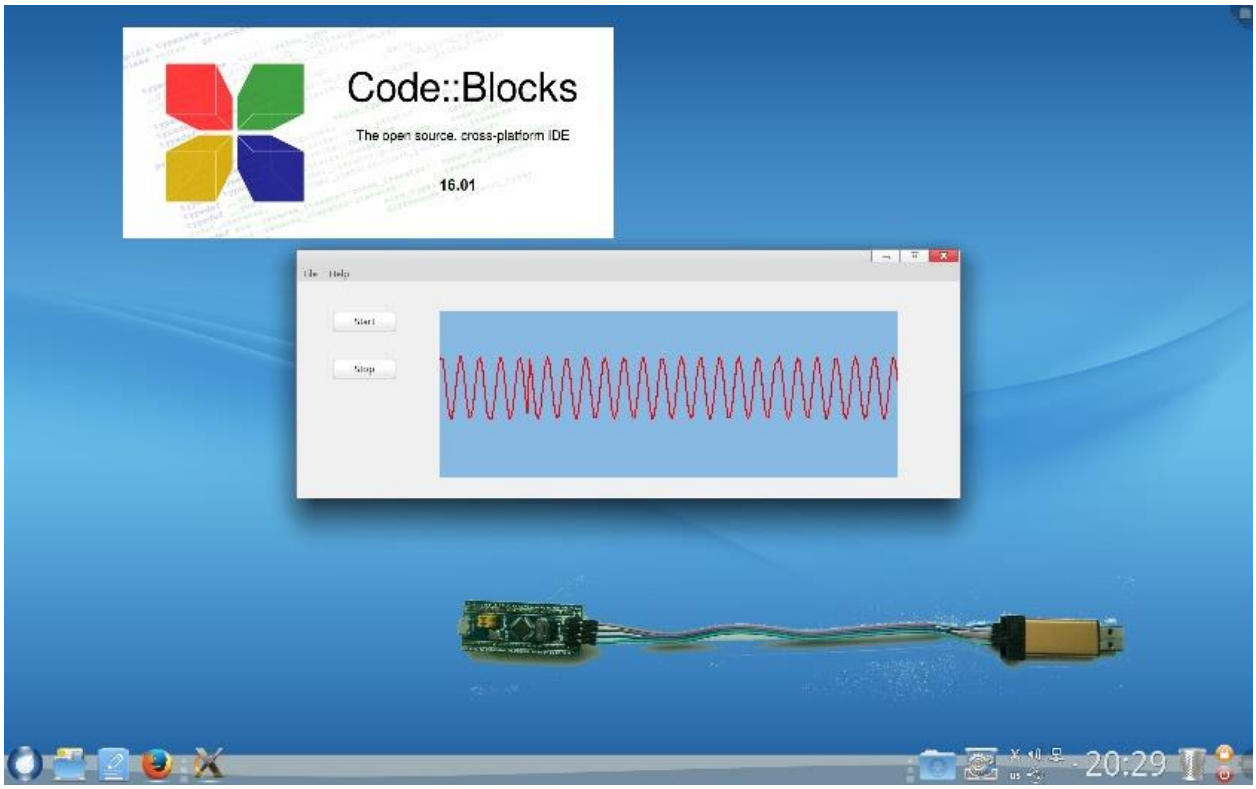
В.Н. Гололобов

STM32F103C8 & LINUX ROSA



Москва - 2017

Оглавление

Операционная система Linux понравилась мне с момента первого с ней знакомства. Если не ошибаюсь, это был дистрибутив ALTLinux. В те времена самым неприятным было то, что не получалось подключение к провайдеру для выхода в Интернет. Позже, используя ASPLinux, я научился, вначале криво-косо, но затем гораздо лучше выходить в Интернет.

Было время, когда все дистрибутивы Linux дружили между собой и с Windows. В то время у меня на компьютере было несколько разных дистрибутивов. Было интересно их сравнивать, было полезно решать какие-то проблемы, устранение которых с чьей-то лёгкой руки стали называть «танцами с бубном».

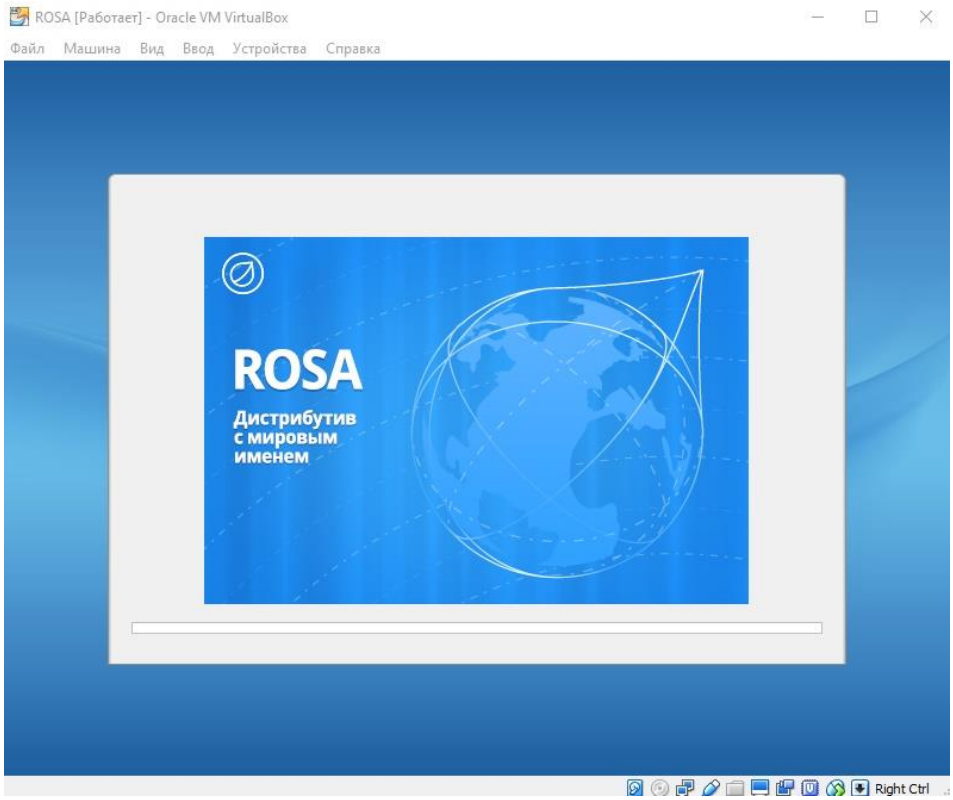
Но затем при установке новой версии дистрибутива Linux, а, к сожалению, не всегда получалось обновление старой версии, приходилось вручную «прописывать» уже установленные дистрибутивы. Более того, настало время, когда для обновления Windows приходилось восстанавливать MBR, без чего обновление не устанавливалось. А последнее, что меня неприятно поразило – это то, что при очередной попытке обновить дистрибутив Fedora, я столкнулся с необходимостью заниматься перестройкой жёсткого диска. Занятие, скажем, на любителя. С тех пор я не устанавливал Linux на свой компьютер. Была и ещё одна причина – в какой-то момент государство решило перевести школы на операционную систему Linux. В этом случае никакой рассказ про Linux не был бы лишним. Но, как и многие решения, это быстро завяло, и сейчас даже трудно сказать, как обстоят дела с этим, например, в школах.

Вместе с тем, может быть, впрочем, и не вместе, меня интересует то, как обстоят дела с этой операционной системой, интересует, насколько можно делать то, чем я сейчас занимаюсь, в Linux. Добавил к этому желанию и тот факт, что американские секретные службы не желают менять сервера на новые – пока они отсканируют твой компьютер после его включения, никакого терпения не хватит…

Без желания устанавливать на компьютер, но с желанием «попробовать», я решил использовать виртуальную машину, на которой установил дистрибутив Linux. Вначале, правда, не сложилось что-то со старой версией VirtualBox от Oracle, но замена её на новую версию решила это недоразумение. Для установки я выбрал отечественный дистрибутив ROSA. Создавая операционную систему на виртуальной машине для ROSA, следует выбрать 1.4 Гбайт оперативной памяти (или больше, если есть возможность) и 20 Гбайт (или больше, если есть возможность) для виртуального жёсткого диска.

Как и любая операционная система, ROSA устанавливается довольно долго. После первых настроек, где выбирается ряд параметров, виртуальная машина осуществляет установку программного обеспечения. Я не пробовал установку этой операционной системы на реальный раздел жёсткого диска, не исключаю, что в таком случае установка может выглядеть несколько иначе, но на моём компьютере с той версией VirtualBox, что я использую, картинка на экране долго не меняется, и не видно изменений линии прогресса.

Как и при установке любой операционной системы, следует набраться терпения, дождавшись, когда появятся последние настройки, появится предложение удалить установочный диск, но его удалять в этот момент не следует, но следует дождаться, когда дисковод откроется и можно будет удалить диск.



Установка ОС ROSA на диск виртуального компьютера

1 Начало установки программ и настройки

Я не готов самостоятельно решить задачу, поэтому пользуюсь советами «бывалых»:

http://www.instructables.com/id/Build-a-Program-for-STM32-MCU-Under-Linux/

При этом я не исключаю, что эти советы есть и на сайтах производителей.

Всё, вероятно, было бы проще, если бы мне удалось скачать программу System Workbench для Linux. Но я использовал скачивание для Windows, а позже не смог это сделать: я пытался использовать логин и пароль, я пытался ещё раз заполнить анкету, но безуспешно.

С другой стороны, и это так, я не уверен, что программа установилась бы, это раз; я не уверен, что программа заработала бы, это два. Немного не разобравшись, я скачал и стал устанавливать программу TrueStudio. Когда результат стал отрицательным, я снёс дистрибутив ROSA, установил дистрибутив Fedora 25, затем повторил установку программы в Ubuntu, где программа работает по сведениям производителя. После этих установок, набравшись опыта, я установил программу в ROSA, но лишь для того, чтобы обнаружить — я могу её использовать бесплатно очень недолго. Такова цена ошибок!

Программа SW4STM32 работает на основе eclipse, но настройки для этой среды разработки я не нашёл, поэтому мой выбор пал на программу Code::Blocks, она есть в дистрибутиве ROSA. Я не уверен, что полностью и точно опишу весь процесс, он занял много времени. Я уже упоминал, что пришлось несколько раз устанавливать и переустанавливать операционные системы, что само по себе отнимает много времени. Остаток времени отнимали загадки, возникающие по мере

продвижения к цели. Как человек увлекающийся, я «не записывал все ходы», что-то могу и пропустить.

> *Это так. Позже я решил повторить установку, что потребовало переустановки операционной системы. Читая написанное ниже, я старался повторять установку, но оказалось, что запомнил я далеко не всё. Вдобавок, я этого не ожидал, во второй раз пришлось делать что-то, что я не делал при первой установке. Всё это я решил записать в виде Добавления к следующей главе.*

Итак. Для работы в среде разработки для STM32 потребуется установить, в дистрибутиве ROSA это не сделано по умолчанию при установке из LiveDVD, пакет Java. После этого можно скачать и установить программу STM32CubeMX. Программа устанавливается без особых трудностей. Скачав и распаковав (я всё, включая установку, делаю в домашней папке) программу, конечно, в версии для Linux, достаточно поправить установочный файл. Для этого следует открыть его свойства...

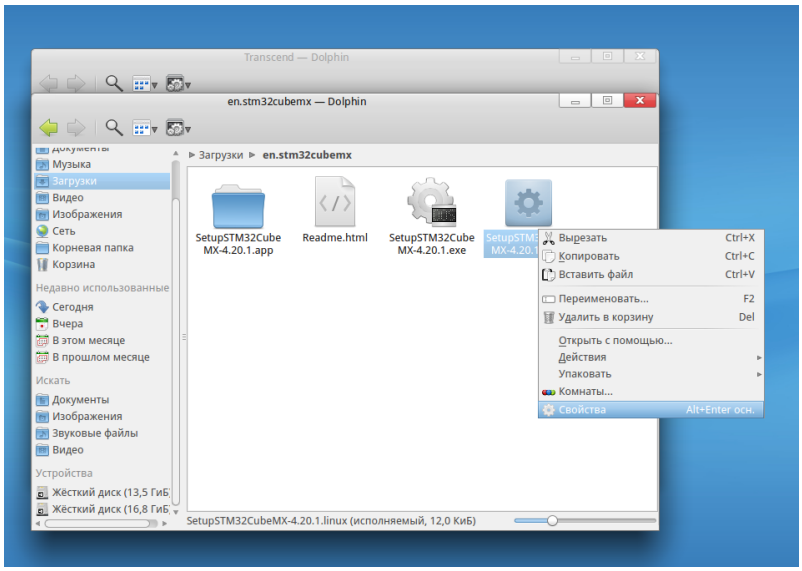


Рис. 1.1. Установочный файл STM32CubeMx для Linux

...где на закладке *Права* установить флажок, который сделает файл исполняемым.

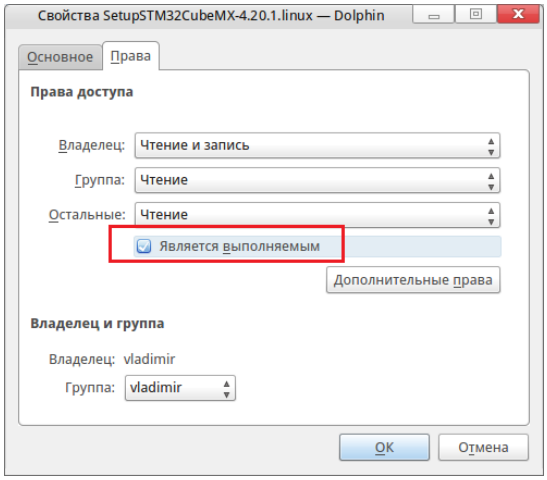


Рис. 1.2. Выполняемый файл установки

Теперь файл можно запустить на установку:

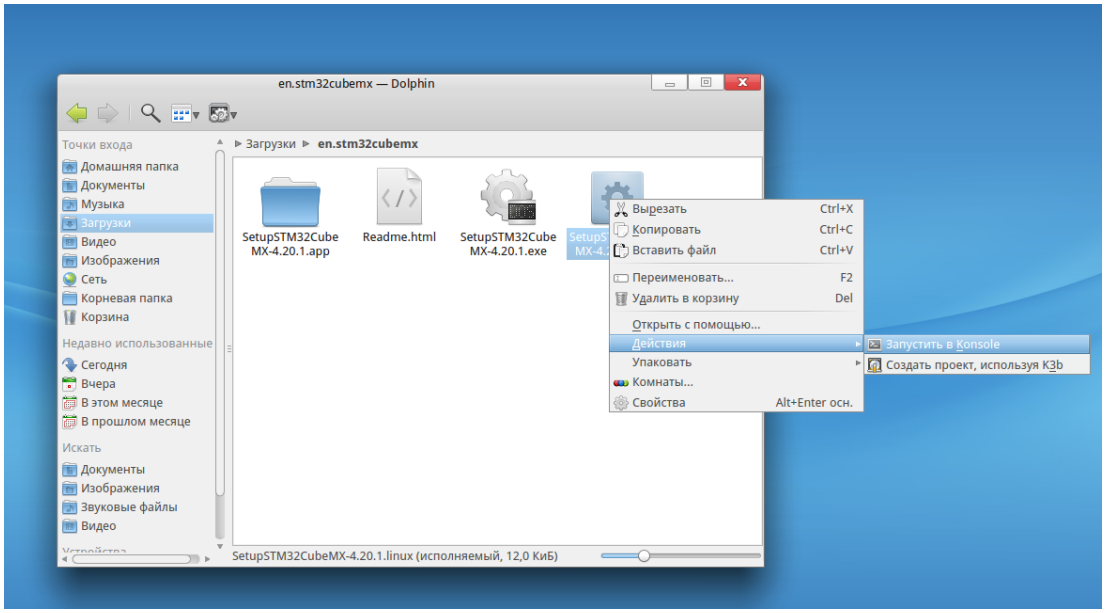


Рис. 1.3. Запуск программы STM32CubeMx на установку

Если всё необходимое из языка Java для работы программы есть, то программа будет запускаться и работать. У меня она в домашней папке расположилась так:

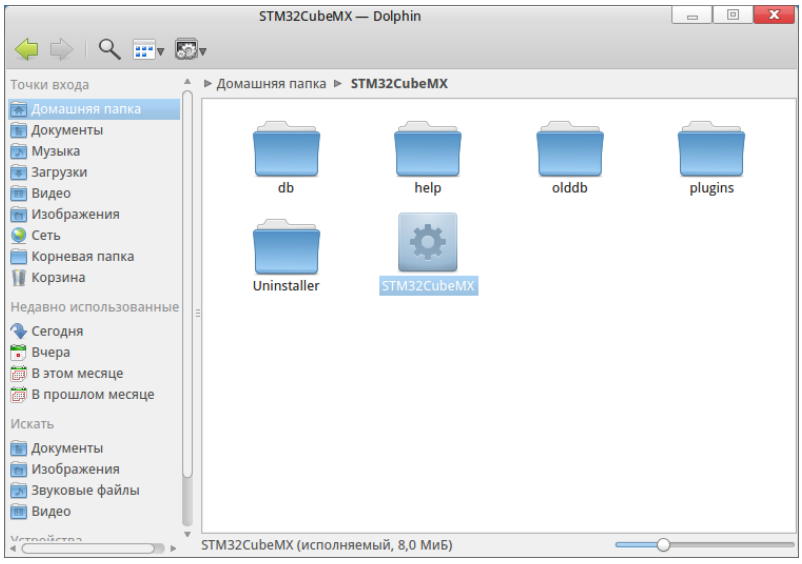


Рис. 1.4. Установленная программа CubeMx

Запустить её можно двойным щелчком левой клавиши мышки.

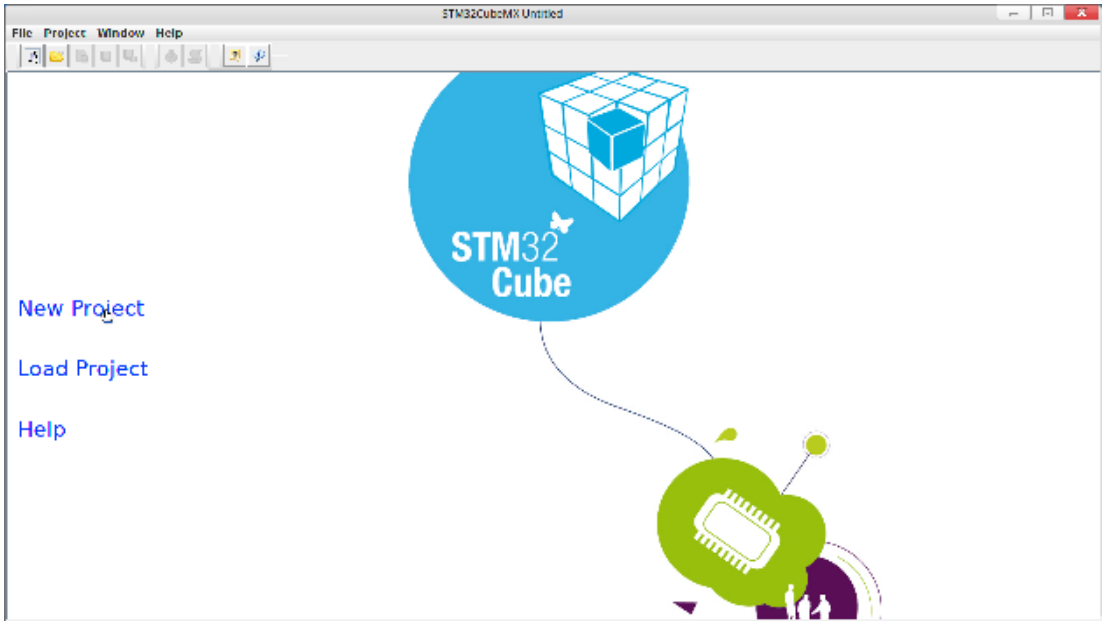


Рис. 1.5. Запуск программы STM32CubeMx

На рисунке видно, что курсор мышки (на строке NewProject) меняет свой вид, когда находится там, где можно начинать работу с программой. Щёлкнув по выбранному разделу, можно приступить к работе.

Создадим новый проект для простого мигания светодиодом, установленным на плате STM32F103C8. Впереди ещё много приключений, поэтому программа должна быть самой простой. А о том, как работать с программой, можно прочитать ещё и в моей книге «Небольшой рассказ о модуле STM32F103C8», и прочитать несколько позже в этом рассказе. Пока же создадим новый проект, настроим генератор (пункт *RCC*), настроим вывод PC13 на выход (щелчок левой клавишей по выводу и выбор *GPIO_Output* из выпадающего меню), зададим необходимые настройки, включая имя проекта, папку, где он будет находиться и формат выходного файла:

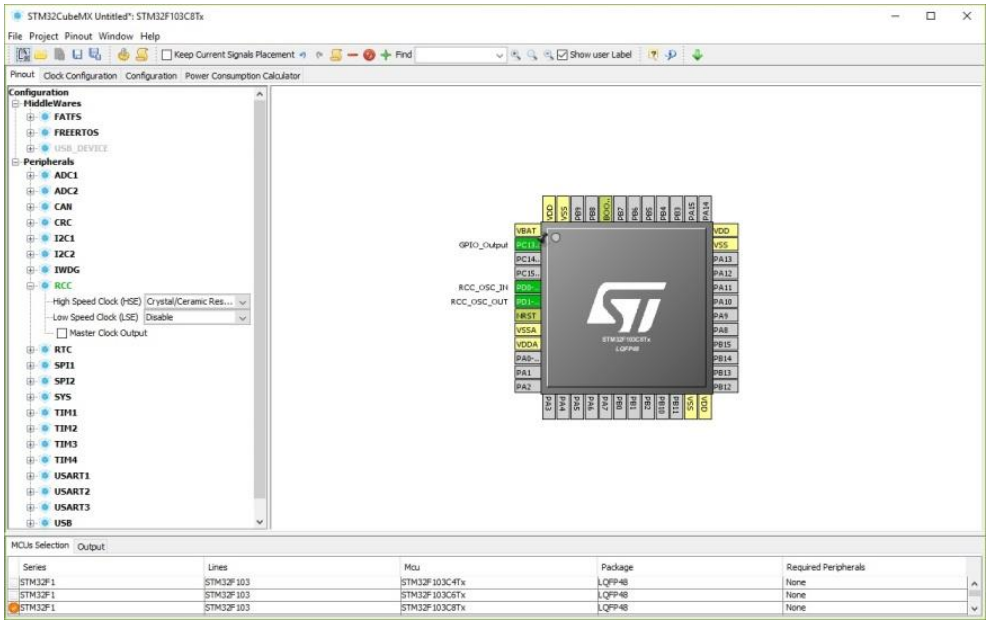


Рис. 1.6. Настройка проекта

Когда шаблон будущего проекта генерируется впервые, если я не ошибаюсь, будет предложено установить необходимые дополнения, с чем следует согласиться. И, следуя советам на указанном выше сайте, необходимо снять флажок, как обозначено на рисунке.

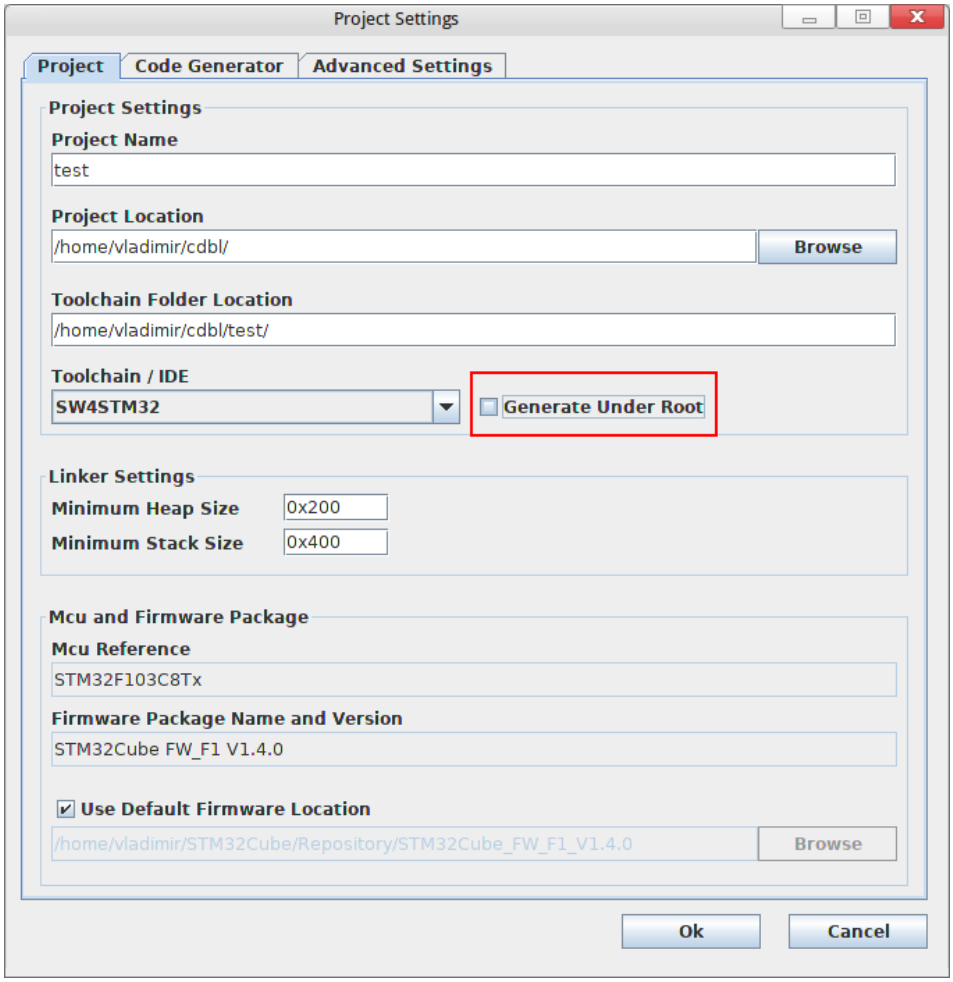


Рис. 1.7. Настройка шаблона проекта

Эта часть установки одна из самых простых. Поэтому я начал с неё, хотя автор руководства по работе с программами советует начать с установки компилятора, что более чем справедливо. Итак. Необходимо по сведениям автора установить три пакета:

- gcc-arm-none-eabi
- binutils-arm-none-eabi
- libnewlib-arm-none-eabi

Если попытаться установить эти пакеты с помощью штатного менеджера установки программ, то можно убедиться, что их для дистрибутива ROSA нет. Приходится искать компилятор на просторах Интернета. Предпочтительнее ввести в строку поиска полную верхнюю строку из рекомендованных пакетов с добавлением слова «скачать».

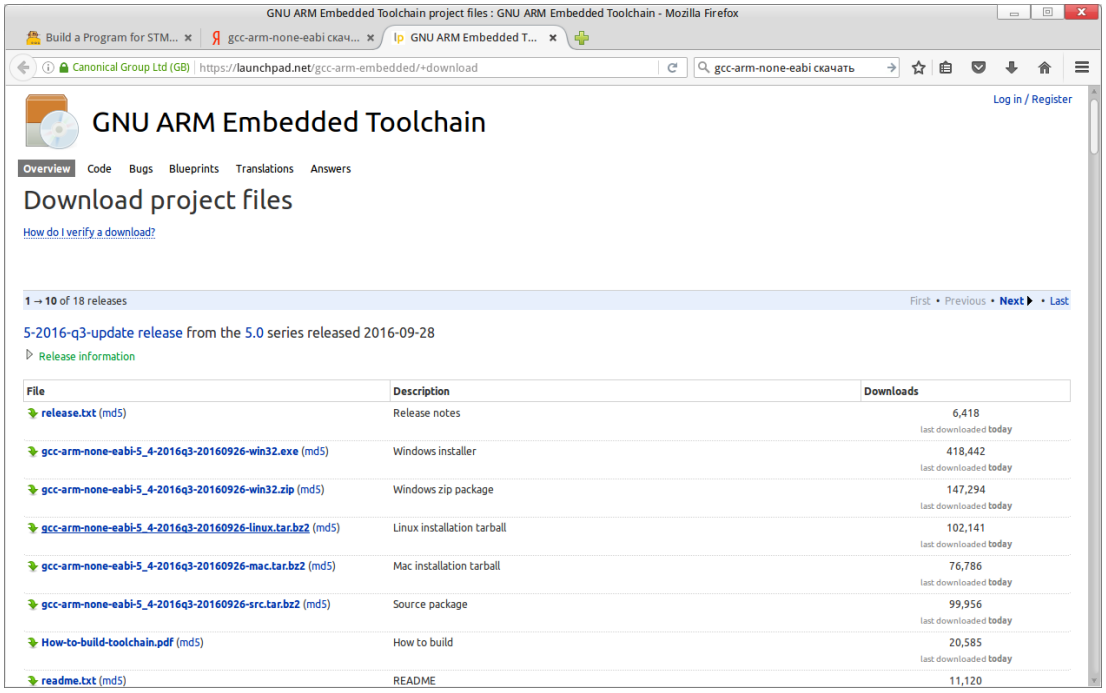


Рис. 1.8. Найденная возможность скачать пакет для arm-контроллера

Я не уверен, что поступил правильно, когда после распаковки пакета решил поместить его в раздел /usr файловой системы. К этому меня подвигло размещение там компилятора для микроконтроллеров AVR:

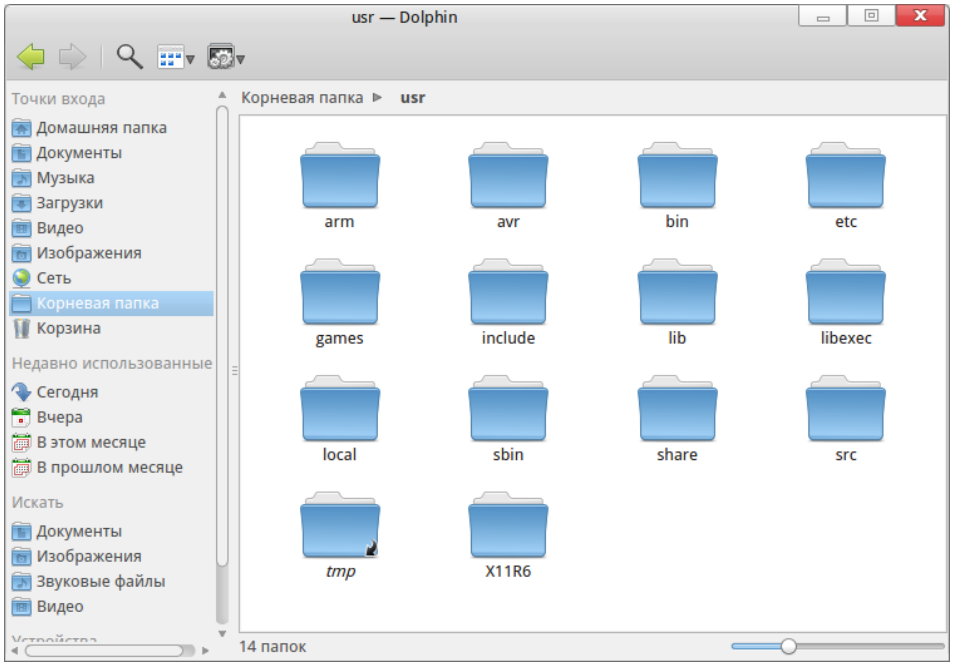


Рис. 1.9. Размещение компиляторов для ARM-контроллеров

И опять, если я не ошибаюсь, в скачанном пакете оказалось всё необходимое. Но…

В своей рабочей директории я создал папку, которую назвал arm. В неё я скопировал содержимое пакета gcc-arm…, и возникла необходимость переместить папку в нужное мне место. Как бы ни так. Современные операционные системы очень сильно защищены от пользователя. В данном

случае я не смог запустить файловый менеджер от имени главного администратора, чтобы выполнить задуманное. Пришлось из консоли выполнять команду *sudo mv* с указанием места назначения и имени папки. Эта проблема будет повторяться, поэтому запаситесь закладкой на страницу в вашем Интернет-браузере с консольными командами Linux.

2 Продолжение установок и настроек

Следующей по порядку будет, наверное, установка программы Makefile4CubeMX, которую следует скачать и установить так, как описано по следующим ссылкам:

github.com/duro80/Makefile4CubeMX https://github.com/duro80/Makefile4CubeMX.git

Программа устанавливается, кажется, при использовании ярлычка *Install*. При этом записывается один файл:

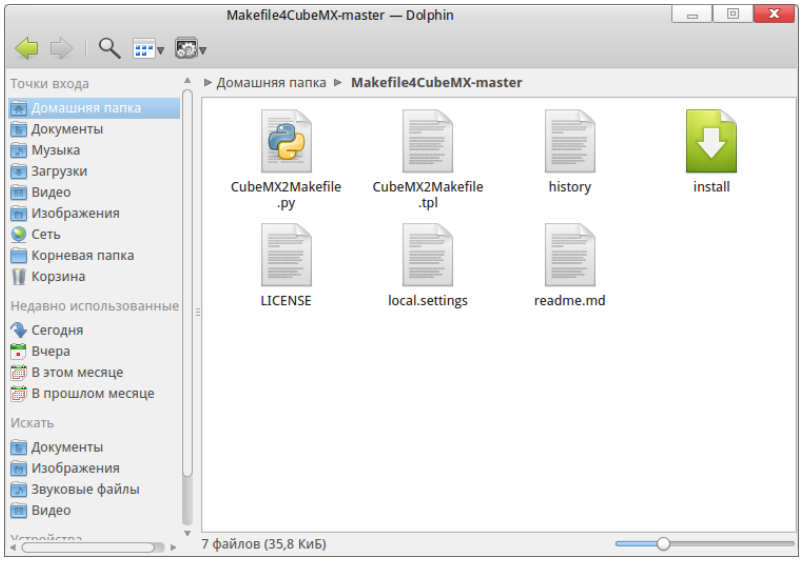


Рис. 2.1. Программа создания файла для Code::Blocks

Строка в консоли позволяет сделать программу доступной по имени:

```
sudo ln -s "$PWD"/CubeMX2Makefile.py /usr/bin/CubeMX2Makefile.py
```

Файл *CubeMX2Makefile.py* следует открыть в текстовом редакторе и подправить две строки:

```
localSetting["GCC_PATH"]="/usr/arm/bin/"
localSetting["FLASH_PATH"]="/usr/local/bin/st-flash"
```

Первая строка подтверждает тот факт, что в это место была перенесена папка компилятора. А вторая строка требует того, чтобы по этому «адресу» был расположен файл, о котором чуть ниже.

Если с этим всё в порядке, то остаётся настроить программатор. Но перед этим хотелось бы создать простую программу и проверить, всё ли правильно (а мне кажется, что есть подводный камень).

Запустим программу STM32CubeMx, настроим её так, как написано выше, получим файл в формате SW4STM32. Я сохраню его в своей домашней папке в подпапке *cdbl* с именем *test*, поскольку не знаю, чем всё закончится.

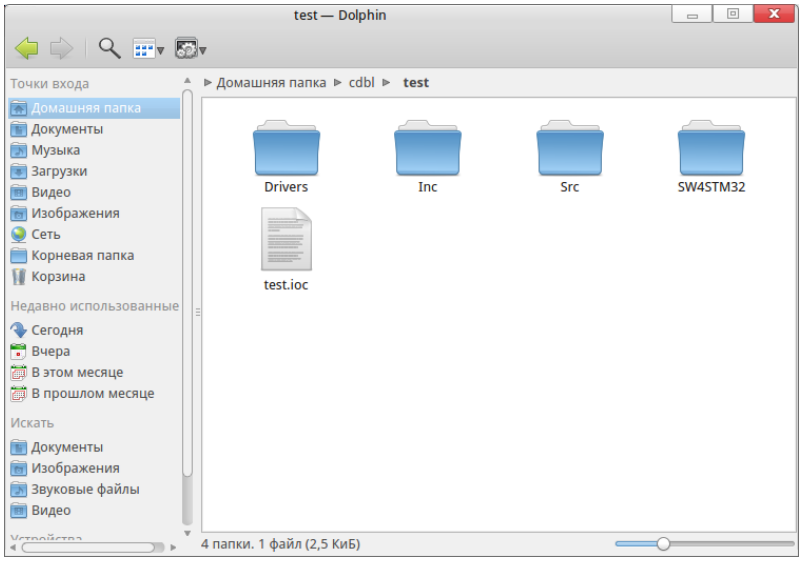


Рис. 2.2. Папка с заготовкой проекта

Теперь открываем консоль (если нет консоли в перечне приложений, то устанавливаем её штатным образом), переходим в папку проекта и запускаем файл *CubeMX2Makefile.py*:

Результат должен выглядеть так:

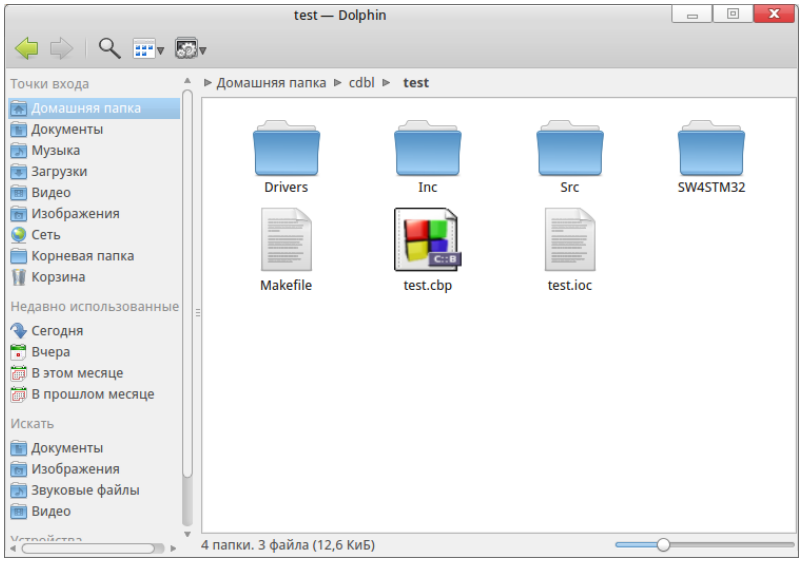


Рис. 2.3. Папка с проектом после использования предыдущей команды сборки

При этом заметьте, что в консоли команда *CubeMX2Makefile.py* должна завершаться пробелом и точкой:

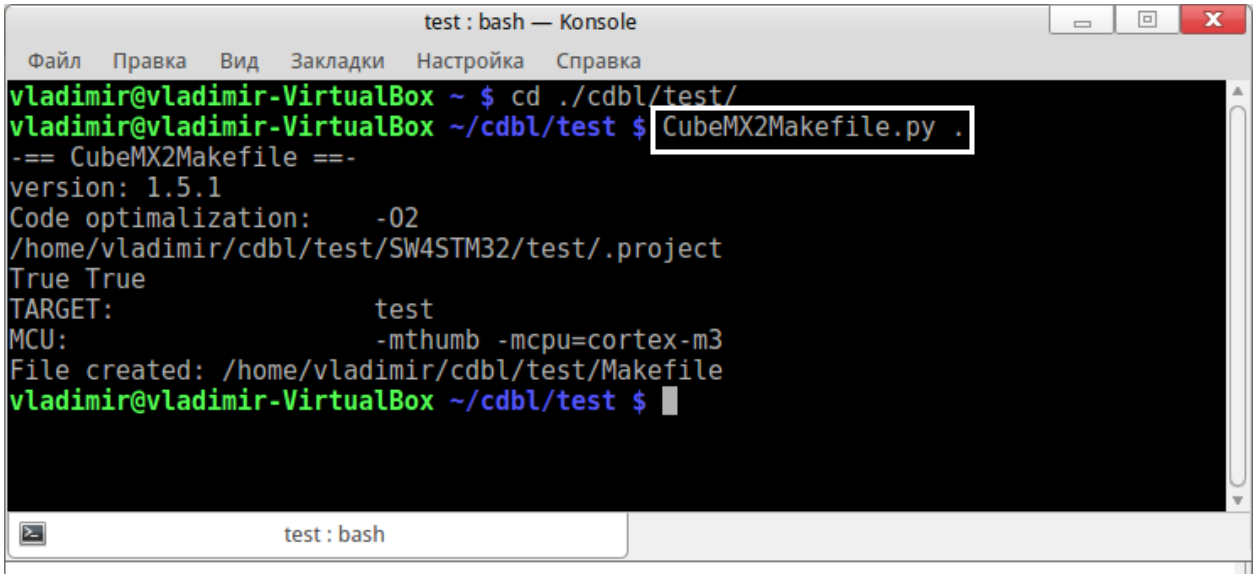


Рис. 2.4. Команда сборки в консоли

Я сейчас не вспомню, в этот ли раз или при установке ST-Link, или в другом месте потребовалось установить штатные пакеты *gtk*, без которых работа не продвигалась.

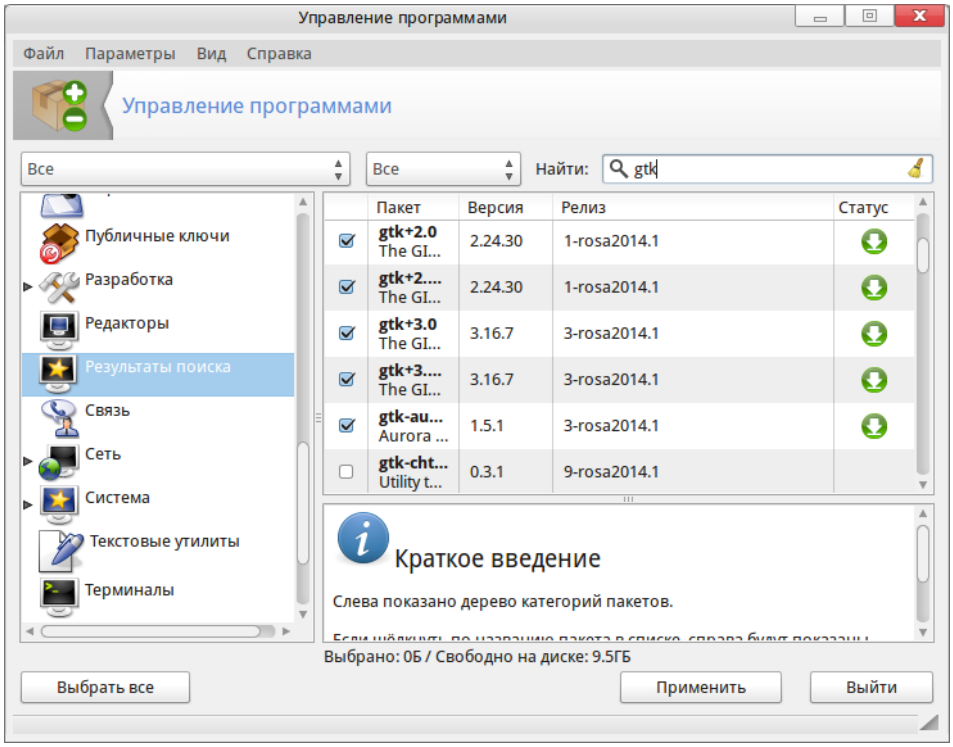


Рис. 2.5. Пакеты gtk в менеджере установки программа

Я не скажу, какие пакеты точно нужны, но можно их устанавливать до тех пор, пока не получится нужный результат.

А вот теперь можно установить (если вы не сделали этого раньше) среду разработки Code::Blocks, запустить программу и открыть проект, созданный ранее.

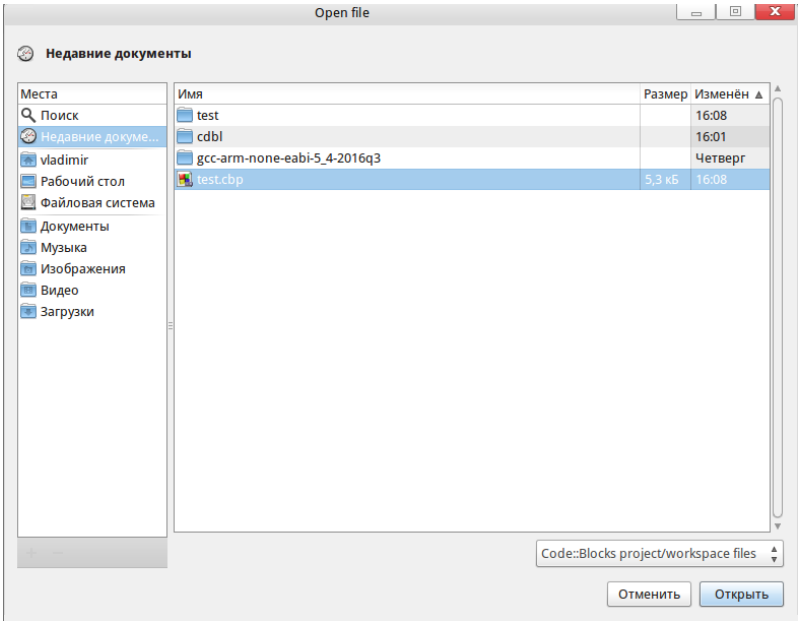


Рис. 2.6. Загрузка ранее созданного проекта в Code::Blocks

Но перед тем как продолжить работу, следует настроить компилятор, выбрав:

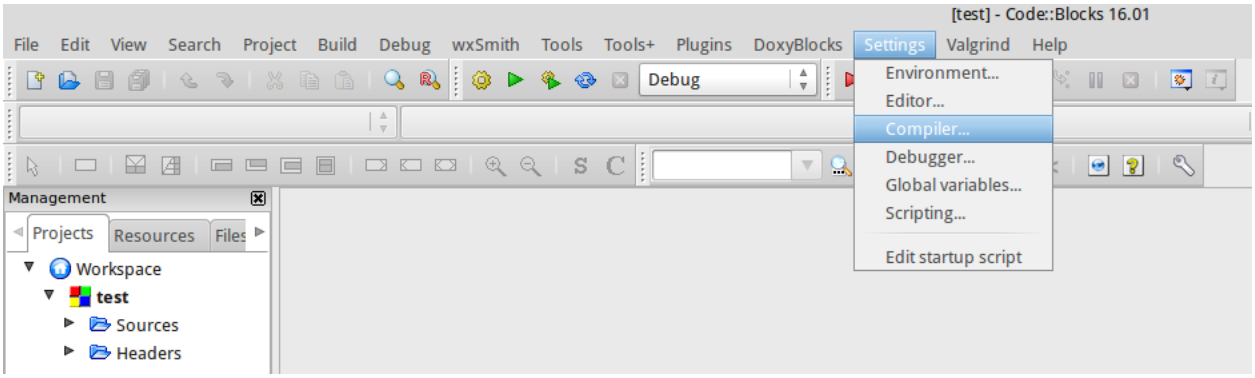


Рис. 2.7. Выбор настройки компилятора

В диалоговом окне следует указать все необходимые компоненты компилятора и сборщика, которые находятся там, куда мы переместили распакованный пакет компилятора:

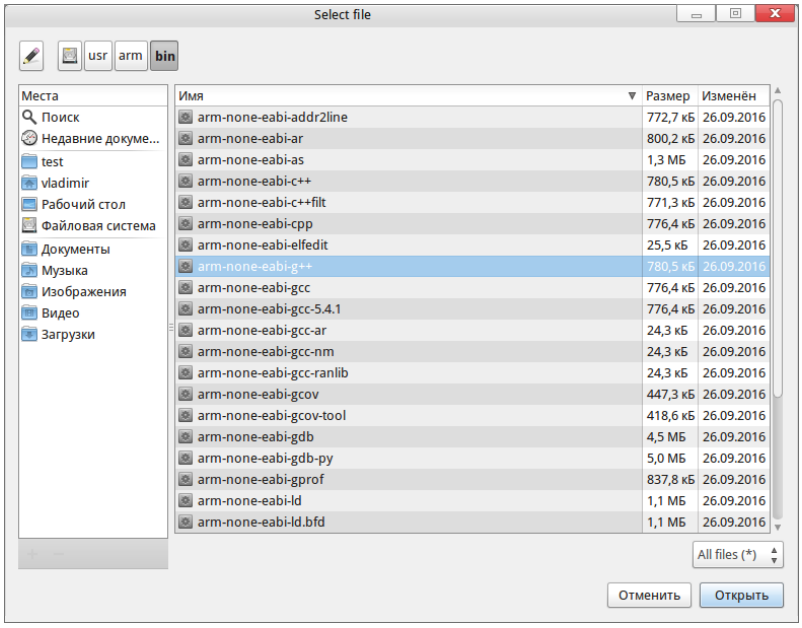


Рис. 2.8. Место расположения необходимых для настройки файлов

Вот так должна выглядеть настройка:

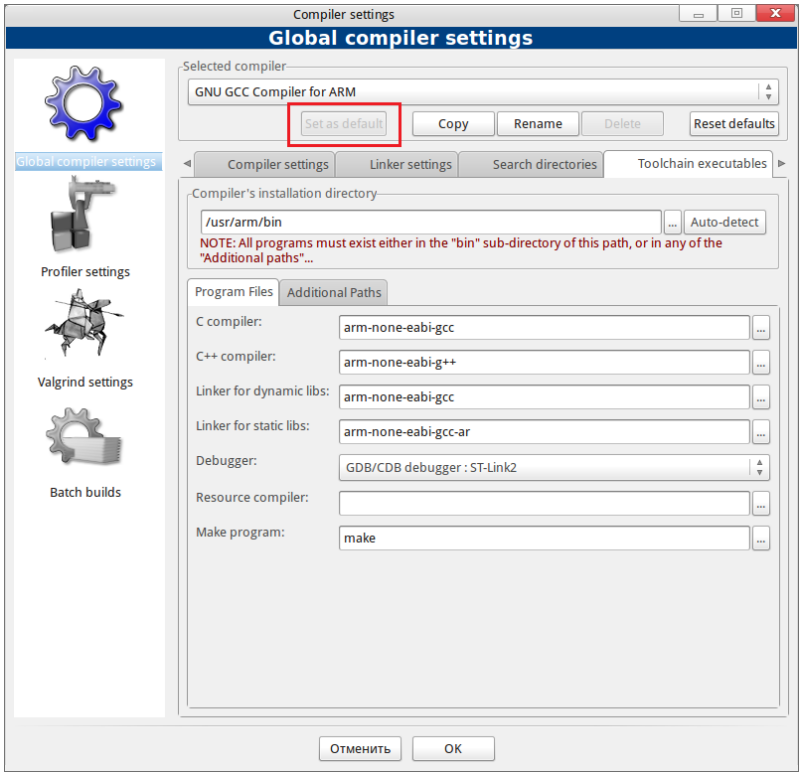


Рис. 2.9. Настройка компилятора в программе Code::Blocks

Настроив компилятор, можно в режиме отладки проверить его работу. Если всё сделано правильно, то сборка проекта осуществится без ошибок (и, возможно, без предупреждений).

А, открыв основной файл программы (main.c), можно добавить пару строк, чтобы проверить сборку программы (основное меню *Build→Build).*

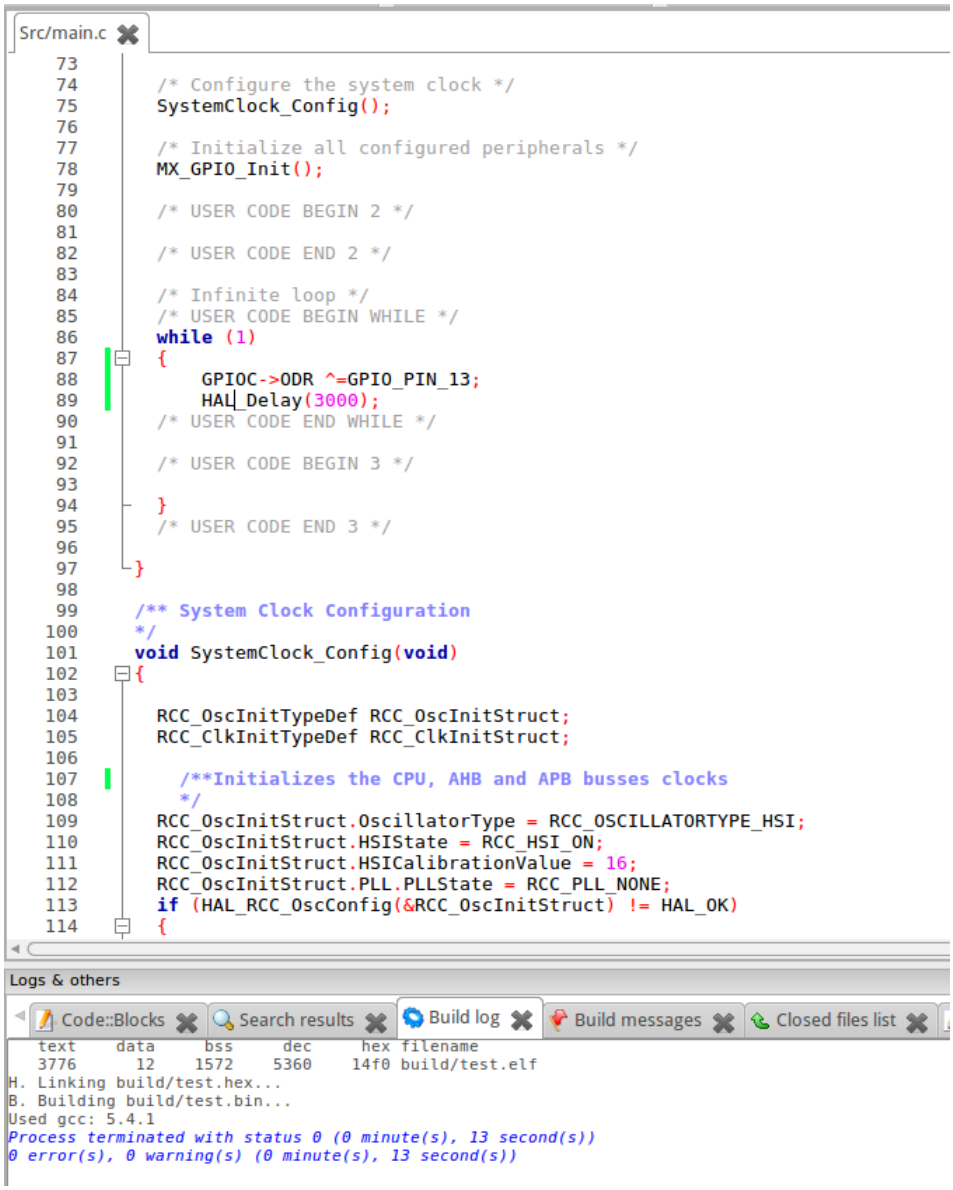


Рис. 2.10. Проверка работы компилятора (и компоновщика) программы

Убедившись, что компилятор и компоновщик работают, можно перейти к заключительной установке необходимой программы — пакета, который необходим для работы с программатором ST-Link.

Скачаем архивированный пакет либо здесь: github.com/texane/stlink, либо последнюю версию здесь: ithub.com/texane/stlink/releases.

Я пробовал установку stlink-master и stlink-1.3.1. Мне кажется, что здесь при установке в консоли этих пакетов потребовалось добавить пакеты *gtk*. После создания двух вариантов Release и Debug…

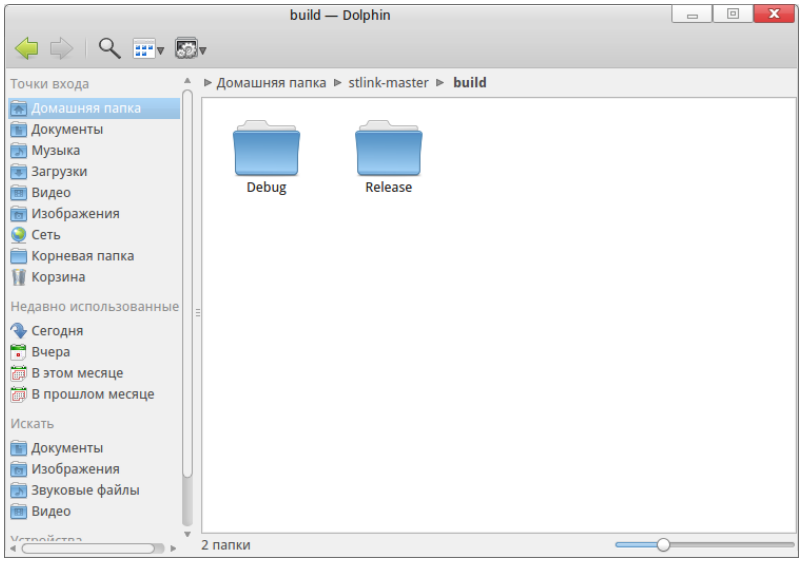


Рис. 2.11. Результат работы команд make release и make debug

При создании этих папок в окне консоли проходит достаточно долгий процесс, а если он быстро завершается, то что-то не так, чего-то не хватает для полной сборки.

И последнее, что потребовалось, и с чем я довольно долго провозился, это переместить файлы, отмеченные на рисунке ниже, в раздел /usr/lib, используя команду sudo cp путь_к_файлам /usr/lib:

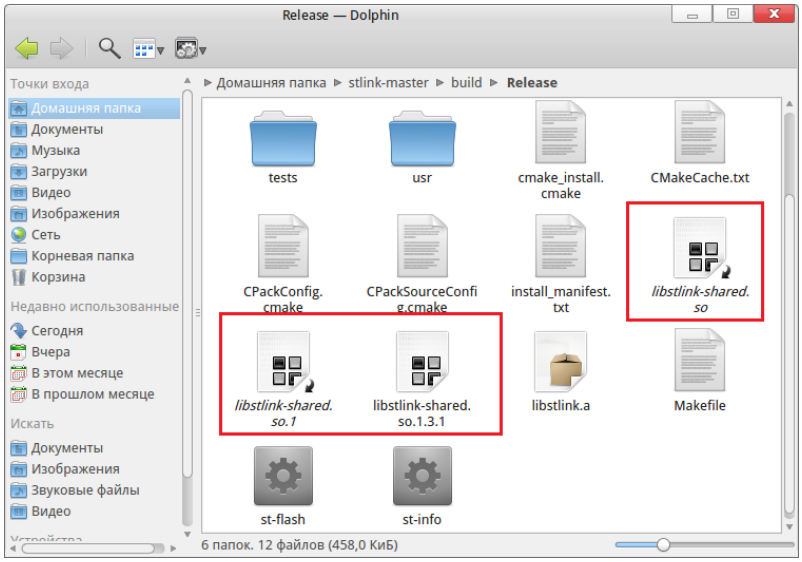


Рис. 2.12. Файлы, требующие перемещения

Без этого перемещения программа жалуется на то, что не может найти файл *libstlink-shared.so.1*.

После правильной установки этого пакета можно проверить полную работу Code::Blocks с загрузкой откомпилированной программы в микроконтроллер.

Это потребует подключения программатора к USB-порту и модуля к программатору через SW-интерфейс (а мне в виртуальной машине нужно подключить это устройство), выбора режима

сборки *Release* и сборки проекта. Если всё сделано правильно, проект скомпилируется и загрузится в модуль, а светодиод будет переключаться с частотой раз в 3 секунды.

Таким образом, в будущем получается удобная среда для разработки проекта, не требующая после получения удачного варианта работы использования дополнительной программы для загрузки кода в микроконтроллер.

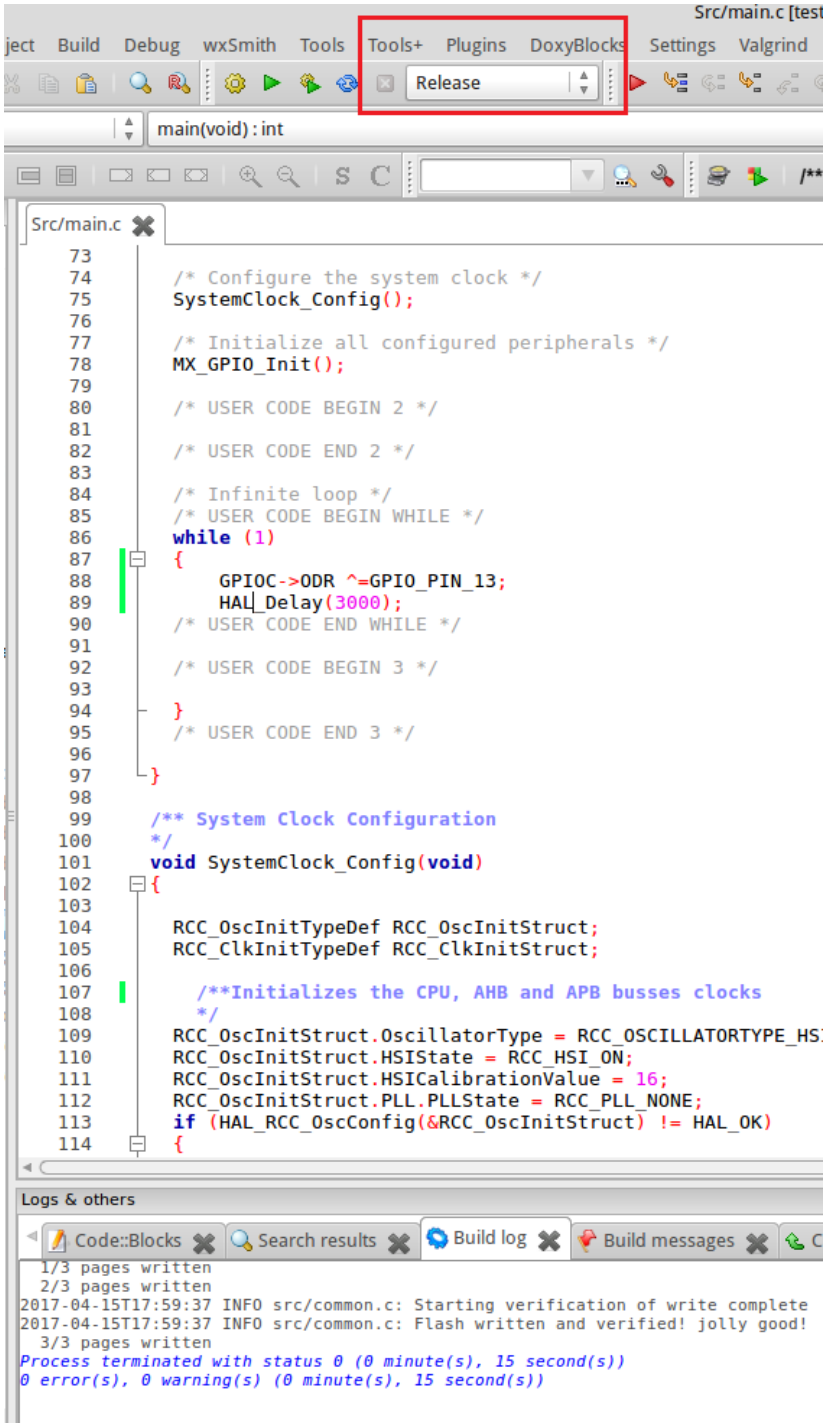


Рис. 2.13. Сборка проекта с загрузкой в модуль

Последнее, что хотелось бы проверить — это отладка с использованием ST-Link. В качестве штатного отладочного механизма используется пакет dbg. Если он не установлен при установке операционной системы, его следует установить с помощью штатного менеджера пакетов. Далее следует настроить отладку (возможно, я напишу лишнее в настройках, заранее прошу меня простить — я делаю это впервые). Во-первых, заходим в свойства проекта (*Project→Properties*), где на закладках *Debugger* должно быть следующее:

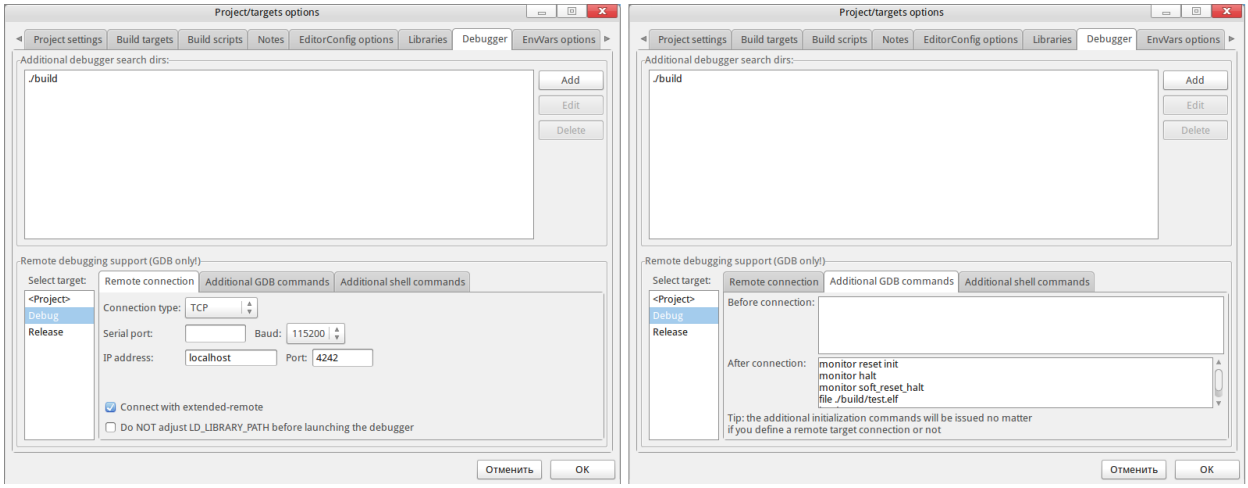


Рис. 2.14. Настройка свойств проекта

Добавляем в отладку наш программатор (*Settings→Debugger, GDB/CDB Debugger, кнопка Create Config*), назвав его *ST-Link2*. Затем, выделив его, указываем путь к отладчику:

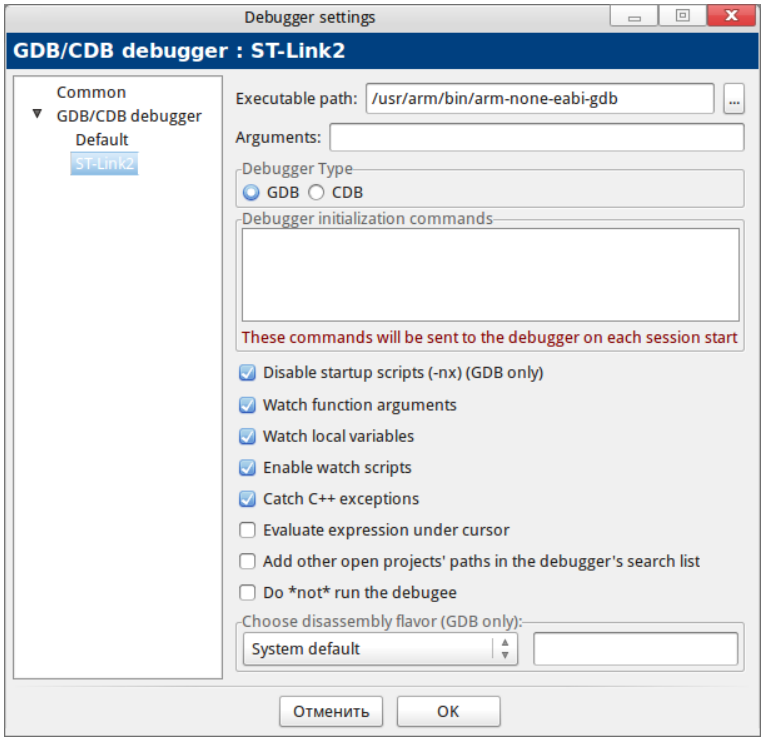


Рис. 2.15. Добавление программатора

Теперь добавим инструмент (в основном меню раздел *Tools→Configure Tools*). Нажимаем кнопку **Add,** добавляем, скажем, *stutil,* что потом настраиваем:

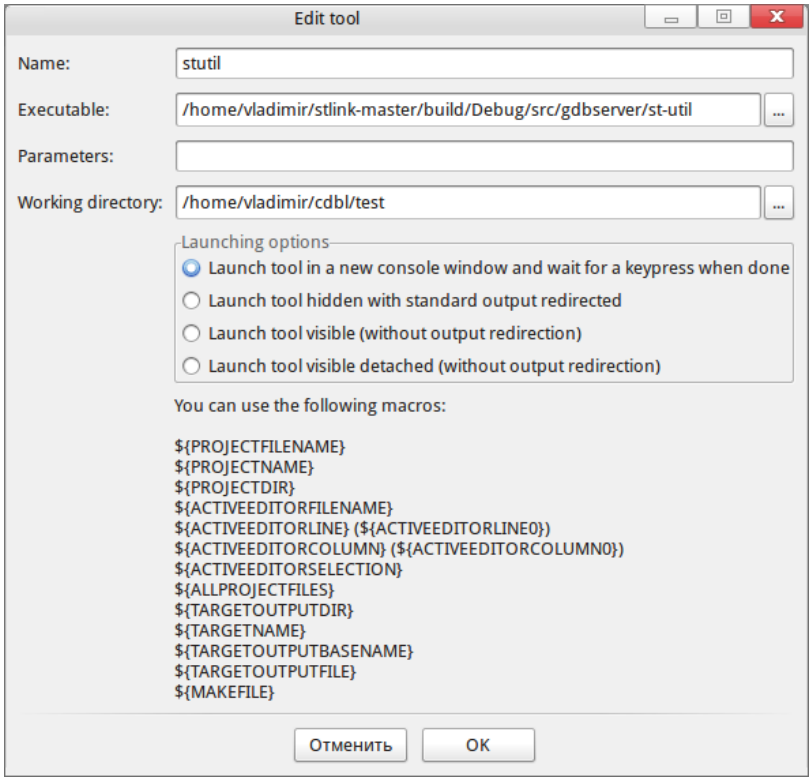


Рис. 2.16. Конфигурация утилиты

Теперь включаем режим отладки в окошке, где мы включали *Release*. Запускаем пересборку проекта. Добавляем точку останова к одной из наших добавленных двух строк. Включаем нашу утилиту (основное меню *Tools→stutil*). Должно появиться окно, которое можно свернуть, чтобы по окончании этапа отладки остановить работу утилиты, выключив её обычным образом.

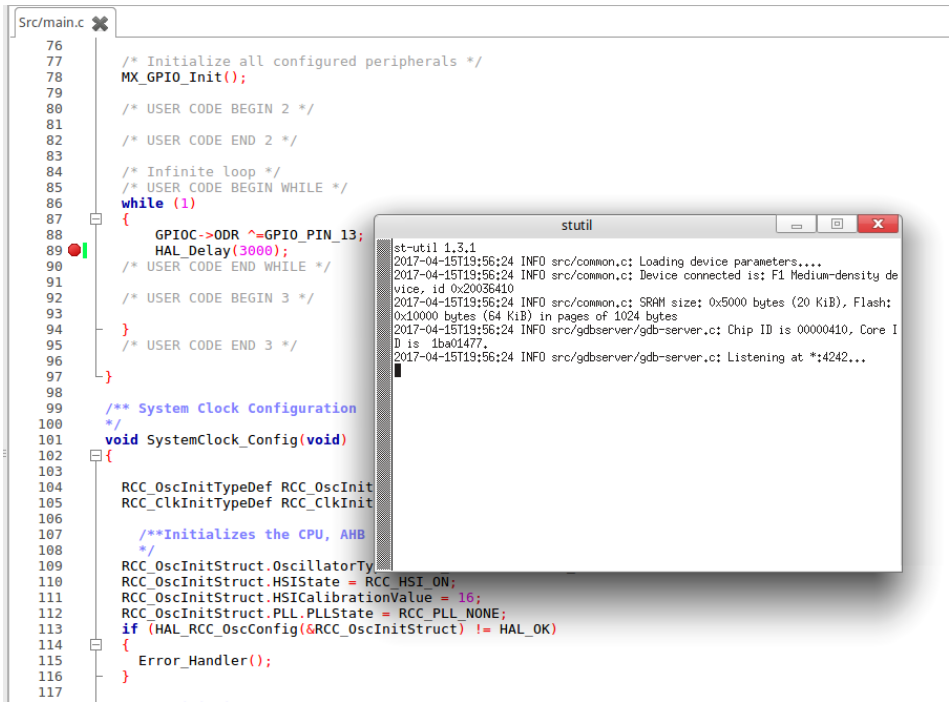


Рис. 2.17. Включён сервер dbg

Теперь можно запустить отладку (*Debug→Start/Continue*), открыв, например, окно дисассемблера (*Debug→Debugging Windows*).

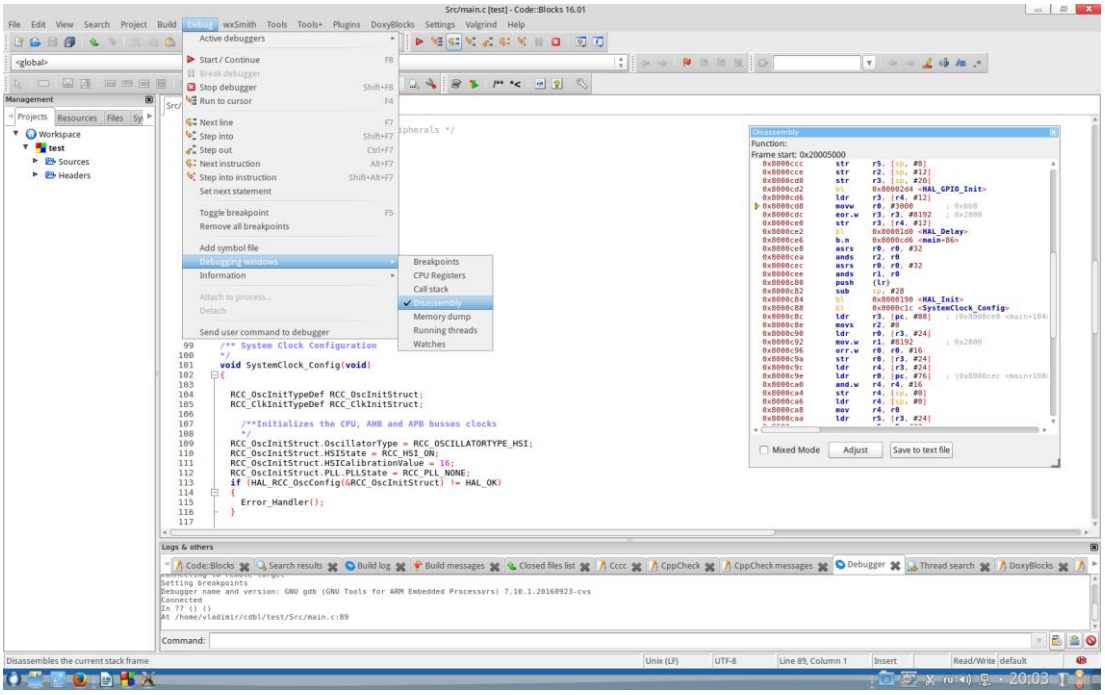


Рис. 2.18. Отладка программы

Завершив отладку, можно выключить отладку (кнопка останова на инструментальной панели или через основное меню, раздел *Debug*) и остановить сервер, выключив его так, как выключается любая программа — закрыть окно.

При удачном скачивании программы System Workbench, надеюсь, вам не придётся повторять что-то из рассказанного мной выше. Возможно, установка программы будет не сложнее установки STM32CubeMx, а всё остальное будет установлено при этой процедуре. Я не уверен, что использование Ubuntu не облегчит вам жизнь весьма существенно. И даже не уверен, что в следующем выпуске ROSE не придётся отыскивать решение заново.

Осталось описать создание какого-нибудь проекта для STM32, где я хочу показать, почему выбрал программу Code::Blocks.

Дополнение

В конце четвёртой главы я написал, что проверю установки и настройки программ для работы с ARM-контроллером. План был такой – читать то, что я написал ранее, и повторять это шаг за шагом, стараясь не забывать, а описывать каждый шаг. Сразу признаюсь – далеко не всегда я следовал этому плану. Оказалось, что ранее я упустил многое в описание, но хуже другое, не всё во второй раз получилось так, как получилось в первый раз.

Для работы программ требуется установка пакета JAVA JRE 1.7 Sun. В Linux ROSA я использую менеджер работы с пакетами, где в окно поиска ввожу java 1.7.0, а в появившемся длинном списке выбираю несколько, начинающихся с java-1.7.0-openjdk…, java-1.7.0-openjdk-headless…, libjavascriptcoregtk1.0_0-GTK+port…

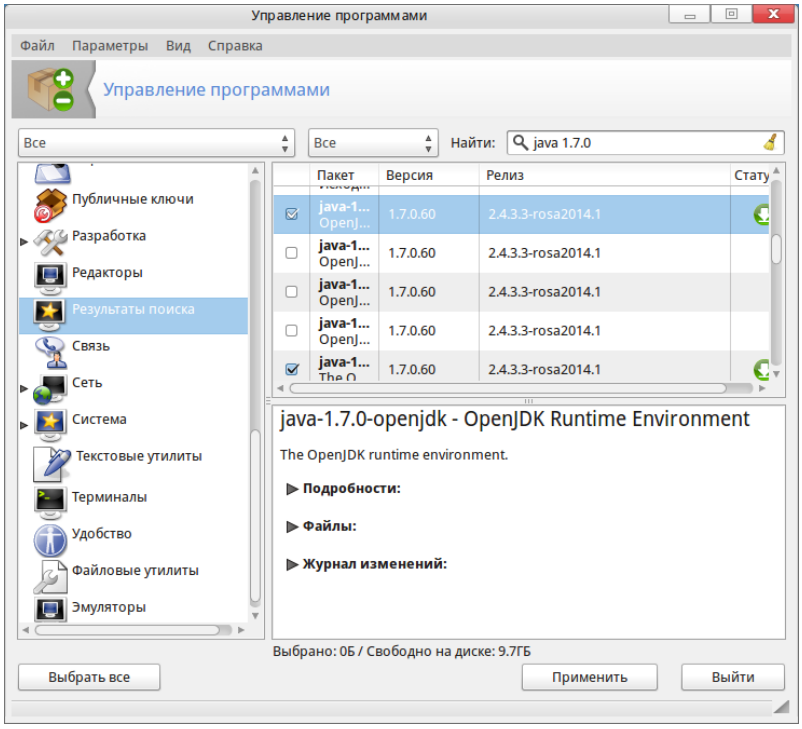


Рис. 2.19. Установка пакета JAVA

Установку программ я решил начать с установки программы STM32CubeMX, поскольку установка этой программы была наиболее простой. Обозначив файл установки исполняемым, как и писал ранее, я устанавливаю программу.

При первом запуске создания шаблона появляется сообщение:

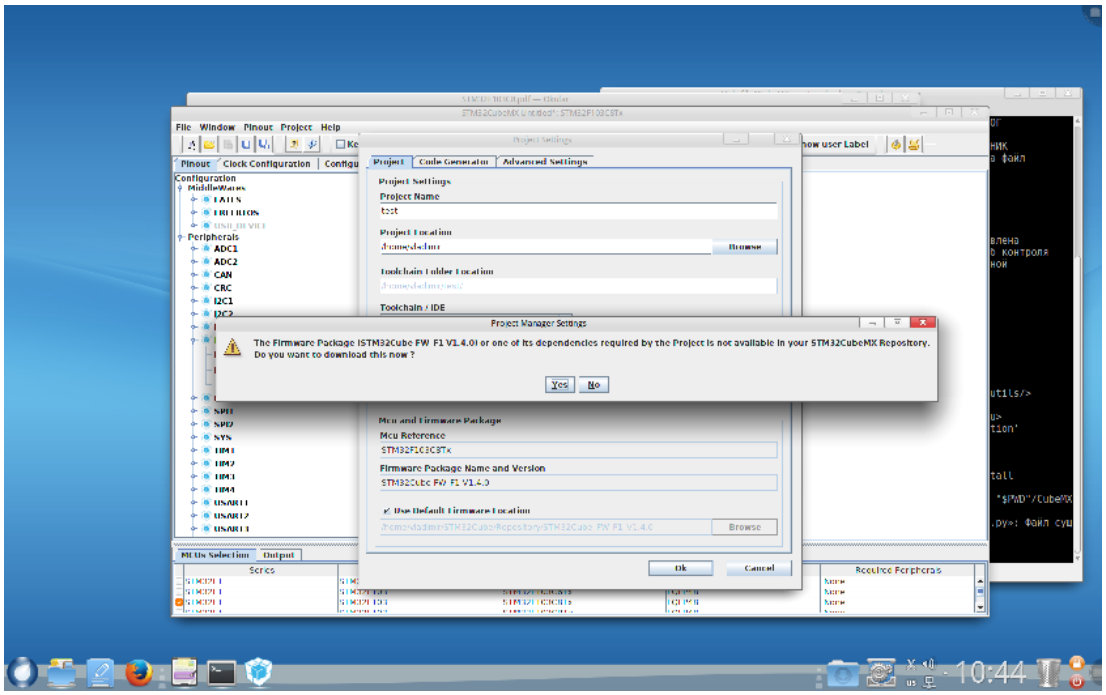


Рис. 2.20. Сообщение о необходимости загрузить и установить дополнение

Кнопка **Yes** приводит в действие механизм скачивания и установки. В результате в домашней директории, откуда я устанавливаю программу, появляются две папки, которые создаются при установке и которые удалять не следует.

С пакетом для работы с arm я вновь поступаю, как и раньше – создаю папку arm, куда копирую все файлы, а затем с помощью консоли отправляю её в раздел /usr.

Далее, похоже, следует установить пакет stlink. Я пробую установить и stlink-master и stlink-1.3.1. Дело в том, что проблемы возникли далее, что потребовало от меня подобной суеты.

Для установки stlink я с помощью менеджера управления программами устанавливаю CMake, libusb, libusb-devel, clang, Ccompile, gtk+-common, libgtk+-devel, libgtk+3.0, libgtk+3.0-devel, gdb, libgdb. Не уверен, что всё перечислил и что не «зацепил» лишнего.

При установке с помощью консоли, выполняя sudo make –DMAKE…, как это написано в руководстве по установке, не следует забыть завершить эту команду двумя точками (если не ошибаюсь). Я создаю установку и для Debug, и для Release, поскольку не уверен, что достаточно только установки для отладчика.

Полная установка в консоли stlink выглядит так:

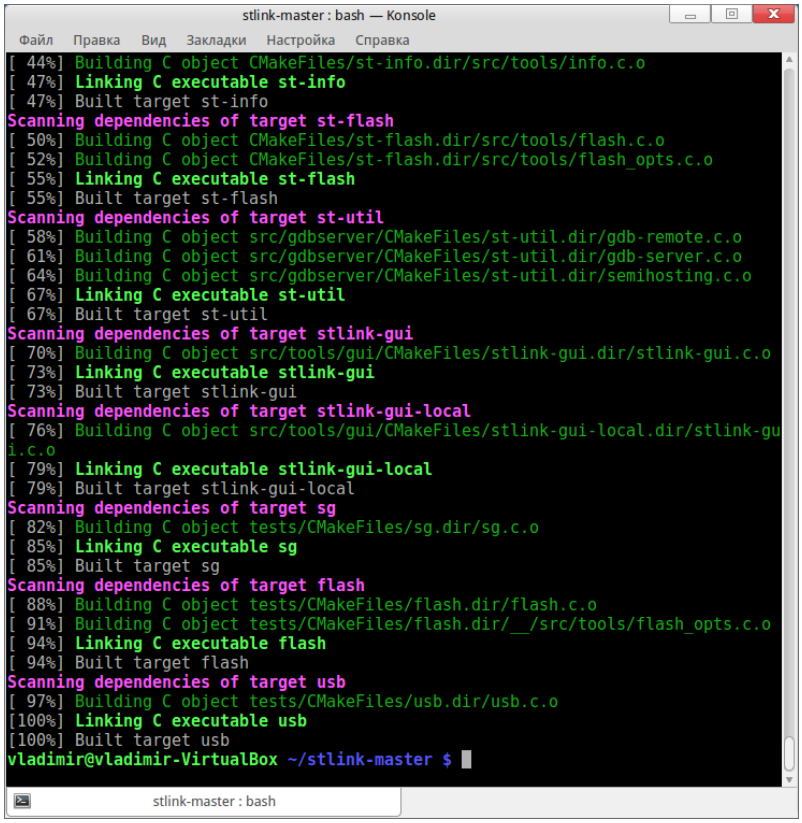


Рис. 2.21. Установка в консоли stlink

Касательно установки Makefile4CubeMX возникает проблема. К слову, сама установка делает в системе (в /usr/bin) только ссылку на папку, где находится эта программа. У меня она в моей домашней директории. А проблема возникает такого рода – программа не находит arm-none-eabi-gcc, хотя я исправил файл, как писал ранее, в части задания путей. В прошлый раз эту проблему я

не отметил. Решить проблему помогает исправление одной строки файла CubeMX2Makefile.py, где я задаю полный путь к команде:

```
p = subprocess.Popen("/usr/arm/bin/arm-none-eabi-gcc -dumpversion",
stdout=subprocess.PIPE, shell=True)
```

При работе в Code::Blocks с arm-контроллером настройка, похоже, потребуется каждый раз, когда выполняется новый проект, чтобы указать нужный компилятор, задав его компилятором по умолчанию.

Пока я довольно долго разбирался с тем, как работать с arm-контроллером, мне приходило в голову – если не получится загружать программу из Code::Blocks, то, что делать?

Оказалось, что достаточно выполнить в консоли команду stlink-gui:

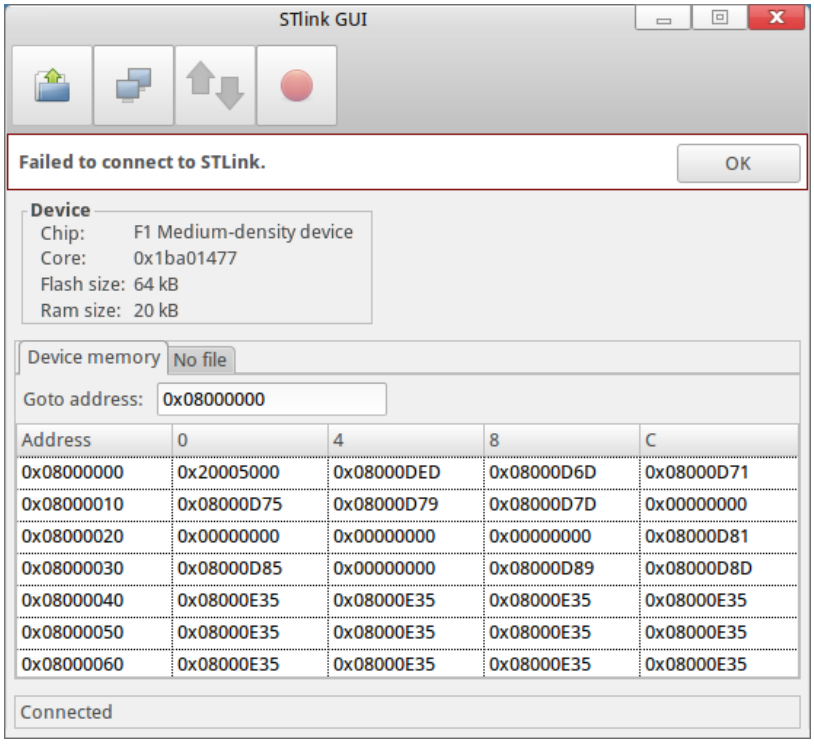


Рис. 2.22. Программа для загрузки bin-файлов в STM32F103C8

3 Установка и настройка (продолжение)

После того, как завершилась установка и настройка Code::Blocks для работы с модулем STM32, я решил, что пора заняться второй частью того, о чём хотел рассказать. Мне показалось, что в этой части не должно быть препятствий. Но, как это часто со мной бывает, я ошибся.

После долгого времени в «безутешных» попытках использовать приложение wxWdgets для программы Code::Blocks, когда я пытался установить загруженное с сайта производителя приложение версии и 2.8.0, и версии 3.0.0, получая либо сообщение об отсутствии всех библиотек, начиная с основного файла app.h, либо сообщение об отсутствии файла setup.h, я

решил прекратить эти попытки, поскольку ни манипуляция с вариантами установки, ни поиск советов в Интернете не дали положительного результата. Получалось так — не судьба.

При поиске решения проблемы в Интернете я часто встречал рассказы о том, как с модулем STM32 и программой Code::Blocks легко работать в дистрибутиве Ubuntu. И я смирился с мыслью, что в этом месте рассказа я перейду к такому варианту развития событий.

Ресурсы компьютера, при всей их значимости, если сравнивать его возможности с тем, что было лет десять назад, тем не менее, не безграничны. Поэтому я удалил установку ROSA на виртуальной машине (всё равно я собирался повторить всю установку и настройку для STM32) и установил Ubuntu 17.04. По первому впечатлению этот дистрибутив работал гораздо «натужнее», чем ROSA, но хуже было другое — я не смог установить и настроить wxWidgets, как ни старался.

Итогом этого «сизифова труда» стал снос «новодела» и возвращение к дистрибутиву ROSA.

После установки дистрибутива на виртуальную машину и установки программы Code::Blocks я решил ещё раз повторить установку wxWidgets версии 3.0.0.

Начало этому процессу положила установка дополнительных библиотек для gtk, без которых даже конфигурация не завершалась успешно. В прошлый раз я установил много дополнений и библиотек, не зная, что нужно, а что нет. Сейчас я решил дополнять библиотеки постепенно, установив для начала libgtk+-devel, libgtk+1.2, libgtk+2.0-devel, libgtk+2.0_0:

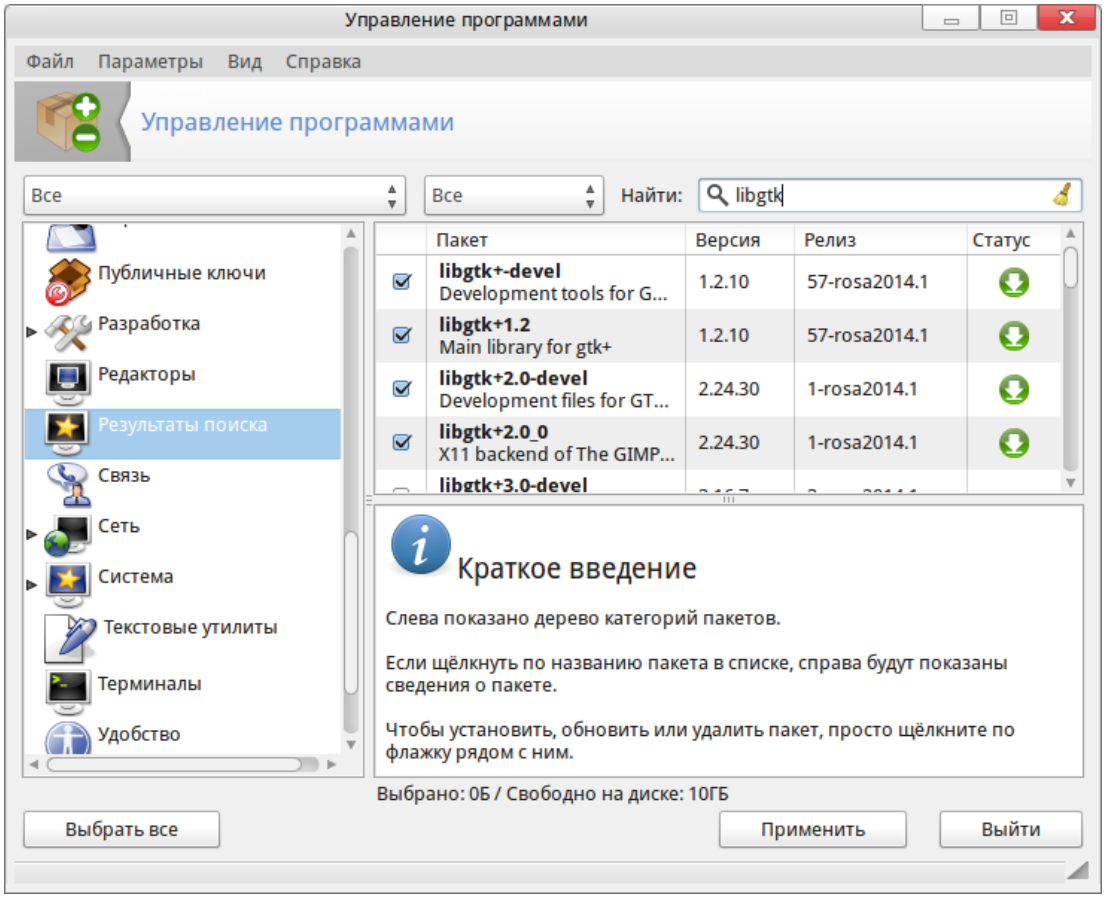


Рис. 3.1. Установка библиотек

При этом по зависимостям автоматически установилось много других библиотек, но уже без моего вмешательства.

В консоли, переместившись в папку с распакованными файлами wxWidgets-3.0.0, командой конфигурации: ./configure --with-gtk=2 --enable-unicode —prefix=/usr/, - я выполнил первый этап установки. Касательно unicode — в предыдущих попытках без установки дополнительной опции использовать эту кодировку появлялась ошибка с сообщением, что не распознаётся некий символ Zx_что-то там. А установка дополнительной опции приводила к отказу из-за отсутствия setup.h. Я находил этот файл, пытался «подпихнуть» его в разные места, но это приводило к появлению ещё большего количества ошибок. А в том, что относится к разным местам, если не добавить prefix, то установка в директорию /usr/local в какие-то моменты требовала дополнительных указаний программе, в каком include искать эти включения. Завершил я это командой sudo ldconfig, как советует установка дополнения.

После завершения удачной конфигурации типовыми командами make и sudo make install я завершил, признаюсь без особой надежды, процесс добавления wxWidgets в программу Code::Blocks.

Создание первого проекта выглядело так:

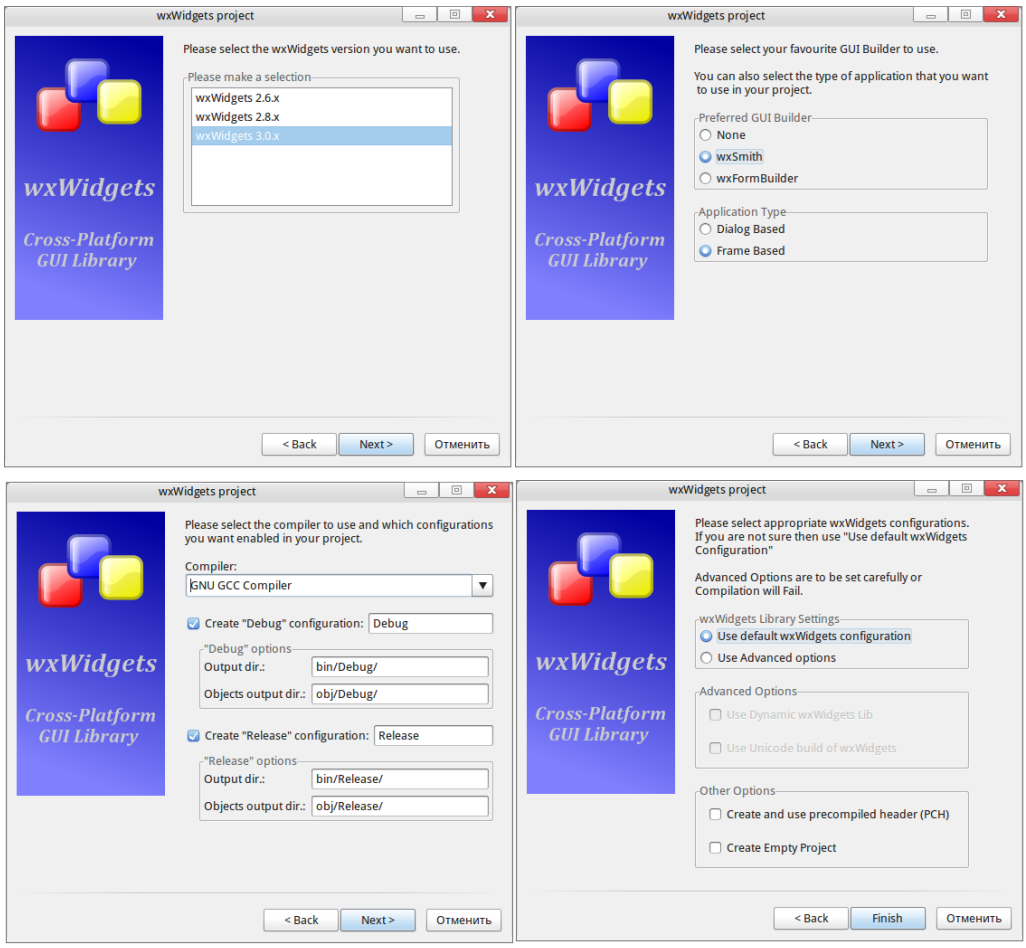


Рис. 3.2. Настройки при создании проекта с wxSmith

Я пропустил очевидные страницы с заданием имени проекта и профиля владельца. Но, когда я запустил сборку пустого проекта с единственной формой, я был удивлён, не буду скрывать этого, сборка завершилась без ошибок. Добавив кнопку на форму, я узнал результат, знакомый мне по предыдущим опытам с Code::Blocks:

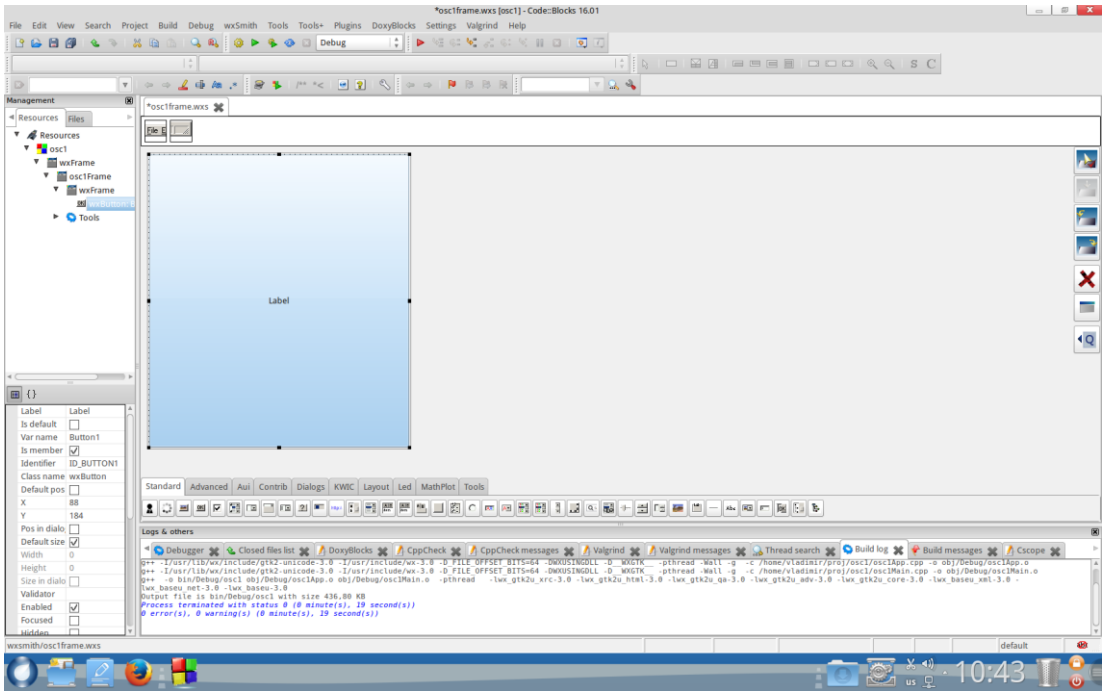


Рис. 3.3. Добавление одной кнопки на форму

Добавление второй кнопки — самый простой способ получить привычный вид формы:

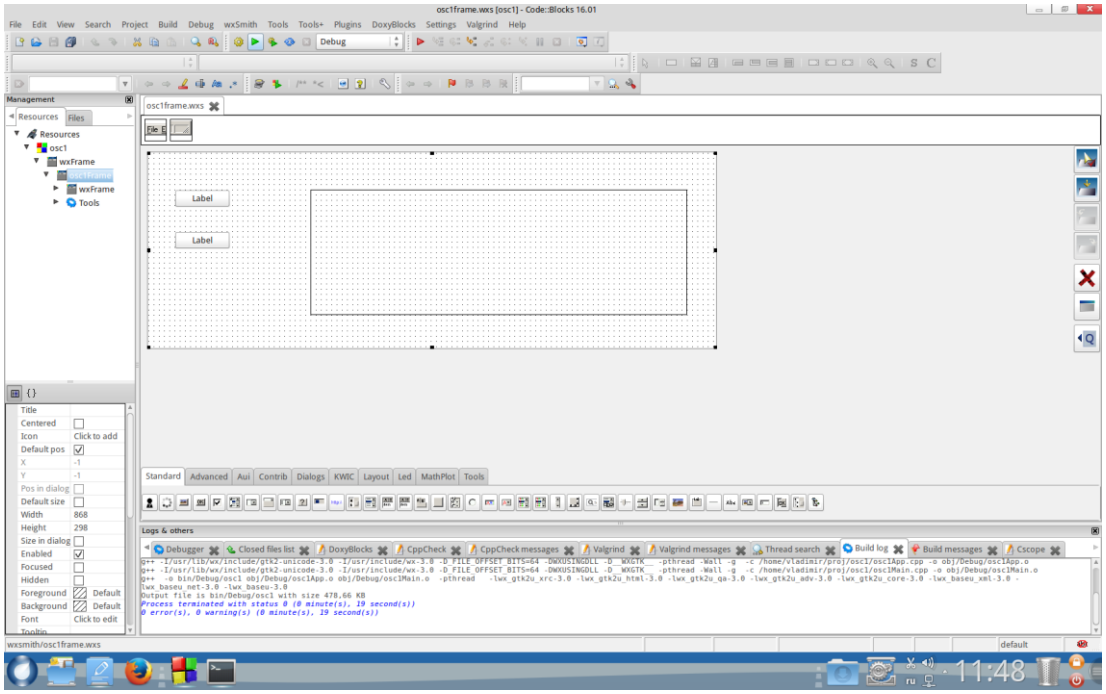


Рис. 3.4. Форма с двумя кнопками и панелью для графика

Что же я хочу? Я хочу в качестве примера работы с ARM-контроллером получать данные от АЦП модуля STM32F103C8 и отображать их на графике формы, созданной в Code::Blocs. В Code::Blocs создаётся и программа для микроконтроллера.

Прежде, чем приступить к работе, я хочу проверить две простые вещи — нарисовать прямую в окне формы и опробовать работу этой программы для компьютера с COM1-портом (/dev/ttyS0).

Итак. Необходимые мне команды я постараюсь найти в Интернете, например, на сайте: http://wiki.codeblocks.org/index.php/WxSmith_tutorial:_Creating_items_with_custom_paint_and_mouse_handling

И полезная ссылка: http://wiki.codeblocks.org/index.php/Tools%2B_reference

В первой тестовой программе двойной щелчок левой клавишей мышки по панели даёт прибавку к файлу osc1Main.cpp (osc1 — это название моего проекта), куда я добавлю несколько строк:

```
void osc1Frame::OnPanel1Paint(wxPaintEvent& event)
{
     wxPaintDC dc( Panel1 );
    dc.SetPen( wxPen( *wxBLUE, 2 ) );
    dc.DrawLine( 0, 0, 100, 100 );
}
```

И добавлю пару строк в раздел включений, чтобы он стал таким:

```
#include "osc1Main.h"
#include <wx/msgdlg.h>
#include <wx/graphics.h>
#include <wx/dcclient.h>

//(*InternalHeaders(osc1Frame)
#include <wx/intl.h>
#include <wx/string.h>
//*)
```

Если теперь собрать проект и запустить его на выполнение, то получится такой результат:

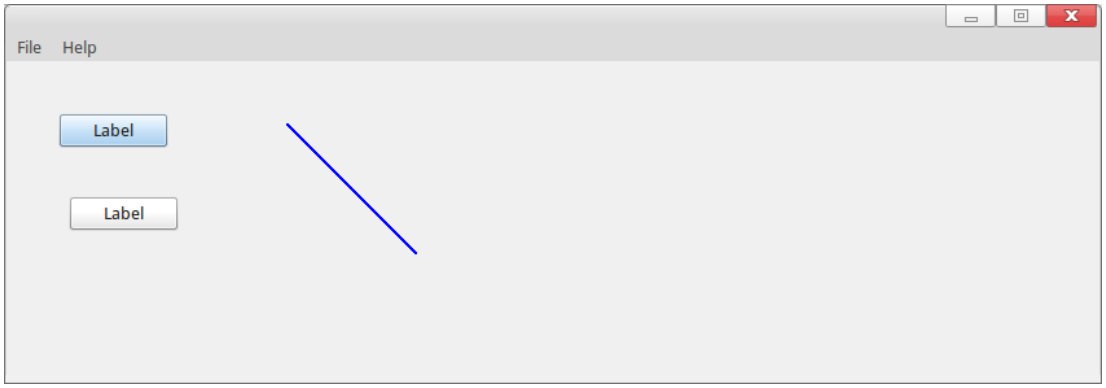


Рис. 3.5. Результат первого рисования на графической панели

Чтобы окончательно убедиться в работоспособности этой части проекта, я добавлю ещё несколько строк:

```
void osc1Frame::OnPanel1Paint(wxPaintEvent& event)
{
    wxPaintDC dc( Panel1 );
    dc.SetPen( wxPen( *wxBLUE, 2 ) );
    dc.DrawLine( 0, 0, 100, 100 );
    dc.Clear();
    dc.SetPen( wxPen( *wxRED, 2 ) );
    dc.DrawLine( 10, 0, 120, 100 );
}
```

Теперь первая синяя линия стирается, а вместо неё появляется красная, которая немного смещена в сторону. Пока меня это устраивает. Тест завершён.

Для второго теста я создаю новый проект, добавляю два файла из скачанного в Интернете пакета seriallib в проект: serialib.cpp, serialib.h. Я их просто копирую в папку проекта, предварительно закрыв его (на всякий случай). Затем открываю его вновь, в окне менеджера проекта перехожу на закладку *Project*, есть такая закладка, и, щёлкнув правой клавишей мышки по названию проекта, выбираю из выпадающего меню добавление файлов. В диалоговом окне отмечаю оба файла и нажимаю кнопку Открыть.

Далее… я дважды щёлкаю левой клавишей мышки по первой кнопке, которую назвал Start, затем по второй, названной Stop. Я хотел добавить на форму этикетку (label), но не нашёл её, добавив ещё одну кнопку, и записываю такой текст:

```
void twoFrame::OnButton1Click(wxCommandEvent& event)
{
    ret = LS.Open(DEVICE_PORT, 9600);
    if(ret !=1) {
         Button3->SetLabel("NOT");
    } else {
                Button3->SetLabel("OK");
}
```

При сборке проекта в файле serialib.h понадобилось заменить NULL числом 0, например:

Вместо `char ReadChar (char *pByte,const unsigned int TimeOut_ms=NULL);`

Следует `char ReadChar (char *pByte,const unsigned int TimeOut_ms=0);`

Эту операцию упрощает *Search→Replace*, где можно провести все замены сразу.

К разделу включений добавилось следующее:

```
#include "serialib.h"
#define  DEVICE_PORT  "/dev/ttyS0/"
serialib LS;
int ret;
```

После сборки проекта проверка выглядит ...

4 Приключения с COM-портом

В виртуальной машине до начала работы с COM-портом, похоже, надо настроить этот порт, используя иконку «Настроить». Следует выбрать из списка устройств порт, выбрать COM1 и вписать его ниже. Теперь после запуска виртуальной ОС с портом можно будет работать.

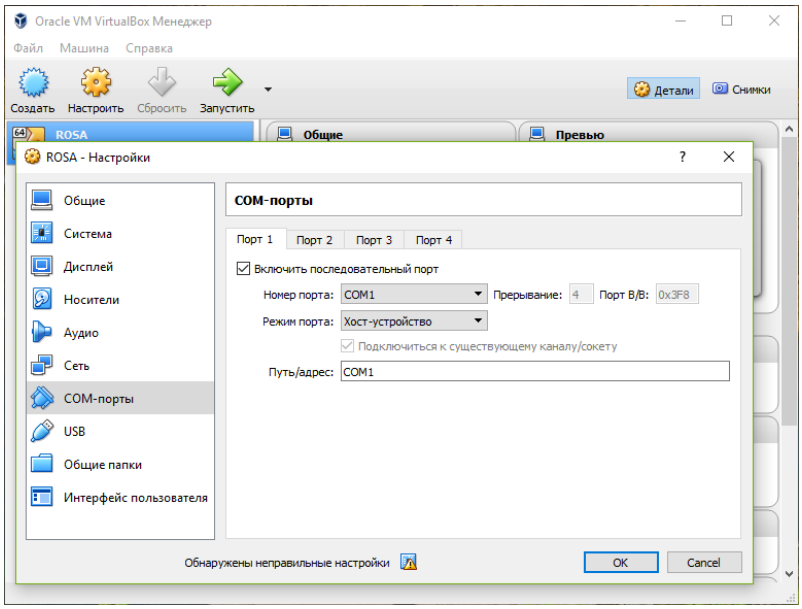


Рис. 4.1. Основная настройка подключения порта

Проверяя работу программы для COM-порта я убедился, что подключение к ttyS0 удачно.

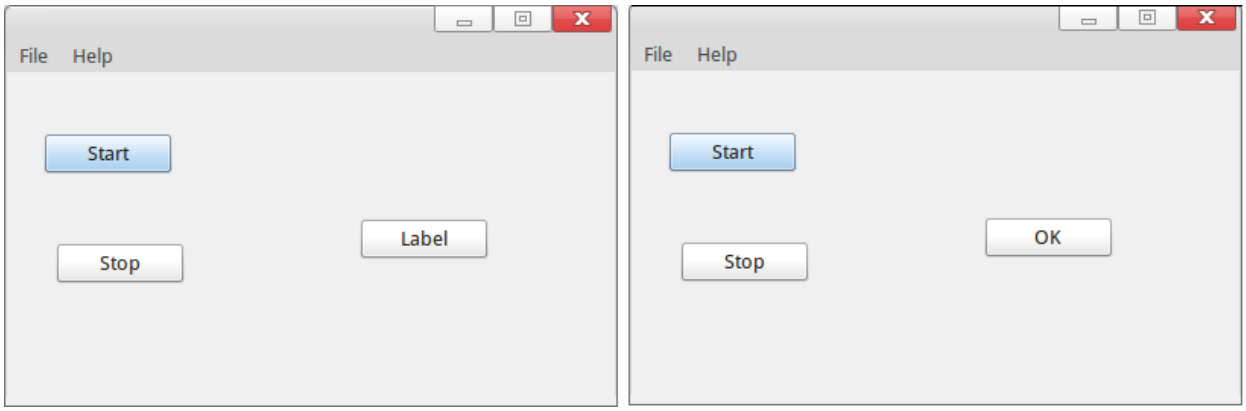


Рис. 4.2. Первая проверка программы

Было бы лучше, если бы после замены ttyS0 на ttyS1 на этикете (третьей кнопке) появилось бы NOT. Но этого не происходит. Для проверки этой странности, в Windows замена COM1 на COM2 даёт именно такой результат, можно воспользоваться консольной командой:

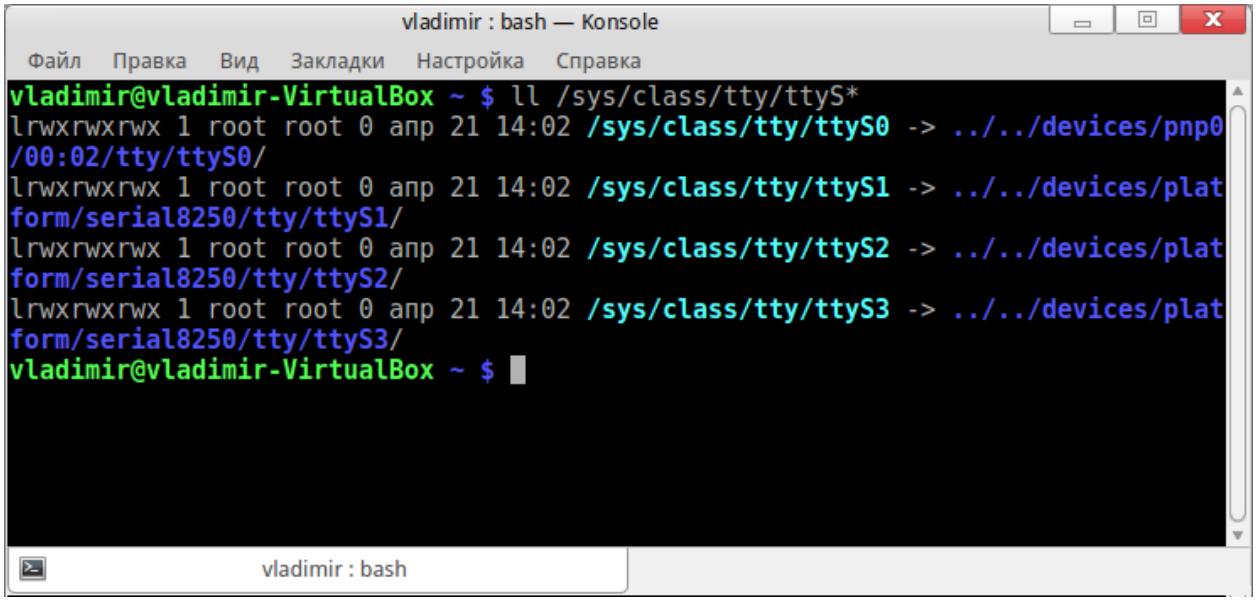


Рис. 4.3. Проверка COM-портов

Теперь мне понятно (или я допускаю, что понятно), отчего и порт ttyS1 ответил OK. Чтобы работать с портом ttyS0, мне понадобится в Linux RUSE сделать пользователя членом группы dialout. Для этого нужно использовать настройки (иконка в правой части панели с шестерёнкой неподалёку от часов и корзинки). Нажав на эту кнопку, в окне «Параметры системы» следует выбрать раздел «Управление пользователями». Ввод пароля root открывает окно, где появятся все пользователи.

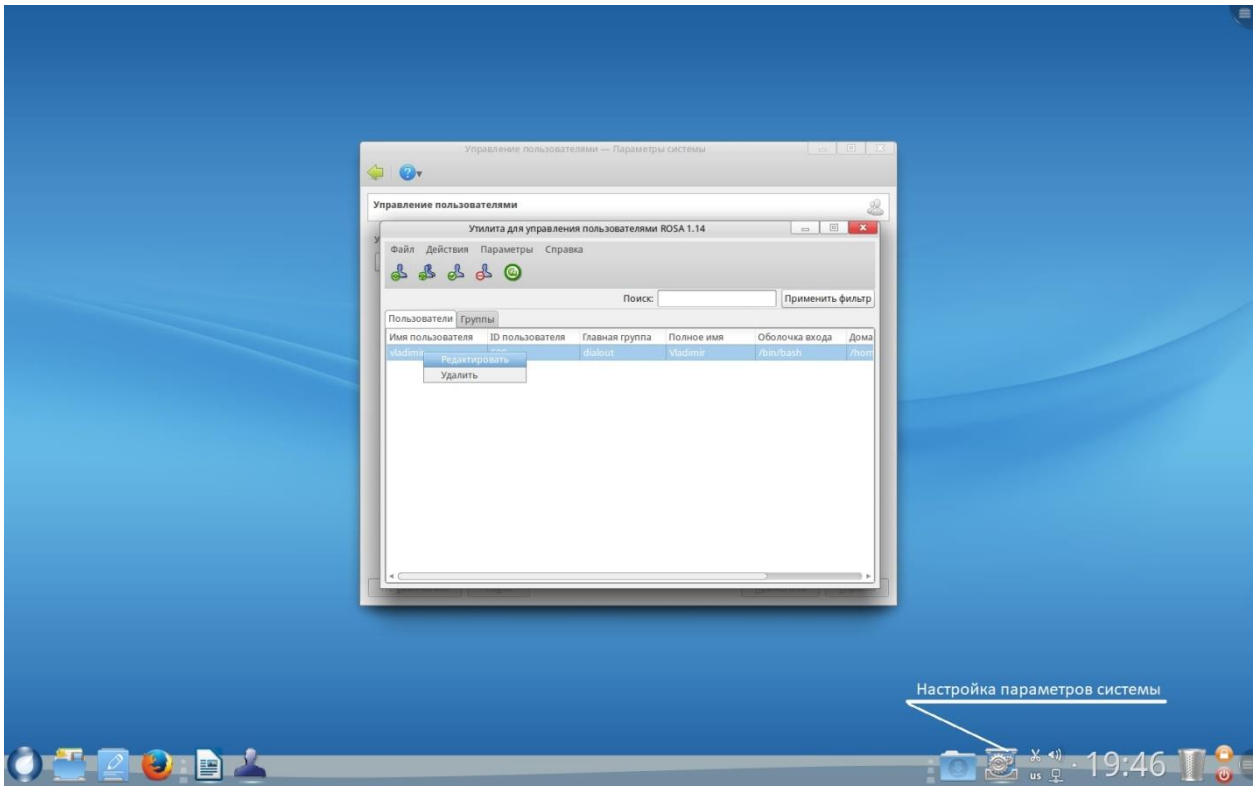


Рис. 4.4. Диалоговое окно выбора пользователя

Щелчком правой клавиши мышки по нужному пользователю можно открыть его для редактирования. А на закладке «Группы» нового диалогового окна можно отметить, в какую группу следует добавить пользователя.

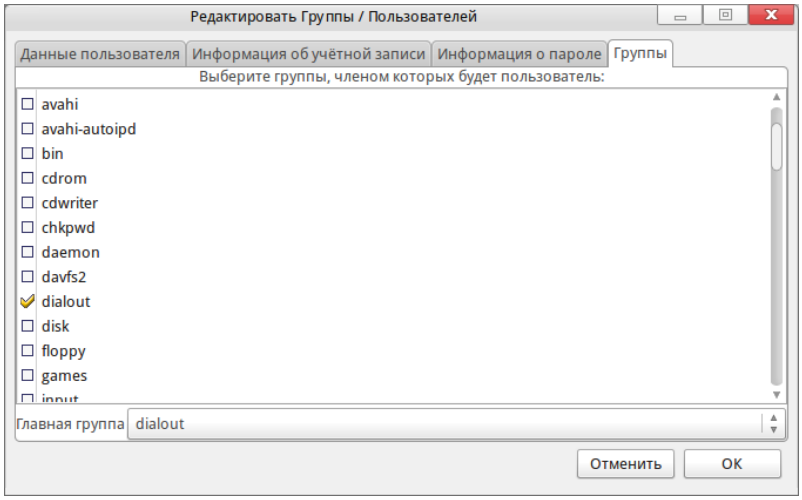


Рис. 4.5. Добавление пользователя в нужную группу

Пользователь, добавленный в группу dialout, может работать с COM-портом. Это можно проверить консольной командой:

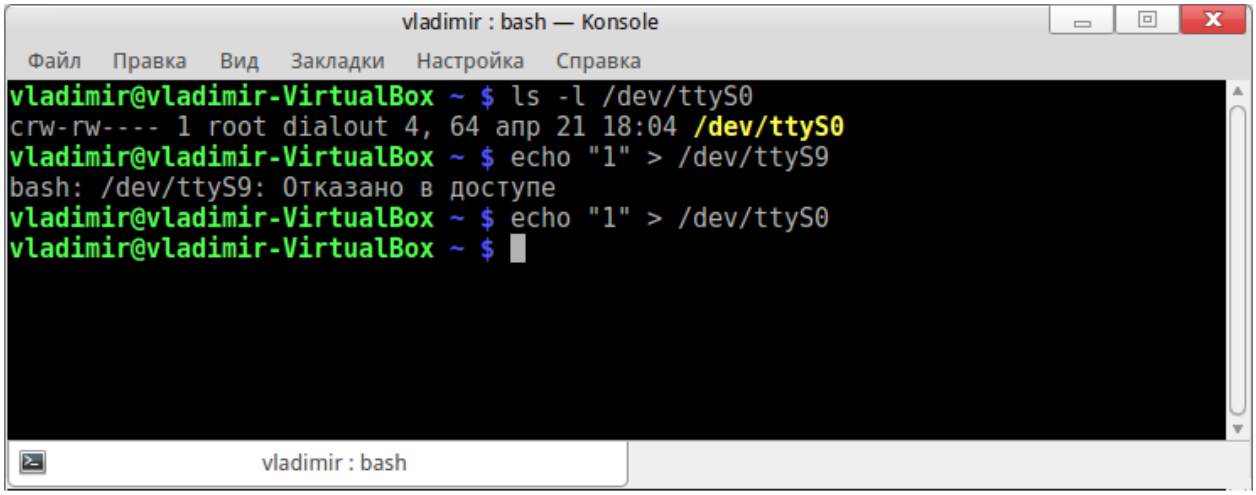


Рис. 4.6. Проверка доступа к COM-порту

Первая команда показывает, какая группа даёт доступ; вторая команда показывает, что к неправильному порту доступа нет; третья команда показывает, что она проходит к правильному порту.

В Windows 10, где я использовал Keil uVision 5 для работы с модулем STM32, а MS Visual C для создания графического интерфейса, нужный порт оказывался виртуальным COM3, поскольку модуль включался в USB-порт. В данном случае, подключив модуль к виртуальной машине...

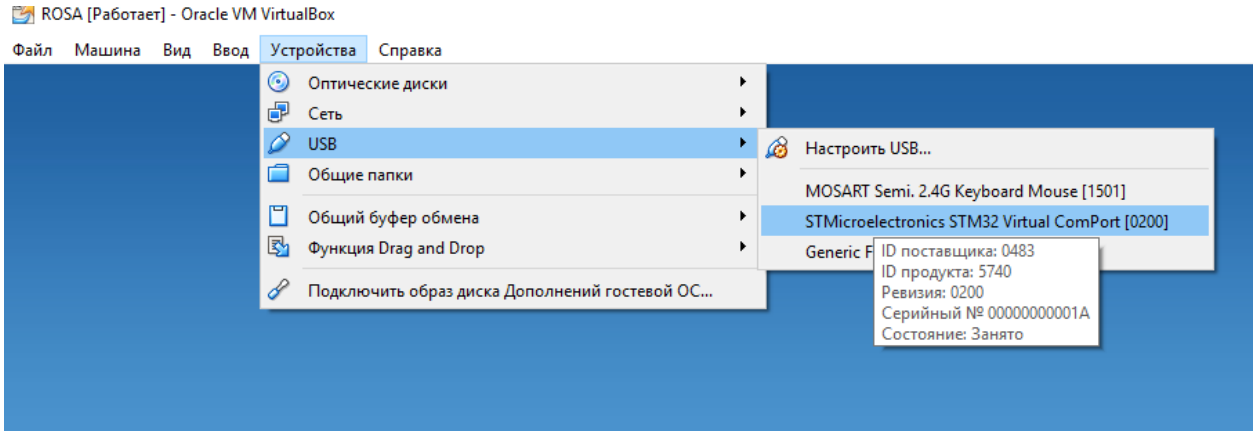


Рис. 4.7. Подключение модуля STM32F103C8 к виртуальной машине

...я, скорее всего, обнаружу его как другое устройство, а не ttySx. Для этого можно использовать раздел «Утилита настройки оборудования» в настройках параметров:

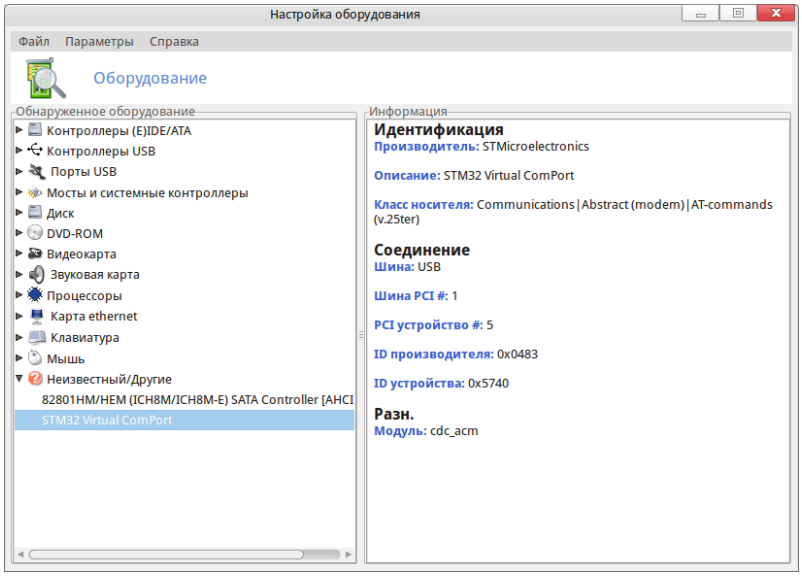


Рис. 4.8. Подключённый модуль STM32F103C8

И обнаруживается он в устройствах как:

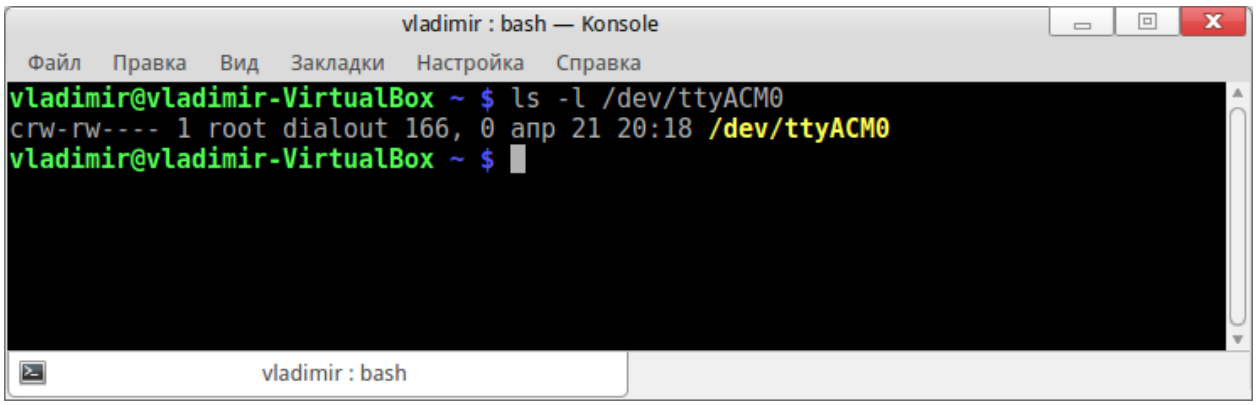


Рис. 4.9. Обращение к модулю

В программе я могу заменить /dev/ttyS0 на /dev/ttyACM0, но пока не очень понимаю, как работать с этим устройством, поскольку не уверен, что при обращении к нему устройство возвращает 1.

В Windows мне для того, что я задумал, очень помогло приложение к AVRStudio Data Visualizer, чем воспользоваться в Linux — это вопрос…

Пока я обдумываю этот вопрос, пока неясно, что делать дальше, я проверю, как и собирался, все настройки Code::Blocks для работы с ARM-контроллерами (что и описал выше в разделе «Дополнение»).

5 Создание проекта для STM32F103C8

Проект, о котором я хочу рассказать, подразумевает, что модуль STM32F103C8, используя АЦП и прямой доступ к памяти (модуля, конечно), сканирует переменное напряжение от генератора и передаёт его через порт USB. Вторая часть проекта – это графический интерфейс, который будет принимать данные, и отображать их на графике. В итоге получается нечто, что напоминает осциллограф-приставку для компьютера.

Начинается работа с запуска программы STM32CubeMX, где создаётся новый проект, который я назову osc_arm. На первой закладке осуществляется выбор, показанный ниже на рисунке:

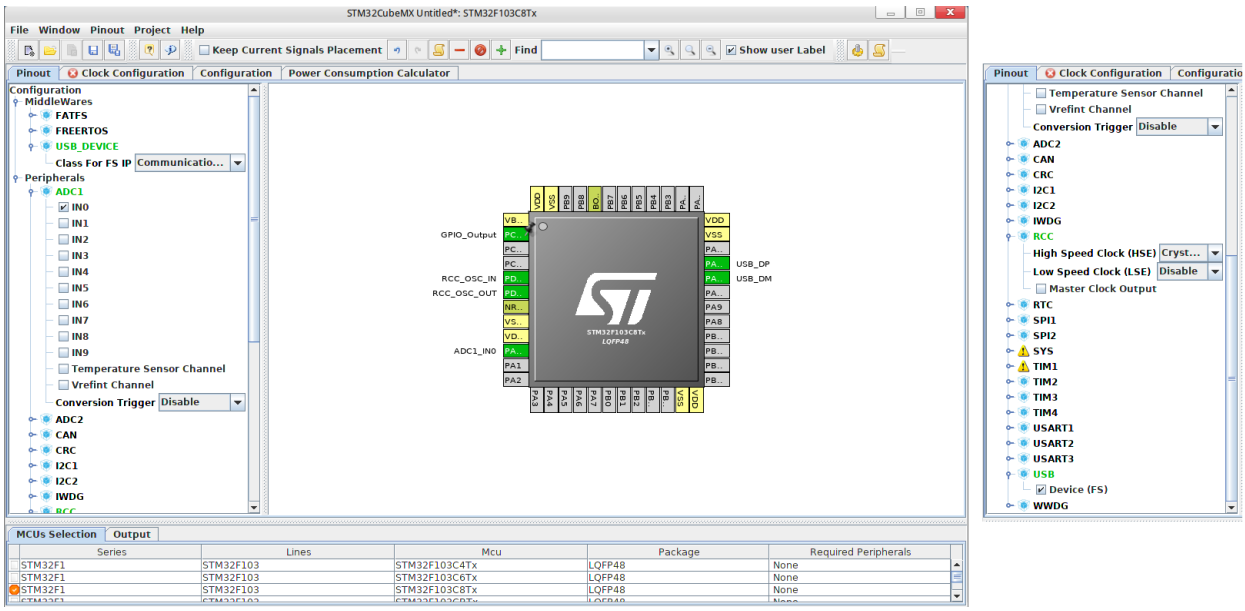


Рис. 5.1. Выбор необходимых компонентов для будущей программы

Список в окне конфигурации слишком длинный, поэтому его продолжение показано справа.

После задания конфигурации переходим на закладку *Clock Configuration*, где можно осуществить настройку всех тактовых частот, используемых модулем. В данном случае лучше положиться на автоматическую настройку, согласившись с предложением.

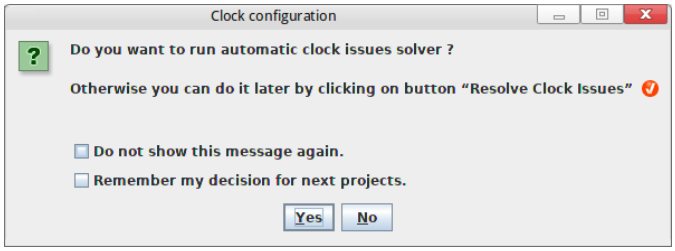


Рис. 5.2. Предложение автоматической настройки тактовых частот

Нажав кнопку **Yes**, вы получите такую картину настройки тактового генератора.

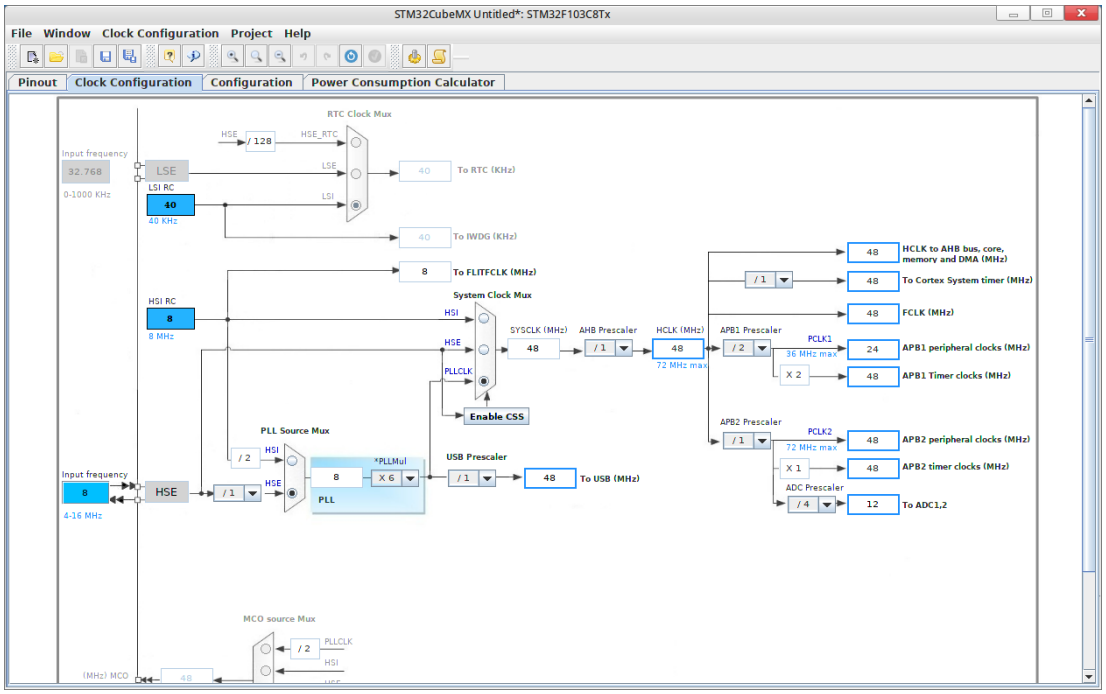


Рис. 5.3. Настройка тактовых частот

Следующая «остановка» - закладка *Configuration*.

На этой закладке можно настроить параметры всех встроенных модулей микроконтроллера STM32. Для многих простых проектов нет необходимости что-то менять в этом разделе, оставив все настройки «по умолчанию».

Тем не менее, постарайтесь заглянуть в этот раздел до того, как возникнет необходимость вносить иземенения.

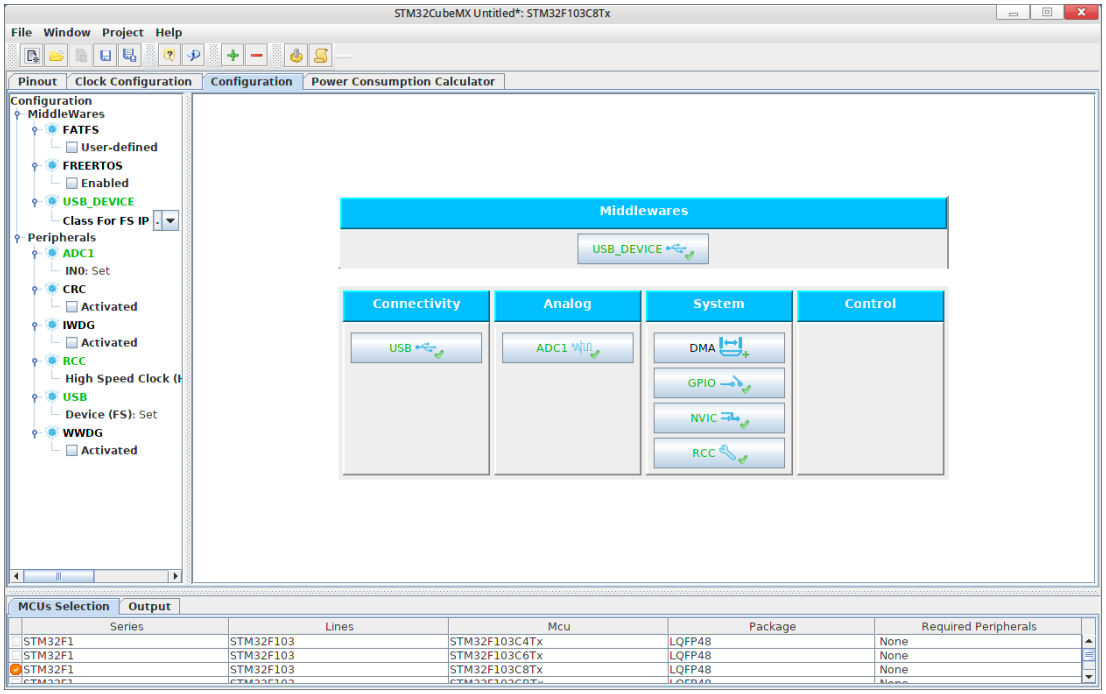


Рис. 5.4. Закладка *Configuration*

В этом месте можно (а что-то нужно) настроить встроенные модули USB, ADC1 и DMA.

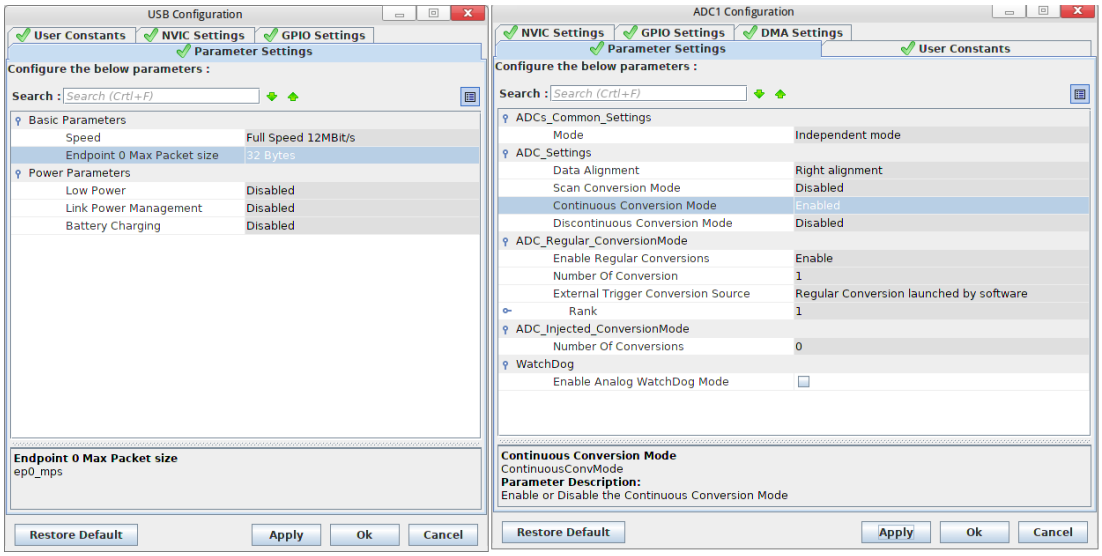


Рис. 5.5. Настройка USB и ADC1

Я не знаю, нужно ли настраивать сейчас USB модуль, но для ADC1 обязательно нужно включить режим продолжительного преобразования.

Последним следует настроить DMA (прямой доступ к памяти).

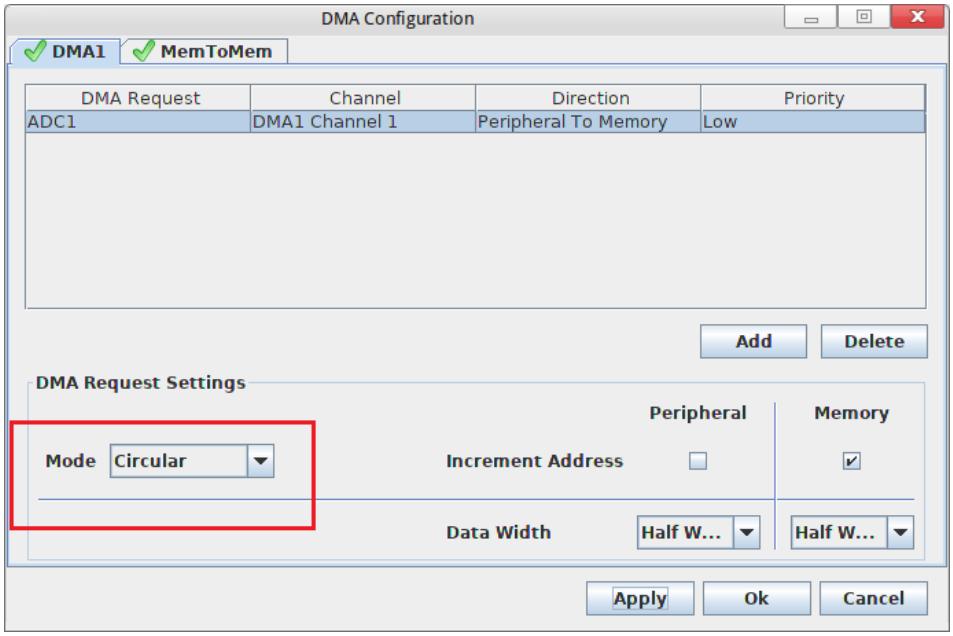


Рис. 5.6. Настройка DMA

Начинается настройка с кнопки **Add,** что даёт возможность выбора в графе *DMA Request*. Щелчком левой кнопки мышки по новому разделу выбираем ADC1. Можно поменять приоритет, если нужно, или оставить всё так, как показано на рисунке. Но обязательно нужно поменять режим на *Circular*. Завершаются настройки, как и предыдущие нажатием на кнопки **Apply** и **OK**. Теперь можно генерировать проект (раздел основного меню *Project*).

После стандартных процедур выбора модели, задания имени проекту и места его расположения запускается генерация шаблона для *SW4STM32*. По её завершению можно закрыть окно сообщения и закрыть программу STM32CubeMX.

Далее, используя консоль, перейти в папку с проектом командой cd и т.д. Когда эта команда выполнена и консоль в папке проекта, вводим команду `CubeMX2Makefile.py .` (пробел и точка, похоже, обязательные параметры команды). Выполнение этой команды должно привести к появлению в папке проекта для Code::Blocks с расширением .cbp.

Запускаем программу Code::Blocks, выбираем раздел «открыть существующий проект, open an existing project» и указываем тот, что получен в результате ранее описанных действий.

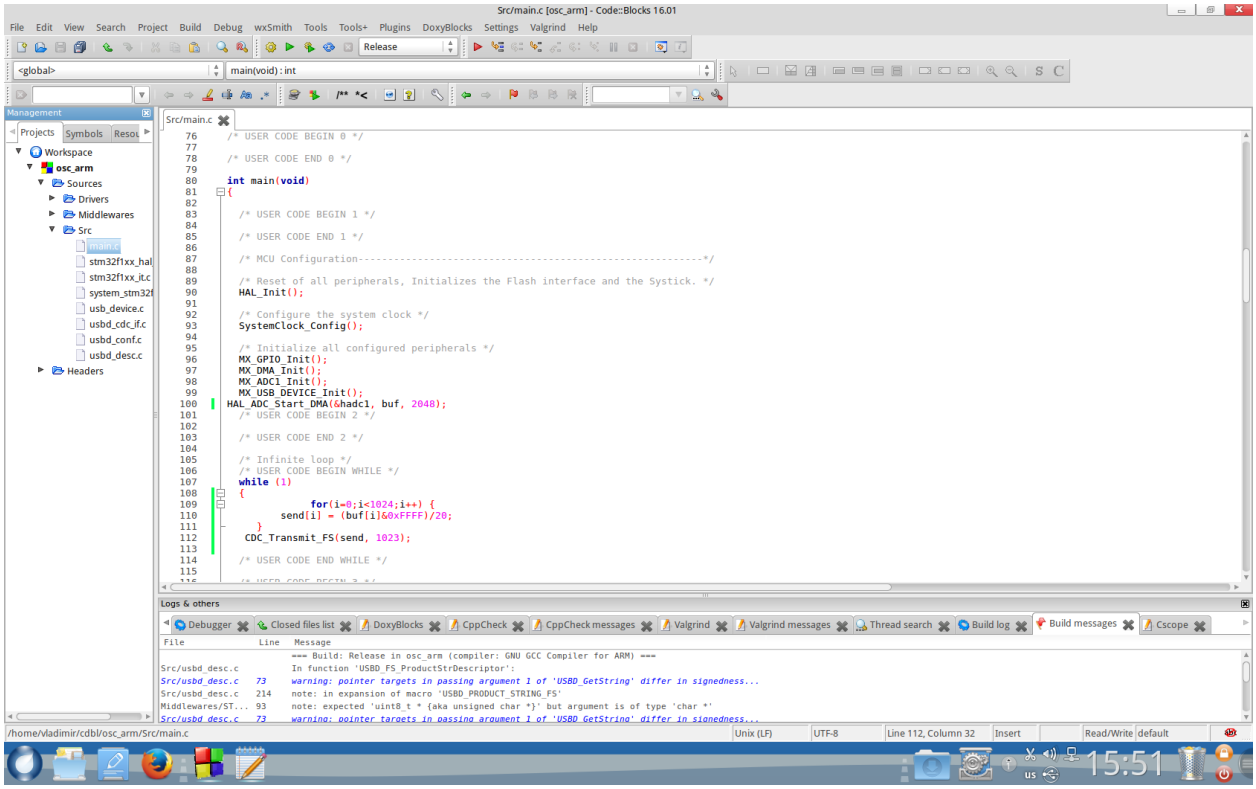


Рис. 5.7. Проект в программе Code::Blocks

Текст проекта имеет строгую разбивку, поэтому все добавления следует делать в местах для них предназначенных. Я не буду вдаваться в детали текста программы, я просто приведу её здесь:

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f1xx_hal.h"
#include "usb_device.h"
#include "usbd_cdc_if.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
uint32_t buf[2048]; // буфер для данных, собранных АЦП
uint8_t send[1024]; // буфер для отправки данных
uint16_t i; // вспомогательная переменная

/* USER CODE END PV */
```

```
/* Private function prototypes -----------------------------------------
-*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------
-*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------
---*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_USB_DEVICE_Init();
HAL_ADC_Start_DMA(&hadc1, buf, 2048); // инициализация работы АЦП
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        for(i=0;i<1024;i++) { // заполняем буфер отправки через буфер АЦП
         send[i] = (buf[i]&0xFFFF)/20;
     }
   CDC_Transmit_FS(send, 1023); // отправляем данные по USB
```

```
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }

  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC|RCC_PERIPHCLK_USB;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
  PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLL_DIV1_5;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
```

```
    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC1 init function */
static void MX_ADC1_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Common config
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Regular Channel
    */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
```

```
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state
*/
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}
```

```
#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/
/****************** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Строки, добавленные мной, я выделил цветом, чтобы было яснее, как мало текста приходится добавлять, а весь текст привёл для того, чтобы вам было легче ориентироваться, ту ли программу вы получили.

Если теперь в программе Code::Blocks выбрать режим *Release*, как показано на рисунке, подключить программатор ST-Link, то запуск на сборку (*Build*) должен завершиться загрузкой программы в модуль STM32.

О проверке того, что модуль работает по USB, я говорил выше, но там же сказал, что не очень понимаю, как увидеть работу программы.

После размышлений и поиска в Интернете я нашёл удобные для меня программы gtkterm и cutecom. Обе есть в составе Linux ROSE. Если теперь запустить любую из программ, подключив модуль STM32F103C8 с помощью кабеля к USB, то можно наблюдать следующее:

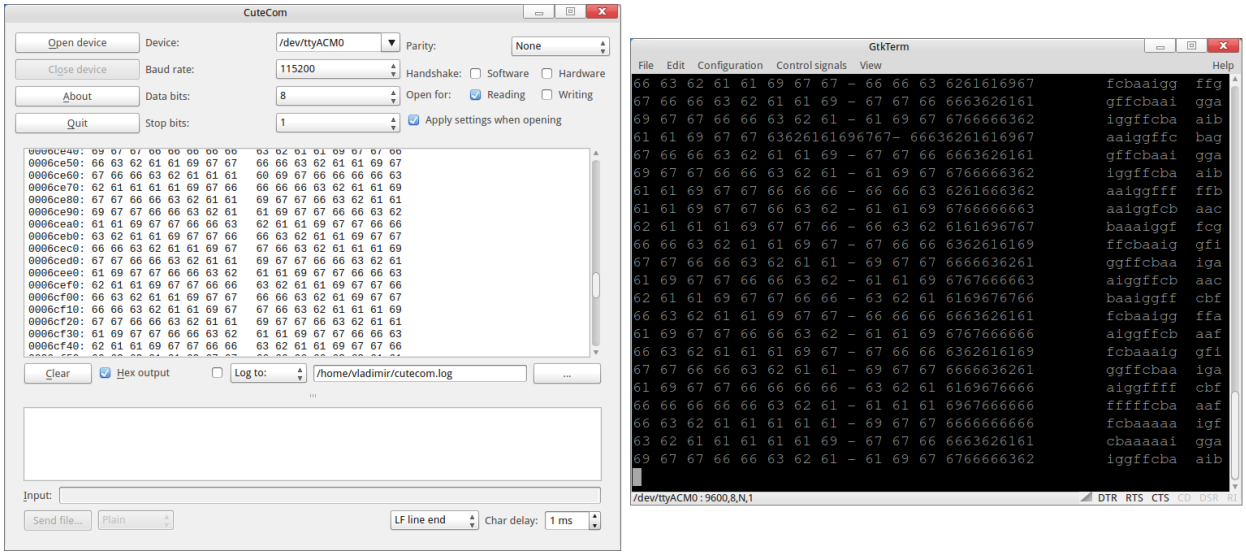


Рис. 5.8. Работа программ CuteCom и GtkTerm

Шестнадцатеричные значения – это шум, который сканирует АЦП. В какой-то момент мне показалось, что программы не подключаются к модулю, тогда я переподключал его, использовал кнопку сброса на плате модуля и проверял его наличие командой в консоли.

С первой частью проекта всё.

Для второй части я создаю проект с wxWidgets (точнее wxSmith).

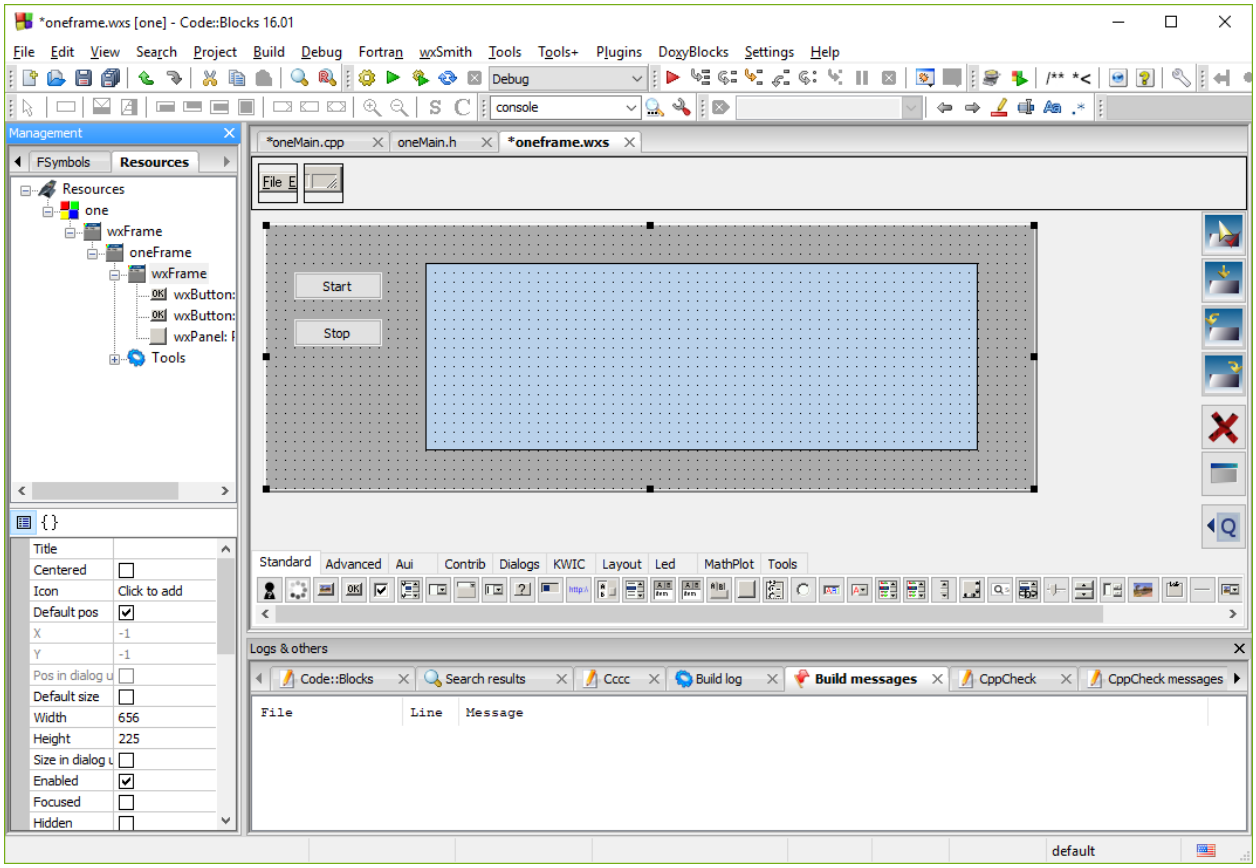


Рис. 5.9. Создание графического интерфейса

После создания заготовки следует добавить два файла из набора serialib: serialib.cpp и serialib.h. Для этого в окне менеджера проекта следует перейти на закладку *Projects* (в верхней части окна), выделить имя проекта правой клавишей мышки, и из выпадающего меню выбрать раздел добавления файлов (*Add files…)*. Можно сразу выделить оба файла и вставить их в проект.

Я не буду, как в первом случае, рассказывать о проекте, а приведу текст программы. В простых проектах, я думаю, только в этом месте программы приходится добавлять что-то.

```
/***************************************************************
 * Name:      osc_graphMain.cpp
 * Purpose:   Code for Application Frame
 * Author:    Vladimir ()
 * Created:   2017-04-24
 * Copyright: Vladimir ()
 * License:
 **************************************************************/

#include "osc_graphMain.h"
#include <wx/msgdlg.h>

//(*InternalHeaders(osc_graphFrame)
#include <wx/settings.h>
#include <wx/intl.h>
#include <wx/string.h>
//*)
#include <wx/graphics.h>
#include <wx/dcclient.h>

#include "serialib.h"
#define  DEVICE_PORT  "/dev/ttyACM0"
serialib LS;


char dat[1024]; // буфер данных
int i; // вспомогательная переменная
bool f = true; // вспомогательный флаг

//helper functions
enum wxbuildinfoformat {
    short_f, long_f };

wxString wxbuildinfo(wxbuildinfoformat format)
{
    wxString wxbuild(wxVERSION_STRING);

    if (format == long_f )
    {
#if defined(__WXMSW__)
        wxbuild << _T("-Windows");
#elif defined(__UNIX__)
        wxbuild << _T("-Linux");
#endif
```

```
#if wxUSE_UNICODE
        wxbuild << _T("-Unicode build");
#else
        wxbuild << _T("-ANSI build");
#endif // wxUSE_UNICODE
    }

    return wxbuild;
}

//(*IdInit(osc_graphFrame)
const long osc_graphFrame::ID_BUTTON1 = wxNewId();
const long osc_graphFrame::ID_BUTTON2 = wxNewId();
const long osc_graphFrame::ID_PANEL1 = wxNewId();
const long osc_graphFrame::idMenuQuit = wxNewId();
const long osc_graphFrame::idMenuAbout = wxNewId();
const long osc_graphFrame::ID_STATUSBAR1 = wxNewId();
//*)

BEGIN_EVENT_TABLE(osc_graphFrame,wxFrame)
    //(*EventTable(osc_graphFrame)
    //*)
END_EVENT_TABLE()

osc_graphFrame::osc_graphFrame(wxWindow* parent,wxWindowID id)
{
    //(*Initialize(osc_graphFrame)
    wxMenuItem* MenuItem2;
    wxMenuItem* MenuItem1;
    wxMenu* Menu1;
    wxMenuBar* MenuBar1;
    wxMenu* Menu2;

    Create(parent, id, wxEmptyString, wxDefaultPosition, wxDefaultSize,
wxDEFAULT_FRAME_STYLE, _T("id"));
    SetClientSize(wxSize(965,311));
    Button1 = new wxButton(this, ID_BUTTON1, _("Start"), wxPoint(24,40),
wxDefaultSize, 0, wxDefaultValidator, _T("ID_BUTTON1"));
    Button2 = new wxButton(this, ID_BUTTON2, _("Stop"), wxPoint(24,104),
wxDefaultSize, 0, wxDefaultValidator, _T("ID_BUTTON2"));
    Panel1 = new wxPanel(this, ID_PANEL1, wxPoint(144,40), wxSize(792,224),
wxTAB_TRAVERSAL, _T("ID_PANEL1"));
    Panel1-
>SetBackgroundColour(wxSystemSettings::GetColour(wxSYS_COLOUR_MENUHILIGHT));
    MenuBar1 = new wxMenuBar();
    Menu1 = new wxMenu();
    MenuItem1 = new wxMenuItem(Menu1, idMenuQuit, _("Quit\tAlt-F4"), _("Quit
the application"), wxITEM_NORMAL);
    Menu1->Append(MenuItem1);
    MenuBar1->Append(Menu1, _("&File"));
    Menu2 = new wxMenu();
    MenuItem2 = new wxMenuItem(Menu2, idMenuAbout, _("About\tF1"), _("Show
info about this application"), wxITEM_NORMAL);
    Menu2->Append(MenuItem2);
```

```
    MenuBar1->Append(Menu2, _("Help"));
    SetMenuBar(MenuBar1);
    StatusBar1     =     new     wxStatusBar(this,     ID_STATUSBAR1,     0,
_T("ID_STATUSBAR1"));
    int __wxStatusBarWidths_1[1] = { -1 };
    int __wxStatusBarStyles_1[1] = { wxSB_NORMAL };
    StatusBar1->SetFieldsCount(1,__wxStatusBarWidths_1);
    StatusBar1->SetStatusStyles(1,__wxStatusBarStyles_1);
    SetStatusBar(StatusBar1);


Connect(ID_BUTTON1,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction)&osc_g
raphFrame::OnButton1Click);

Connect(ID_BUTTON2,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction)&osc_g
raphFrame::OnButton2Click);
    Panel1-
>Connect(wxEVT_PAINT,(wxObjectEventFunction)&osc_graphFrame::OnPanel1Paint,0,
this);

Connect(idMenuQuit,wxEVT_COMMAND_MENU_SELECTED,(wxObjectEventFunction)&osc_gr
aphFrame::OnQuit);

Connect(idMenuAbout,wxEVT_COMMAND_MENU_SELECTED,(wxObjectEventFunction)&osc_g
raphFrame::OnAbout);
    //*)
}

osc_graphFrame::~osc_graphFrame()
{
    //(*Destroy(osc_graphFrame)
    //*)
}

void osc_graphFrame::OnQuit(wxCommandEvent& event)
{
    Close();
}

void osc_graphFrame::OnAbout(wxCommandEvent& event)
{
    wxString msg = wxbuildinfo(long_f);
    wxMessageBox(msg, _("Welcome to..."));
}

// Включаем процесс работы
void osc_graphFrame::OnButton1Click(wxCommandEvent& event)
{
         f = false;
            LS.Open(DEVICE_PORT, 9600);
            wxPaintDC dc(this);
         Refresh();
}
```

```
// Выключаем процесс работы
void osc_graphFrame::OnButton2Click(wxCommandEvent& event)
{
    LS.Close();
    f = true;
}

// Рисуем график по данным от АЦП
void osc_graphFrame::OnPanel1Paint(wxPaintEvent& event)
{
            char wrem1;
            char wrem2;
      wxPaintDC dc( Panel1 );

            if (f == false) {

            LS.ReadChar (&wrem1,0);
            LS.ReadChar (&wrem2,0);

            while (wrem1 > wrem2) {
            LS.ReadChar (&wrem1,0);
            LS.ReadChar (&wrem2,0);
                  }

            LS.ReadChar (&wrem1,0);


                  while ((wrem1 < 80) || (wrem1 > 100)) {
            LS.ReadChar (&wrem1,0);
                  }


                  for (i = 0;i < 1024;i++) {
                        LS.ReadChar (&dat[i],0);
                  }

            dc.SetPen( wxPen( *wxRED, 2 ) );

                  for (i = 0;i < 256;i++) {
     dc.DrawLine( i*3, 180 - dat[i], (i + 1) * 3, 180 - dat[i + 1] );
                  }
               Refresh();

                  for (i = 255;i < 512;i++) {
     dc.DrawLine( i*3, 180 - dat[i], (i + 1) * 3, 180 - dat[i + 1] );
                  }
               Refresh();

                  for (i = 511;i < 767;i++) {
     dc.DrawLine( i*3, 180 - dat[i], (i + 1) * 3, 180 - dat[i + 1] );
                  }
               Refresh();

                  for (i = 766;i < 1024;i++) {
```

```
            dc.DrawLine( i*3, 180 - dat[i], (i + 1) * 3, 180 - dat[i + 1] );
                    }
                Refresh();
            }
}
```

Результат можно проверить в режиме отладки, а позже можно запустить сборку в режиме *Release*, что приведёт к созданию исполняемого файла. Я использую генератор на частоте 50 кГц со следующими настройками:

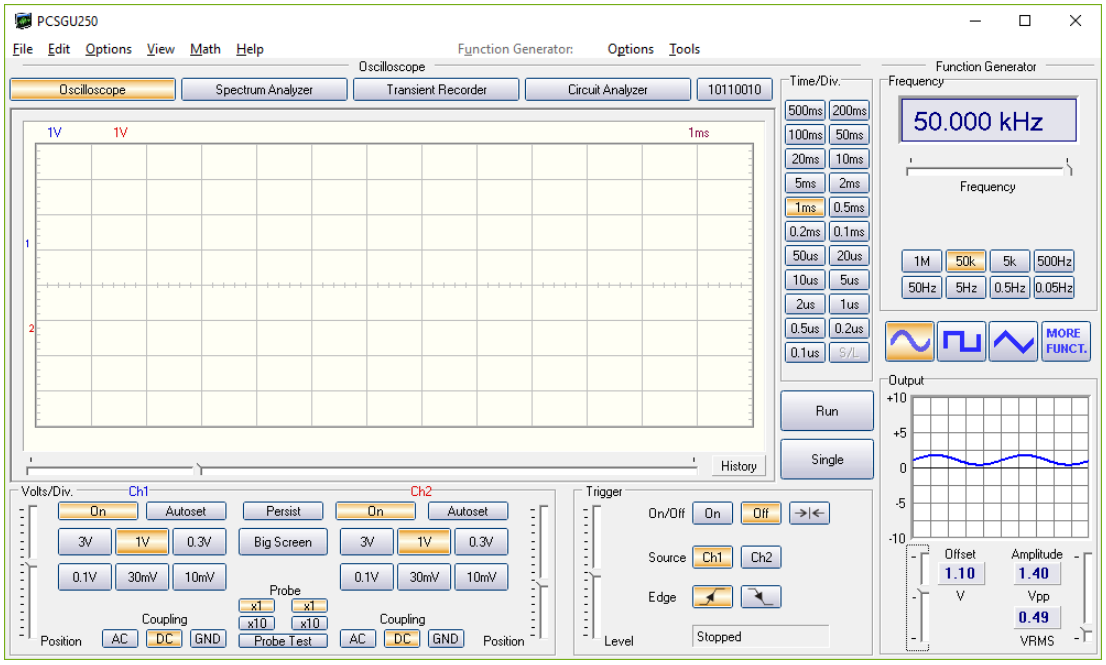


Рис. 5.10. Настройка генератора

В основном это сводится к тому, чтобы на вход PA0 модуля STM32F103C8 попадало только положительное напряжение с амплитудой, которая допустима для АЦП.

Если говорить о реальном проекте, то потребовалось бы, скорее всего, добавлять конденсатор и напряжение смещения, чтобы, например, разнополярный сигнал переменного напряжения мог правильно восприниматься АЦП и правильно отображаться на графике. Впрочем, для реального устройства пришлось бы озаботиться и добавлением делителя напряжения на плату. Пришлось бы внести изменения и в графический интерфейс добавлением шкалы, а ещё лучше сетки к графической панели, добавления переключения верхней рабочей частоты (задача для микроконтроллера), и много другого, включая возможность прочитать сделанную в процессе наблюдения запись данных сигнала. Но в данном случае это лишнее.

Вид работы программы, исполняемого файла, в Linux ROSE такой:

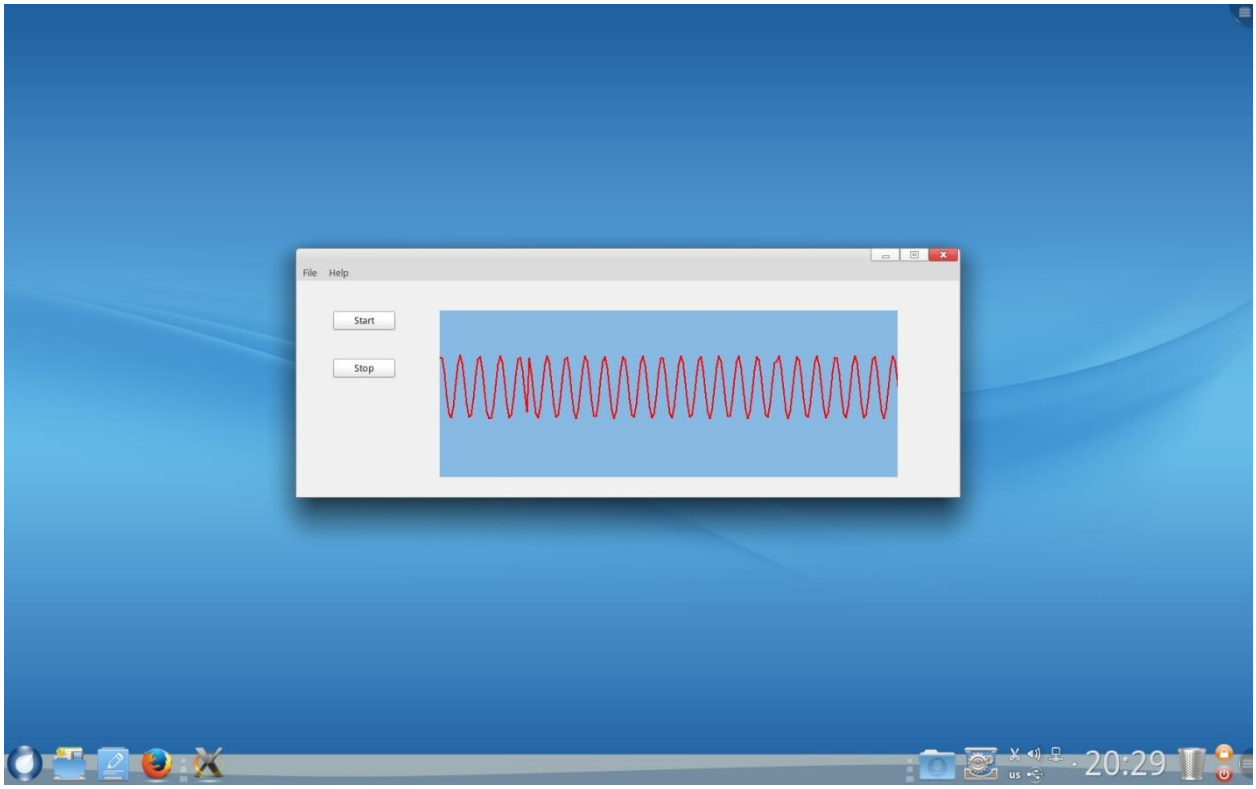


Рис. 5.11. Работающая программа с подключённым модулем STM32

Меня, мягко говоря, работа программы меня не удовлетворяет. Не удовлетворяет вид сигнала. Это так. Но я собирался рассказать о том, что в операционной системе Linux ROSE можно создать удобную среду для работы с STM32. Более того, в одной и той же программе Code::Blocks можно работать и с arm- контроллером, и создавать графический интерфейс для взаимодействия с контроллером. Что я и сделал. Рассказ получился несколько «путаный», и я в какой-то момент, когда ничего не получалось, думал отказаться от него полностью. Однако получилось. И, надеюсь, кому-то он будет интересен, а кому-то поможет.

Напомню, что про простые опыты с STM32 можно прочитать в моём рассказе «Несколько простых опытов с модулем STM32F103C8». А о программе Code::Blocs в трёх рассказах «О переводе одного руководства». Всё это есть на моём сайте для свободного скачивания и свободного использования.