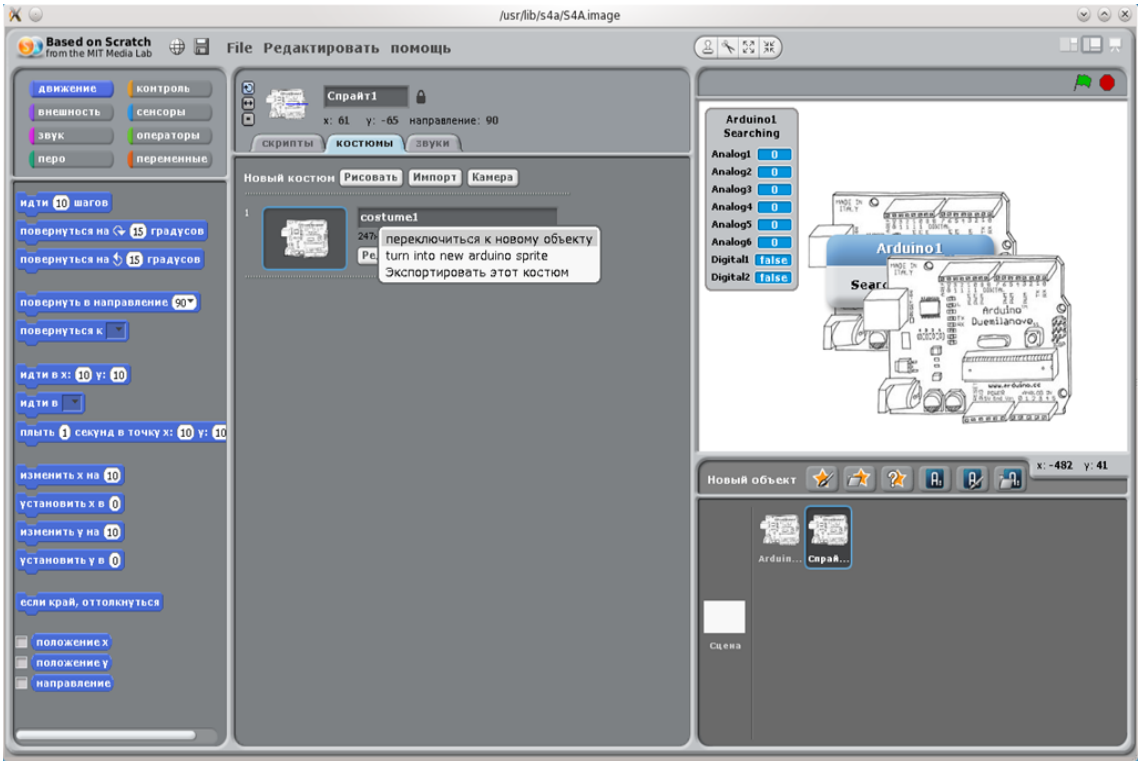


Рис. 5.19. Добавление второго модуля Arduino

И последнее замечание. Всё, что мы делаем в программе S4A, мы делаем, используя язык программирования Scratch. Как вам это?

Глава 6. Введение в язык программирования Scratch

Мы использовали программу S4A для работы с модулем Arduino. Но программа позволяет сделать много больше. Она помогает научиться программировать (не путать с написанием кода программы на традиционных языках программирования).

Для Linux, если заглянуть в раздел /usr/lib/s4a, то можно увидеть папку:

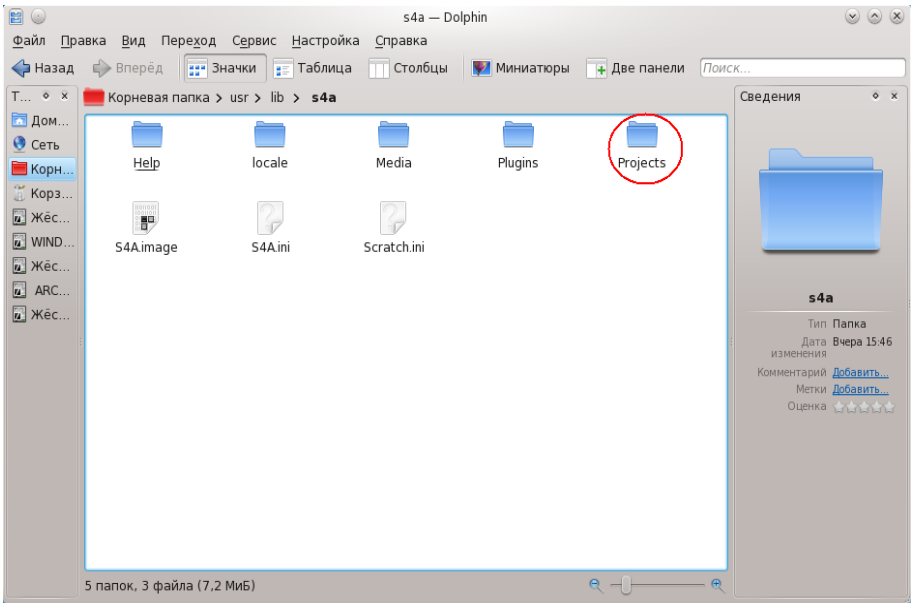


Рис. 6.1. Место расположения учебных проектов S4A

Запустим программу, выберем в основном меню раздел «File» и подраздел «Открыть...». Появится диалоговое окно выбора проекта. Используем клавишу «Компьютер», чтобы начать перемещение по файловой системе. Используем движок прокрутки справа (или колёсико мышки), чтобы перемещаться вниз, и двойным щелчком левой клавиши мышки будем открывать нужные нам папки.

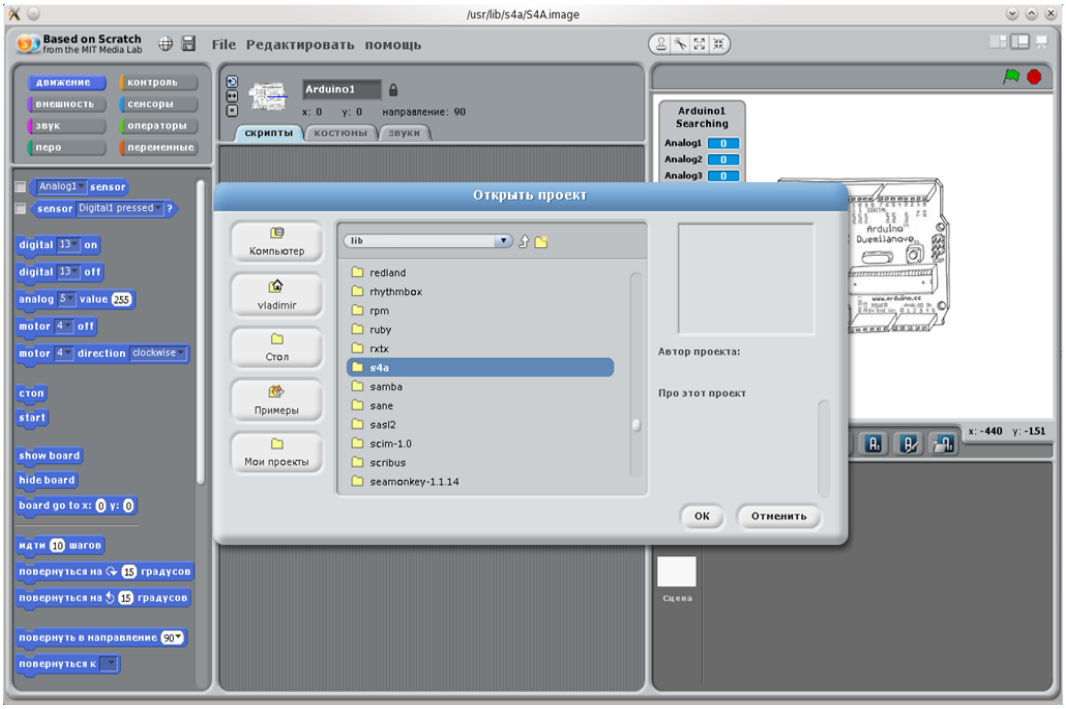


Рис. 6.2. Менеджер файлов в программе S4A

Дойдя до папки s4a, откроем её, откроем папку «Projects», в которое тоже много папок, но мы используем сейчас первую «Animation», где выберем проект Playground.

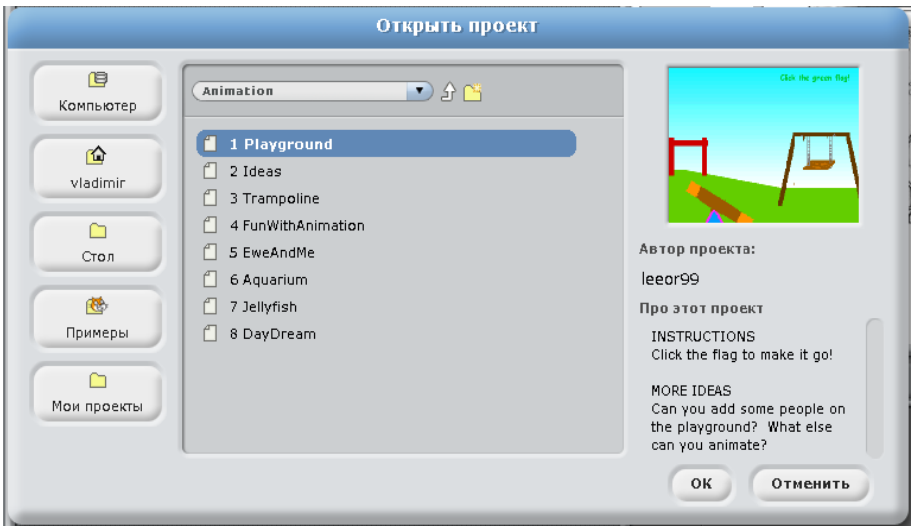


Рис. 6.3. Содержание папки примеров

Можно выбрать другой проект. И я думаю, вам интересно будет посмотреть все предложенные проекты, но для начала разберёмся с первым.

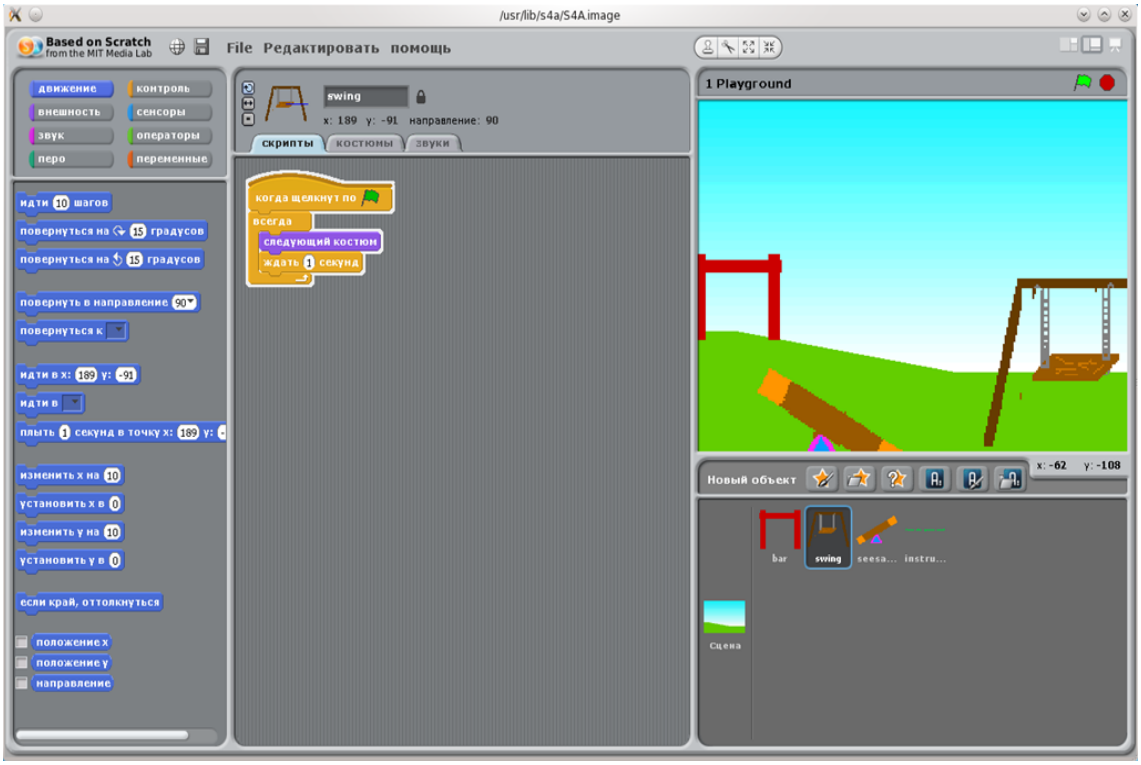


Рис. 6.4. Один из проектов примеров

Щёлкнув по флажку над правым верхним окошком, вы увидите, как качели начинают качаться. Это работы программы, которая собрана в центральном окошке, и которая начинается с условия «когда щёлкнут по флажку».

Пользователи Windows найдут аналогичные примеры по адресу: C:\Program Files\S4A\Projects.

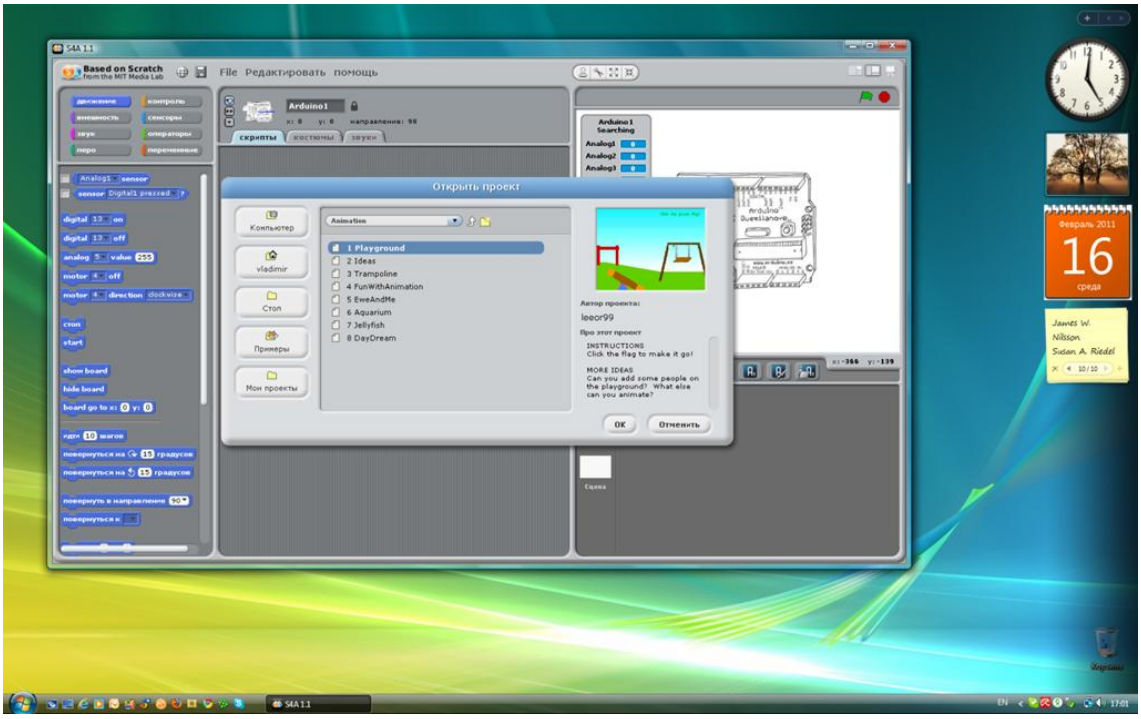


Рис. 6.5. Расположение примеров в Windows

Сразу скажу, что есть руководства по использованию языка Scratch на русском языке, которые можно найти на сайте: http://info.scratch.mit.edu/ru/Languages.

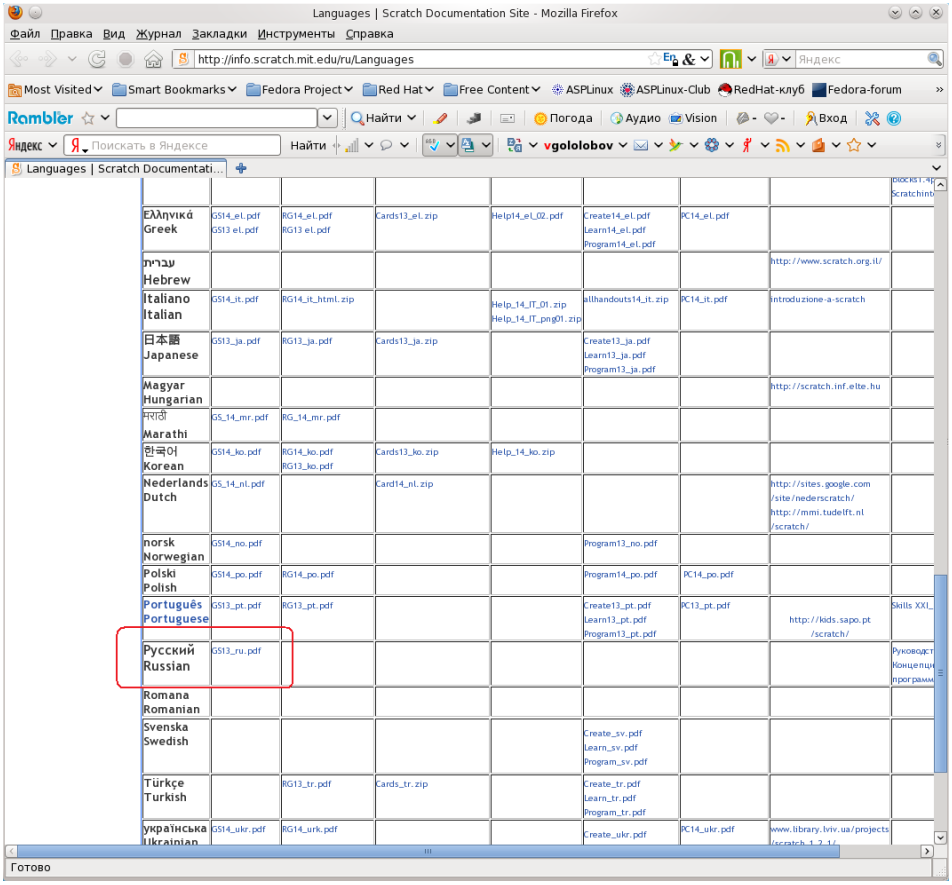


Рис. 6.6. Русскоязычные руководства к программе S4A

Я не вижу смысла пересказывать их в этой книге, но для тех, кто не найдёт эти руководства, я приведу пример, взятый из первой главы «Начнём сначала», благо есть некоторые отличия в

программах руководства и нашей S4A.

Для соответствия описания, сделанного для программы Scratch, откроем проект в папке «Greetings» с длинным названием «HellowInManyLanguages». Удалим скрипт — щелчок правой клавишей мышки по блоку «когда щёлкнут по флажку» программы и «Удалить» из выпадающего меню. Первое, что предлагается сделать в руководстве — перетащить блок «идти».

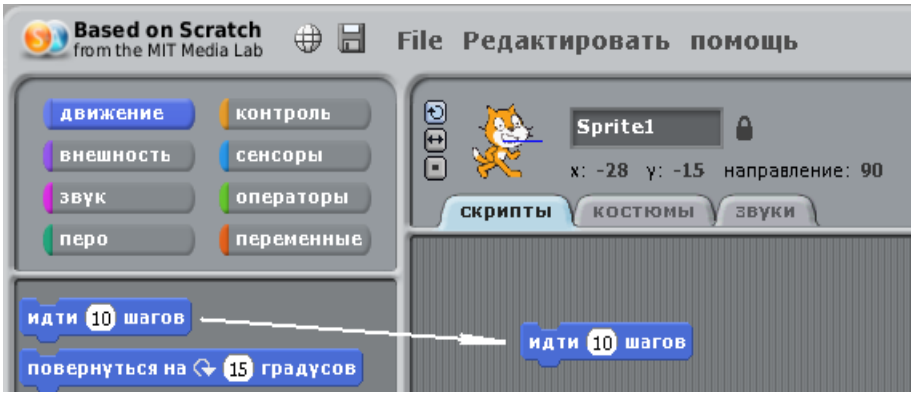


Рис. 6.7. Начало работы с программой

Если сделать двойной щелчок мышкой по этому блоку, котик сдвинется на картинке вправо. Затем из раздела «звук» перетаскиваем новый блок, который соединяем с предыдущим.

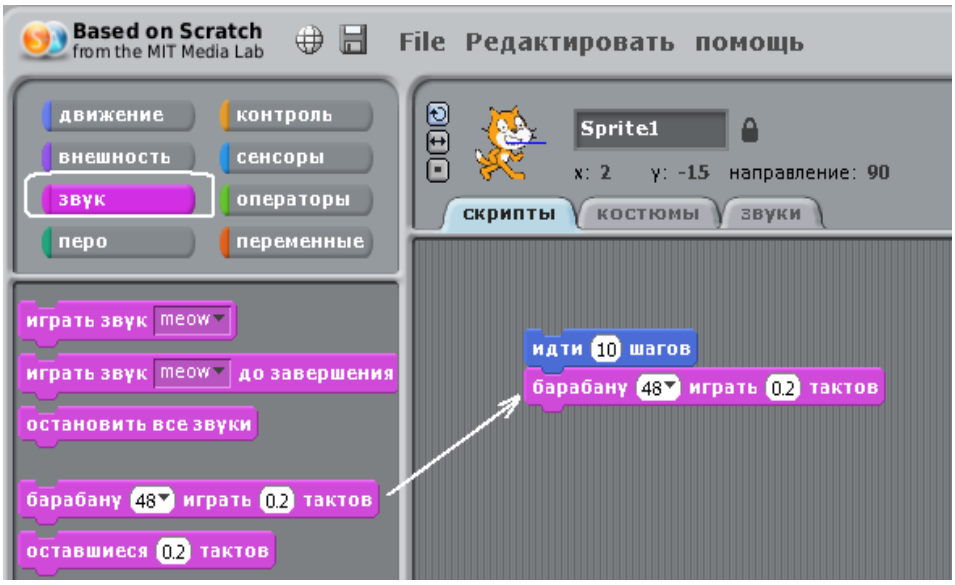


Рис. 6.8. Добавление нужных действий

Повторив двойной щелчок по любому блоку, мы услышим звук барабана. Из меню, которое можно найти, нажав на стрелочку рядом с цифрой 48, можно выбрать другой звук, например, под номером 49.

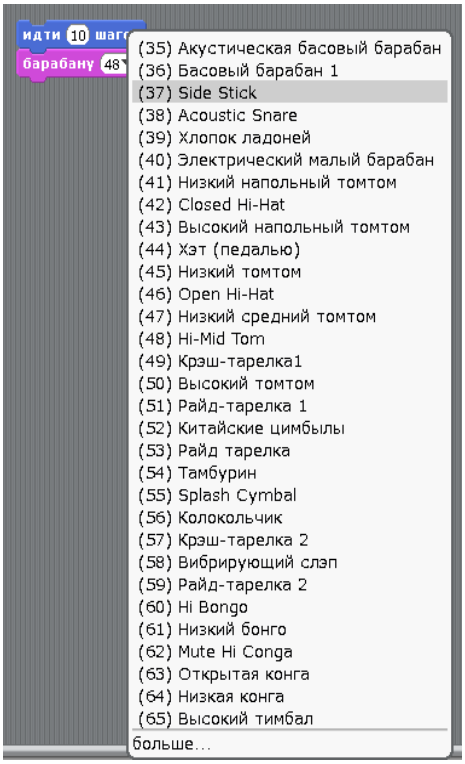


Рис. 6.9. Выбор звука барабана

Добавим к программе ещё один элемент «идти». Теперь, выделив цифру 10 внутри блока, заменим её на -10. И добавим второй блок «играть барабану», заменив барабан. В итоге получим следующую программу.

Рис. 6.10. Продолжение построения программы

Двойной щелчок по любому блоку выполнит собранные команды. Осталось обернуть это в цикл.

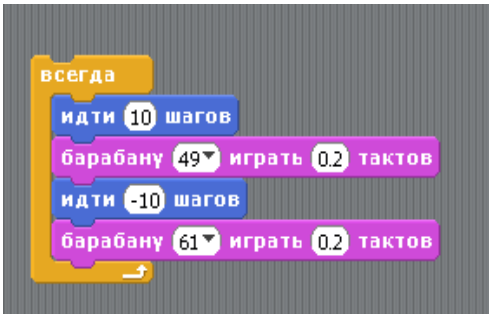


Рис. 6.11. «Зацикливание» программы

Можно добавить смену костюмов (вида) из раздела «внешность».

Запустите программу двойным щелчком по любому объекту и получите некоторое удовольствие от проделок этого котика.

О, я и не заметил, увлёкся, не заметил, как почти всё пересказал. Но почти, ещё не всё. Остальное вы сами...

Используя язык Scratch можно научиться программировать. И учиться с удовольствием, просматривая результаты работы сразу после создания программы. Однако вернёмся к модулю Arduino. Создавая свою программу, можно ошибиться. Я сейчас приведу такой пример. Вернёмся к схеме с кнопкой и светодиодом.

Мы можем рассуждать так: светодиод включается, если кнопка нажата, и выключен без нажатой кнопки. И собрать схему так, как показано ниже.

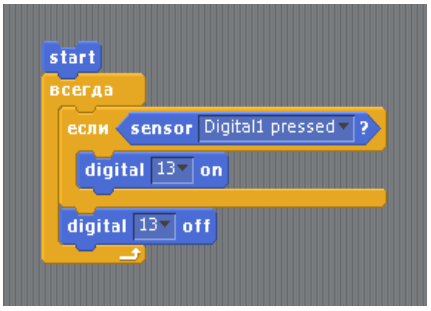


Рис. 6.12. Один из вариантов программы

Запустите программу, нажмите кнопку и посмотрите на модуль. Светодиод не горит. В чём проблема? Что мы сделали не так, как следует? Программа S4A поможет разобраться в этом. Во-первых, запустив пошаговое выполнение, мы можем убедиться, что при нажатой кнопке мы выполняем команду включения (напомню, раздел «Редактировать» пункт «Начать пошаговое выполнение»).

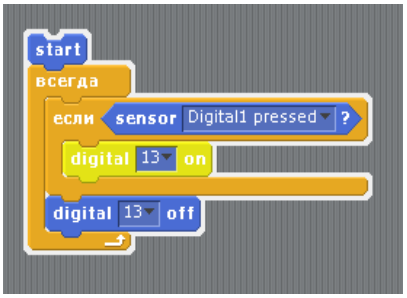


Рис. 6.13. Пошаговая проверка программы

Добавим паузы, это поможет разобраться, отчего мы не видим включения.

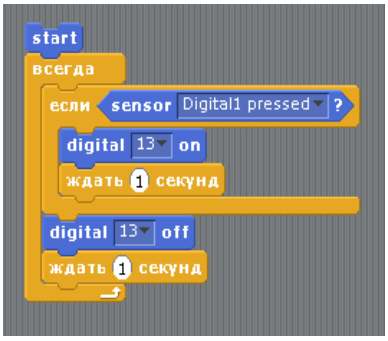


Рис. 6.14. Изменение программы для отладочных целей

Вот теперь мы видим светодиод включённым. Когда кнопка нажата, светодиод мигает с интервалом раз в секунду. А, значит, в предыдущей версии он мигает так быстро, что мы не видим этого.

То, что мы сейчас проделали, называется отладкой программы. И S4A помогает это сделать

быстро, легко и понятно.

Посмотрите, как изменится поведение светодиода, если изменить программу следующим образом.

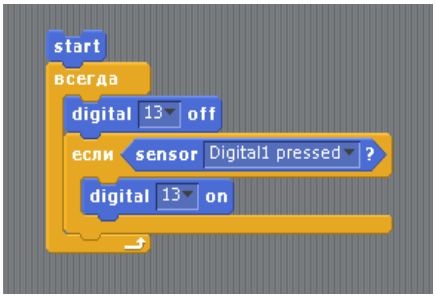


Рис. 6.15. Модификация программы

Мы подчас забываем, что процессор модуля Arduino работает очень и очень быстро. И забываем, что одни команды выполняются быстро, а другие команды, работаем ли мы с языком Scratch или Си, для процессора разбиваются на множество команд, требующих на своё выполнение времени. Отладка программы позволяет разобраться с возникающими проблемами.

Прежде, чем продолжить рассказ об отладке программ, ещё одно маленькое замечание. Хотя о состоянии цифровых входов можно узнать из сводной таблицы (рядом с изображением модуля), можно, если поставить галочку рядом с названием программного элемента (как отмечено на рисунке ниже), появится отдельное информационное окно, относящееся к цифровому входу.

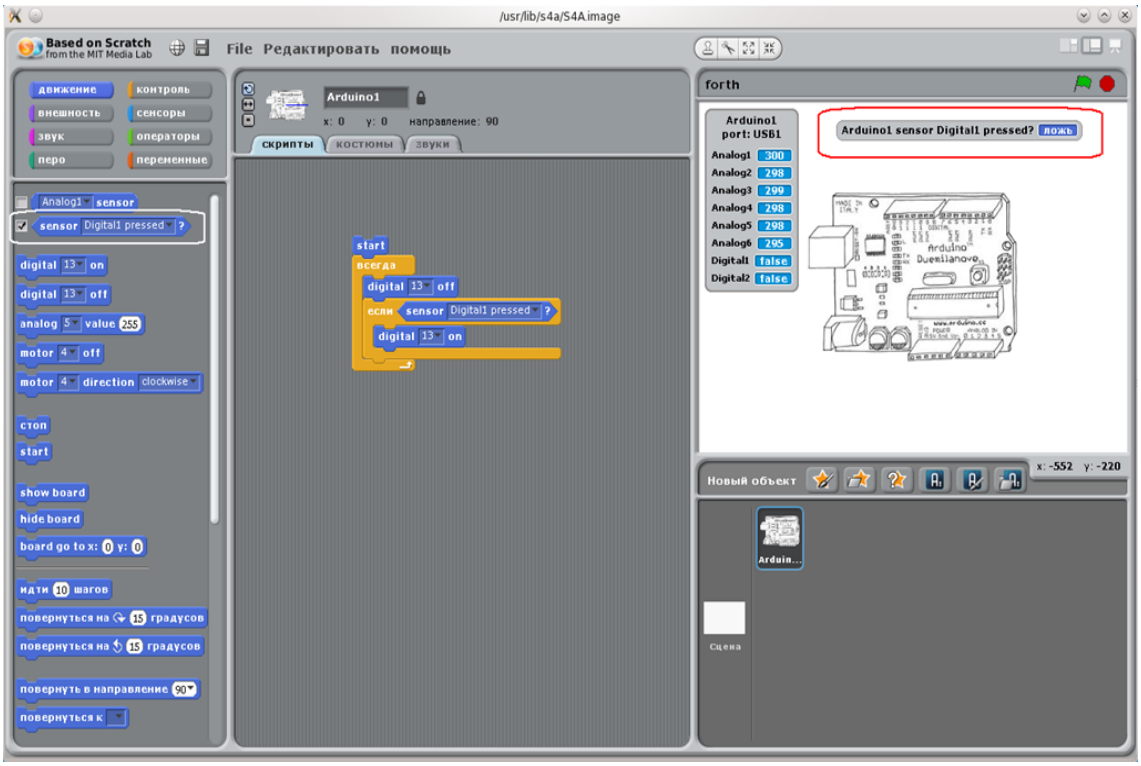


Рис. 6.16. Вынос состояния цифрового входа

Кроме программы S4A есть ещё одна, помогающая отлаживать программу, написанную для модуля Arduino. К сожалению, её версия есть только для Windows.

Глава 7. Отладка программы на виртуальной плате

Любую программу следует отлаживать. Поэтому удобство работы со средой разработки программ во многом обусловлено встроенным в неё отладчиком (debugger). Чем мощнее отладчик, тем легче проверить работу программы, и особенно это относится к программированию микроконтроллеров. О том, как можно отлаживать программу в среде разработки Arduino мы поговорим в следующих главах. О том, как научиться программировать и проверить работу программы в среде Scratch for Arduino, мы обсудили в предыдущей главе. А сейчас обратимся к ещё одной программе, которая называется VirtualBreadboard. Её можно найти на сайте проекта: http://visualbreadboard.com.

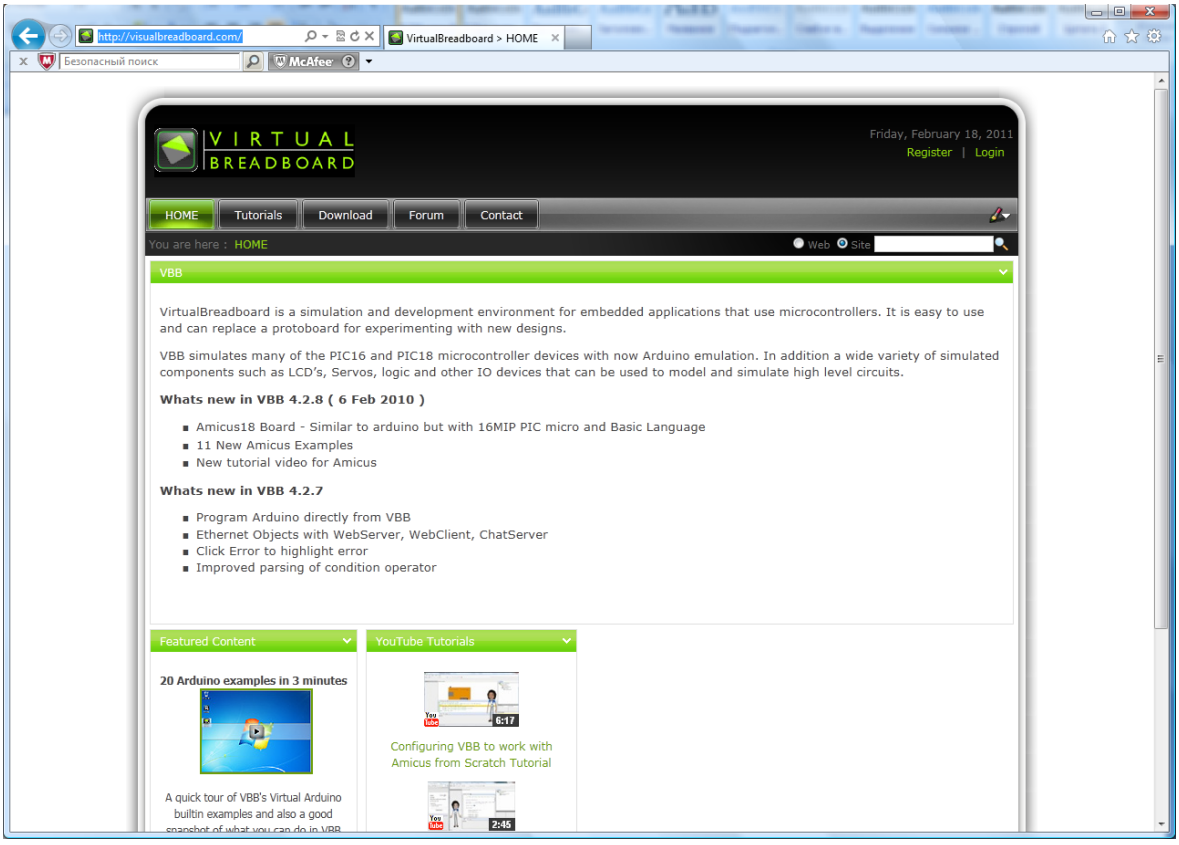


Рис. 7.1. Сайт программы виртуальной макетной платы

На первой странице вы найдёте много примеров работы программы в виде видеоуроков. Перейдя на страницу Download, вы можете загрузить программу. К сожалению, на момент написания этой главы версии для Linux нет. И попытка запустить программу в Linux под Wine закончилась неудачей.

Чуть позже мы поговорим о том, как установить программу, а сейчас несколько примеров из набора, предлагаемого автором. После запуска программы появляется диалоговое окно выбора с несколькими закладками: использовать примеры (открыта), использовать существующий проект или обратиться к тем, с которыми работали недавно.

Рядом с разделом Arduino есть плюсик, щёлкнув по которому, вы можете открыть примеры, созданные автором проекта.

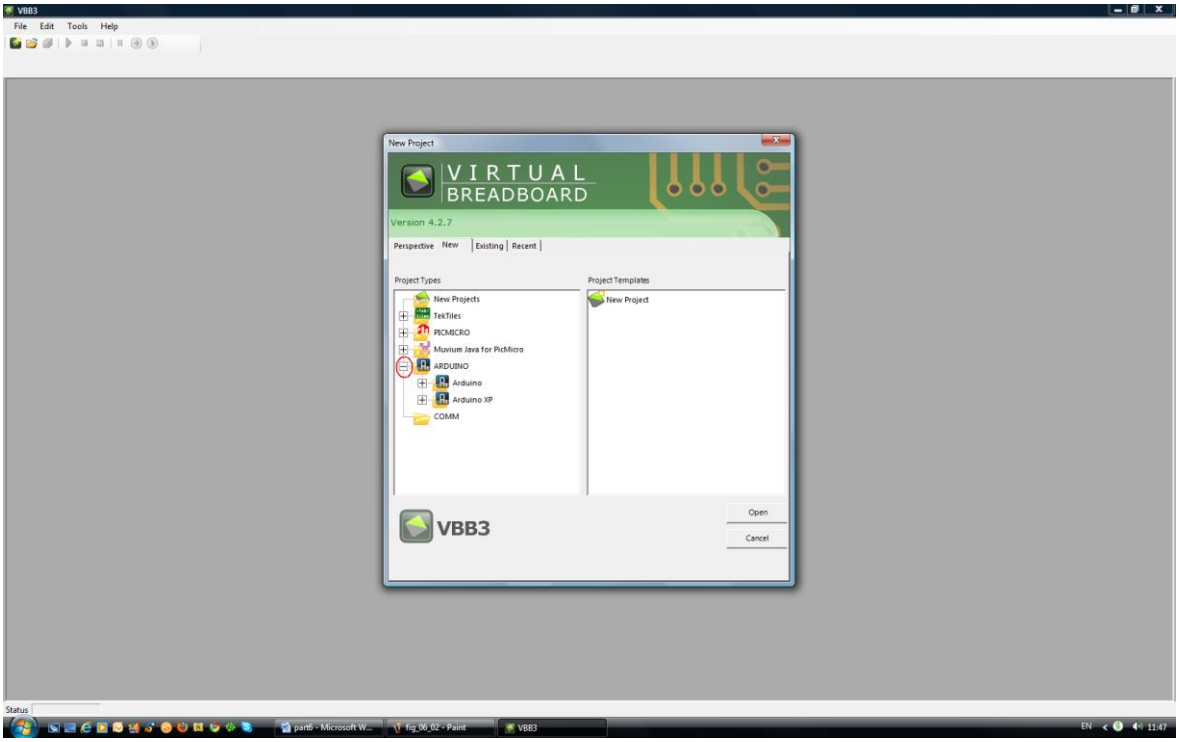


Рис. 7.2. Начальный диалог программы

Перемещаясь по дереву примеров, можно выбрать разные проекты, как, например, такой интересный проект: после запуска программы, если перемещать движок потенциометра, мигание светодиода меняет свой характер.

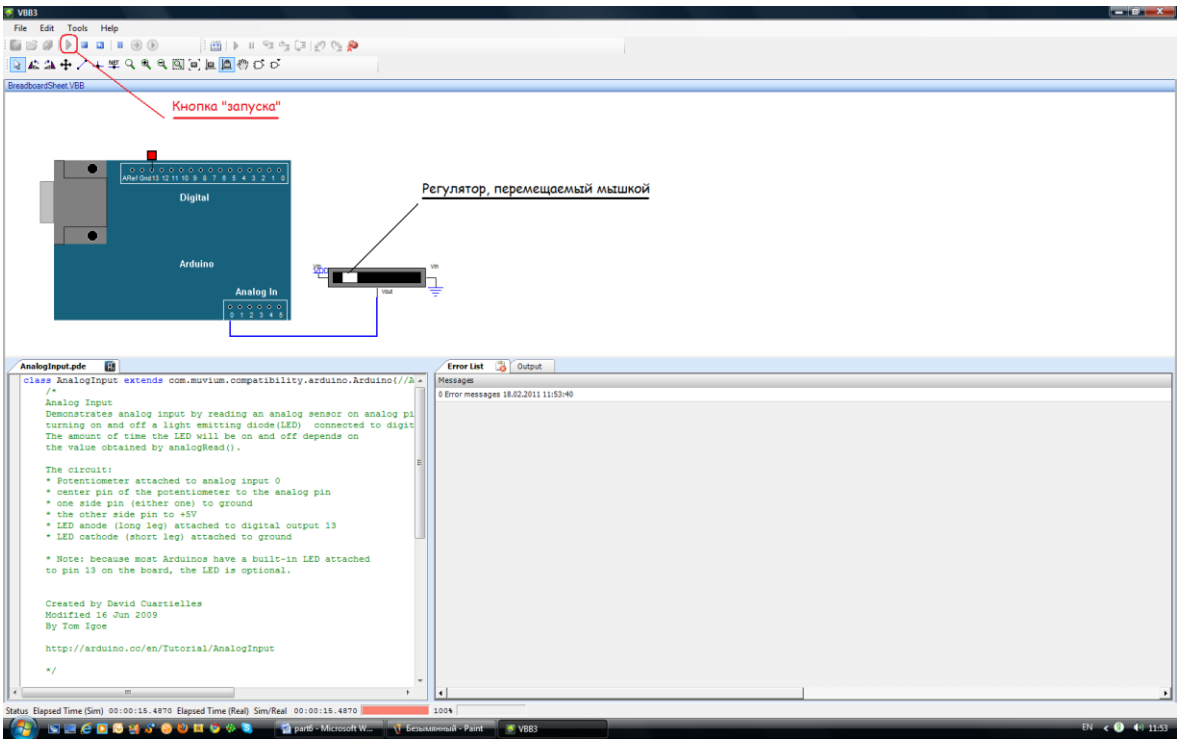


Рис. 7.3. Один из примеров работы программы

Или такой проект.

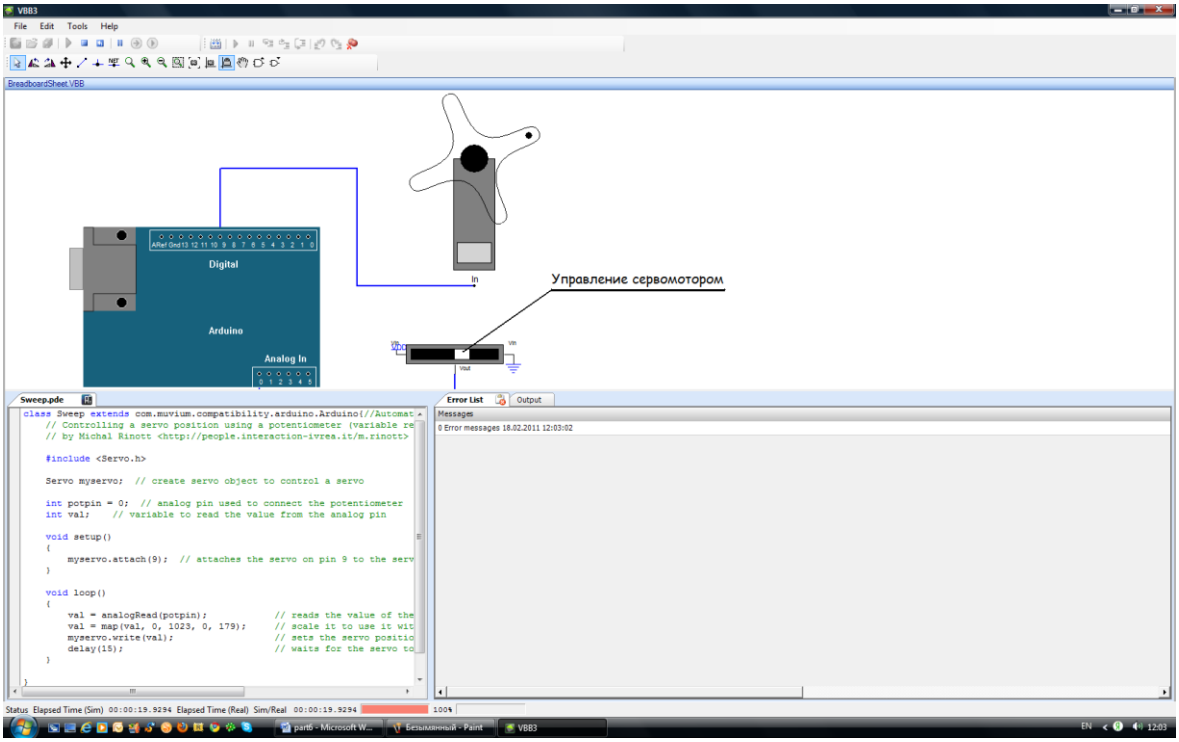


Рис. 7.4. Ещё один проект из набора примеров

Перемещая движок потенциометра, вы управляете поворотом сервопривода. Словом, первое, что я советую сделать после запуска программы – посмотреть приведённые для модуля Arduino примеры.

Теперь об установке программы. Собственно, сама программа не требует установки – после скачивания её можно разместить в удобном месте и запускать. На сегодняшний день есть версия VBB 4.2.8, но я советую сегодня использовать версию VBB 4.2.7.

Для использования программы следует дополнительно загрузить некоторые вспомогательные программы, перечень которых есть на странице загрузки. Вот, что сказано об этом:

…программа может располагаться на рабочем столе или в другом удобном месте.
Однако VirtualBreadboard имеет следующие зависимости, которые нужно дополнить установкой на вашем компьютере. Они уже могут быть установлены на вашем компьютере, но могут потребовать установки до VBB.

- Latest version of DirectX. Click here to download from Microsoft
- .Net 2.0 Redistributable. Click here to download from Microsoft
- J# 2.0 Redistributable. Click here to download from Microsoft
- Java JRE 1.6 (version 6). Click here to download from Oracle

Выделенное мною указание «Click here» – это ссылка на загрузку. Щёлкнув по ней, вы попадёте в нужное место, где можно найти всё необходимое для загрузки.

Посмотрев видеоролик на сайте проекта, я создал папку с именем arduino в своей директории, куда и отправил программу. Забегая вперёд, в ту же папку я распаковал и программу Arduino. И, опять таки на сегодняшний день, я советую, хотя использовал версию программы Arduino 0022, для совместной работы применить более раннюю версию Arduino 0018.

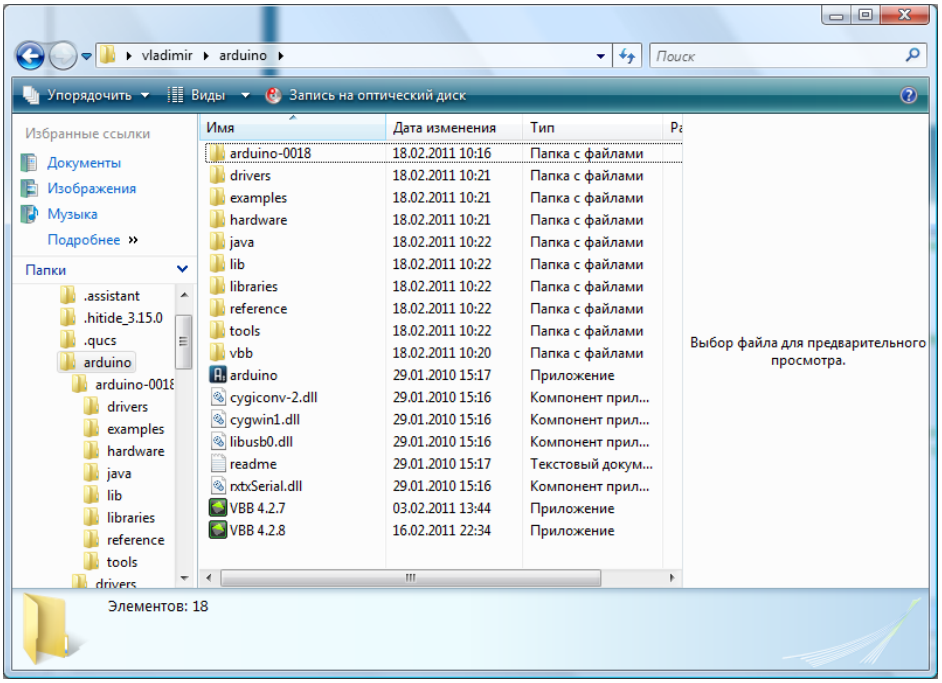


Рис. 7.5. Выбор версии Arduino для работы с программой

Сочетание, например, vbb-4.2.8 и arduino-0022 мне в операционной системе Vista SP2 заставить работать должным образом не удалось. Это относится к использованию возможности программы VirtualBreadboard программировать модуль Arduino непосредственно, без перехода в программу Arduino. В остальном же, видимо, не возникнет проблем и при использовании последней на сегодня версии.

Проделав все загрузки программ, собрав всё нужное в удобном вам месте, вы можете столкнуться с ещё одной проблемой при попытке загрузить программу в модуль Arduino.

Кстати, ещё одна функция программы весьма полезная. К полезным функциям я отнёс бы и возможность работать с разными модулями, и возможность работать с PIC-контроллерами. Я не советую начинающим «прыгать» от модуля к модулю, от контроллера одного производителя к контроллеру другого. Но это будет полезно, когда вы освоитесь с первыми (и последующими) шагами и захотите создавать свои конструкции, разрабатывать свои программы.

Вернёмся к установке и работе с программой VirtualBreadboard, которую я считаю очень полезной, особенно для начинающих. Потратив некоторое время на её установку, вы не пожалеете. Итак.

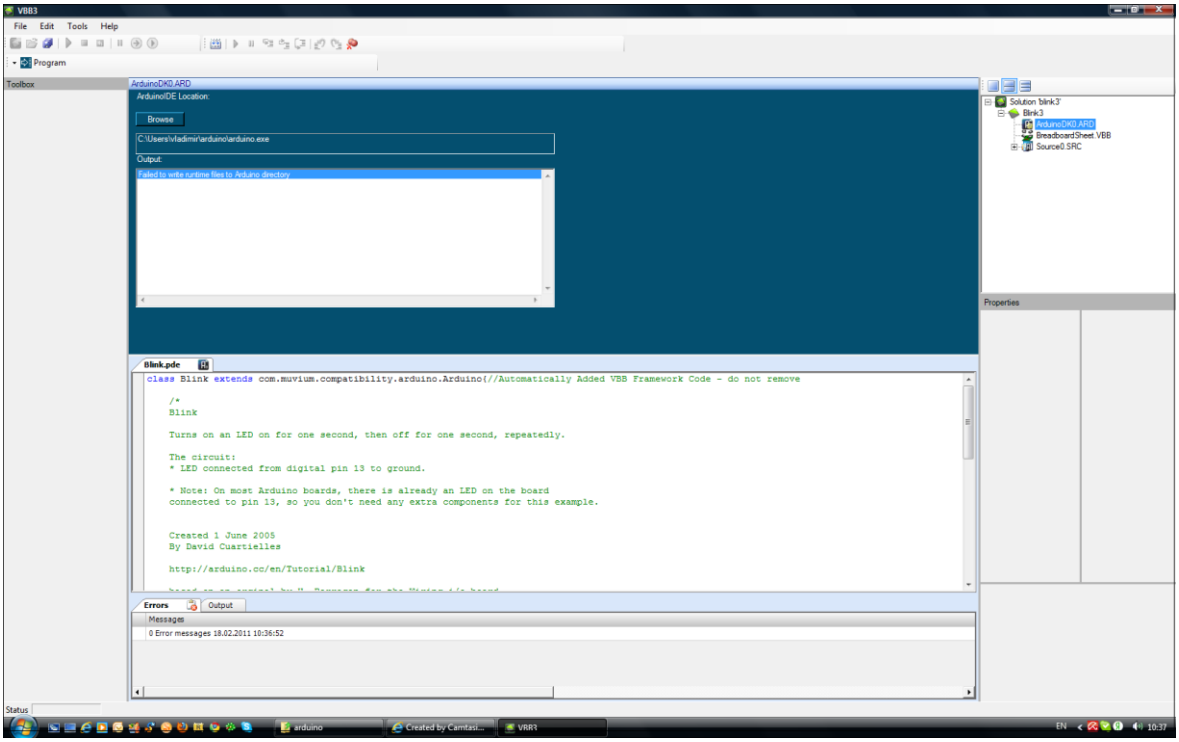


Рис. 7.6. Сообщение о проблемах в работе программы

Программа сообщает, что не может записать нужные ей файлы в директорию Arduino. Не могу сказать, столкнётесь ли вы с этой проблемой в Windows XP, некоторые считают, что можете столкнуться, но в Windows Vista я столкнулся (видимо, и в Windows 7 это будет иметь место). Операционная система защищена от несанкционированного изменения файлов. В данном случае работает эта защита. Её можно отключить, я пробовал, но потом долго и не без труда включал эту защиту – весь процесс получился столь долгим, что у меня, просто, не хватало терпения дождаться, когда оживёт операционная система, которая на мою просьбу перейти в диалог включения защиты задумалась…

Есть ещё одна причина, по которой я не советую снимать защиту – ослаблять защиту Windows, это, знаете, себе дороже. Я сейчас использую бесплатную версию антивирусной программы Avira. В последнее время её усовершенствовали, она проверяет компьютер до подключения его к сети, доступ к Интернету закрыт, и длится это долго. Я, конечно, злюсь, но вспоминаю, как много лет назад, переустанавливая Windows, я не поставил антивирусную программу. Решив, что вначале я поменяю разбивку жесткого диска, я запустил программу… В то время был такой вирус, который самопроизвольно перезагружал компьютер. В результате жёсткий диск оказался испорчен, и я с большим трудом восстановил только некоторые, самые необходимые для работы, файлы. Злись, не злись, но лучше не снимать защиту с компьютера. Может быть, хотя и не факт, по этой причине я в основном работаю в Linux, обращаясь к Windows в случаях, подобных сегодняшней необходимости описать работу с VirtualBreadboard.

Но, что же делать? Перед запуском (или получив такое сообщение, когда вы нажали клавишу «Program» в верней части окна) следует проделать несложную операцию: в проводнике перейти к папке, которую вы создали и назвали arduino; щёлкнуть по ней правой клавишей мышки…

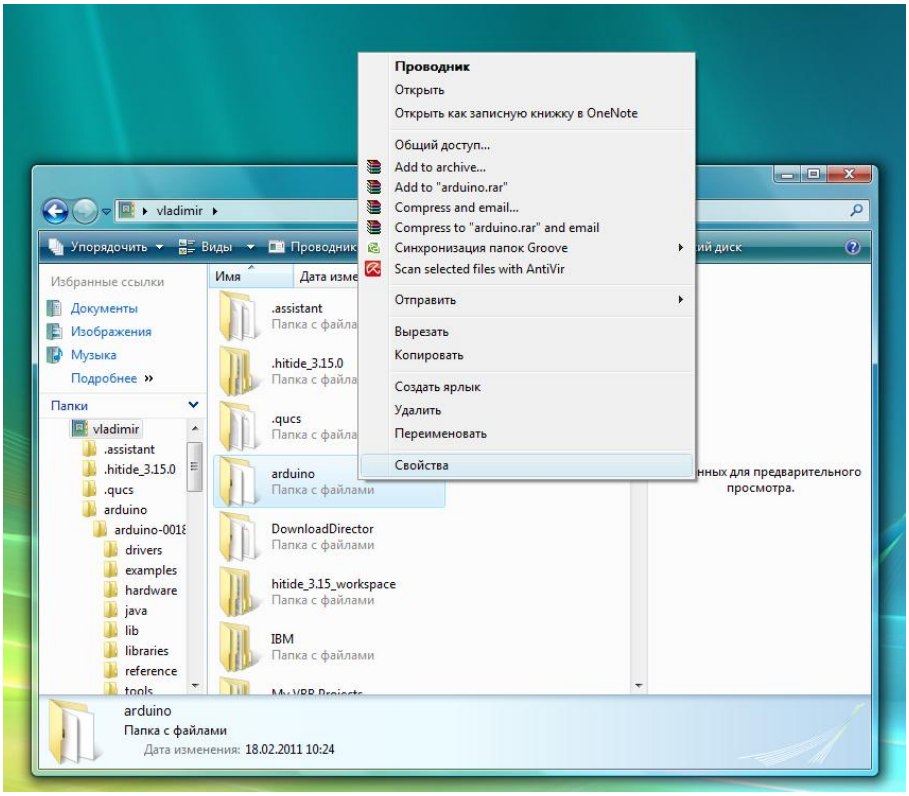


Рис. 7.7. Доступ к свойствам папки

В открывающемся меню выберите раздел «Свойства». Открывается диалоговое окно свойств папки.

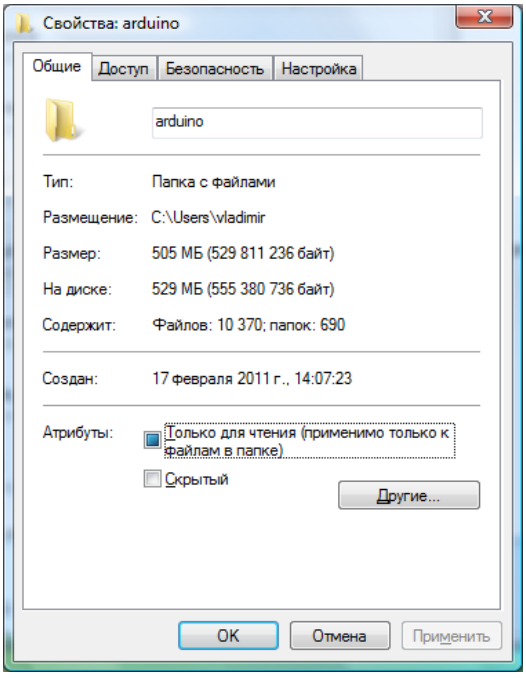


Рис. 7.8. Изменение свойств папки

Как вы видите, папка «Только для чтения». Такое свойство помогает защитить программы и файлы от несанкционированного изменения. Но нам нужно снять эту опцию – щёлкнуть левой клавишей мышки по синему квадратику рядом с надписью. Откроется следующее окно диалога.

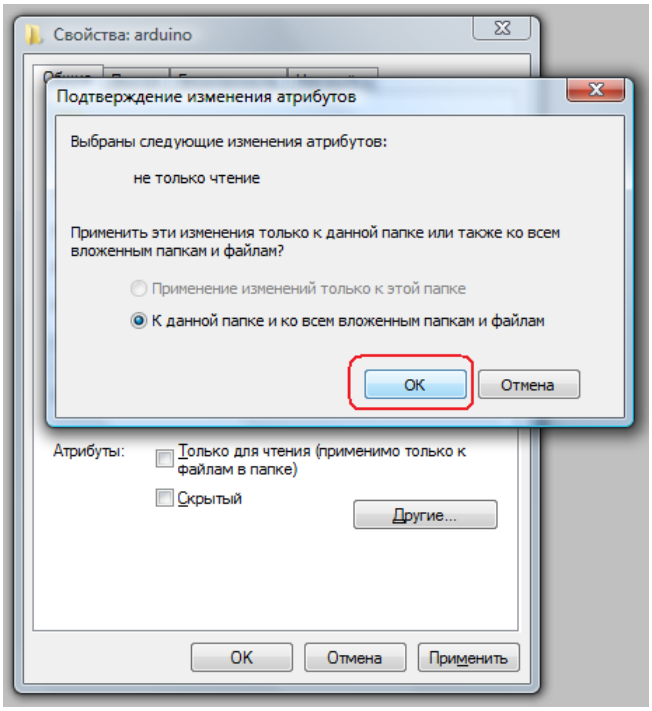


Рис. 7.9. Подтверждение изменения свойств

Достаточно нажать выделенную кнопку «OK», что закрывает диалог, нажать кнопку «Применить» в следующем окне, дождаться пока завершится процесс разблокирования файлов, и нажать кнопку «OK», когда кнопка «Применить» перестанет быть активна.

Теперь, в программе VirtualBreadboard клавиша «Program» запускает загрузку модуля, о чём можно судить и по активному миганию светодиодов, работающих с COM-портом (виртуальным) компьютера.

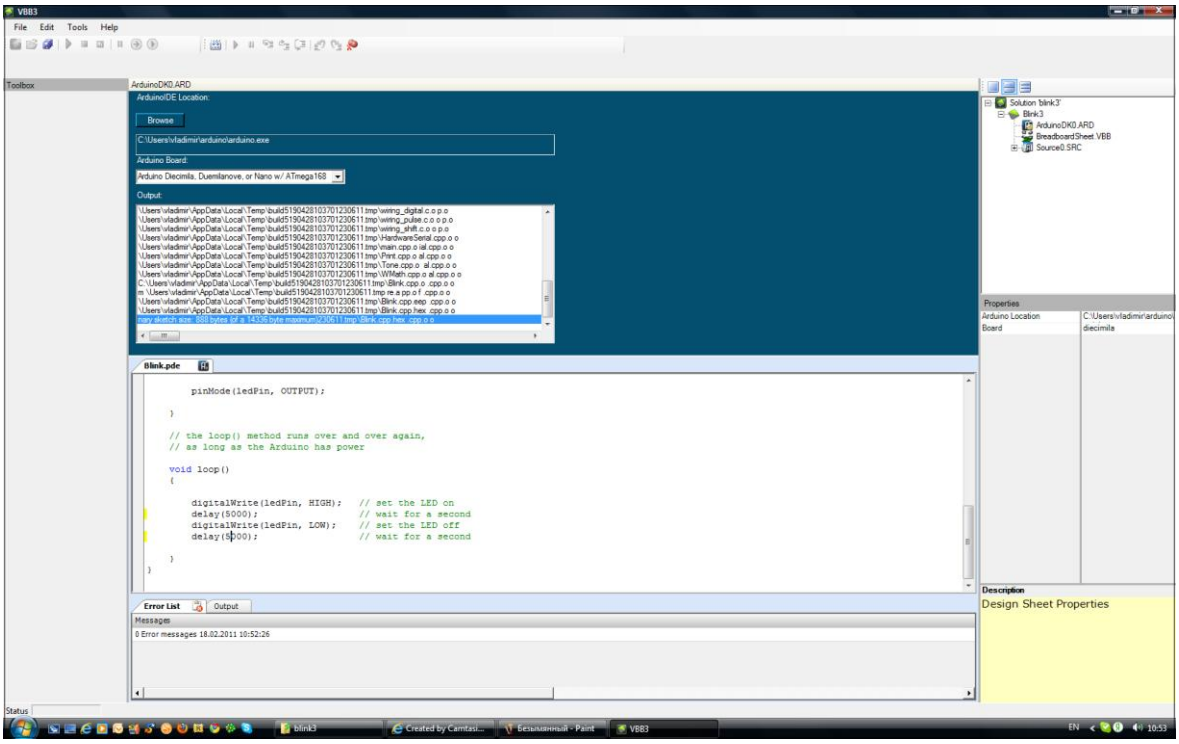


Рис. 7.10. Правильная работа программы

Если вы последовали моему совету, то, как проверить, что всё у вас получилось?

Я не буду ничего придумывать, я перескажу только то, что увидел на видеоролике, размещённом на сайте проекта.

Как советует автор, запускаем программу и выбираем нужный пример.

Рис. 7.11. Выбор одного из примеров в начальном диалоге

Открываем его, нажав на кнопку «Open». Мы можем проверить работу программы, запустив моделирование кнопкой на инструментальной панели (отмечена на рисунке ниже).

Рис. 7.12. Основное меню и инструментальные панели программы

Убедившись, что светодиод мигает с частотой раз в секунду, мы должны сохранить проект. Для этого используем в разделе основного меню «File» пункт «Save as…, сохранить как».

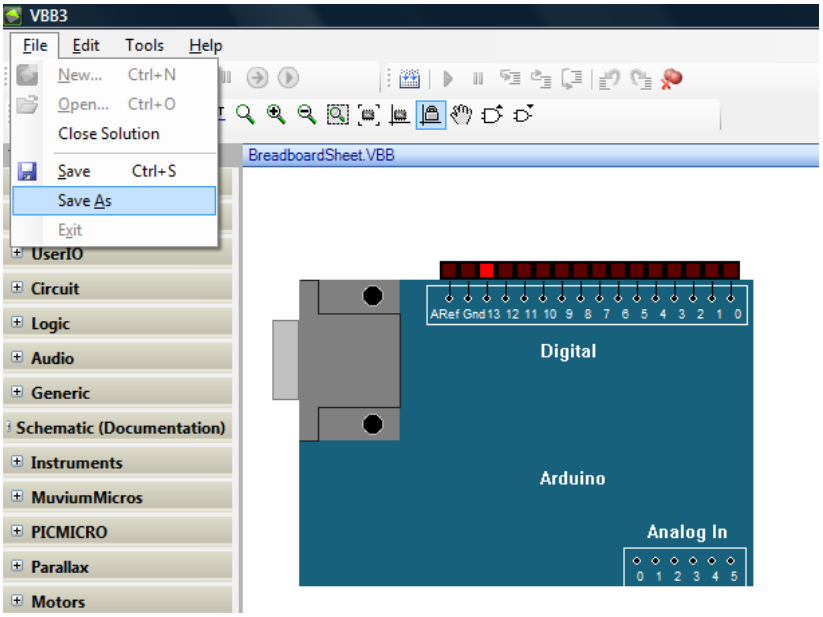


Рис. 7.13. Содержание раздела File основного меню

При этом открывается диалоговое окно, в котором обратите внимание на выделенный мною фрагмент.

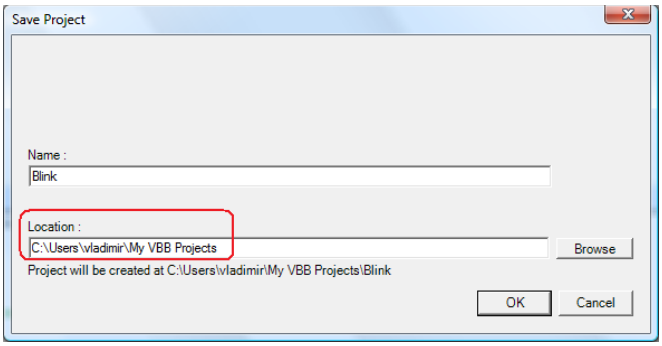


Рис. 7.14. Создание папки проектов при первом сохранении

Этой папки пока у вас нет. Она будет создана, когда вы нажмёте кнопку «ОК».

Теперь обратимся к правой части рабочего окна программы. Туда, где отображаются все компоненты проекта.

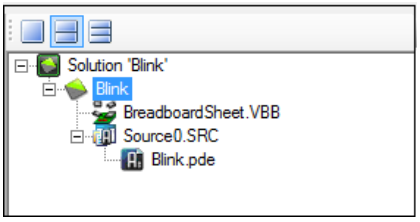


Рис. 7.15. Окно менеджера проекта

Щёлкнув по выделенной папке проекта правой клавишей мышки, выбираем из выпадающего меню вначале пункт «Code Generators», что открывает подменю, затем в нём пункт «Add New Arduino Code Generator, добавить новый Arduino генератор кода».

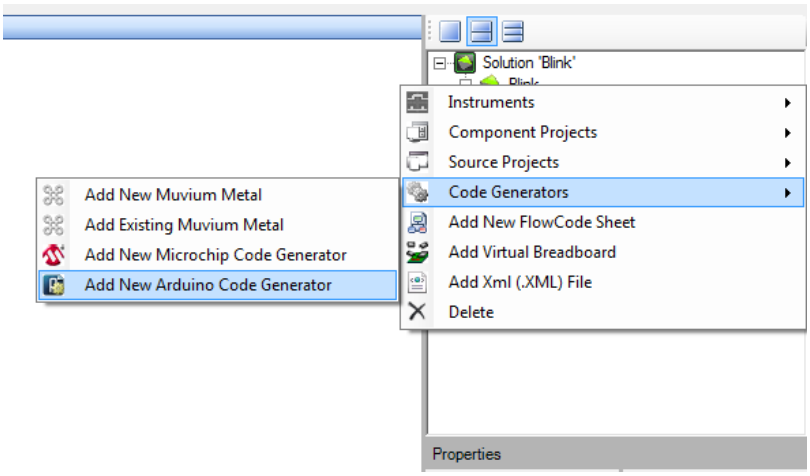


Рис. 7.16. Выбор генератора кода

После этого «дерево» проекта несколько изменится.

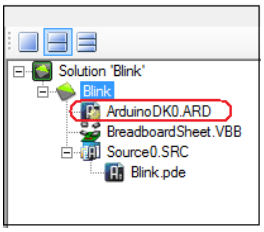


Рис. 7.17. Изменение вида дерева проекта

Подцепите этот появившийся элемент мышкой и перенесите его к верхней кромке окна с рисунком модуля. Там его и «сбросьте».

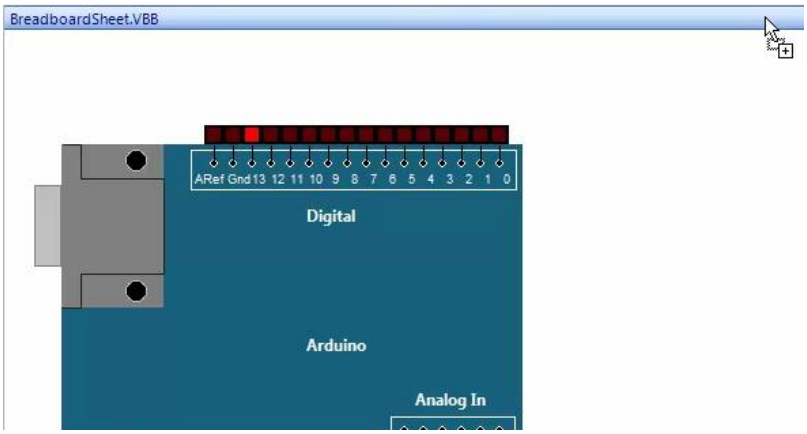


Рис. 7.18. Перенос «генератора кода» в рабочее окно

Вид рабочего окна изменится. А, если щёлкнуть по этому элементу в дереве проекта дважды левой клавишей мышки, то ниже появится окно свойств. Следующие шаги, которые нам предстоит выполнить, это указать место расположения программы Arduino и выбрать свою модель модуля Arduino. Для указания места расположения служит кнопка «Browse», а модуль выбирается из выпадающего списка, когда вы нажмёте кнопку со стрелкой рядом с окошком модели, озаглавленным «Arduino Board:». Как видно из списка, программа работает со многими модулями.

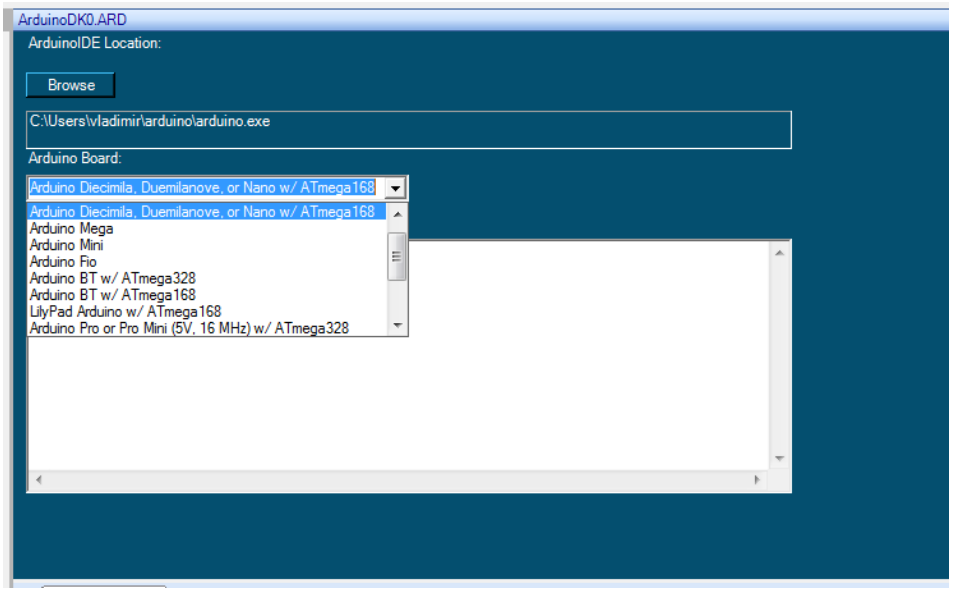


Рис. 7.19. Выбор модели модуля Arduino

Далее двойным щелчком в дереве проекта по компоненту Source0.SRC мы откроем его свойства (в окошке ниже), где должны выбрать генератор кода (отмечены на рисунке).

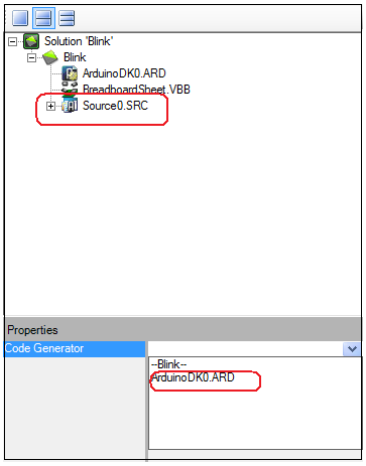


Рис. 7.20. Настройка исходного кода

После выбора в окне свойств рядом с «Code Generator» появится нужный нам вариант.

Следующая процедура – трансляция исходного кода. Проще всего это сделать с помощью отмеченной на рисунке кнопки на инструментальной панели.

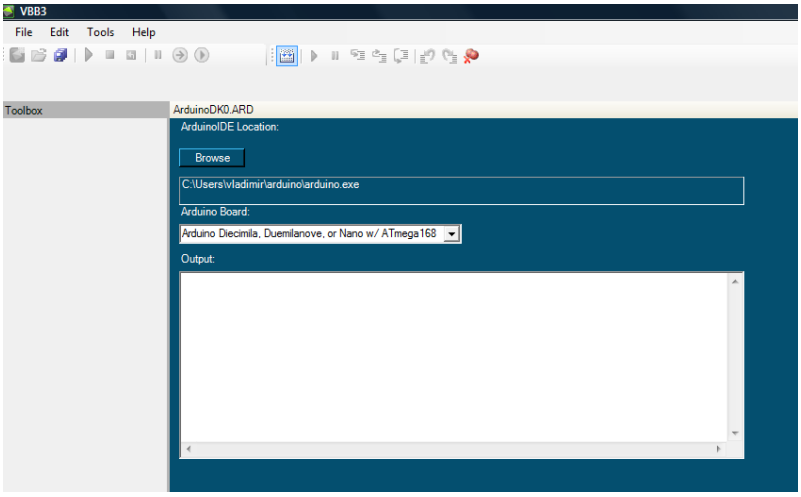


Рис. 7.21. Окончание настроек генератора кода

Остаётся указать порт для связи с модулем…

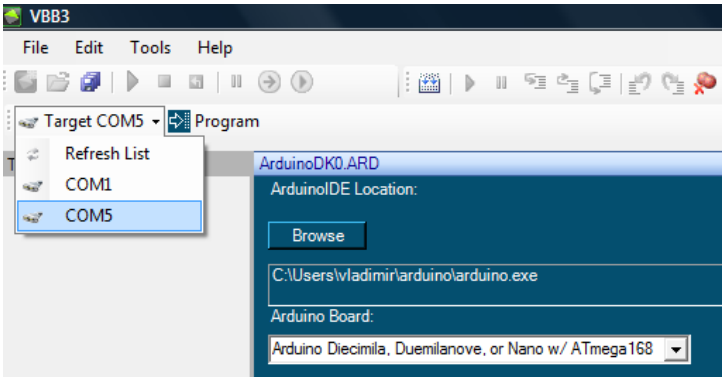


Рис. 7.22. Порт связи компьютера с модулем Arduino

…и можно нажать, наконец, кнопку «Program».

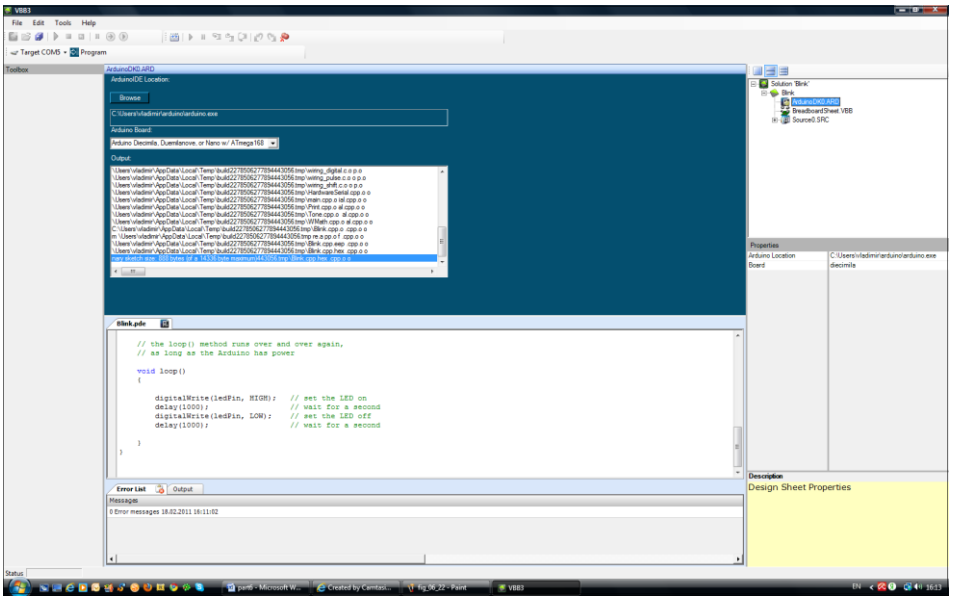


Рис. 7.23. Загрузка программы в модуль из VBB

Мой модуль Arduino, надеюсь, что и ваш, начинает весело мигать всеми своими светодиодами.

Глава 8. Немного больше о программе VirtualBreadboard

Хорошо, мы установили эту программу. Мы отладили «механизм» работы с модулем. И посмотрели несколько примеров. Но хотелось бы и самим создавать что-то полезное. Попробуем.

Сначала закроем предыдущий проект. Для этого в разделе «File» основного меню выберем пункт «Close Solution». Затем, например, на инструментальной панели выберем кнопку создания нового проекта.

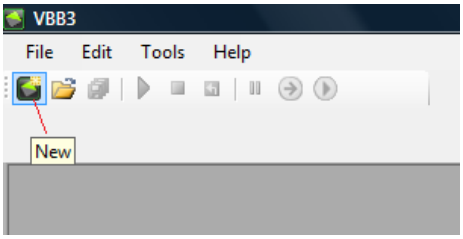


Рис. 8.1. Создание нового файла кнопкой инструментальной панели

Можно использовать и основное меню, где в разделе «File» есть пункт «New», а можно использовать «горячие» клавиши клавиатуры «Ctrl+N». Кому, как нравится. В появившемся диалоговом окне выбираем «New Project».

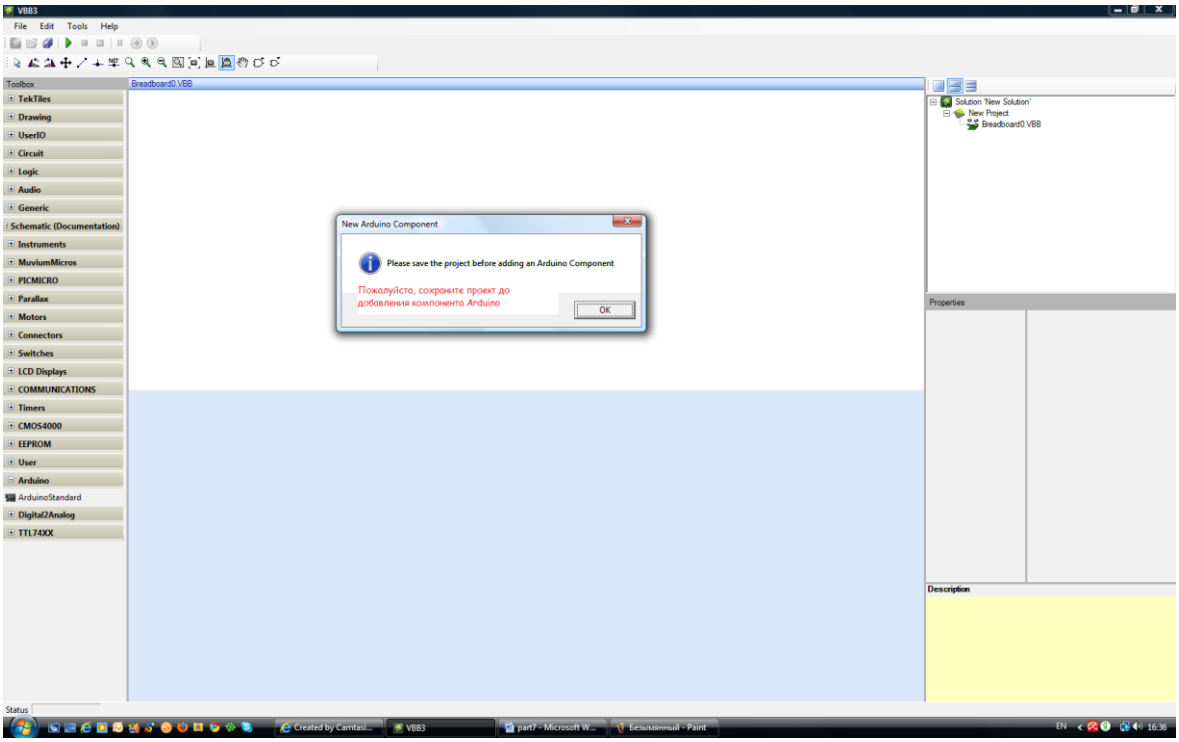


Рис. 8.2. Предупреждение о необходимости сохранить проект

Сообщение, которое вы видите на рисунке, получено при попытке добавить модуль Arduino в проект – необходимо перед этим сохранить проект, дав ему имя (или оставив то, что задано по умолчанию). Команды сохранения проекта, конечно, отыщутся в разделе «File» основного меню. А в окне открывающегося диалога можно изменить при желании имя проекта.

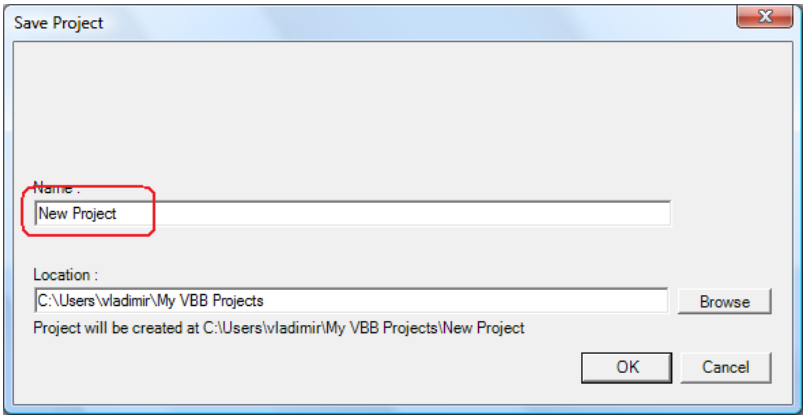


Рис. 8.3. Окно задания имени проекта

После сохранения проекта выберем на левой панели компонентов тот, что назван Arduino, щелчком левой клавиши по значку «+» раскроем меню и выберем (хотя он там и единственный) «ArduinoStandart». Чтобы его перенести в рабочее поле, достаточно щёлкнуть по нему левой клавишей мышки и курсор мышки переместить в нужное место, где щёлкнуть мышкой ещё раз.

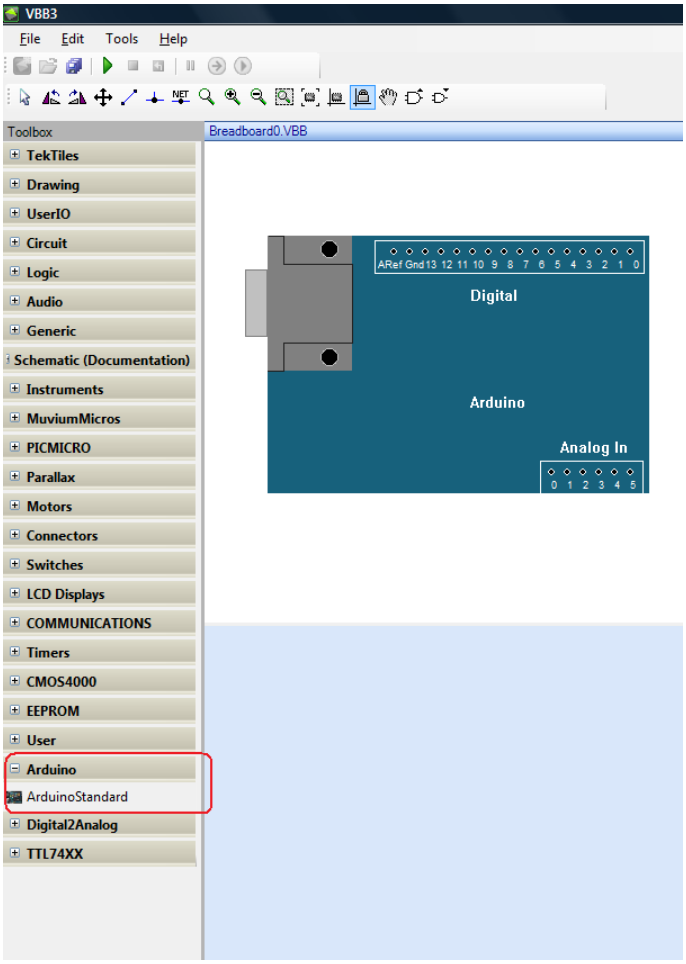
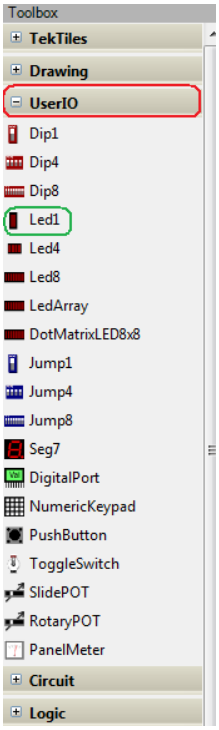


Рис. 8.4. Выбор модуля Arduino в разделе компонентов программы

Чтобы повторить предыдущую программу, а повторение, как известно, мать учения, я хочу добавить светодиод. Его можно найти среди компонентов программы под заголовком «UserIO”.



Щёлкнув по отмеченному «Led1» левой клавишей мышки, переносим компонент к модулю так, чтобы его нижний вывод попал в гнездо под номером 13. Следующий щелчок мышки оставляет светодиод на месте.

Рис. 8.5. Место расположения других компонентов, подключаемых к модулю

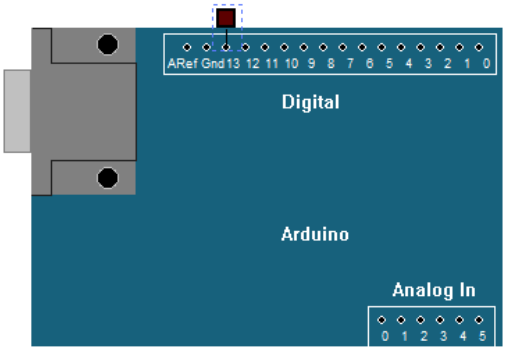


Рис. 8.6. Подключение светодиода к модулю

Чтобы что-то заработало, полагаю, нам нужно добавить исходный код программы. С этой целью обратимся к дереву проекта, где щелчком по «New Project» правой клавишей мышки вызываем меню.

Напомню, в разделе «Source Projects» из подменю выбираем исходный код для проекта Arduino.

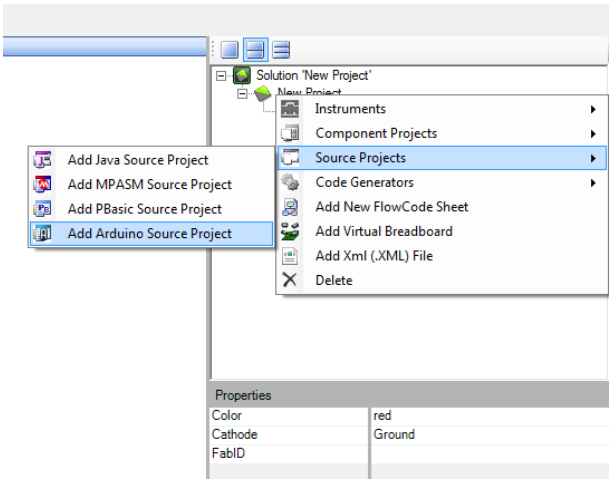


Рис. 8.7. Выбор типа исходного кода программы

В окне диалога можно изменить имя заголовка кода, но можно оставить заданное по умолчанию.

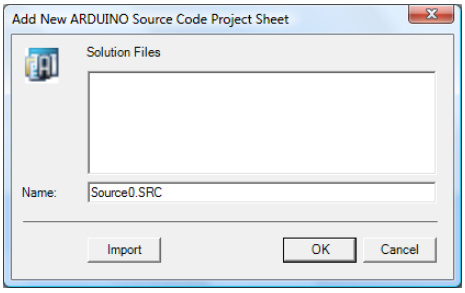


Рис. 8.8. Окно ввода имени для модуля кода

Ещё раз обратимся к дереву проекта, где появившийся заголовок, если по нему щёлкнуть правой клавишей мышки, предоставит возможность создать файл программы.

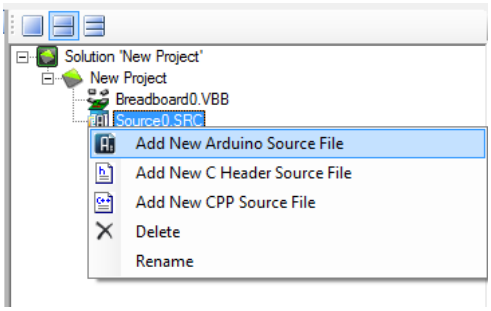


Рис. 8.9. Добавление нового файла программы

Аналогичное диалоговое окно позволяет дать файлу своё имя (или оставить то, что есть). Сохраним все, используя кнопку сохранения на инструментальной панели. Она находится правее кнопки открывания файла. После сохранения закроем проект. Вы помните – это «File-Close Solution». Повторно мы его открываем используя кнопку «Открыть» инструментальной панели или «File-Open». Теперь нас в диалоговом окне интересует закладка «Existing, существующие». Проект открывается в том же виде, что мы оставили, когда выходили из проекта. За одним исключением.

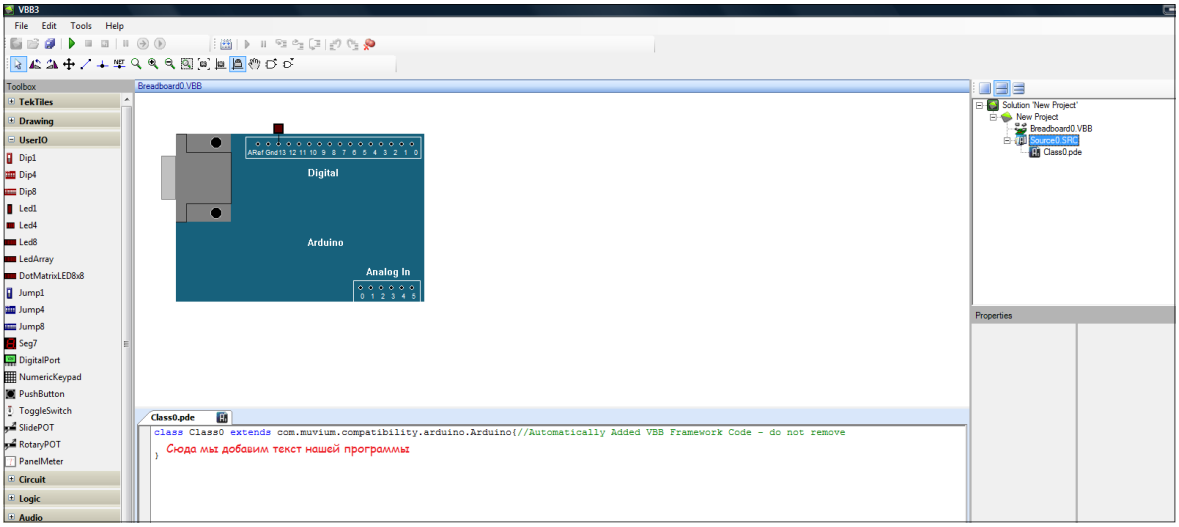
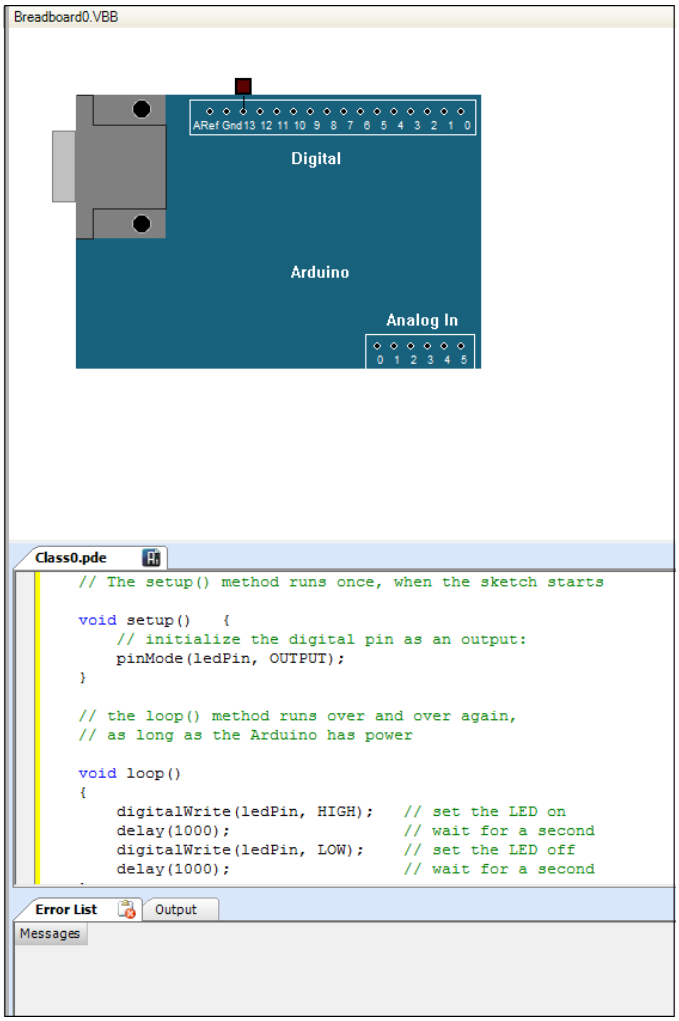


Рис. 8.10. Программа с окном редактора текста программы



Текст программы, согласен, можно ввести вручную, но, и этим я воспользуюсь, можно просто скопировать и вставить из файла примера Arduino или файла этой программы.

Рис. 8.11. Добавление текста в шаблон программы

Выделим плату модуля щелчком левой клавиши мышки. И, пока не забыл, программа имеет разные режимы работы. Для выбора режима выделения используйте кнопку инструментальной панели.

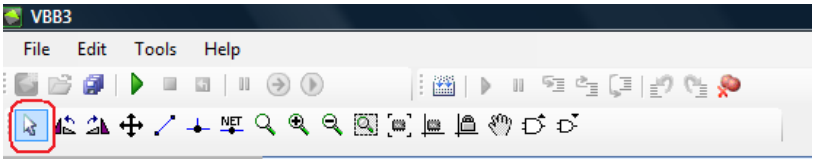


Рис. 8.12. Кнопка выделения на инструментальной панели

Выделив плату, обратим внимание на окно свойств. Откроем свойство модуля (компонента), названное «Application», щелчком в окне рядом и выберем…

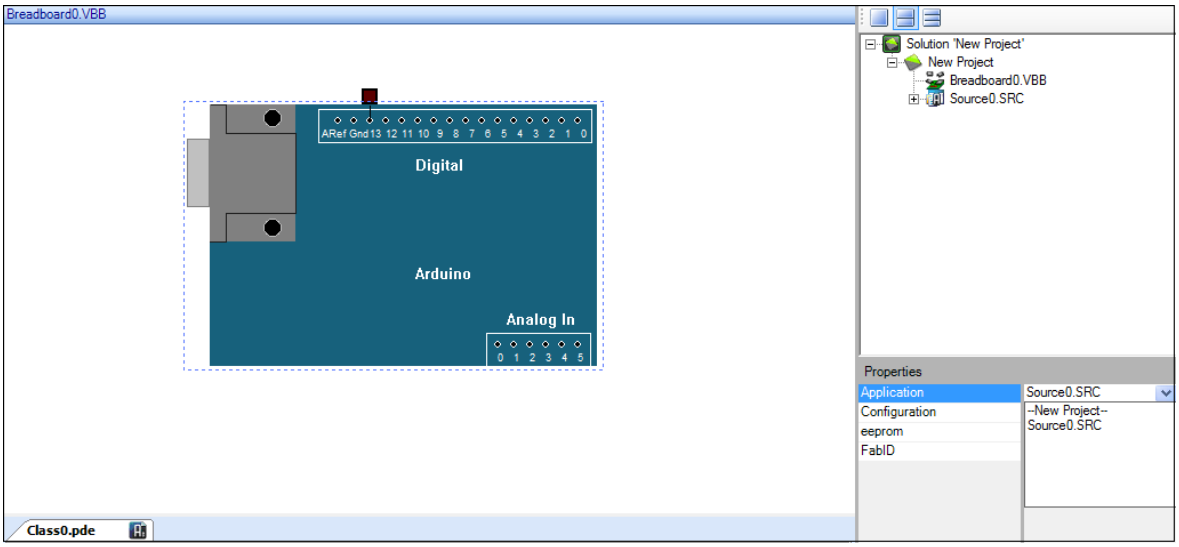


Рис. 8.13. Окно свойств программного модуля Arduino

Иначе при попытке запустить моделирование вы получите сообщение.

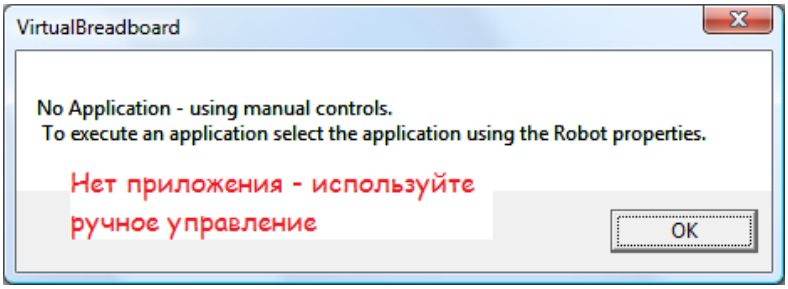


Рис. 8.14. Сообщение, что не выбран автоматический режим работы с модулем

А выполнив изменение свойств приложения, мы можем видеть работу программы.

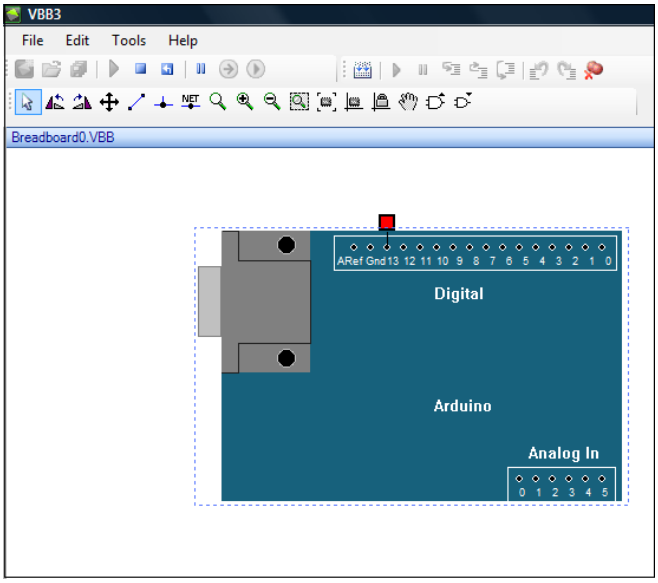


Рис. 8.15. Запуск моделирования программы

А если мы продолжим выполнение операций по подготовке к загрузке, как это было описано ранее, и согласимся (после нажатия на клавишу «Program») создать папку для кода программы, то сможем загрузить программу в модуль Arduino. Разумно при этом было бы изменить время в строке с 1000 на:

```
delay(5000);
```

Это сделает более заметной разницу. И не забудьте транслировать код программы перед загрузкой.

Мы не использовали готовую программу. Мы создали всё сами. Теперь мы можем создавать другие программы. Но для этого следует лучше узнать элементы программы. Начнем с основного меню. Первый раздел «File», конечно предназначен для работы с файлами.

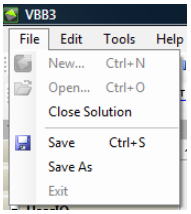


Рис. 8.16. Содержание раздела File основного меню

Практически обо всех пунктах этого раздела мы говорили: New… – новый проект; Open… – открыть проект; Close Solution – закрыть проект; Save – сохранить проект; Save as – сохранить как; Exit – выйти.

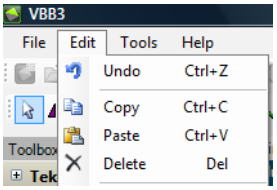


Рис. 8.17. Содержание раздела Edit основного меню

Этот набор команд относится к редактированию. Здесь тоже все команды привычны: Undo – отмена последних изменений; Copy – копировать выделение; Paste – вставить скопированное;

Delete – удалить выделенное. Заметьте, что этот набор команд применим и к редактированию текста программы в окне текстового редактора, и к редактированию схемы. Если вы выделили компонент схемы, а затем пытаетесь перейти в окно редактора текста и редактировать текст, то могут возникнуть осложнения.

Остальные два пункта меню относятся к терминалу (Tools) и подсказке по версии программы.

При редактировании текста вы можете пользоваться выпадающим меню, которое появляется при щелчке правой клавиши мышки.

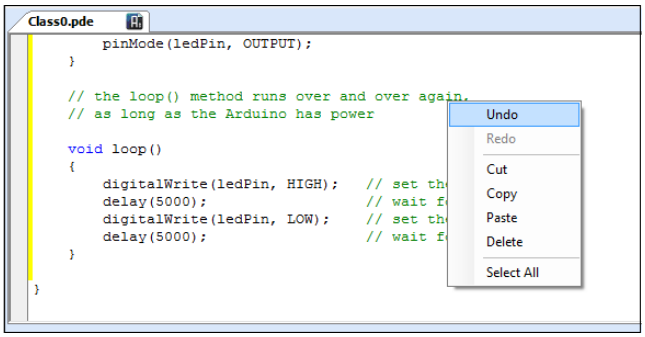


Рис. 8.18. Выпадающее меню редактора текста программы

Ниже основного меню инструментальные панели.

Рис. 8.19. Инструментальные панели программы

Разобьём их на две группы. Не забывайте, что меню контекстно-чувствительно – если вы не выполняете какие-либо операции, то часть, относящаяся к этим операциям, не будет активна. К первой группе отнесём следующую инструментальную панель.

Рис. 8.20. Инструментальная панель управления проектом

Команды (слева-направо): новый проект; открыть проект; сохранить всё. Далее: запустить моделирование; остановить моделирование; рестарт моделирования; пауза симуляции; шаг симуляции; возобновить симуляцию. Следующий набор команд относится к компиляции кода, затем к отладочным операциям: включить отладку; сделать паузу; сделать шаг; шаг вне; шаг через; отменить; вернуть; снять точку останова.

Напомню, что программа может работать, например, с PIC-контроллерами. Обратите внимание на эти кнопки инструментальной панели на рисунке ниже.

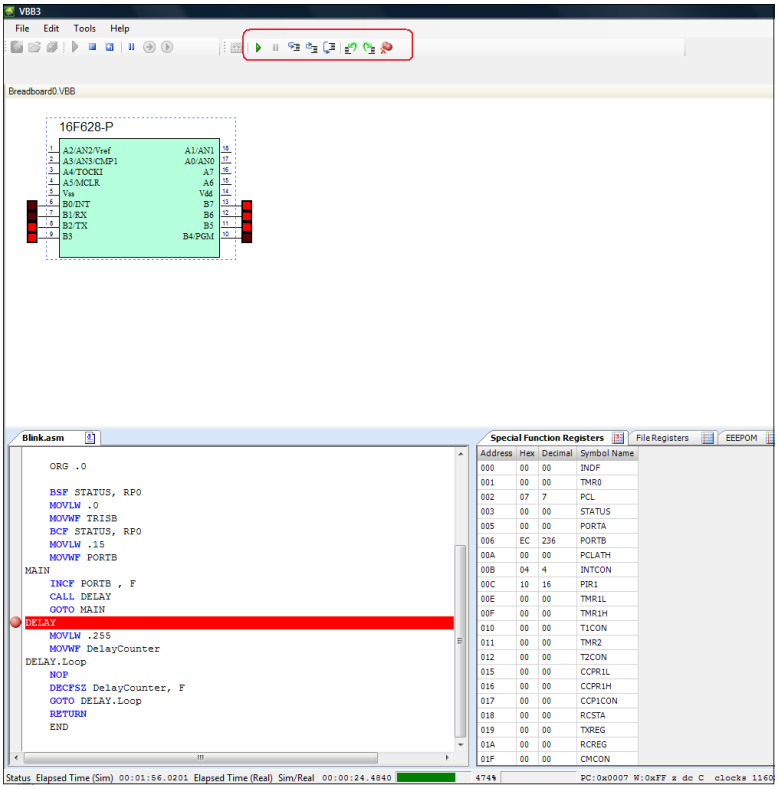


Рис. 8.21. Задание точки останова при отладке PIC-контроллера

Но вернёмся ко второй группе команд инструментальной панели.

Рис. 8.22. Инструментальная панель работы с компонентами программы

Они окажутся полезны в первую очередь при построении собственных схем. Тоже слева-направо: кнопка выделения; вращение против часовой стрелки; вращение по часовой стрелке; перемещение; добавление связи; соединение с проводом; добавление метки провода; масштабирование; увеличение; уменьшение; область масштабирования; масштабирование пространства; восстановление оригинала; закрепление оригинала; панорамирование; увеличение компонента; уменьшение компонента (элемента объекта).

Вот пример соединения модуля с дополнительным элементом.

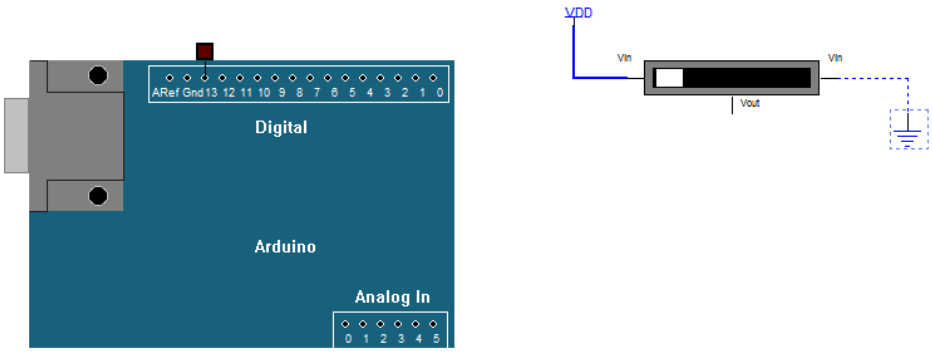


Рис. 8.23. Пример соединения компонентов программы

Добавив компоненты, нажимаем кнопку добавления связи – курсор мышки меняет вид. Щёлкаем по нужному концу элемента, ведём линию, щёлкаем по другому нужному концу и щёлкаем правой клавиши мышки, чтобы закончить соединение.

Самый насыщенный раздел, пожалуй, это область выбора компонентов программы (Toolbox).

Каждый раздел этого меню открывается, если нажать на «плюсик» рядом с его названием. Откроем, например, раздел элементов электрической цепи.

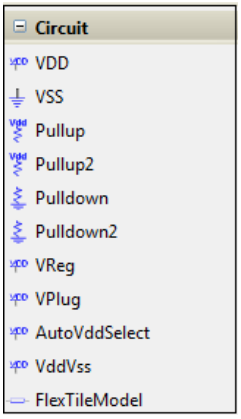


Рис. 8.24. Окно компонентов программы

Рис. 8.25. Содержание раздела схем

Множество элементов этого раздела помогают правильно организовать работу с микроконтроллерами. Практически все элементы имеют вполне привычное обозначение.

О любой программе, которая упоминалась выше, я думаю можно написать книгу. Поэтому в беглом обзоре трудно рассказать всё. Но, прежде чем перейти к продолжению рассказа, я хочу остановиться на ещё одной возможности этой программы.

Как вы помните, мы говорили о графическом языке программирования. Программа VirtualBreadboard тоже поддерживает такой язык для PIC-контроллеров. Рассмотрим, как использовать это, используя пример из набора программы.

Открываем нужный проект.

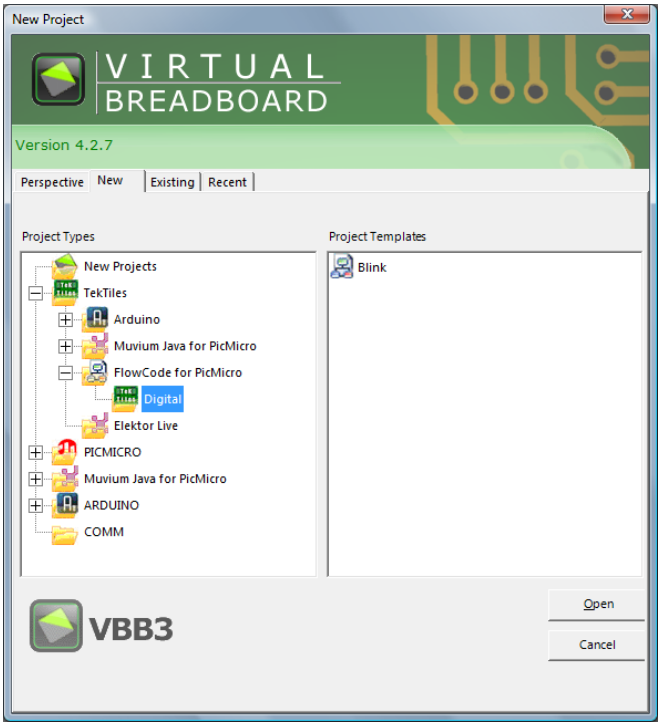


Рис. 8.26. Выбор режима графического программирования

Выбрав проект Blink в правом окне, нажимаем кнопку «Open», и получаем проект. Как можно заметить сразу, программа создана с помощью графического языка.

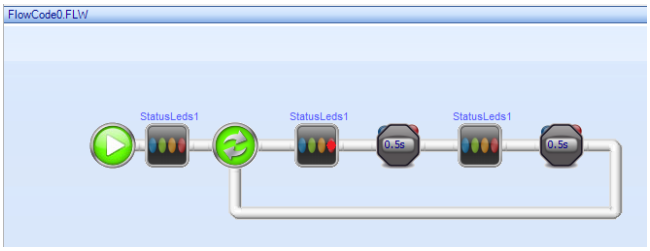


Рис. 8.27. Программа на языке графического программирования

Запустив её обычным образом, можно наблюдать, как мигает светодиод.

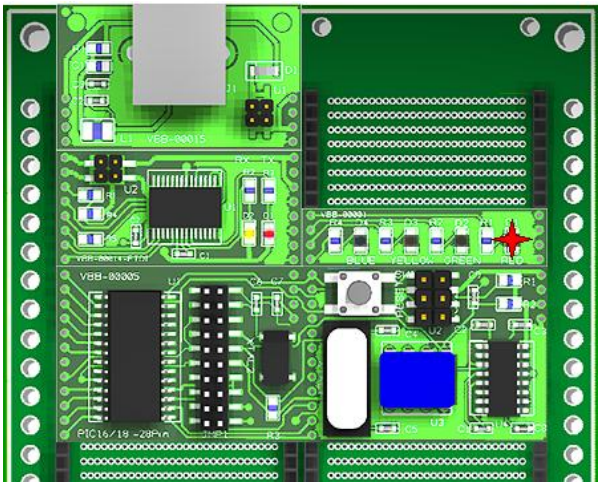


Рис. 8.28. Работа программы FlowCode

Я не уверен, но эта часть проекта, похоже, ещё в разработке.

Для PIC-контроллеров есть возможность, лучше всего посмотреть, как это делается, на сайте автора, соединить работу программы VirtualBreadboard с программой MPLAB. И, кроме того, есть возможность использовать готовый hex-файл. Вот как об этой возможности пишет сам автор проекта (есть и видеоролик!).

Скажем, вы уже создали HEX файл, используя ваш компилятор, или как-то ещё, а теперь вы хотите симулировать его. В данном руководстве *показаны ваши основные шаги, чтобы:*

1. *Запустить VBB и создать новый проект.*
2. *Перетащить PIC-контроллер (автор использует PIC18F8722).*
3. *Присоединить некоторые виртуальные компоненты, LCD и монитор COM-порта.*
4. *Сохранить проект.*
5. *Указать свойства PIC и соединить свойства приложения и HEX файла для запуска.*
6. *Запустить виртуальное приложение.*
7. *Отредактировать свойства для улучшения приложения.*

Я, пожалуй, тоже заготовлю hex-файл и постараюсь повторить урок. Создаю новый проект. Отыскиваю в разделе компонентов нужный мне контроллер.

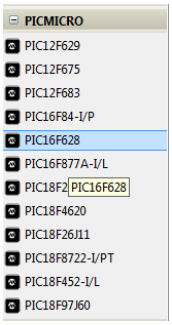


Рис. 8.29. Набор моделей PIC-контроллеров

Перетаскиваю его в рабочее поле. Я предполагаю использовать вывод 0 порта B, чтобы повторить уже знакомый (и очень простой) проект Blink. Автор VirtualBreadboard предлагает более интересный пример, но его вы можете увидеть сами. Итак. Добавим из раздела UserIO светодиод. И соединим его с выводом B0. О том, как сделать это, мы говорили выше.

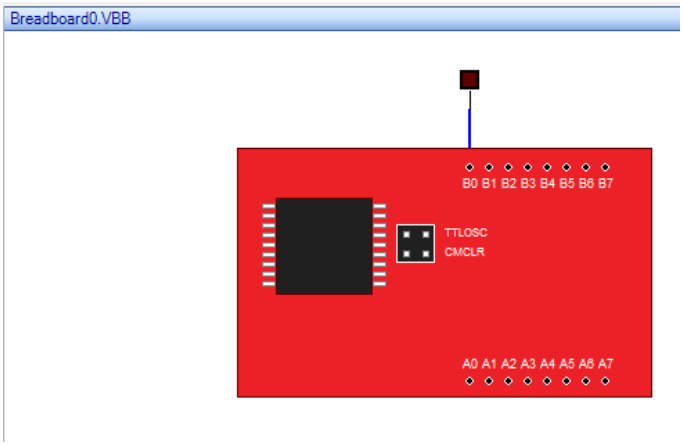


Рис. 8.30. Подготовка схемы программы

Сохраняем проект, следуя инструкции. Чтобы изменить свойства контроллера и указать место

расположения hex-файла, выделяем PIC, щелкнув по контроллеру левой клавишей мышки, и переходим в окно свойств.

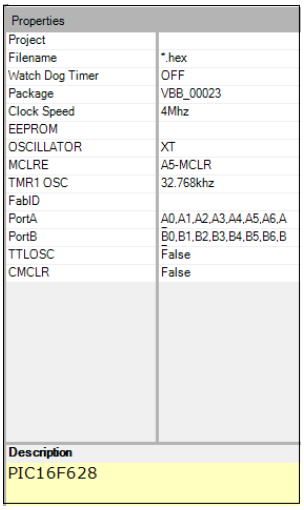


Рис. 8.31. Окно свойств микроконтроллера

Раздел свойств Project пока пуст. А имя файла – шаблон для всех файлов с расширением hex. Кроме того, я использовал внутренний тактовый генератор, работающий на частоте 4 МГц. Щелчком левой клавиши мышки по имени свойства «Filename» мы «оживляем» окно, где появляется кнопка выбора пути к hex-файлу. Программа использует проводник Windows, в котором вы перемещаетесь обычным образом, чтобы указать путь к файлу. Всё готово к пуску. Пуск:

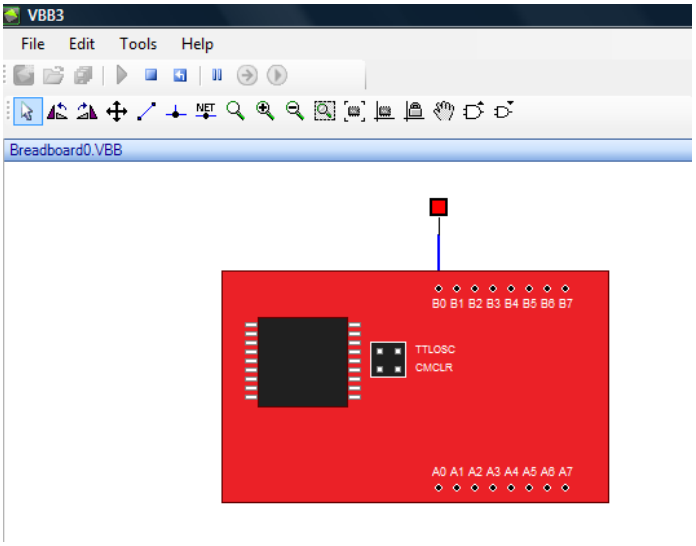


Рис. 8. 32. Проверка работы программы

Светодиод мигает с заданной мной частотой раз в 5 секунд.

Порой важно посмотреть на экране, что происходит на выходе микроконтроллера. Программа предлагает для этой цели такой хороший прибор, как логический анализатор. Чтобы выполнить проверку предыдущей программы, у которой я изменил время «мигания» с 5 секунд на 5 миллисекунд, добавим логический анализатор.

Откроем предыдущую программу, сохраним её под другим именем. Заменим исходный hex-файл новым. Запустим программу – нужно же проверить, работает ли она.

Добавим логический анализатор.

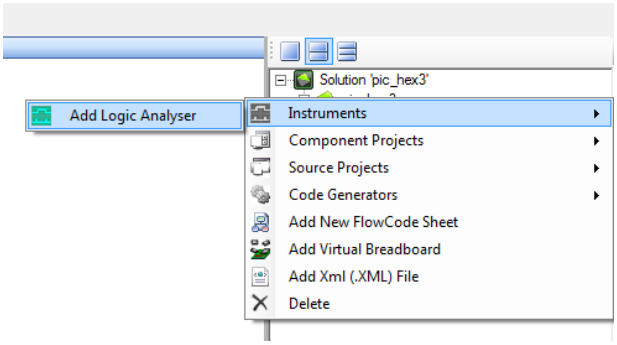


Рис. 8.33. Добавление логического анализатора

Переключим работу с двух окон к трём.

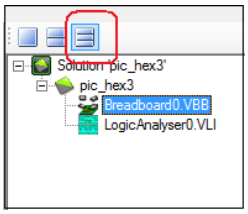


Рис. 8.34. Выбор количества окон программы

Подцепим логический анализатор мышкой и перенесём его в нижнее окно.

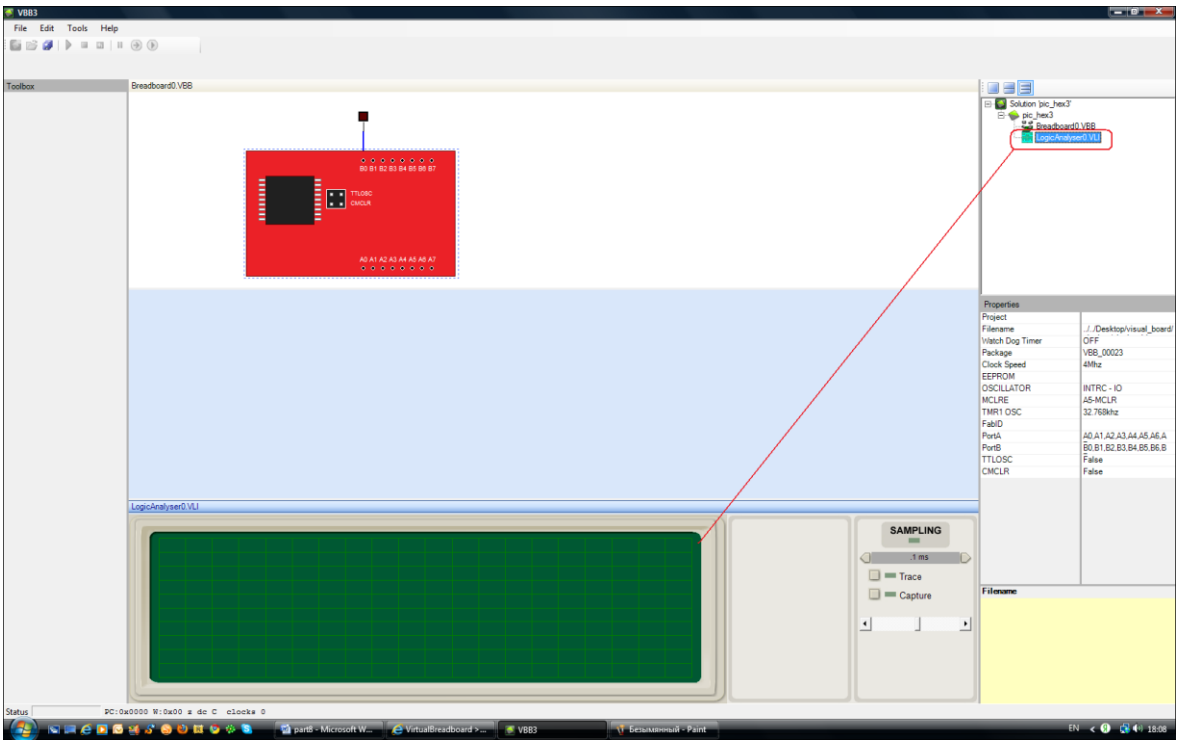


Рис. 8.35. Перенос логического анализатора в рабочее поле

Вернёмся к плате, щёлкнув по BreadBoard0.VBB. Выберем на панели компонентов раздел Istruments, где выбираем пробник логического анализатора.

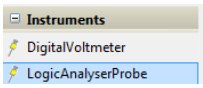


Рис. 8.36. Местонахождение логического пробника

Переносим его к макетной плате, где соединяем со светодиодом.

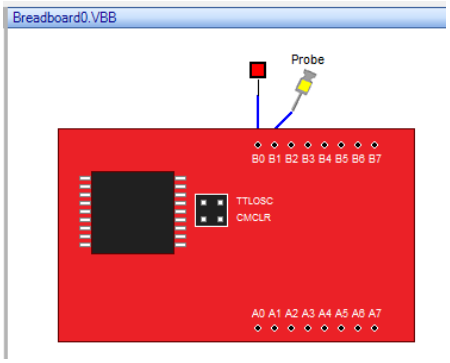


Рис. 8.37. Добавление к схеме логического пробника

В свойствах, выделив логический анализатор, выбираем Instrument щелчком левой клавиши мышки. И выбираем LogicAnalyser0.VLI (после щелчка мышки появляется кнопка выбора из выпадающего списка).

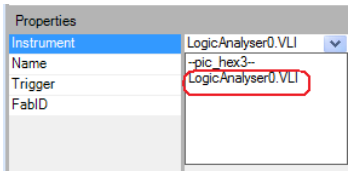


Рис. 8.38. Изменение свойств логического анализатора

Включаем кнопку логического анализатора «Trace» и запускаем моделирование. После остановки можно увидеть, что на экране анализатора что-то появилось. Изменим время отображения, сделав его удобным для наблюдения. Можно даже убедиться, что время включения (и выключения) равно 5и миллисекундам.

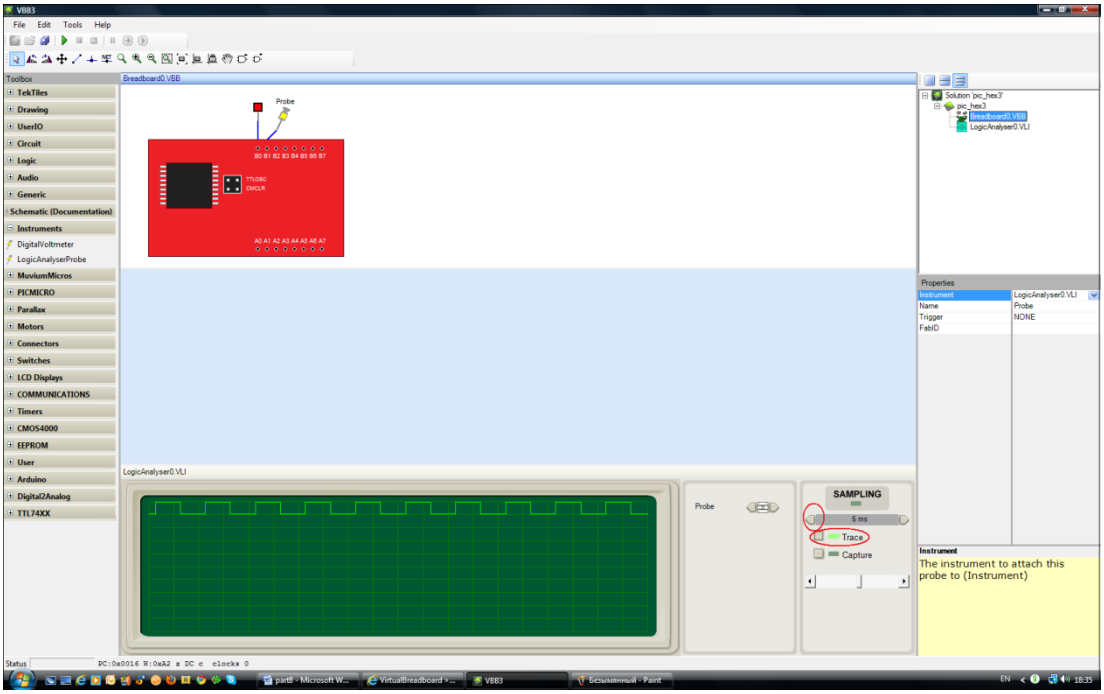


Рис. 8.39. Работа логического анализатора

Я уже говорил, что хорошее описание программы потребует отдельной книги. Возможно, я соберусь написать её, а пока… стоп, я забыл, пока ещё не покинул Windows, я хотел рассказать еще об одной программе. Называется она Fritzing.

Найти её можно на сайте: http://fritzing.org/.

Что же это за программа? Как и другие, она имеет примеры, которые можно посмотреть. Запустим программу, дальше «Файл-Открыть пример» в основном меню.

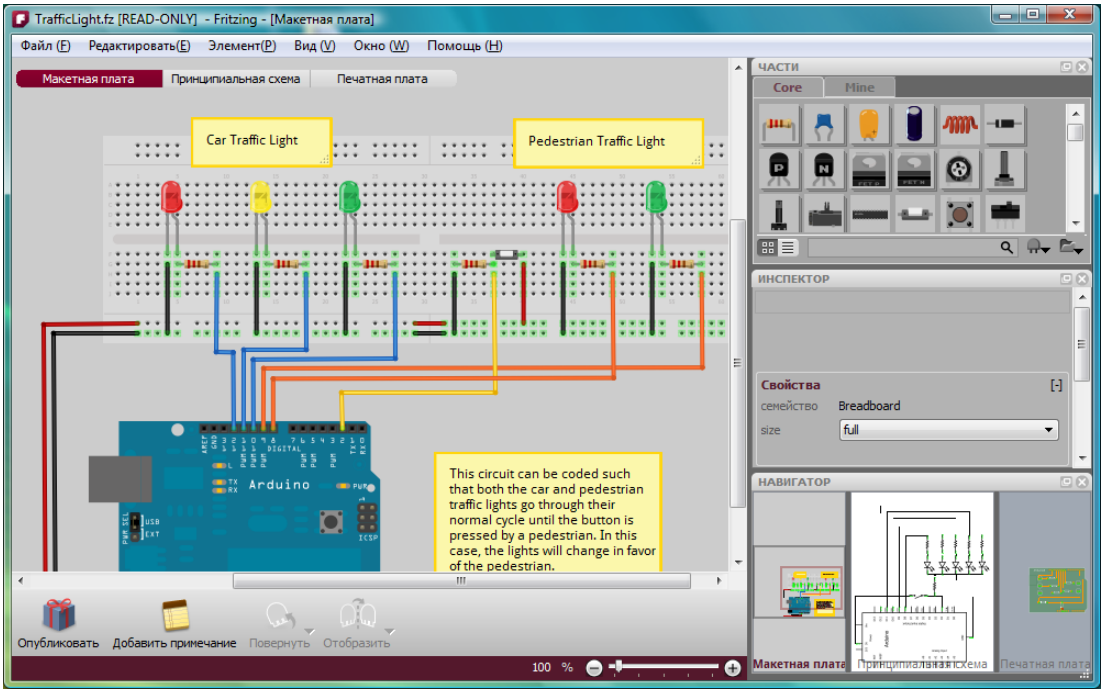


Рис. 8.40. Один из примеров, установленных с программой Fritzing

Перед тем, как приступить к макетированию схемы, можно проделать это в программе Fritzing.

Кроме макетной платы, а вы можете использовать такую плату, не требующую пайки, и в «живом» виде, кроме неё вы можете посмотреть принципиальную схему.

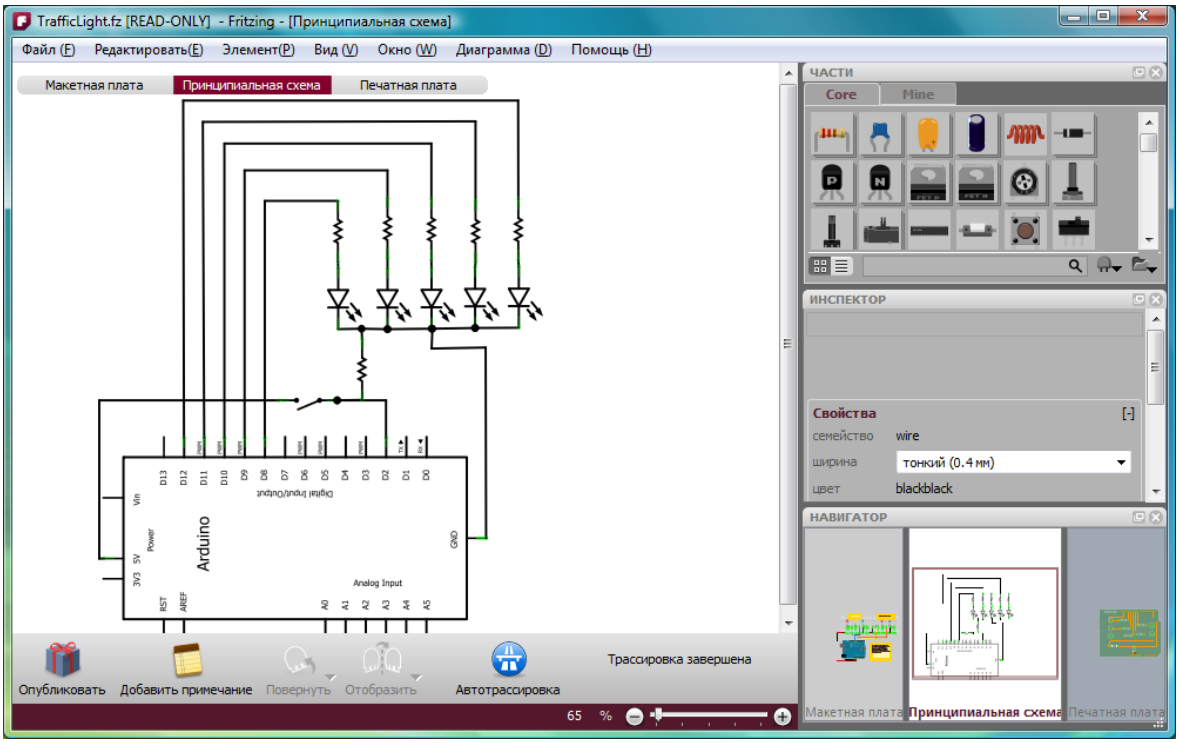


Рис. 8.41. Окно электрической схемы

И, если вы решите перенести схему с макетной платы на рабочую печатную плату, то увидеть, как выглядит печатная плата.

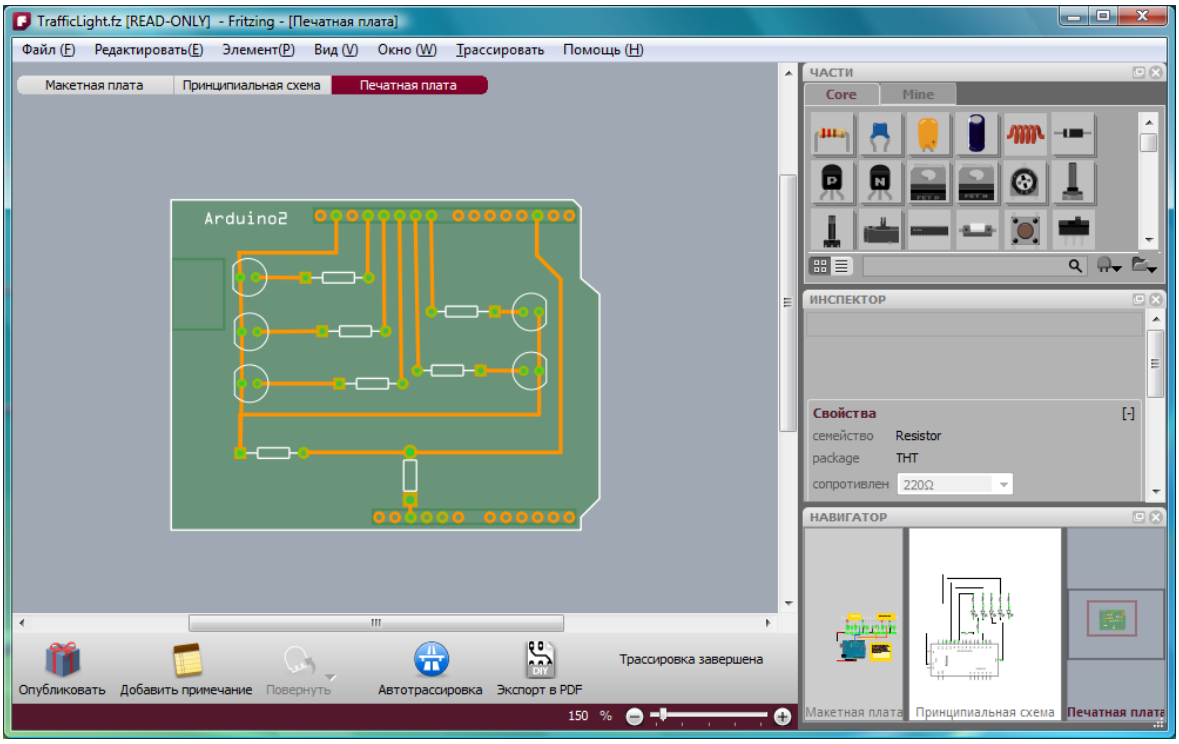


Рис. 8.42. Окно разводки печатной платы программы Fritzing

Такие платы, как платы расширения, легко добавляются к модулю Arduino, в результате вы

получаете готовое устройство на базе модуля Arduino.

Используя эту программу в дальнейшей работе, мы больше узнаем о ней. А пока, пока пора вернуться к рассказу.

Глава 9. Паровозик из Ромашково, продолжение

Взять быка за рога — это круто: сразу начать паять, «программить». Но и быки не лыком шиты — могут так наподдать, мало не покажется.

Поэтому оставим их в покое, мы не ковбои, и постараемся без спешки разобраться, что нужно сделать с «паровозиком и семафором». Для начала упростим задачу, отбросив обмен сигналами между ними. Положим, паровозик имеет устройство, которое раз в секунду отправляет инфракрасный сигнал, а семафор имеет другое устройство, которое постоянно «смотрит» на дорогу, а, увидев сигнал паровозика, зажигает зелёный свет на некоторое время, а затем вновь включает красный.

Теперь у нас две задачи для двух устройств. Пусть устройство для паровозика будет выполнено на PIC-контроллере. А устройство для семафора мы выполним на модуле Arduino.

Начнём с паровозика. Устройство «я свой» работает почти так же, как работают пульты управления телевизором, музыкальным центром или DVD-проигрывателем. Оно должно иметь излучатель ИК-сигнала. Для этой цели служит светодиод ИК диапазона. Но мы вначале испытаем красный светодиод АЛ307. Если «дальнобойности» этого излучателя не хватит, испытаем другой источник.

Пульты управления, работающие с инфракрасными излучателями, отправляют довольно сложные сигналы. При этом пульты разных производителей используют разные стандарты для кода управления. Однако почти все они, за редким исключением, излучают пачки импульсов, прерываемых паузами, на какой-то частоте в диапазоне от 20 кГц до 500 кГц. Используем частоту, её называют несущей частотой, 36-37 кГц, предполагая, что приёмник семафора мы построим с помощью микросхемы TSOP. Есть такая микросхема, в состав которой входит фотоприёмник, фильтр, выпрямитель и компаратор. Когда TSOP не получает сигнала нужной частоты, на его выходе высокий логический уровень. Когда сигнал приходит, на выходе микросхемы низкий логический уровень. Микросхема имеет три вывода: напряжение питания, выход и общий. Её легко будет подключить к модулю Arduino.

Теперь о сигнале, который будет излучать устройство паровозика. Мы не обязаны, как, впрочем и производители пультов управления, придерживаться какого-либо стандарта. Поэтому используем простой код: импульсы с частотой 36 кГц в течение 4 миллисекунд, пауза 1 миллисекунда, импульсы в течении 1 миллисекунды, пауза 1 миллисекунда, импульсы в течение 2 миллисекунд. Вот и весь сигнал.

Из тех программ, о которых мы говорили, с PIC контроллерами работает программа VirtualBreadboard. Но в ней удобно проверить работу кода, но не писать код программы. Поэтому для написания кода используем версию mikroBasic Lite, которая имеет некоторые ограничения, но для нас сейчас не существенные. Найти эту версию можно на сайте производителя:

http://www.mikroe.com/eng/products/view/9/mikrobasic-pro-for-pic/.

После запуска программы создадим новый проект (Project-New Project…). В этом нам поможет помощник.

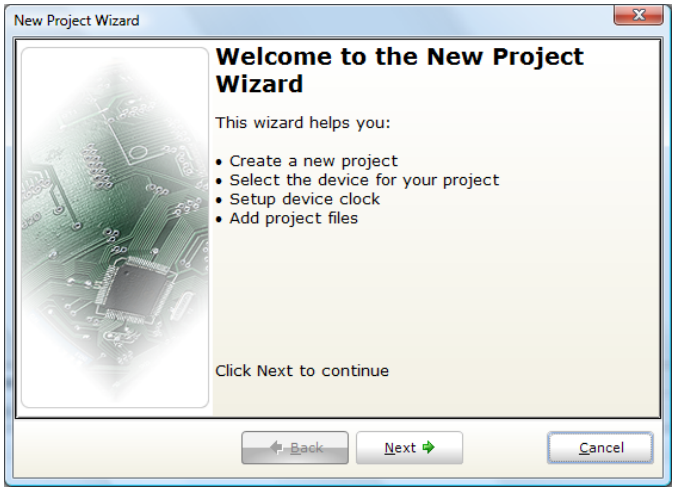


Рис. 9.1. Помощник создания нового проекта в microBasic

Нажимая клавишу «Next» проходим все этапы создания проекта: задание модели нашего контроллера, выбор тактовой частоты, имя проекта и место его хранения и т.д. Очень полезно поставить галочку в опции конфигурации.

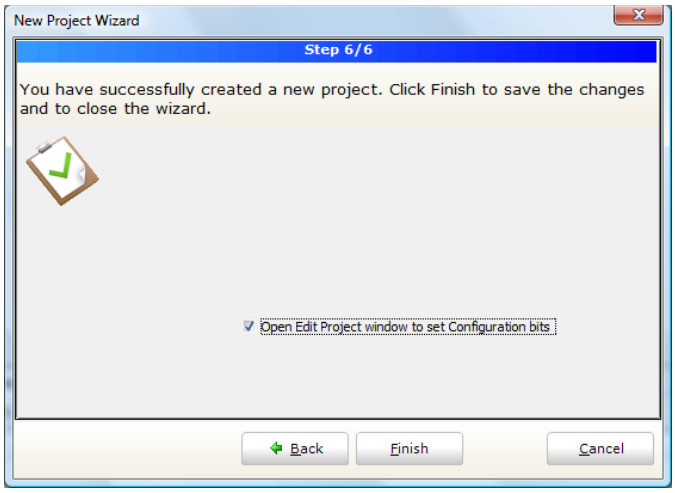


Рис. 9.2. Опция задания слова конфигурации микроконтроллера

В этом случае, закончив с созданием проекта, это происходит, когда вы нажимаете клавишу «Finish», вы попадаете в окно настройки конфигурации.

Я настраиваю микроконтроллер так, чтобы работал внутренний тактовый генератор, а всё остальное отключаю.

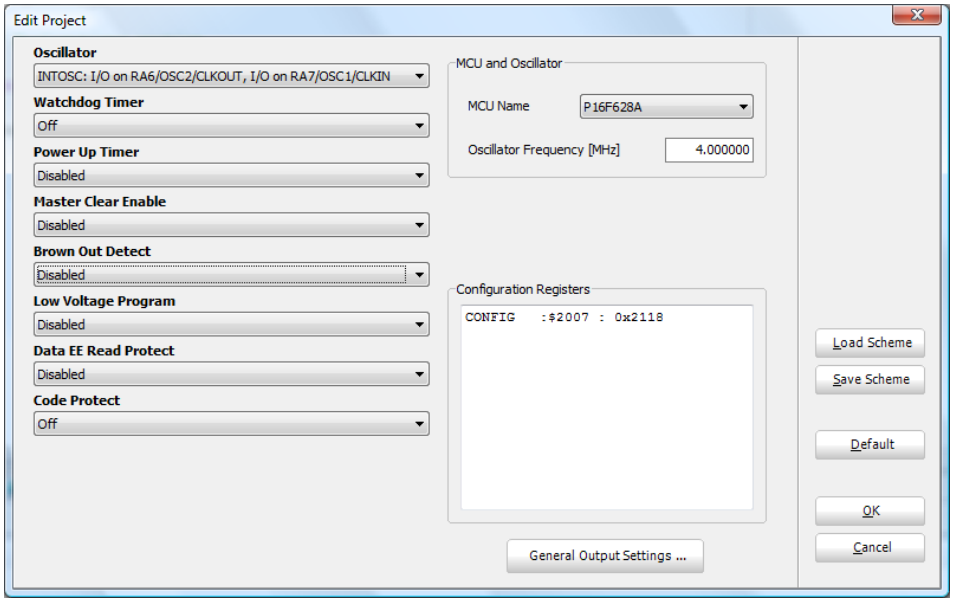


Рис. 9.3. Окно диалога создания слова конфигурации

Первое, что мне хотелось бы получить в программе – генерацию несущей частоты. Сначала рассчитаем период для частоты 36 кГц. Для этого разделим 1/36000, а результат дважды умножим на 1000, чтобы получить ответ в микросекундах. Это около 27 микросекунд. Значит 13 микросекунд – это половина периода.

Не мудрствуя лукаво, я открываю в примерах простую программу Blink, которую переделываю для себя. Выглядит она так:

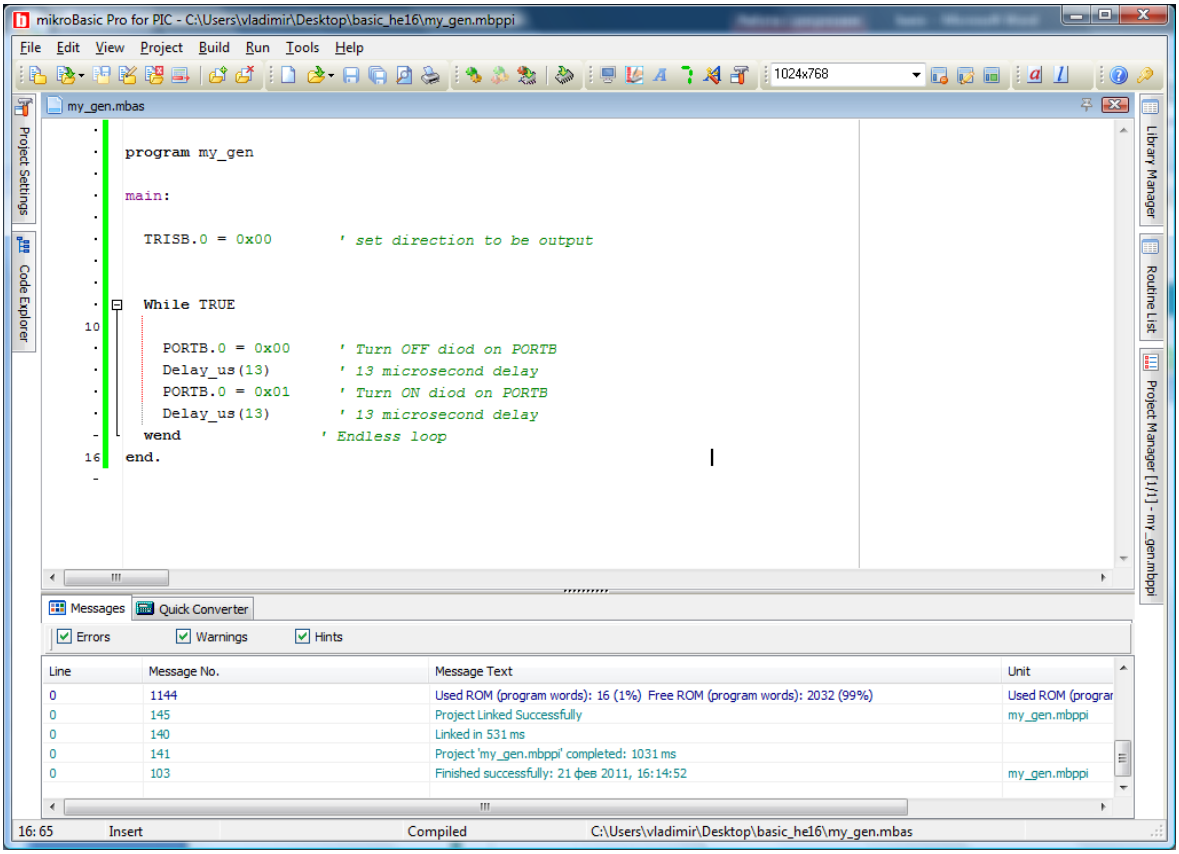


Рис. 9.4. Рабочее окно программы microBasic

Оттранслируем программу (Build-Build). И полученный hex-файл используем в программе, мы о ней говорили, VBB.

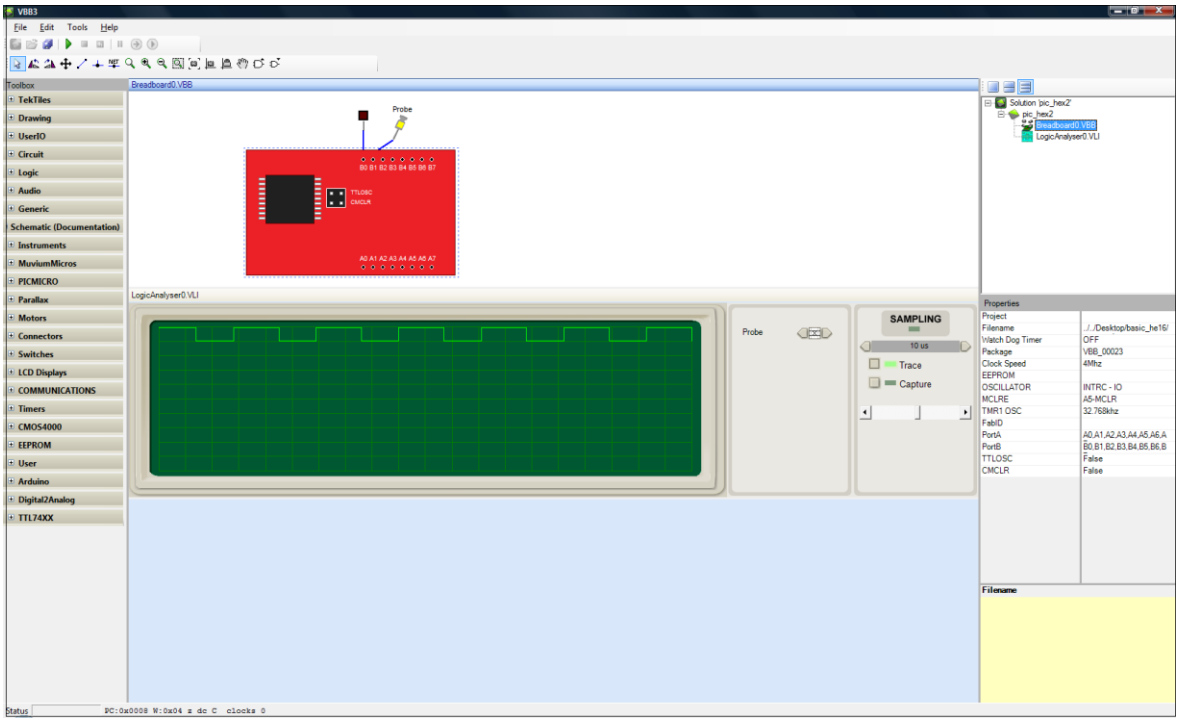


Рис. 9.5. Добавление hex-файла в программе VBB

Будем считать, что получили несущую частоту около 36 кГц. Далее, многие предпочитают использовать таймеры и прерывания, но я не вижу в данном случае такой необходимости. Мне нужно вначале рассчитать, сколько раз помещается период несущей частоты в 1 миллисекунде. Разделим 1000 на 27 и получим 37. Теперь мы знаем, что для первой пачки импульсов нам нужно в 4 раза больше, то есть, 148.

Добавим цикл for в программу, который отсчитает нам нужное количество периодов несущей частоты.

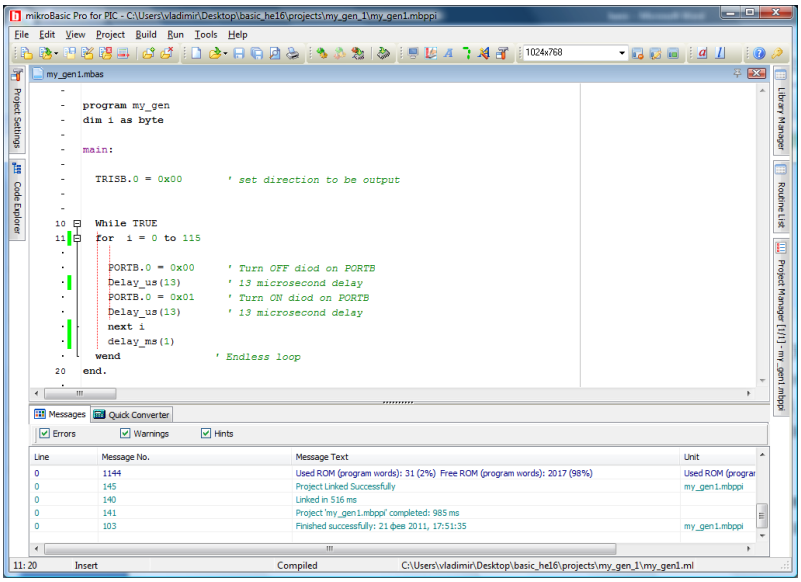


Рис. 9.6. Изменение программы в microBasic

Чтобы получить «правильную картинку», я подгоняю нужные значения. И эту пачку импульсов можно увидеть в VBB.

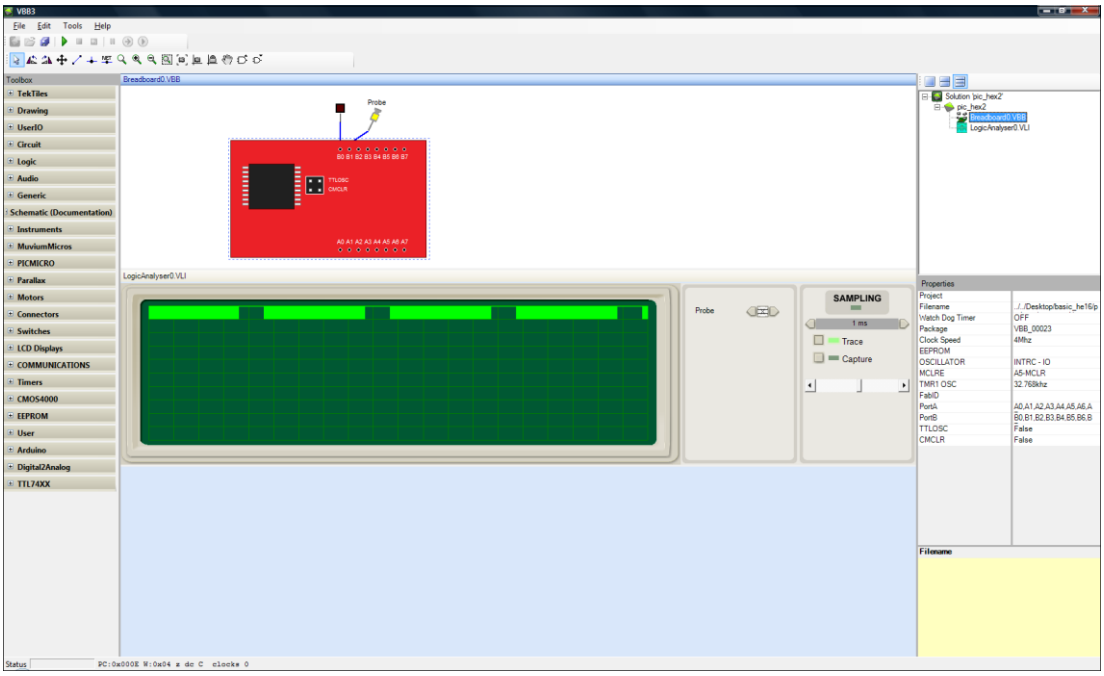


Рис. 9.7. Наблюдение за сигналом в логическом анализаторе VBB

Теперь можно сформировать полный сигнал.

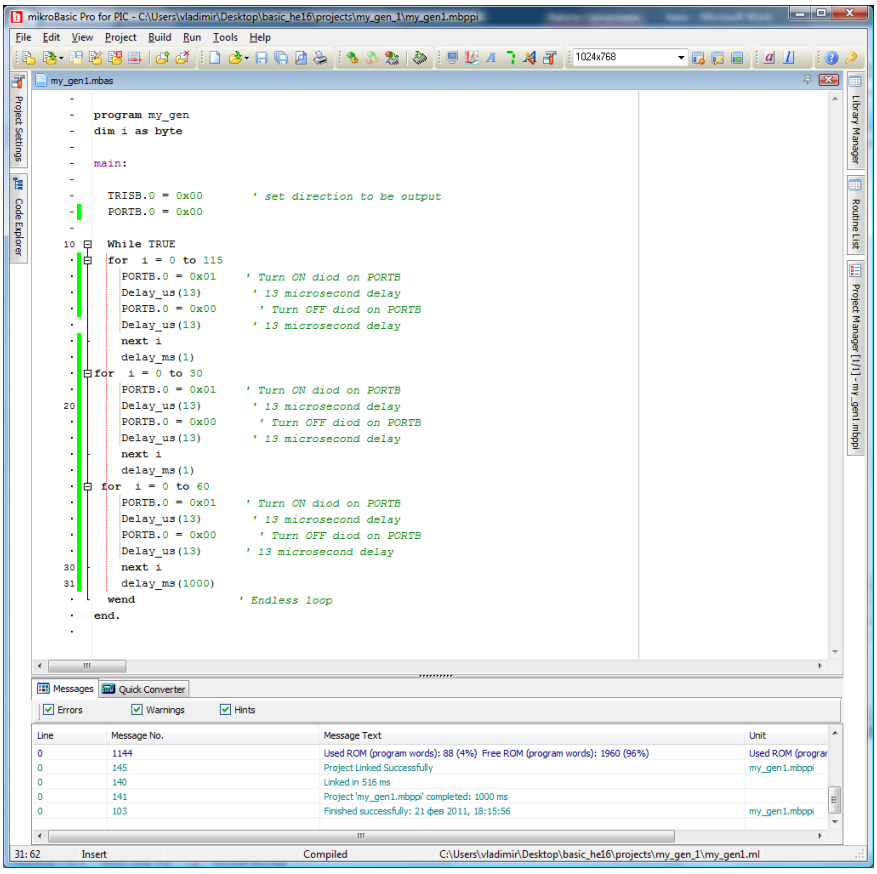


Рис. 9.8. Окончательный вид программы в microBasic

И проверить его с помощью VBB. Для проверки я последнюю паузу уменьшаю до 10 мС – удобнее наблюдать результат.

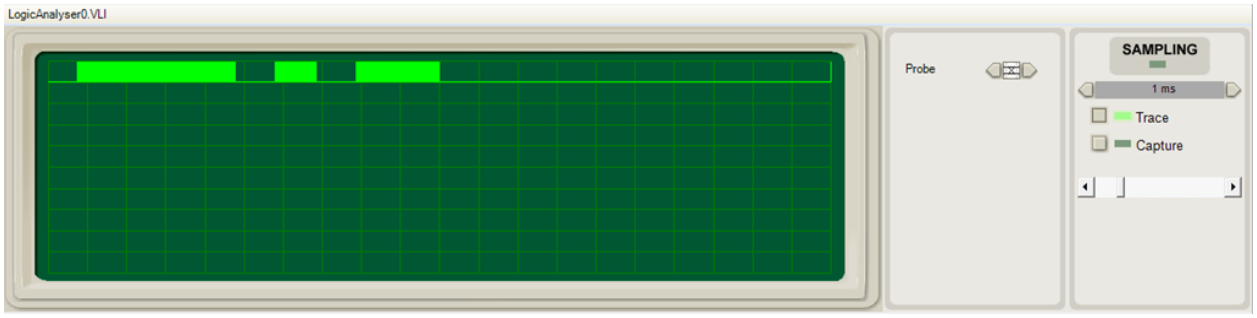


Рис. 9.9. Проверка вида сигнала в VBB

Вот так должны выглядеть наши пачки импульсов. Можно было бы сейчас собрать макет, но наблюдать пачки импульсов на обычном осциллографе, если паузы между посылками импульсов в 1 секунду, достаточно сложно. Поступим иначе. Отложим это до того момента, когда определимся со вторым устройством.

Несколько слов о «подгонке» для получения картинки. В конечном счёте, когда устройство отправки кода будет собрано, нас будут интересовать длительности пачек импульсов. Выполняя определение длительностей на языке высокого уровня, мы можем получить ошибочный результат, если не учтём, что для выполнения операции прохождения цикла процессору потребуется выполнить больше команд, чем нам кажется. Практически все команды языка высокого уровня после трансляции в машинные коды (операции для процессора) распадаются на несколько команд. А выполнение каждой из них требует времени. Поэтому общее время прохождения цикла может оказаться больше ожидаемого, а при большом количестве этих циклов… вы уже поняли, о чём я говорю. Так что, программирование устройства излучения кода для паровозика мы отложим до момента, когда сможем посмотреть полученный результат с помощью фотоприёмника, тогда и подправим длительности пачек импульсов окончательно.

Как мы уже говорили, в качестве фотоприёмника мы используем микросхему TSOP. Рекомендации по подключению микросхемы от производителя на рисунке ниже. Учтём их.

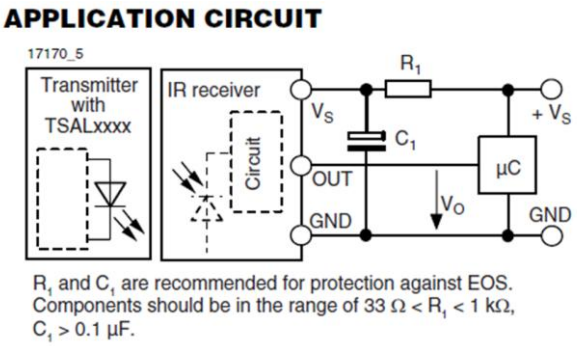


Рис. 9.10. Рекомендации производителя по подключению фотоприёмника TSOP

Однако до того, как начать работу над устройством приёма сигналов, я предлагаю посмотреть, что приходит с фотоприёмника TSOP, используя для этой цели модуль Arduino. Дело в том, что Arduino имеет встроенный аналогово-цифровой преобразователь. Как ясно из названия, последний преобразует аналоговый сигнал в цифровые данные. Если аналоговый сигнал – это только постоянное напряжение, то понятно, это напряжение переводится (каким-то образом) в цифровую форму. А когда аналоговый сигнал меняется? Аналогово-цифровой преобразователь делает, как бы мгновенные снимки, то есть, в какой-то момент времени «прочитывает» его, как

если бы это было постоянное напряжение. Последовательность таких «снимков» даёт последовательность цифровых данных. Программа на компьютере (или какой-то графический дисплей) может эти данные выводить в виде графика, как мы строим график кривой по точкам. В результате мы получим вид сигнала в таком виде, какой увидели бы на экране осциллографа. Мы используем компьютер и модуль Arduino в качестве осциллографа, чтобы посмотреть вид получаемого сигнала от фотоприёмника (когда, конечно, спаяем и запрограммируем передатчик!).

Программу для получения данных от модуля Arduino и построения кривой можно написать самостоятельно, очень интересная и полезная задача для освоения программирования, но можно, как это сделаю я, воспользоваться тем, что уже написано, что предназначено для работы с Arduino. Есть ряд готовых решений, из которых я выбираю:

http://www.compcar.ru/forum/showthread.php?t=4457

Код, загружаемый в модуль Arduino, можно скопировать на странице форума. Он невелик.

```
//oscilloscope
//http://compcar.ru
byte MyBuff[800];
unsigned int i=0;

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

    void loop()
    {

          for (i=0;  i < 800; i++)
          {
                MyBuff[i] = analogRead(0)/4;
          }
          Serial.write(MyBuff,800);
}
```

А программа для компьютера, oscilloscope.exe, не требует установки. Её можно расположить, например, в своей папке, сделать ярлык на рабочем столе и запускать, подключив запрограммированный модуль Arduino. У модуля используется аналоговый вход A0. Есть некоторые ограничения: напряжение на входе должно быть в диапазоне 0-5 В, не больше; и напряжение не должно быть отрицательным; верхняя рабочая частота, обозначенная автором программ, 4 кГц. Этого должно хватить для многих экспериментов. Вот как выглядит сигнал на мониторе, если коснуться рукой входа осциллографа.

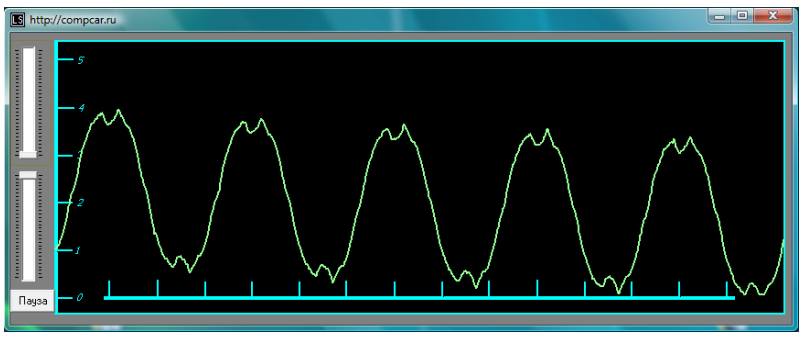


Рис. 9.11. Работа одной из программ осциллографа на базе модуля Arduino

Это наводки с частотой 50 Гц. Думаю, что для чтения сигнала с фотоприёмника этого

осциллографа будет достаточно. Пришла пора спаять передатчик ИК кода, добавить фотоприёмник к осциллографу и посмотреть результат. Если понадобится, то подогнать длительность фрагментов полученного сигнала.

Перед пайкой воспользуемся программой Fritzing, чтобы нарисовать необходимые подключения для проведения экспериментов и, возможно, для последующей работы с паяльником.

В программе Fritzing нет нужной мне микросхемы TSOP. Это неприятно, но не страшно – мы добавим нужное сами. Для рисования, а понадобится нарисовать микросхему, используем графический редактор Inkscape. Сам я, не художник я, признаюсь, чаще использую простейший графический редактор как Paint в Windows. Иногда использую Gimp (его аналог в Windows – Photoshop). Все эти графические редакторы работают как в Linux, так и в Windows. Но они, если я не ошибаюсь, не поддерживают формат svg векторной графики, а Inkscape для векторной графики и предназначен. Он тоже есть в версии для Windows, а скачать его можно: http://inkscape.org.

Для создания рисунка в векторном графическом редакторе я считаю, лучше всего использовать прототип, есть подходящий – voltage_regulator. Рисунок можно найти в папке программы Fritzing по адресу, показанному на рисунке:

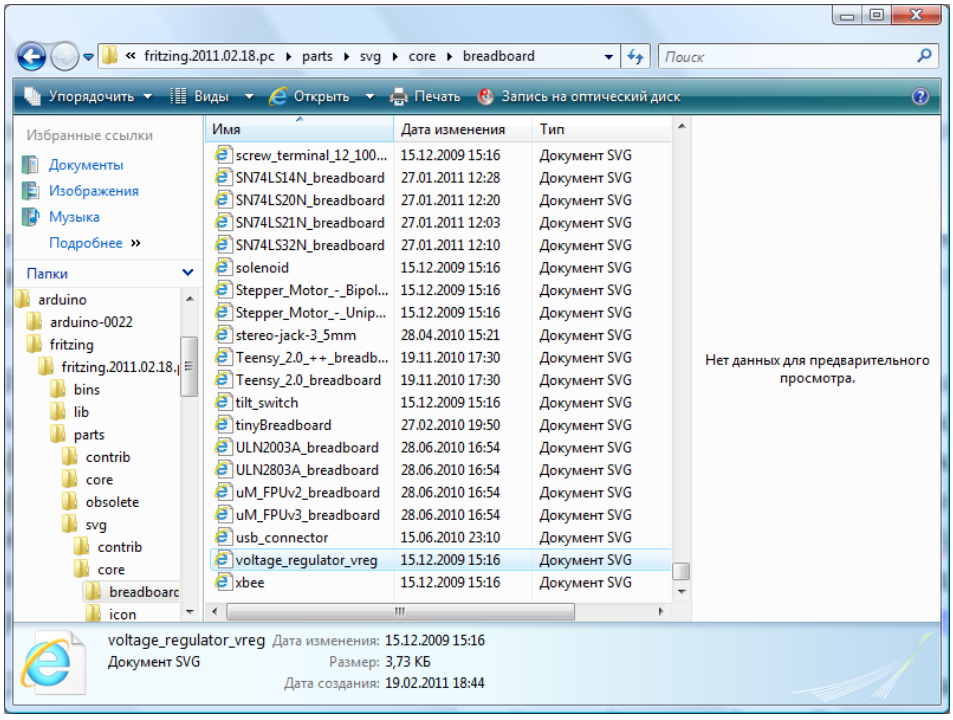


Рис. 9.12. Рисунок элемента регулятора напряжения

Открыв этот файл в Inkscape, преобразуем его во что-то похожее на TSOP. Можно воспользоваться рекомендациями, которые даются на сайте программы Fritzing, попасть туда можно из раздела «Помощь» основного меню (программы Fritzing!), где есть пункт «Онлайн-руководство». Когда web-браузер откроет страницу руководства, то можно найти главу, которая называется «Creating Custom Parts». В этой главе подробно описывается, как создать свой элемент. Следуя советам, запускаем Fritzing, выбираем стабилизатор напряжения 7805 (в качестве прототипа), переносим его на макетную плату и заходим в раздел редактирования.

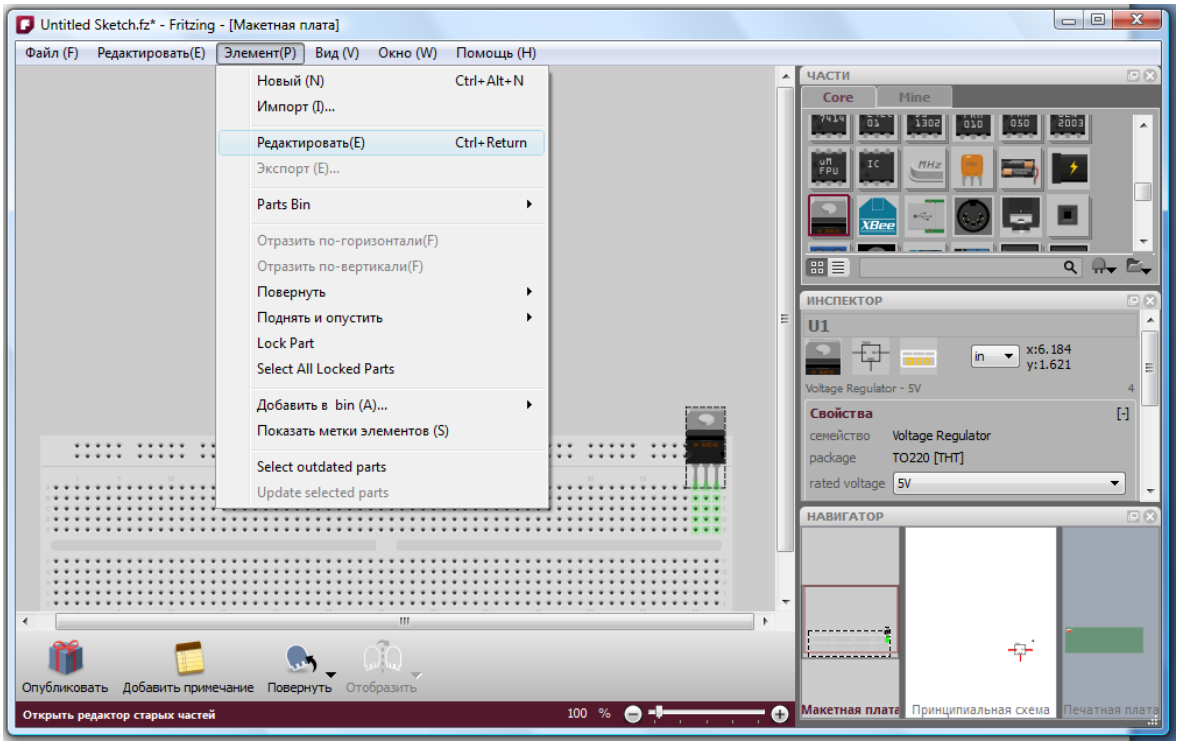


Рис. 9.13. Пункт редактирования элемента в программе Fritzing

Как и написано в руководстве, щёлкаем по заголовку, где меняем voltage_regulator на TSOP173x.

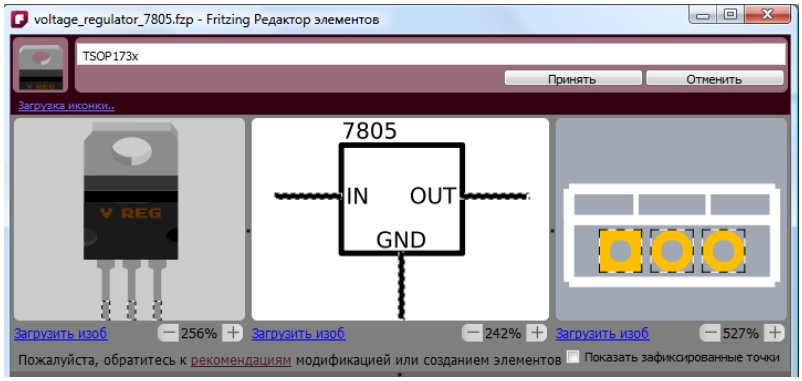


Рис. 9.14. Настройка рисунков для добавления нового элемента

Далее вы переходите к редактированию графики. Советую следовать всем рекомендациям руководства, конечно, если вся эта работа вам интересна. Я же только обозначу то, что делаю.

Загружаю в Inkscape рисунок стабилизатора из того места, о котором шла речь выше. В папке user программы Fritzing перед этим я создаю пустые папки с именами, которые видел в папке core. Далее, разгруппировав рисунок, удаляю ненужное, перемещаю, растягиваю и дорисовываю, используя средства графического редактора, всё необходимое, чтобы получить свой рисунок. Последнее, что я делаю, группирую объект, выделив его (Объект-сгруппировать). Сохраняю его в пользовательской папке как простой (есть такой формат) svg рисунок с именем tsop_breadboard.svg.

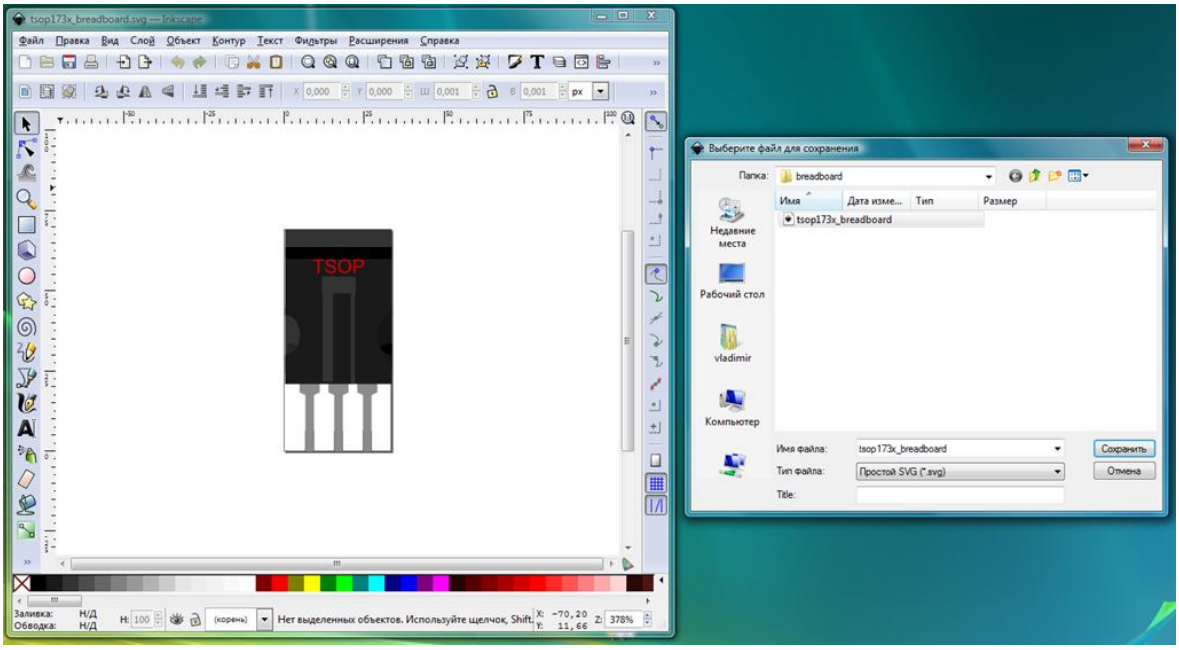


Рис. 9.15. Создание нового рисунка в программе Inkscape

Открываю файл иконки стабилизатора напряжения. Выделяю и удаляю всё. Второе окно графического редактора я не закрывал, поэтому выделяю картинку, щёлкнув по ней, копирую и вставляю в качестве иконки. Сохранив эти файлы, я могу использовать их при загрузке изображений в редакторе компонентов Fritzing.

Остаётся подправить и сохранить файл schematic. Не следует менять соединения, достаточно исправить имена, но при желании можно переделать всё полностью, как это описано в руководстве.

Теперь я готов заменить все изображения и сохранить объект как новый. При выходе из программы Fritzing я отказываюсь от сохранения проекта, но соглашаюсь сохранить новый объект.

При следующем запуске программы Fritzing на закладке «Mine» появляется новый элемент.

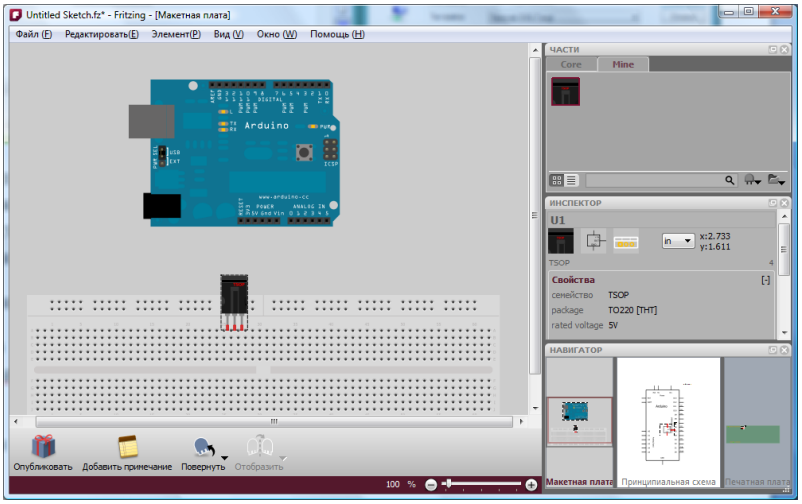


Рис. 9.16. Появление нового элемента на закладке Mine

И ничто уже не мешает нарисовать схему эксперимента по чтению кода с устройства передачи ИК-команды, используя программу осциллографа и аналоговые возможности модуля Arduino.

Итак, вот вид эксперимента.

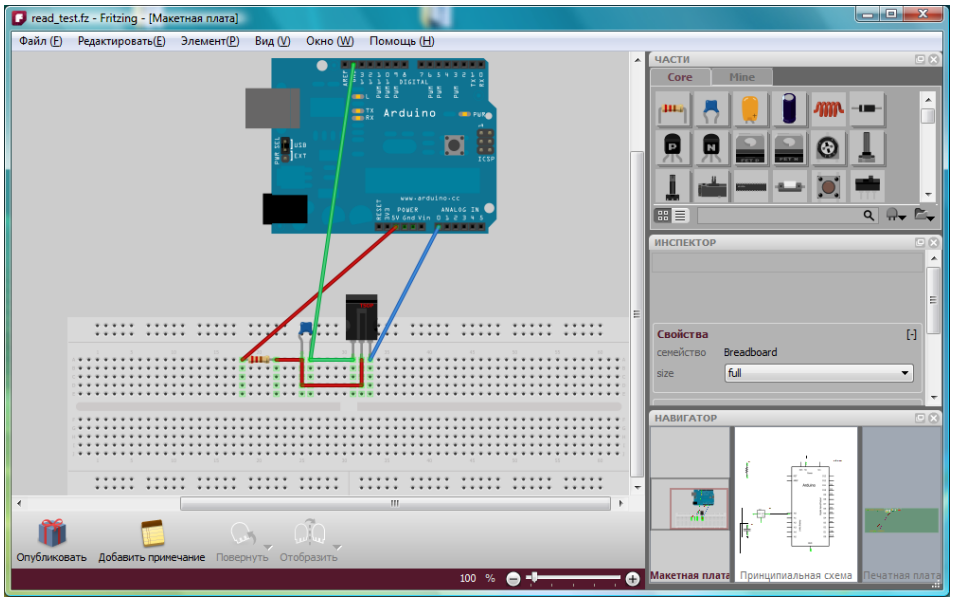


Рис. 9.17. Подготовка соединений для эксперимента

Так выглядит электрическая схема.

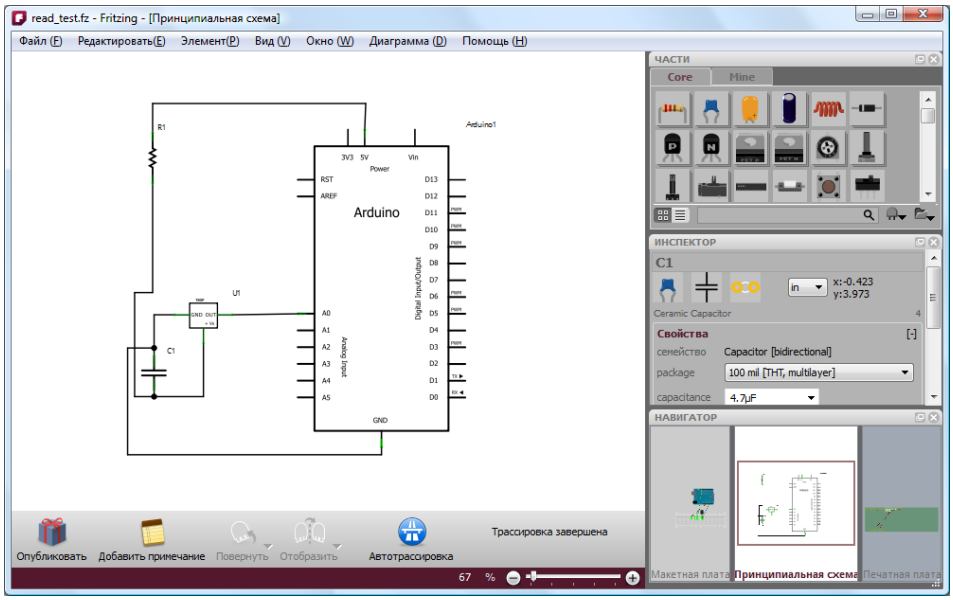


Рис. 9.18. Электрическая схема соединений модуля с фотоприёмником

А так может выглядеть печатная плата (хотя в данном случае она совсем лишняя сущность).

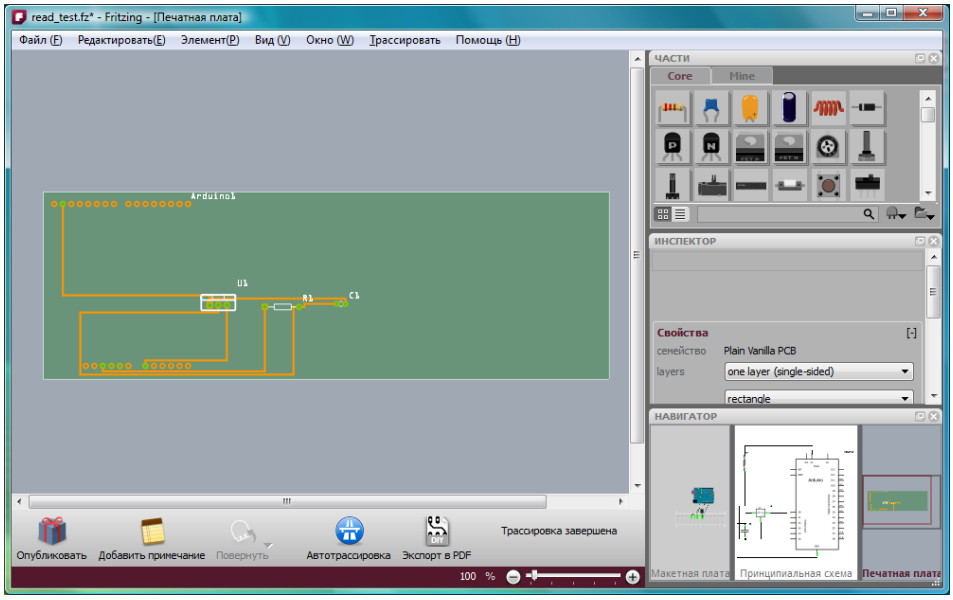


Рис. 9.19. Возможный вид печатной платы расширения модуля Arduino

Вместе с тем я рад, что потратил некоторое время на подготовку к эксперименту. Пока я занимался с программой Fritzing, я понял, что используя программу осциллографа, о которой писал раньше, я попаду в ситуацию такую же, что и с обычным осциллографом: короткие пачки импульсов кода и длинная пауза между ними. Увидеть, а особенно разглядеть что-то будет, мне кажется невозможно. Что же делать?

Когда я просматривал использование модуля Arduino в качестве осциллографа, я видел ещё одну программу. От её применения меня остановило то, что файл для загрузки в модуль был не в формате языка Arduino, а был hex-файлом. То есть, файлом, предназначенным для загрузки с помощью программатора. Программа называется xoscillo:

https://code.google.com/p/xoscillo/

Но, если я раньше я отверг эту программу, то теперь ситуация изменилась – программа, возможно, решит проблему, без решения которой нет смысла проводить эксперимент. Прочитав всё, что сказано о работе программы и подготовке к её реализации на модуле Arduino, я скачал программу и hex-файл, загружаемый в модуль. Всё оказалось не так сложно, как это воображаешь, когда читаешь описание того, что нужно сделать. Хотя и там есть некоторые детали, которые приходится менять.

В корневой директории диска C:\ у меня есть копия программы Arduino. Она лежит в папке с именем arduino. В корневую же директорию я копирую файл arduinooscillo.cpp.hex. На сайте xoscillo есть даже команда (её следует ввести после запуска в Windows командной строки), её можно скопировать и вставить в окно командной строки. Но приходится внести некоторые изменения (включая то, что виртуальный COM-порт для работы через USB у меня com5):

```
  c:\arduino\arduino-0022\hardware\tools\avr\bin\avrdude -c stk500v1 -p m168
-P com5 -b 19200 -U flash:w:c:\arduinooscillo.cpp.hex:i -C
c:\arduino\arduino-0022\hardware\tools\avr\etc\avrdude.conf
```

Какие изменения я вносил в команду, скопированную с сайта? Конечно, я изменил путь к файлу avrdude, убрал перед этим файлом в команде всё, что было до папки bin. Кроме этого я добавил после hex-файла опцию :i и указал полный путь к файлу конфигурации avrdude.

После этого команда полностью выполняется и не даёт ошибок, которые я получал, пытаясь использовать команду с сайта. И, конечно, модуль Arduino я подключил до выполнения команды.

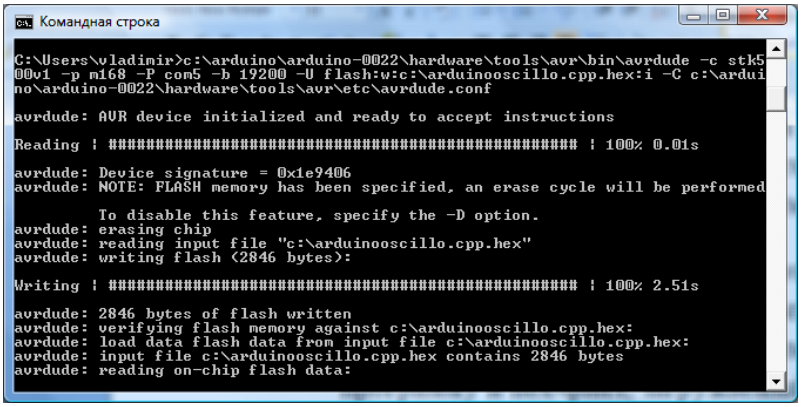


Рис. 9.20. Выполнение команды в командной строке для загрузки hex-файла

Сама программа xoscill лежит у меня на рабочем столе в папке с этим же именем. Открыв эту папку, я запускаю программу обычным образом. В разделе «File» основного меню выбираю «New Arduino».

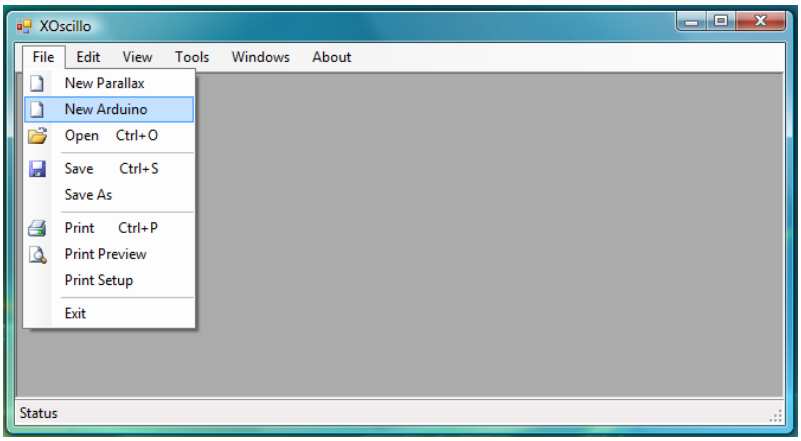


Рис. 9.21. Запуск программы осциллографа

И через некоторое время появляется следующее:

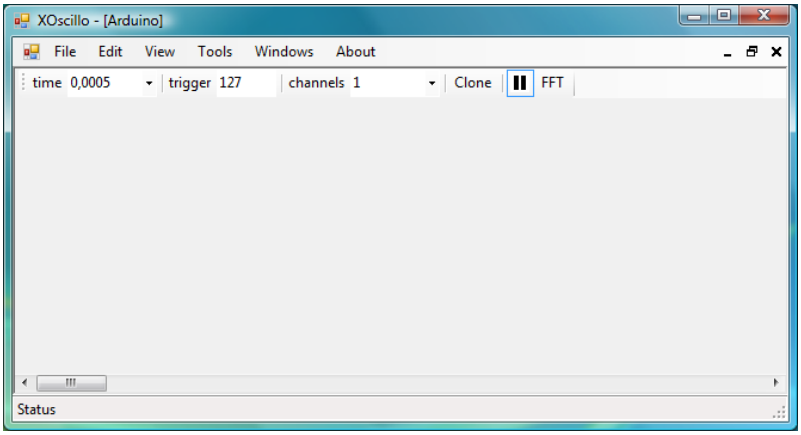


Рис. 9.22. Подключение программы к модулю Arduino

У модуля Arduino, подготовленного мною к предыдущим опытам, есть вывод от аналогового входа A0. Коснувшись его рукой, я получаю картину наводок.

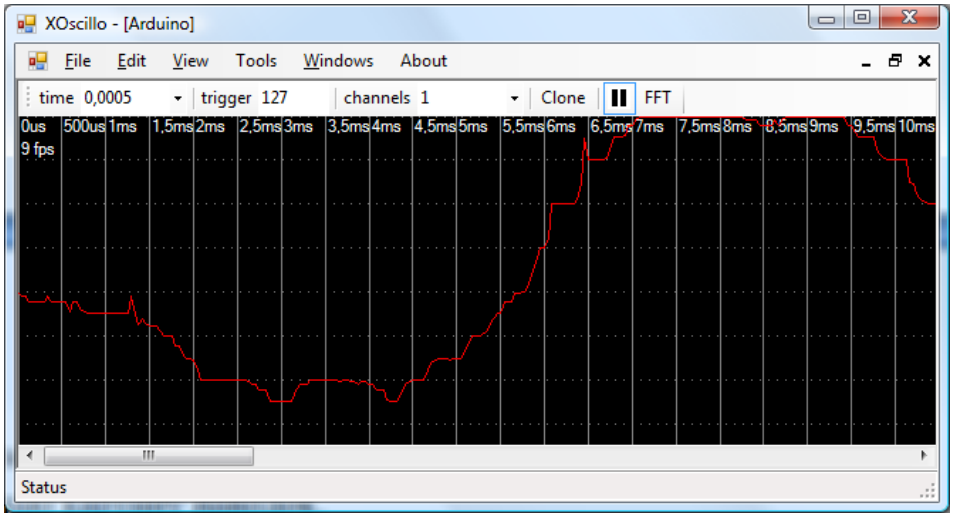


Рис. 9.23. Вид наводок на экране осциллографа на базе модуля Arduino

Когда я убираю руку от вывода, картинка замирает. Теперь я могу её сохранить: «File-Save as…», имя и место сохранения я ввожу сам. Позже я могу, запустив программу, открыть этот файл и ещё раз внимательно рассмотреть получившуюся диаграмму.

Пока я не вполне освоился, и для выхода из программы я нажимаю обычный значок «закрыть» и ещё раз касаюсь рукой вывода аналогового входа. Программа закрывается. Я вновь открываю программу, но в этот раз загружаю файл, который перед этим сохранил. Да, я могу его просмотреть ещё раз. Я могу изменить, выбрав новый, интервал времени деления.

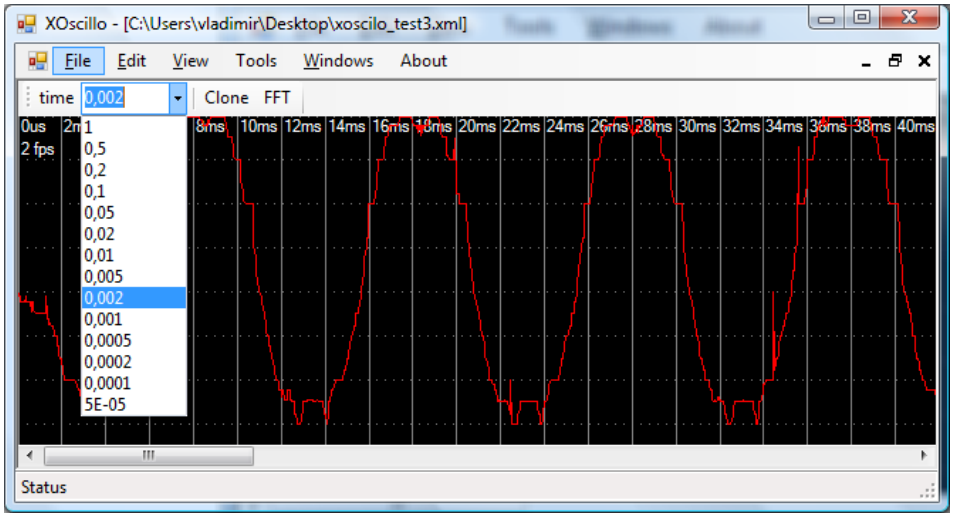


Рис. 9.24. Выпадающий список времени отсчёта

Полагаю, эта программа будет удобнее, и она позволит мне провести нужный эксперимент, чтобы увидеть на выходе фотоприёмника сигнал, передаваемый устройством излучения ИК-кода.

Я потратил некоторое время на работу с первой программой осциллографа. Это так. Но она проще и позволяет наблюдать периодические сигналы, что тоже может понадобиться. Я потратил некоторое время, разбираясь с тем, как использовать вторую программу виртуального осциллографа, но она решает мои проблемы. Более того, открою вам ещё одну тайну – теперь, используя метод записи в Arduino hex-файла для осциллографа, вы (и я за компанию) можете создать такой файл, например, в программе AVRStudio, а загрузить в модуль Arduino с помощью утилиты avrdude.

И ещё одно, что осталось для меня интересным – будет ли этот осциллограф работать в Linux?

На сайте проекта моё внимание привлекло то, что программа работает и в Windows, и в Linux. С примечанием об использовании Mono. И впрямь, программа работает. И, что более всего приятно, что всё просто. В ALTLinux 5.1 и Fedora 14 Mono оказалось установлено со всеми необходимыми библиотеками и приложениями. Достаточно было использовать команду:

```
mono /home/vladimir/Xoscillo/XOscillo.exe
```

Программа xoscillo у меня в домашней папке. Создав команду запуска (в Windows ярлык), можно запустить осциллограф, получить осциллограмму, сохранить её и рассмотреть при следующем запуске xoscillo.

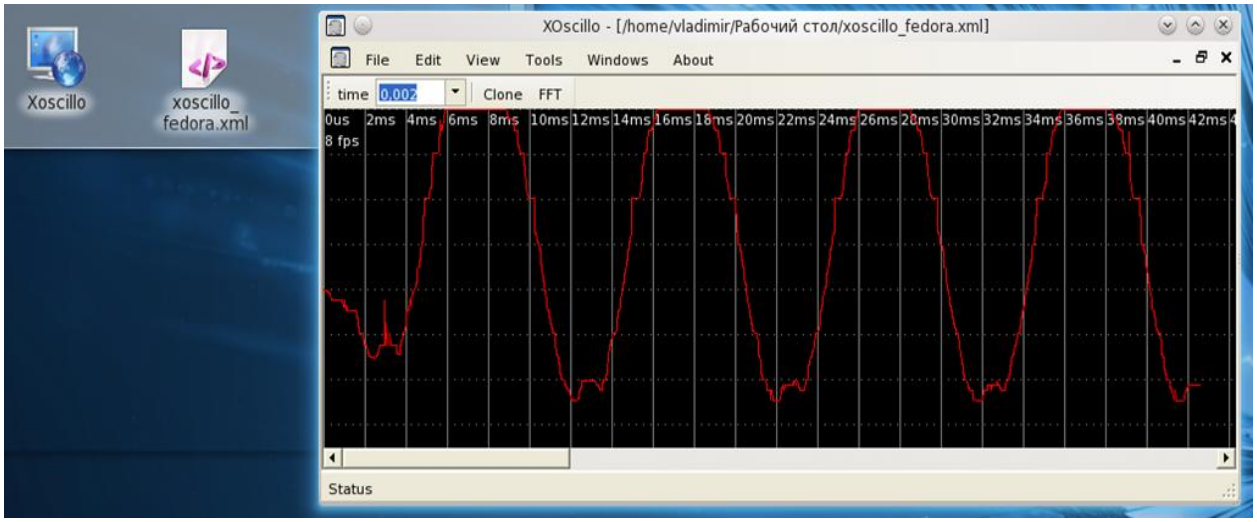


Рис. 9.25. Работа программы осциллографа в дистрибутиве Fedora 14

В openSUSE прошло не так гладко – я попытался загрузить только ядро Mono, но этого не хватило для работы осциллографа, поэтому, не хватило и у меня терпения, я установил mono-complete, объём, конечно, больше, но загрузка решает все проблемы.

Что осталось сделать? Спаять, запрограммировать PIC-контроллер и проверить ИК-код. Перед программированием PIC-контроллера, поскольку я использую макетную плату, оставшуюся от предыдущих экспериментов, где установлена панелька для микросхемы, и где есть светодиод, транзистор и резистор, мне проще заменить в программе вывод B0 на A0, как это сделано на макете…

Макетная плата передатчика ИК-кода готова и красный светодиод АЛ307, который я использую при прямом токе 40 мА (средний за период ток меньше), раз в секунду мигает. К модулю Arduino подключён фотоприёмник, выход которого соединён с аналоговым входом A0. Запускаем программу осциллографа, для проверки я делаю это в Windows и в openSUSE, и наблюдаем сигнал.

Кстати, используя обычный светодиод, я заинтересован в определении расстояния, с которого сигнал «видит» фотоприёмник. Сейчас это примерно 10-15 см. Если дальности не хватит, можно будет разбить сопротивление, последовательно установленное со светодиодом, на два, сумма сопротивлений которых равна первоначальному, и одно из сопротивлений зашунтировать конденсатором. Это увеличит дальность считывания сигнала (вернее, увеличит «дальнобойность» светодиода за счёт того, что появится короткий импульс большего тока).

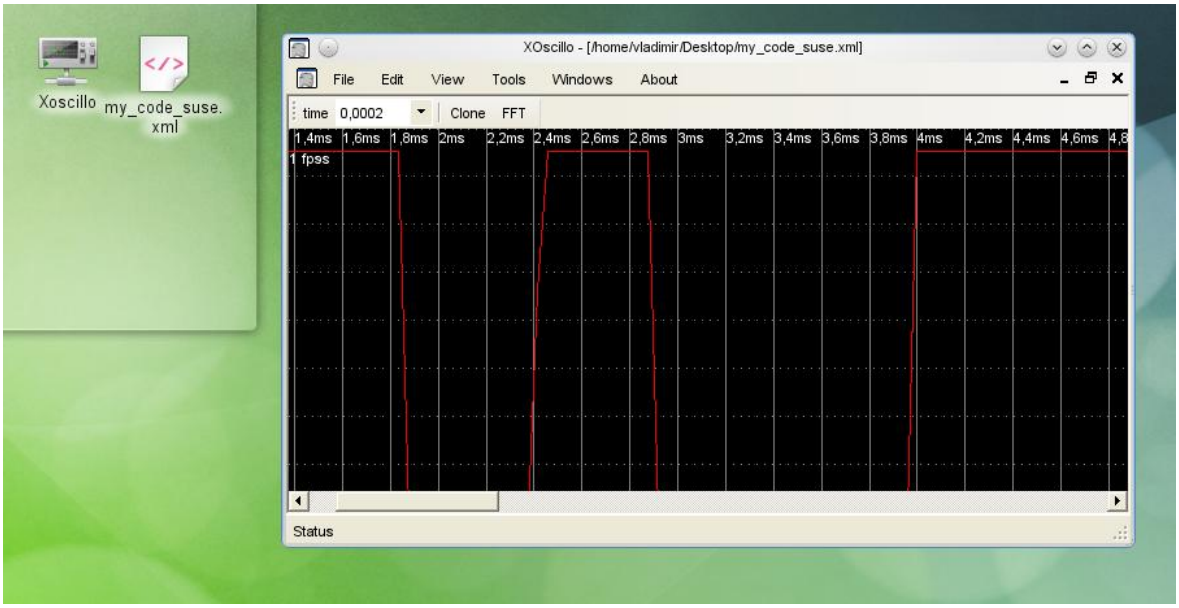


Рис. 9.26. Работа программы осциллографа в openSUSE

Напомню, что фотоприёмник TSOP в отсутствии несущей частоты на выходе поддерживает высокий уровень, а при наличии несущей переходит в состояние низкого логического уровня.

Осциллограмма похожа на то, что я ожидал бы, если бы не два момента: я не вижу первой пачки импульсов; времена вдвое меньше ожидаемых.

Я могу предположить, что осциллограф запускается несколько позже, чем при приходе первой пачки импульсов. Так ли это? Ответ я могу получить, изменив программу излучения так, чтобы интервалы между кодами были меньше, не 1 секунда, а, скажем, 10 миллисекунд. Посмотрим, как выглядит такой вариант.

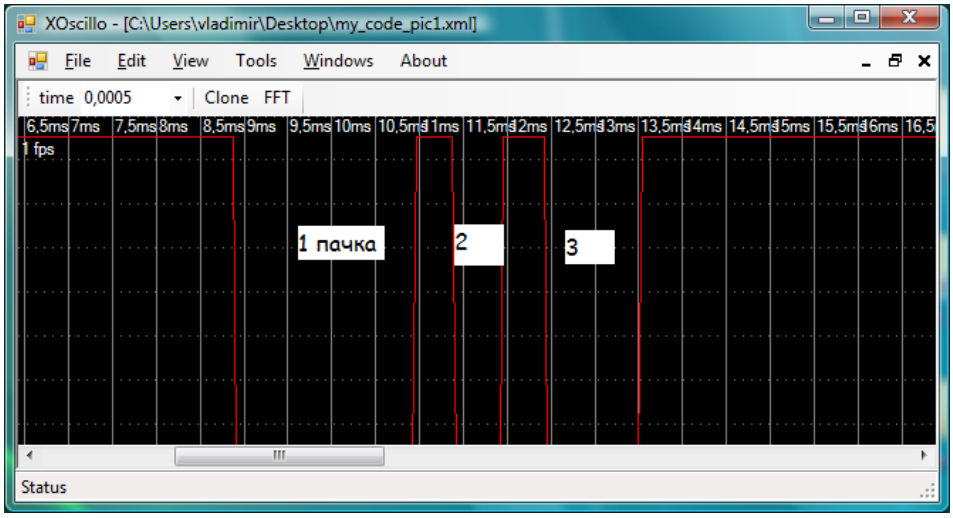


Рис. 9.27. Проверка приёма пачек импульсов сигнала фотоприёмником

Теперь все пачки импульсов на месте, а времена… я не калибровал осциллограф Xoscill. Можно, конечно, это сделать сейчас, но можно отложить до того времени, когда возникнут проблемы с работой приёмного устройства. Если они возникнут.

Глава 10. С чего начинаются роботы?

Пришло время написать программу для семафора, который будет работать под управлением модуля Arduino.

Как должна работать программа? В самом общем виде: если семафор «видит» ИК-код, он включает зелёный свет, который выключает через несколько секунд, включая красный.

Видеть ИК-код семафор будет с помощью фотоприёмника TSOP. Но не просто видеть какой-либо инфракрасный сигнал, а «прочитывать» придуманный нами код. Если код совпадает, то семафор переключается.

У осциллографа мы подключали фотоприёмник к аналоговому входу. Но на выходе фотоприёмника сигнал, который удобно читать цифровым входом. Как вы помните, а если забыли, то посмотрите программу ещё раз, похожую работу выполняет программа для модуля Arduino, которая называется «Button, кнопка». Мы вполне можем себе представить работу фотоприёмника, как работу кнопки, которую нажимают на определённые промежутки времени. Поэтому, взяв программу за основу, мы модифицируем её. Вот эта программа на языке Arduino.

```
// set pin numbers:
const int buttonPin = 2;     // номер вывода подключённой кнопки
const int ledPin =  13;      // номер вывода светодиода
// variables will change:
int buttonState = 0;         // переменная для чтения состояния кнопки

void setup() {
  // инициализация вывода светодиода, как выхода
  pinMode(ledPin, OUTPUT);
  // инициализация вывода кнопки, как входа
  pinMode(buttonPin, INPUT);
}

void loop(){
  // читаем состояние кнопки, как значение
  buttonState = digitalRead(buttonPin);

  // проверяем, нажата ли кнопка
  // если да, то buttonState имеет значение HIGH:
  if (buttonState == HIGH) {
    // включаем LED
    digitalWrite(ledPin, HIGH);
  }
  else {
    // выключаем LED
    digitalWrite(ledPin, LOW);
  }
}
```

В первую очередь изменим задание выводов.

```
// установка выводов

const int photoPin = 2;  // номер вывода, к которому подключён фотоприёмник
const int ledPinGreen =  13;      // номер вывода зелёного светодиода
const int ledPinRed =  12;       // номер вывода красного светодиода
```

Зададим переменную для приёма состояния цифрового входа D2.

```
  // переменная для хранения состояния входа
  int photoState = HIGH;            // переменная для чтения состояния
фотоприёмника
```

В разделе инициализации запишем:

```
void setup() {
  // инициализируем выходы для светодиодов
  pinMode(ledPinGreen, OUTPUT);
  pinMode(ledPinRed, OUTPUT);
  // инициализируем вход для фотоприёмника
  pinMode(photoPin, INPUT);
}
```

Основная программа будет похожа на оригинал — мы постоянно читаем вход, когда он переходит в состояние с низким логическим уровнем, мы будем читать код.

```
void loop(){
    // включаем красный свет
    digitalWrite(ledPinRed, HIGH);
  // читаем состояние входа
  photoState = digitalRead(photoPin);

  // проверяем есть ли пачка импульсов
  // если есть, то photoState становится LOW
  if (photoState == LOW) {
    // начинаем читать код и переключать светодиоды, если код «свой»
  }
```

Теперь, как мы будем читать код? Мы получили переход на цифровом входе в низкое состояние, а с правильным кодом это состояние продлится 4 мС. Затем последует пауза (нет несущей частоты), когда вход перейдёт в высокое состояние. Пауза длится 1 мС. Для проверки нашего кода достаточно сделать паузу в 4.5 мС и проверить, перешёл ли вход в высокое состояние?

Затем будем проверять состояние, стараясь попадать в «середину» состояний после переходов (нашего кода), если это не так, то возвращаемся в начало программы.

```
if (photoState == LOW) {
    // начинаем читать код и переключать светодиоды, если код «свой»
    delayMicroseconds(4500);      // ждём 4.5 мС
    photoState = digitalRead(photoPin);
    if (photoState == HIGH){
           delay(1);      // ждём 1 мС
           photoState = digitalRead(photoPin);
           if (photoState == LOW){
                  delay(1);      // ждём 1 мС
                  photoState = digitalRead(photoPin);
                  if (photoState == HIGH){
                         delay(1);      // ждём 1 мС
                         photoState = digitalRead(photoPin);
                         if (photoState == LOW){
                                delayMicroseconds(3500);      // ждём 3.5 мС
                                if (photoState == HIGH){
                                digitalWrite(ledPinRed, LOW); // выключаем красный
                                digitalWrite(ledPinGreen, HIGH);//включаем зелёный
                                delay(2000); // зелёный 2 секунды
                                digitalWrite(ledPinGreen, LOW);//выключаем зелёный
                                }
                         }
                  }
           }

    }
}// иначе ничего не делаем
```

И теперь соберём всю программу вместе (в тексте программы есть ошибка, о которой мы поговорим позже!).

```
// установка выводов

const int photoPin = 2;     // номер вывода, к которому подключён фотоприёмник
const int ledPinGreen =  13;       // номер вывода зелёного светодиода
const int ledPinRed =  12;           // номер вывода красного светодиода

int photoState = HIGH;      // переменная для чтения состояния фотоприёмника


void setup() {
  // инициализируем выходы для светодиодов
  pinMode(ledPinGreen, OUTPUT);
  pinMode(ledPinRed, OUTPUT);
  // инициализируем вход для фотоприёмника
  pinMode(photoPin, INPUT);
}

void loop(){
    // включаем красный свет
    digitalWrite(ledPinRed, HIGH);
  // читаем состояние входа
  photoState = digitalRead(photoPin);

  // проверяем есть ли пачка импульсов
  // если есть, то photoState становится LOW
if (photoState == LOW) {
    // начинаем читать код и переключать светодиоды, если код «свой»
    delayMicroseconds(4500);  // ждём 4.5 мС
    photoState = digitalRead(photoPin);
    if (photoState == HIGH){
          delay(1);     // ждём 1 мС
          photoState = digitalRead(photoPin);
          if (photoState == LOW){
                delay(1);     // ждём 1 мС
                photoState = digitalRead(photoPin);
                if (photoState == HIGH){
                      delay(1);     // ждём 1 мС
                      photoState = digitalRead(photoPin);
                      if (photoState == LOW){
                            delayMicroseconds(3500);  // ждём 3.5 мС
                            if (photoState == HIGH){
                            digitalWrite(ledPinRed, LOW); // выключаем красный
                            digitalWrite(ledPinGreen, HIGH);//включаем зелёный
                            delay(2000); // зелёный 2 секунды
                            digitalWrite(ledPinGreen, LOW);//выключаем зелёный
                            }
                      }
                }
          }

    }
}// иначе ничего не делаем

}
```

Думаю, проверить её на макетной плате будет сложнее, чем за компьютером. Поэтому до загрузки программы в модуль Arduino попробуем проверить всё в программе VirtualBreadboard.

Генератора, способного формировать нужный сигнал, который имитировал бы работу фотоприёмника, я не надеюсь найти, но хочу использовать в качестве такого генератора обычную кнопку. А для отладочных операций, кроме светодиодов семафора, я добавлю (и в программу, и на макетную плату) ещё один светодиод. В результате макет выглядит следующим образом.

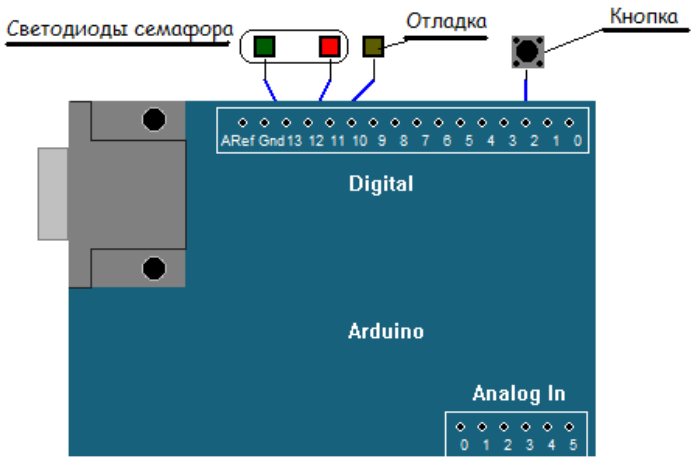


Рис. 10.1. Подготовка к проверке программы в VBB

В программу добавлена строка в определение переменных и констант:

```
const int ledPinDebug = 11; // отладочный LED
```

И в начале основного цикла:

```
digitalWrite(ledPinDebug, LOW);
```

Для кнопки (в её свойствах после выделения на макете), щёлкнув по клавише рядом «Contact Type», я выбираю режим, при котором нажатая кнопка даёт низкий логический уровень.

Properties	
Contact Type	2. On=GND, Off=VDD
Button	1. circle
ColorON	blue
ColorOFF	black
FabID	

Рис. 10.2. Окно свойств виртуальной кнопки

А в качестве первого шага проверки я добавляю в программу, проверяя вход расшифровки ИК-кода, несколько строк:

```
if (photoState == LOW) {
delayMicroseconds(4500);
      digitalWrite(ledPinDebug, HIGH); // first test
```

Запустив программу, щёлкаю несколько раз кнопкой, чтобы убедиться, что при низком логическом уровне на цифровом входе я попадаю в блок программы расшифровки кода.

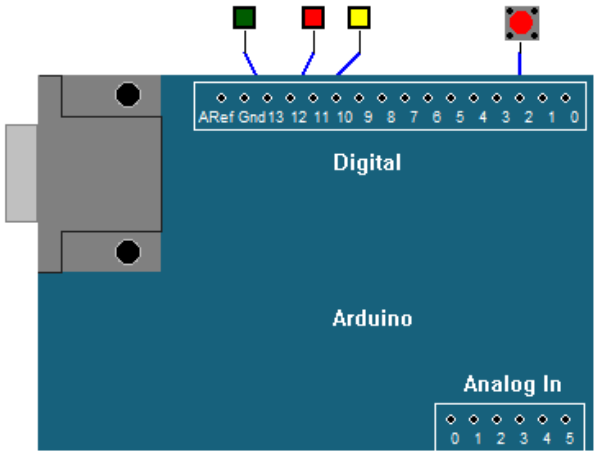


Рис. 10.3. Первая проверка программы

Удалив строку проверки (закомментировав её), я переношу её в следующую проверку состояния цифрового входа: я не успею щёлкнуть кнопкой, а, значит, отладочный светодиод не должен загораться. Запустив программу, щёлкая кнопкой несколько раз, я убеждаюсь, что это так. К сожалению, такие отладочные средства, как точка останова и пошаговое прохождение программы, у меня не работают. Но можно проявить выдумку?!

Я добавляю ещё один светодиод на следующий выход. И добавляю паузы по 3 секунды после каждого тестового включения светодиода отладки (или светодиодов). После этой хитрости я успеваю щёлкать кнопкой после каждого изменения состояния отладочных светодиодов и могу проверить расшифровку кода (конечно, только логику работы программы) до самого конца.

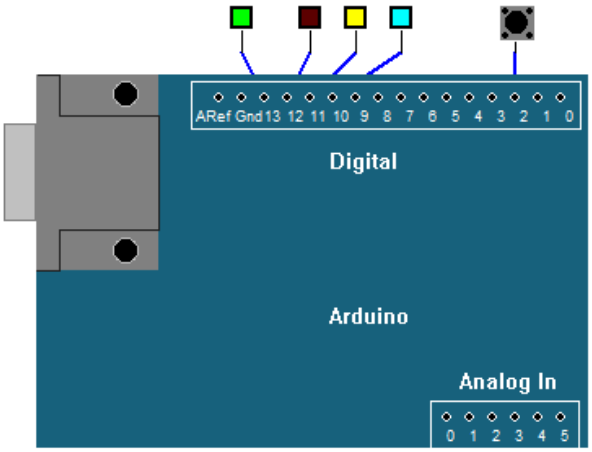


Рис. 10.4. Модификация эксперимента

Отладочная программа выглядит так:

```
class Class0 extends com.muvium.compatibility.arduino.Arduino{//Automatically Added
VBB Framework Code - do not remove

      const int photoPin = 2;
      const int ledPinGreen = 13;
      const int ledPinRed = 12;
      const int ledPinDebug1 = 11; // debuging LED 1
      const int ledPinDebug2 = 10; // debuging LED 2
```

```
int photoState = HIGH;

void setup() {
	pinMode(ledPinGreen, OUTPUT);
	pinMode(ledPinRed, OUTPUT);
	pinMode(ledPinDebug1, OUTPUT);
	pinMode(ledPinDebug2, OUTPUT);
	pinMode(photoPin, INPUT);
}

void loop(){

	digitalWrite(ledPinRed, HIGH);
		digitalWrite(ledPinDebug1, LOW);
		digitalWrite(ledPinDebug2, LOW);
	photoState = digitalRead(photoPin);

	if (photoState == LOW) {
		delayMicroseconds(4500);
				digitalWrite(ledPinDebug1, HIGH); // test 1t
				delay(3000);
		photoState = digitalRead(photoPin);
		if (photoState == HIGH){
			delay(1);
				digitalWrite(ledPinDebug1, LOW); // test 2d
				delay(3000);
			photoState = digitalRead(photoPin);
			if (photoState == LOW){
				delay(1);
					digitalWrite(ledPinDebug2, HIGH); // test 3d
					delay(3000);
				photoState = digitalRead(photoPin);
				if (photoState == HIGH){
					delay(1);
					digitalWrite(ledPinDebug2, LOW); // test 4h
					delay(3000);
					photoState = digitalRead(photoPin);
					if (photoState == LOW){
						delayMicroseconds(3500);
						digitalWrite(ledPinDebug1, HIGH);// 5h
						digitalWrite(ledPinDebug2, HIGH);
						delay(3000);
						photoState = digitalRead(photoPin);
						if (photoState == HIGH){
							digitalWrite(ledPinRed, LOW);
							digitalWrite(ledPinGreen, HIGH);
							delay(5000);
							digitalWrite(ledPinGreen, LOW);
						}
					}
				}
			}

		}
	}
}
```

Места проверки я выделил, чтобы было понятно, как должны вести себя отладочные светодиоды.

Что ж, всё, что можно было проверить за компьютером, я проверил, и настал момент, когда пора проверить совместную работу устройств на «живых» макетных платах. Тем более, что были сомнения относительно времён посылок кода.

Загрузив программу в модуль Arduino, я включаю оба устройства и… и… и ничего интересного не вижу. Что-то, где-то не так.

Рис. 10.5. Загрузка программы в модуль Arduino

Как проверить, где ошибка или неточность?

Попробуем сделать что-то похожее на отладку в программе VirtualBreadBoard. Но, чтобы не делать лишних паек, не добавлять отладочные светодиоды, используем тот светодиод, что уже стоит на плате модуля Arduino. Он на 13 цифровом выводе, который я использую для включения зеленого света. Поскольку программа не работает должным образом, это светодиод не загорается. Для начала я добавлю строку отладки в исходный текст при начале расшифровки кода:

```
if (photoState == LOW) {
    // начинаем читать код и переключать светодиоды, если код «свой»
    delayMicroseconds(4500);      // ждём 4.5 мС
       digitalWrite(ledPinGreen, HIGH);
    photoState = digitalRead(photoPin);
```

Загружаем программу и, поднося излучатель ИК-кода к фотоприёмнику, убеждаемся, что светодиод «зелёный свет» загорается. Скопируем эту строку, закомментируем её в этом месте и вставим в следующую проверку.

Так, перемещаясь по программе вниз, доходим до того места, где появилась ошибка.

```
    if (photoState == LOW){
          delayMicroseconds(3500);      // ждём 3.5 мС
Место ошибки!!!! Нужно добавить: photoState = digitalRead(photoPin);
          if (photoState == HIGH){
          digitalWrite(ledPinRed, LOW); // выключаем красный
          digitalWrite(ledPinGreen, HIGH);//включаем зелёный
          delay(2000); // зелёный 2 секунды
          digitalWrite(ledPinGreen, LOW);//выключаем зелёный
          }
    }
```

Перед последней проверкой не было команды проверки состояния входа! И, следовательно, последняя проверка if (photoState == HIGH) не подтвердила код.

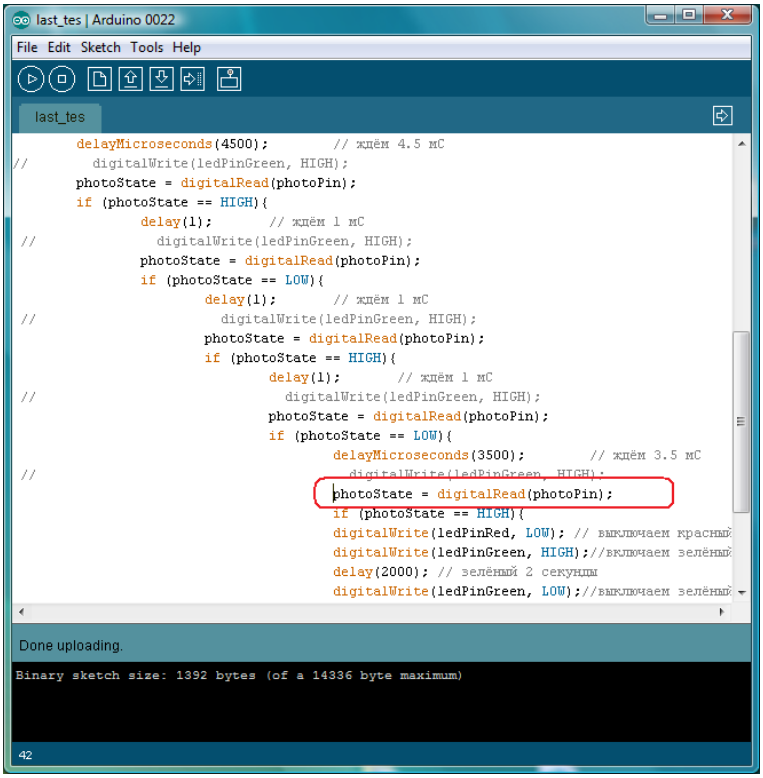


Рис. 10.6. Отладка (выделена пропущенная команда) в программе Arduino

Добавляем утерянную строку и проверяем, загрузив правильный текст программы в модуль, окончательную работу программы. Зелёный светодиод загорается, когда к фотоприёмнику подносится излучатель ИК-кода, и гаснет через 5 секунд, что я выбрал в качестве паузы между переключениями семафора. Вот полный и правильный текст программы.

```
// установка выводов

const int photoPin = 2;  // номер вывода, к которому подключён фотоприёмник
const int ledPinGreen =  13;       // номер вывода зелёного светодиода
const int ledPinRed =  12;        // номер вывода красного светодиода

int photoState = HIGH;     // переменная для чтения состояния фотоприёмника


void setup() {
  // инициализируем выходы для светодиодов
  pinMode(ledPinGreen, OUTPUT);
  pinMode(ledPinRed, OUTPUT);
  // инициализируем вход для фотоприёмника
  pinMode(photoPin, INPUT);
}

void loop(){
    // включаем красный свет
    digitalWrite(ledPinRed, HIGH);
  // читаем состояние входа
  photoState = digitalRead(photoPin);

  // проверяем есть ли пачка импульсов
  // если есть, то photoState становится LOW
if (photoState == LOW) {
    // начинаем читать код и переключать светодиоды, если код «свой»
    delayMicroseconds(4500);       // ждём 4.5 мС
    photoState = digitalRead(photoPin);
```

```
    if (photoState == HIGH){
          delay(1);   // ждём 1 мС
          photoState = digitalRead(photoPin);
          if (photoState == LOW){
                delay(1);   // ждём 1 мС
                photoState = digitalRead(photoPin);
                if (photoState == HIGH){
                      delay(1);   // ждём 1 мС
                      photoState = digitalRead(photoPin);
                      if (photoState == LOW){
                            delayMicroseconds(3500);      // ждём 3.5 мС
                            photoState = digitalRead(photoPin);
                            if (photoState == HIGH){
                                  // выключаем красный
                            digitalWrite(ledPinRed, LOW);
                                  //включаем зелёный
                            digitalWrite(ledPinGreen, HIGH);
                            delay(2000); // зелёный 2 секунды
                                  //выключаем зелёный
                            digitalWrite(ledPinGreen, LOW);
                            }
                      }
                }
          }

    }
}// иначе ничего не делаем
}
```

И глава, и книга должны были быть о роботах. А мы всё о программах, о паровозиках да семафорах, когда же про роботов?

Прежде, чем робот сделает первое движение, вы должны сделать первые шаги в освоении этой техники. Вот вы и сделали эти шаги, если прочитали все предыдущие главы, повторили всё, о чём в них написано. Если ваши устройства для паровозика и семафора заработали, то вы можете модифицировать описанные ранее программы так, чтобы паровозик «видел» семафор — добавьте к модулю фотоприёмник TSOP — а семафор, отправлял сигнал паровозику, что горит красный свет. Паровозик, принимая сигнал, отправлял бы сигнал «я свой», а семафор, получив его, включал зелёный свет. Вы можете добавить реле (включив его через транзистор к устройству в паровозике), которое будет отключать двигатель паровозика, получив сигнал от семафора, что горит красный свет, а «увидев» зелёный, включало бы двигатель. И вы получили первый робот-паровозик.

Впрочем, оставим «Паровозик из Ромашкова». Что нам мешает собранные и проверенные модули применить иначе. Например, если у вас есть тележка с электрическим моторчиком, умеющая поворачивать, то... Установим на эту тележку модуль семафора. Он будет принимать тот ИК-код, который мы проверили. А модуль излучения ИК-кода мы модифицируем следующим образом: вместо одного светодиода (АЛ307А) включим две гирлянды из светодиодов. Если использовать напряжение 12 В, то можно включить в гирлянду 8-9 светодиодов. И ничто не мешает сделать не две, а скажем четыре гирлянды, подключив все транзисторы к одному выходу микроконтроллера.

Зачем нам эта новогодняя ёлка? Выложим этими гирляндами периметр в виде прямоугольника. Внутрь поместим тележку-робота, которую запрограммируем так, чтобы, получив сигнал от излучателя, она останавливалась, отъезжала назад, поворачивала и вновь пыталась проехать к «ограде». Если мы оставим в ограде ворота, то получим робота, который должен найти выход из «загона».

Экспериментируя с «дальнобойностью» излучателей, скоростью движения робота-тележки, вы наработаете некоторый материал, который поможет создавать более сложные устройства. Сегодня и в Интернете, и в магазинах можно поискать и найти многое, что позволит собирать роботы. Есть много таких полезных датчиков, как ультразвуковой датчик расстояния, исполнительных механизмов, таких как сервомоторы и т.д.

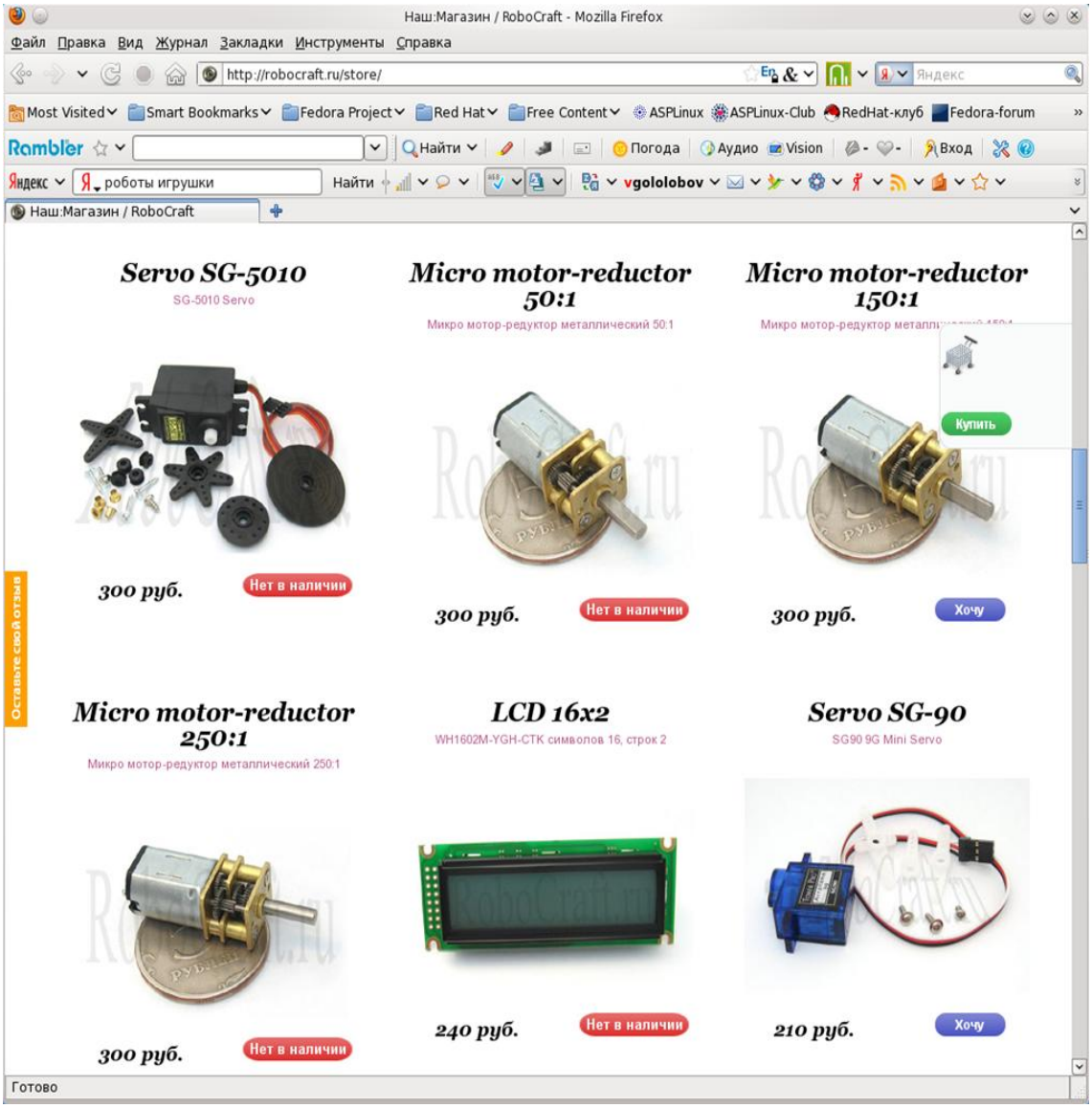


Рис. 10.7. Компоненты для создания роботов в Интернет-магазине

Есть достаточно много заготовок, которые можно использовать в роботостроении. Вот, например, что можно найти в магазине:

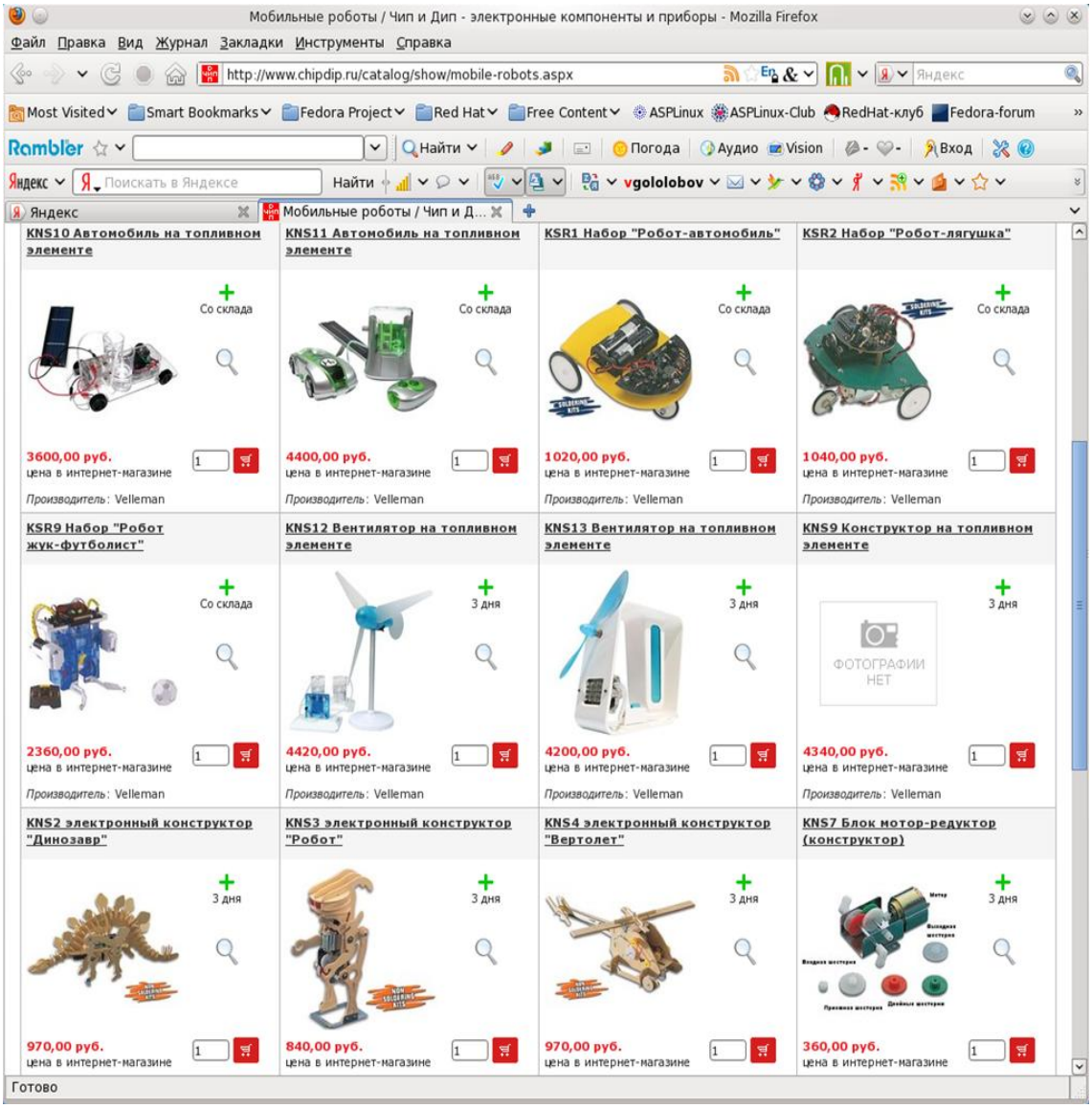


Рис. 10.8. Готовые модули и блоки в радиомагазине

Но самое главное, чем вам следует обзавестись, это желанием научиться всему, терпением и умением придумывать что-то своё и реализовывать эти задумки. Возможно, обретя эти навыки, вы выберете себе профессию, увлекательную и современную, создателя роботов. Успехов вам!

Приложение А. О языке программирования Arduino

Arduino блокнот программиста

Brian W. Evans

Arduino Programming Notebook
Написано и скомпилировано Brian W. Evans

С информацией или навеянным с:
http://www.arduino.cc
http://www.wiring.org.co
http://www.arduino.cc/en/Booklet/HomePage
http://cslibrary.stanford.edu/101/

Включает материалы, написанные:
Massimo Banzi
Hernando Barragán
David Cuartielles
Tom Igoe
Daniel Jolliffe
Todd Kurt
David Mellis
and others

Published:
First Edition August 2007

Оглавление

Оглавление....121
предисловие....123
структура....125
setup()....125
loop()....126
функции....126
{} фигурные скобки....127
; точка с запятой....127
* ... */ блок комментария....128
однострочный комментарий....128
переменные....129
объявление переменных....130
границы переменных....131
byte....132
int....132
long....132
float....132
массивы....133
арифметика....134
смешанное присваивание....134
операторы сравнения....135
логические операторы....135
константы....136
true/false....136
high/low....136
input/output....136
управление программой....137
if....137
if...else....138
for....139
while....140

do...while 140
цифровой ввод/вывод 141
pinMode (pin, mode) 141
digitalRead (pin) 142
digitalWrite (pin, value) 142
analogRead (pin) 143
analogWtite (pin, value) 144
время и математика 145
delay (ms) 145
millis() 145
min (x, y) 145
max (x, y) 145
случайные числа 146
randomSeed (seed) 146
random (max) 146
random (min, max) 146
последовательный обмен 147
Serial.begin (rate) 147
Serial.println (data) 147
приложение 148
цифровой выход 149
цифровой ввод 150
сильноточный выход 151
pwm выход 152
вход с потенциометра 153
вход от переменного резистора 154
серво вывод 155

предисловие

Этот блокнот следует рассматривать, как удобное, лёгкое в использовании руководство по структуре команд и синтаксису языка программирования контроллера Arduino. Для сохранения простоты, были сделаны некоторые исключения, что улучшает руководство при использовании начинающими в качестве дополнительного источника информации - наряду с другими web-сайтами, книгами, семинарами и классами. Подобное решение, призвано акцентировать внимание на использовании Arduino для автономных задач и, например, исключает более сложное использование массивов или использование последовательного соединения.

Начиная с описания структуры программы для Arduino на языке C, этот блокнот содержит описание синтаксиса наиболее общих элементов языка и иллюстрирует их использование в примерах и фрагментах кода. Блокнот содержит примеры функций ядра библиотеки Arduino, а в приложении приводятся примеры схем и начальных программ. Благодарности O'Sullivan и Igoe с их *Physical Computing*.

За введением в Arduino и в интерактивную разработку обратитесь к *Getting started with Arduino*, Banzi, aka *Adruino Booklet*. Для особо отважных, интересующихся программированием на Си — Керниган и Ричи *Язык программирования Си*, второе издание, равно как и Принз и Кроуфорд, *Си в двух словах*, дающие понимание оригинального синтаксиса программирования.

Помимо прочего, этот блокнот не появился бы без большого сообщества созидателей и массы оригинального материала, который можно найти на официальном web-сайте и формуе Arduino: http://www.arduino.cc.

Перевод на русский:

Гололобов Владимир Николаевич vgololobov@yandex.ru http://vgololobov.narod.ru

Редакция и правка:

команда сайта http://robocraft.ru

структура

Базовая структура программы для Arduino довольно проста и состоит, по меньшей мере, из двух частей. В этих двух обязательных частях, или функциях, заключён выполняемый код.

```
void setup()
{
операторы;
}

void loop()
{
операторы;
}
```

Где setup() — это подготовка, а loop() — выполнение. Обе функции требуются для работы программы.

Перед функцией setup - в самом начале программы, обычно, идёт, объявление всех переменных. setup - это первая функция, выполняемая программой, и выполняемая только один раз, поэтому она используется для установки режима работы портов (pinMode()) или инициализации последовательного соединения

Следующая функция loop содержит код, который выполняется постоянно — читаются входы, переключаются выходы и т.д. Эта функция — ядро всех программ Arduino и выполняет основную работу.

setup()

Функция setup() вызывается один раз, когда программа стартует. Используйте её для установки режима выводов или инициализации последовательного соединения. Она должна быть включена в программу, даже если в ней нет никакого содержания.

```
void setup()
{
        pinMode (pin, OUTPUT);      // устанавливает ‘pin’ как выход
}
```

loop()

После вызова функции setup() – управление переходит к функции loop() , которая делает в точности то, что означает её имя — непрерывно выполняется, позволяя программе что-то изменять, отвечать и управлять платой Arduino.

```
void loop()
{
        digitalWrite(pin, HIGH);        // включает ‘pin’
        delay(1000);                    // секундная пауза
        digitalWrite(pin, LOW);         // выключает ‘pin’
        delay(1000);                    // секундная пауза
}
```

функции

Функция — это блок кода, имеющего имя, которое указывает на исполняемый код, который выполняется при вызове функции. Функции void setup() и void loop() уже обсуждались, а другие встроенные функции будут рассмотрены позже.

Могут быть написаны различные пользовательские функции, для выполнения повторяющихся задач и уменьшения беспорядка в программе. При создании функции, первым делом, указывается тип функции. Это тип значения, возвращаемого функцией, такой как 'int' для целого (integer) типа функции. Если функция не возвращает значения, её тип должен быть void. За типом функции следует её имя, а в скобках параметры, передаваемые в функцию.

```
type functionName (parameters)
{
операторы;
}
```

Следующая функция целого типа delayVal() используется для задания значения паузы в программе чтением значения с потенциометра. Вначале объявляется локальная переменная v, затем v устанавливается в значение потенциометра, определяемое числом между 0 — 1023, затем это значение делится на 4, чтобы результирующее значение было между 0 и 255, а затем это значение возвращается в основную программу.

```
 int delayVal()
{
int v;                      // создаём временную переменную 'v'
v = analogRead(pot); // считываем значение с потенциометра
v /= 4;                     // конвертируем 0 – 1023 в 0 – 255
return v;                   // возвращаем конечное значение
}
```

{} фигурные скобки

Фигурные скобки (также упоминаются как просто «скобки») определяют начало и конец блока функции или блока выражений, таких как функция void loop() или выражений (statements) типа for и if.

```
type function()
{
операторы;
}
```

За открывающейся фигурной скобкой { всегда должна следовать закрывающаяся фигурная скобка }. Об этом часто упоминают, как о том, что скобки должны быть «сбалансированы». Несбалансированные скобки могут приводить к критическим, неясным ошибкам компиляции, вдобавок иногда и трудно выявляемым в больших программах.

Среда разработки Arduino, включает возможность удобной проверки баланса фигурных скобок. Достаточно выделить скобку, или даже щёлкнуть по точке вставки сразу за скобкой, чтобы её пара была подсвечена.

; точка с запятой

Точка с запятой должна использоваться в конце выражения и разделять элементы программы. Также точка с запятой используется для разделения элементов цикла for.

```
int x = 13;  // объявляет переменную ‘x’ как целое 13
```

Примечание: Если забыть завершить стоку точкой с запятой, то это приведёт к возникновению ошибки компиляции. Текст ошибки может быть очевиден и указывать на пропущенную точку с запятой, но может быть и не таким очевидным. Если появляется маловразумительная или нелогичная ошибка компилятора, первое, что следует проверить — не пропущена ли точка с запятой вблизи строки, где компилятор выразил своё недовольство.

/* ... */ блок комментария

Блок комментария или однострочный комментарий — это область текста, которая игнорируется программой и используется для добавления текста с описанием кода или примечаний. Комментарий помогают другим понять эту часть программы. Он начинается с /* и заканчивается */ и может содержать множество строк.

```
/* это «огороженный» блок комментария,
и не забудьте «закрыть» комментарий -
он должен быть сбалансирован!
*/
```

Поскольку комментарии игнорируются программой, а, следовательно, не занимают места в памяти, они могут быть достаточно ёмкими, но кроме того, они могут использоваться для «пометки» блоков кода с отладочной целью.

Примечание: Хотя допускается вставка однострочного комментария в блоке комментария, второй блок комментария не допускается.

// однострочный комментарий

Однострочный комментарий начинается с // и заканчивается (внутренним) кодом перехода на другую строку. Как и блок комментария, он игнорируется программой и не занимает места в памяти.

```
// вот так выглядит однострочный комментарий
```

Однострочный комментарий часто используется после действенного выражения, чтобы дать больше информации о том, что выражение выполняет или в качестве напоминания на будущее.

переменные

Переменные — это способ именовать и хранить числовые значения для последующего использования программой. Само название - переменные, говорит о том, что переменные - это числа, которые могут последовательно меняться, в отличие от констант, чьё значение никогда не меняется. Переменные нужно декларировать (объявлять), и, что очень важно - им можно присваивать значения, которые нужно сохранить. Следующий код объявляет переменную inputVariable, а затем присваивает ей значение, полученное от 2-го аналогового порта:

```
int inputVariable = 0;                  // объявляется переменная и
                                        // ей присваивается значение 0
inputVariable = analogRead(2); // переменная получает значение
                                        // аналогового вывода 2
```

'inputVariable' — это наша переменная. Первая строка декларирует, что она будет содержать int, короткое целое. Вторая строка присваивает ей значение аналогового вывода 2. Это делает значение на выводе 2 доступным в любом месте программы.

Когда переменной присвоено значение, или пере-присвоено, вы можете проверить это значение, если оно встречается в некотором условии, или использовать его непосредственно. Рассмотрим пример, иллюстрирующий три операции с переменными. Следующий код проверяет, не меньше ли 100 значение переменной, а если так, переменной inputVariable присваивается значение 100, а затем задаётся пауза, определяемая переменной inputVariable, которая теперь, как минимум, равна 100:

```
if (inputVariable < 100) // проверяем, не меньше ли 100 переменная
{
inputVariable = 100;     // если так, присваиваем ей значение 100
}
delay(inputVariable);    // используем переменную, как паузу
```

Примечание: Переменные должны иметь наглядные имена, чтобы код был удобочитаемый. Имена переменных как tiltSensor или pushButton помогают программисту при последующем чтении кода понять, что содержит эта переменная. Имена переменных как var или value, с другой стороны, мало делают для понимания кода, и здесь используются только в качестве примера. Переменные могут быть названы любыми именами, которые не являются ключевыми словами языка программирования Arduino.

объявление переменных

Все переменные должны быть задекларированы до того, как они могут использоваться. Объявление переменной означает определение типа её значения: int, long, float и т.д., задание уникального имени переменной, и дополнительно ей можно присвоить начальное значение. Всё это следует делать только один раз в программе, но значение может меняться в любое время при использовании арифметических или других разных операций.

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

```
int  inputVariable = 0;
```

Переменная может быть объявлена в разных местах программы, и то, где это сделано, определяет, какие части программы могут использовать переменную.

границы переменных

Переменные могут быть объявлены в начале программы перед void setup(), локально внутри функций, и иногда в блоке выражений таком, как цикл for. То, где объявлена переменная, определяет её границы (область видимости), или возможность некоторых частей программы её использовать.

Глобальные переменные таковы, что их могут видеть и использовать любые функции и выражения программы. Такие переменные декларируются в начале программы перед функцией setup().

Локальные переменные определяются внутри функций или таких частей, как цикл for. Они видимы и могут использоваться только внутри функции, в которой объявлены. Таким образом, могут существовать несколько переменных с одинаковыми именами в разных частях одной программы, которые содержат разные значения. Уверенность, что только одна функция имеет доступ к её переменной, упрощает программу и уменьшает потенциальную опасность возникновения ошибок.

Следующий пример показывает, как декларировать несколько разных типов переменных, и демонстрирует видимость каждой переменной:

```
int value;                      // 'value' видима
                                // для любой функции
void setup()
{
        // нет нужды в предустановке
}
void loop(0
{
        for (int i =0; i < 20;)     // ‘i’ видима только
        {                           // внутри цикла for
        i++;
        }
float f;                        // ‘f’ видима только
}                               // внутри loop
```

byte

Байт хранит 8-битовое числовое значение без десятичной точки. Он имеет диапазон от 0 до 255.

```
byte someVariable = 180;    // объявление ‘someVariable’
                            // как имеющей тип byte
```

int

Целое — это первый тип данных для хранения чисел без десятичной точки, и хранит 16-битовое значение в диапазоне от 32767 до -32768.

```
int someVariable = 1500;    // объявление ‘someVariable’
                            // как переменной целого типа
```

Примечание: Целые переменные будут переполняться, если форсировать их переход через максимум или минимум при присваивании или сравнении. Например, если x = 32767 и следующее выражение добавляет 1 к x, x = x +1 или x++, в этом случае x переполняется и будет равен -32678.

long

Тип данных увеличенного размера для больших целых, без десятичной точки, сохраняемый в 32-битовом значении с диапазоном от 2147383647 до -2147383648.

```
long someVariable = 90000;      // декларирует 'someVariable'
                                // как переменную типа long
```

float

Тип данных для чисел с плавающей точкой или чисел, имеющих десятичную точку. Числа с плавающей точкой имеют большее разрешение, чем целые и сохраняются как 32-битовые значения в диапазоне от 3.4028235E+38 до -3.4028235E+38.

```
float someVariable = 3.14;      // объявление 'someVariable'
                                // как переменной типа floating point
```

Примечание: Числа с плавающей точкой не точные, и могут выдавать странные результаты при сравнении. Вычисления с плавающей точкой медленнее, чем вычисления целых при выполнении расчётов, так что, без нужды, их следует избегать.

массивы

```
int myArray[ ] = {value0, value1, value2…}
```

Массив — это набор значений, к которым есть доступ через значение индекса. Любое значение в массиве может быть вызвано через вызов имени массива и индекса значения. Индексы в массиве начинаются с нуля с первым значением, имеющим индекс 0. Массив нуждается в объявлении, а дополнительно может заполняться значениями до того, как будет использоваться.

Схожим образом можно объявлять массив, указав его тип и размер, а позже присваивать значения по позиции индекса:

```
int myArray [5];   // объявляет массив целых длиной в 6 позиций
myArray[3] = 10; // присваивает по 4у индексу значение 10
```

Чтобы извлечь значение из массива, присвоим переменной значение по индексу массива:

```
x = myArray[3];   // x теперь равно 10
```

Массивы часто используются в цикле for, где увеличивающийся счётчик применяется для индексации позиции каждого значения. Следующий пример использует массив для мерцания светодиода. Используемый цикл for со счётчиком, начинающимся с 0, записывает значение из позиции с индексом 0 массива flicker[], в данном случае 180, на PWM-вывод (широтно-импульсная модуляция) 10; затем пауза в 200 ms, а затем переход к следующей позиции индекса.

```
int ledPin = 10;                          // LED на выводе 10
byte flicker[ ] = {180, 30, 255, 200, 10, 90, 150, 60};
                                          // выше массив из 8
void setup()                              // разных значений
      {
      pinMode(ledPin, OUTPUT);            // задаём OUTPUT вывод
      }
void loop()
{
      for (int i = 0; i < 7; i++)         // цикл равен числу
      {                                   // значений в массиве
      analogWrite(ledPin, flicker[i];// пишем значение по индексу
      delay(200);                         // пауза 200 мС
      }
}
```

арифметика

Арифметические операции включают сложение, вычитание, умножение и деление. Они возвращают сумму, разность, произведение или частное (соответственно) двух операндов.

```
y = y + 3;
x = x – 7;
i = j * 6;
r = r / 5;
```

Операция управляется используемым типом данных операндов, так что, например, 9/4 даёт 2 вместо 2.25, поскольку 9 и 4 имеют тип int и не могут использовать десятичную точку. Это также означает, что операция может вызвать переполнение, если результат больше, чем может храниться в данном типе.

Если используются операнды разного типа, то для расчётов используется больший тип. Например, если одно из чисел (операндов) типа float, а второе целое, то для вычислений используется тип с плавающей точкой.

Выбирайте типы переменных достаточные для хранения результатов ваших вычислений. Прикиньте, в какой точке ваша переменная переполнится, а также, что случится в другом направлении, то есть, (0-1) или (0- -32768). Для вычислений, требующих дробей, используйте переменные типа float, но остерегайтесь их недостатков: большой размер и маленькая скорость вычислений.

Примечание: Используйте оператор приведения типа (нзвание типа) для округления, то есть, (int)myFloat - для преобразования переменной одного типа в другой «на лету». Например, i = (int) 3.6 - поместит в i значение 3.

смешанное присваивание

Смешанное присваивание сочетает арифметические операции с операциями присваивания. Чаще всего встречается в цикле for, который описан ниже. Наиболее общее смешанное присваивание включает:

```
x ++            // то же, что x = x + 1, или увеличение x на +1
x --            // то же, что x = x - 1, или уменьшение x на -1
x += y          // то же, что x = x + y, или увеличение x на +y
x -= y          // то же, что x = x - y, или уменьшение x на -y
x *= y          // то же, что x = x * y, или умножение x на y
x /= y          // то же, что x = x / y, или деление x на y
```

Примечание: Например, x *= 3 утроит старое значение x и присвоит полученный результат x.

операторы сравнения

Сравнения одной переменной или константы с другой используются в выражении для if, чтобы проверить истинность заданного условия. В примерах на следующих страницах ?? используется для обозначения любого из следующих условий:

```
x == y          // x равно y
x != y          // x не равно y
x < y           // x меньше, чем y
x > y           // x больше, чем y
x <= y          // x меньше, чем или равно y
x >= y          // x больше, чем или равно y
```

логические операторы

Логические операторы, чаще всего, это способ сравнить два выражения и вернуть ИСТИНА или ЛОЖЬ, в зависимости от оператора. Есть три логических оператора: AND, OR и NOT, часто используемые в конструкциях if:

```
Logical AND:
if (x > 0 && x < 5)       // true, только если оба
                          // выражения true
Logical OR:
if (x > 0 || y > 0)       // true, если любое из
                          // выражений true
Logical NOT:
if  (!x > 0)              // true, если только
                          // выражение false
```

константы

Язык Arduino имеет несколько предопределённых величин, называемых константами. Они используются, чтобы сделать программу удобной для чтения. Константы собраны в группы.

true/false

Это Булевы константы, определяющие логические уровни. FALSE легко определяется как 0 (ноль), а TRUE, как 1, но может быть и чем-то другим, отличным от нуля. Так что в Булевом смысле -1, 2 и 200 — это всё тоже определяется как TRUE.

```
if (b == TRUE)
{
        что-нибудь сделаем;
}
```

high/low

Эти константы определяют уровень выводов как HIGH или LOW и используются при чтении или записи на логические выводы. HIGH определяется как логический уровень 1, ON или 5 вольт(3-5), тогда как LOW — 0, OFF или 0 вольт(0-2).

```
digitalWrite(13, HIGH);
```

input/output

Константы используются с функцией pinMode() для задания режима работы цифровых выводов: либо как INPUT (вход), либо как OUTPUT (выход).

```
pinMode (13, OUTPUT);
```

управление программой

if

Конструкция if проверяет, будет ли выполнено некое условие, такое, как, например, будет ли аналоговое значение больше заданного числа, и выполняет какое-то выражение в скобках, если это условие true (истинно). Если нет, то выражение в скобках будет пропущено. Формат для if следующий:

```
if (someVariable ?? value)
{
        что-нибудь сделаем;
}
```

Пример выше сравнивает someVariable со значением (value), которое может быть и переменной, и константой. Если выражение или условие в скобках истинно, выполняется выражение в фигурных скобках. Если нет, выражение в фигурных скобках пропускается, и программа выполняется с оператора, следующего за скобками.

Примечание: Остерегайтесь случайного использования «=», как в if (x = 10), что технически правильно, определяя x равным 10, но результат этого всегда true. Вместо этого используйте «==», как в if (x == 10), что осуществляет проверку значения x — равно ли оно 10 или нет. Запомните «=» - равно, а «==» - равно ли?

if...else

Конструкция if...else позволяет сделать выбор «либо, либо». Например, если вы хотите проверить цифровой вход и выполнить что-то, если он HIGH, или выполнить что-то другое, если он был LOW, вы должны записать следующее:

```
if (inputPin == HIGH)
{
        делаемА;
}
else
{
        делаемБ;
}
```

else может также предшествовать другой проверке if так, что эти множественные, взаимоисключающие проверки могут запускаться одновременно. И возможно даже неограниченное количество подобных else переходов. Хотя следует помнить, что только один набор выражений будет выполнен в зависимости от результата проверки:

```
if (inputPin < 500)
{
        делаемА;
}
else if (inputPin >= 1000)
        {
                делаемБ;
        }
        else
        {
                делаемВ;
        }
```

Примечание: Конструкция if просто проверяет, будет ли выражение в круглых скобках истинно или ложно. Это выражение может быть любым правильным, относительно языка Си, выражением, как в первом примере if (inputPin == HIGH). В этом примере if проверяет только то, что означенный вход в состоянии высокого логического уровня или действительно ли напряжение на нём 5 вольт.

for

Конструкция for используется для повторения блока выражений, заключённых в фигурные скобки заданное число раз. Наращиваемый счётчик часто используется для увеличения и прекращения цикла. Есть три части, разделённые точкой с запятой, в заголовке цикла for:

```
for (инициализация; условие; выражение)
{
        что-нибудь делаем;
}
```

«Инициализация» локальной переменной, или счётчика, имеет место в самом начале и происходит только один раз. При каждом проходе цикла проверяется «условие». Если условие остаётся истинным, то следующее выражение и блок выполняются, а условие проверяется вновь. Когда условие становится ложным, цикл завершается.

Следующий пример начинается с целого i равного 0, проверяет, остаётся ли i ещё меньше 20, и, если так, увеличивает i на 1 и выполняет блок в фигурных скобках:

```
for (int i = 0; i < 20; i++)      // декларируем i, проверяем, меньше ли оно
                                  // чем 20, увеличиваем i на 1
{
digitalWrite (13, HIGH);          // устанавливаем вывод 13 в ON
delay (250);                      // пауза в ¼ секунды
digitalWrite (13, LOW);           // сбрасываем вывод 13 в OFF
delay (250);                      // пауза в ¼ секунды
}
```

Примечание: В Си цикл for более гибок, чем это можно обнаружить в других языках программирования, включая Basic. Любые или все три элемента заголовка могут быть опущены, хотя точка с запятой требуется. Также выражения для инициализации, условия и выражения могут быть любыми правильными выражениями Си с несвязанными переменными. Такие необычные типы выражений могут помочь в решении некоторых редких программных проблем.

while

Цикл while продолжается, и может продолжаться бесконечно, пока выражение в скобках не станет false (ложно). Что-то должно менять проверяемую переменную, иначе из цикла никогда не выйти. И это должно быть в вашем коде, как, скажем, увеличение переменной, или внешнее условие, как, например, проверяемый сенсор.

```
while (someVariable ?? value)
{
что-нибудь делаем;
}
```

Следующий пример проверяет, будет ли someVariable меньше 200, и если да, то выполняются выражения в фигурных скобках, и цикл продолжается, пока someVariable остаётся меньше 200.

```
while (someVariable < 200)  // проверяем, меньше ли 200 переменная
{
что-нибудь делаем;          // выполняем операции в скобках
someVariable++;             // увеличиваем переменную на 1
}
```

do...while

Цикл do управляемый «снизу» цикл, работающий на манер цикла while, с тем отличием, что условие проверки расположено в конце цикла, таким образом, цикл выполнится хотя бы один раз.

```
do
{
      что-нибудь делаем;
} while (someVariable ?? value);
```

Следующий пример присваивает readSensor переменной x, делает паузу на 50 миллисекунд, затем цикл выполняется, пока x меньше, чем 100.

```
do
{
      x = readSensors();      // присваиваем значение
                              // readSensors() переменной x
      delay (50);             // пауза 50 миллисекунд
} while (x < 100);            // продолжение цикла, если x меньше 100
```

цифровой ввод/вывод

pinMode (pin, mode)

Используется в void setup () для конфигурации заданного вывода, чтобы он работал на вход (INPUT) или на выход (OUTPUT).

```
pinMode (pin, OUTPUT);      // устанавливаем ‘pin’ на выход
```

Цифровые выводы в Arduino предустановлены на вход, так что их нет нужды явно объявлять как INPUT с помощью pinMode (). Выводы, сконфигурированные как INPUT, подразумеваются в состоянии с высоким импедансом (сопротивлением).

В микроконтроллере Atmega, есть также удобные, программно доступные подтягивающие резисторы 20 кОм. Эти встроенные подтягивающие резисторы доступны следующим образом:

```
pinMode (pin, INPUT);       // настраиваем ‘pin’ на вход
digitalWrite (pin, HIGH);   // включаем подтягивающие резисторы
```

Подтягивающие резисторы, как правило, используются при соединении входов с переключателями. Заметьте, что в примере выше нет преобразования pin на выход, это просто метод активизации встроенных подтягивающих резисторов.

Выводы, сконфигурированные как OUTPUT, находятся в низкоимпедансном состоянии и могут отдавать 40 мА в нагрузку (цепь, другое устройство). Это достаточный ток для яркого включения светодиода (не забудьте последовательный токоограничительный резистор!), но не достаточный для включения реле, соленоидов или моторов.

Короткое замыкание выводов Arduino или слишком большой ток могут повредить выходы или даже всю микросхему Atmega. Порой, не плохая идея — соединять OUTPUT вывод через последовательно включённый резистор в 470 Ом или 1 кОм.

digitalRead (pin)

Считывает значение заданного цифрового вывода (pin) и возвращает результат HIGH или LOW. Вывод должен быть задан либо как переменная, либо как константа (0-13).

```
value = digitalRead (Pin);      // задаём 'value' равным
                                // входному выводу ‘Pin’
```

digitalWrite (pin, value)

Выводит либо логический уровень HIGH, либо LOW (включает или выключает) на заданном цифровом выводе pin. Вывод может быть задан либо как переменная, либо как константа (0-13).

```
digitalWrite (pin, HIGH);       // устанавливаем ‘pin’ в высокое состояние
```

Следующий пример читает состояние кнопки, соединённой с цифровым входом, и включает LED (светодиод), подключённый к цифровому выходу, когда кнопка нажата:

```
int led = 13;       // соединяем LED с выводом 13
int pin = 7;        // соединяем кнопку с выводом 7
int value = 0;      // переменная для хранения прочитанного значения

void setup ()
{
pinMode (led, OUTPUT);      // задаём вывод 13 как выход
pinMode (pin, INPUT);       // задаём вывод 7 как вход
}

void loop()
{
value = digitalRead (pin);      // задаём ‘value’ равной
                                // входному выводу
digitalWrite (led, value);      // устанавливаем ‘led’ в
                                // в значение кнопки
}
```

analogRead (pin)

Считывает значение из заданного аналогового входа (pin) с 10-битовым разрешением. Эта функция работает только на аналоговых портах (0-5). Результирующее целое значение находится в диапазоне от 0 до 1023.

```
value = analogRead (pin);    // задаём значение 'value' равным 'pin'
```

Примечание: Аналоговые выводы не похожи на цифровые, и нет необходимости предварительно объявлять их как INPUT или OUTPUT (если только вы не планируете использовать их в качестве цифровых портов 14-18).

analogWtite (pin, value)

Записывает псевдо-аналоговое значение, используя схему с широтно-импульсной модуляцией (PWM), на выходной вывод, помеченный как PWM. На новом модуле Arduino с ATmega168 (328), эта функция работает на выводах 3, 5, 6, 9, 10 и 11. Старый модуль Arduino с ATmega8 поддерживает только выводы 9, 10 и 11. Значение может быть задано как переменная или константа в диапазоне 0-255.

```
analogWrite (pin, value);      // записываем ‘value’ в аналоговый ‘pin’
```

Значение 0 генерирует устойчивое напряжение 0 вольт на выходе заданного вывода; значение 255 генерирует 5 вольт на выходе заданного вывода. Для значений между 0 и 255 вывод быстро переходит от 0 к 5 вольтам — чем больше значение, тем чаще вывод в состоянии HIGH (5 вольт). Например, при значении 64 вывод будет в 0 три четверти времени, а в состоянии 5 вольт одну четверть; при значении 128 половину времени будет вывод будет в 0, а половину в 5 вольт; при значении 192 четверть времени вывод будет в 0 и три четверти в 5 вольт.

Поскольку эта функция схемная (встроенного модуля), вывод будет генерировать устойчивый сигнал после вызова analogWrite в фоновом режиме, пока не будет следующего вызова analogWrite (или вызова digitalRead или digitalWrite на тот же вывод).

Примечание: Аналоговые выводы, не такие как цифровые, и не требуют предварительной декларации их как INPUT или OUTPUT.

Следующий пример читает аналоговое значение с входного аналогового вывода, конвертирует значение делением на 4 и выводит PWM сигнал на PWM вывод:

```
int led = 10;       // LED с резистором на выводе 10
int pin = 0;        // потенциометр на аналоговом выводе 0
int value;          // переменная для чтения

void setup() {}     // setap не нужен

void loop()
{
value = analogRead (pin);    // задаёт ‘value’ равной ‘pin’
value /= 4;                  // конвертируем 0 – 1023 в 0 – 255
analogWrite (led, value);    // выводим PWM сигнал на LED
}
```

время и математика

delay (ms)

Приостанавливает вашу программу на заданное время (в миллисекундах), где 1000 равно 1 секунде.

```
delay (1000);        // ждём одну секунду
```

millis()

Возвращает число миллисекунд, как unsigned long, с момента старта программы в модуле Arduino.

```
value = millis();   // задаёт ‘value’ равной millis()
```

Примечание: Это число будет переполняться (сбрасываться в ноль), после, примерно, 9 часов.

min (x, y)

Вычисляется минимум двух чисел любого типа данных и возвращает меньшее число.

```
value = min (value, 100);      // устанавливает 'value' в наименьшее из
                               // ‘value’ и 100, обеспечивая, что
                               // оно никогда не превысит 100
```

max (x, y)

Вычисляется максимум двух чисел любого типа данных и возвращает большее число.

```
value = max (value, 100);      // устанавливает 'value' в наибольшее из
                               // ‘value’ и 100, обеспечивая, что
                               // оно никогда не меньше 100
```

случайные числа

randomSeed (seed)

Устанавливает значение, или начальное число, в качестве начальной точки функции random().

```
randomSeed (value);   // задаёт ‘value’ как начальное значение random
```

Поскольку Arduino не может создавать действительно случайных чисел, randomSeed позволяет вам поместить переменную, константу или другую функцию в функцию random, что помогает генерировать более случайные «random» числа. Есть множество разных начальных чисел, или функций, которые могут быть использованы в этой функции, включая millis(), или даже analogRead() для чтения электрических шумов через аналоговый вывод.

random (max)
random (min, max)

Функция random позволяет вам вернуть псевдослучайное число в диапазоне, заданном значениями min и max.

```
value = random (100, 200);  // задаёт 'value' случайным
                            // числом между 100 и 200
```

Примечание: Используйте это после использования функции randomSeed().

Следующий пример создаёт случайное число между 0 и 255 и выводит PWM сигнал на PWM вывод, равный случайному значению:

```
int randNumber;  // переменная для хранения случайного значения
int led = 10;        // LED с резистором на выводе 10
void setup() {}     // setup не нужен

void loop ()
{
randomSeed (millis());              // задаёт millis() начальным числом
randNumber = random (255);          // случайное число из 0 – 255
analogWrite (led, randNumber);      // вывод PWM сигнала
delay (500);                        // пауза в полсекунды
}
```

последовательный обмен

Serial.begin (rate)

Открывает последовательный порт и задаёт скорость для последовательной передачи данных. Типичная скорость обмена для компьютерной коммуникации — 9600, хотя поддерживаются и другие скорости.

```
void setup ()
{
      Serial.begin (9600);     // открывает последовательный порт
                               // задаёт скорость обмена 9600
}
```

Примечание: При использовании последовательного обмена, выводы 0 (RX) и 1 (TX) не могут использоваться одновременно как цифровые.

Serial.println (data)

Передаёт данные в последовательный порт, сопровождая автоматическим возвратом каретки и переходом на новую строку. Команда такая же, что и Serial.print(), но легче для последующего чтения на данных в терминале.

```
Serial.println (analogValue);        // отправляет значение
                                     // ‘analogValue’
```

Примечание: За дальнейшей информацией о различных изменениях Serial.println() и Serial.print() обратитесь на сайт Arduino.

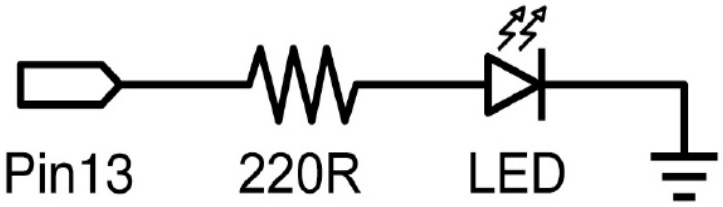
Следующий простой пример читает аналоговый вывод 0 и отсылает эти данные на компьютер каждую секунду.

```
void setup ()
{
      Serial.begin (9600);                  // задаём скорость 9600 bps
}

void loop ()
{
      Serial.println (analogRead(0);        // шлём аналоговое значение
      delay (1000);                         // пауза 1 секунда
}
```

приложение

цифровой выход



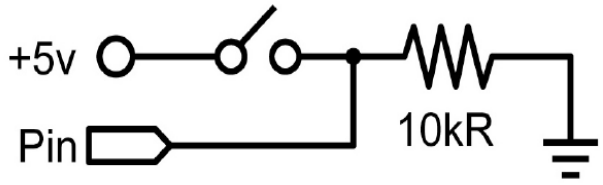
Это базовая программа «hello world», используемая для включения и выключения чего-нибудь. В этом примере светодиод подключён к выводу 13 и мигает каждую секунду. Резистор в данном случае может быть опущен, поскольку на 13-м порту Arduino уже есть встроенный резистор.

```
int ledPin = 13;                    // LED на цифровом выводе 13

void setup ()                       // запускается один раз
{
pinMode (ledPin, OUTPUT);           // устанавливаем вывод 13 на выход
}

void loop ()                        // запускается вновь и вновь
{
digitalWrite (ledPin, HIGH);        // включаем LED
delay (1000);                       // пауза 1 секунда
digitalWrite (ledPin, LOW);         // выключаем LED
delay (1000);                       // пауза 1 секунда
}
```

цифровой ввод



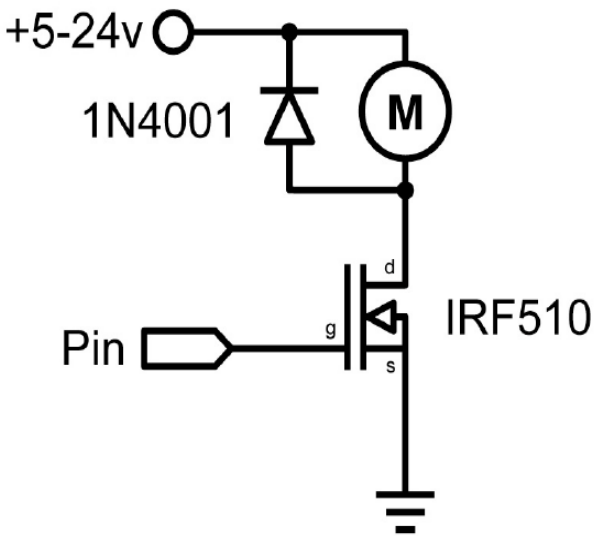
Это простейшая форма ввода с двумя возможными состояниями: включено или выключено. В примере считывается простой переключатель или кнопка, подключённая к выводу 2. Когда выключатель замкнут, входной вывод читается как HIGH и включает светодиод.

```
int ledPin = 13;                    // выходной вывод для LED
int inPin = 2;                      // входной вывод (для выключателя)

void setup ()
{
pinMode (ledPin, OUTPUT);           // объявляем LED как выход
pinMode (inPin, INPUT);             // объявляем выключатель как вход
}

void loop ()
{
if (digitalRead (inPin) == HIGH)          // проверяем, вход HIGH?
        {
        digitalWrite (ledPin, HIGH);      // включаем LED
        delay (1000);                     // пауза 1 секунда
        digitalWrite (ledPin, LOW);       // выключаем LED
        delay (1000);                     // пауза 1 секунда
        }
}
```

сильноточный выход



Иногда возникает необходимость в управлении более, чем 40 мА от Arduino. В этом случае может использоваться транзистор MOSFET для коммутации сильноточной нагрузки. В следующем примере MOSFET быстро включается и выключается 5 раз в секунду.

Примечание: Схема показывает мотор и диод защиты, но другие, не индуктивные, нагрузки могут включаться без диода.

```
int outPin = 5;      // выходной вывод для MOSFET

void setup ()
{
       pinMode (outPin, OUTPUT);         // задаём вывод 5 как выход
}

void loop ()
{
for (int i=0; i<5; i++)      // цикл 5 раз
{
       digitalWrite (outPin, HIGH);         // включаем MOSFET
       delay (250);                         // пауза в ¼ секунды
       digitalWrite (outPin, LOW);          // выключаем MOSFET
       delay (250);                         // пауза в ¼ секунды
       }
delay (1000);                               // пауза 1 секунда
}
```

pwm выход

Широтно-импульсная модуляция (PWM) — это способ имитировать аналоговый выход с помощью импульсного сигнала. Это можно использовать для гашения и увеличения яркости светодиода или позже для управления сервомотором. Следующий пример медленно увеличивает яркость и гасит LED, используя цикл for.

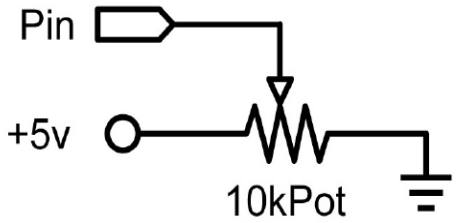
```
int ledPin = 9;                    // PWM вывод для LED

void setup () {}                   // setup не нужен

void loop ()
{
      for (int i=0; i<=255; i++)   // растущее значение для i
      {
      analogWrite (ledPin, i); // устанавливаем уровень яркости для i
      delay (100);                 // пауза 100 миллисекунд
      }

      for (int i=255; i>=0; i--)   // уменьшающееся значение для i
      {
      analogWrite (ledPin, i); // устанавливаем уровень яркости для i
      delay (100);                 // пауза 100 миллисекунд
     }
}
```

вход с потенциометра

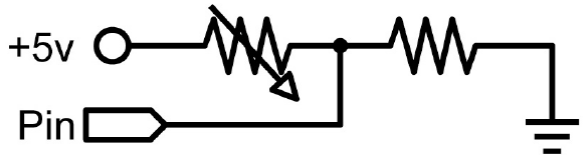


Использование потенциометра и одного из аналоговых портов Arduino (аналого-цифрового преобразователя (ADC)) позволяет читать аналоговые значения в диапазоне 0-1023. Следующий пример показывает использование потенциометра для управления временем мигания светодиода LED.

```
int potPin = 0;                 // входной вывод для потенциометра
int ledPin = 13;                // выходной вывод для LED
void setup ()
{
pinMode (ledPin, OUTPUT);       // объявляем ‘ledPin’ как OUTPUT
}

void loop ()
{
digitalWrite (ledPin, HIGH);    // включаем 'ledPin’
delay (analogRead(potPin));     // пауза
digitalWrite (ledPin, LOW);     // выключаем ‘ledPin’
delay (analogRead (potPin));    // пауза
}
```

вход от переменного резистора



Переменные резисторы включают фотоприёмники, термисторы, тензодатчики и т.д. Данный пример использует функцию чтения аналогового значения и задаёт время паузы. Этим управляется скорость, с которой меняется яркость светодиода LED.

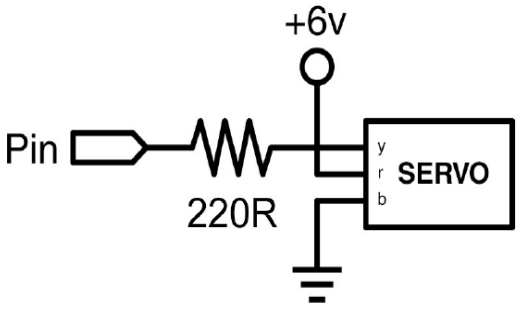
```
int ledPin = 9;             // PWM для LED
int analogPin = 0;          // переменный резистор на аналоговом выводе 0

void setup () {}            // setup не нужен

void loop ()
{
for (int i=0; i<=255; i++)      // увеличивающееся значение для i
{
analogWrite (ledPin, i);        // устанавливаем уровень яркости по i
delay (delayVal());             // берём значение времени и паузы
}
for (int i=255; i>=0; i--)`     // уменьшающееся значение для i
{
analogWrite (ledPin, i);        // устанавливаем уровень яркости по i
delay (delayVal());             // берём значение времени и паузы
}

int delayVal()
{
int v;                                  // создаём временную переменную
v = analogRead (analogPin);             // читаем аналоговое значение
v /= 8;                                 // конвертируем 0 – 1024 в 0 – 128
return v;                       // возвращаем окончательное значение
}
```

серво вывод



Любительские сервомашинки — это разновидность полу-автономного мотор-редуктора, который может поворачиваться на 180^0. Всё, что нужно — это отправлять импульсы каждые 20 мС. В данном примере используется функция servoPulse для поворота мотора от 10^0 до 170^0 и обратно.

```
int servoPin = 2; // привод соединён с цифровым выводом 2
int myAngle;      // угол привода грубо 0 – 180
int pulseWidth;   // переменная функции servoPulse()
void setup ()
{
pinMode (servoPin, OUTPUT);     // устанавливаем вывод 2 на выход
}
int servoPulse (int servoPin, int myAngle)
{
pulseWidth (myAngle * 10 + 600); // определяем паузу
digitalWrite (servoPin, HIGH);       // устанавливаем привод в HIGH
delayMicroseconds (pulseWidth);  // микросекундная пауза
digitalWrite (servoPin, LOW);        // устанавливаем привод в LOW
}
void loop ()
{       // привод стартует с 10 и поворачивается до 170 градусов
for (myAngle = 10; myAngle <= 170; myAngle++)
{
servoPulse (servoPin, myAngle);          // отправляем вывод и угол
delay (20);                              // обновляем цикл
}
// привод стартует со 170 и поворачивается до 10 градусов
for (myAngle = 170; myAngle >= 10; myAngle--)
{
servoPulse (servoPin, myAngle);          // отправляем вывод и угол
delay (20);                              // обновляем цикл
}

}
```

Приложение Б. Работа с модулем Arduino в других средах разработки

Внимание! При работе с модулем Arduino в других средах разработки следует внимательно относиться к конфигурации микроконтроллера (Fuses). До тех пор, пока вы точно не знаете, к чему может привести изменение конфигурации, настоятельно советую загружать в модуль только программу (Flash).

В основном речь пойдёт о программах для Windows. Их установку и работу в Linux мы рассмотрим отдельно, как и программы для Linux, работающие с микроконтроллерами AVR. Советы по установке программ и самим программам можно найти на сайте RoboCraft, о котором часто упоминалось в книге, и на сайте DI HALT'a «Электроника для всех»:

http://easyelectronics.ru/

Первая из программ, это AVR Studio фирмы Atmel. Следуя советам бывалых, перед установкой этой программы следует установить программу WinAVR, которая и сама позволяет писать программы на Си и ассемблере.

При работе с языком программирования Си используется свободная версия компилятора GCC.

Итак, установив обе программы в Windows, я использую Vista и версию AVR Studio 4.18, можно запускать программу, которая появится в основном меню в разделе Atmel AVR Tools. При запуске программы вначале открывается диалог, позволяющий создать новый проект или открыть существующий. От этого диалога можно отказаться, но не советую это делать в самом начале работы с программой.

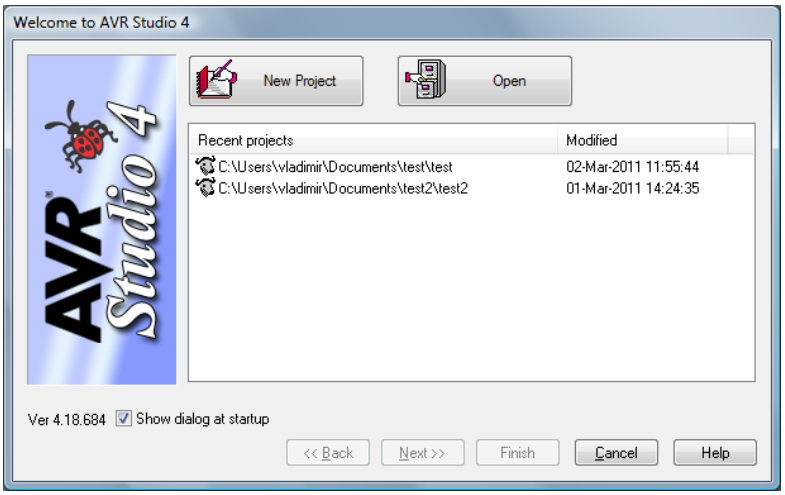


Рис. 1. Диалог выбора продолжения работы с AVR Studio

Если снять галочку рядом с надписью «Show dialog at startup» (показывать диалог при запуске), то диалог не будет появляться.

Выбрав кнопку (в верхней части) «New Project», вы переходите в окно помощника создания нового проекта. Поначалу, пока не освоитесь полностью, обратите внимание на отмеченные ниже пункты:

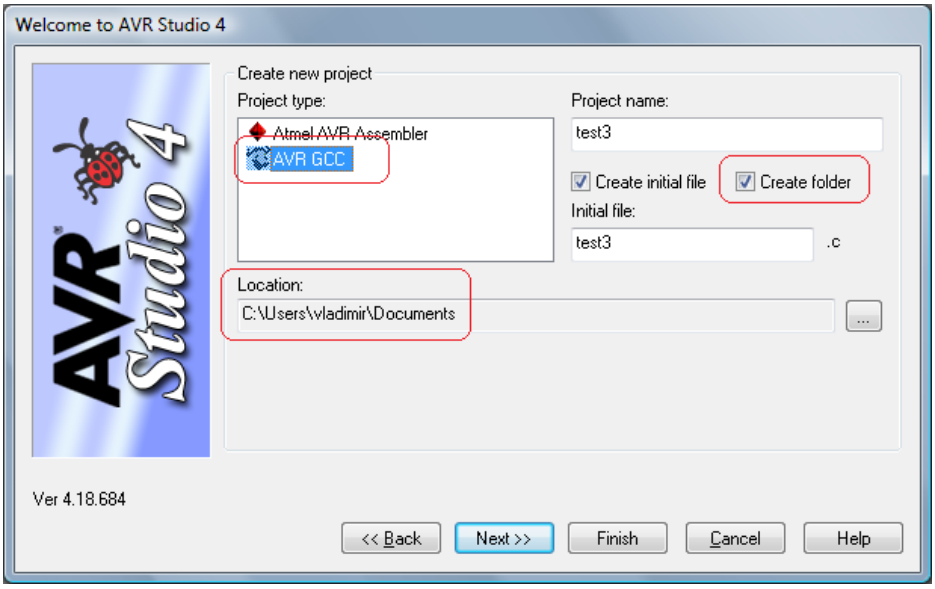


Рис. 2. Выбор языка программирования и атрибутов создаваемого файла

Отметив AVR GCC, вы предполагаете работу с языком Си. Проверьте, стоит ли галочка опции «Create folder». Иногда её нет, и файл сохраняется вместе со всеми остальными, что создаёт через некоторое время трудности в отыскании нужного файла. И местоположение файла, играет ли это роль в данном случае, не берусь судить, но лучше размещать свои проекты при работе с программами разработки в директориях, названных одним словом, написанным латиницей. Так будет меньше проблем. Например, некоторые программы не могут работать, если путь к нужным им файлам выглядит как «Program Files». Или, если ваша папка с документами обозначена как «Документы». Можно, конечно, проверить это, но лучше сразу вырабатывать привычку избегать подобных осложнений. Иногда они выявляются легко, но в ряде случаев сообщения об ошибках столь туманны, что и не сразу поймёшь, в чём дело.

Закончив с атрибутикой файла, можно нажать кнопку «Finish», если вам ничего больше не нужно, но лучше нажать кнопку «Next>>».

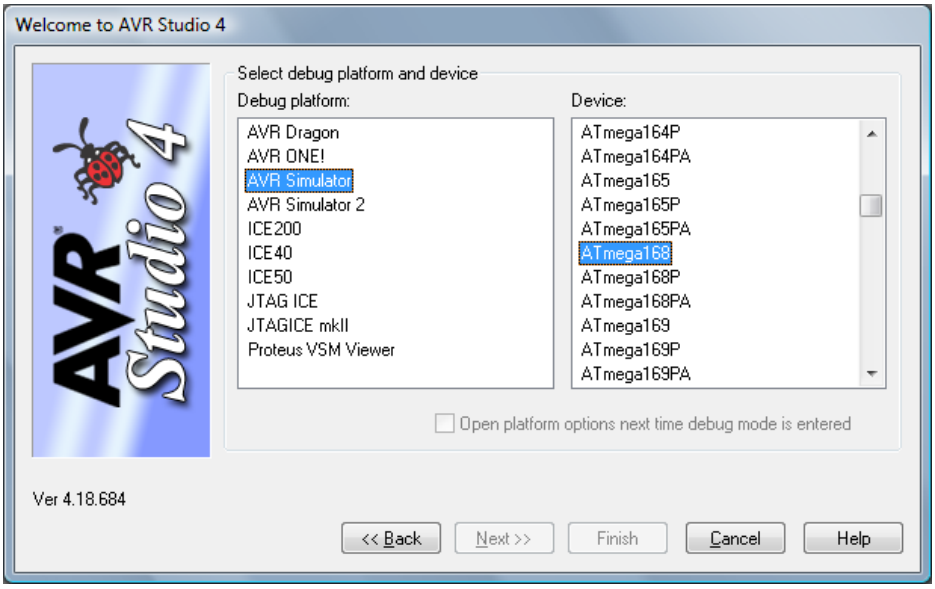


Рис. 3. Выбор отладочных средств и модели микроконтроллера

В этом окне диалога вы выбираете отладочные средства и модель вашего микроконтроллера. Если у вас есть программа Proteus, очень хорошая, но не бесплатная, то выбирайте Proteus VSM Viewer.

Завершив выбор, можно закончить работу помощника создания нового проекта (Finish).

Я не буду рассказывать о том, как работать с AVR Studio – есть руководство, есть много сайтов в Интернете, где хорошо рассказывается об этом. То же и о работе с компилятором AVR GCC. Приведу только один пример «классической» программы для модуля Arduino.

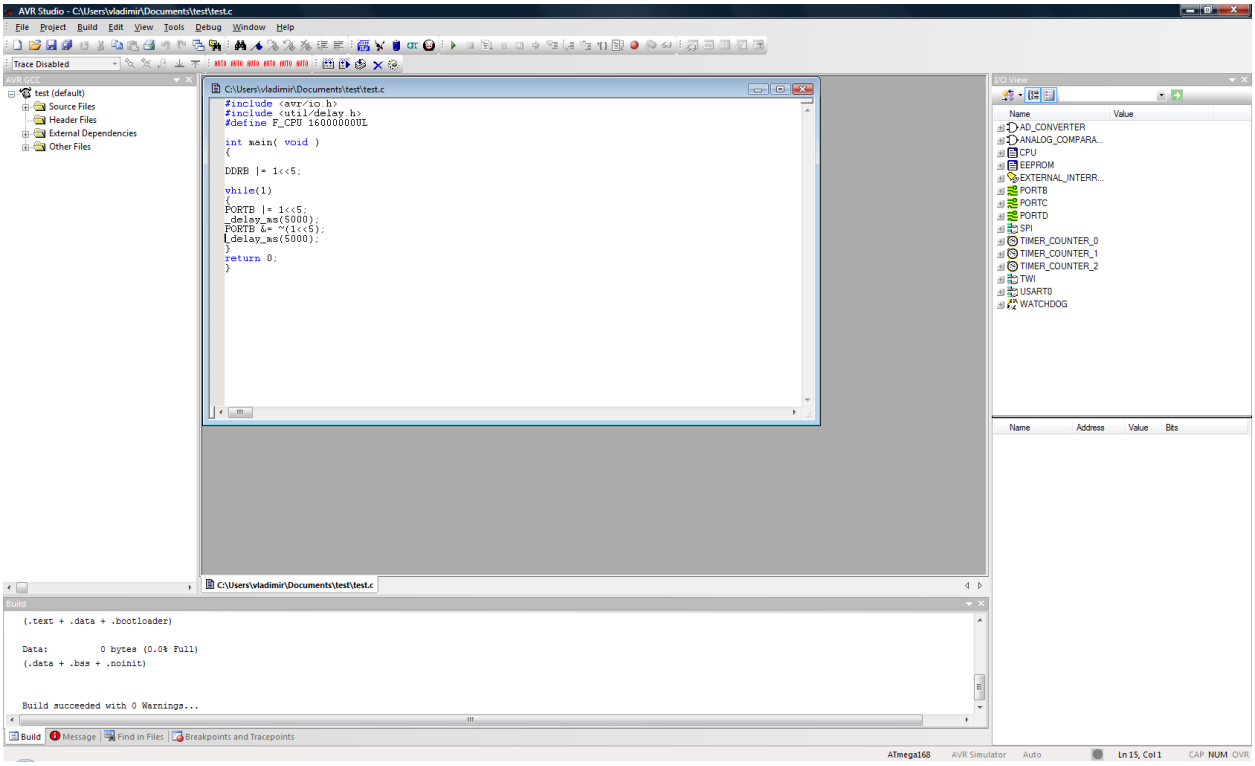


Рис. 4. Первый проект для модуля Arduino

Текст программы я приведу ещё раз.

```
#include <avr/io.h>
#include <util/delay.h>
#define F_CPU 16000000UL

int main( void )
{
DDRB |= 1<<5;
      while(1)
      {
      PORTB |= 1<<5;
      _delay_ms(5000);
      PORTB &= ~(1<<5);
      _delay_ms(5000);
      }
return 0;
}
```

Программу я «срисовал», выбрав самую простую. Первый оператор в основной программе устанавливает вывод RB5 (вывод 13 модуля Arduino) на выход. В цикле while этот вывод переводится в состояние высокого уровня, а после паузы в 5 секунд, в низкое.

Написав программу, в разделе «Build» основного меню выбираем команду «Build», и транслируем текст. Если нет ошибок, то вы получаете сообщение об удачном завершении, если есть ошибки, то они будут выведены с предупреждающими красными пометками и расшифровкой характера ошибки. Нам нужен hex-файл, который вы найдёте в папке проекта, где появится папка «default».

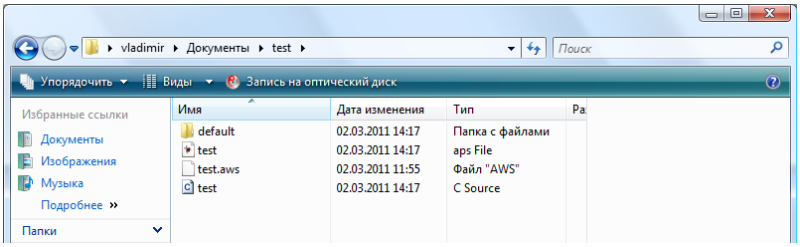


Рис. 5. Созданная программой папка для размещения оттранслированных файлов

Среда разработки AVR Studio позволяет написать код программы и отладить его. Но она предназначена и для загрузки программы в микроконтроллер. Однако, хотя в списке программаторов есть stk500, использовать эту возможность мне не удалось.

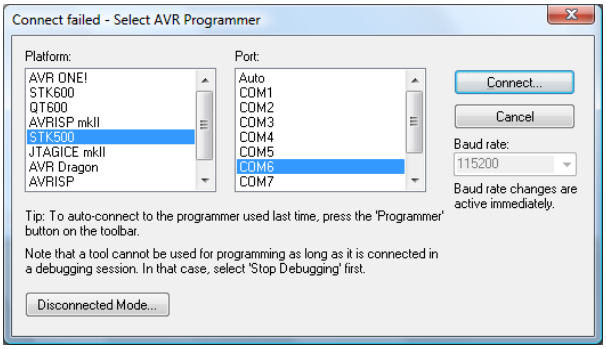


Рис. 6. Список программаторов

Найти этот список можно выбрав соединение с программатором.

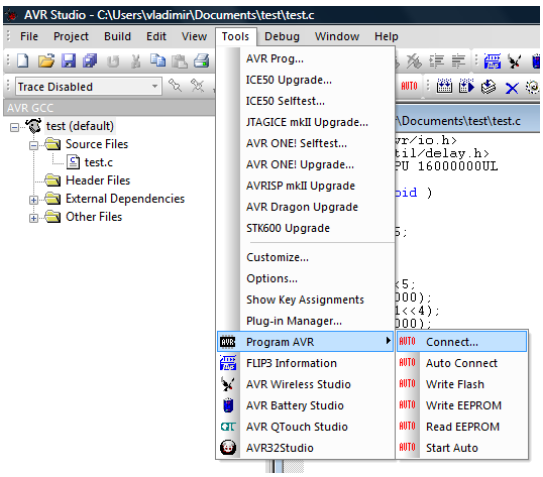


Рис. 7. Переход к программатору

Возможно, проблема кроется в скорости работы с программатором. Вместе с тем, если вы вспомните, как мы поступили с hex-файлом для превращения модуля Arduino в осциллогаф, то можете аналогичным образом загрузить и полученную в AVRStudio программу. Программатор avrdude по мнению многих очень мощный. А его единственный недостаток, по мнению некоторых (вроде меня), в отсутствии пользовательского интерфейса. Но, читая статьи на вышеуказанных сайтах, я выбрал достаточно удобную оболочку. Называется она SinaProg. Она не требует установки на компьютер, вы её можете расположить по своему усмотрению там, где вам удобнее. Единственное, на что я советую обратить внимание сразу – версия программы. В версии 1.4.5.10 есть возможность изменить скорость работы с портом.

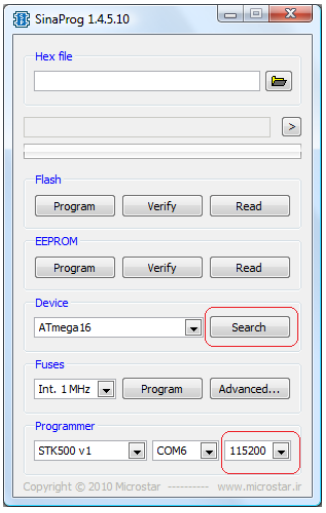


Рис. 8. Запуск графической оболочки SinaProg для avrdude

По умолчанию скорость работы выбрана большая. Если попробовать, выбрав версию программатора и порт (у меня он после очередного обновления Vista изменился с COM5 на COM6), нажать кнопку «Search», то поиск программатора завершится неудачей.

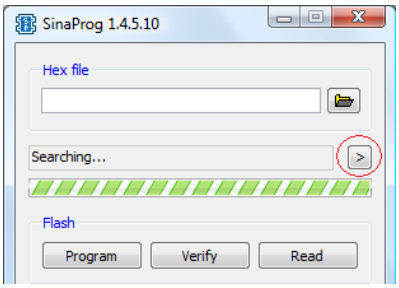


Рис. 9. Поиск программатора и модели контроллера

О причинах неудачи можно прочитать, если нажать по завершению процесса кнопку, отмеченную на рисунке. Изменим скорость, задав 19200.

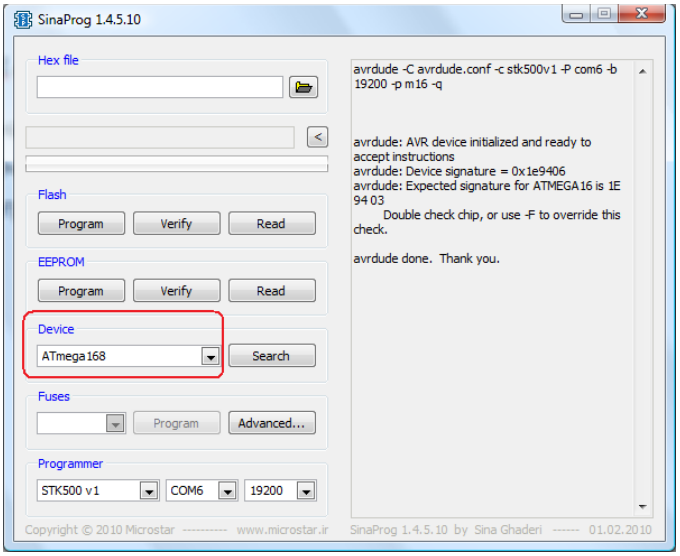


Рис. 10. Удачная попытка подключения программатора

Как видите, модель контроллера найдена. Следовательно, есть надежда, что программу, которую

следует загрузить, используя кнопку рядом с окном, обозначенным как «Hex file», мы сможем получить не в отладочном, а в «живом» виде.
Используем загрузку программы (кнопка «Program» под надписью «Flash»). Программа загружается, это видно по миганию светодиодов, связанных с портом, и светодиод мигает с частотой раз в 5 секунд. Если это не так, то загляните в AVR Studio в раздел «Project-Configuration options» основного меню.

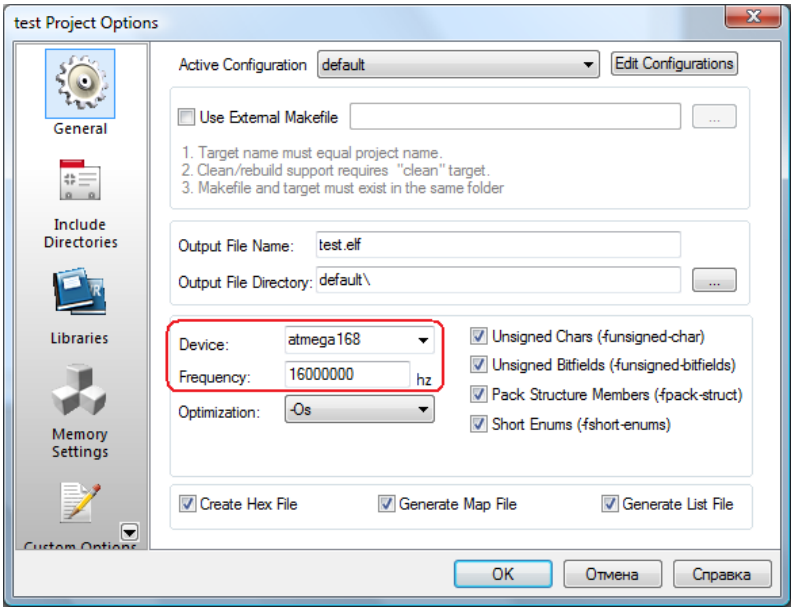


Рис. 11. Диалоговое окно настроек проекта AVR Studio

Задайте правильную частоту процессора (хотя мы её и задавали в программе), проверьте правильность модели контроллера. При неправильном выборе частоты тактового генератора микроконтроллера программа неверно рассчитает временные интервалы.

Впрочем, перед установкой AVRStudio мы установили программу WinAVR. Она сама позволяет работать с микроконтроллерами AVR, используя язык Си. Достаточно хорошо процесс описан на сайте RoboCraft:

http://robocraft.ru/blog/arduino/116.html

Я постараюсь не повторять его полностью, но бегло пересказать, как и что следует делать.
Начинать следует с загрузки блокнота – это текстовый редактор, приспособленный к работе с WinAVR. Его легко найти в основном меню операционной системы.

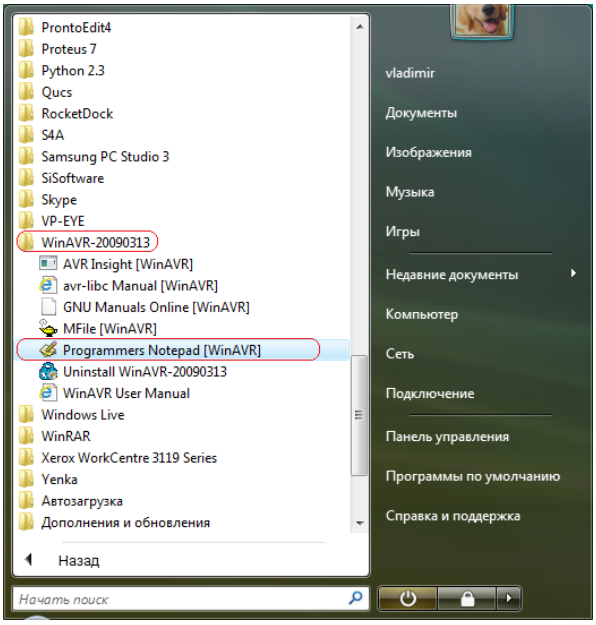


Рис. 12. Состав пакета WinAVR

Как и в предыдущем случае, текст программы можно «срисовать», то есть, скопировать и вставить в окно редактирования. Что я и сделаю, скопировав его выше и вставив в программу. Попутно я создам в месте хранения предыдущего проекта папку с подходящим названием, чтобы сохранить файл, записанный в редактор. Времена паузы я изменю, чтобы получить отличие от предыдущего эксперимента.

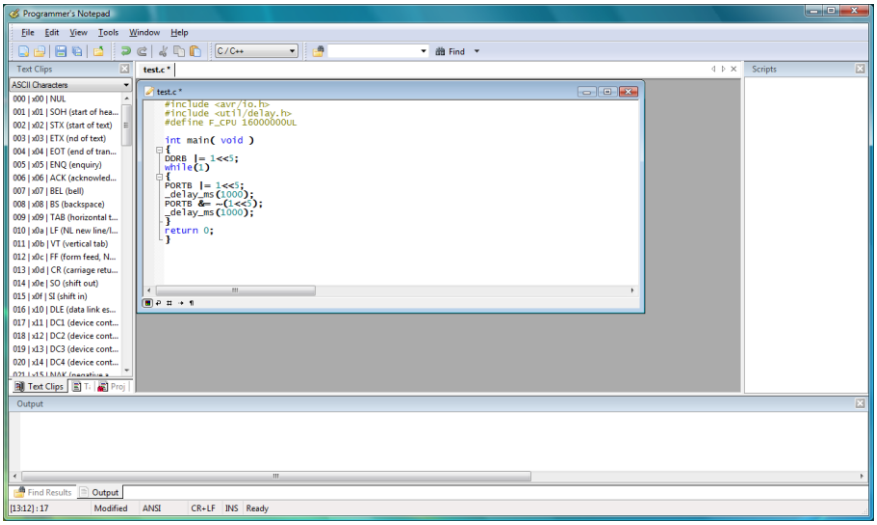


Рис. 13. Блокнот программиста пакета WinAVR

Как советует автор статьи, переносим из пакета WinAVR шаблон makefile'a. Найти его не трудно, но это зависит, видимо, от версии программы.

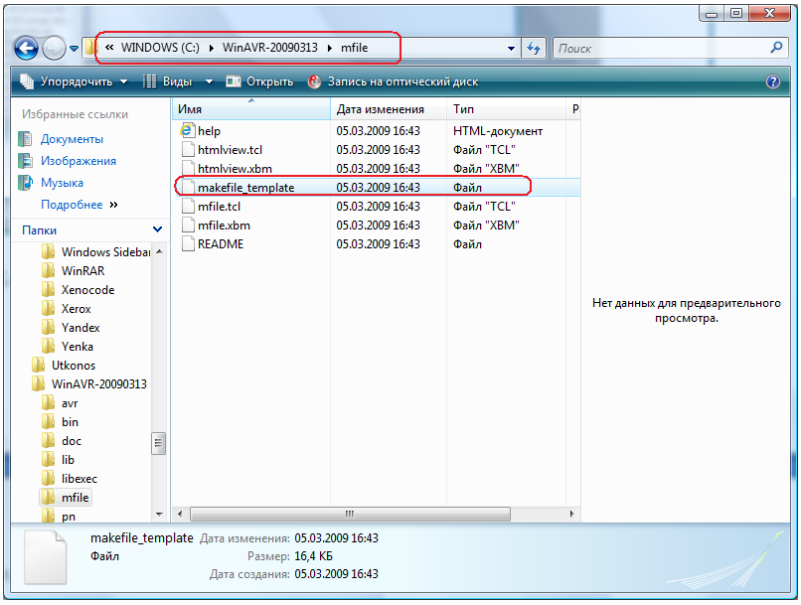


Рис. 14. Местонахождения шаблона makefile'a

Открываем его в редакторе (открывается второе окно) и правим: задаём модель, тактовую частоту, указываем наш рабочий файл, тип программатора и порт.

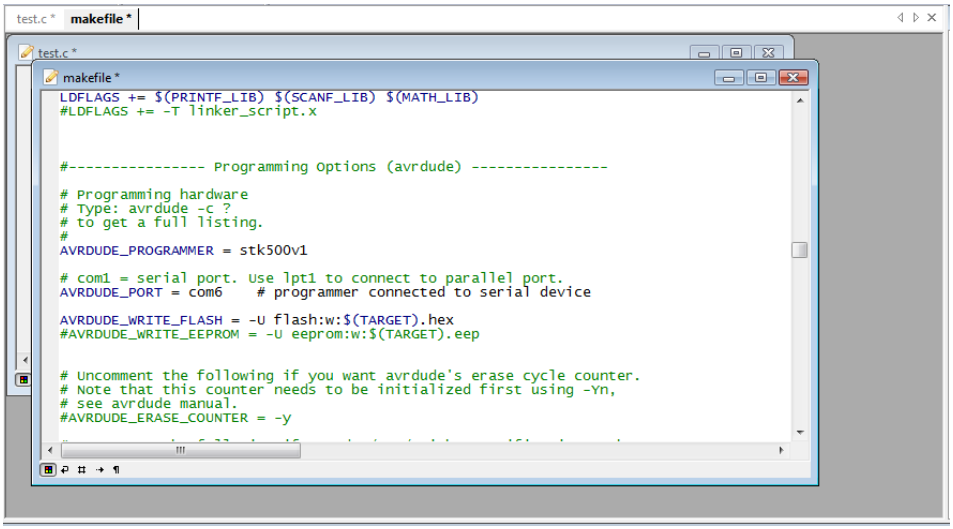


Рис. 15. Правка файла makefile

Кроме тех, что есть на картинке, правка коснулась следующих строк:

```
# MCU name
MCU = atmega168
# Processor frequency.
F_CPU = 16000000

# Output format. (can be srec, ihex, binary)
FORMAT = ihex

# Target file name (without extension).
TARGET = test
```

Внеся изменения, следует сохранить оба файла. И запустить преобразование исходного текста на языке Си в hex-файл.

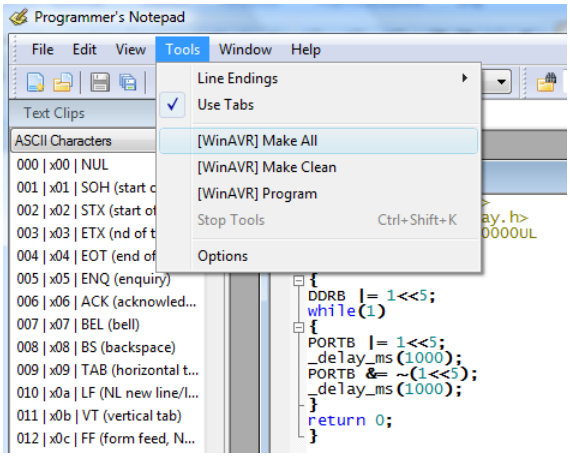


Рис. 16. Преобразование программы в hex-файл

Заглянув теперь в папку, где мы сохранили текст программы и makefile, мы увидим:

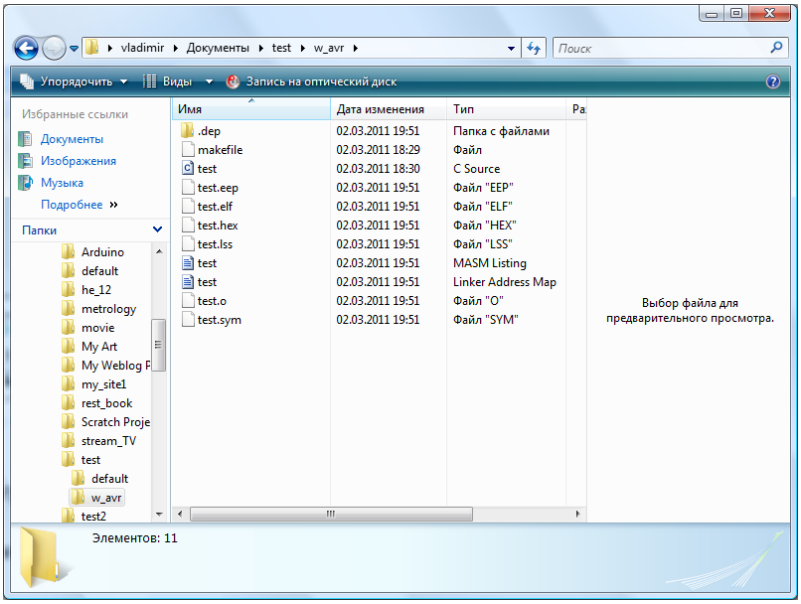


Рис. 17. Состав папки с проектом после выполнения команды «Make All»

Если после проделанной нами работы подключить модуль Arduino и выполнить команду Program (ниже отмеченной команды), то мы получим сообщение об ошибке. Что же мы упустили?

Скорость работы порта. Внесём ещё одно изменение в makefile, там где выполнены настройки программатора:

```
# Programming hardware
AVRDUDE_PROGRAMMER = stk500v1

# com1 = serial port. Use lpt1 to connect to parallel port.
AVRDUDE_PORT = com6 -b19200   # programmer connected to serial device

AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
#AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep
```

Я выделил добавленный мною параметр –b19200. Теперь программировать можно непосредственно из программы WinAVR (возможно, между двумя последовательными «заливками» программы модуль следует отключать).

На этом мне не хотелось бы завершать рассказ о программах, которые могут работать с модулем Arduino. Есть ещё одна замечательная программа, особенно для начинающих (часто при таких словах «бывалые» морщатся – но это их трудности). Эта программа не бесплатная и называется FlowCode for AVR. О программе и проектах, использующих программу FlowCode, можно узнать на сайте:

http://www.flowcode.info/

или прочитать в моей книге «Qucs и Flowcode».

Работа с программой проста, о графическом языке программирования мы говорили и в этой книге. Повторим программу «Hello World» для микроконтроллеров.

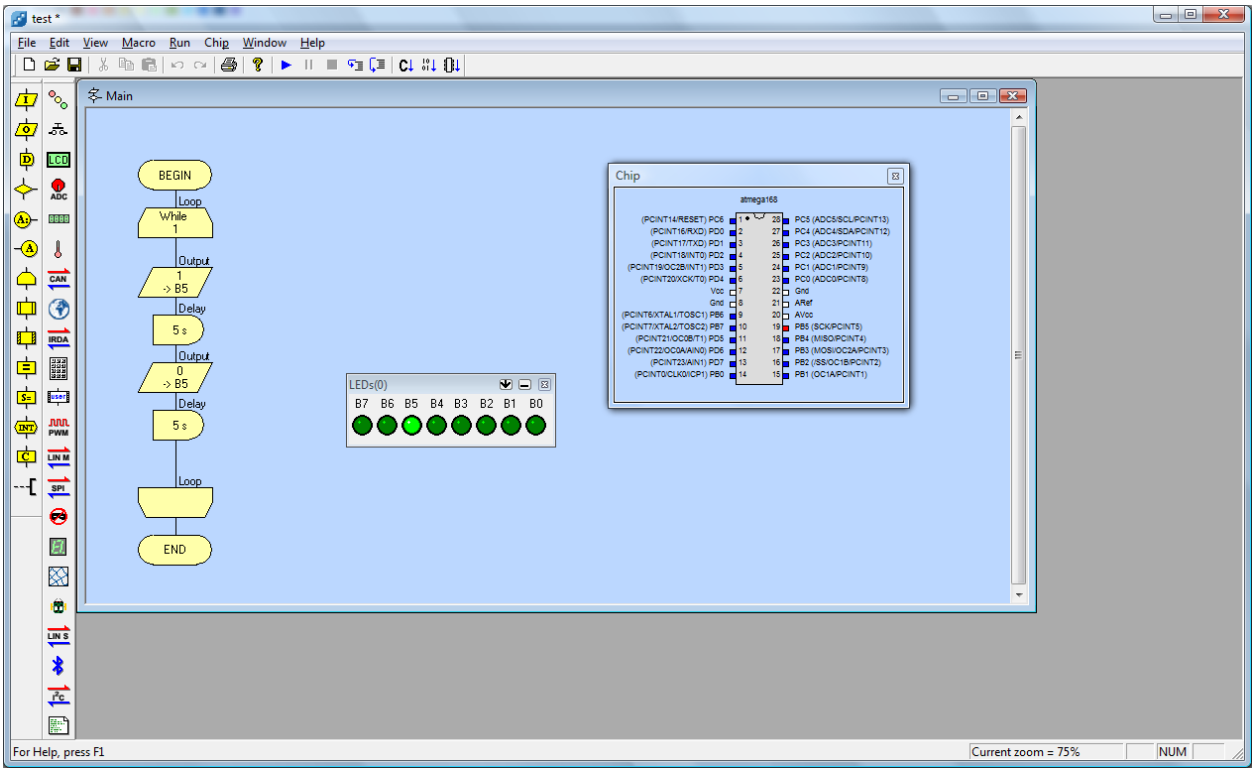


Рис. 18. Повторение программы в среде разработки Flowcode

Программа имеет встроенный отладчик. Написав (собрав) программу, можно проверить её работу. Не следует, пока вы не будете уверены в себе, менять конфигурацию (fuses) контроллера, но в разделе «Chip» основного меню на вкладке «Clock Speed…» следует указать тактовую частоту генератора.

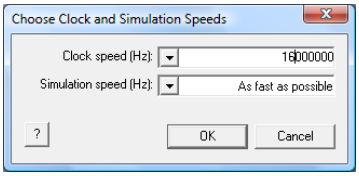


Рис. 19. Изменение тактовой частоты

Для работы с модулем Arduino в разделе «Chip» основного меню следует открыть раздел «Compiler Options…» и внести изменения в раздел программатора:

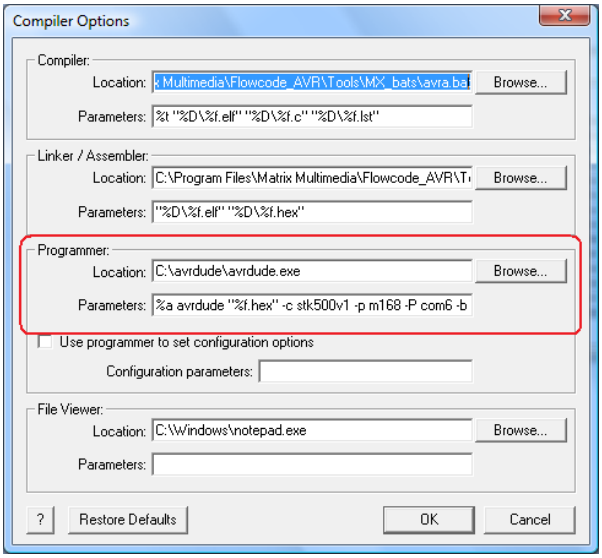


Рис. 20. Настройка программатора

Путь к программатору avrdude следует указывать реальный; я установил программу программатора, скопировав два файла из программы SinaProg в папку с именем программатора, которую счёл удобным оставить в корневой директории. В качестве параметров добавлена строка:

```
%a avrdude "%f.hex" -c stk500v1 -p m168 -P com6 -b 19200 -Uflash:w:"%f.hex":i
-C C:\avrdude\avrdude.conf
```

Сохранив эти изменения, можно нажать на кнопку «ОК». Если вам придётся вернуть прежние настройки, то есть кнопка «Restore Defaults». И вы готовы к загрузке программы в модуль Arduino. Есть пункт «Compile to Chip…» в разделе «Chip», есть иконка с рисунком микросхемы на инструментальной панели. Используйте любой вариант. Конечно, повторюсь, программа платная. Есть демо-версия, но в ней только одна модель микроконтроллера и, боюсь, слишком мало возможностей.

Я не уверен, что рассказал обо всех интересных программах общего назначения для работы с контроллерами AVR, но и то, что есть, предлагает достаточный выбор. Но это всё о Windows. А что же Linux? Не обошли ли его вниманием? Не обошли.

Во-первых, в Linux есть среда для работы Windows программ, которая называется Wine. Пользуясь советами DI HALT я намерен повторить установку AVR Studio 4 и других программ в Linux на трёх дистрибутивах: Fedora 14, ALTLinux 5.1 и openSUSE 11.3. Все три дистрибутива я использую с графической оболочкой KDE 4.

Процедуры установки и настройки нужных программ в разных дистрибутивах схожи, но могут отличаться деталями. Если такие отличия будут существенны, я о них расскажу. Итак, Fedora 14.

Перед установкой WinAVR, которая должна предшествовать установке AVRStudio 4, следует, используя терминал, скачать в Интернете одно дополнение к Wine. Называется оно winetricks. Для его получения используется терминал.

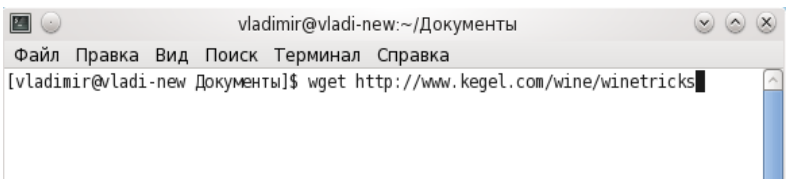


Рис. 21. Строка загрузки добавления к Wine

Строку, которую следует ввести, я повторю ещё раз (её можно будет скопировать из текста):

```
wget http://www.kegel.com/wine/winetricks
```

После нажатия на клавишу Enter вашей клавиатуры вы получите следующее:

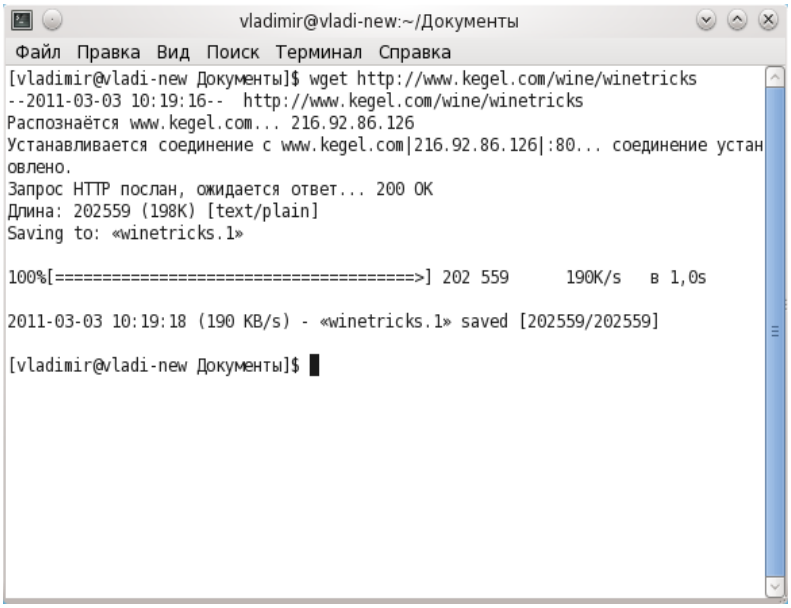


Рис. 22. Выполнение загрузки winetricks

Вновь используя терминал, запускаем winetricks командой: bash winetricks. В окне диалога с приложением вы отмечаете те элементы, которые вам нужны.

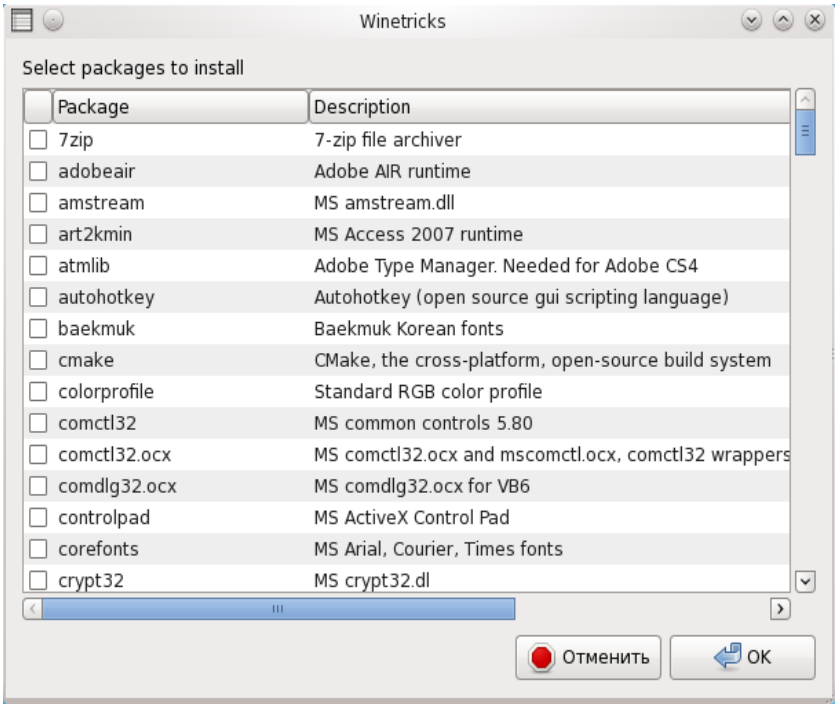


Рис. 23. Состав дополнений к Wine

Из предлагаемого DI HALT списка:

- corefonts
- dcom98

- gdiplus
- gecko
- mdac28
- msxml3
- vcrun2005
- allfonts
- fakeie6

находится не всё, что-то может не установиться, но добавим то, что установится. А fakeie6, возможно, будет выглядеть как ie6. Дождавшись завершения установки приложений к Wine, можно начинать установку программ. В Fedora для установки программ в среде Wine достаточно щёлкнуть по файлу установки правой клавишей мышки (я часто пакеты установки выкладываю в корневую директорию диска c:\ в Wine) и выбрать из выпадающего меню раздел запуска с помощью Wine.

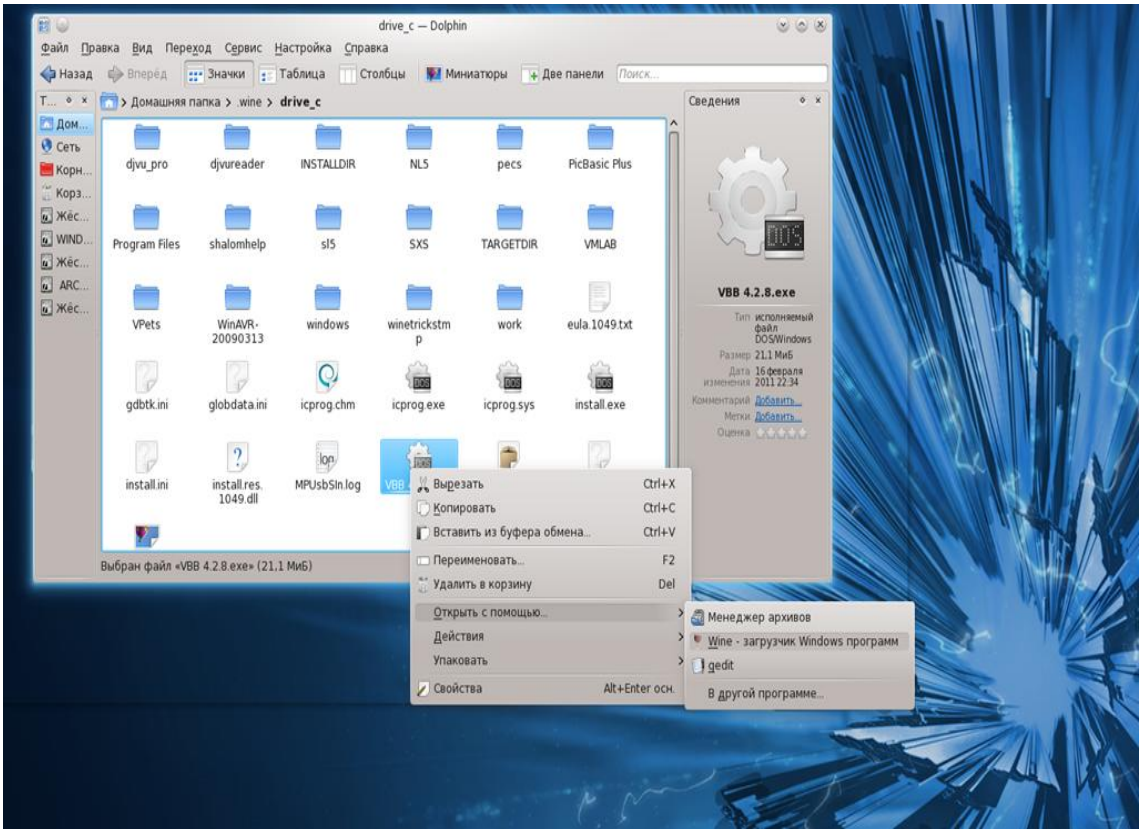


Рис. 24. Выполнение установки Windows программы в Linux

После установки двух программ, продолжая следовать советам, создадим ссылку:

```
ln -s /dev/ttyUSB0 <путь к вашей домашней директории>/.wine/dosdevices/com1
```

Для создания этой ссылки используйте переход в терминале к работе с правами root: либо командой su (и введите пароль), либо командой sudo.

Мне приходится повторить эту процедуру трижды. При этом в ALTLinux, я обновлял версию через Интернет, что может стать причиной, установку я произвожу, используя файловый менеджер Wine.

Но теперь в основном меню, где у меня есть раздел «Wine программы» я могу найти AVRStudio.

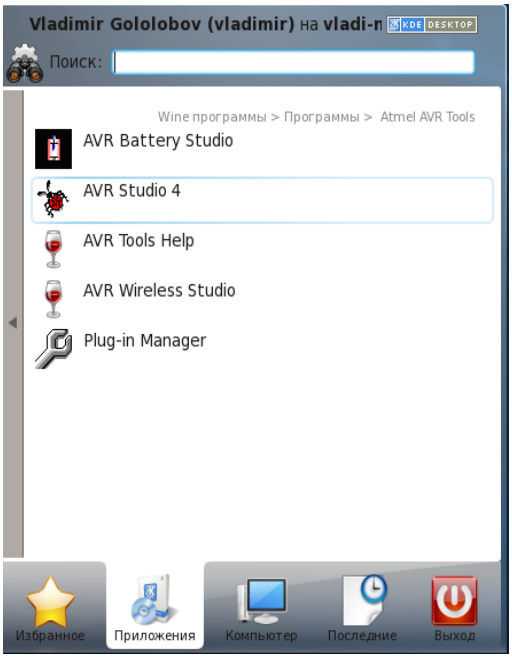


Рис. 25. Программа Windows в Linux

Как и в Windows, при запуске программы появляется окно диалога, в котором можно открыть уже существующий проект, а можно создать новый.

Рис. 26. Диалог создания нового проекта в AVR Studio 4

Как и в Windows, лучше, это моё мнение, создать новую папку, отметив соответствующую опцию «Create folder». И, по причине отсутствия — у вас может быть иначе — программы Proteus, я отмечаю AVR Simulator в следующем окне, когда нажимаю кнопку «Next>>».

Как и в Windows, можно скопировать нужный файл и вставить в окно редактора.

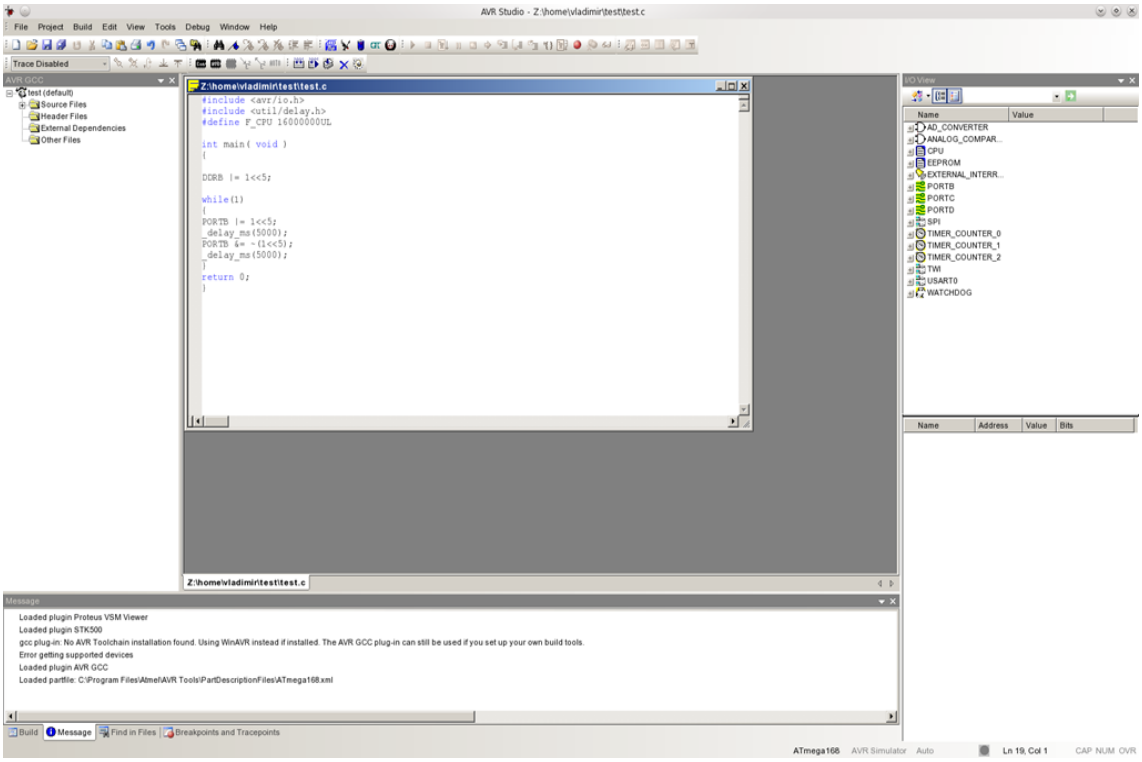


Рис. 27. Вид окна программы в Linux

Но не как в Windows, следует обратить внимание на одну деталь. Откроем окно свойств проекта (Ptoject-Configuration Options):

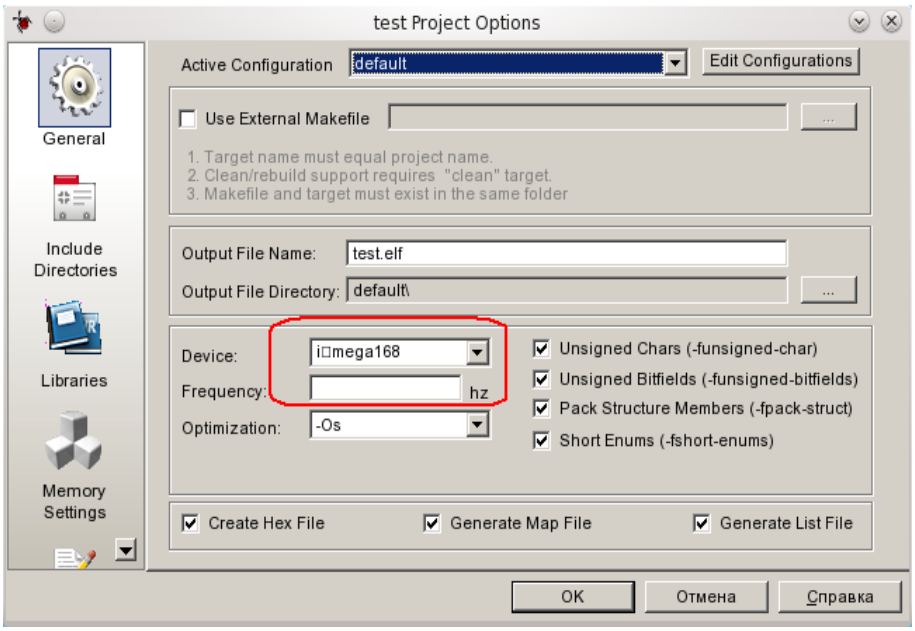


Рис. 28. Правка свойств проекта

Думаю, что возникает некоторая проблема со шрифтами — вместо Atmega168... Но это не беда, щёлкнув по кнопке, открывающей список моделей, выбираем нужную модель.

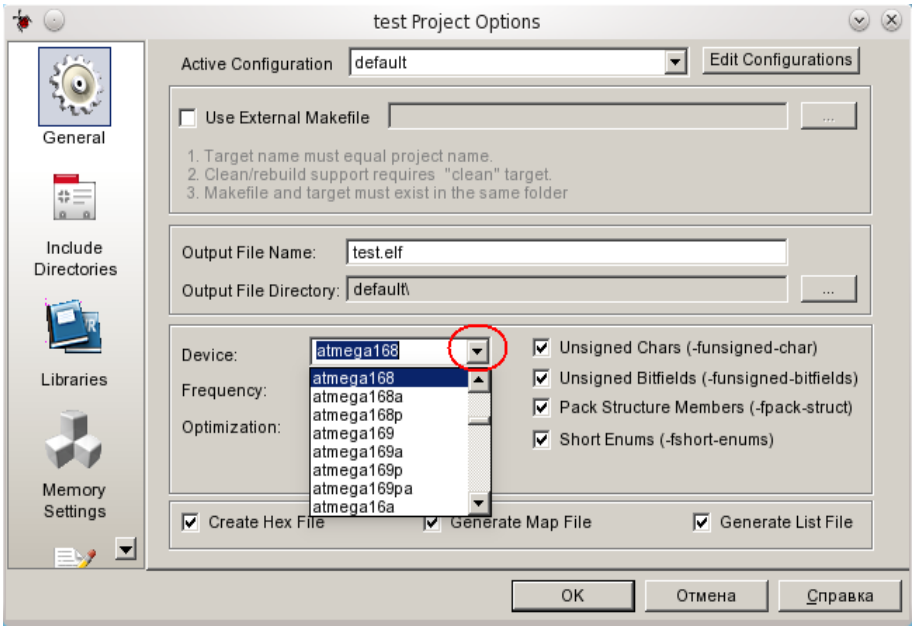


Рис. 29. Выбор модели микроконтроллера

И не забываем указать частоту в поле «Frequency:» 16000000.

Запускаем трансляцию файла, сохранив его предварительно, и убеждаемся в наличии и папки default, которую создаёт программа, и в наличии всех нужных файлов в ней.

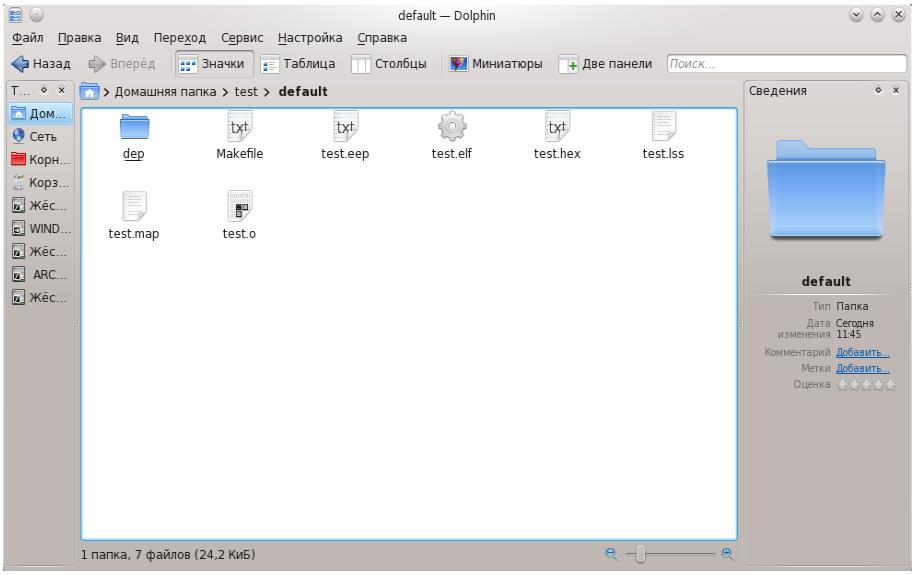


Рис. 30. Содержание папки проекта после выполнения трансляции

Но при попытке соединиться с программатором в Fedora возникает фатальная ошибка и программа «виснет». Я не исключаю, что это происходит от неудачной попытки установить что-то в winetricks. Но это не огорчает меня, поскольку я почти уверен в безуспешности попытки работать с модулем Arduino.

Как и в Windows, воспользуемся программой SinaProg:

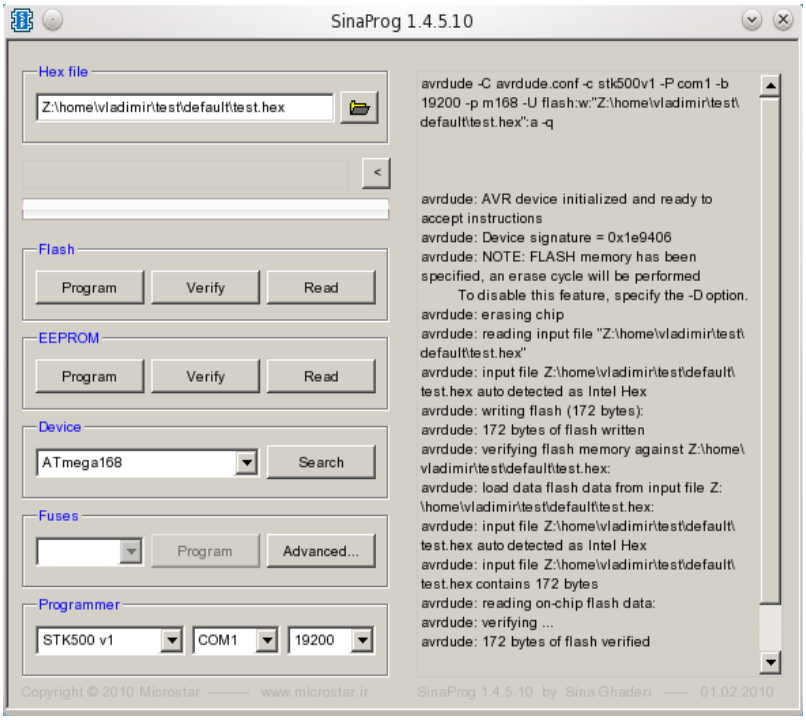


Рис. 31. Настройки программы SinaProg в Linux

Отличие в том, что в Wine использован порт COM1. И работает ссылка (спасибо DI HALT).

В ALTLinux эта программа не заработала. Но можно использовать команду в терминале. Или использовать WinAVR. Я почти забыл о ней.

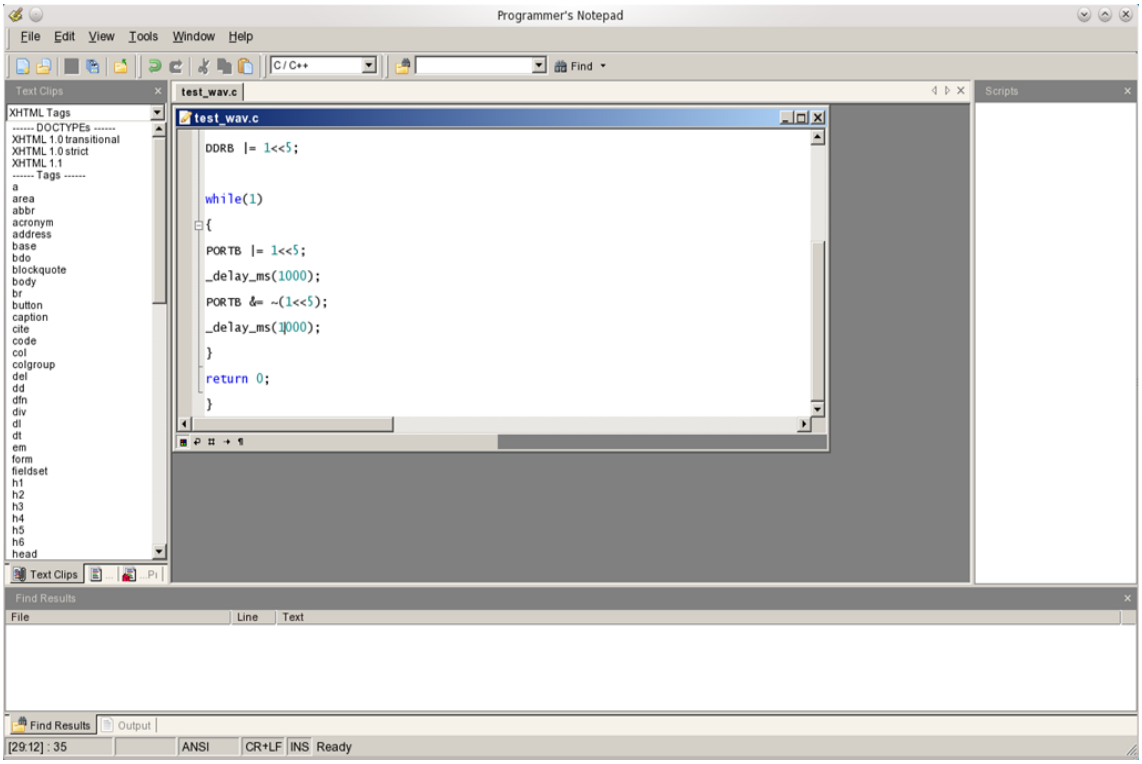


Рис. 32. Программа WinAVR в Linux

Переносим шаблон makefile, как мы это делали в Windows, и вносим те же правки.

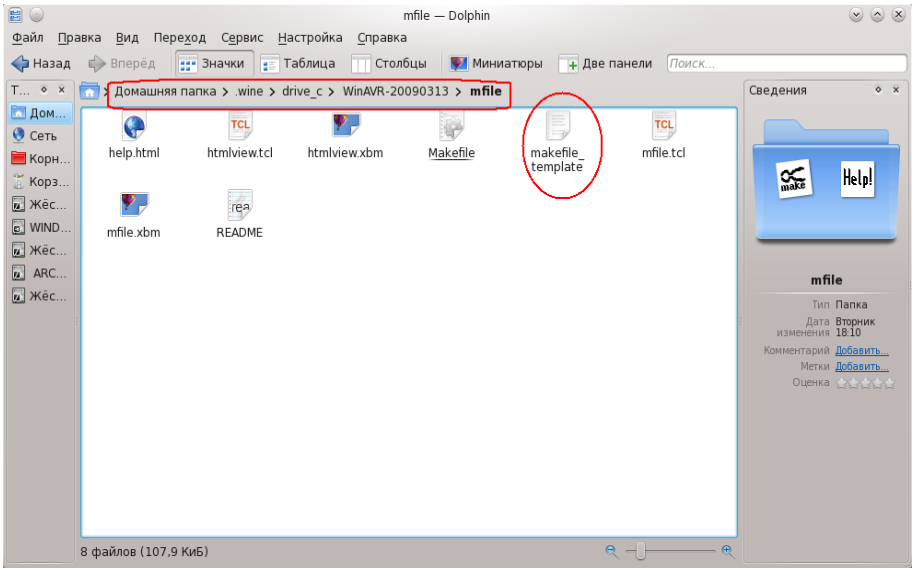


Рис. 33. Расположение шаблона makefile

Обратите внимание на путь к нужному файлу шаблона. Внеся изменения, переименуем его, удалив слово шаблон. И попробуем запустить... И получаем ошибку.

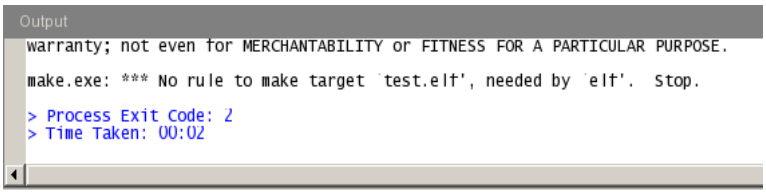


Рис. 34. Сообщение об ошибке при трансляции файла в WinAVR

В данном случае проблема возникает с файлом, который нужен для работы с программой Proteus. Но это мешает. И, обращаясь к советам бывалых, я выбираю, удалить из makefile всё, что касается elf-файла, или воспользоваться утилитой мастера создания makefile. Под именем Mfile [WinAVR] она есть в основном меню рядом с блокнотом программиста. Запускаем утилиту, заходим в раздел «Makefile» и задаём нужные параметры.

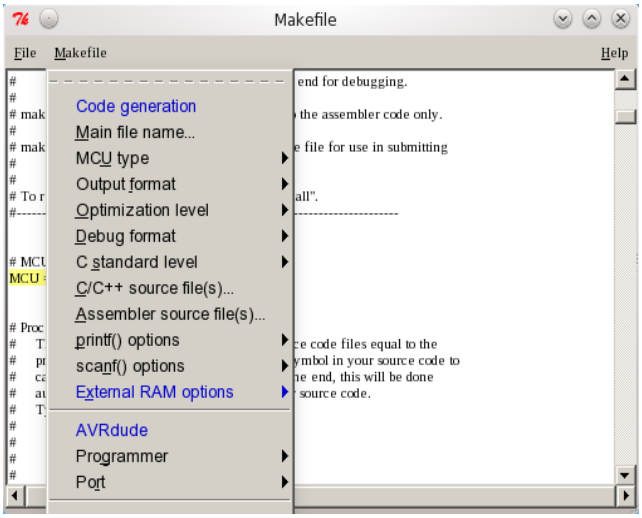


Рис. 35. Использование утилиты создания makefile

Из числа нужных мне параметров не удалось изменить частоту и настройки скорости COM1. Для

этого служит опция (в самом низу), разрешающая редактировать makefile. Но, отредактировав всё, сохранив файл рядом с исходным, я не могу выполнить трансляцию. Всё та же ошибка. Проверяю, перезагружаясь, как работает программа в ALTLinux и openSUSE. Всё работает. Возвращаюсь в Fedora, повторяю всё от начала и до конца. И... работает, включая загрузку программы в микроконтроллер. Что я могу сказать? Программа для Windows и, если с ней чудеса в Linux, то это мои трудности.

В Windows работала программа FlowCode для AVR. Интересно, будет ли она работать в Linux.

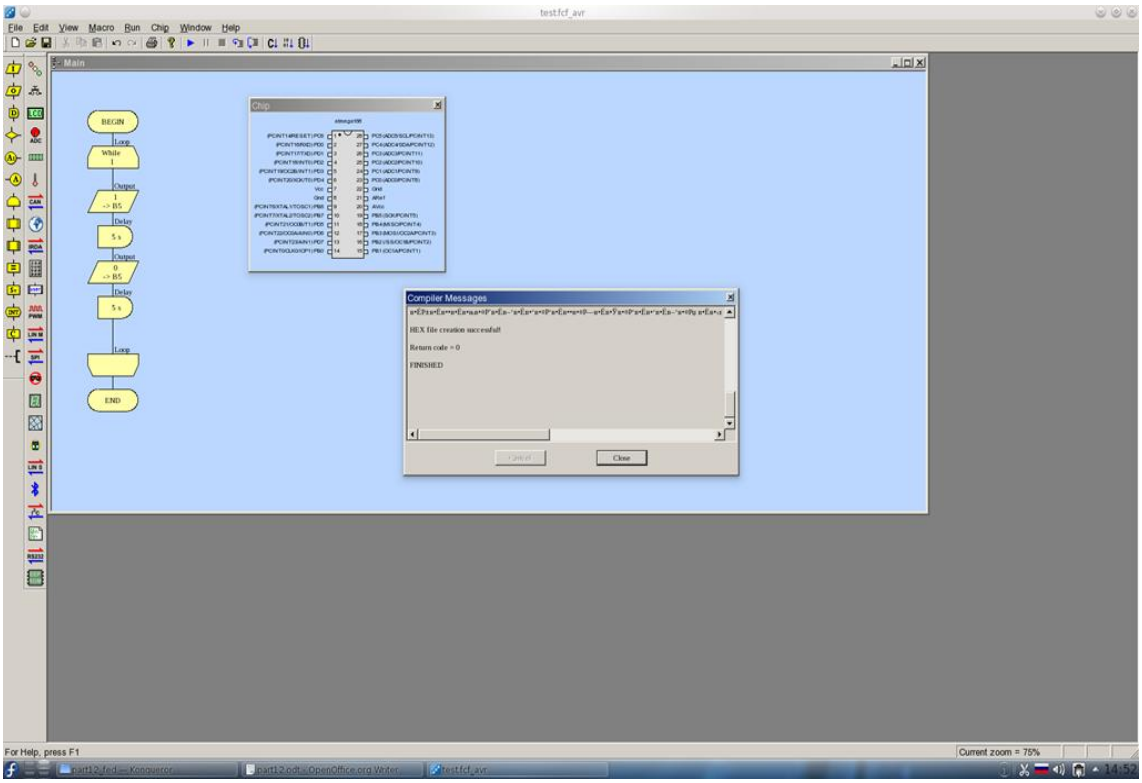


Рис. 36. Программа Flowcode в Linux

Работает. Правки, которые я делаю для программатора, повторяют те, что сделаны для Windows, но указываю я COM1. На диске C:\ (от Wine) я создаю в корневой директории папку с именем avrdude, куда копирую файлы из проекта Arduino-0022. Программа загружается.

Я не проверял Windows программы в Linux в больших форматах. Скажем, Flowcode, похоже, не будет поддерживать работу встроенных макросов и, возможно, вставок на Си. Но простые программы должны работать. И программы, не следует забывать, созданы для Windows.

В Linux есть своя программа для работы с AVR-контроллерами. Называется она kontrollerlab. В ALTLinux программа есть в репозитории. В Fedora и openSUSE её можно установить из исходных файлов.

В Fedora проблема возникает при использовании команды make. Появляются ошибки. Проще показать, что к чему, чем долго об этом рассказывать. Итак, скачиваем с сайта проекта исходный код:

http://www.cadmaniac.org/projectMain.php?projectName=kontrollerlab&section=download

Я его распаковываю в домашнюю папку, где он расположен в папке kontroller.

В терминале вначале переходим в эту папку: cd /home/vladimir/kontroller .

Следом даём команду: ./configure. Эта команда в Fedora проходит успешно, но в openSUSE требуется установить ещё ряд библиотек, касающихся X и Qt. Их можно отыскать по ключевым словам devel, X11, Qt и т.п.

Но (и в Fedora, и в openSUSE) после команды make появляются ошибки.

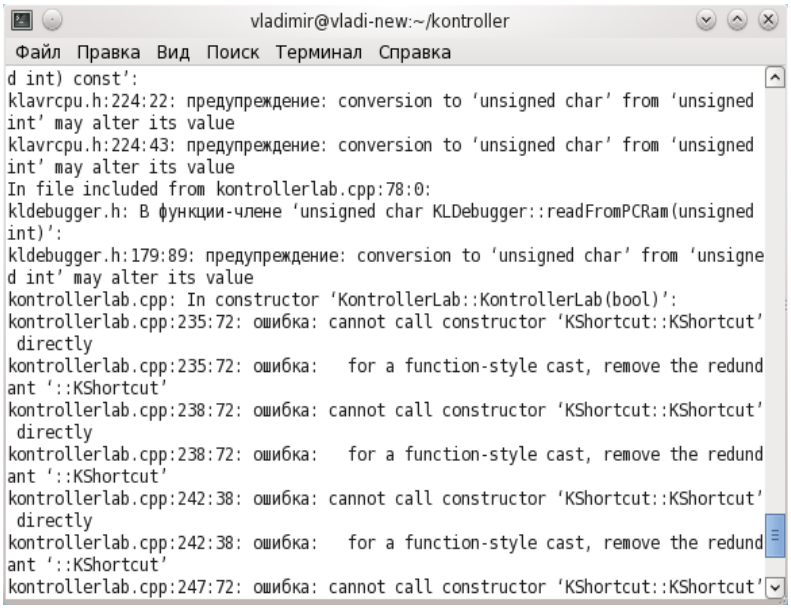


Рис. 37. Ошибки при создании программы kontrollerlab из исходного текста

Первая ошибка возникает в строке 235 файла kontrollerlab.cpp. О характере ошибки есть сообщение. Откроем этот файл...

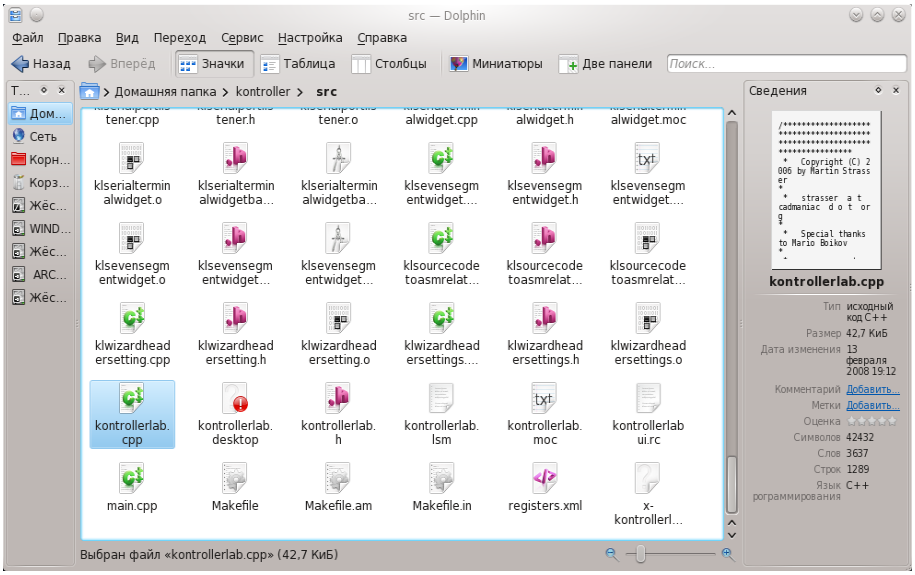


Рис. 38. Файл в исходном тексте программы, который нуждается в правке

...и найдём нужную строку.

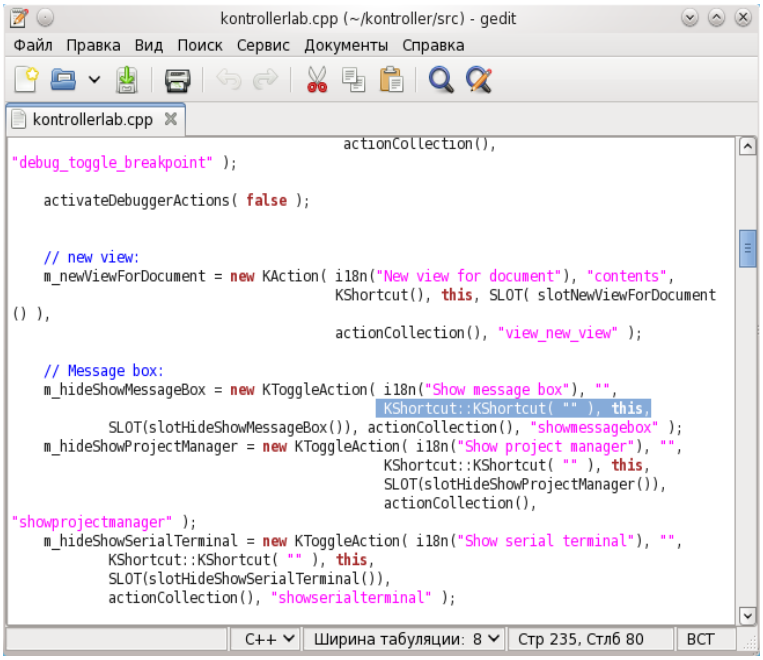


Рис. 39. Строка, которую нужно править

Заменим KShortcut::KShortcut("") на KShortcut(). Аналогично, просмотрев все указанные в перечне ошибок строки, вносим исправления. Запускаем make ещё раз, убеждаемся, что всё прошло без ошибок. И командой sudo make install устанавливаем программу. Устанавливается она по адресу: /usr/local/kde/bin/kontrollerlab. Именно эту команду следует ввести в терминал, чтобы запустить программу.

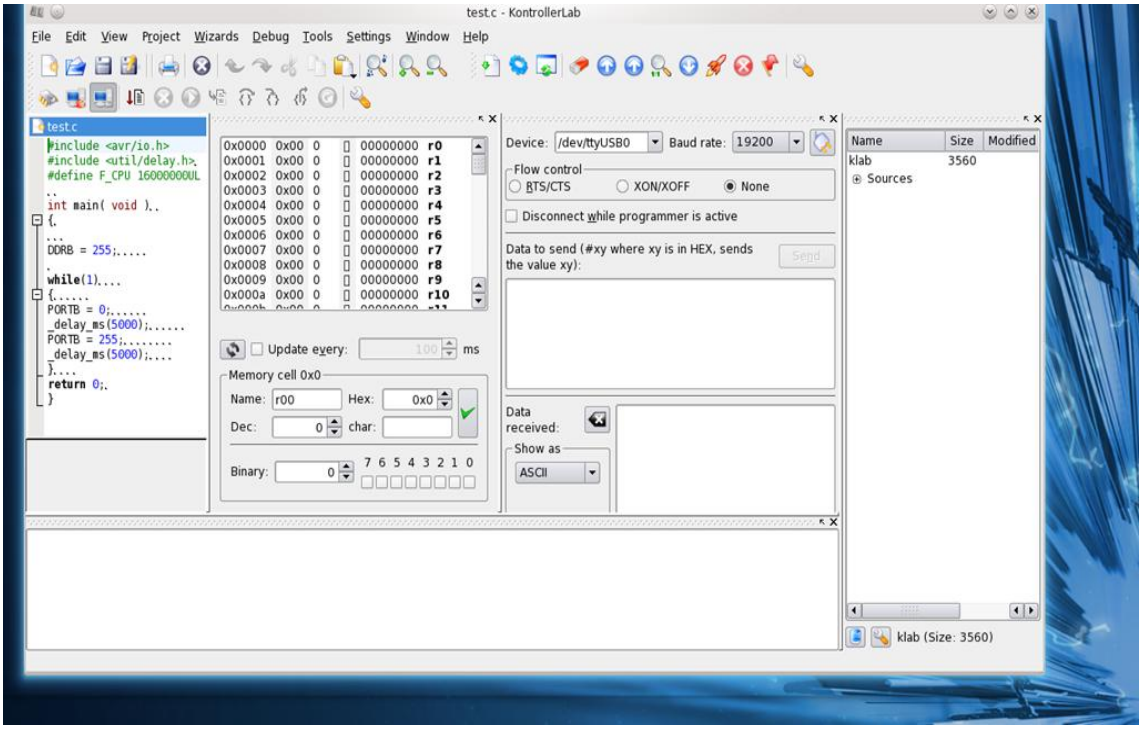


Рис. 40. Запуск программы kontrollerlab в дистрибутиве Fedora 14

А для удобства при последующем использовании программы можно создать то, что называется ярлыком в Windows и ссылкой на приложение в Linux, где прописывается команда запуска на закладке «Приложение».

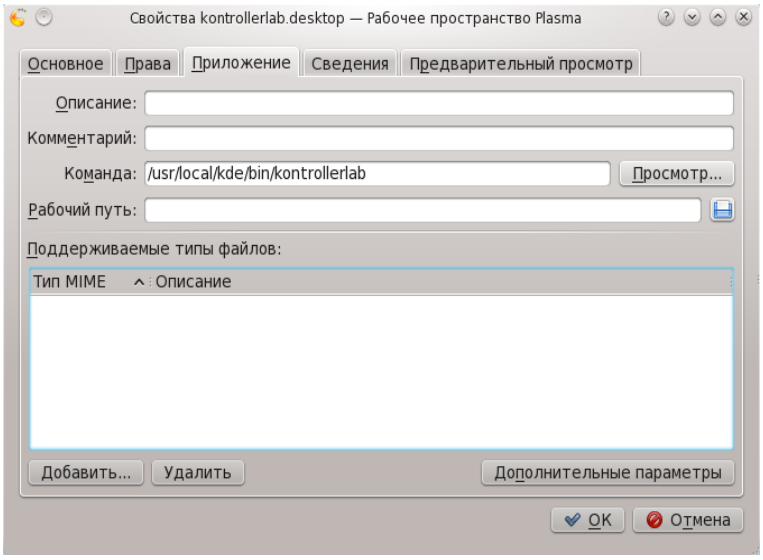


Рис. 41. Командная строка запуска программы

Подобрав подходящую картинку, кнопку запуска приложения располагаем на рабочем столе.

После запуска kontrollerlab в разделе «Project» выбираем «New Project», а появившемся диалоге указываем имя проекта и место, где он будет храниться.

Устанавливаем конфигурацию проекта (Project-Configure Project):

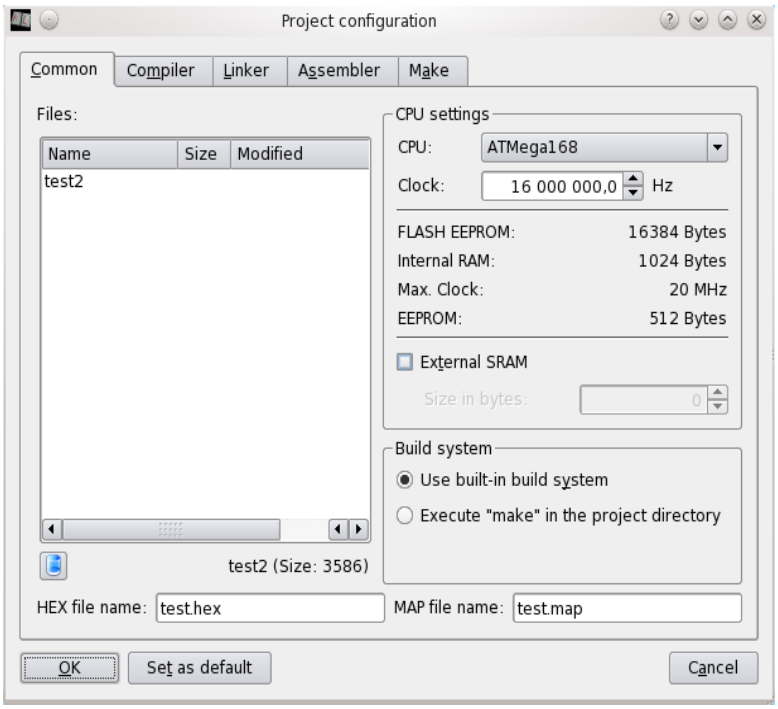


Рис. 42. Установка устройства и частоты в конфигурации проекта

На закладке «Compiler» устанавливаем опцию поддержки скорости работы процессора.

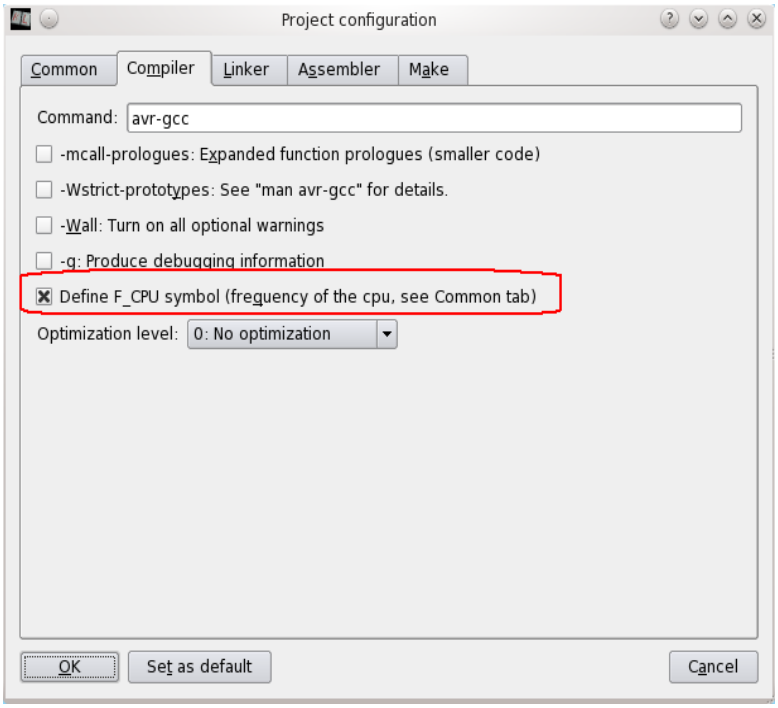


Рис. 43. Задание опции поддержки строки определения частоты в тексте программы

Если эту опцию не установить, то проходит компиляция, можно загрузить программу, но работать она будет так, как если бы тактовая частота была той, что установлена в программе по умолчанию, а записанное в исходном тексте определение частоты игнорируется. Перед тем как закрыть диалог, следует нажать кнопку «Set as default». Закрываем диалог кнопкой «OK».

Теперь следует настроить программатор (Project-Configure programmer): на первой закладке добавляем модель программатора, с которой работает модуль Arduino.

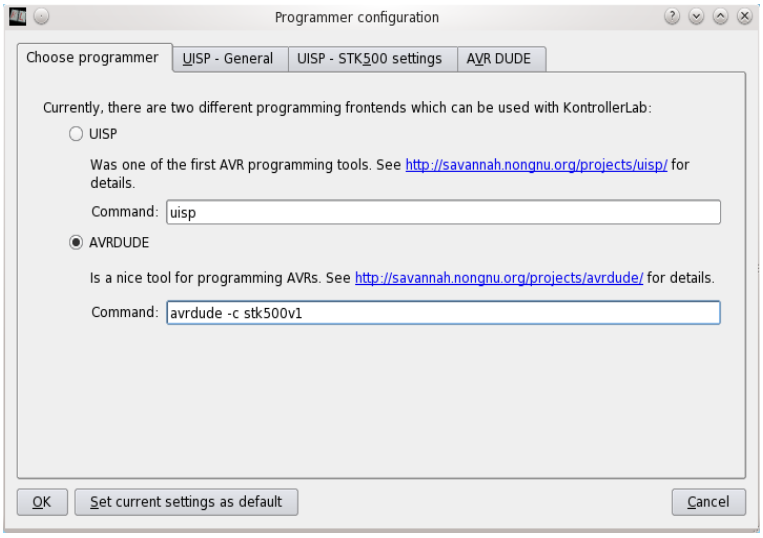


Рис. 44. Настройка программатора для работы с модулем Arduino

На закладке «AVR DUDE» устанавливаем две опции. Нажимаем кнопку «Set current settings as default» и затем кнопку «OK».

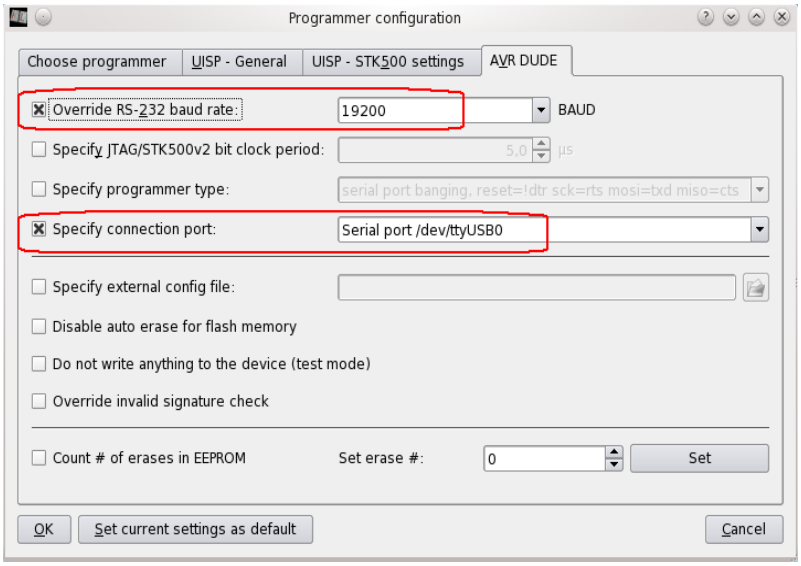


Рис. 45. Настройка на закладке программатора

Создаём новый файл (File-New, или кнопка на инструментальной панели), выбирая в окне диалога исходный файл на языке Си.

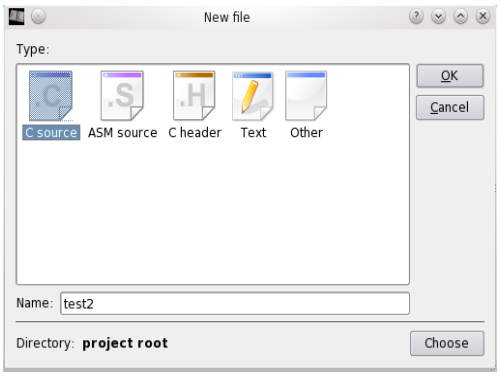


Рис. 46. Диалог создания нового файла

В окно текстового редактора текст программы можно перенести копированием, а затем сохранить файл. После сохранения он приобретает вид, который можно было видеть на рисунке в начале рассказа о программе kontrollerlab. Если в свойствах проекта поставить опцию «-g» (чуть выше опции поддержки тактовой частоты), то файл будет содержать информацию для отладчика.

Запустив компиляцию и компоновку проекта командой «Build project» раздела «Project» основного меню, мы получаем (если всё правильно) сообщение об удачном построении.

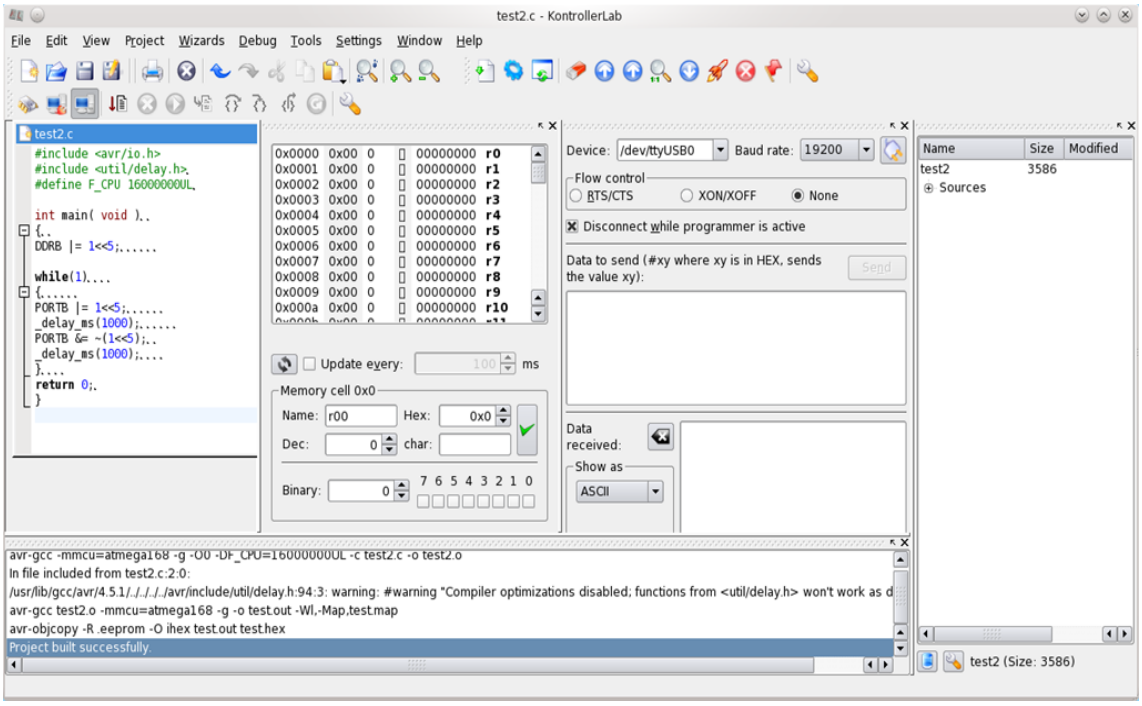


Рис. 47. Успешная трансляция и компоновка проекта

Загрузка программы в программатор выполняется командой «Upload» раздела «Project» основного меню, или можно использовать иконку с изображением ракеты на инструментальной панели. При удачной загрузке вы получите в окне вывода сообщение.

Рис. 48. Успешная загрузка программы в модуль Arduino

Чтобы убедиться, что всё правильно, можно изменить время паузы на 5 секунд и повторить операции по трансляции и загрузке.

Настройки программы kontrollerab в ALTLInux имеют больше особенностей, чем в других дистрибутивах. Хотя и avr-gcc, и сама программа загружались из репозитория ALT, при попытке оттранслировать программу сразу появляется сообщение, что файлы включённых заголовков не найдены.

Проблема решается, если указать место, где эти файлы находятся, явным образом. Заходим в пункт «Configure Project» раздела «Project» основного меню, открываем закладку «Compiler» и добавляем в команду несколько слов, указывающих к нужному месту в файловой системе.

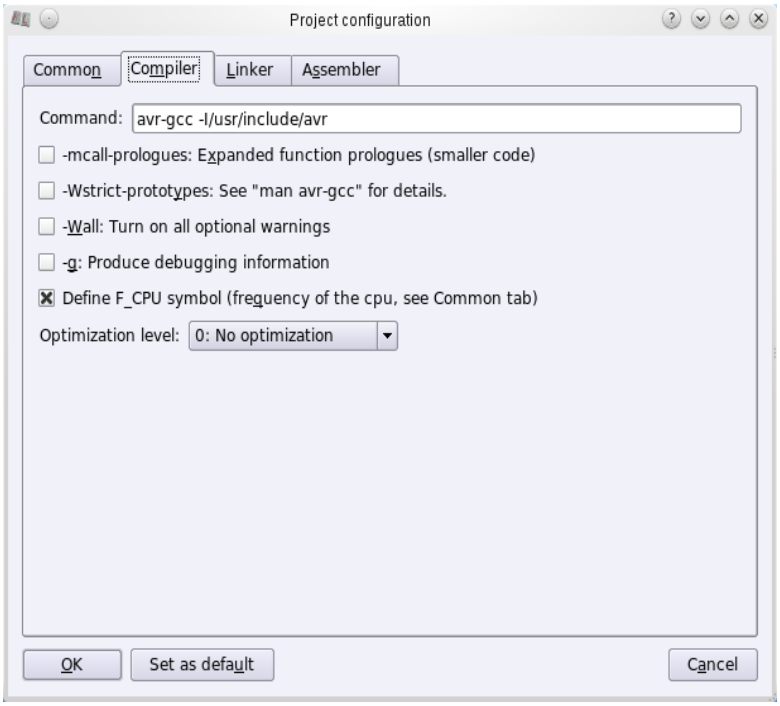


Рис. 49. Изменение команды вызова компилятора в ALTLinux

Строка должна выглядеть так:

```
avr-gcc -I/usr/include/avr
```

Попутно устанавливаем опцию «Define F_CPU symbol». Эта проблема касается всех дистрибутивов: без этой опции тактовая частота остаётся равной заданной по умолчанию.

Теперь компиляция (Prolect-Build project) проходит. В настройках программатора, помимо выбора, сделанного в основном окне, установлены опции на закладке «AVR DUDE».

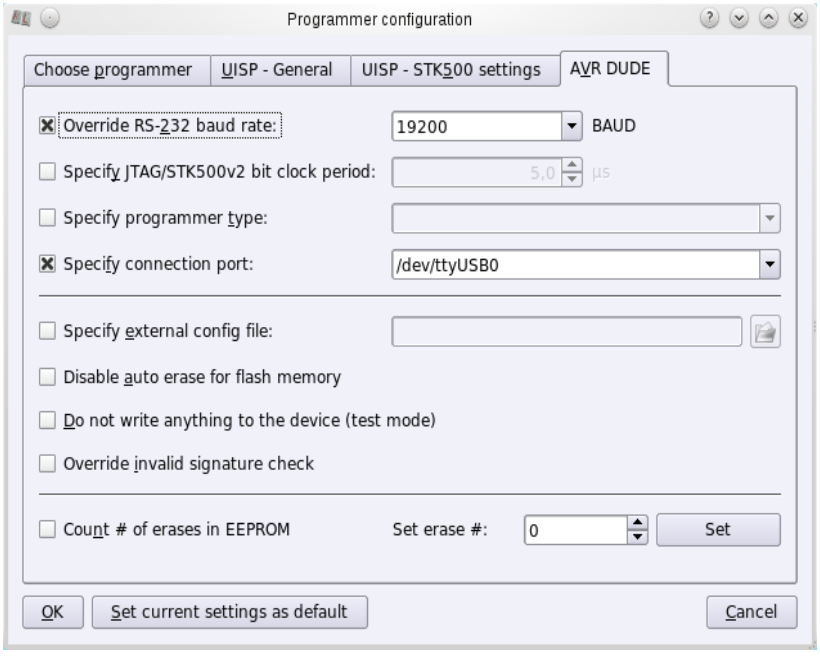


Рис. 50. Окончательная настройка программатора в ALTLinux

Хотя kontrollerlab имеет отладчик, но с ним у меня отношения не сложились. Конечно, можно

было бы выяснить причину этого, но средства отладки модуля Arduino достаточно обширны и без поиска причин неудачи с отладчиком этой программы.

Почти все среды разработки на базе микроконтроллеров AVR, о которых рассказано выше, используют компилятор AVR-GCC, полнофункциональное СПО. Прочитать о компиляторе можно, например, на сайте:

http://electrons.psychogenic.com/modules/arms/art/3/AVRGCCProgrammingGuide.php

Можно загрузить руководство по использованию языка в формате pdf на сайте:

http://nettopdf.info/en/pdf/+AVR-GCC+Manual-.html

Оба описания на английском языке. Но, во-первых, основа – это язык Си, а книг на русском об этом языке программирования более чем предостаточно; во-вторых, если вы решили использовать язык Си, то как-то нужно приспосабливаться к английскому языку; и, наконец, если вы только начинаете работать с микроконтроллерами, с модулем Arduino, то пройдёт достаточно времени до того момента, когда вам потребуется программирование на языке Си, и вы успеете освоиться с английским языком.

Когда, работая над рассказом, я коснулся темы AVR Studio, я обнаружил, что появилась новая версия этой среды разработки. Чтобы скачать программу, требуется регистрация. Не вполне ясно, отчего обязательными являются пункты, которые относятся только к пользователям в США, но, как получилось, я заполнил бланк. Если возможность скачать программу реализуется, я постараюсь рассказать немного о новой версии. Если же нет, то на нет и суда нет…

Однако всё получилось. Пакет весит изрядно, но, будем надеяться, оно того стоило. Запускаем установку программы, которая начинается с установки необходимых компонентов Microsoft Visual Studio.

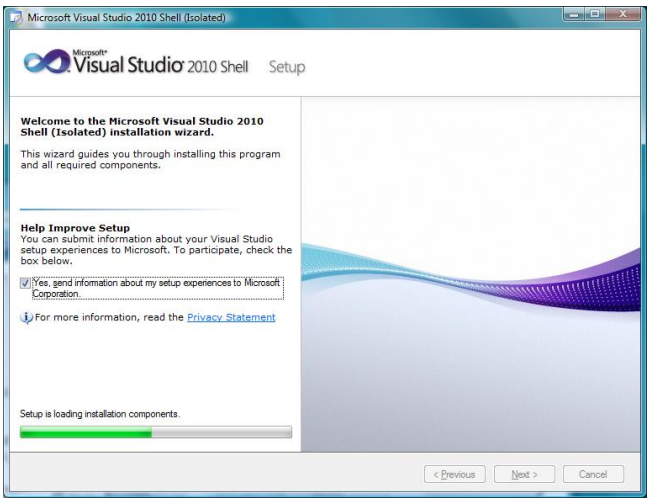


Рис. 51. Установка компонентов от Microsoft

По окончании этой части процесса начинается установка AVR Studio.

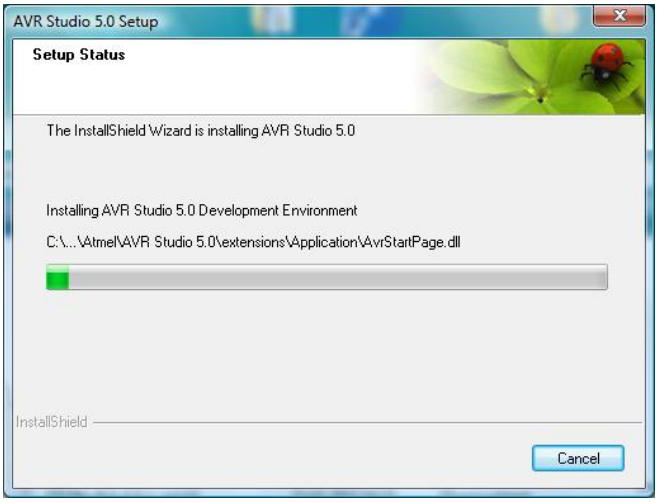


Рис. 52. Начало установки программы

Хотя время в таких ситуациях тянется невыносимо долго, но всему неприятному приходит конец.

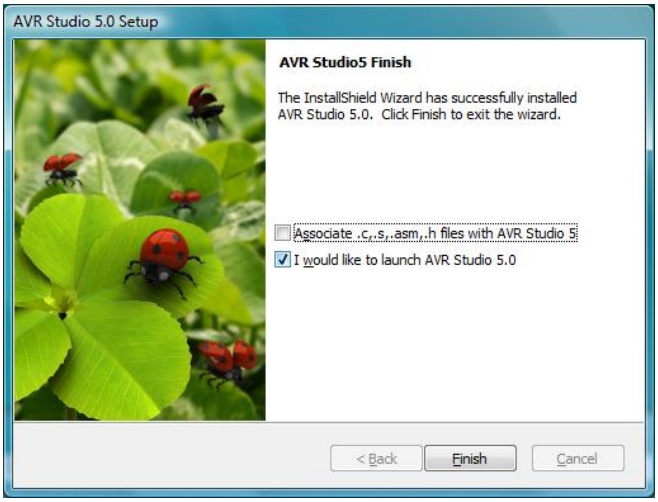


Рис. 53. Завершение установки AVR Studio 5

После того, как вы нажмёте кнопку «Finish», благодаря установленной опции, запускается сама программа.

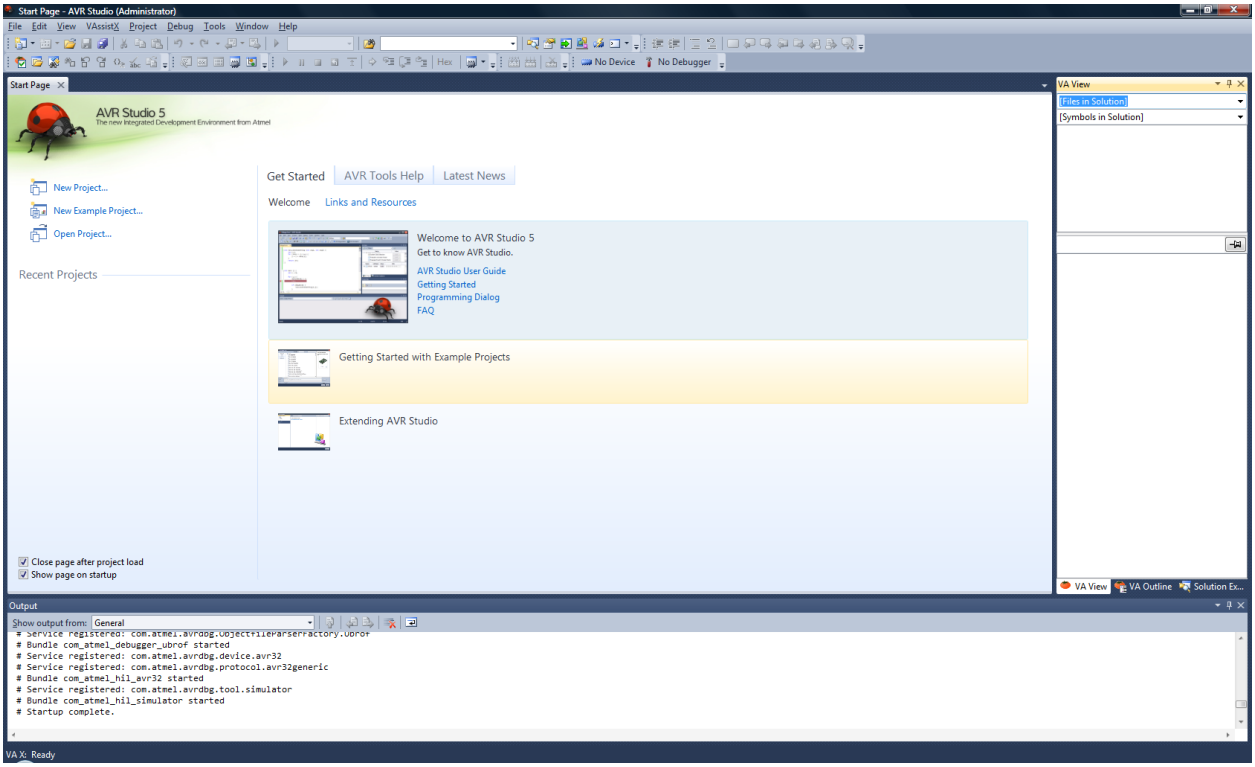


Рис. 54. Первый запуск программы

Собственно, это ещё не запуск программы, а диалог выбора: вы можете создать проект (New Project), использовать проект примера или открыть существующий проект (Open Project). Всё это вы выбираете слева. А на основных закладках можете выбрать знакомство с программой на закладке «Get Starting», ознакомиться с инструментарием для работы с AVR-контроллерами (AVR Tools Help) и подписаться на новости.

При выборе знакомства с программой следует учесть, что файл будет получен из Интернета, то есть компьютер должен быть подключён к сети.

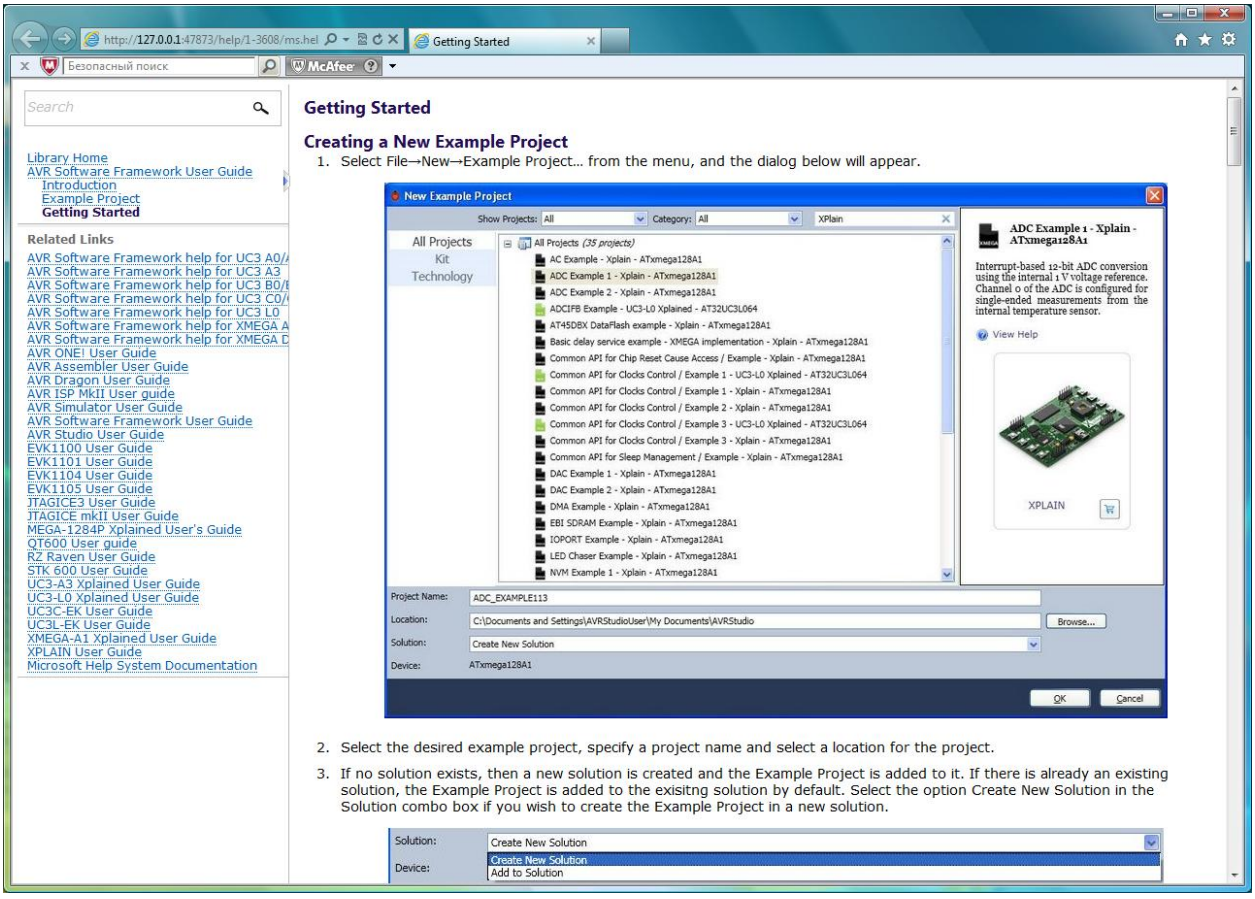


Рис. 55. Руководство по начальным шагам работы с программой

Выбрав новый проект, вы можете продолжить выбор создания проекта: я, например, выбираю пустой проект для GCC компилятора. Там же выбираю имя проекта, а предлагаемое место хранения проекта оставляю без изменений. Далее есть возможность, об этом позаботился помощник создания нового проекта, выбрать модель микроконтроллера.

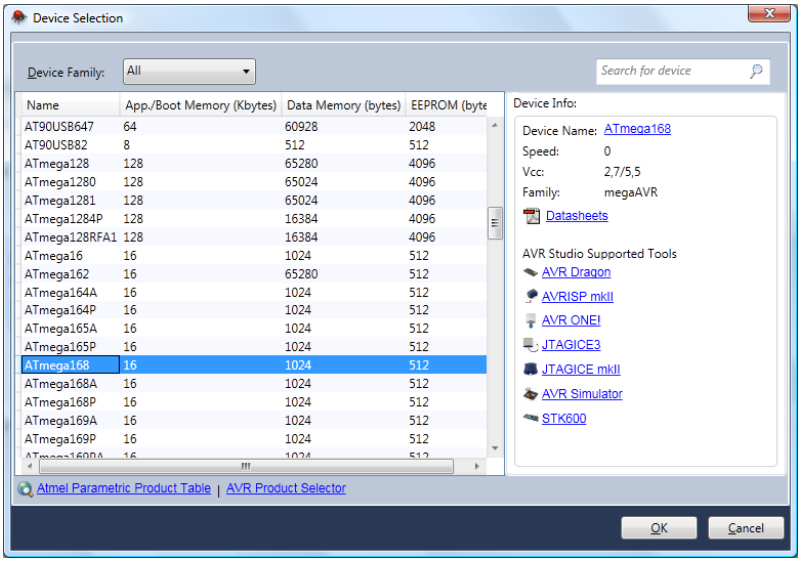


Рис. 56. Выбор модели контроллера

Если вам лень продвигаться по списку вручную, можете ввести модель в окно поиска. Выбор завершён, вы получаете окно редактирования с готовым шаблоном, куда можете сразу скопировать текст вашей программы, если он, конечно, есть (у меня так).

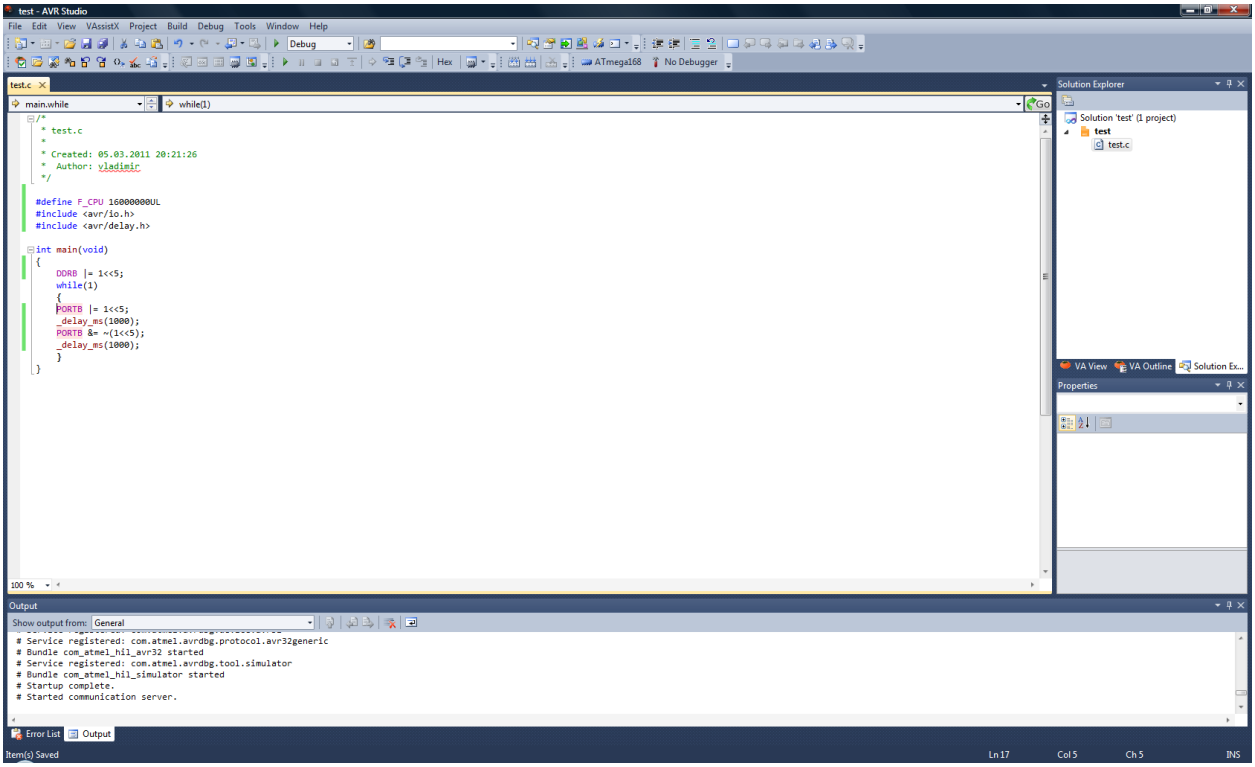


Рис. 57. Добавление текста программы копированием готового варианта

В окне меню вы можете выбрать режим работы: Debug для отладки, Release для готового проекта. В любом случае программа создаст необходимые папки для работы с файлами. Следует проверить свойства проекта, в разделе «Project» основго меню есть команда «test Properties». На первой закладке вы устанавливаете опции файлов, которые хотите получить в результате трансляции. А на последней закладке выбираете отладчик. Нажав кнопку подтверждения выбора, сохраните всё, используя, например, иконку на инструментальной панели с множеством дискет. Можно запустить трансляцию (Build-Build <имя вашей программы>). Если в программе есть ошибки – вы могли не добавить включение нужного файла заголовков – вы получите сообщение в окне выхода в нижней части экрана. Это полезно, когда вам нужно выяснить причины неудач.

Новая версия стала мощнее, интереснее, несомненно, но меня более всего устраивает то, что теперь я могу использовать её непосредственно с модулем Arduino.

Заходим в раздел «Tools» основного меню и открываем «External Tools…». Предварительно скопировав два файла avrdude (исполняемый и конфигурации) в папку с документами, где программа создаёт новую папку «AVRStudio», я прописываю в окне диалога название программатора, место, где находится исполняемый файл программатора и те аргументы, которые будут переданы в программу avrdude.

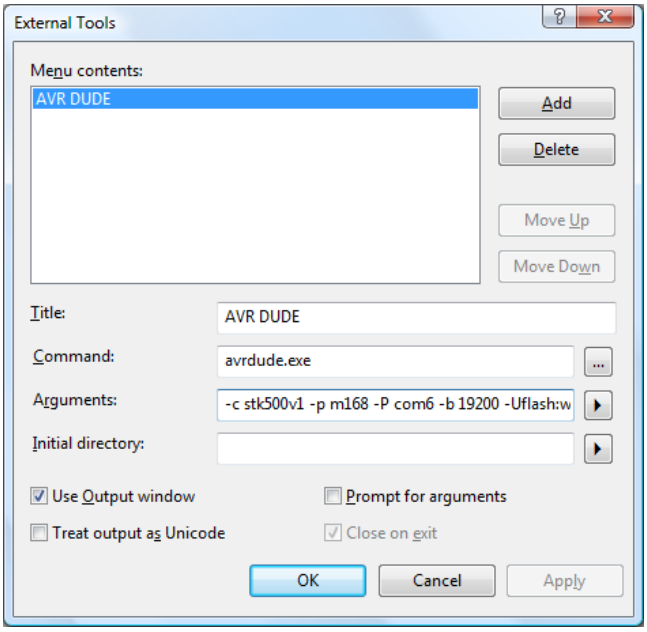


Рис. 58. Создание нового программатора для AVR Studio 5

Не без ложки дёгтя. Возможно, есть элегантное решение вопроса, но не при первом запуске. Вот эта ложка:

```
-c stk500v1 -p m168 -P com6 -b 19200 -
Uflash:w:C:\Users\vladimir\Documents\AVRStudio\test\test\Debug\test.hex:i -C
C:\Users\vladimir\Documents\AVRStudio\avrdude.conf
```

Приходится прописывать полный путь и имя проекта. Пока работаешь с одним проектом, это не так страшно – один раз прописать всё, и пользуйся на здоровье. Но при смене проекта эту строку нужно будет не забывать править. Впрочем, возможно, есть и «правильные» решения. Но теперь у вас появляется новый инструмент:

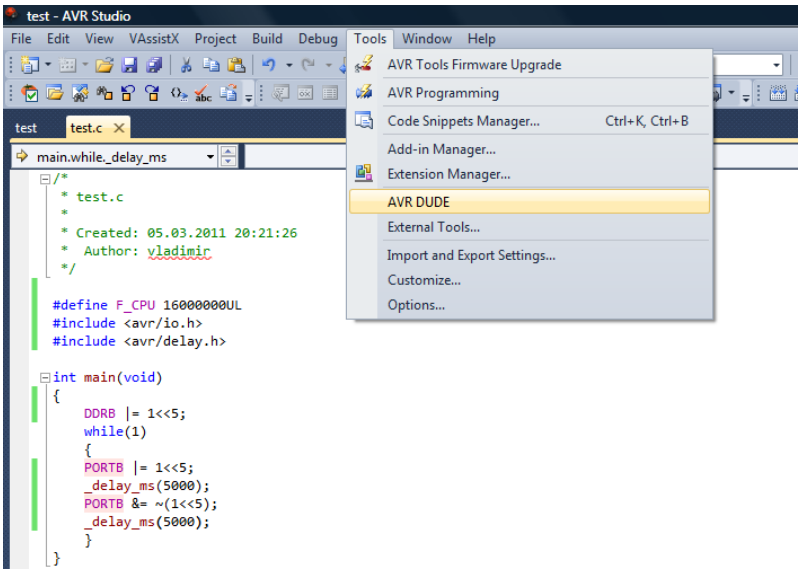


Рис. 59. Появление нового инструментального средства

Оттранслировав программу (обратите внимание на предыдущий рисунок, где установлена опция «Use Output window») вы используете эту команду для прошивки модуля. Вывод всех сообщений в окно, определяемое опцией, удобен – вы сразу можете прочитать, какие проблемы возникли при загрузке программы в модуль. И попытаться разобраться с ними, как это сделал я, поскольку далеко не сразу получил желаемый результат.

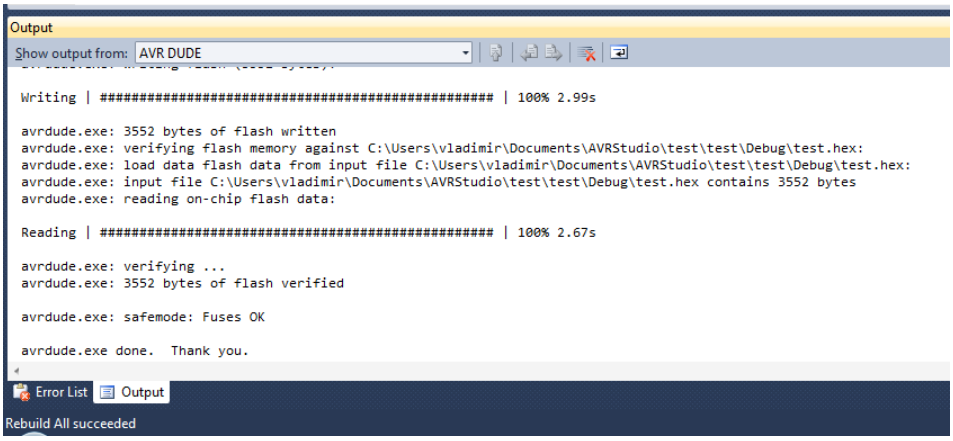


Рис. 60. Загрузка программы в модуль Arduino

В отличие от kontrollerlab отладка в AVR Studio устроена лучше. Что важно, согласитесь.

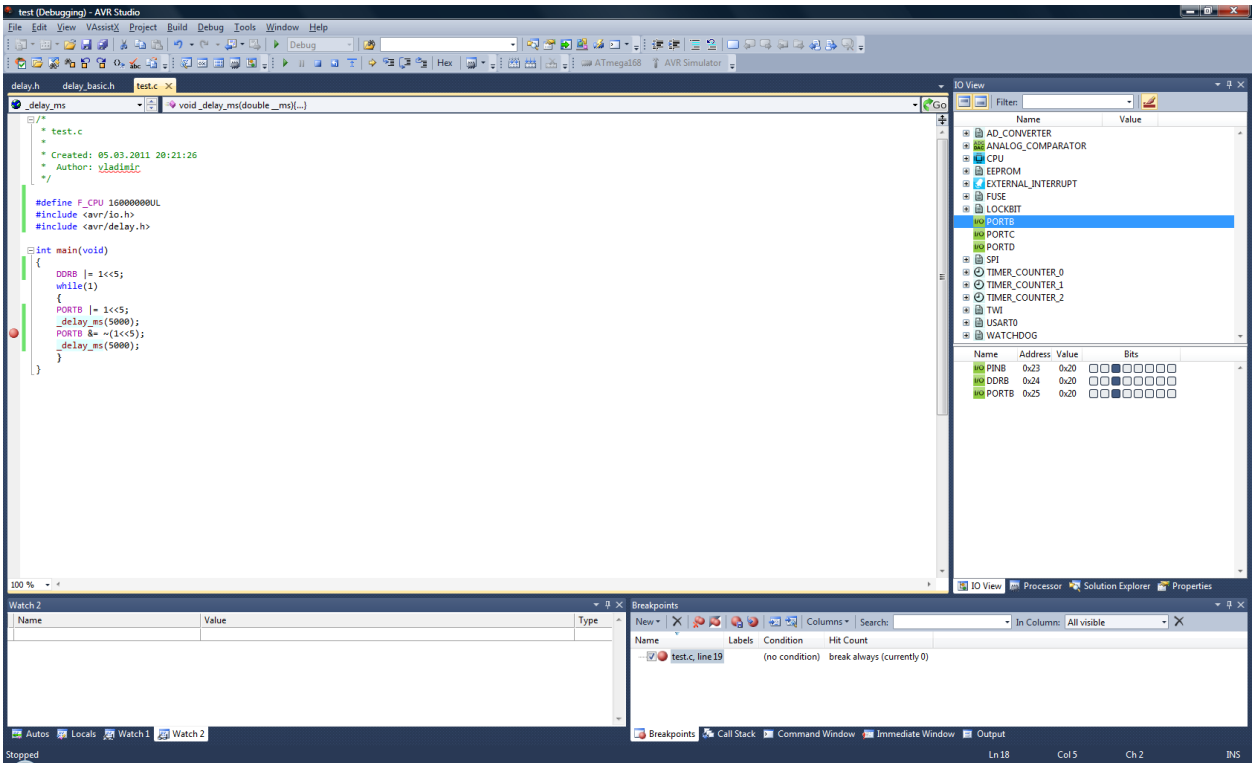


Рис. 61. Программа AVR Studio 5 в режиме отладки