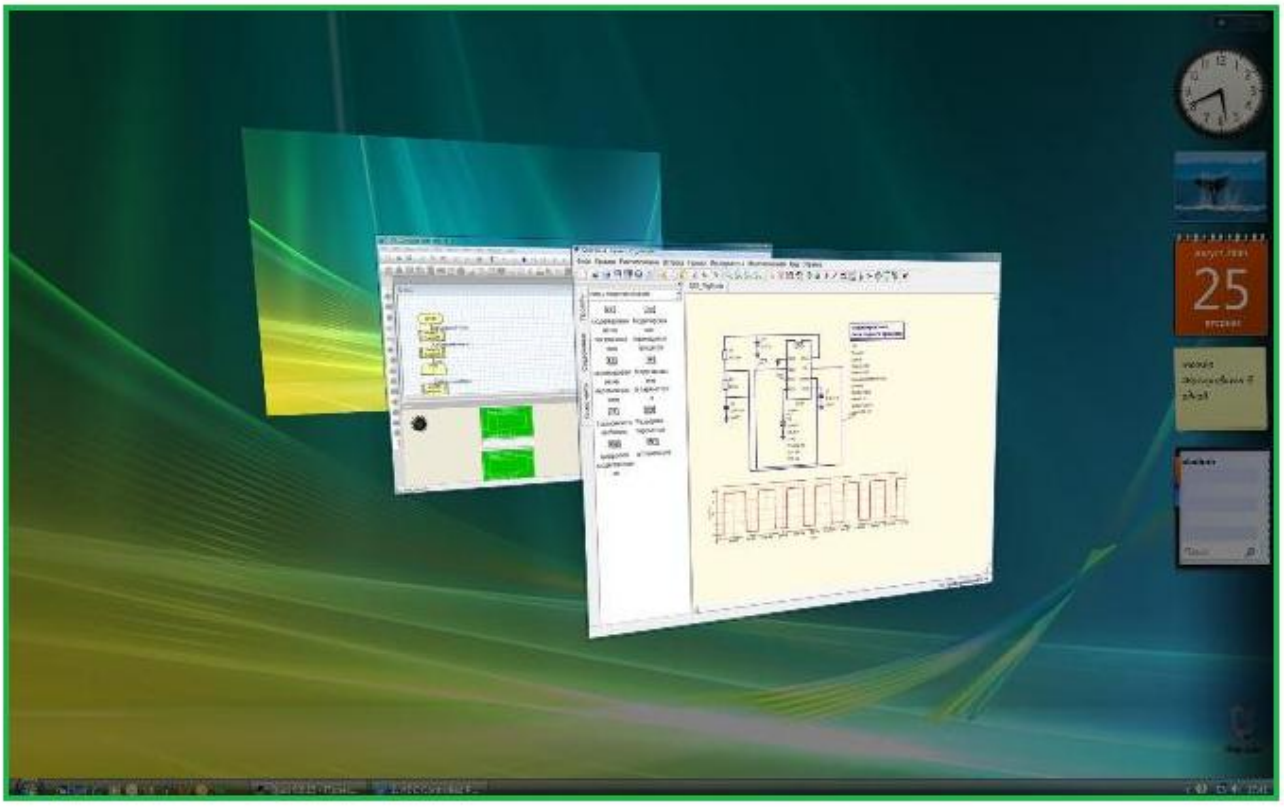
В.Н. Гололобов

Qucs и FlowCode. Программы для тех, кто интересуется электроникой



Москва 2009

Оглавление

Книга 1. Qucs – почти универсальный симулятор электрических схем

Предисловие

Многие учебные заведения в последние годы с успехом используют в учебном процессе компьютерные программы. Не исключение и институты или колледжи радиоэлектронной направленности. Такие программы, как Multisim или MicroCAP, позволяют студентам и соединить теоретические знания с практическими примерами построения электрических схем, и увидеть влияние различных факторов на поведение будущих электронных устройств, и научиться разработке этих устройств в форме близкой к той, что ждет их после окончания учебного заведения.

Программы САПР (или EDA) полезны преподавателям при подготовке методических материалов; они используются в научной работе; они очень удобны при написании статей и рефератов, относящихся к электронике.

Хотя можно предположить, что эти программы из инструментария разработчиков электроники, они с не меньшим успехом могут использоваться теми профессионалами, которые не связаны непосредственно с разработкой новых устройств. А таких специалистов гораздо больше. Для одних программы САПР хорошие помощники в определении возникающих проблем при эксплуатации оборудования, другим помогут при создании нестандартного оборудования, нужда в котором всегда есть у любого предприятия.

И, наконец, есть категория людей особенно склонных к любым инновациям в этой области – это радиолюбители. Для одних из них радиолюбительство отдых от работы, для других, возможно, предтеча будущей профессии.

Существует достаточно много программ или сред компьютерной разработки электроники; ничего плохого о них сказать нельзя, они развиваются и совершенствуются по мере развития и совершенствования самих компьютеров. Разработчики программного обеспечения стараются включить в состав своих пакетов и такие средства, как программы разводки печатных плат и подготовки их к производству. В последнее время в состав компонентов стараются добавить специализированные микросхемы, как, например, стабилизаторы напряжения или микроконтроллеры. Симуляторы электрических цепей, лежащие в основе таких программ, тоже совершенствуются, а проблемы, существующие в их работе, преодолеваются с применением разных математических и программных решений.

Однако во всем многообразии программ для изучения радиоэлектроники с применением компьютеров есть один элемент, на который все чаще обращают внимание многие учебные заведения. Это стоимость программ. Даже с теми скидками, которые предоставляются учебным заведениям, использование компьютеров, а как их не использовать? – использование компьютеров становится слишком дорогостоящим.

Но и здесь есть неплохой выход – применение свободно распространяемого программного обеспечения, включая и операционную систему. Все разговоры о трудностях перехода с Windows, а я не думаю, что в учебных заведениях используют другие операционные системы, на любой из дистрибутивов Linux, скорее, надуманы. Программа Qucs существует в версиях для всех общеупотребительных операционных систем: Windows, Linux, MacOS. Программа может использоваться и распространяться бесплатно в некоммерческих целях. Все аргументы, если бесплатное, значит плохое, имеют сомнительное происхождение. Достаточно напомнить, сколько неприятностей принесло применение Windows покупателям этой операционной системы.

Qucs достаточно неприхотливая программа по нынешним временам. Если на компьютер устанавливается Windows 2000, значит, и программа будет работать. Требования к компьютеру в основном определялись тем фактом, что сам проект появился не так давно, когда основная масса компьютеров использовала операционную систему Windows XP, с теми требованиями, которые к компьютеру предъявляла эта операционная система. Хотя симуляция некоторых схем может с трудом проходить и на вполне современном компьютере, но это уже другой вопрос. Таким образом, программу следует установить и опробовать, если вы собираетесь с ней работать. Для симуляции цифровых устройств вам понадобится дополнительно установить еще несколько свободно распространяемых программ. Одной из особенностей программы, которую следует отметить, является наличие в составе базовых компонентов множества таких, которые ориентированы на применение в современных радиотехнических устройствах. Подобные компоненты редко встречаются в других программах общего применения. Но это никак не влияет на возможности использовать программу в изучении любых других электрических цепей.

Как многие проекты свободного программного обеспечения, Qucs постоянно развивается, и каждая новая версия доступна для свободного использования, достаточно иметь доступ к Интернету, чтобы получить новую версию. И подобно всем программам с открытым кодом, Qucs для серьезных пользователей, умеющих программировать, дает возможность полностью переделать всю программу под свои нужды или интересы.

В отличие от многих аналогов программа полностью русифицирована при участии ее создателей, что избавляет от случайных ошибок или появления проблем, связанных с пользовательской русификации программы. В разделе «Справка» есть прекрасный раздел быстрого начала работы с Qucs. Опытным пользователем его будет достаточно. На сайте проекта много документов, статей и примеров, восполняющих то, чего нет в справочной системе. И, наконец, эта книга для начинающих работать с Qucs должны помочь всем, кого интересуют программы этого направления, познакомиться с программой.

Часть 1. Начальные сведения

Глава 1. Основные элементы интерфейса

Вид программы

Внешний вид основного окна программы может зависеть от вашей операционной системы, но отличия не столь существенны. Однако вот этот вид в трех операционных системах.

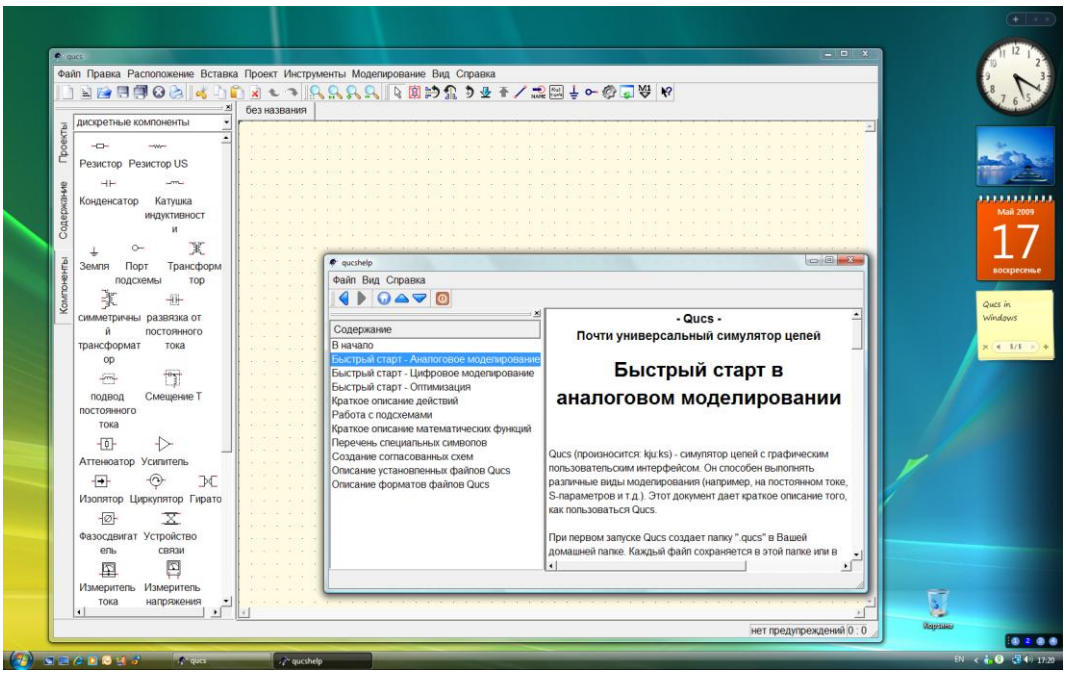


Рис. 1.1. Программа Qucs в операционной системе Vista

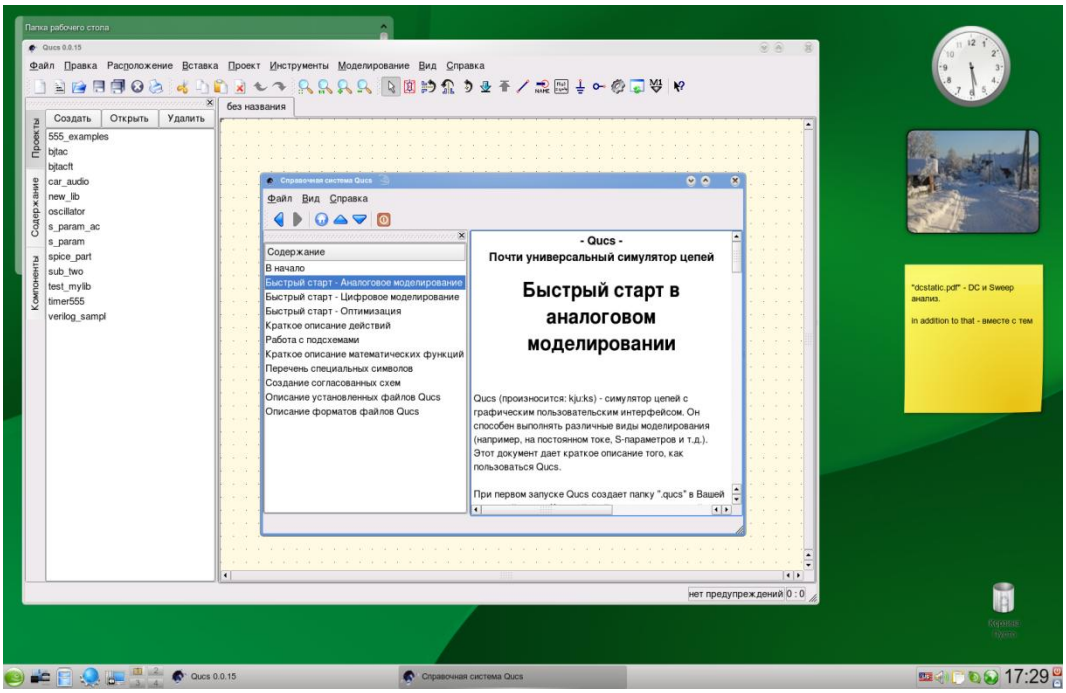


Рис. 1.2. Программа Qucs в операционной системе openSUSE 11.1

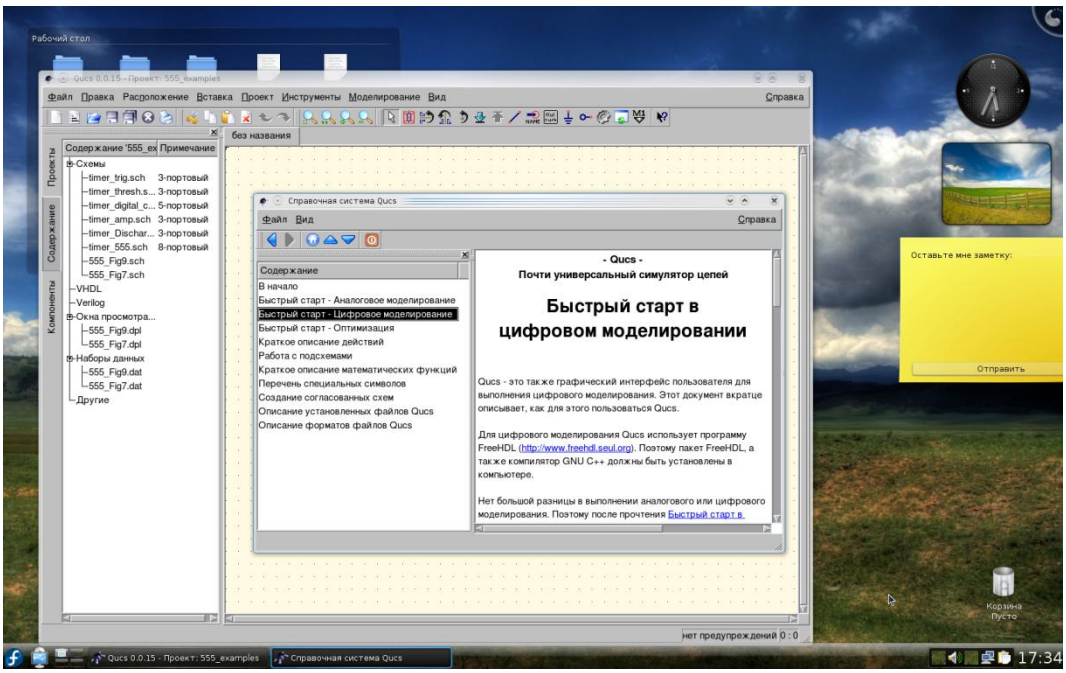


Рис. 1.3. Программа Qucs в операционной системе Fedora 10 с KDE 4.2

И для пользователей Windows, и для пользователей Linux программа появляется в том виде, к которому они привыкли, используя множество других программ.

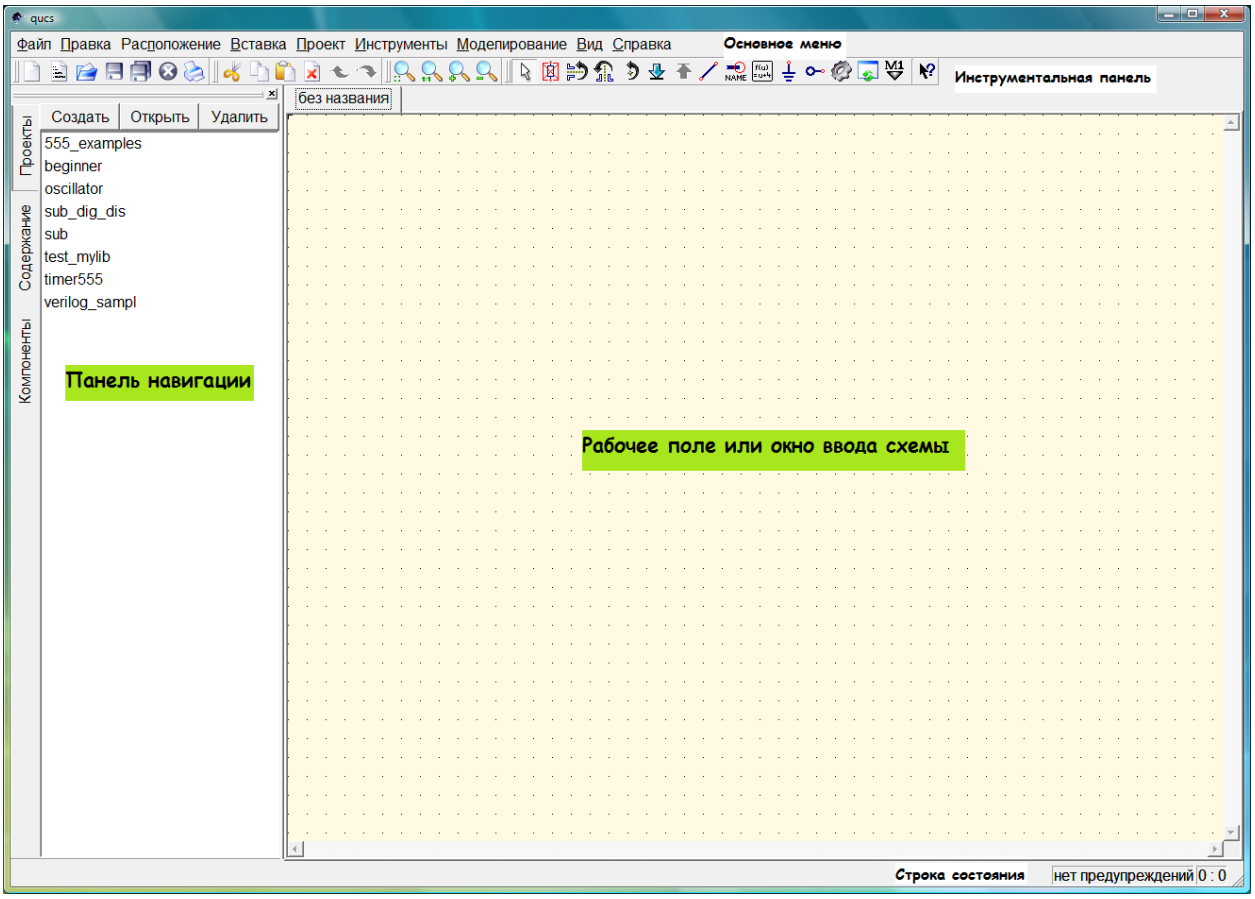


Рис. 1.4. Основное окно программы Qucs

Основное меню и инструментальная панель, как правило, выполняют одинаковые функции – выполняют команды программы. Но инструментальная панель выполняет эти команды нажатием одной кнопки, что существенно ускоряет работу для наиболее часто употребляемых команд. А основное меню при нажатии раздела открывает подменю, в котором содержатся все команды данного раздела. Кроме этих средств доступа к командам, как и другие программы, Qucs использует «горячие клавиши», выпадающие меню, зависящие от конкретного этапа работы, которые вызываются нажатием правой клавиши мышки; используются также двойной щелчок левой клавишей мышки для доступа к свойствам объектов и колесико мышки, которое в сочетании с клавишами клавиатуры позволяет перемещаться в рабочем поле программы. Последнее, в свою очередь, не исчерпывается видимой частью, а расширяется по мере необходимости, что важно для больших вводимых схем.

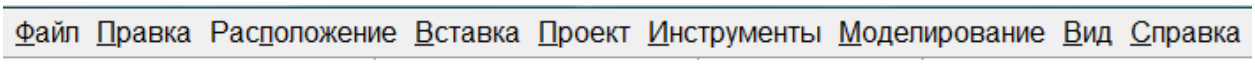


Рис. 1.5. Основное меню программы Qucs

Как можно понять, оно содержит общие для всех программ разделы, как **Файл** – для работы с файлами; такие разделы, как **Правка** – их можно найти в любом текстовом редакторе; и специфические разделы, как **Моделирование**. К содержанию этих разделов мы вернемся в следующей главе, а сейчас посмотрим на другие составляющие интерфейса пользователя.

Рис. 1.6. Инструментальная панель Qucs

Многие кнопки инструментального меню имеют понятный вид мало отличающийся, как и основное меню от других программ. Например, увеличительное стекло на кнопке, наверняка показывает, что с помощью этих кнопок можно управлять масштабом изображения.

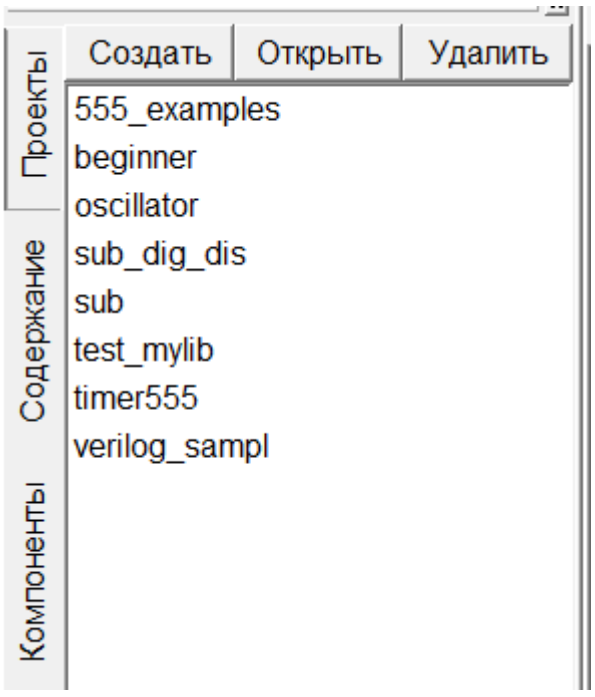
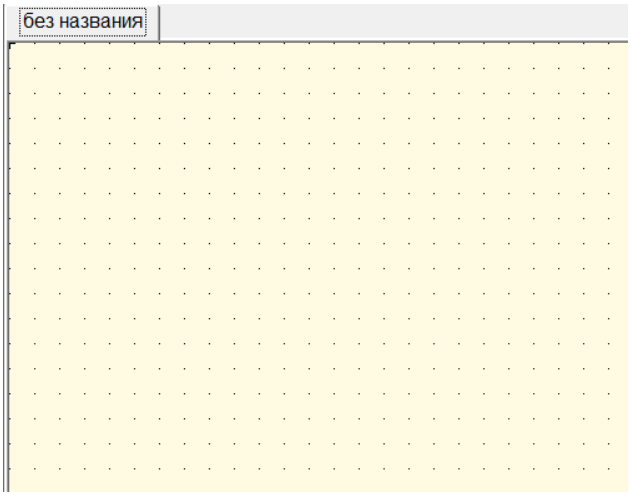


Рис. 1.7. Панель навигации программы Qucs

Панель навигации имеет вид, который зависит от закладки (в левой части), которая выбрана в данный момент.

Например, сейчас панель навигации открыта на закладке **Проекты**. В верхней части есть три кнопки команд: ***Создать***, ***Открыть*** и ***Удалить***. Команды относятся к проектам. Выделив проект *beginner* в списке доступных проектов, можно нажать кнопку ***Открыть***, после чего откроется закладка **Содержание** панели навигации, а в самой панели появятся все файлы этого проекта.

Чуть позже мы рассмотрим это подробнее.



Рабочее поле или окно ввода схемы – основное окно программы, в котором должна появиться схема (вашими усилиями), в которое будут добавлены и вид моделирования, и уравнения, если в последних есть необходимость.

Как и многие графические редакторы, рабочее поле имеет сетку. Параметры сетки могут быть изменены. И сетку можно выключить, если вы используете программу для получения рисунков.

Рис. 1.8. Рабочее поле программы

После заполнения рабочего поля нужными компонентами, их соединения в схему, добавления вида моделирования, ваше рабочее поле приобретает следующий, отметим не единственно возможный, вид:

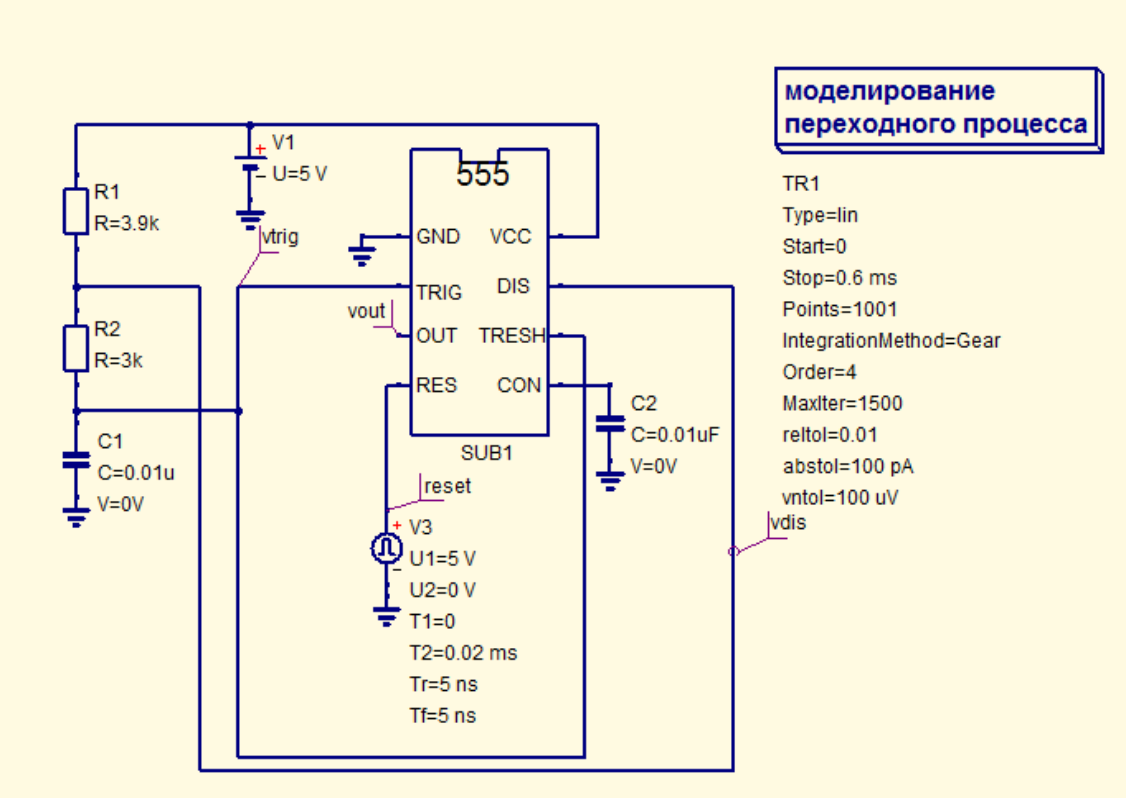


Рис. 1.9. Вид подготовленной к работе схемы в Qucs

После выполнения моделирования программа автоматически открывает новую страницу для графического отображения результатов. Но, если вам удобнее, вы можете скопировать график в основное рабочее поле. Если при этом убрать опцию в настройках документа, открывающую

автоматически страницу для графиков, все полученные изменения в характере работы схемы вы увидите в основном рабочем поле. А вид программа приобретет следующий:

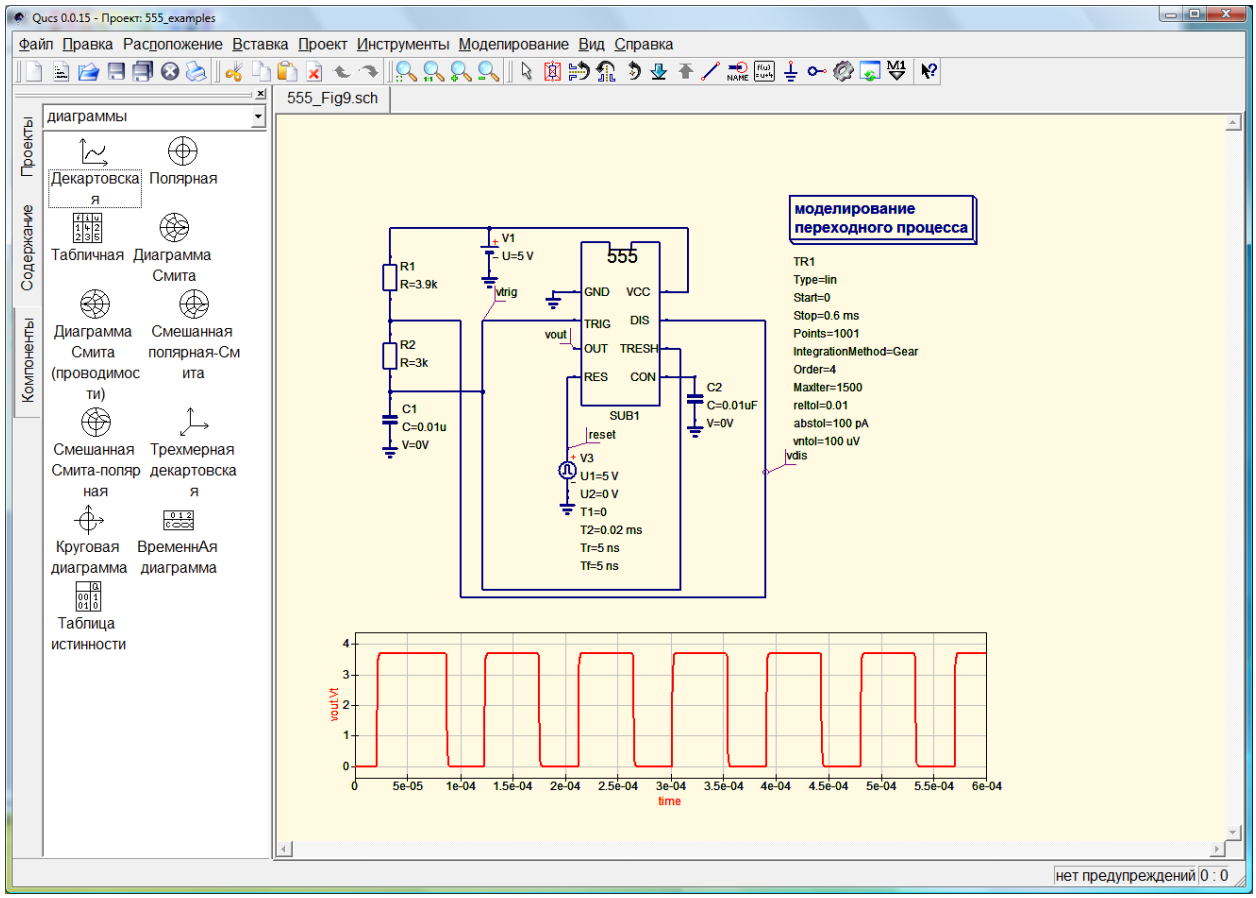


Рис. 1.10. Вид работающей программы Qucs

Подобно многим аналогичным программам Qucs использует графику для отображения сигналов, которые схема транслирует или производит. Но в отличие от многих других программ Qucs позволяет получить, например, графики зависимости малосигнальных параметров от частоты. Впрочем, и это тема следующих глав.

Но прежде, чем перейти к более детальному рассмотрению пользовательского интерфейса, несколько слов о том, как удобнее работать с Qucs. Программа позволяет создавать отдельные файлы схем и записывать их в любое место по вашему выбору. После моделирования схемы появятся файлы с данными и графикой. Одна-две сохраненных модификации схемы и разобраться с появившимися файлами становится трудно. Разумно создать папку, в которой и хранить все, что относится к схеме, а каждую из модификаций схемы хранить в своей папке.

Такой механизм работы со схемами в Qucs создает понятие проекта. Если для каждой из схем, включая модификации схемы, создавать свой проект, то разобраться в созданных файлах будет легче, поскольку при создании нового проекта для него автоматически создается папка. По умолчанию все проекты сохраняются в основной папке программы. Для Linux эта папка скрытая, ее имя начинается с точки «.qucs». Папка создается при первом запуске программы Qucs в домашней директории пользователя (например, путь к ней /home/valdimir/.qucs). В Linux домашняя директория пользователя выглядит так:

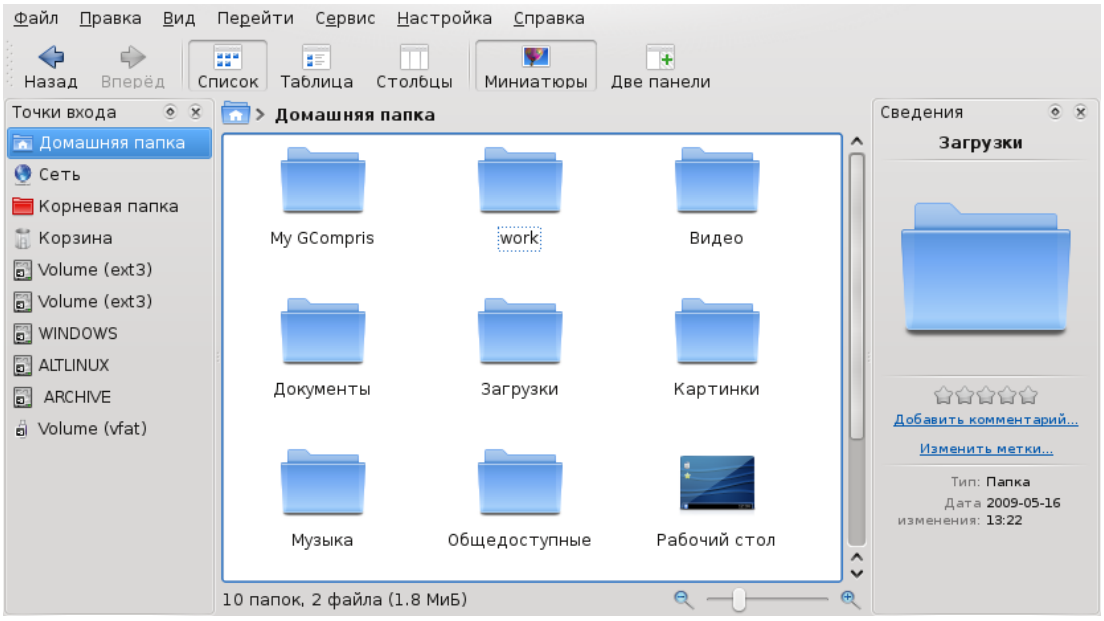


Рис. 1.11. Вид домашней директории пользователя в диспетчере файлов KDE

Чтобы увидеть папку с проектами Qucs, следует в основном меню диспетчера файлов выбрать пункт **Вид**, в котором установить опцию ***Показывать скрытые файлы***.

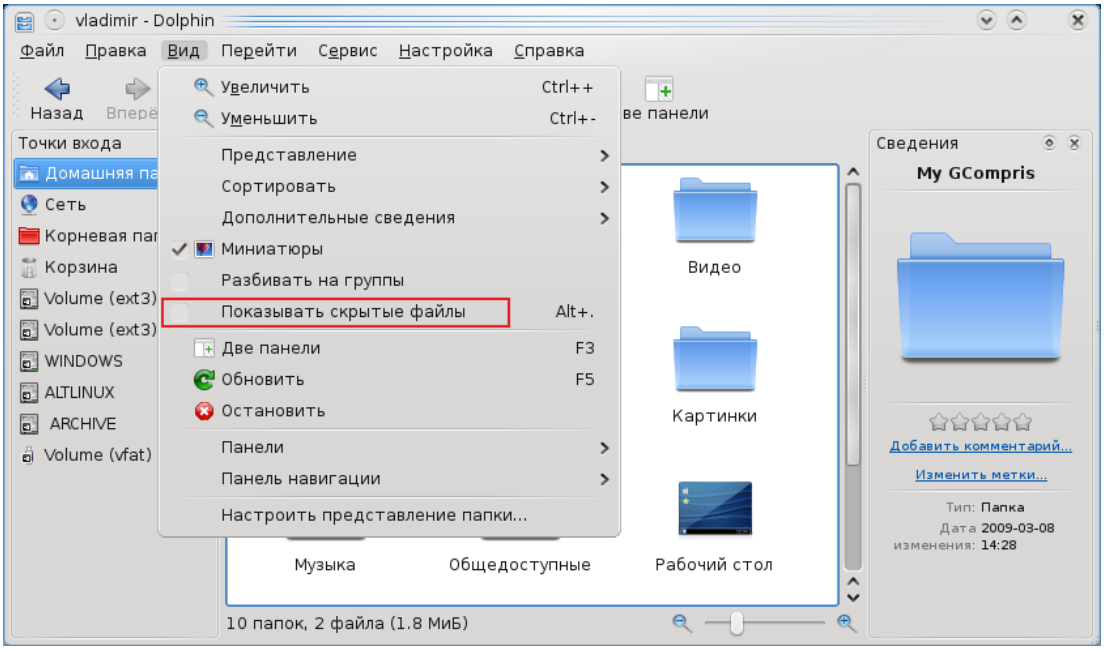


Рис. 1.12. Настройка диспетчера файлов для отображения скрытых файлов

После установки опции домашняя папка пользователя в Linux пополнится очень большим количеством папок – это папки настроек конфигурации многих программ. Подобные служебные файлы есть и в Windows. И они появляются в поле зрения пользователя только после соответствующей настройки проводника.

Среди скрытых папок и находится папка с проектами Qucs.

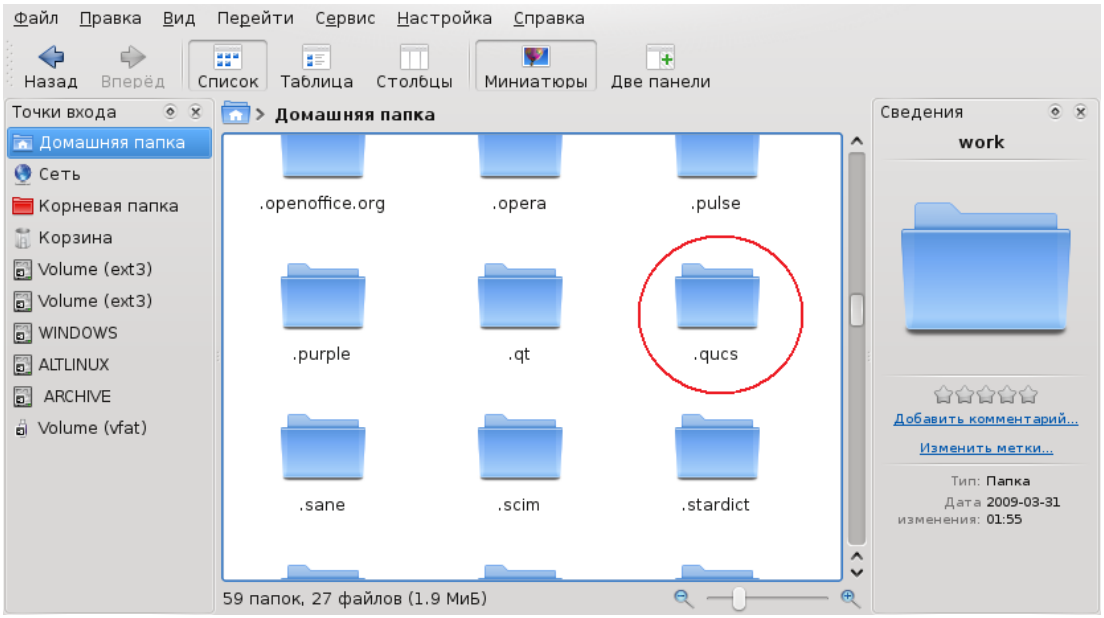


Рис. 1.13. Папка с проектами, создаваемая автоматически при первом запуске Qucs

Аналогично тому, как папка с проектами создается в Linux, она создается при первом запуске программы и в Windows. И создается в том же месте, то есть в папке пользователя, и создается с тем же именем «.qucs».

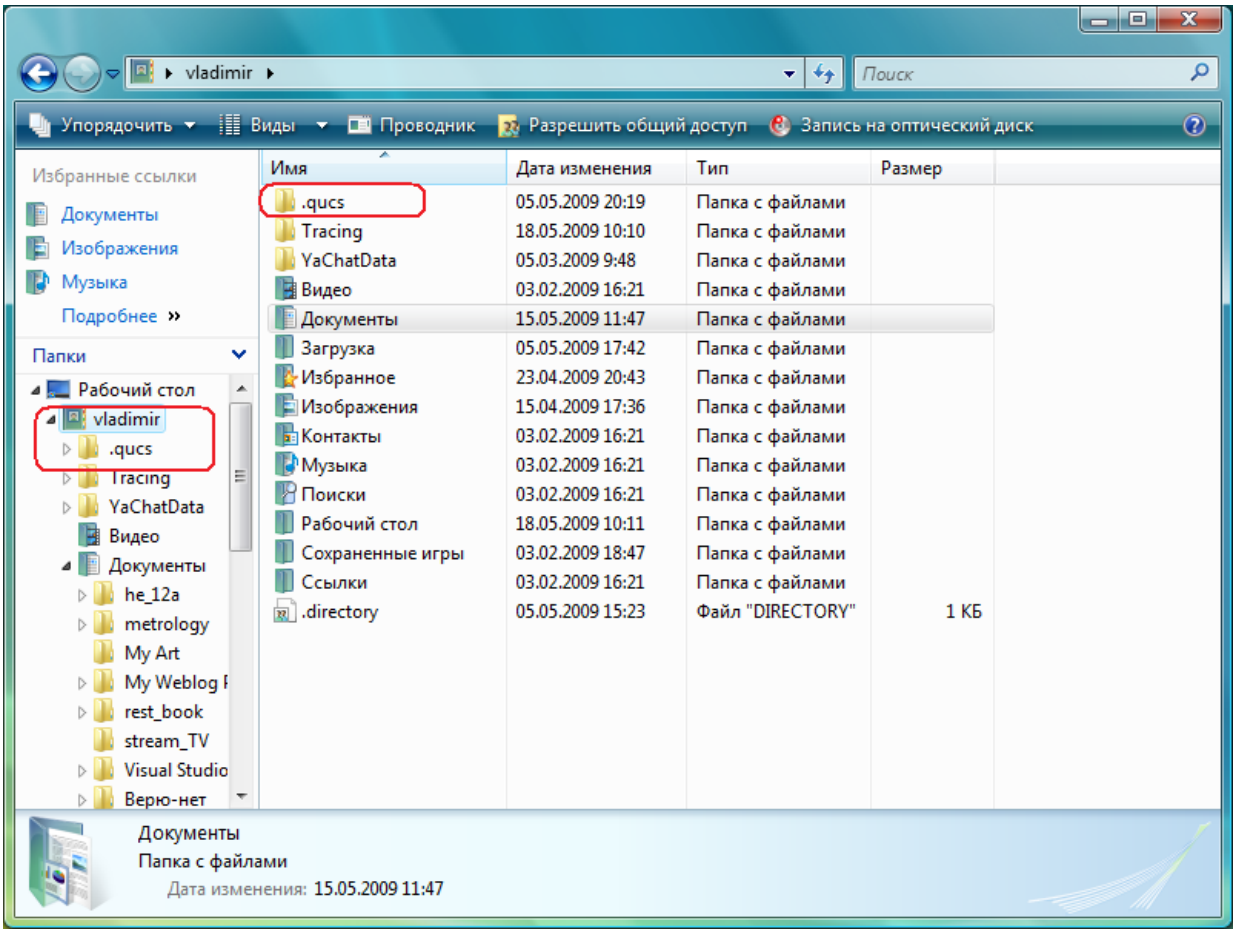


Рис. 1.14. Папка с проектами Qucs в Windows

Если вам по какой-то причине хочется поместить эту папку в другое место, то вы можете сделать это, используя переменную окружения *HOME*.

Как это сделать в среде Windows Vista будет показано ниже. Но это не столь сильно отличается от того, что потребуется от вас в Windows XP. Первое, что нужно сделать, открыть главное меню операционной системы, щелкнув по кнопке… раньше, в Windows XP, она называлась ***Пуск***, теперь это картинка с эмблемой Microsoft. В правой части меню есть пункт ***Компьютер***. Поместив на него курсор, щелкните правой клавишей мышки, а из выпадающего меню выберите пункт ***Свойства***. Это откроет окно свойств вашего компьютера:

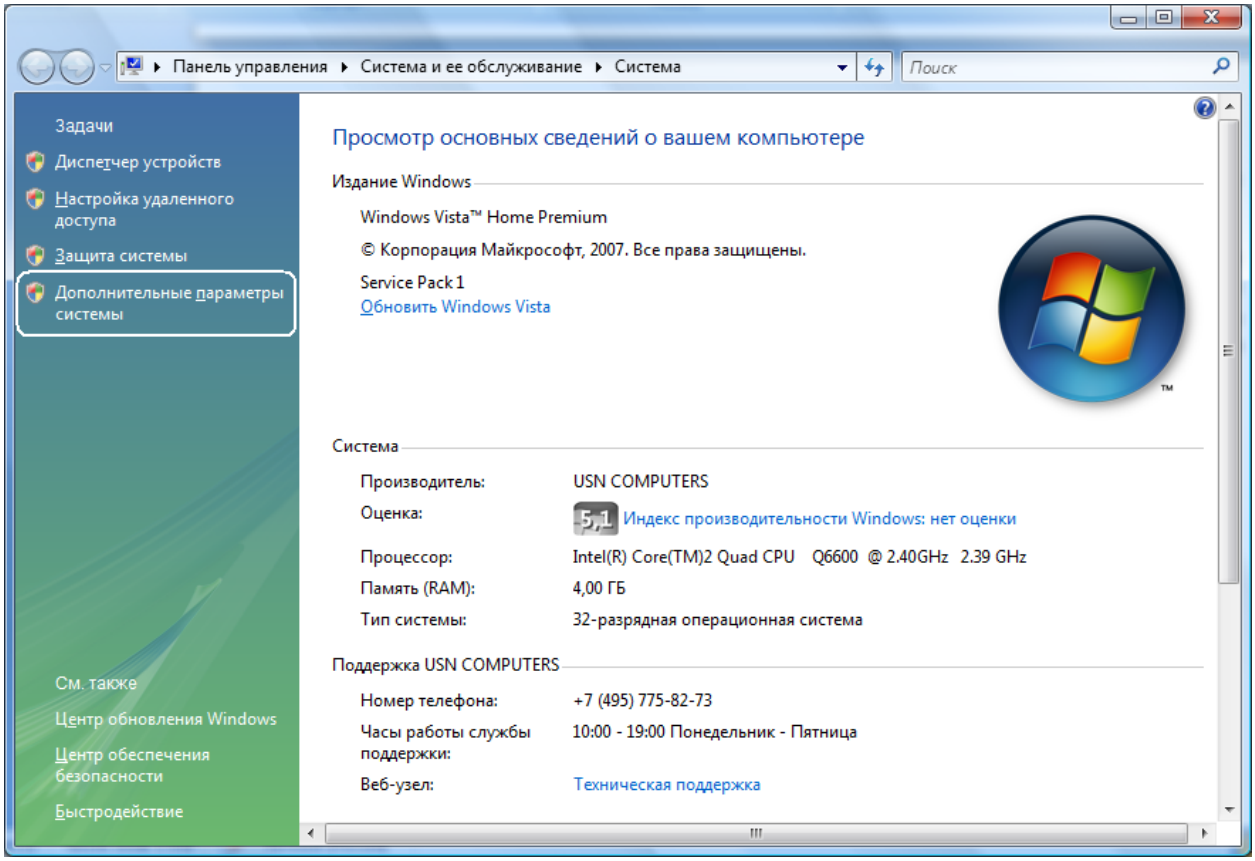


Рис. 1.15. Окно свойств компьютера в Vista

Кстати, значок с эмблемой Microsoft – это та самая «бывшая» кнопка ***Пуск***. Раздел диалогового окна, который нужен, отмечен в левом поле: **Дополнительные параметры системы**. В новом диалоговом окне следует воспользоваться кнопкой Переменные среды:

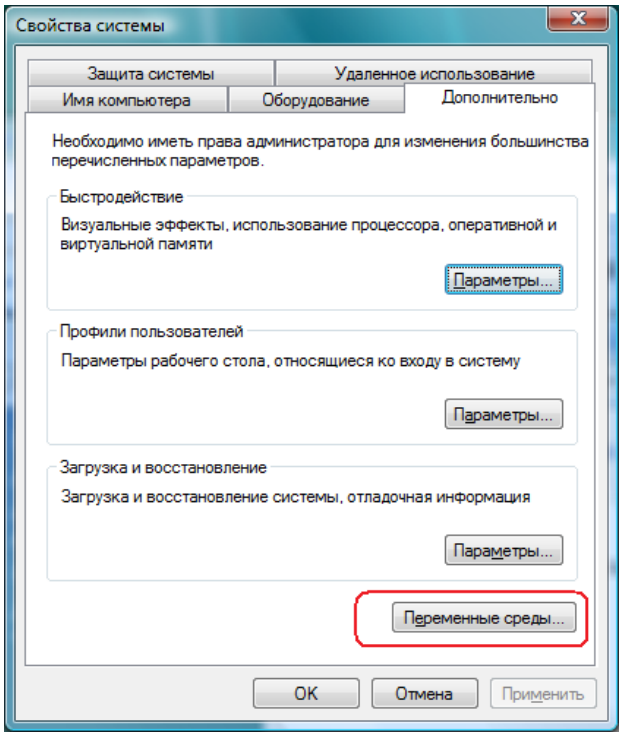


Рис. 1.16. Диалоговое окно свойств системы

После того, как откроется окно переменных среды, следует выделить переменную *HOME*, нажать кнопку ***Изменить***... и изменить путь, скажем, добавив «Documents» в поле значения переменной. Когда в следующий раз вы создадите новый проект в Qucs, вы найдете его в новом месте.

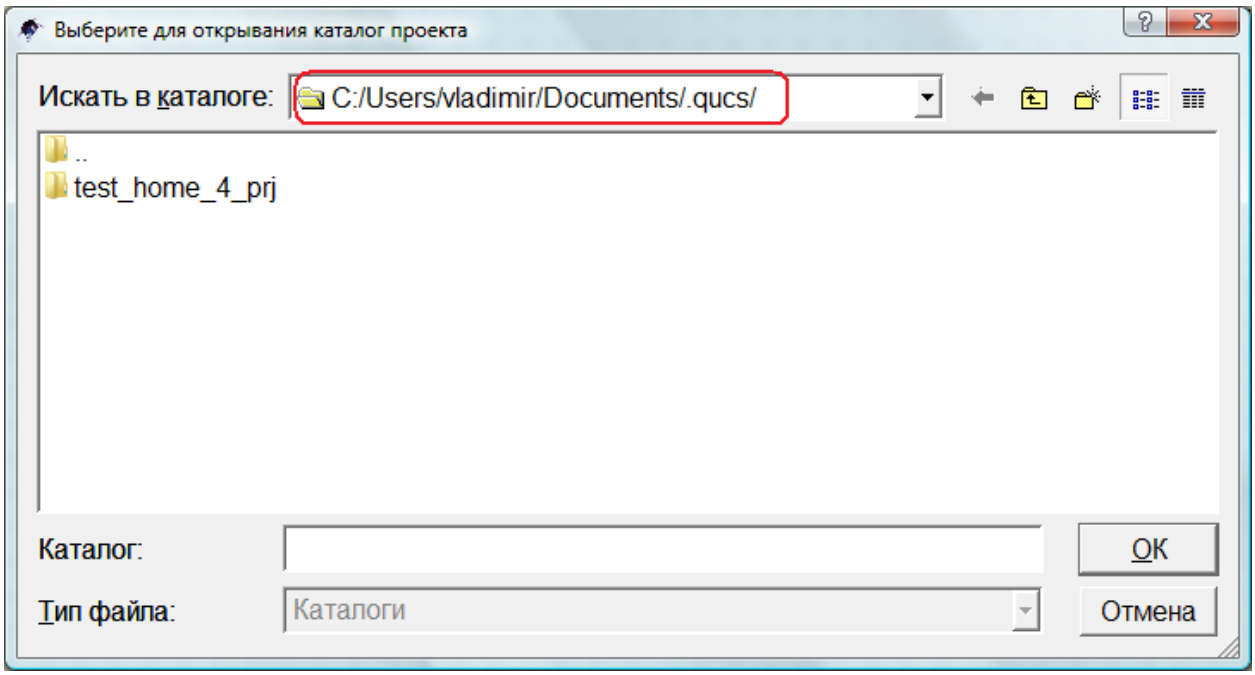


Рис. 1.17. Новое место расположения папки Qucs после изменения маршрута

Программа Qucs позволяет сделать эти изменения, но следует ли их делать?

Если вы достаточно опытный пользователь, если готовы к неожиданностям при работе других программ, то вам не составит труда перенести папку проектов Qucs. Однако не забывайте, что, например, системная папка «Мои документы» из-за того, что ее имя написано не латиницей, а

кириллицей, из-за пробела между двумя словами, может помешать работе компилятора или программе FreeHDL, когда вы будете выполнять цифровое моделирование. А другие программы, чувствительные к написанию пути к файлу, вообще откажутся работать. Появление сообщения, что программа не может найти указанный вами файл, поможет вам заподозрить причину этого явления, но программа может по умолчанию предлагать вам запись файла в папку «Мои документы», а при попытках продолжить работу давать ошибку с очень непонятными свойствами.

Самое разумное, особенно для начинающих, оставить все без изменений и использовать названия латиницей для файлов проектов. Такая привычка не будет лишней.

Для полного использования возможностей программы потребуется дополнить ее еще несколькими свободно распространяемыми: MinGW32 (для Windows), FreeHDL, Icarus Verilog, ASCO. Все программы для Windows устанавливаются обычным образом. Эти же программы для Linux могут устанавливаться, как и остальные, двояко – из установочного пакета, если он есть для конкретного дистрибутива, либо из исходных файлов, как и сама программа Qucs.

При установке программ можно столкнуться с некоторыми неожиданностями. Так для Windows и Qucs, и остальные программы проверялись и работают на сегодняшний день в версии от Windows 98 до XP. Мне довелось устанавливать программу на операционную систему Windows Vista. Сама программа установилась и заработала без каких-либо особенностей. А с поддержкой компонента программы **Файл Verilog**, о нем речь пойдет в следующих главах, не заладилось. Программа Icarus Verilog устанавливалась без каких-либо претензий, а при проверке работы компонента моделирование обрывалось, появлялось сообщение, которое не получалось прочитать из-за неверной кодировки символов. После небольшой работы по перекодировке сообщения выяснилось, что: «iverilog» не является внутренней или внешней командой, исполняемой программой или пакетным файлом». Это сообщение генерировалось из DOS (если есть DOS в Vista), поскольку программа *iverilog* выполняется в командной строке, а вызывается из программы Qucs. Получается, что программа просто не была зафиксирована операционной системой, как выполняемая программа. Проблему удалось решить, быть может, не самым лучшим образом, добавив в пакетный файл из состава Qucs с названием *qucsveri.bat* одну строку:

```
set VERILOG=C:\IcarusVerilog\bin\
```

и подправив две другие:

```
%VERILOG%iverilog -o%NAME%.bin -sTestBench %NAME%.v
```

```
%VERILOG%vvp %NAME%.bin -vcd
```

При этом, как ни странно для тех, кто привык пользоваться блокнотом, править файл удобнее было в WordPad. Файл открывается и в Qucs, но нужны права администратора для правки пакетного файла. Расположен он в Program Files\Qucs\bin.

Похожая история случилась с дистрибутивом Linux openSUSE 11.1, в котором не удалось найти готовых пакетов для типовой установки. Правда, пакеты из состава Fedora вполне удачно устанавливались, но, увы, если сама программа Qucs работала, и даже работала поддержка файлов Verilog, но не работала поддержка моделирования цифровых устройств. Не то, что мне очень было нужно, чтобы все работало, нет. Но, собираясь написать эту книгу, я решил, хотя бы в первом приближении, навести порядок с «неполадками». Несколько попыток установить разные

версии и программы Qucs, и программы FreeHDL, не увенчались успехом, а появляющееся сообщение об ошибке не удалось прочитать, несмотря на все ухищрения. Более удачной оказалась попытка скомпилировать и установить последние версии на сегодня Qucs и FreeHDL из исходных файлов. Моделирование не проходило, но сообщение на английском языке гласило, что следует добавить «--tag». Конечно, следовало бы задать вопрос создателям программы, куда добавить эту опцию, и как это сделать? но… Оказалось, что добавить следует в файл, подобный предыдущему пакетному файлу, с именем *qucsdigi*:

```
echo -n "linking..."
  $LIBTOOL --quiet --mode=link --tag=digi $CXX $NAME._main_.o $NAME.o $LIBS
```

Файл можно найти по адресу: /usr/bin.

При этом при симуляции цифровых схем появляется сообщение об ошибке, а в строке состояния есть предупреждение об этом, но сама симуляция, игнорируя файл *digi*, указанный мной только для того, чтобы заполнить параметр, сама симуляция проходит. Я думаю, что подобные мелкие неприятности не омрачат жизнь пользователей программы. Для начинающих самым разумным было бы не попадать в эти ситуации, используя штатные программы своей операционной системы.

Думаю, поклонники Windows и коммерческого программного обеспечения к ней могут сказать: «Вот, вот цена бесплатного «сыра» в мышеловке свободного ПО!». Не хочу им возражать, но скажу, что я не учился всем этим «копаниям вглубь проблем» специально. Весь опыт получен при работе с предыдущими версиями Windows и программами для этой операционной системы. Я давно пользуюсь Linux, но время от времени возвращаюсь к Windows, скорее из признательности и уважения к тем программистам, что создавали и эту операционную систему, и программы для нее. У меня нет неуважительного отношения к работе других людей, и, когда я говорю, что есть возможность не использовать «пиратские копии», я имею в виду только это – уважение к чужому труду. Если те же профессионалы хотят бесплатно отдавать плоды своего труда, они создают прекрасное свободное программное обеспечение.

Однако вернемся к теме рассказа. О том, как использовать компоненты, о которых речь шла выше, будет рассказано в дальнейшем. А сейчас следует просто понять, что высокая функциональность программ достигается их усложнением, а это требует некоторых усилий и от пользователей, хотя бы на то, чтобы прочитать руководство к программе. Еще лучше, а сегодня у многих есть возможность обратиться к Интернету, зайти на сайт проекта и прочитать если не все, то многое из того, что там есть. Да, проект международный, а сайт англоязычный, но усилия окупятся впоследствии. Часть документов, немного устаревшая, есть в переводе на моем сайте:

```
http://vgololobov.narod.ru
```

Надеюсь, что, если эта книга будет дописана, пусть она и повторяет руководство пользователя, и она поможет освоиться с работой в программе Qucs.

Основное меню, Файл

Рассмотрим подробнее состав основного меню программы, перемещаясь по нему слева-направо, и рассматривая подменю каждого его пункта. Вызов подменю происходит, как и у большинства программ, с помощью щелчка левой клавишей мышки по этому пункту основного меню или с помощью клавиатуры. В данном случае она должна быть в режиме русскоязычной раскладки, следует нажать и удержать клавишу ***Alt*** и нажать клавишу ***Ф***. Такой механизм (мышка и горячие клавиши) используется многими программами. Первым пунктом в очереди, как у большинства программ, будет:

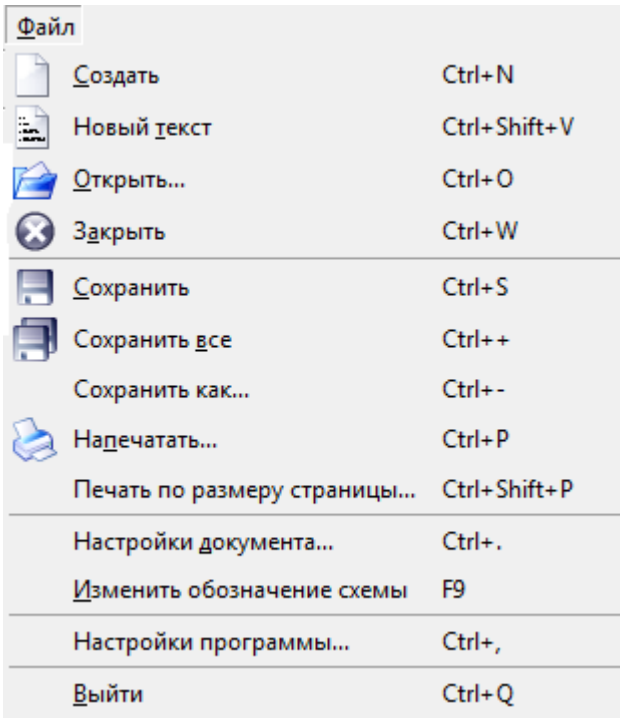


Рис. 2.1. Подменю пункта Файл основного меню

В подменю первый раздел (или команда) – **Создать**. Относится это к созданию нового файла схемы.

Еще раз напомню, что лучше создавать новый проект, в папке которого держать все файлы.

При создании нового проекта файл схемы открывается автоматически, но появляется «без названия», о чем гласит закладка файла. И для опытных, и для новичков, полезно придерживаться правила – сохранить файл с понятным для себя названием, используя раздел подменю **Сохранить как…** И по мере черчения схемы, время от времени, повторять сохранение. Как бы вы ни доверяли компьютеру, жаль будет потерять несколько часов работы.

Этой командой подменю пункта **Файл** основного меню вы будете пользоваться, если ваша схема состоит из нескольких файлов подсхем, будете вы или нет пользоваться механизмом подсхем Qucs, о чем будет рассказано позже.

Следующий раздел подменю (или команда), **Новый текст**, относится к редактору написания текста для файла *VHDL*. В нем можно написать любой текст, например, плана работы над проектом, примечания к схеме или спецификацию, но в первую очередь он удобен для создания файла на языке *VHDL*. Что делать дальше с этим файлом, подробнее рассмотрим в дальнейшем, а сейчас можно упомянуть, что Qucs в составе цифровых компонентов имеет **Файл VHDL**, который удобно рассматривать как подсхему цифрового устройства. Язык *VHDL* удобен для описания работы компонентов, которых вам не хватает, если не хватает, для работы.

Следующие два раздела подменю, **Открыть** и **Закрыть**, типичные для любой программы. Подразумевается открыть и закрыть файл. Для того чтобы открыть ранее созданный проект, есть другая команда. И открывание, и закрывание файлов вызывают появление системных диалоговых окон, в которых вы можете переместиться по директориям, чтобы указать нужный файл в первом случае, или сохранить файл, если он не был сохранен в нужном вам месте, во втором случае. Если файл был сохранен перед этим, или не был изменен, он будет просто закрыт.

Команда **Сохранить** относится к текущему активному файлу. Если он открывался и был изменен, он будет сохранен под тем же именем, в том же месте, с теми же настройками. Если он не был сохранен до выполнения этой команды, то команда откроет диалоговое окно создания имени файла и указания места его хранения. Это опять системное диалоговое окно.

Раздел **Сохранить все** подразумевает, что будут сохранены все открытые файлы проекта, то есть, файлы схем, файлы данных, файлы графики. Они будут записаны вновь на месте прежних с теми именами, с которыми были открыты или созданы.

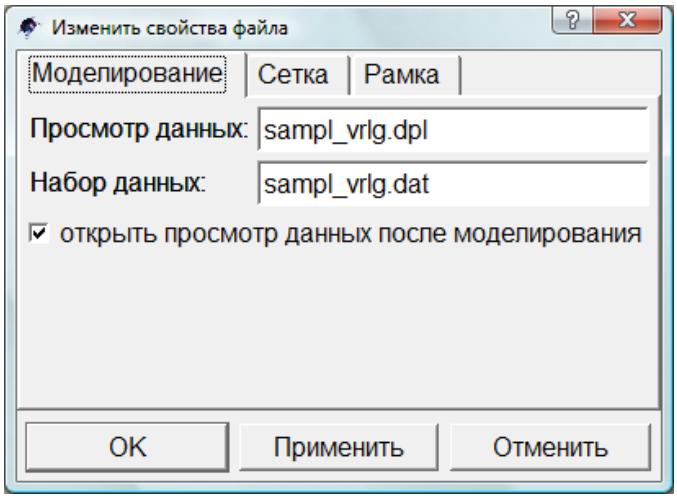
Команда **Сохранить как…** позволяет вам изменить имя файлы, если вы хотите, например, сохранить вариант схемы. Она же удобна при первом сохранении файла, хотя все равно обращается к системному диалоговому окну. Следует иметь в виду, что сохраняя версии схемы, при моделировании можно столкнуться с ситуацией, когда используются данные, полученные при моделировании предыдущей версии файла. Чтобы избежать этого, файл с расширением «.dat» можно удалить, используя панель навигации.

Напечатать. Открывается диалоговое системное окно печати, где вы можете выбрать принтер для печати, указать количество страниц для печати и количество копий. Можно настроить режим печати на выбранный принтер. В Windows Vista опция печати в файл остается не активизирована, возможно, следует установить дополнительно что-то, что поддерживает формат печати в файл. А в Linux эта опция позволяет сохранить схему или, что особенно важно при написании статьи или описания, полученную при моделировании графику в файле с расширением «.ps». Этот графический файл дает лучшие результаты по качеству при его вставке в текст.

Печать по размеру страницы отличается от предыдущей команды размером полученной копии рисунка на бумаге. В последнем случае рисунок займет всю страницу, а в первом будет пропорционально уменьшен.

Следующие три раздела подменю следует разобрать несколько подробнее. Итак.

Настройки документа…



По этой команде открывается диалоговое окно настроек документа. Первая закладка **Моделирование** позволяет изменить, если это зачем-то нужно, файлы графики и данных. Но еще и позволяет установить или снять флажок опции открыть просмотр данных после моделирования.

Дело в том, что после моделировании автоматически открывается окно настройки и просмотра графики. Удобно. А если снять флажок?

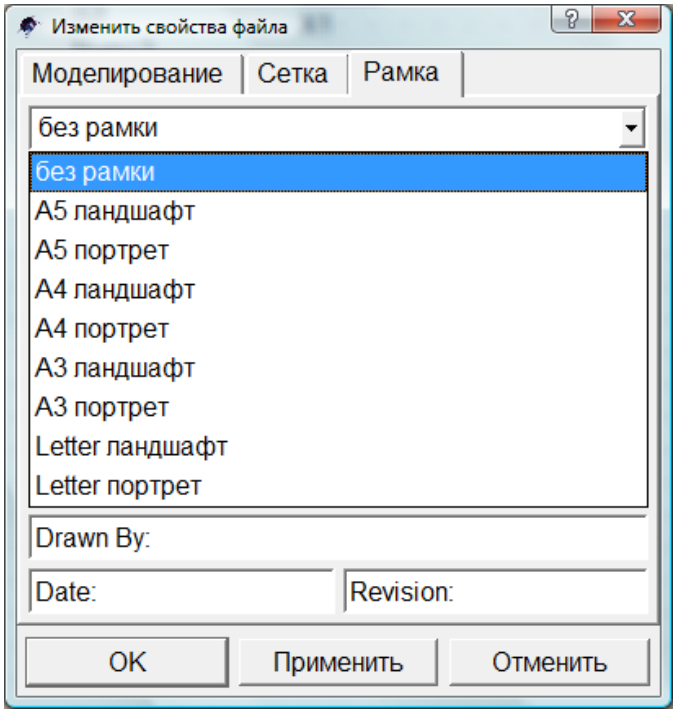
Рис. 2.2. Диалоговое окно настройки документа

Иногда, работая со схемой, налаживая ее или меняя свойства, удобно скопировать получающийся при моделировании график в поле самой схемы. Каждый из запусков

моделирования будет автоматически открывать в рабочем поле страницу с закладкой просмотра данных (в виде графики). Если же флажок снять, то после запуска вы останетесь на странице схемы. Иногда так удобнее.

Следующая закладка диалогового окна настройки документа относится к сетке. Как любой графический редактор, редактор схемы имеет сетку. Ее можно отключить, сняв флажок опции ***показывать сетку*** на этой вкладке. Если вы делаете копию экрана со схемой, удобнее убрать сетку. Кроме того, вы можете изменить параметры сетки для вновь создаваемого файла схемы. Иногда при создании символа не получается удобно разместить все компоненты при крупной сетке. В этом случае размер сетки следует уменьшить (именно на этой закладке).

Последняя закладка этого диалогового окна позволяет вам оформить чертеж. Называется закладка Рамка.



По умолчанию выбран вариант ***без рамки***. Но, нажав на кнопку со стрелкой вниз, справа от этой надписи, вы обнаружите, что рамка может быть создана для нескольких форматов в двух вариантах размещения.

Помимо рамки оформляется и штамп. Вы впечатываете название схемы, кто чертил, дату и версию схемы.

На рисунке видно, что в Windows Vista надписи появляются на английском языке. Пусть это вас не смущает, вы вольны изменить их на свой вкус.

Рис. 2.3. Закладка «Рамка» диалога настройки документа

Даже если вы осуществляете работу для себя, полезно отображать весь процесс создания схемы на печатных копиях, где есть рамка, есть дата создания данной версии схемы, и номер этой версии, да и версии можно называть, модифицируя основное название устройства. Впоследствии это поможет вам легче найти все файлы, относящиеся к проекту. Особенно по прошествии года или больше.

Если программа используется в учебных целях, такой удобный механизм позволит лучше контролировать работу учащихся, а их приучит к аккуратности, которая крайне важна как при работе с электроникой, так и при работе с электричеством, даже если это проектная работа.

Схема в рабочем поле программы приобретает вид, показанный ниже.

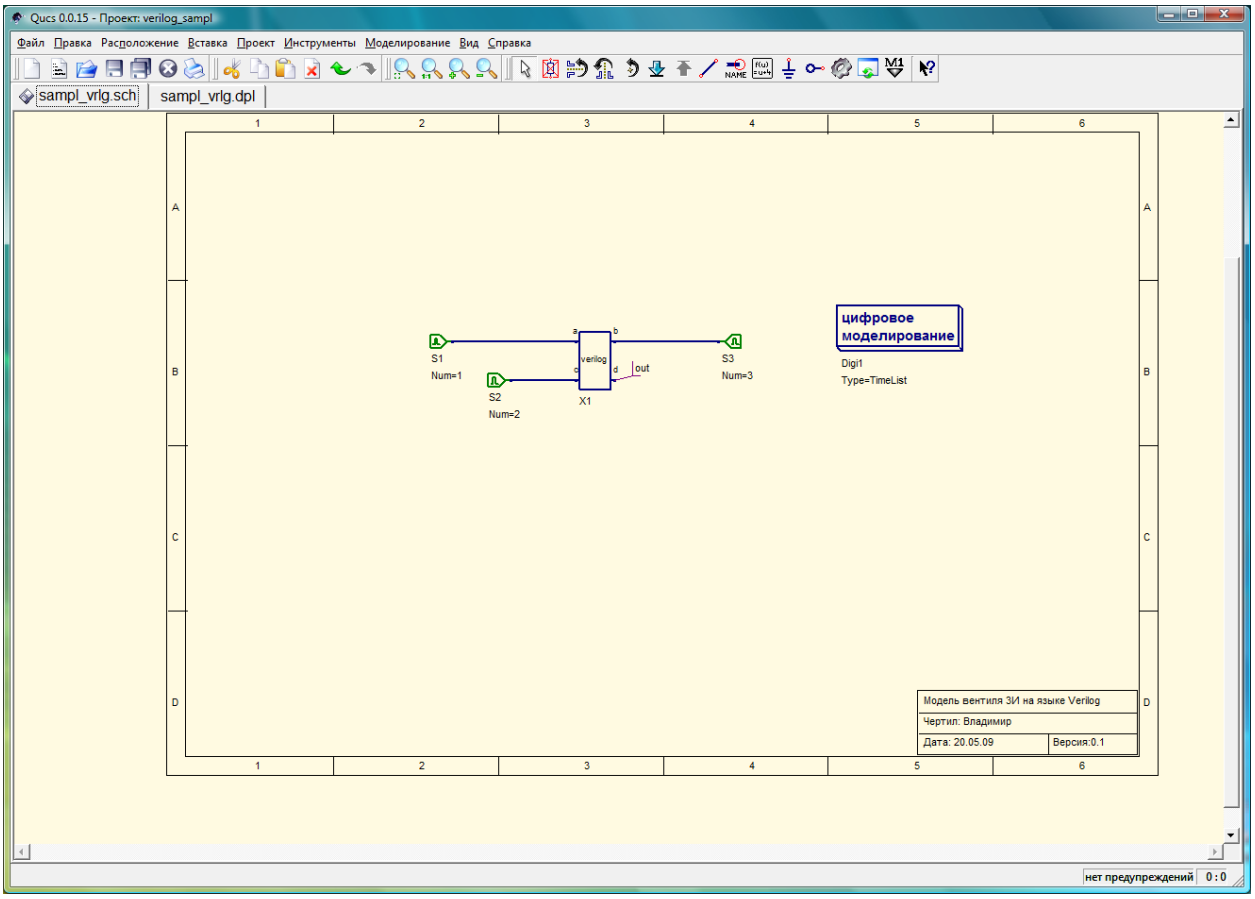


Рис. 2.4. Вид схемы в окне редактора с оформленной рамкой

Изменить обозначение схемы. Так выглядит раздел подменю, когда в рабочем поле схема. Но после того как вы щелкните левой клавишей мышки по этой команде, или, некоторые предпочитают пользоваться клавиатурой – нажав на клавишу ***F9***, ваша схема превратится в символ, скажем, подсхемы. Вот как выглядит схема первоначально:

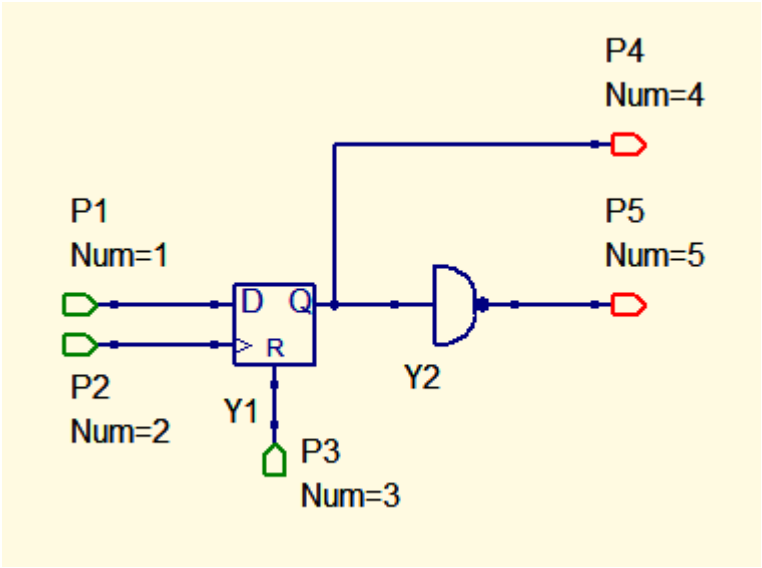


Рис. 2.5. Цифровая схема на базе D-триггера с прямым выходом

P1-P5 – это порты. Зачем они, почему выглядят по-разному, обо всем этом мы узнаем позже, а сейчас посмотрим, как выглядит эта схема после выполнения команды.

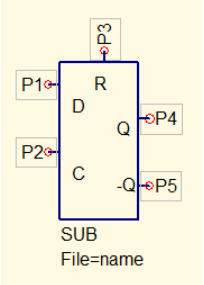


Рис. 2.6. Ее символ

Этот символ был дополнительно перерисован, но об этом тоже позже, когда речь пойдет о подсхемах. Сейчас можно только добавить, что после превращения схемы в символ подсхемы, надпись **Изменить обозначение схемы** превратилась в **Изменить схему**.

Не менее важный смысл, чем настройки документа, имеет следующий раздел **Настройки программы**.

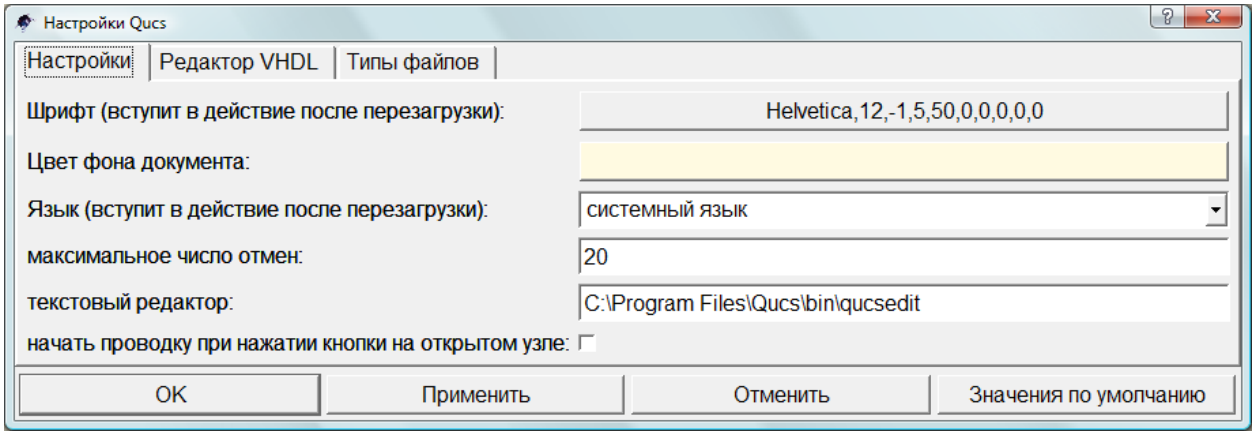


Рис. 2.7. Диалоговое окно настроек программы

В первую очередь в этом диалоговом окне можно изменить язык интерфейса. Не думаю, что вам это нужно, но это можно. Программа поддерживает около двух десятков языков. Вам это будет полезно в том смысле, что программа созданная, скажем, на португальском языке, вполне может быть использована вами, если обозначение входных и выходных сигналов сделано на английском, что является обычной практикой. Если нет, то не сложно переделать эти обозначения.

Цвет фона рабочего поля программы можно изменить в этом диалоге, если вам не нравится тот, что придумали авторы. Также и шрифт. Программа Qucs имеет встроенный текстовый редактор. Вы можете использовать свой – достаточно указать путь к нему в поле ***текстовый редактор***, чтобы пользоваться другим редактором.

На закладке есть опция ***начать проводку при нажатии кнопки на открытом узле***. Если установить флажок, то при соединении схемы не нужно будет нажимать на кнопку ***Проводник*** инструментального меню, или использовать раздел **Проводник** подменю **Вставка** в основном меню, или нажимать сочетание клавиш ***Ctrl+E*** на клавиатуре. Можно будет просто щелкнуть по открытому узлу левой клавишей мышки и провести проводник к нужному узлу компонента или другому проводнику.

На закладке **Редактор VHDL** можно изменить цвета текста и фона редактора, а наличие этой возможности говорит о том, что редактор VHDL построен по образцу других редакторов программирования – в нем выделяются цветом, например, служебные слова, числа, строковые константы и т.п. Такая организация текста программы делает его удобным для восприятия и позволяет легче разобраться в ошибках при отладке программы.

Последняя закладка **Типы файлов** позволит вам использовать другие программы при работе с Qucs, если такая нужда у вас возникнет. Достаточно вписать расширение файла и указать путь к исполняемому файлу, чтобы нужная программа открывала файл с заданным расширением.

Изменяя настройки программы, вы можете принять эти изменения, нажав на кнопку ***Применить*** и затем ***ОК***, можете отменить изменения с помощью кнопки ***Отменить***, и можете вернуться к прежним настройкам, нажав на кнопку ***Значения по умолчанию***.

Завершается подменю Файл очевидной командой **Выйти**. Можно использовать сочетание клавиш ***Ctrl+Q***. Если у вас есть файлы, которые вы изменили, но не сохранили, появится сообщение с запросом, действительно ли вы хотите выйти из программы, а если вы подтвердите это, то появится диалоговое окно, где вам будет предложено сохранить все файлы, которые были изменены или отказаться от сохранения.

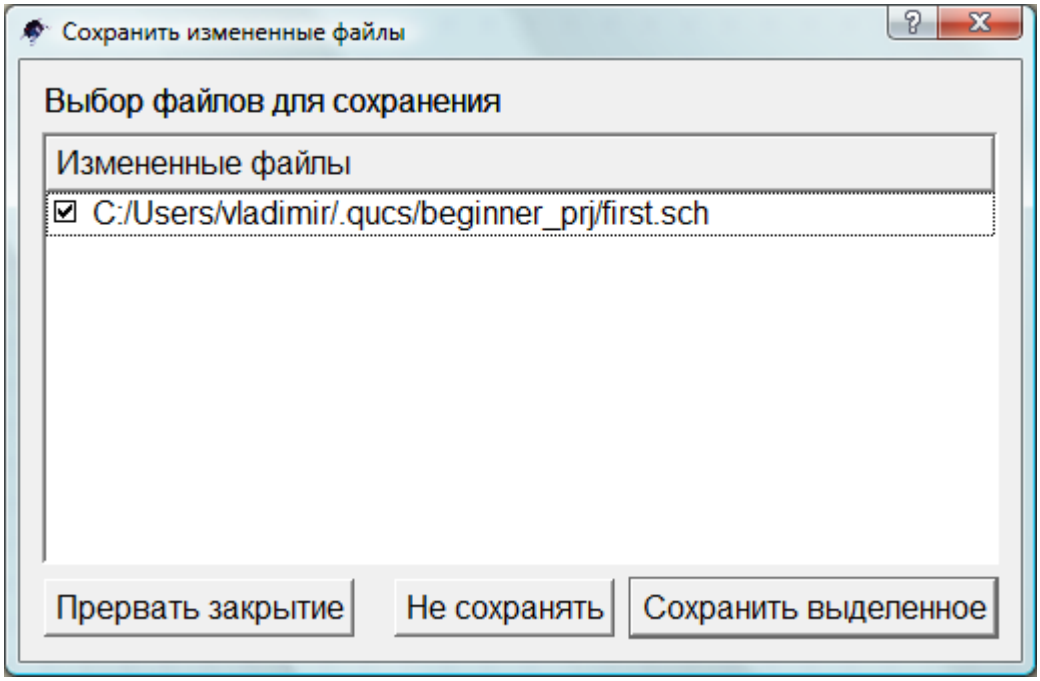
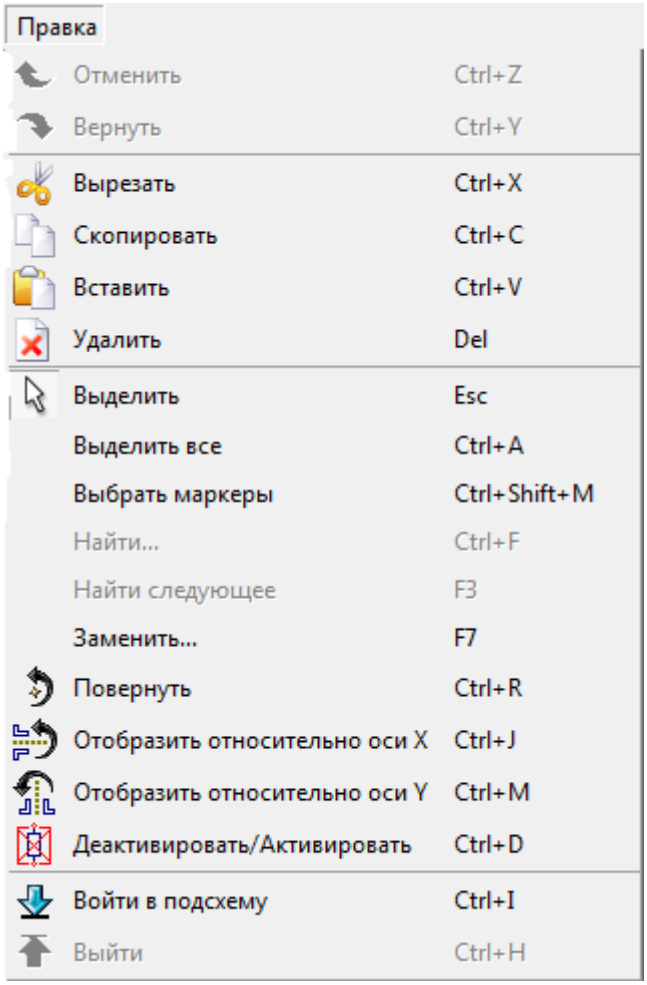


Рис. 2.8. Диалоговое окно выхода из программы

Вы можете, сняв флажки возле файлов, которые вы не намерены менять, оставить только то, что действительно нуждается в сохранении. А, если вы погорячились, нажав на системную кнопку закрывания окна программы, что бывает, когда промахиваешься и нажимаешь не то, что хотел; а потом, не прочитав сообщение, нажали кнопку ***Да***, когда программа спрашивала, хотите ли вы выйти, то остался последний шанс все исправить в этом диалоге, нажав на кнопку ***Прервать закрытие***.

Основное меню, Правка



Как любой редактор, текстовый или графический, редактор схем в Qucs должен обладать всеми необходимыми для редактирования средствами, впрочем, знакомыми вам по другим программам: вырезать, вставить, копировать и т.д.

Эти же команды редактирования, если вы на странице, положим, отображения данных (графиков) или на странице редактора VHDL, помогут вам редактировать текст, скопировать график и перенести его на страницу схемы, или удалить ненужные элементы.

Открывает подменю команда **Отменить**. Она относится к последнему действию, которое вы совершили по ошибке, или к действию, которое вы передумали выполнять.

Рис. 2.9. Подменю Правка основного меню

Команда **Отменить** применяется к последнему действию, но отменить вы можете десятка два последних команд, как минимум. Маловероятно, что вы сделаете такое количество ошибочных действий при вводе схемы, но, кто знает…

Команда **Вернуть** отменяет последнее действие команды **Отменить**, если вы выполнили ее по ошибке или передумали. Команда имеет такую же глубину вложенности, как и команда **Отменить**. Если вы предпочитаете использовать клавиатуру, то этим командам соответствуют «горячие клавиши» ***Ctrl+Z*** и ***Ctrl+Y***, соответственно.

Команда **Вырезать** подразумевает, что вы выделили элемент схемы, если вы в редакторе схем. Для выделения, если вы в режиме выделения, о чем свидетельствует «утопленная» одноименная клавиша (рядом с командой **Выделить** в подменю и на инструментальной панели), для выделения достаточно щелкнуть по элементу. И после этого выполнить команду **Вырезать,** выбрав ее из подменю, воспользовавшись инструментальной панелью или нажав «горячие клавиши» ***Ctrl+X***. Если следом вы выполните команду **Вставить** в редакторе схемы, то вырезанный элемент появится в виде контура, привязанного к курсору мышки, что позволяет вам выбрать место для вставки. Если вы вырезали элемент схемы по ошибке, затем выполнили команду **Отменить**, а следом команду **Вставить**, то вставленный элемент будет иметь номер на единицу

больше, чем последний аналогичный элемент в схеме. То есть, нумерация элементов поддерживается автоматически и распространяется и на механизм редактирования.

Многие команды пункта **Правка** основного меню относятся не только к редактору схем, с их помощью вы можете редактировать графику или текст, но в последнем случае не все разделы подменю будут работать. Можно править и текст в редакторе VHDL, тогда как встроенный текстовый редактор имеет свое меню.

Как и команда **Вырезать**, команда **Скопировать** требует выделения элемента или нескольких элементов. Все, что выделено в рабочем поле, будет скопировано в буфер обмена с тем, чтобы впоследствии можно было вставить его в новое место с помощью команды **Вставить**. То, что находится в буфере обмена, будет вставлено. Как и в случае вырезания, при вставке объекта из буфера обмена он появляется в виде контура, привязанного к концу курсора. Перемещая курсор мышки в нужное место чертежа, вы затем щелкаете левой клавишей мышки, чтобы **Вставить** объект.

Рис. 2.10. Перенос объекта после команд **Скопировать** и **Вставить**

Команда **Удалить**, в отличие от команды **Вырезать**, не помещает удаленный объект в буфер обмена для последующей вставки. Однако удаленный объект (или выделенные и удаленные объекты) можно вернуть на место командой **Отменить**.

Следующий раздел подменю **Правка** называется **Выделить**. Скорее это не команда, а режим работы, в котором по большей части вы находитесь. Например, когда вы соединяете компоненты, вы в режиме **Проводник**, но, закончив соединение, с помощью клавиши Esc возвращаетесь в режим выделения, о чем свидетельствует обычный вид курсора и нажатая кнопка с рисунком курсора на инструментальной панели. Для выделения одного элемента схемы достаточно щелкнуть по нему левой клавишей мышки, когда курсор находится над этим элементом.

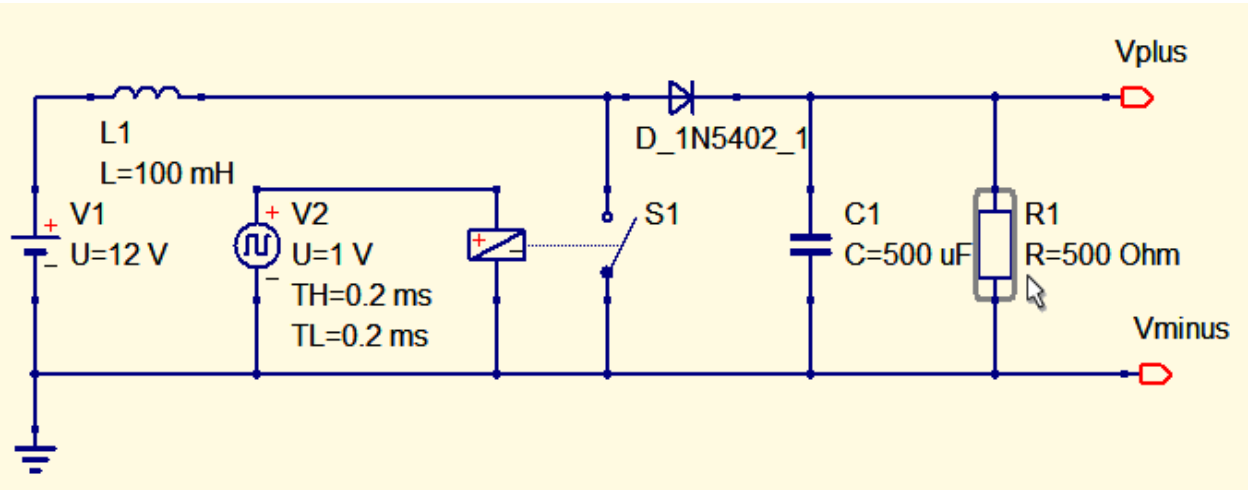
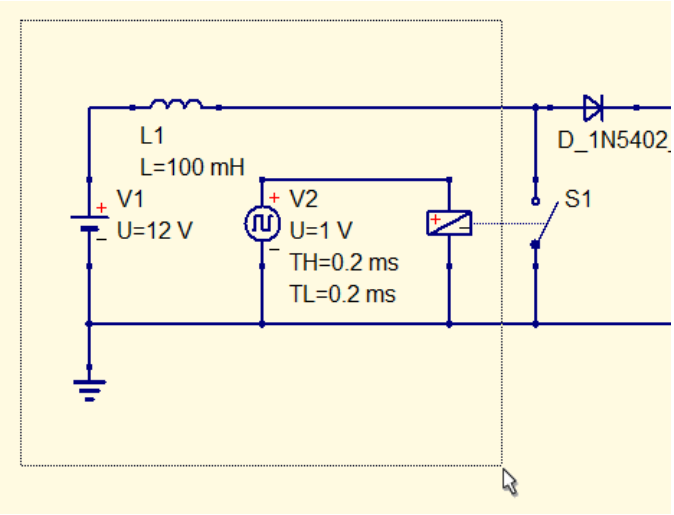


Рис. 2.11. Выделение одного элемента схемы

Чтобы выделить несколько объектов, поместите курсор на свободное место над группой объектов, нажмите левую клавишу мышки и, удерживая ее, начните движение курсора по диагонали вниз. Появится контур прямоугольника выделения. Обведите этим прямоугольником все нужные компоненты, а затем отпустите клавишу мышки. Все объекты внутри прямоугольника выделения окажутся выделены.



Чтобы снять выделение, если оно вас чем-то не устраивает, достаточно щелкнуть левой клавишей мышки в любом месте рабочего поля, где нет компонентов схемы.

Рис. 2.12. Прямоугольное выделение объектов в редакторе схемы

Выделить все. Эта команда выделяет все, что находится в рабочем поле чертежа. После команды можно использовать команды **Вырезать**, **Скопировать** или **Удалить**. Обычно эта команда полезна, если вы хотите создать копию схемы, которую вставите на новую страницу, как часть нового проекта или при создании новой версии.

Выбрать маркеры. Эта команда требует пояснения. Симуляция в программе Qucs, как во многих профессиональных программах, это способ получить графическое представление результатов работы схемы. Очень часто вас интересуют значения функций в отдельные моменты времени. Для облегчения вашей жизни программа предлагает механизм маркеров. Установить маркер на график просто – выбираете либо из пункта Вставка основного меню раздел Установить маркер на диаграмме, либо используете кнопку с буквой ***M1*** на инструментальной панели, либо используете ***Ctrl+B*** на клавиатуре. После чего помещаете курсор мышки в нужное место графика и щелкаете левой клавишей мышки. Таких маркеров может быть много, как на рисунке.

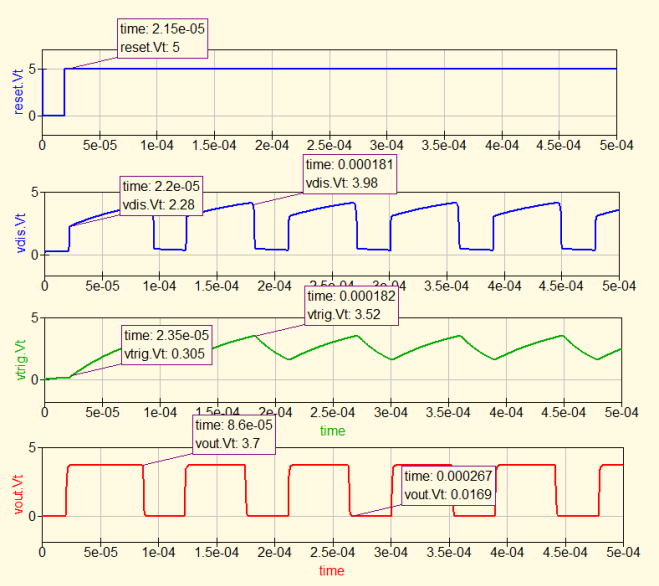


Рис. 2.13. Графики с маркерами

Команда **Выбрать маркеры** относится именно к ним.

Команды **Найти** и **Найти следующее**, как это видно из рисунка подменю, не активизированы. Они активизируются, когда открыт текст в редакторе VHDL, где текст может быть сложным и использование команд поиска оказывается очень полезно.

Следующая команда **Заменить** открывает диалоговое окно, вид которого зависит от того, какое окно активно в рабочем поле программы: окно редактора VHDL или окно редактора схем.

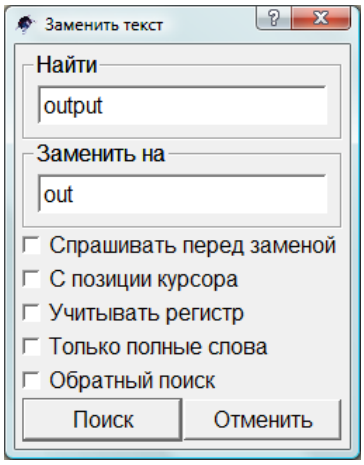


Рис. 2.14. Активно окно VHDL

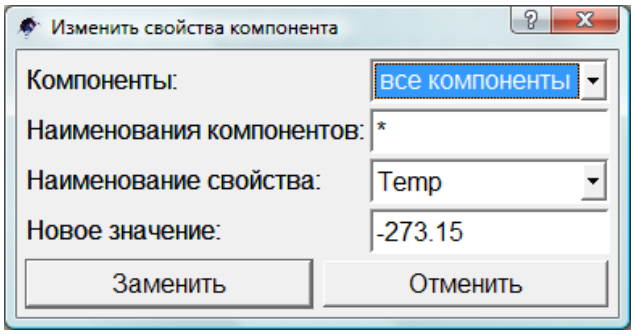
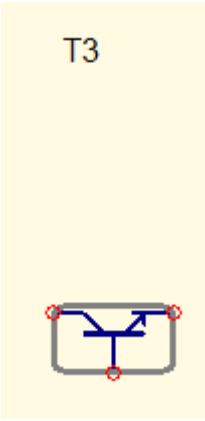


Рис. 2.15. Активно окно редактора схем

Команда **Повернуть**. Команда работает с элементами схемы. Выделив элемент, его можно повернуть на 90 градусов против часовой стрелки. Каждый раз, когда вы используете эту команду, выделенный элемент поворачивается на 90 градусов. Некоторые компоненты имеют много параметров, отображаемых на схеме по умолчанию. Можно отключить показ этих параметров. Но, когда вы сначала повернете компонент, затем отключите отображение ненужных компонентов, вы можете получить нечто подобное:



Есть два варианта, как этого избежать: снять флажки отображения параметров до поворота; использовать пункт Расположение, где есть раздел Переместить текст компонента.

Рис. 2.16. Вид элемента схемы после поворота

К компонентам схемы относятся и команды отображения: **Отобразить относительно оси X** и **Отобразить относительно оси Y**. Они тоже работают в редакторе схем. И, если по отношению к таким асимметричным компонентам, как транзистор, выгода от существования команд очевидна, то, казалось бы, зачем нужно применять эти команды к резисторам или конденсаторам. Но не

следует забывать о таких элементах схемы, как обозначения резисторов и конденсаторов, особенно в схемах, где элементов много.

Следующая команда **Деактивировать/Активировать** относится в первую очередь к таким обязательным компонентам схемы, как вид моделирования (или симуляция). Для схемы можно применить несколько видов симуляции, но иногда полезно (или необходимо) часть этих процессов не выполнять. Чтобы не удалять уже настроенный элемент работы, можно его деактивировать, и он не будет принимать участие в процессе симуляции. Повторное применение команды активирует его. Деактивировать можно и другие компоненты схемы. Чтобы, например, увидеть влияние этого компонента на работу схемы. Для этого достаточно выделить компонент, щелкнув по нему левой клавишей мышки, и выполнить команду **Деактивировать/Активировать**.

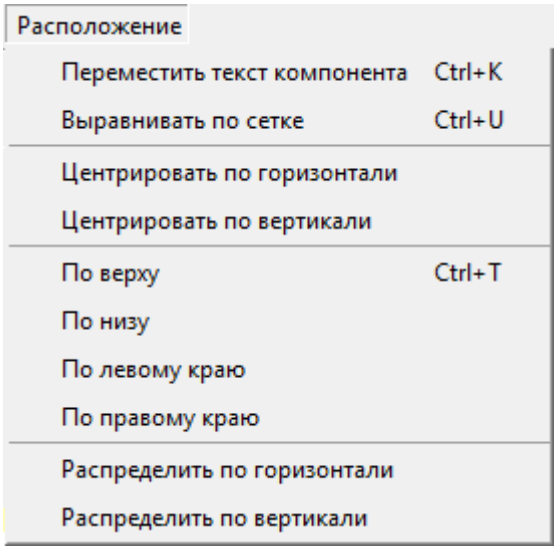
Последние две команды **Войти в подсхему** и **Выйти** позволяют вам быстро увидеть подсхему, которая есть (и если есть) в вашей схеме, и вернуться к схеме. Последняя команда не активна до тех пор, пока вы не войдете в подсхему.

Использование подсхем создает ряд удобств. Вам может оказаться совсем ненужным отображение ряда схем, если вы используете их для построения своего компонента, что можно видеть на примере построения таймера 555. Вас в этом случае интересует микросхема таймера, которую вы используете как целое для построения, скажем, осциллятора. В этом случае подсхемы являются вспомогательными элементами. Но при изменении некоторых параметров конечной схемы вам может потребоваться изменение параметров подсхем. Используйте команду Войти в подсхему. Если вы привыкли активно использовать в работе клавиатуру, достаточно нажать клавиши ***Ctrl+I***.

В сложных схемах обилие простых компонентов может отвлекать внимание от существа процессов в работе устройства. И в этом случае применение подсхем позволяет обозначить функцию вспомогательной, хотя, может быть, и очень важной, части схемы в виде блока. Но, когда вы налаживаете схему, чтобы изменить параметры внутри этого функционального блока, нужно войти в подсхему. Выделите функциональный блок подсхемы и нажмите, например, кнопку на инструментальной панели.

Разработка электрических схем во многом схожа с написанием текста, где вместо букв используются такие компоненты, как резисторы, конденсаторы, транзисторы или триггеры в цифровых устройствах. Как и текст, схема, пока она окончательно не создана и не проверена, нуждается в редактировании, то есть, в правке. Все разделы этого подменю предназначены к правке «текста электрических цепей» и полностью закрывают все нужды редактора. А, если каких-то «ножниц» для правки не хватает, то они могут быть в других подменю основного меню, о чем будет рассказано дальше.

Основное меню, Расположение



Подменю пункта **Расположение** основного меню ориентировано на выравнивание компонентов схемы.

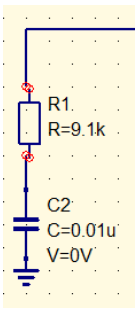
Рис. 2.17. Подменю Расположение

Первая команда **Переместить текст компонента**, как это ясно из названия, относится к текстовым элементам схемы. Выше упоминалось, что при поворотах компонентов, если вслед за этим отключать видимость параметров, может получиться так, что обозначение элемента схемы оказывается слишком далеко от него. Чтобы поправить это, достаточно нажать на этот раздел подменю или нажать и удержать на клавиатуре клавишу ***Ctrl*** и нажать клавишу ***K*** (***Ctrl+K***), появится курсор с рамкой справа, подведите его к нужному тексту, нажмите левую клавишу мышки и перетащите текст в предназначенное к этому место. Для отказа от режима перемещения текста можно нажать клавишу ***Esc*** на клавиатуре или щелкнуть по кнопке с рисунком курсора (режим выделения).

Перемещаются все текстовые атрибуты компонента, если позже вы захотите сделать видимыми те атрибуты, которые были скрыты, достаточно установить флажок видимости в свойствах компонента; при появлении нового параметра остальные сдвинутся, освобождая место для нового атрибута, если ниже текста находится символ элемента схемы.

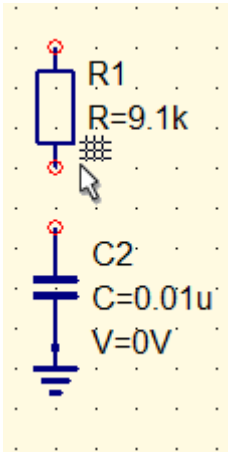
Режим перемещения не распространяется на компоненты группы рисунки или элементы графического отображения данных.

Выравнивать по сетке. Создавая схему, вы, такое бывает со сложными схемами, вы используете сетку графического редактора иную, чем задана по умолчанию. Параметры сетки задаются в настройках документа. Позднее, используя отработанную и проверенную ранее схему в новом проекте, вы можете столкнуться с ситуацией, когда не получается соединение элементов.



Чтобы не переделывать всю схему, вы можете либо выделить элемент, с которым не получается соединение, а затем щелкнуть по разделу **Выравнивать по сетке**; либо использовать команду **Выравнивать по сетке**, после чего курсор меняет вид, а затем, переместив его к нужному компоненту, щелкнуть левой клавишей мышки по «непокорному» компоненту. Он будет выровнен по сетке.

Рис. 2.18. Компонент, который отказывается соединяться в цепь



В таком положении вы можете оказаться и при копировании элементов из схемы, где сетка была задана с другими параметрами. После вставки компонентов в новую схему они отказываются соединяться проводниками. Выделите их с помощью прямоугольного выделения и используйте команду **Выравнивать по сетке**, например, с помощью клавиатуры ***Ctrl+U***.

Рис. 2.19. Режим выравнивания по сетке

Команда **Центрировать по горизонтали** относится к компонентам электрической цепи. Наберите несколько резисторов во вновь созданный файл.

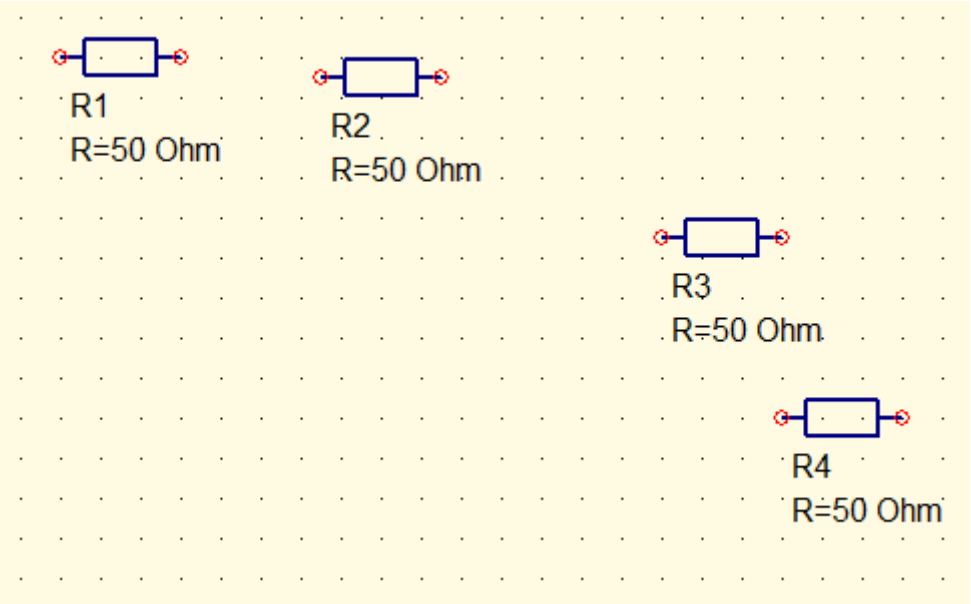


Рис. 2.20. Несколько резисторов, расположенных случайным образом

Если вы не выделили элементы, то после команды **Центрировать по горизонтали** появится сообщение о том, что следует выделить хотя бы два элемента. Если вы выделили все резисторы, то после команды получите следующий результат.

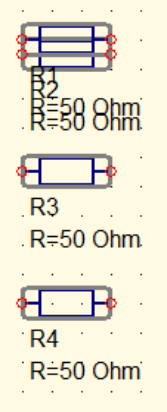


Рис. 2.21. Выполненная команда в применении к предыдущему размещению

А выполнение следующей команды **Центрировать по вертикали**, даст:

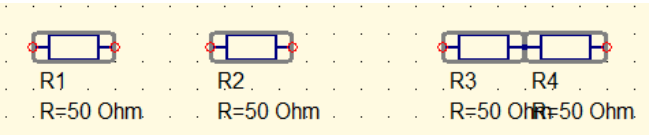


Рис. 2.22. Выполненная команда **Центрировать по вертикали**

Обе команды могут быть полезны при размещении компонентов перед соединением их в цепь.

По верху и **По низу**, **По левому краю** и **По правому краю**. Эти команды выравнивают объекты. Первая выравнивает все выделенные объекты по уровню самого верхнего из них, следующая по уровню самого нижнего, следующие две по уровням левого и правого. Если команды центрирования выравнивают объекты по осевым линиям, то эти четыре команды выравнивают выделенные объекты по крайним из них.

Команда **Распределить по горизонтали** распространяется на все выделенные объекты и распределяет их между двумя крайними на равном расстоянии друг от друга.

Аналогично, только по вертикали, распределяет выделенные объекты на равных расстояниях и следующая команда **Распределить по вертикали**.

Команды центрирования, выравнивания и распределения действуют и на компоненты электрической схемы, и на компоненты группы **рисунки**.

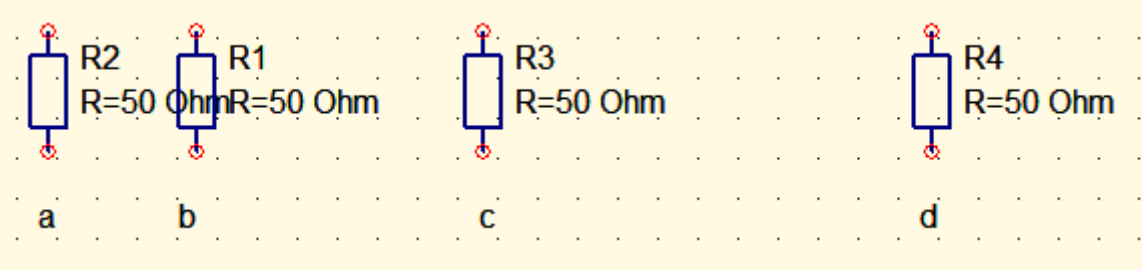


Рис. 2.23. Первоначальное расположение компонентов схемы

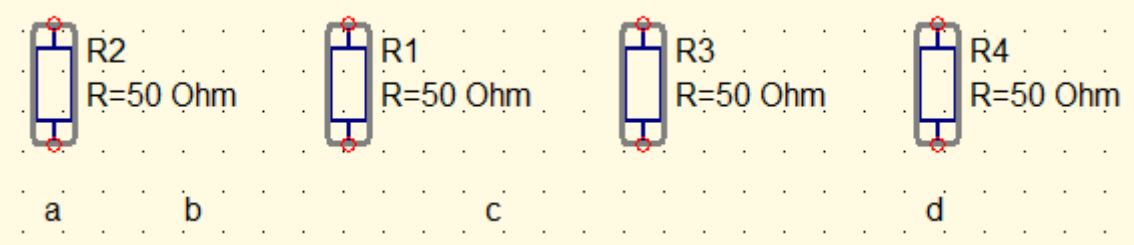


Рис. 2.24. Расположение компонентов после выполнения команды распределения

Если в предыдущем случае выделены не только резисторы, но и текстовые объекты под ними, то команда **Распределить по горизонтали** приводит к другому результату. Будьте внимательны.

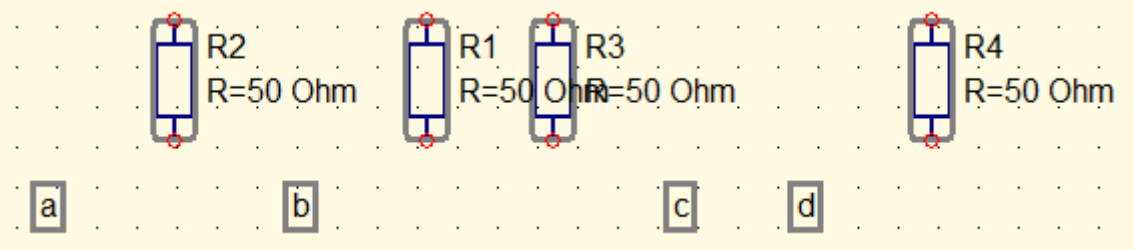
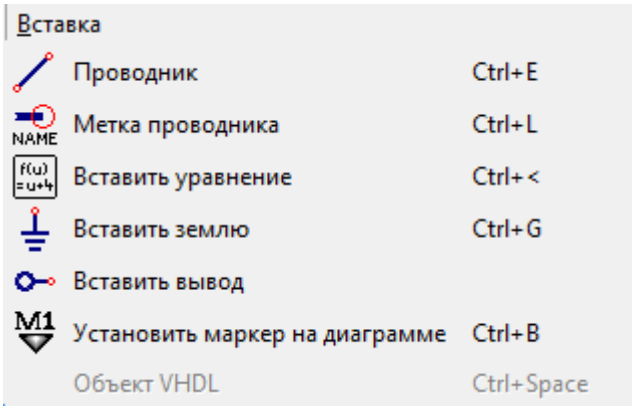


Рис. 2.25. Совместное распределение компонентов

Основное меню, Вставка



В подменю **Вставка** собраны команды, превращающие вынесенные в рабочее поле компоненты в электрическую цепь.

В первую очередь это относится к команде **Проводник**, переключающей редактор из режима **Выделить** в режим проведения соединений.

Рис. 2.26. Подменю **Вставка**

Соединение электрических компонентов может производиться множеством разных способов. От способа соединения зависит и то, что из себя будет представлять схема, как она будет работать, какие функции будет выполнять. Одни и те же компоненты можно соединить в разные схемы. А одинаковые по назначению схемы можно строить из разных компонентов. Поэтому работу по соединению компонентов в схему, как правило, не автоматизируют, соединения производятся вручную.

Используя команду **Проводник** из подменю, используя кнопку инструментального меню с тем же рисунком или нажимая клавиши ***Ctrl+E*** на клавиатуре, программу переводят в режим соединения, что заметно по изменившемуся виду курсора.

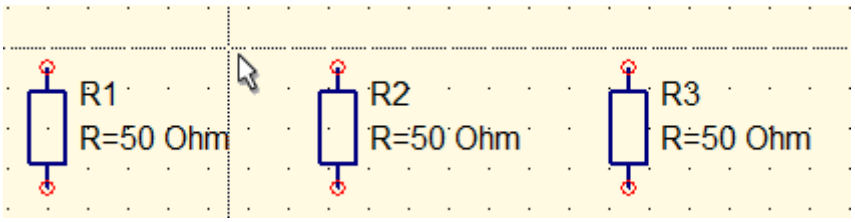
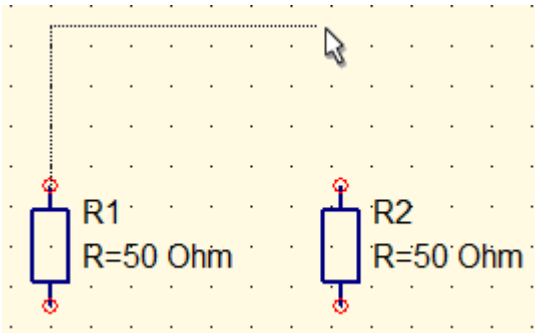


Рис. 2.27. Программа в режиме проведения соединения

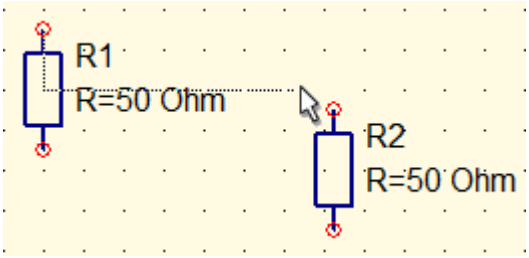
Горизонтальная и вертикальная направляющие помогают проведению соединений. Чтобы начать соединение, нужно щелкнуть левой клавишей мышки по выводу компонента в виде красного кружка, за курсором потянется пунктирная линия, которую следует провести к следующему соединяемому компоненту.



Соединять можно не только два компонента между собой. Можно соединить два уже существующие проводника, или присоединить существующий проводник к компоненту.

Рис. 2.28. Проведение соединения

При проведении соединения может возникать ситуация, показанная ниже.



При соединении резистора R1 с резистором R2 вам достаточно щелкнуть правой клавишей мышки, чтобы соединение изменило свое направление.

Рис. 2.29. Неверное положение соединения при изменении направления

Завершается соединение щелчком левой клавиши мышки, когда курсор находится над уже существующим проводником или открытым выводом компонента, помеченным красным кружком.

Некоторая доля автоматизации в соединении присутствует. Если вы соединяете резисторы, как показано на рисунке ниже, вам достаточно переместить курсор к открытому выводу резистора R3, где щелкнуть левой клавишей мышки.

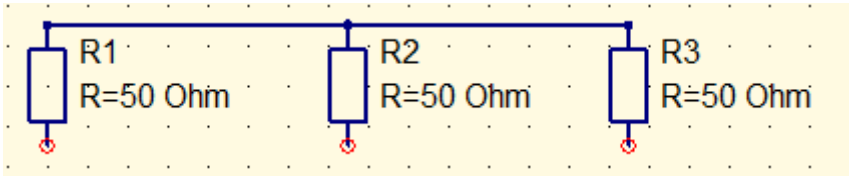


Рис. 2.30. Проведение соединения трех компонентов

Но подобная автоматизация принесет пользу только тогда, когда вы собирались соединить все три резистора, если же вы хотели соединить только резисторы R1 и R3, то автоматизация соединения, скажем, сыграла с вами «злую шутку». Как можно исправить положение или избежать этого?

Можно было поднять курсор над резисторами так, чтобы обойти вывод резистора R2, провести курсор по направлению к резистору R3, щелкнуть левой клавишей мышки над ним и продолжить движение к нужному узлу.

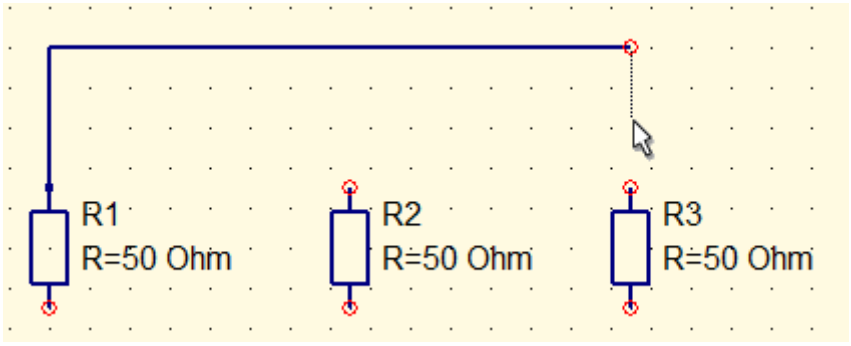


Рис. 2.31. Обход ненужного в соединении узла компонента

Можно поступить и иначе. После проведения соединения можно «поднять» провода над компонентами. Для этого достаточно поместить курсор мышки на провод, нажать и удержать левую клавишу мышки и переместить проводник в нужное положение.

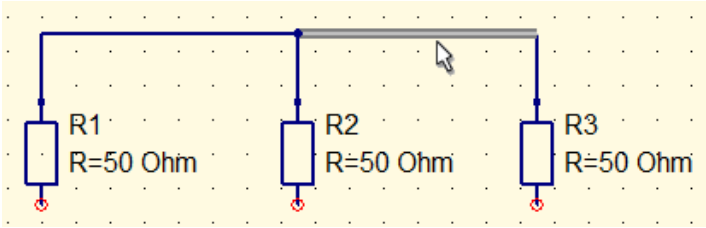


Рис. 2.32. Перемещение проводников с помощью мышки

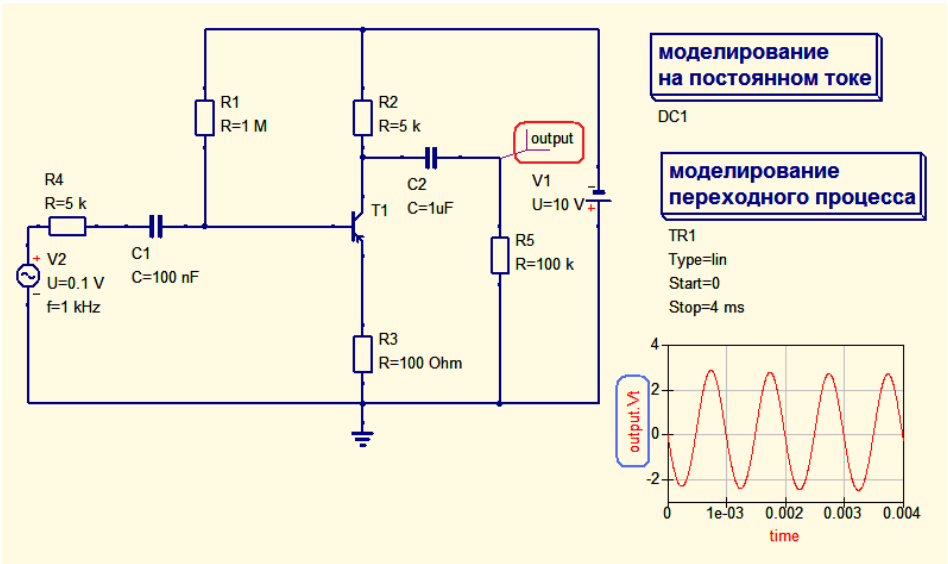
Теперь можно выделить, щелкнув мышкой по ненужному проводнику к резистору R2, и удалить его, нажав клавишу ***Delete*** на клавиатуре, используя команду **Удалить** из подменю **Правка,** или используя кнопку инструментального меню.

Если кому-то покажется удобным сэкономить время на переходе в режим соединения, можно, напомню, установить опцию ***начать проводку при нажатии кнопке на открытом узле*** в настройках программы (основное меню, **Файл**, **Настройки программы**…). В этом случае при щелчке на открытом узле левой клавишей мышки вы автоматически переходите в режим соединения.

При проведении соединения щелчок правой клавишей мышки меняет направление соединения.

Метка проводника. Вставка в первую очередь удобная для отметки точек наблюдения в схеме. При моделировании переходных процессов, пожалуй, самом распространенном виде моделирования, метку можно устанавливать на проводник, подходящий к компоненту, или к выводу компонента, если он представлен микросхемой.

При налаживании схемы, при разборе работы схемы нас очень часто интересует наличие и вид сигнала, генерируемого схемой или преобразованного схемой, например, усиленного.



В данном случае метка ***output*** используется для наблюдения выходного сигнала.

Она появляется на графике, как функция от времени.

Рис. 2.33. Использование метки проводника в качестве точки наблюдения

Для того чтобы добавить метку к выводу компонента или к точке наблюдения, достаточно выбрать эту команду в подменю Вставка, нажать кнопку с надписью ***name*** на инструментально панели или нажать ***Ctrl+L*** на клавиатуре.

Курсор приобретает вид, показанный на рисунке. Теперь его достаточно подвести к нужному проводнику или выводу, где щелкнуть левой клавишей мышки.

Рис. 2.34. Вид курсора при установке метки

После щелчка мышки открывается диалоговое окно ввода имени метки.

Вы вольны называть метку, как вам удобнее, но использовать для метки можно только латиницу. При попытке ввести что-то кириллицей, вы не увидите ничего. Введя метку, можно нажать на клавишу ***OK*** или ***Отменить***, если вы передумали. Метку можно переместить – достаточно выделить ее, подцепить мышкой и перенести в другое место.

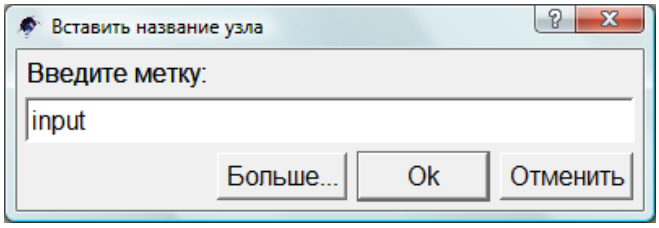
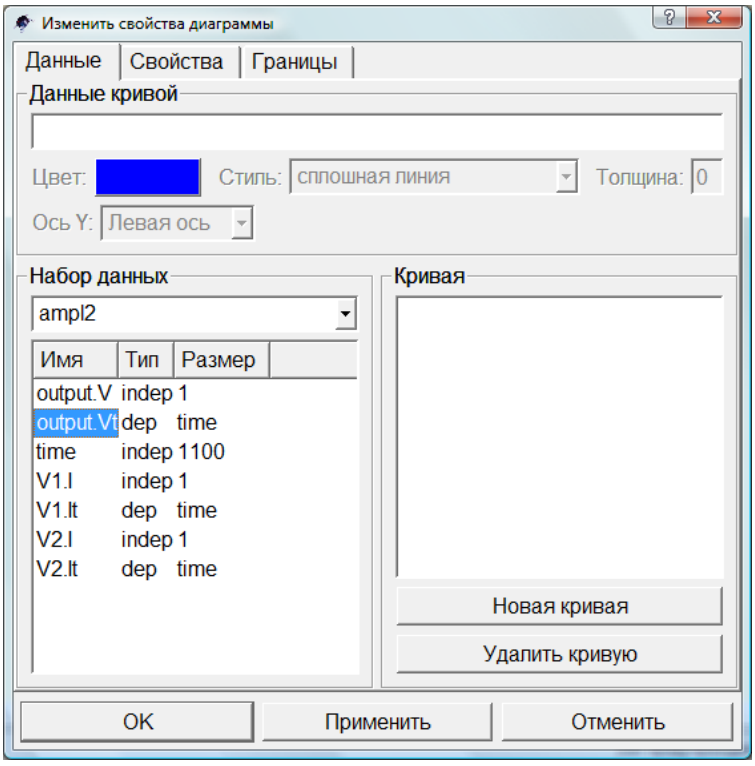


Рис. 2.35. Диалоговое окно метки проводника

Когда после моделирования схемы, скажем, показанной выше, программа открывает страницу отображения данных, а вы выбираете **Декартовскую** диаграмму, откроется окно диалога выбора наблюдаемого сигнала.



Источники, напряжения и тока появляются в диалоге автоматически, а точки наблюдения отображены те, что заданы с помощью метки проводника. Метка ***output*** в двух видах: независимая и зависимая от времени. Двойной щелчок по ней переведет ее в правое окно диалога, где показаны все функции, видимые на графике.

Рис. 2.37. Окно диалога выбора графика функции

Вставить уравнение. Программа Qucs многое делает сама. В процессе симуляции она многократно рассчитывает и пересчитывает нарисованную вами электрическую цепь. Когда вы выбрали способ отображения полученных при симуляции данных, она рассчитает и построит график функции, соответствующий сигналу в выбранной вами точке схемы. Но иногда вам удобнее немного «поправить» полученный результат. Рассмотрим простейший случай со схемой однокаскадного усилителя. Моделирование на переменном токе позволяет получить амплитудно-частотную характеристику каскада.

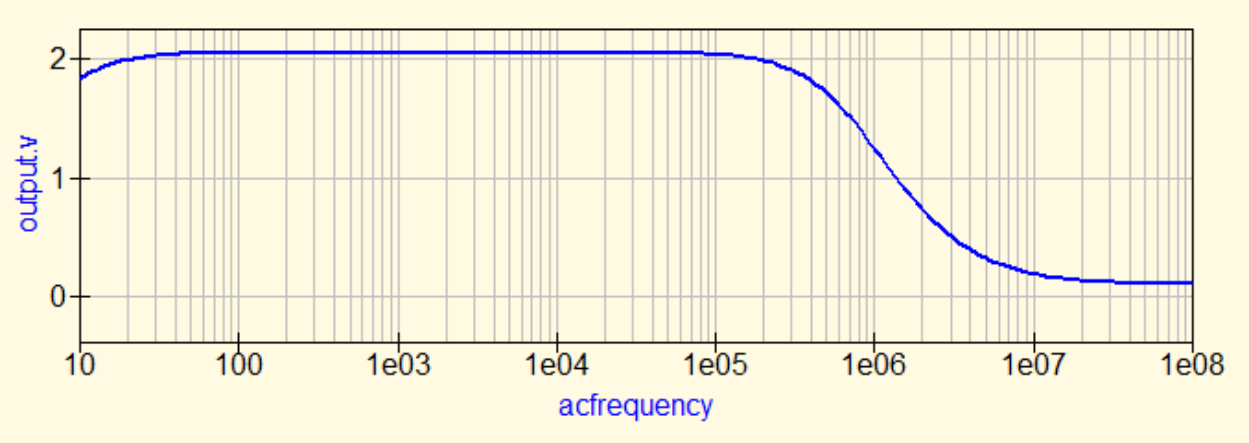
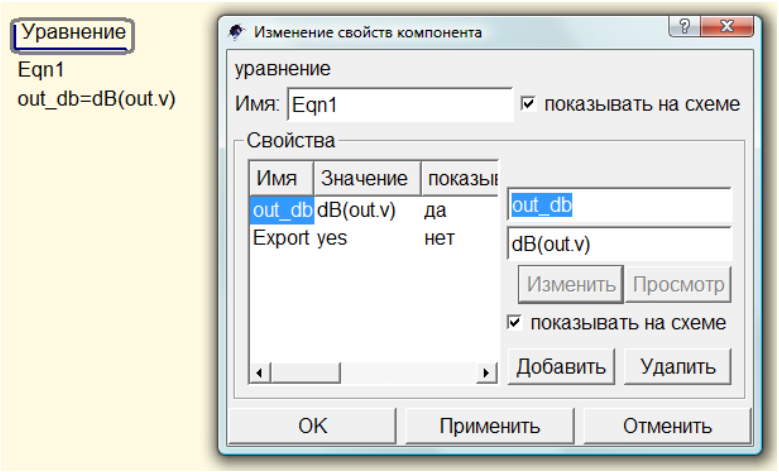


Рис. 2.38. Вид АЧХ каскада усиления на транзисторе

Обычно эту характеристику удобнее рассматривать в относительных единицах, децибелах. Чтобы получить вид АЧХ в децибелах, достаточно вставить уравнение в рабочее поле схемы, используя подменю Вставить, ***Ctrl+<*** на клавиатуре или кнопку инструментального меню. В программе Qucs есть встроенная функция для преобразования АЧХ к виду в относительных единицах, называется она dB. Ее можно применить к точке наблюдения ***out.v***.



В первом текстовом поле диалога указывается название новой переменной, во второе поле записывается само уравнение.

В уравнении можно использовать все встроенные математические функции программы Qucs.

Рис. 2.39. Запись уравнения в диалоговом окне

В этом диалоговом окне можно записать несколько уравнений, используя кнопку ***Добавить***. Можно показывать эти уравнения на схеме или нет, используя опцию ***показывать на схеме***. И с помощью уравнений можно значительно расширить возможности отображения данных в графическом виде или повлиять на процесс симуляции. Второй параметр **Export** компонента **Уравнение** позволяет вам поместить его в набор данных или отказаться от этого.

После применения уравнения к той же схеме выходные данные изменят свой вид, достаточно выбрать в диалоговом окне **Декартовой** диаграммы новую кривую, удалив прежнюю.

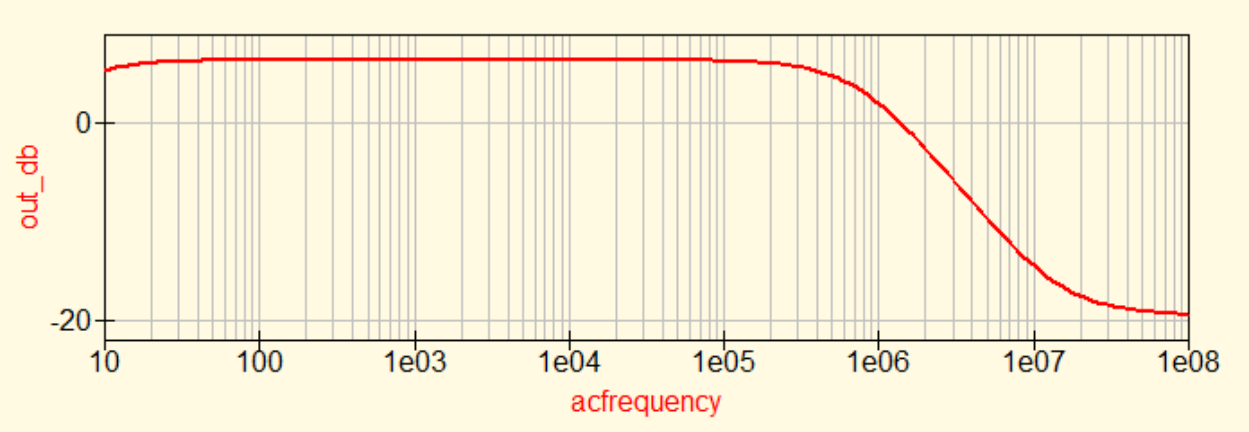


Рис. 2.40. Вид предыдущей АЧХ в децибелах

Используя маркер, о нем чуть ниже, вы можете определить верхнюю граничную частоту, нижнюю граничную частоту или частоту единичного усиления.

Следующая вставка – **Вставить землю**. Без земли не будет работать моделирование аналоговых цепей. При построении схемы можно использовать единственную землю для общего провода схемы. Но в больших схемах получается так, что трудно проводить этот общий провод. Самым разумным, да и вид схемы станет понятнее без лишних линий, применить несколько компонентов схемы под названием **Земля**. То, что этот компонент вошел в подменю **Вставка**, показывает его востребованность.

Вставить вывод. Эта команда вставляет компонент программы Qucs **Порт подсхемы**. Вы можете использовать его и вне механизма организации подсхем, но там он более уместен. Не секрет, что прочитать схему тем легче, чем удачнее в ней выделены основные элементы и чем удачнее скрыты все второстепенные детали. Например, при разработке активного фильтра можно все пассивные RCL компоненты фильтра выделить в подсхему, отладить до включения в основную схему, а в основную схему включить как функциональный блок. Позже мы подробнее рассмотрим, как создать подсхему.

Установить маркер на диаграмме. При рассмотрении построения АЧХ в децибелах уже упоминались маркеры. Именно эта команда позволяет, впрочем, как и кнопка инструментального меню с буквой ***M1***, как и клавиши ***Ctrl+B***, позволяет вставить маркер на график. Достаточно щелкнуть левой клавишей мышки по разделу подменю или кнопке инструментальной панели, чтобы курсор изменил вид.

Осталось поместить его на график и щелкнуть левой клавишей мышки. Маркер покажет значения аргумента и функции в этой точке. Маркер можно переместить по графику – выделить окно отображения данных и нажать клавишу влево/вправо на курсорной клавиатуре. Если точек графика много, а количество точек определяется в настройках вида моделирования, то перемещение маркера не заметно. Нажмите и удержите клавишу на клавиатуре, движение становиться очевидным.

Если вас не устраивает положение окна отображения данных, выделите его щелчком левой клавиши мышки, подцепите его мышкой и перенесите в другое место.

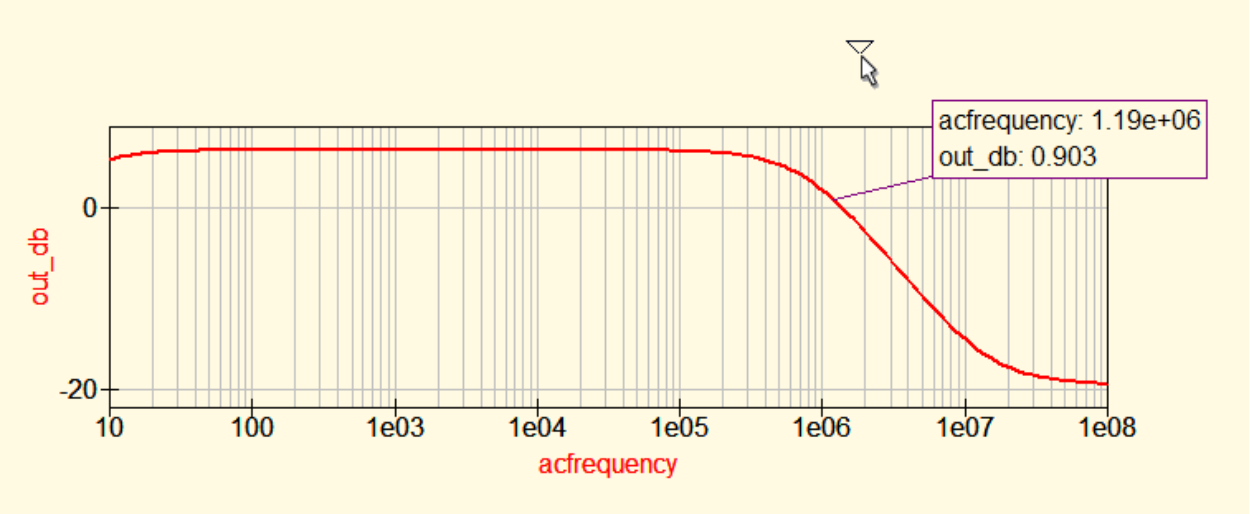


Рис. 2.41. Расстановка маркеров на полученном графике

На графике можно расставить столько маркеров, сколько вам нужно. Для показанной выше кривой вас могут интересовать верхняя граничная частота, нижняя, усиление в плоской области кривой. На рисунке выше показана частота единичного усиления.

Если уменьшить значение конденсатора на входе усилителя, АЧХ усилителя изменится, появится ярко выраженная низшая граничная частота в заданном диапазоне частот. Но при определении этой частоты трудно найти нужную точку. Это связано с тем, что при моделировании был задан параметр количества расчетных точек кривой 190. Увеличьте это количество до 1900, и вы легко переместите маркер в нужную точку.

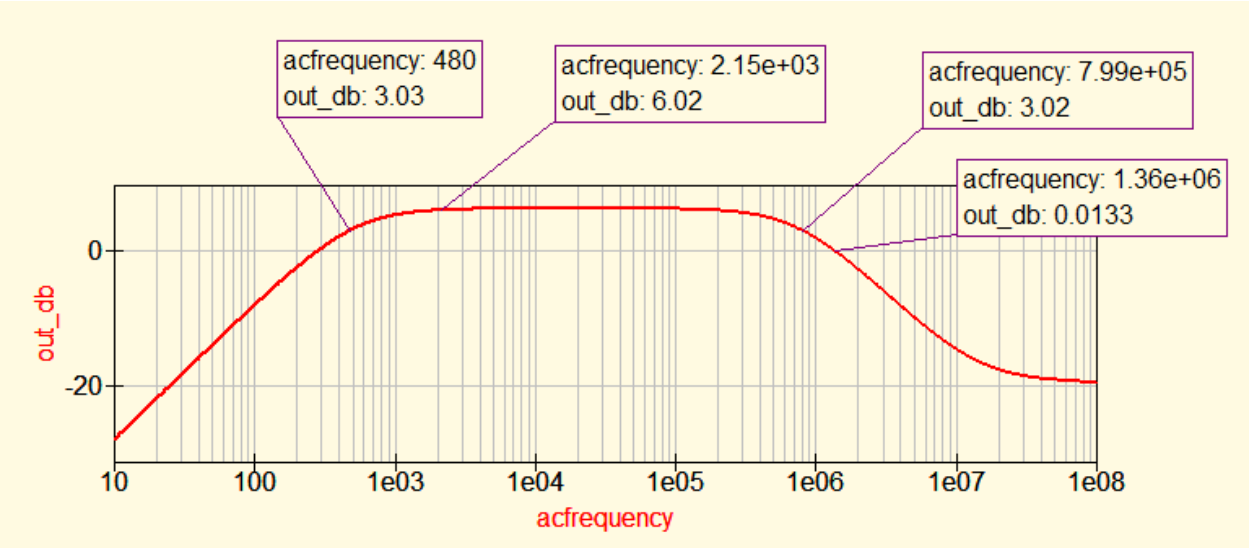
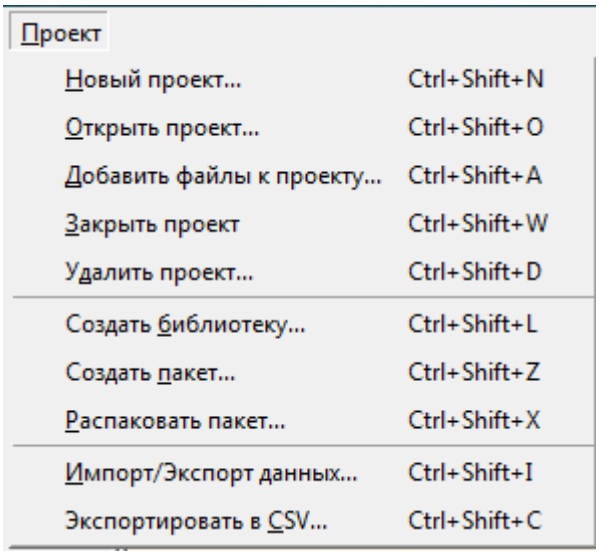


Рис. 2.42. График амплитудно-частотной характеристики усилителя с маркерами

Раздел **Объект VHDL** не активен.

Основное меню, Проект



Самый удобный способ работы с Qucs – это создание нового проекта для предстоящей работы.

В этом случае программа создает папку с названием нового проекта, папка создается в основной директории программы, в папке проекта будут храниться все файлы схемы или схем, файлы данных и графики, можно включать текстовые файлы VHDL и Verilog, файлы плана работы, рабочей тетради.

Даже в том случае, когда вы создаете несколько версий схемы, их удобнее размещать в разных проектах.

Рис. 2.43. Подменю пункта Проект основного меню

Новый проект. Диалоговое окно, которое появится после выполнения этой команды (можно использовать подменю или «горячие клавиши» ***CTRL+Shift+N***), достаточно лаконично.

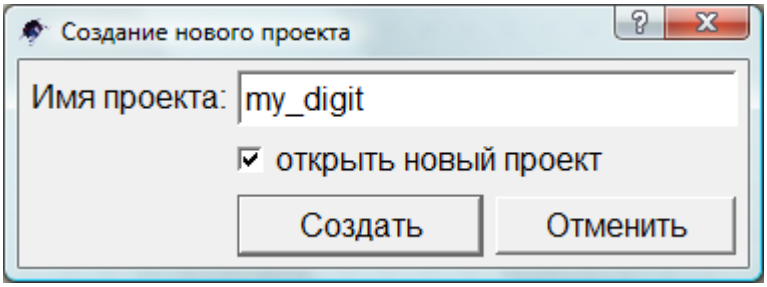


Рис. 2.44. Диалоговое окно создания нового проекта

Введя имя проекта, лучше использовать латиницу, чтобы избежать дальнейшей путаницы с кодировкой, вы можете отказаться от создания проекта, нажав клавишу диалога ***Отменить***, можете создать новый проект с помощью клавиши ***Создать***, и можете оставить флажок опции ***открыть новый проект*** или снять его. Если флажок установлен, вы сразу попадаете в новый проект. Выражается это в том, что в панели навигации открывается дерево проекта, а в рабочей области открывается страница для ввода схемы.

Если флажок опции ***открыть новый проект*** снять, то новый проект будет создан, но вы останетесь в том проекте, с которым работали. Возможно, именно при создании вариантов схемы такой подход уместен – вы можете создать несколько проектов, если у вас есть несколько идей по тому, как следует модифицировать схему.

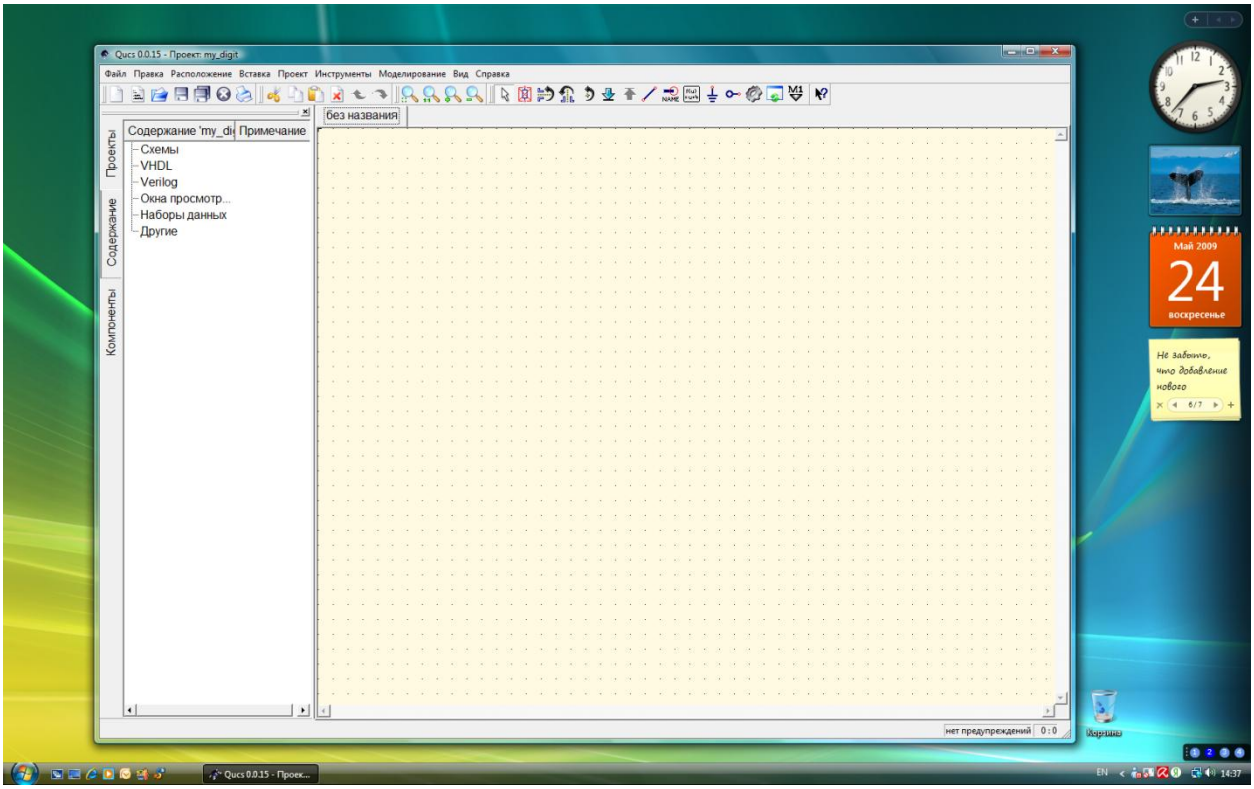
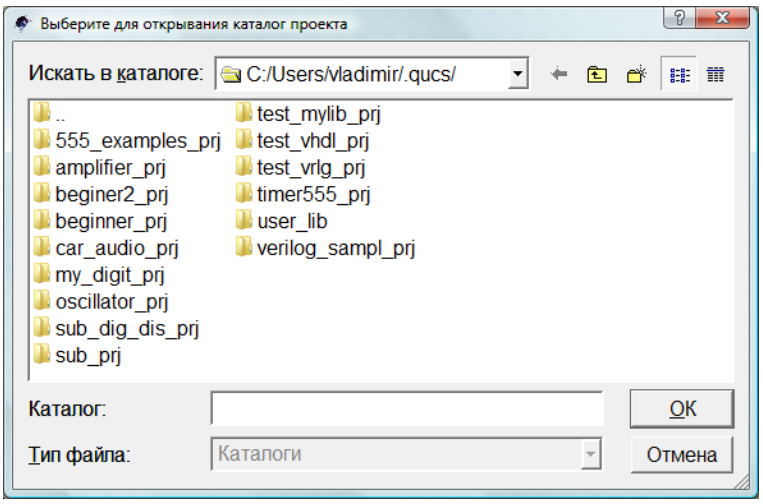


Рис. 2.45. Создание нового проекта

Вновь открытый файл для создания схемы по умолчанию не имеет названия. Вы можете сразу назвать файл схем, используя пункт **Файл**, раздел **Сохранить как**, или можете нарисовать схему, а при сохранении файла схемы вам будет предложено выбрать подходящее имя для вновь созданной схемы. В любом случае диалоговое окно сохранения файла системное.

В дереве проекта панели навигации после сохранения файла схемы появится новый файл в разделе *Схемы*. Вы можете добавить другие файлы, используя раздел ***Добавить файлы к проекту***. Все включенные в проект файлы будут появляться в дереве проекта в соответствующих разделах, определение этого раздела происходит по расширению имени файла.

Любой ранее созданный проект вы можете открыть, используя раздел **Открыть проект**.



Появляющееся диалоговое окно открывает основной каталог (.qucs) программы, где по умолчанию должны находиться все проекты.

Диалоговое окно имеет все средства навигации по файловой системе, которые представлены на инструментальной панели.

Рис. 2.46. Диалоговое окно открывания проекта

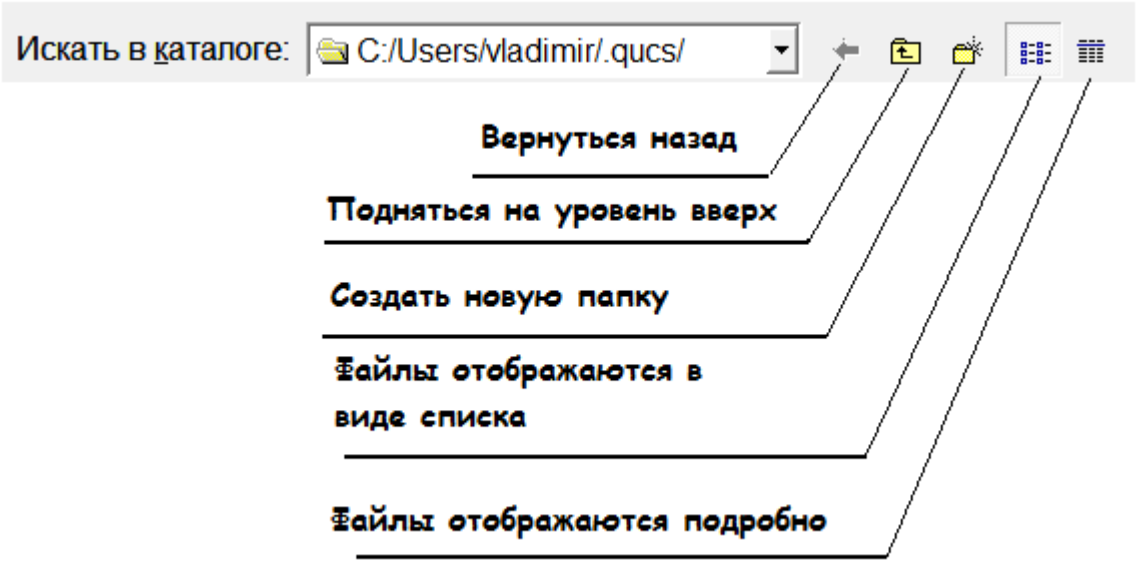


Рис. 2.47. Инструментальная панель диалогового окна открывания проекта

Как и в основной программе при наведении курсора на кнопку появляется подсказка. По умолчанию в диалоге принимается папка проекта, а не файл схемы.

Открыв проект, вы видите все его составляющие на закладке **Содержание** панели навигации, которая открывается автоматически. Если раздел содержания имеет включенные в него файлы, рядом появляется значок в виде плюса. Щелкнув мышкой по этому значку, вы открываете содержание раздела. Ваш проект может иметь несколько файлов схемы, несколько файлов данных и графических страниц отображения данных. Все они будут показаны после щелчка по значку «+».

Команда **Добавить файлы к проекту** открывает диалоговое окно навигации по файловой системе.

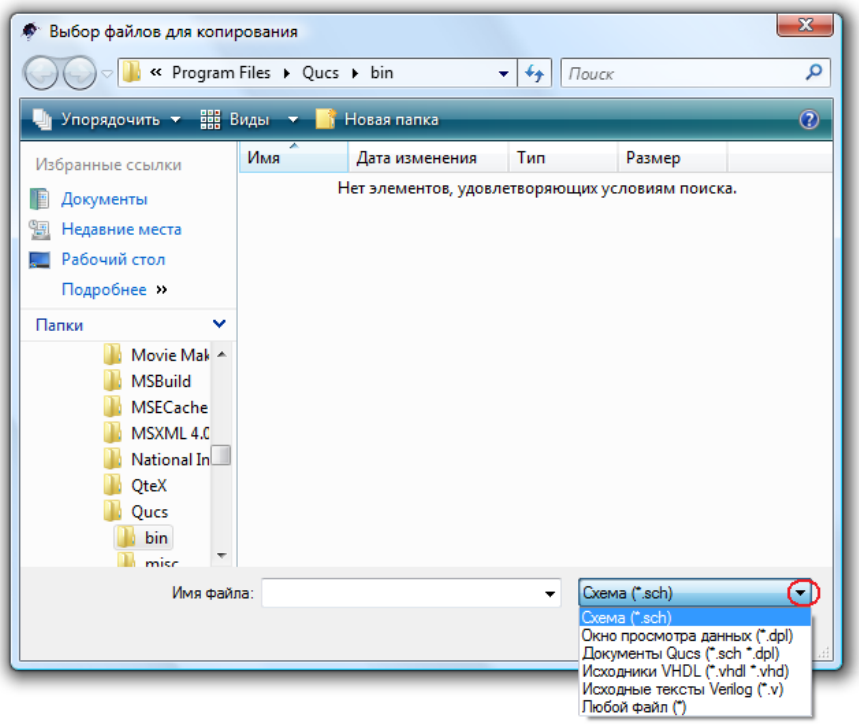


Рис. 2.48. Диалоговое окно раздела **Добавить файлы к проекту**

Рядом с окном ввода имени файла есть окно со списком типов файлов. Если нажать кнопку со стрелкой вниз, то можно увидеть список типовых файлов проекта. Чтобы выбрать файл, вы можете перемещаться в системном навигаторе файловой системы к нужному файлу и выбрать его так, как выбираете файл обычно – щелкнуть по файлу и нажать на кнопку ***Открыть*** диалога.

Добавление файлов VHDL и Verilog к проекту, которые будут впоследствии использованы совместно с цифровыми файловыми компонентами, не вызывает особых вопросов. Но добавление файла отображения графиков может вам понадобится, когда вы добавляете в проект графику другой схемы, но не хотите добавлять саму схему. Добавив к проекту файл данных (с расширением «.dat») по результату уже проведенного моделирования, вы можете обращаться к странице графического представления этих результатов, работая над схемой текущего проекта.

Команда **Закрыть проект** закроет все активные файлы проекта и очистит закладку **Содержание** панели навигации. Если среди файлов проекта были измененные без сохранения нового файла, программа предложит вам прервать выход, сохранить новый вид файлов или отказаться от этого. Файлы, которые были изменены, вписаны в диалоговом окне и возле них установлены флажки. Можно сбросить флажки тех файлов, которые вы не хотите сохранять – достаточно щелкнуть мышкой по флажку.

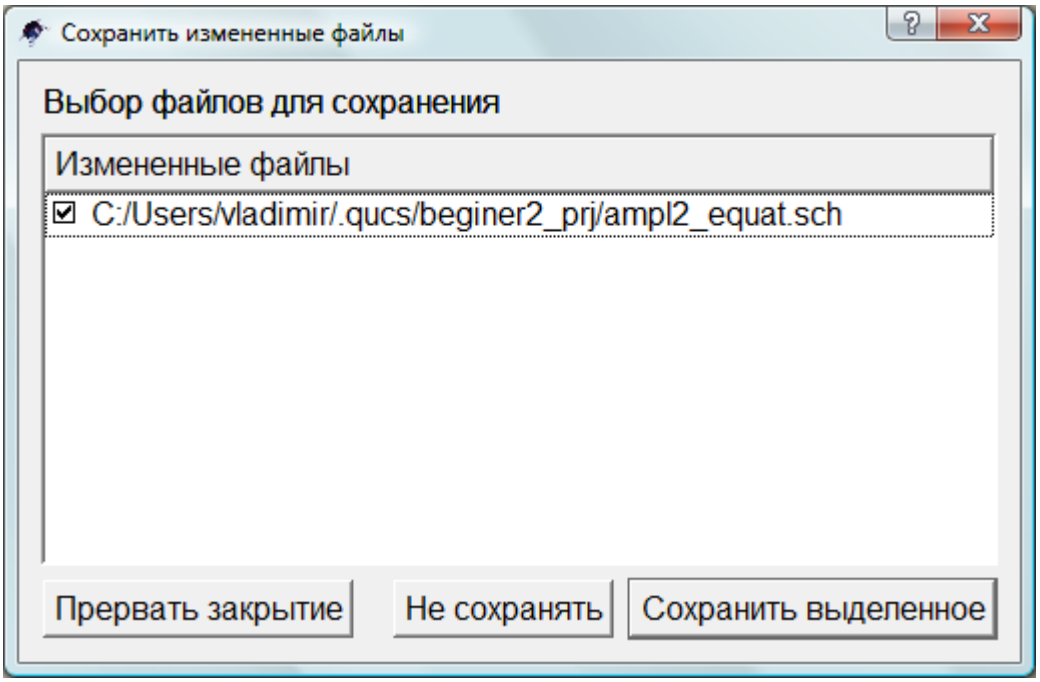


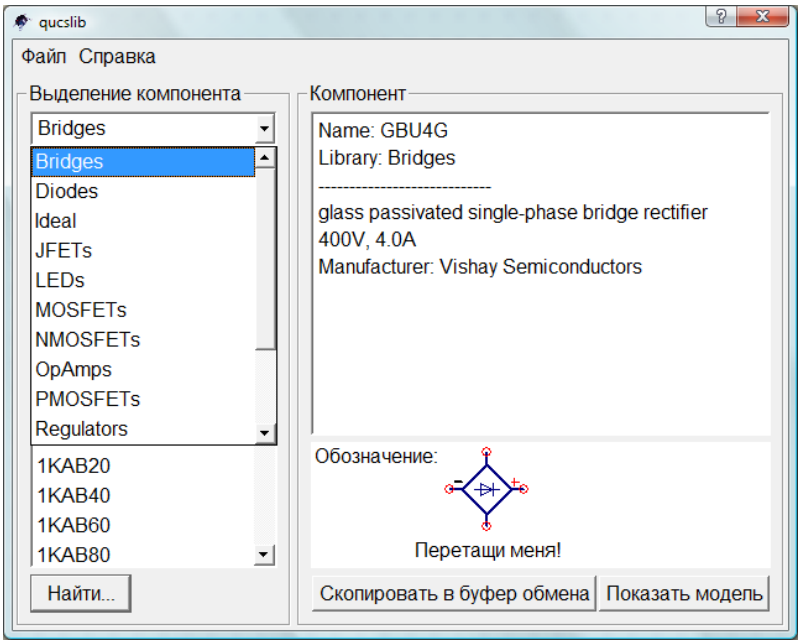
Рис. 2.49. Окно закрывания проекта при несохраненных файлах

Команда **Удалить проект** позволит вам удалять проекты из числа проектов, видимых на закладке **Проекты** панели навигации. Есть два способа удалить проект. Первый – воспользоваться этим разделом подменю **Проект**. Команда открывает диалоговое окно навигации по файловой системе в каталоге основной директории «.qucs». Можно выбрать любой проект для удаления, кроме открытого активного проекта. При попытке удалить открытый проект появится сообщение о том, что это невозможно сделать.

Второй способ удалить проект – выделить его в списке проектов на закладке **Проекты** и воспользоваться кнопкой ***Удалить*** навигационной панели. Удалить открытый проект и в этом случае не получится.

Создать библиотеку. Эта команда требует пояснения. В состав программы Qucs входит библиотека компонентов, состоящая из моделей реальных компонентов: мостов, транзисторов, диодов и т.д. Модели этих компонентов созданы на основе параметров, публикуемых производителями электронных компонентов. Но ничто не мешает вам пополнить библиотеку, создав свою собственную. Именно это подразумевает команда **Создать библиотеку**.

Библиотека компонентов открывается в подменю **Инструменты**, и выглядит так.



Библиотечные компоненты разбиты на группы: мосты, диоды, транзисторы и т.п.

В каждой группе компоненты представлены разными моделями, описание которых есть в правом окне, там же и символ, представляющий компонент.

Как создать свою библиотеку, рассмотрим на примере двоичного счетчика.

Рис. 2.50. Библиотека компонентов из пункта **Инструменты** основного меню

Двоичный счетчик достаточно просто можно построить из четырех D-триггеров. Каждый из них, если соединить его инверсный выход с входом данных, превращается в делитель на два. Соединив их так, как показано на схеме, можно получить двоичный счетчик.

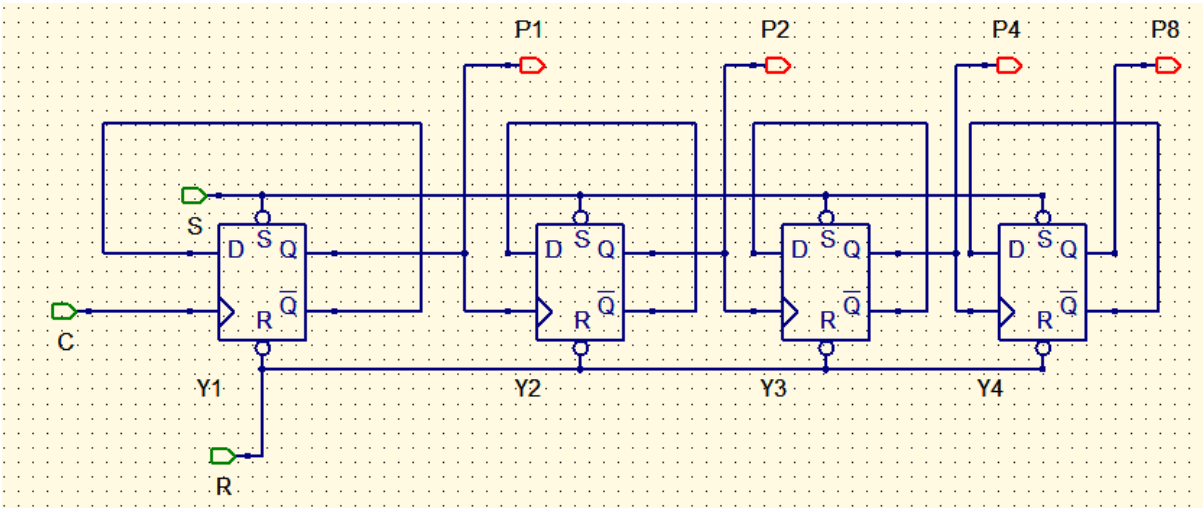
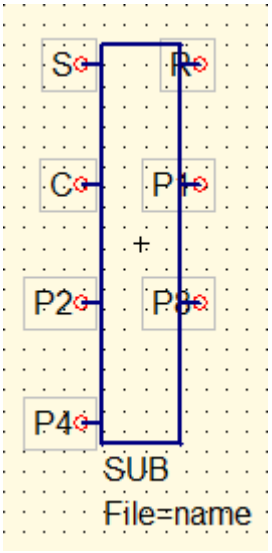


Рис. 2.51. Схема двоичного счетчика

Чтобы удобнее было работать с символом счетчика, параметры сетки были изменены (**Файл**, **Настройки документа**, закладка **Сетка**) – заданные по умолчанию значения 10 заменены на 5. В схему были добавлены компоненты **Порт подсхемы** из состава дискретных компонентов (панель навигации, закладка **Компоненты**). Те выводы, которые предназначены для входа, в свойствах порта были обозначены как параметр «*in*», те, что для выхода, как «*out*». О том, как это сделать, будет рассказано в главе, посвященной подсхемам.

Теперь можно преобразовать символ счетчика. Первое, что нужно сделать, это выбрать команду **Изменить обозначение схемы** в пункте **Файл** основного меню.

После этого вы увидите символ, представляющий вашу схему.



Расположение выводов на рисунке, конечно, никак не соответствует привычному для нас изображению двоичного счетчика.

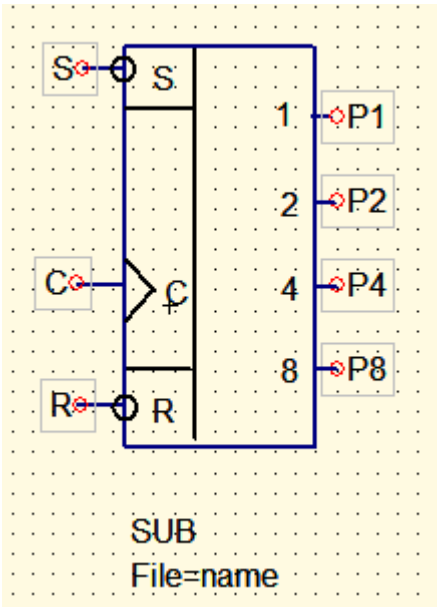
Поэтому первое, что следует сделать, это изменить символ, используя компоненты группы **рисунки** закладки **Компоненты** навигационной панели.

Ко всем элементам рисунка применимы средства редактирования графики: их можно перетащить, подцепив мышкой; можно выделить группу элементов и переместить все вместе; можно повернуть или отобразить любой элемент.

Рис. 2.52. Символ схемы счетчика

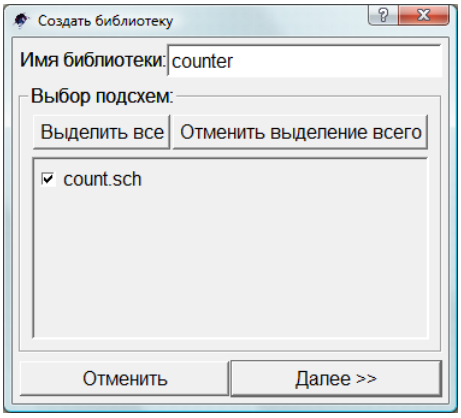
Используя средства редактирования и добавив необходимые детали с помощью графических средств Qucs, рисунок можно приспособить к своим нуждам.

Рис. 2.53. Изменение вида символа



Если использовать этот рисунок в качестве подсхемы, то можно его сохранить в таком виде, и на этом закончить работу. Но при создании библиотеки потребуется немного больше.

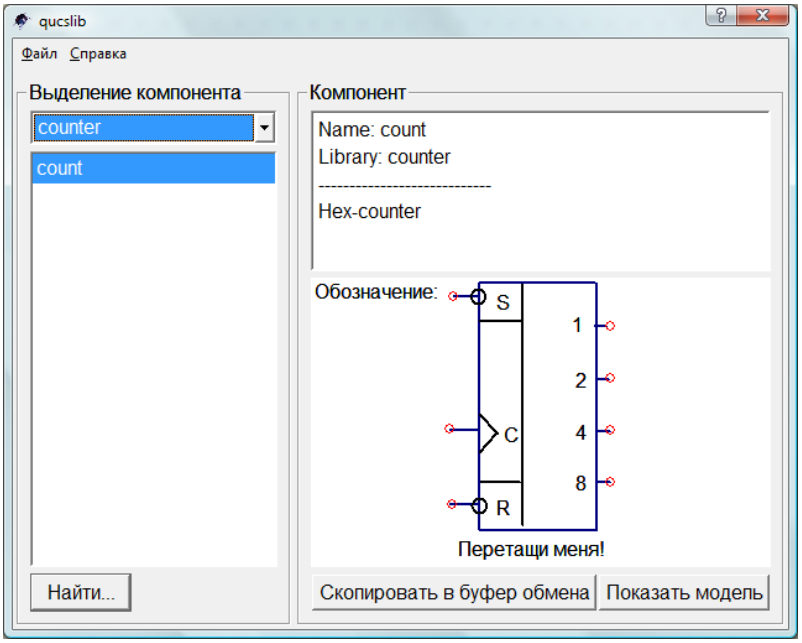
После команды **Создать библиотеку** появится диалоговое окно.



Следует ввести имя пользовательской библиотеки, выбрать файл (или файлы) и нажать кнопку ***Далее>>***, где будет предложено ввести описание нового компонента. Если файлов больше одного, то и описаний будет предложено сделать несколько. Можно отменить создание библиотеки в следующем окне, нажав кнопку ***Отменить***, или создать библиотеку, нажав кнопку ***Создать***.

Рис. 2.54. Диалоговое окно создание библиотеки пользователя

В диалоговом окне появится сообщение об успешно созданной библиотеке, а когда вы создадите новый проект, в котором захотите использовать этот компонент, вы найдете его в разделе **Библиотека компонентов** пункта **Инструменты** основного меню.



Использовать этот компонент можно так же, как и остальные компоненты библиотеки.

Например, можно перетащить его, следуя совету, в рабочее поле чертежа.

В новом проекте лучше использовать параметры сетки, заданные при создании нового компонента.

Рис. 2.55. Новый компонент в Библиотеке компонентов

Во вновь созданном проекте, добавив к новому компоненту необходимые источники, можно моделировать его работу.

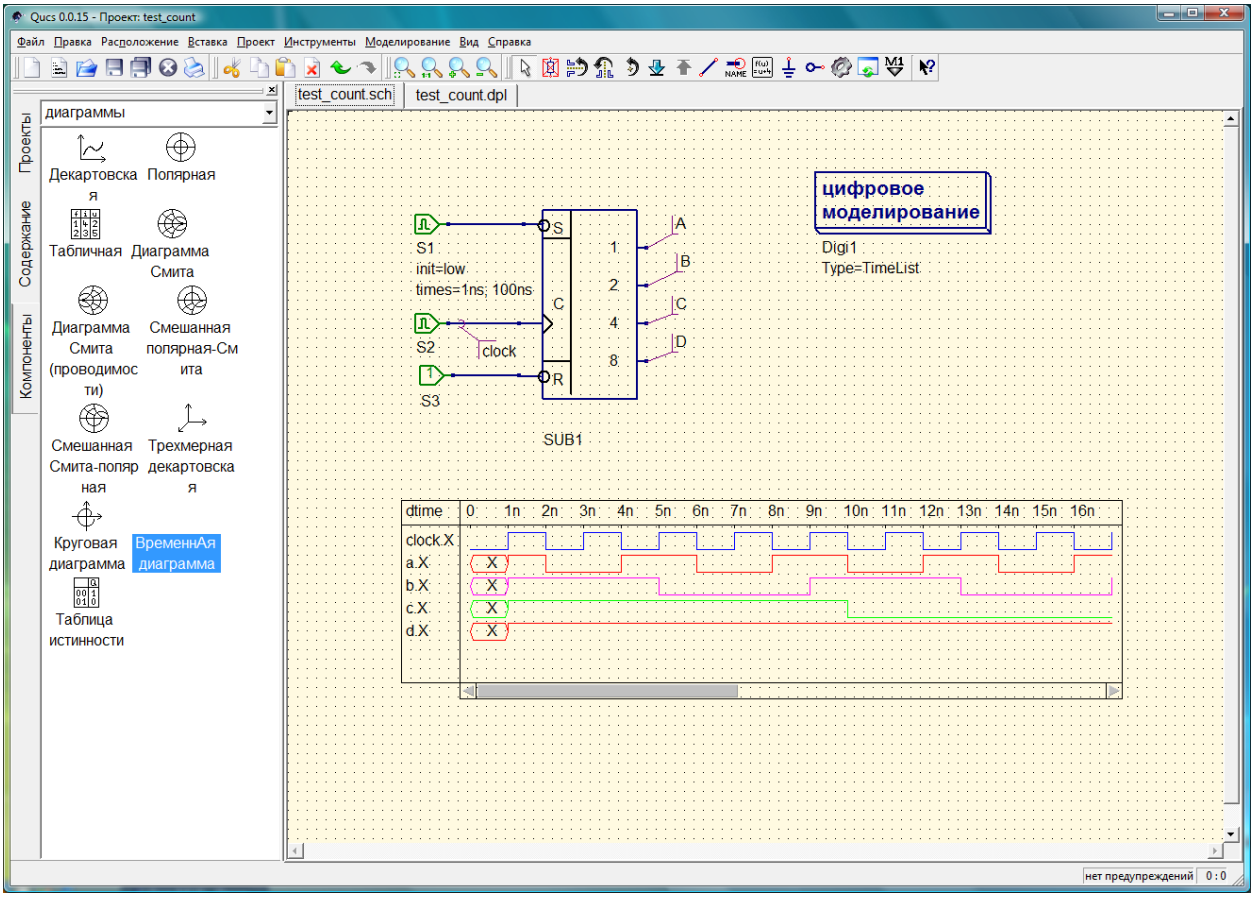


Рис. 2.56. Проверка работы нового компонента в программе

Вы можете обнаружить в директории «.qucs» появление новой папки с именем *user_lib*.

Следующее в списке подменю **Проект – Создать пакет**. Эта команда создает пакет с расширением “.qucs» из существующего проекта. В диалоговом окне можно выбрать все файлы, которые должны входить в пакет. Можно добавить пользовательскую библиотеку и задать имя пакета.

Созданный в предыдущем случае пакет можно открыть командой **Распаковать пакет**. Если проекта с именем проекта, содержащегося в пакете, не существует, то он будет создан. Если библиотека, включенная в проект, уже существует, то в появившемся сообщении об этом будет сказано.

Импорт/Экспорт данных. Данные из проекта могут быть экспортированы для использования в другом проекте или импортированы из другого проекта (или формата), если это понадобится.

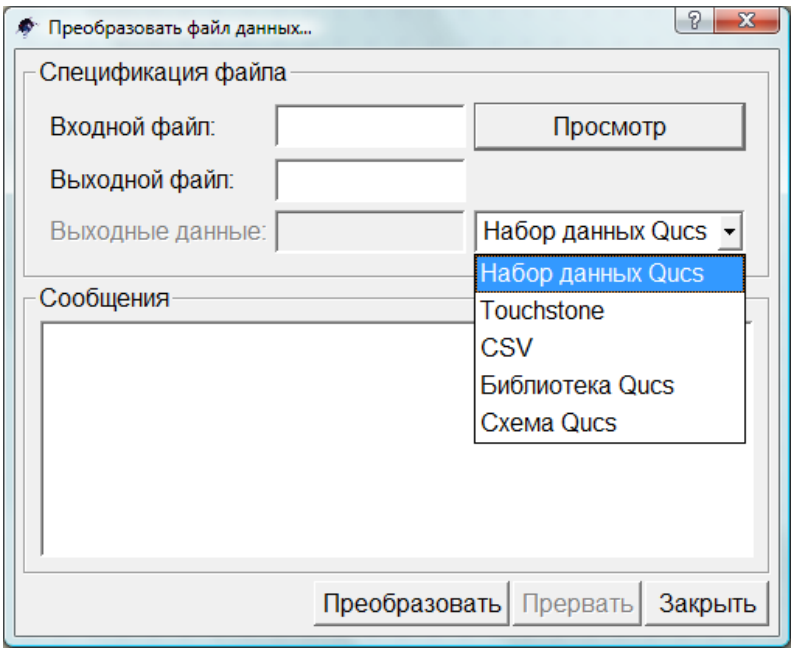


Рис. 2.57. Диалоговое окно импорта/экспорта данных

График диаграммы можно экспортировать в формат CSV – текстовый формат данных, разделенных запятыми – с помощью команды **Экспортировать в CSV**. Для этого следует выделить нужный график на диаграмме на странице отображения данных и щелкнуть по разделу в пункте **Проект** основного меню или нажать на клавиатуре ***Ctrl+Shift+C***. Диалоговое окно сохранения данных системное.

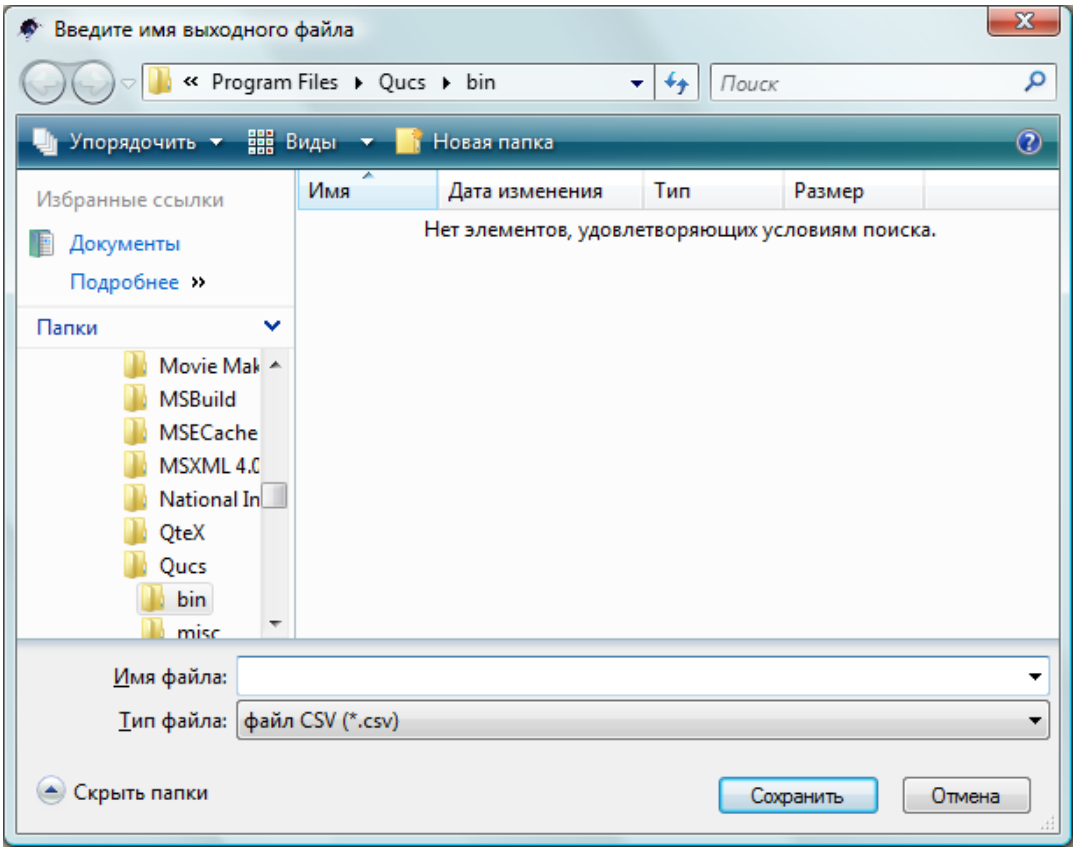
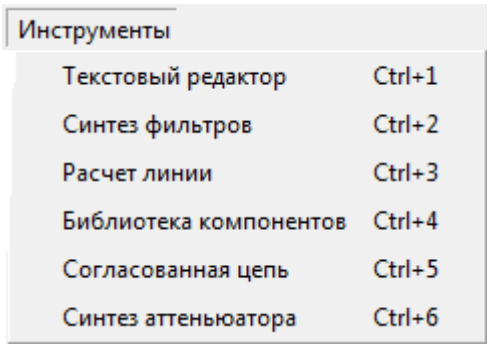


Рис. 2.58. Диалоговое окно сохранения файла в формате CSV.

Основное меню, Инструменты

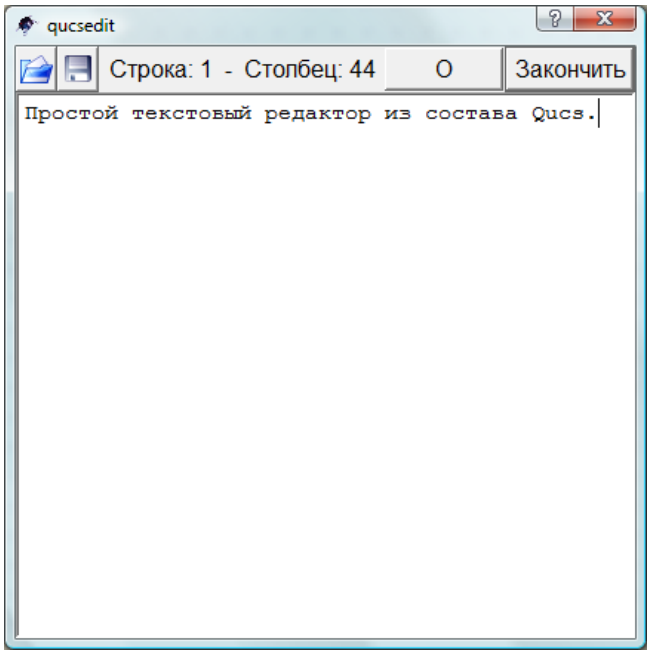


В этом пункте основного меню собраны все инструментальные средства, дополняющие возможности программы Qucs.

Каждый из инструментов требует не только отдельного рассмотрения, но и практики в применении.

Рис. 2.59. Подменю Инструменты

Текстовый редактор. Программа Qucs имеет встроенный простой текстовый редактор. Он гарантирует, что все текстовые файлы, созданный в нем, будут иметь нужный формат для чтения другими программами или компонентами программы Qucs. Речь идет в первую очередь о таких компонентах, как, например, netlist, или файл Verilog.



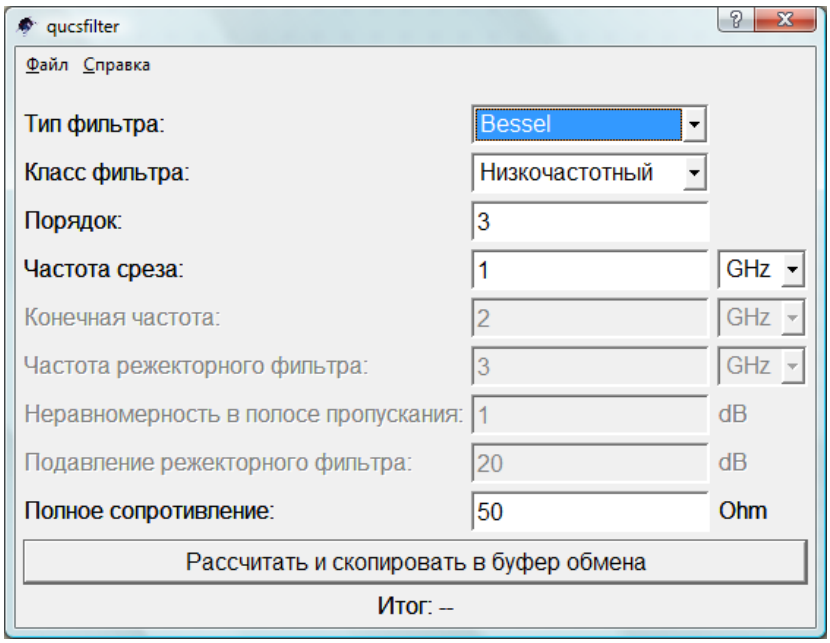
Если вы уверены в своих действиях, и у вас есть свой любимый текстовый редактор, вы можете использовать его.

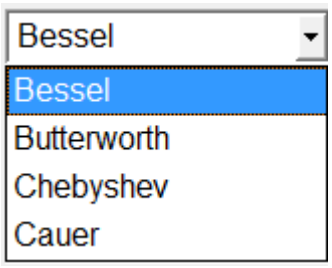
Для этого в настройках программы (пункт **Файл** основного меню) на закладке **Настройки** вы можете указать путь к своему редактору текста в окне ***текстовый редактор***.

При следующей загрузке программы Qucs будет использоваться указанный редактор текста.

Рис. 2.60. Текстовый редактор программы Qucs

Синтез фильтров. Не секрет, что методики расчета фильтров давно и хорошо разработаны. Нет особенных проблем в применении готовых методик. Но, думаю, и не секрет, что пересчитав два-три раза фильтр, теряешь всякий интерес к этому занятию. Программа расчета фильтров помогает избежать лишних расчетов.





Кнопка со стрелкой вниз позволяет выбрать из списка тип фильтра. И ниже класс фильтра

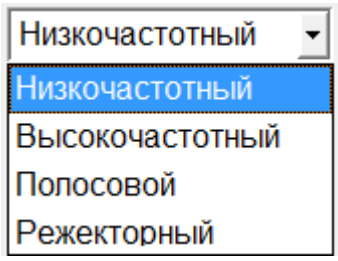


Рис. 2.61. Программа расчета фильтров в Qucs и фильтры, представленные в ней.

Если заполнить окна параметров фильтра, то достаточно нажать кнопку ***Рассчитать и скопировать в буфер обмена***, чтобы получить готовый фильтр. Вид диалогового окна, активность окон параметров, зависит от выбора класса фильтра и типа фильтра. После расчета фильтра появляется сообщение о его успешном окончании. Теперь готовую схему с настройками моделирования можно вставить в рабочее поле программы.

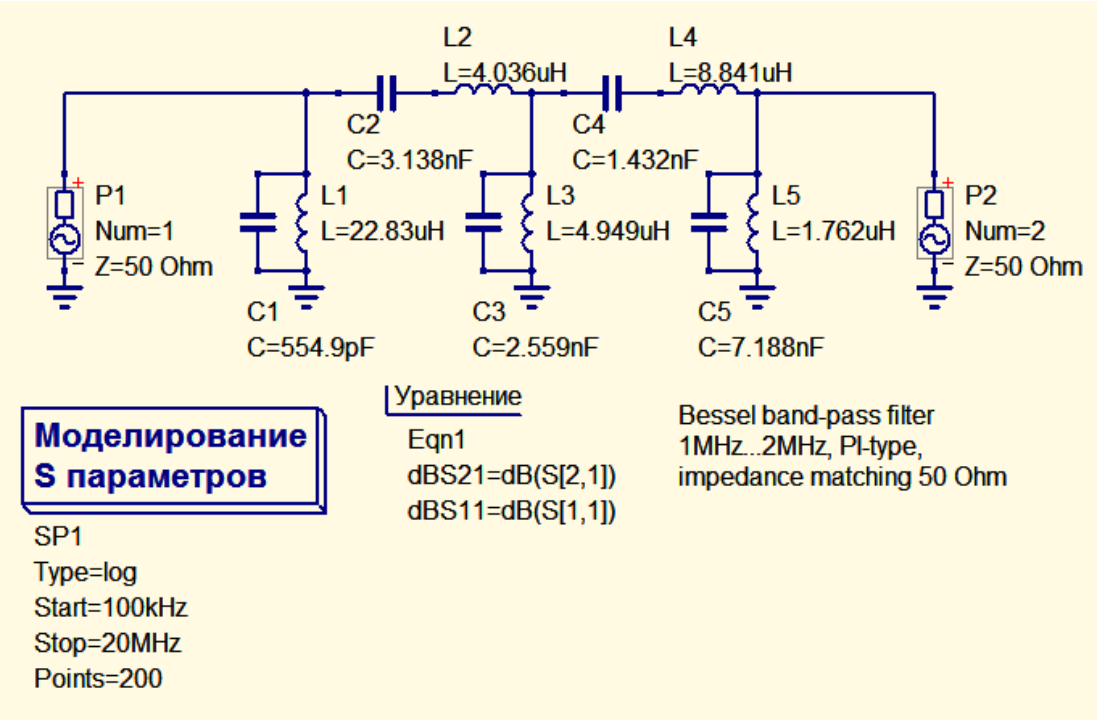
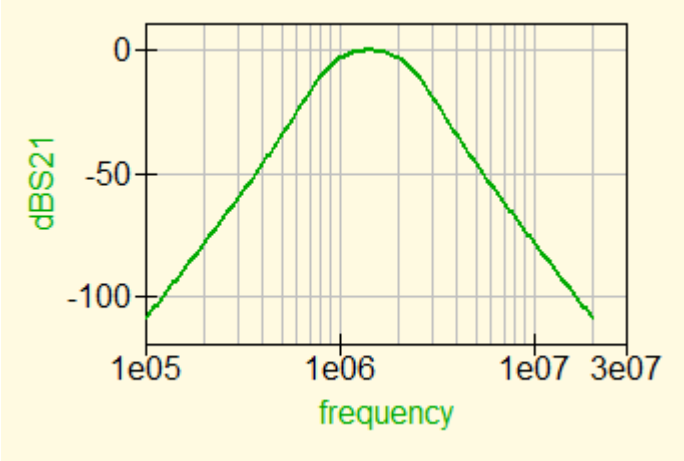


Рис. 2.64. Готовый фильтр в рабочем поле чертежа

И параметры компонентов, и параметры моделирования, и уравнения – все вы получаете в готовом виде, чтобы можно было посмотреть результаты, внести коррективы, если возникнет необходимость, и повторить эту несложную операцию.



Фильтр, схема которого приведена на предыдущем рисунке, ориентирован на использование в радиочастотном диапазоне. Поэтому диаграмму вы можете выбрать не декартову, а другую, на которой отобразить все S-параметры фильтра.

Рис. 2.65. Результаты моделирования работы фильтра

Расчет линий. Еще одна в чем-то схожая программа, облегчающая вам расчетные работы. Параметры, заданные по умолчанию, обрисовывают частотную область, где применение этого инструмента особенно актуально. Однако не стоит забывать, что сегодня все чаще применяют цифровые компоненты в сочетании с аналоговыми, где частота цифровых элементов схемы давно выходит за пределы нескольких сотен килогерц. При разработке печатных плат образуется много линий, параметры которых очень полезно увидеть до физической реализации устройства.

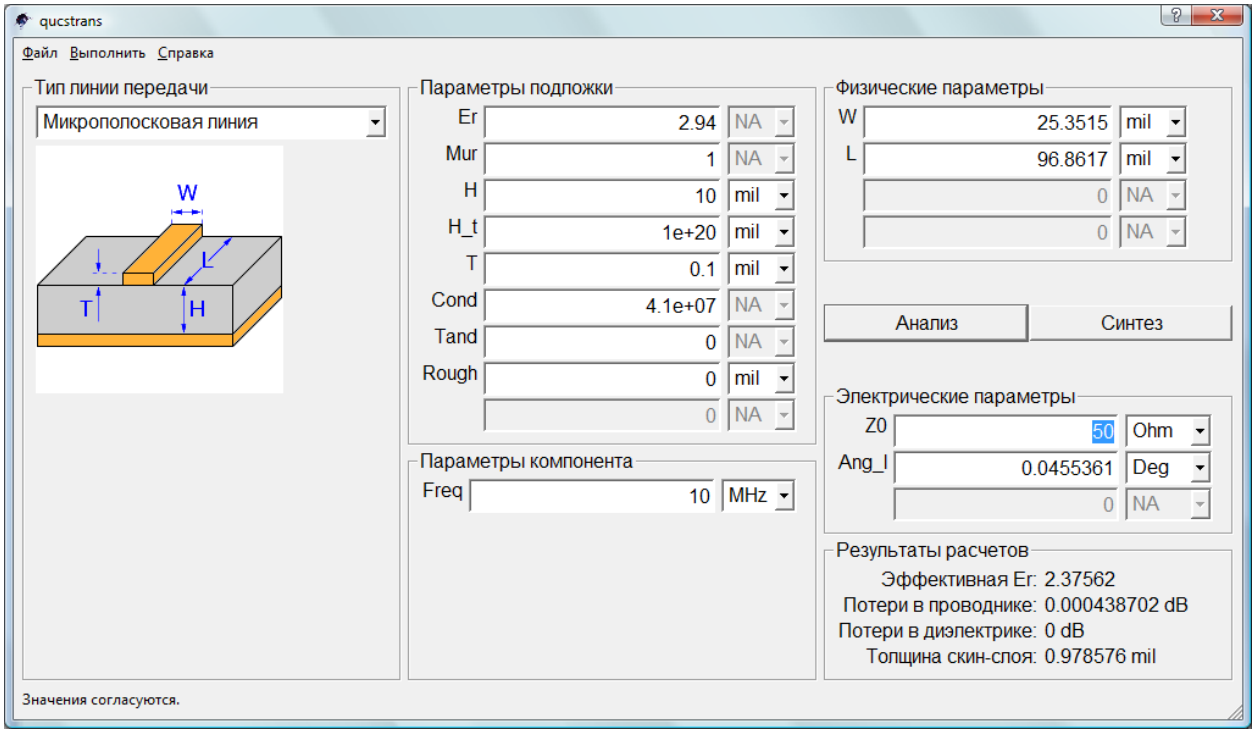


Рис. 2.66. Программа расчета линий в Qucs

Так для согласования линии при входном сопротивлении 50 Ом на частоте 10 МГц вы можете использовать те результаты, которые получены с помощью синтеза (кнопка ***Синтез***) после ввода Z0 = 50 Ом. Единицы размера, как «mil» – одна тысячная дюйма – могут быть заменены метрами, сантиметрами и т.д. Расчеты возможны для нескольких типов линий.

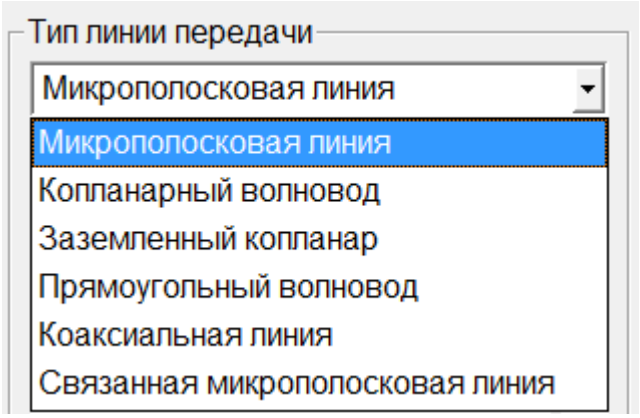
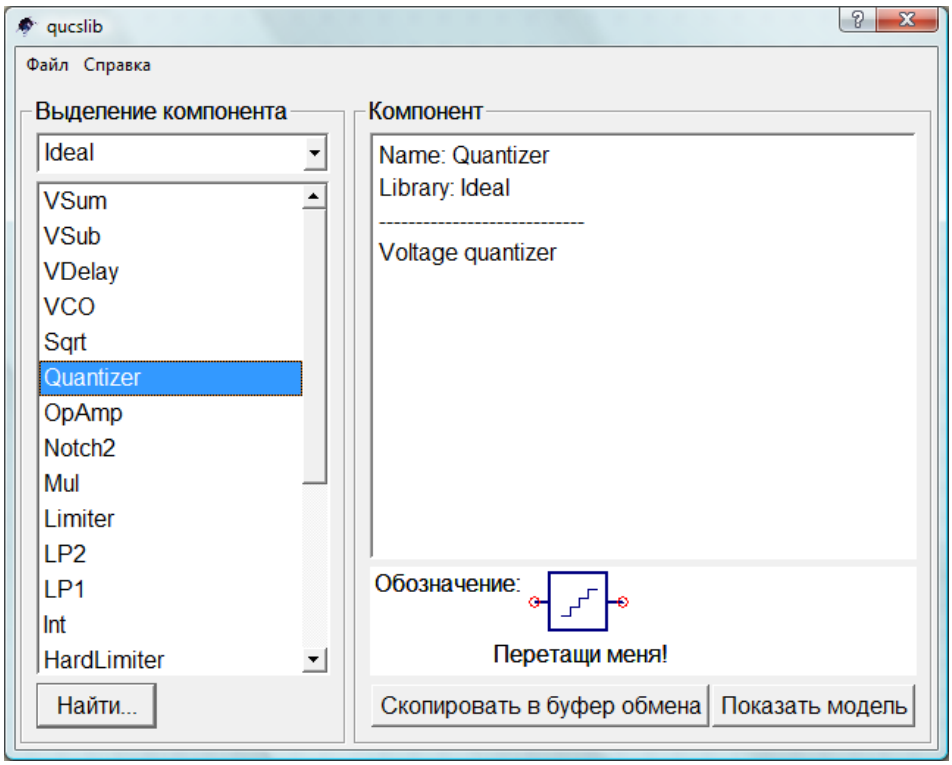


Рис. 2.67. Типы линий в программе расчета

Библиотека компонентов. О библиотеке компонентов уже говорилось выше. В последней на сегодняшний день версии появилась новая группа компонентов – **Ideal**.



Эти компоненты, используемые как подсхемы, полезны в образовательных целях, равно как и на практике.

Рис. 2.68. Группа идеальных компонентов в библиотеке

Такая группа компонентов, как транзисторы, достаточно обширная, очень ярко показывает разницу между применением транзистора из набора компонентов группы **нелинейные компоненты** на закладке **Компоненты** панели навигации и транзисторами из библиотеки.

Достаточно посмотреть на АЧХ однокаскадного усилителя, где последовательно применен транзистор общего применения и транзисторы из библиотеки.

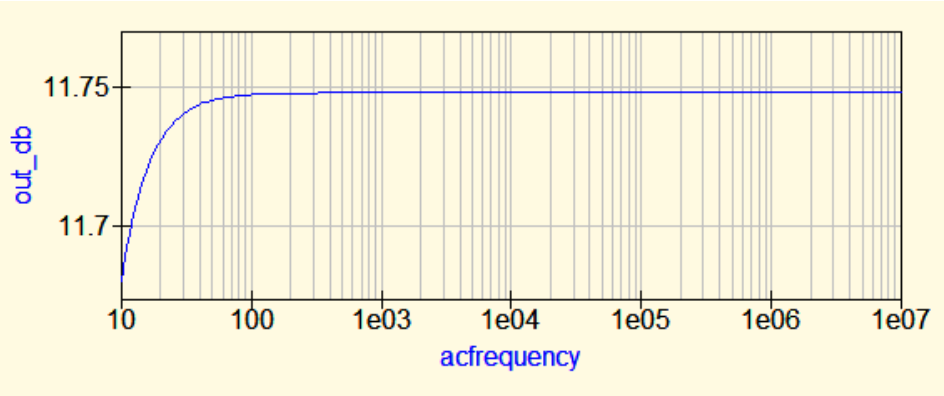


Рис. 2.69. Амплитудно-частотная характеристика с транзистором из группы **Компоненты**

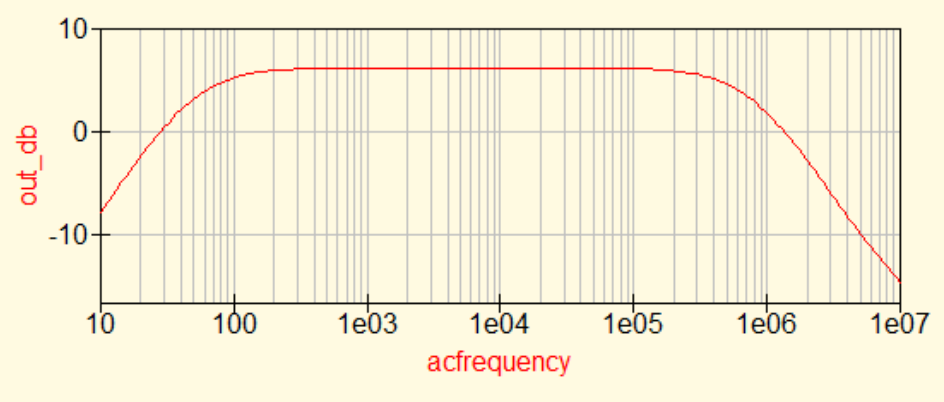


Рис. 2.70. Амплитудно-частотная характеристика с транзистором 2DA1774R

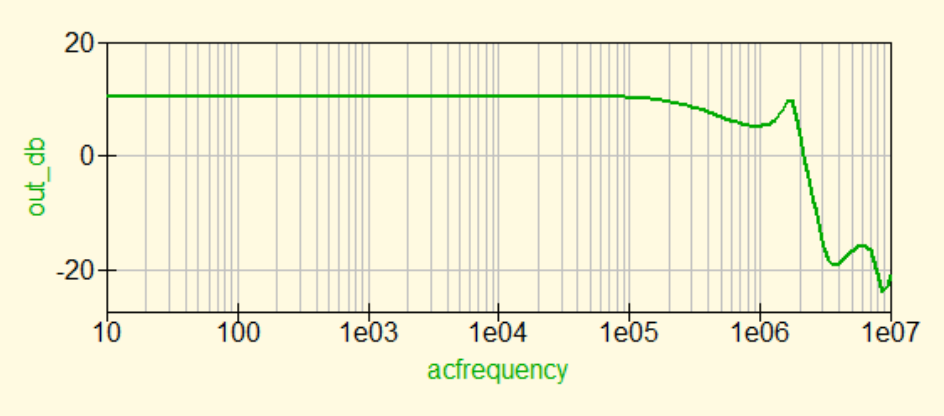


Рис. 2.71. Амплитудно-частотная характеристика с транзистором 2N2955

Диалоговое окно библиотеки компонентов позволяет вам посмотреть модель (кнопка ***Показать модель***), скопировать ее буфер обмена, и имеет еще одну полезную функцию – поиск (кнопка ***Найти***).

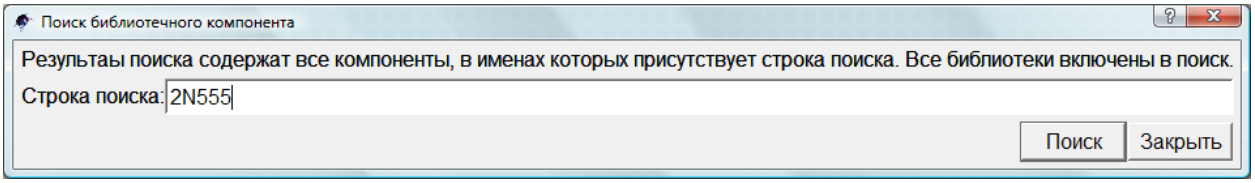


Рис. 2.72. Диалоговое окно поиска компонента

Результаты поиска будут отображены в основном окне библиотеки компонентов.

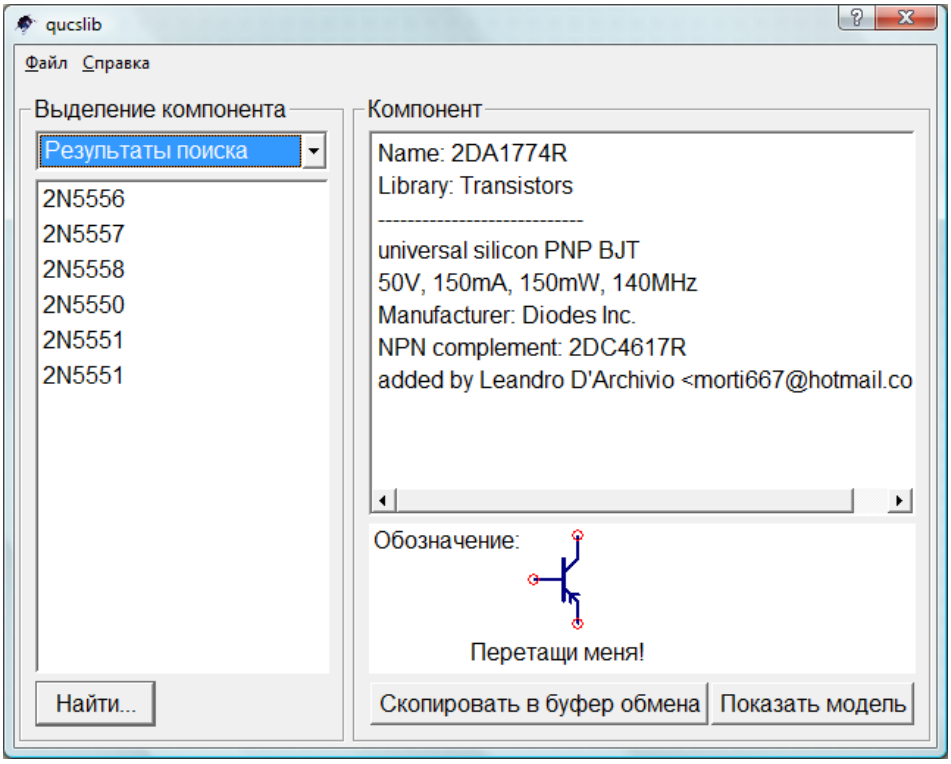


Рис. 2.73. Результаты поиска в окне библиотеки компонентов

Согласованная цепь. Еще одна полезная программа в составе Qucs. Вот как выглядит результат.

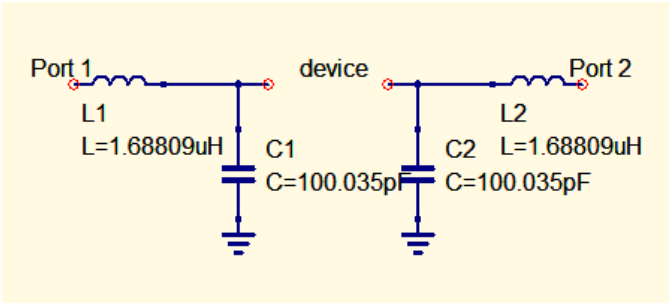


Рис. 2.74. Результат работы программы расчета согласованной цепи

А данные, которые были заложены при расчете этой цепи, вводятся в диалоговом окне программы.

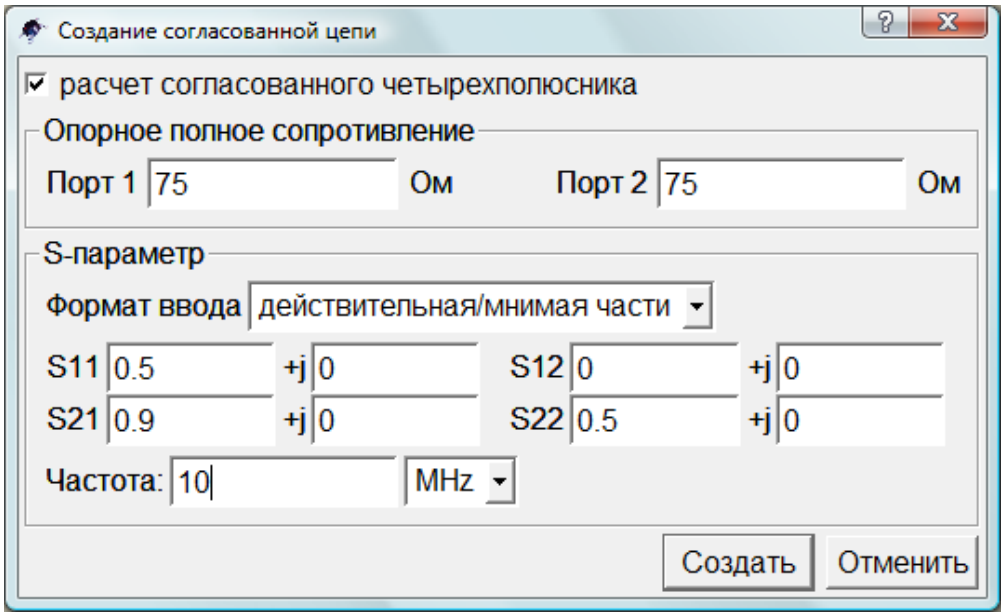
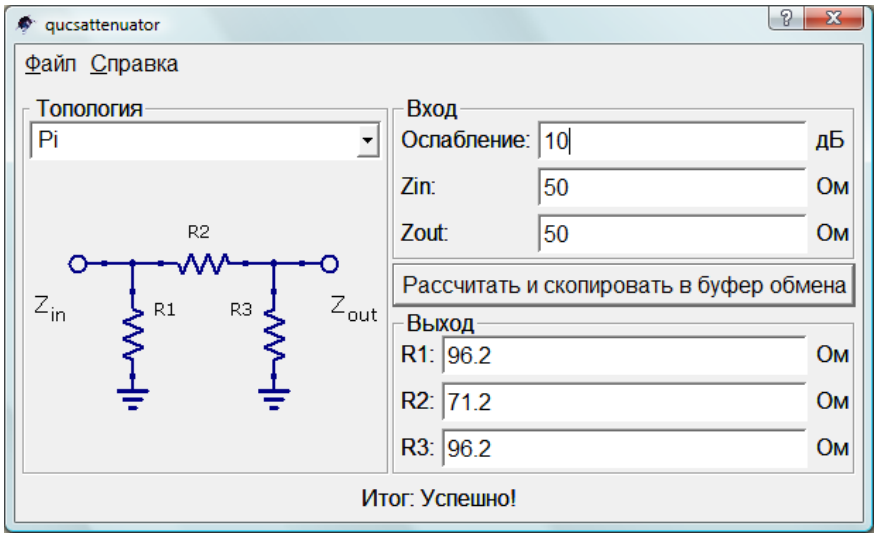


Рис. 2.75. Ввод данных для расчета согласованной цепи

Зная входной и выходной импеданс, S-параметры, вы можете получить расчет цепи. Для этого достаточно нажать кнопку **Создать**. Полученный результат вы можете сразу вставить в рабочее поле чертежа – он появляется в виде контура, прикрепленного к концу курсора.

Очень похожими свойствами обладает и следующий раздел пункта Инструменты основного меню – **Синтез аттенюатора**. Это тоже полезная расчетная программа.



Задав ослабление, полные входное и выходное сопротивления, достаточно нажать на кнопку ***Рассчитать и скопировать в буфер обмена***, чтобы получить результат.

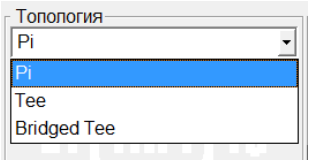
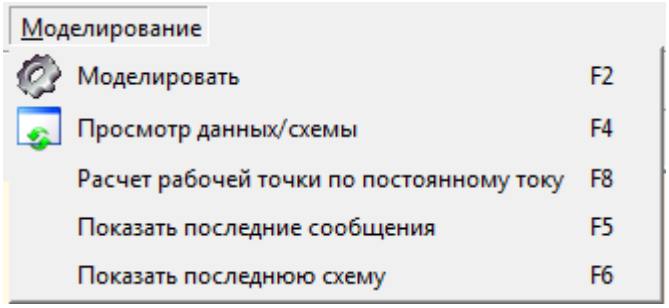


Рис. 2.76. Программа расчета аттенюатора

Есть несколько топологий аттенюатора, поддерживаемых программой. После расчета вы можете вставить полученный аттенюатор в схему из буфера обмена.

Основное меню, Моделирование

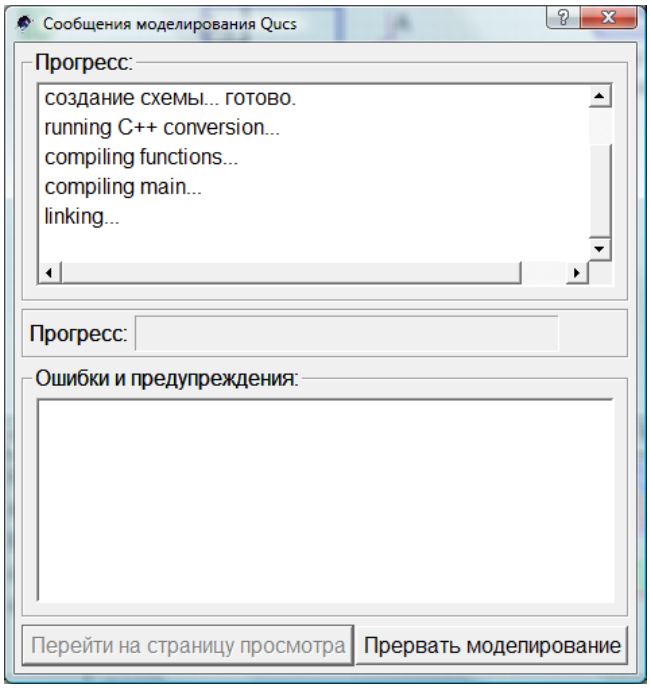


Моделирование или симуляция – основная функция программы Qucs.

Конечно, можно использовать ее для черчения схемы, можно получить файл netlist.

Рис. 7.78. Подменю пункта Моделирование

Открывает подменю команда **Моделировать**. Вы можете использовать клавишу F2 на клавиатуре, вы можете использовать кнопку инструментальной панели. Но прежде, чем моделировать схему, вы должны выбрать вид моделирования из группы **виды моделирования** закладки **Компоненты** панели навигации. И, чаще всего, нужно настроить процесс моделирования.



В верхнем окне отображен, собственно, процесс, что для некоторых видов моделирования состоит из ряда операций.

Ниже в процентах показан прогресс.

Если есть ошибки и предупреждения, они появляются в нижнем окне.

Вы можете прервать процесс, используя кнопку ***Прервать моделирование***.

Рис. 7.79. Отображение процесса моделирования

Процесс моделирования может проходить почти мгновенно, но может занимать много времени. Все зависит от сложности расчета схемы и мощности компьютера. Иногда процесс моделирования не может быть завершен из-за расходимости уравнений. Можно попытаться настроить моделирование несколько иначе, используя, например, другие методы решения уравнений или меняя время моделирования. Если вы попытаетесь запустить команду, когда активно окно отображения данных, а не схемы, вы немедленно получите сообщение об ошибке. Моделирование следует запускать со страницы ввода схемы. Подробнее о моделировании будет написано в главе, посвященной видам моделирования.

Просмотр данных/схемы. Эта команда (клавиша ***F4*** или закладка страниц проекта) переключает страницы проекта: страницу схемы и страницу обзора данных виде диаграмм.

Расчет рабочей точки по постоянному току. Это важный этап работы при построении транзисторных, например, усилителей.

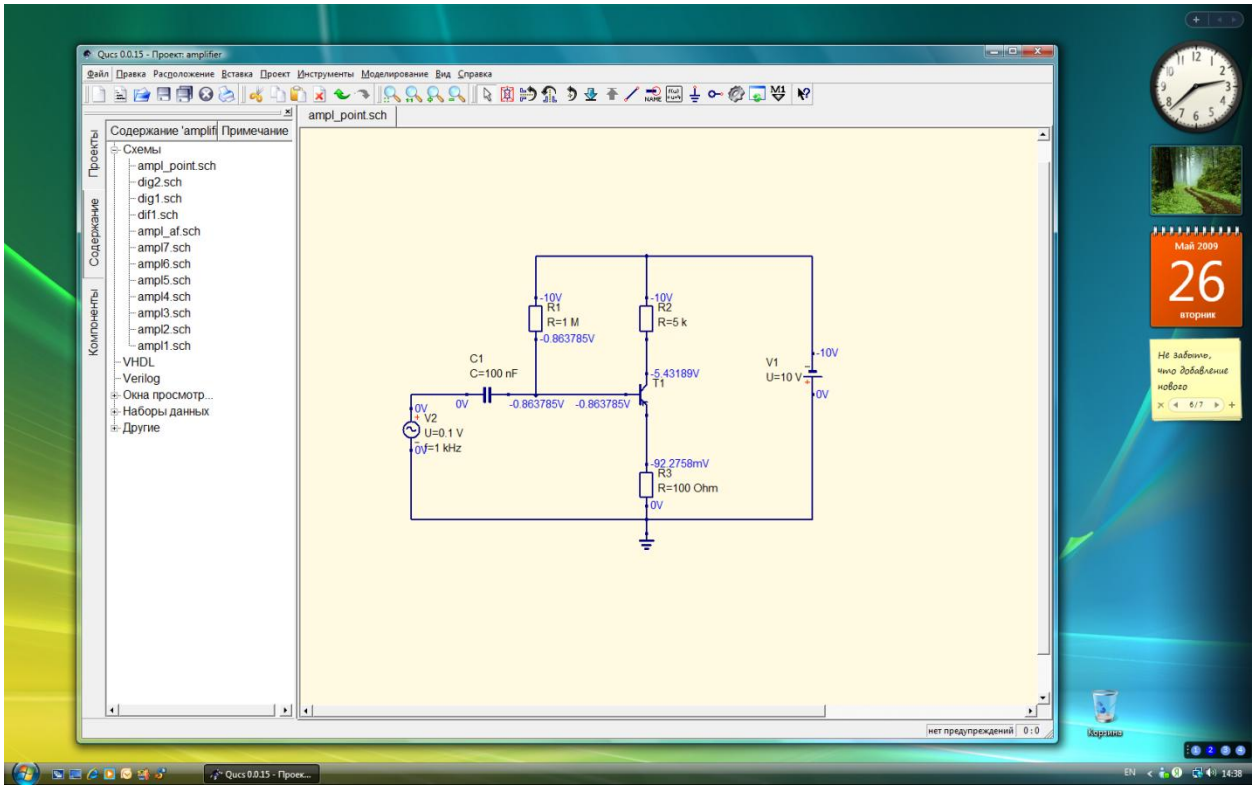


Рис. 2.80. Результат выполнения расчета рабочей точки

В данном случае рассчитываются все напряжения вашей схемы по постоянному току. Но, если вы добавите измерители тока, то по этой команде (клавиша ***F8***) рассчитаются и токи.

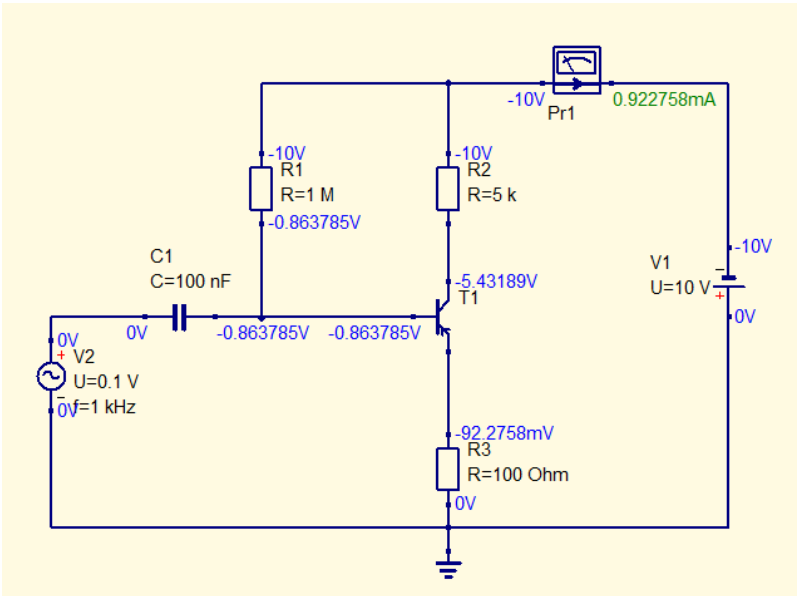
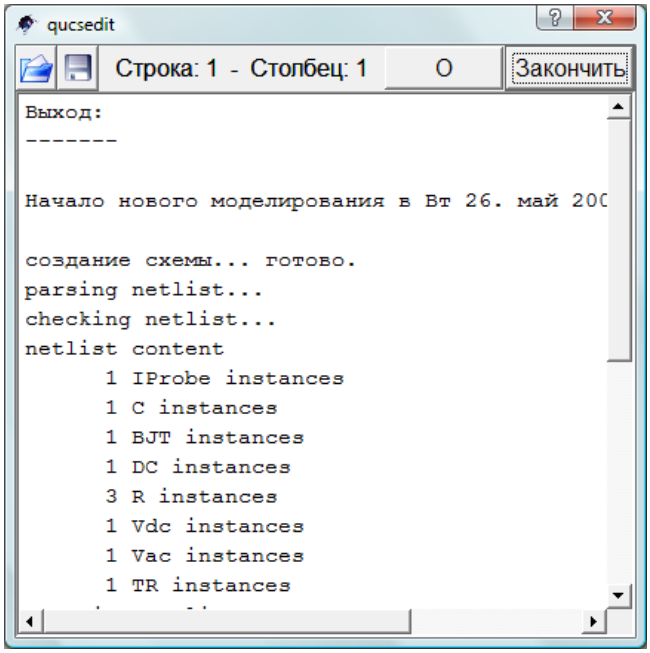


Рис. 2.81. Расчет рабочей точки по постоянному току

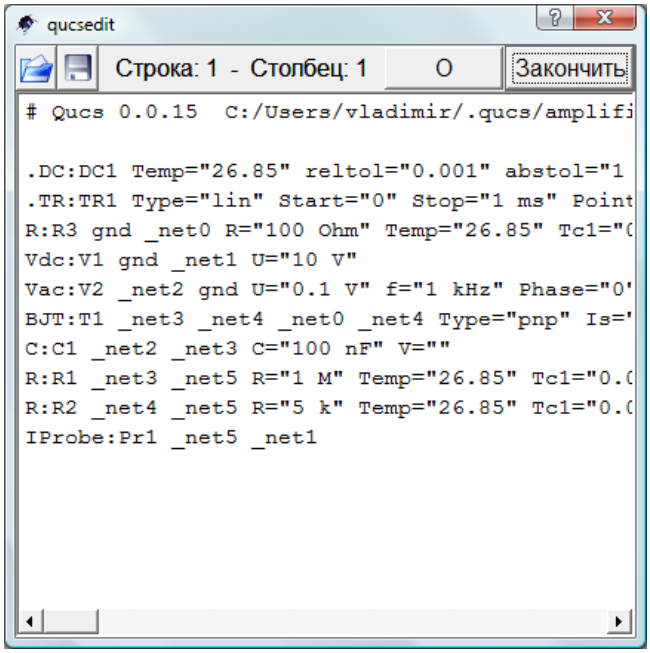
Показать последние сообщения. По этой команде (клавиша ***F5***) открывается текстовый редактор (или тот, что вы указали для использования в качестве текстового редактора), в котором появится сообщение о последнем моделировании. Оно появлялось в верхнем окне диалога процесса моделирования.



Далеко не всегда процесс моделирования проходит гладко. Приходится настраивать что-то, менять что-то в схеме. Полезно бывает вернуться к сообщениям, чтобы уточнить, в каком месте моделирование было прервано.

Рис. 2.82. Последнее сообщение о процессе моделирования

Показать последнюю схему. Собственно, не схему, а netlist, который открывается в редакторе.

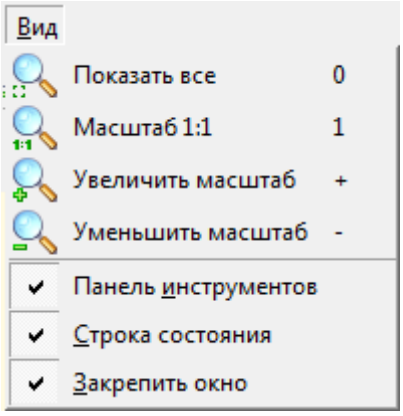


Файл этот текстовый, его можно сохранить под другим именем, чтобы использовать с другим симулятором, подправив формат файла.

Или для того, чтобы привести его к виду, принимаемому программой разводки печатной платы.

Рис. 2.83. Вид netlist в формате Qucs

Основное меню, Вид



Это предпоследний пункт основного меню, позволяющий настроить вид активного листа: схемы или диаграмм.

Рис. 2.84. Подменю **Вид**

Показать все. Работая с большими схемами, вы выбираете такое положение листа и масштаб, которые позволяют вам комфортно выполнять все нужные операции, например, при создании схемы. Но, когда схема завершена, она может отображаться на листе только частично. Кроме схемы на листе должны быть выбранные виды моделирования, их может быть несколько, могут появиться уравнения, и вы можете скопировать с листа диаграмм нужные вам графики для налаживания схемы.

Колесико мышки позволяет вам перемещать лист вниз и вверх. При нажатой клавише ***Shift*** движения колесика перемещают схему влево-вправо. А нажав и удерживая клавишу ***Ctrl***, вы с помощью колесика мышки можете масштабировать схему.

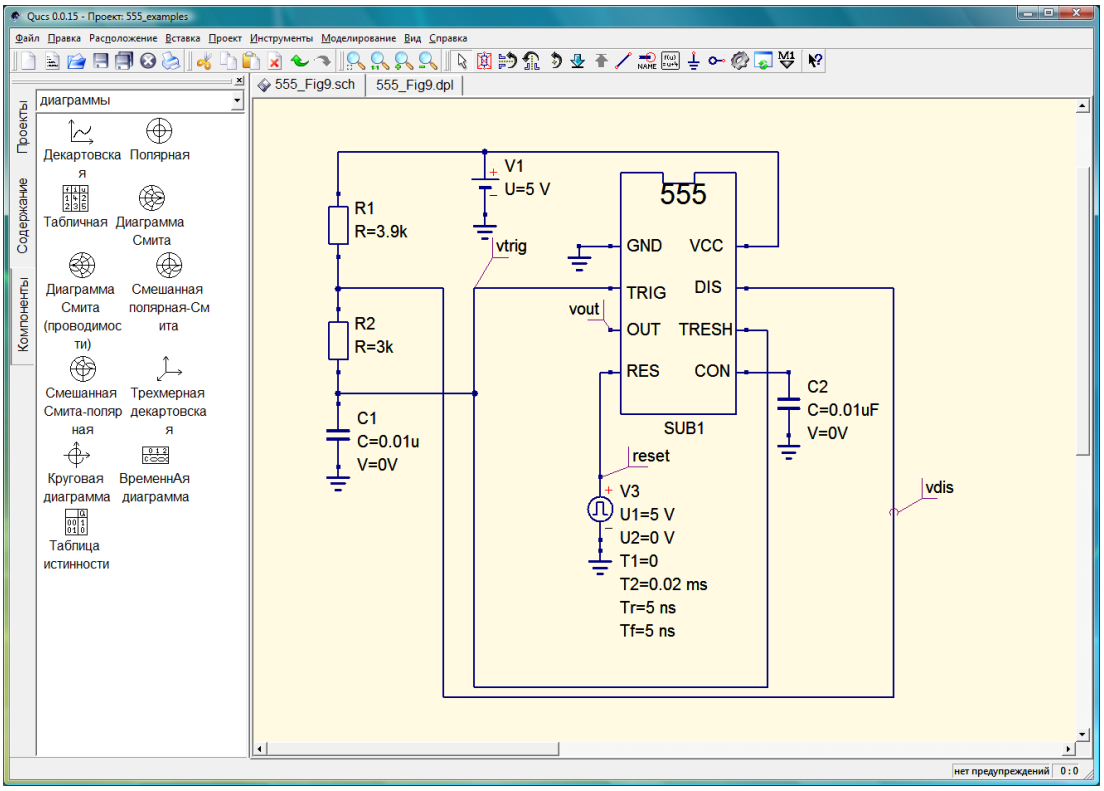


Рис. 2.85. Вид рабочей схемы

После команды **Показать все** вид рабочего листа изменится.

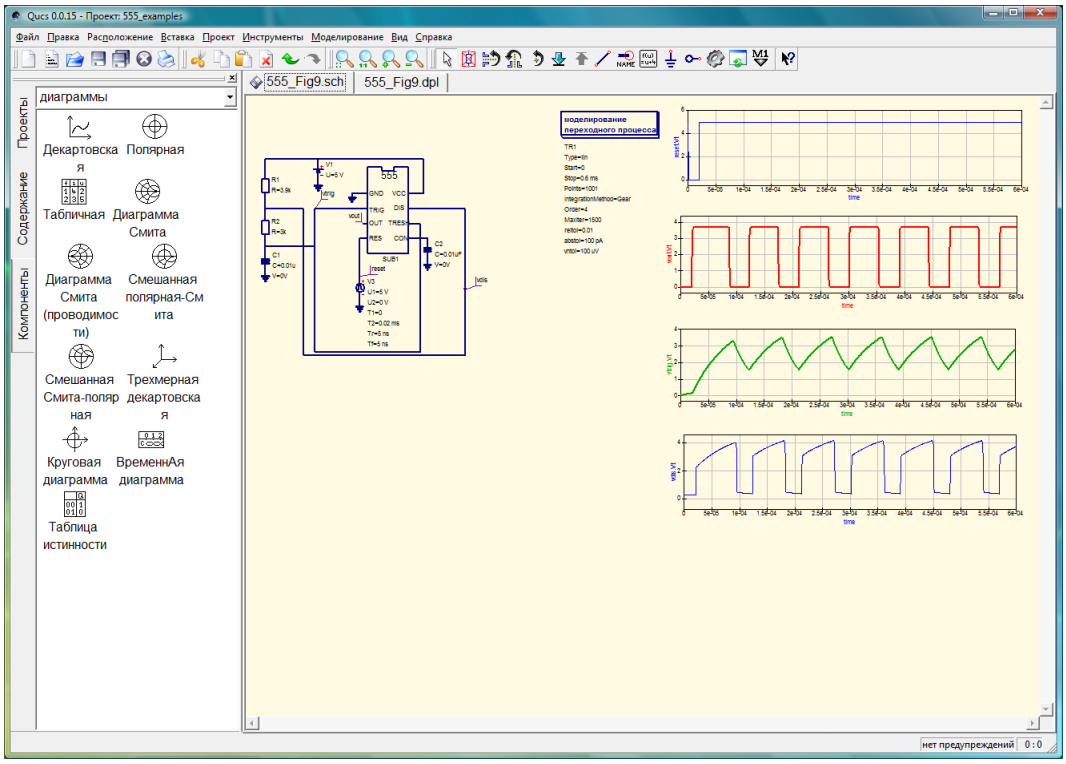


Рис. 2.86. Полный вид рабочего листа

Масштаб 1:1. Посмотрите, как выглядит предыдущая схема после выполнения команды.

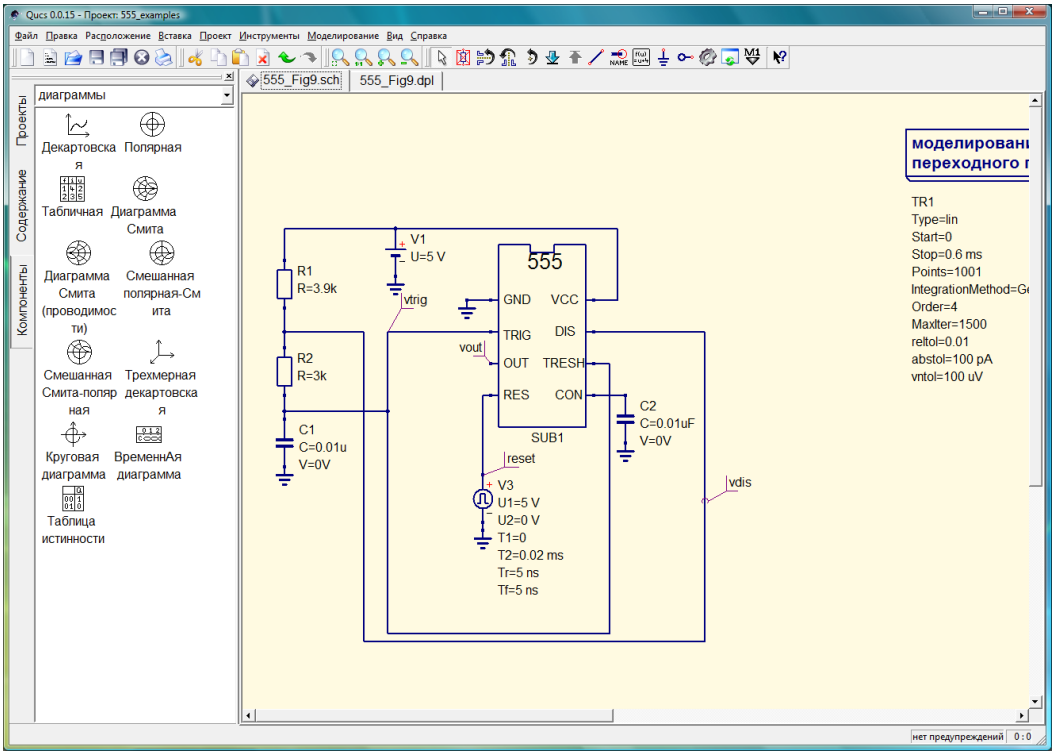


Рис. 2.87. Вид рабочего листа после выполнения команды

Увеличить масштаб (клавиша **+**). После выполнения команды возле курсора появляется значок плюса в круге. Переместив курсор к нужной части чертежа, щелкнув левой клавишей мышки, вы получите увеличенный вид этой области рабочего поля. Команду можно повторять несколько раз.

Чтобы уменьшить изображение, достаточно воспользоваться командой **Уменьшить масштаб** (клавиша **-**). Каждое нажатие на раздел подменю будет уменьшать вид рабочего поля.

Разделы подменю: **Панель инструментов**, **Строка состояния** и **Закрепить окно**, – они относятся к управлению отображения этих панелей в основном окне программы. Сбрасывая флажки (или устанавливая их) можно увеличить рабочее поле. Это особенно удобно для тех, кто привык использовать клавиатуру, а не мышку. Вот, как выглядит окно программы при отключенных панелях.

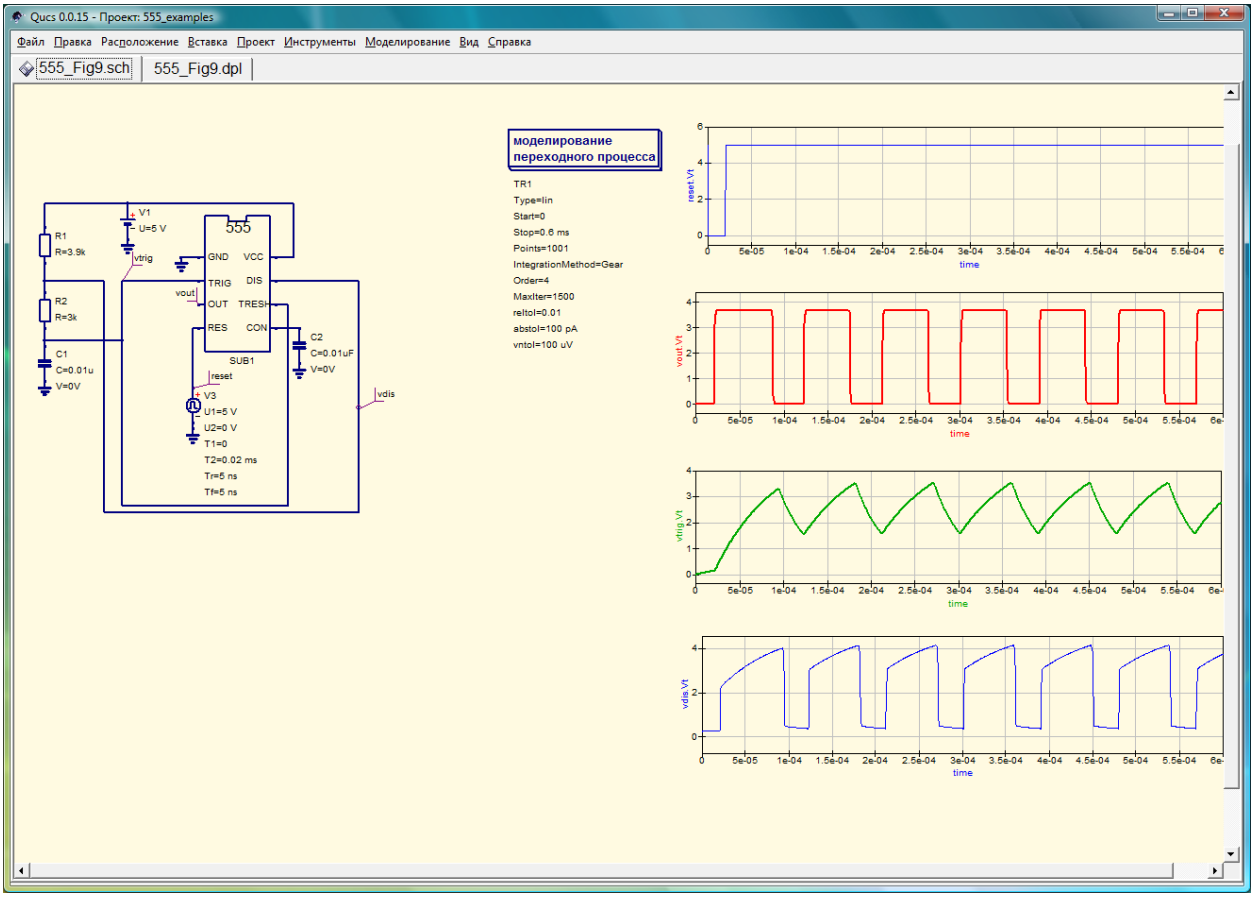
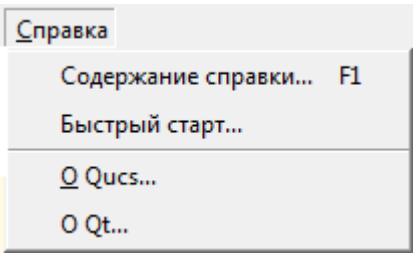


Рис. 2.88. Вид окна программы Qucs при выключенных панелях

Иногда можно закрыть панель навигации по ошибке: самая верхняя кнопка над панелью расположена рядом с кнопкой выбора групп компонентов. Если такая неприятность случается, то достаточно установить флажок, щелкнув левой клавишей мышки, рядом с разделом **Закрепить окно** подменю **Вид**.

Основное меню, Справка



Содержание справки и Быстрый старт дают почти одинаковый результат – открывают весьма хорошо написанное введение в быстрое начало работы с программой.

Рис. 2.89. Подменю пункта Справка основного меню программы

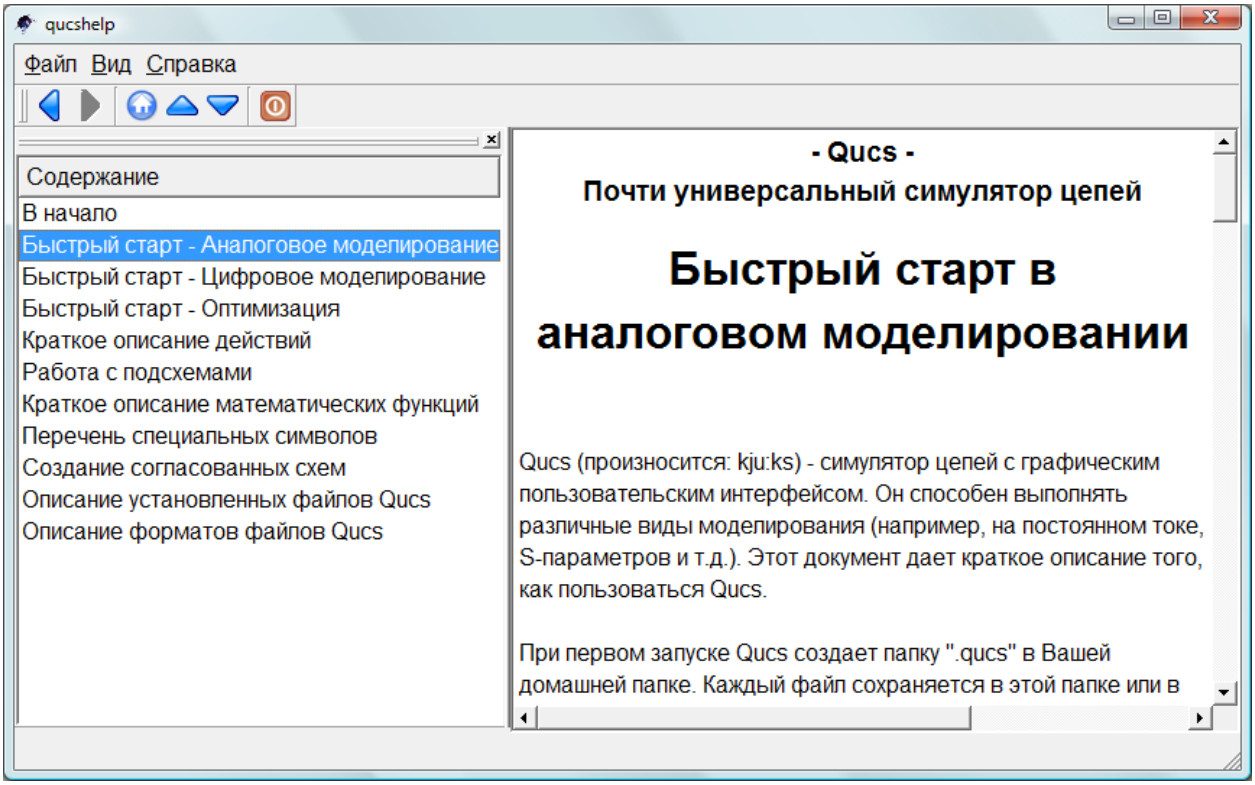


Рис. 2.90. Подпрограмма справки к программе Qucs

Кроме введения в работку с программой, в разделе справки много полезной информации, как это видно из содержания.

Два последних раздела дают краткую информацию о составе группы, работающей над проектом Qucs, и о Qt – средстве разработки многоплатформенных графических интерфейсов, благодаря которому программа Qucs работает и на Windows, и на Linux, и на MacOS.

Глава 2. Дополнительные элементы интерфейса

Инструментальные панели Qucs

Инструментальная панель программы Qucs, в сущности, это несколько панелей, которые можно располагать в произвольном порядке.

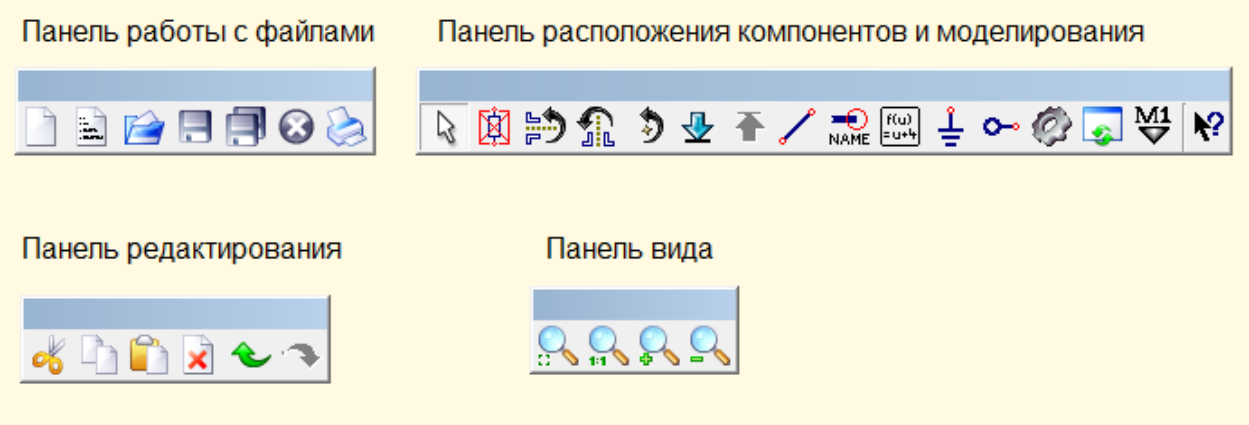


Рис. 3.1. Инструментальные панели в программе Qucs

Инструментальные панели повторяют команды основного меню, но выполнение этих команд требует только щелчка по нужной кнопке на панели.

Панель работы с файлами

Иконка	Назначение	Клавиши
	**Создать** – открывает рабочее поле новой схемы. Она будет «untitled», пока вы не сохраните ее, дав имя.	***Ctrl+N***
	**Новый текст** – открывает текстовый редактор для вашего VHDL файла.	***Ctrl+Shift+V***
	**Открыть** вашу схему или другой файл проекта.	***Ctrl+O***
	**Сохранить** вашу текущую работу.	***Ctrl+S***
	**Сохранить все** для сохранения и схемы, и данных.	***Ctrl++***
	**Закрыть** любой файл, который вам больше не нужен.	***Ctrl+W***
	**Напечатать** – открывает диалоговое окно, где вы можете задать все опции печати.	***Ctrl+P***

Панель редактирования

Иконка	Назначение	Клавиши
	**Вырезать** – вырезать выделенный фрагмент.	***Ctrl+X***
	**Скопировать** – скопировать выделенный фрагмент (или компонент) в буфер обмена.	***Ctrl+C***
	**Вставить** – содержимое буфера обмена в рабочую область чертежа.	***Ctrl+V***
	**Удалить** – выделенный фрагмент.	***Del***
	**Отменить** – предыдущую операцию.	***Ctrl+Z***
	**Вернуть** – предыдущую отменную операцию.	***Ctrl+Y***

Панель вида

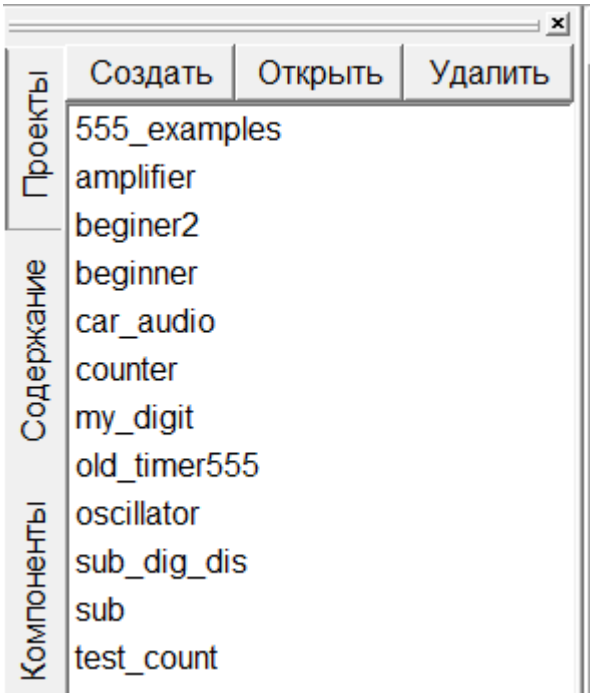
	**Показать все** – адаптирует чертеж так, чтобы все было видно на одной странице.	0
	**Масштаб 1:1** – позволяет вам увидеть чертеж как есть.	1
	**Увеличить масштаб**	+
	**Уменьшить масштаб**	-

Панель расположения компонентов и моделирования

	**Выделить** – режим выделения курсором компонента или фрагмента чертежа.	***Esc***
	**Деактивировать/Активировать** – позволяет вам деактивировать/активировать симуляцию.	***Ctrl+D***
	**Отразить относительно оси X** – отражает выделенное по X.	***Ctrl+J***
	**Отразить относительно оси Y** – отражает выделенное по Y.	***Ctrl+M***
	**Повернуть** – поворачивает выделенное на 90 градусов против часовой стрелки.	***Ctrl+R***
	**Войти в подсхему** – если она есть, вы будете там.	***Ctrl+I***
	**Выйти** – вернуться на уровень выше.	***Ctrl+H***
	**Проводник** – основная операция для создания цепи.	***Ctrl+E***
	**Метка проводника** – позволяет именовать сеть и может быть точкой наблюдения сигнала.	***Ctrl+L***
	**Вставить уравнение** – расширяет ваши возможности при разработке схемы.	***Ctrl+<***
	**Вставить землю** – без земли вы не запустите симуляцию аналоговой электрической цепи.	***Ctrl+G***
	**Вставить вывод** – для подсхемы, например.	
	**Моделировать** – основная функция Qucs, запускает симуляцию.	***F2***
	**Просмотр данных/схемы** – переключает ваши страницы диаграмм и схемы.	***F4***
	**Установить маркер на диаграмме** – очень полезно для получения значения на диаграмме.	***Ctrl+B***
	**Что это?** - подсказка по всем меню и элементам инструментальных панелей. Щелкните и еще раз по любой кнопке панели — вы увидите, что она делает.	

Панель навигации

Левая часть **Основного окна** программы — это **Панель навигации**. Она имеет три закладки, расположенные по вертикали: **Проекты**, **Содержание** и **Компоненты**. Если вы создали хотя бы один проект, и он содержится в основной папке «.qucs», вы увидите этот проект, щелкнув по закладке Проекты.



Каждый проект, который вы создаете с помощью команды **Новый проект**, получит свою папку с названием, который вы задали в диалоговом окне создания проекта.

Чтобы открыть интересующий вас проект, вы можете выделить его щелчком левой клавиши мышки и нажать кнопку ***Открыть***. Впрочем, можете использовать и основное меню или горячие клавиши, но в этих случаях вам придется выбирать проект в файловом менеджере.

Вы можете открыть проект и с помощью двойного щелчка левой клавиши мышки по названию проекта в окне **Проекты**.

На закладке **Проекты** вы можете создать новый проект, используя кнопку ***Создать***.

Вы можете удалить проект, используя кнопку ***Удалить***. Исключение для открытого проекта, последний не может быть удален.

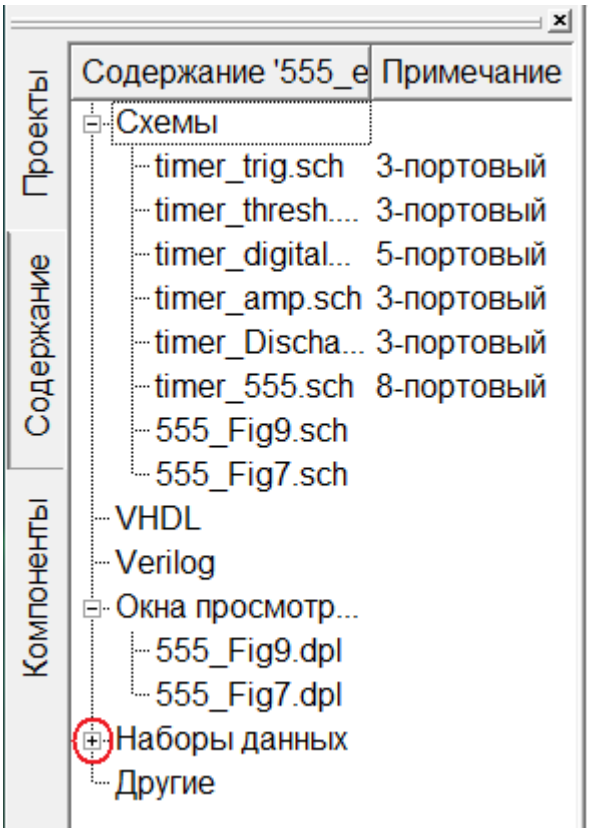
Рис. 3.2. Панель навигации, закладка **Проект**

Иногда при работе чуть-чуть не хватает рабочего поля чертежа. Вы можете скрыть панель навигации, повторно щелкнув по открытой закладке. Следующий щелчок вернет панель навигации на место. При этом скрывается панель навигации, но не закладки. Когда же вы используете пункт **Вид** основного меню, где снимаете флажок у раздела **Закрепить окно**, скрывается и панель навигации, и закладки.

Для того чтобы проект появился в окне проектов, не обязательно создавать новый проект – вы можете скопировать проект в папку «.qucs» с другого компьютера, или скачать проект с сайта разработчиков программы, или распаковать пакет, содержащий проект, используя команду **Распаковать пакет** из подменю **Проект** основного меню.

Если вы открываете проект, скажем двойным щелчком по его имени, активизируется закладка **Содержание** панели навигации, автоматически показывая «дерево проекта», то есть все файлы, входящие в проект: схему, данные, диаграммы и т.д.

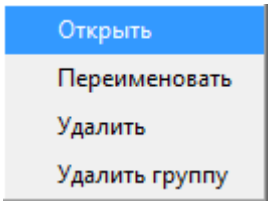
Не все файлы в разделах окна Содержание могут быть показаны при этом переходе. Часть из них скрыта. В этом случае рядом с разделом вы увидите значок «+». Щелкните по нему левой клавишей мышки, и раздел распахнется, показывая все файлы, содержащиеся в нем.



На рисунке отмечен значок, скрывающий содержание раздела *Наборы данных*.

Все схемы проекта, все страницы диаграмм вы можете открыть двойным щелчком левой клавиши мышки по названию файла.

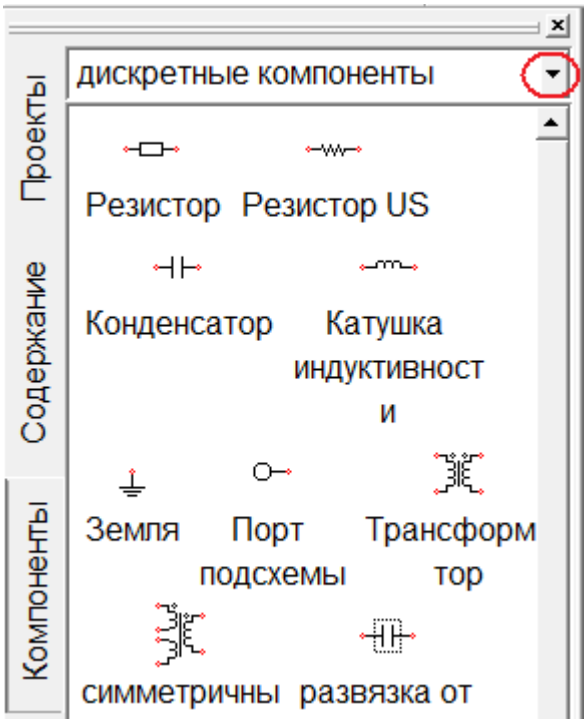
Если вы щелкните правой клавишей мышки по названию файла, то появится выпадающее меню:



которое вы тоже можете использовать для тех операций, которые названы в нем.

Рис. 3.3. Окно закладки Содержание и выпадающее меню этого окна

Пожалуй, самое значимое для создания схемы находится на закладке Компоненты.



На рисунке отмечена кнопка со стрелкой вниз, нажав на которую вы получите список групп компонентов.

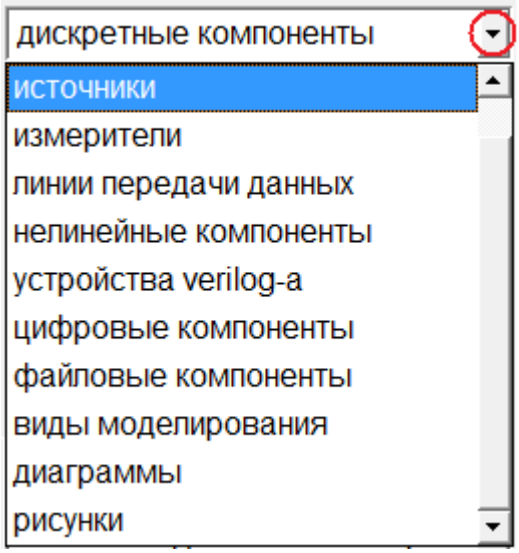


Рис. 3.5. Окно закладки Компоненты и выпадающий список групп компонентов

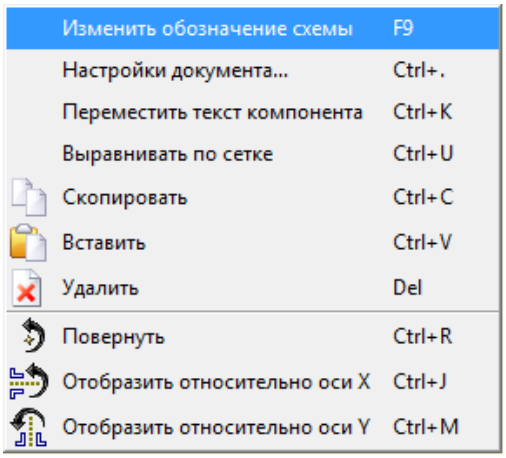
Группы компонентов представлены таким образом, чтобы можно было легко найти нужный для схемы элемент. Но, например, группа дискретных компонентов кроме очевидных, как резистор или конденсатор, пополнена такими компонентами, как гиратор или фазосдвигатель. В программе достаточно четко проведена граница между аналоговым и цифровым моделированием, но в ней возможна работа и смешанных устройств.

Группа **виды моделирования** представляет все возможные в программе виды симуляции. А группа **диаграммы** позволяет отобразить данные всех видов симуляции. Группа рисунки предназначена к созданию графических дополнений любого чертежа: примечаний, таблиц или сложных штампов.

Подробнее все компоненты, доступные в панели навигации, будут рассмотрены позже, когда речь зайдет именно о компонентах электрических схем.

Выпадающие меню

Кроме выпадающего меню панели навигации программа позволяет использовать другие меню, которые появляются после щелчка правой клавишей мышки. Щелкнув по свободному месту схемы, вы получите выпадающее меню общего назначения.

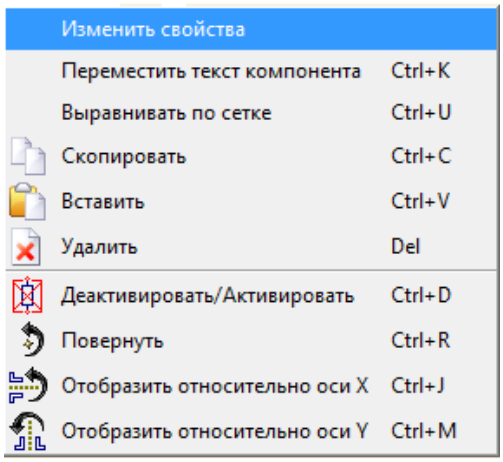


Все команды этого выпадающего меню выбраны из команд основного меню программы, но выбраны те, что наиболее часто будут применяться при вводе схемы.

Команды редактирования, команды поворота и отображения дополнены командами выравнивания и настройки.

Рис. 3.7. Выпадающее меню страницы схемы

Если на странице схемы вы щелкните по компоненту: транзистору или виду моделирования, – вы получите похожее, но отличное от предыдущего выпадающее меню.



Отличие этого выпадающего меню от предыдущего незначительно, но первая команда открывает диалоговое окно свойств компонента, а к командам расположения добавлены команды по активации компонента, что бывает полезно при моделировании работы схемы.

Рис. 3.8. Выпадающее меню компонента

Щелчок правой клавиши мышки по графику, отображающему результаты моделирования, открывает следующее выпадающее меню.

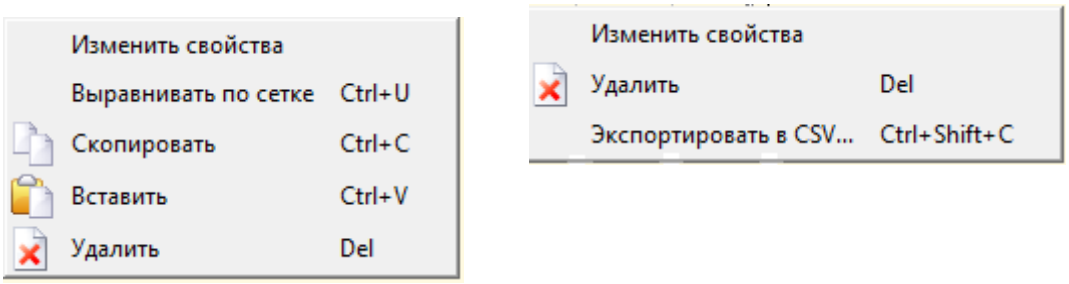
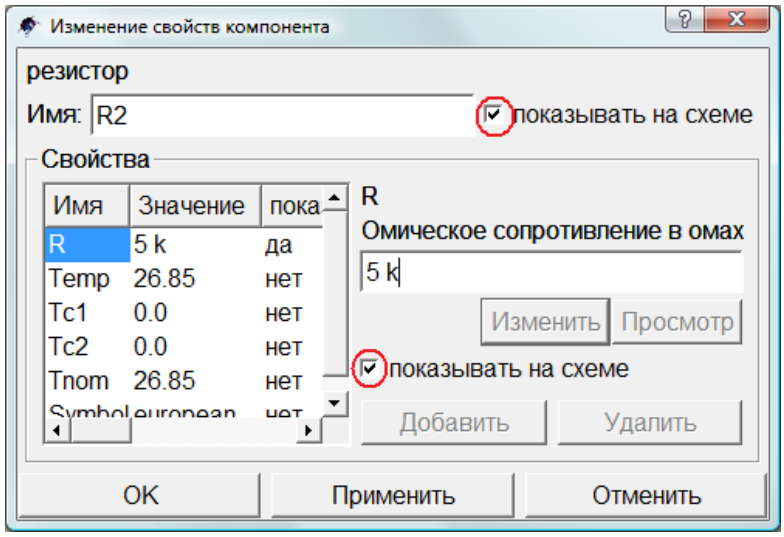


Рис. 3.9. Выпадающие меню отображения данных и кривой графика

Диалоговые окна

Кроме системных диалоговых окон таких команд, как **Открыть** или **Сохранить как**, каждый компонент имеет собственное диалоговое окно свойств.



Все резисторы появляются со значением сопротивления, заданного по умолчанию, введя значение в текстовое поле ***Омическое сопротивление в омах***, вы получите нужную величину сопротивления.

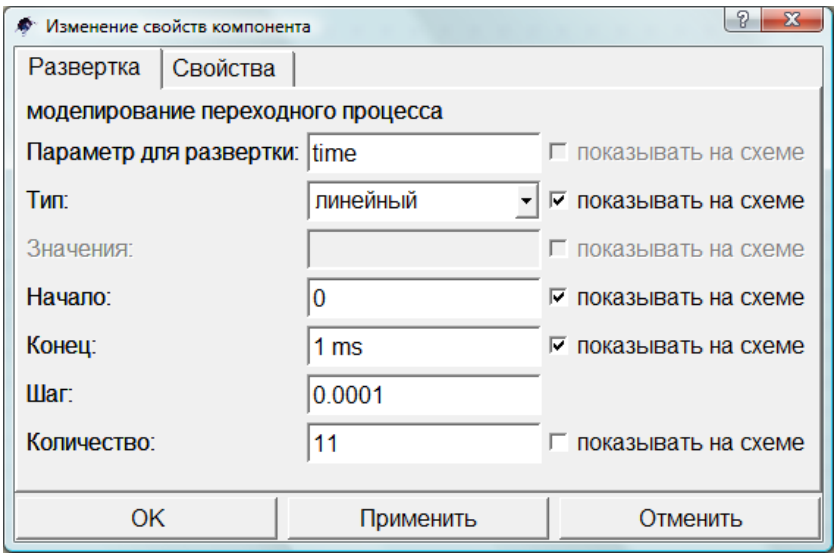
Рис. 3.11. Диалоговое окно свойств резистора

Практически все диалоговые окна свойств имеют опцию **показывать на схеме**. Устанавливая или снимая флажок (на рисунке отмечен) вы регулируете количество параметров, необходимых для показа на чертеже. Разные компоненты имеют разное количество параметров. Вы можете менять некоторые из них или все.

> Если вы установили опцию **показывать на схеме**, вы можете редактировать свойства непосредственно в рабочем поле чертежа. Просто выделите нужное, как обычный текст, введите новое значение и нажмите клавишу ***Enter***.

Изменяя параметр, следует нажать кнопку ***Применить***. Иначе параметр может вернуться к первоначальному значению, что особенно важно, когда вы меняете несколько параметров в диалоговом окне свойств. Если вы закончили задание нужных параметров, нажмите кнопку ***ОК***. Если передумали менять параметры, нажмите кнопку ***Отменить***.

Разные компоненты имеют разные диалоговые окна свойств. И не следует забывать, что диаграммы и виды моделирования тоже компоненты.



Это окно имеет две закладки.

От свойств этого компонента зависит то, как будут отображаться результаты на графике.

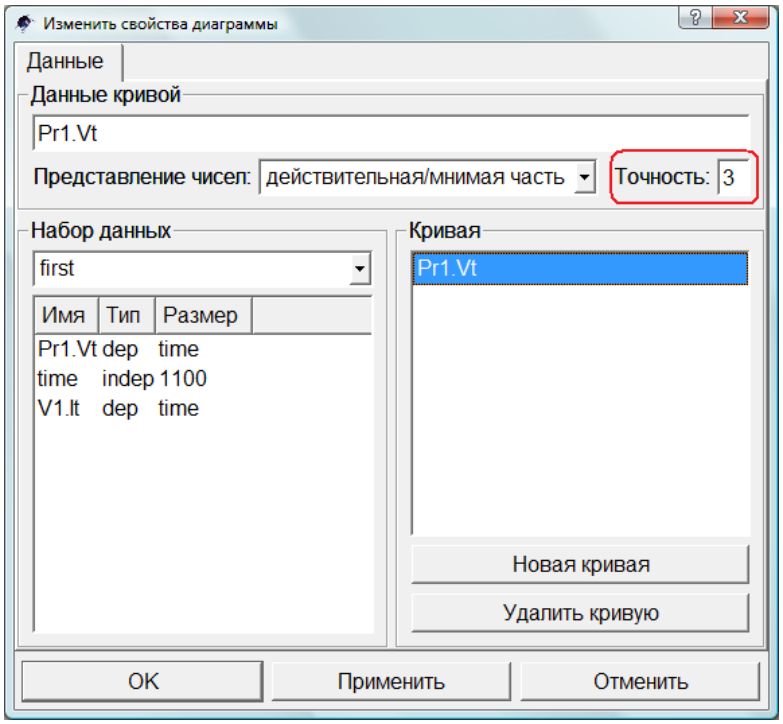
Рис. 3.12. Диалоговое окно свойств моделирования переходного процесса

Числа и имена в программе Qucs

В диалоговых окнах свойств компонентов вы, как правило, вводите значения. Основные числа имеют вид 10 или 200, но вы можете ввести и 0.1 или 0.34.

Вам нужно использовать «k» для килоом и килоампер, «M» для мегаом и «m» для миллиампер. Все в порядке, добавляете вы или нет пробел: «5k» и «5 k» одно и то же.

Если вы используете пробник, например, **Измеритель напряжения**, и хотите увидеть результат в виде таблицы, вы можете выбрать нужную точность в диалоговом окне отображения данных.



time	Pr1.Vt
0	0
1.82e-06	0.0572
3.64e-06	0.114
5.46e-06	0.171
7.28e-06	0.229
9.1e-06	0.286
1.09e-05	0.343

Рис. 3.13. Диалоговое окно свойств таблицы данных и полученные данные

В программе Qucs используются следующие суффиксы в инженерном формате значений:

- dBm – $10 \cdot \log (x/0.001)$
- dB – $10 \cdot \log (x)$
- T – 10^{12}
- G – 10^{9}
- M – 10^{6}
- k – 10^{3}
- m – 10^{-3}
- u – 10^{-6}
- n – 10^{-9}
- p – 10^{-12}
- f – 10^{-15}
- a – 10^{-18}

Допустимы значения в свойствах компонентов: в стандартной (1000), научной (1e-3) или инженерной (1k) нотации по выбору. Допустимы также единицы:

- Ohm – resistance / Ω
- s – time / Seconds
- S – conductance / Siemens
- K – temperature / Kelvin
- H – inductance / Henry
- F – capacitance / Farad
- Hz – frequency / Hertz
- V – voltage / Volt
- A – current / Ampere
- W – power / Watt
- m – length / Meter (не используются отдельно)

Для получения графика, показывающего сигнал в интересующей вас точке, вы можете использовать измеритель, но можете вставить **Метку проводника** (**Основное меню,** кнопка на **Панели моделирования** или ***Ctrl+L***).

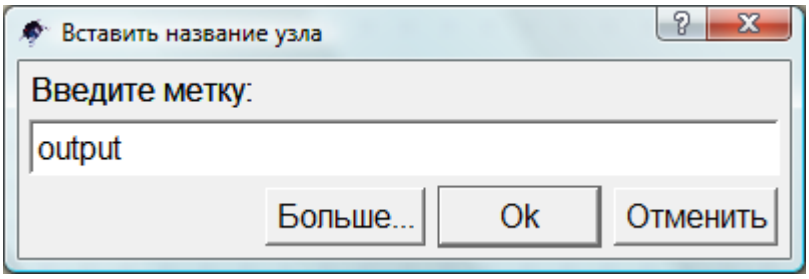


Рис. 3.15. Диалог ввода метки проводника

Все имена лучше писать латиницей.

Формулы

Возможность добавить к схеме уравнение, используя пункт **Вставка** основного меню, кнопку инструментальной панели или клавиши ***Ctrl+<,*** расширяет ваши возможности по моделированию схем. В формулах можно использовать все доступные функции программы и их сочетания.

Самым простым примером использования элемента программы **Уравнение** будет, пожалуй, амплитудно-частотная характеристика усилителя, которую удобнее отображать в относительных

единицах, децибелах. Для получения амплитудно-частотной характеристики к схеме усилителя добавляется **Моделирование на переменном токе**. Но полученная характеристика отображается в абсолютных единицах. Чтобы изменить вид диаграммы, достаточно вставить в схему уравнение.

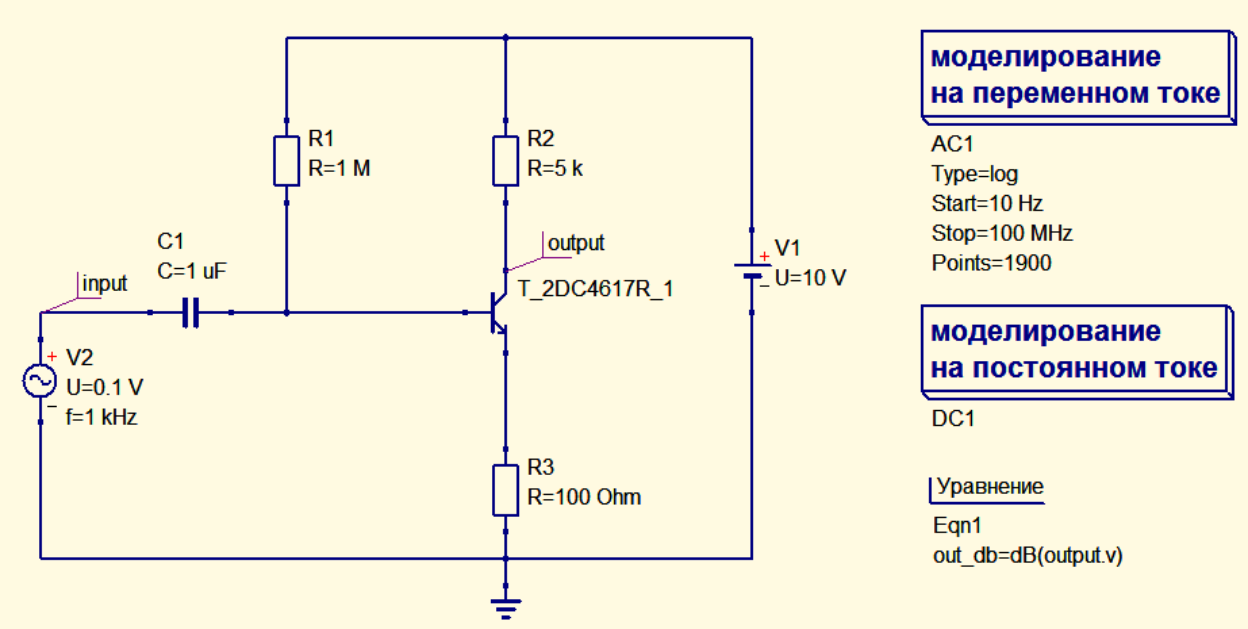


Рис. 3.16. Добавление уравнения к схеме

При добавлении элемента Уравнение в появившемся диалоговом окне вводится обозначение функции и формула, по которой будет рассчитываться результат.

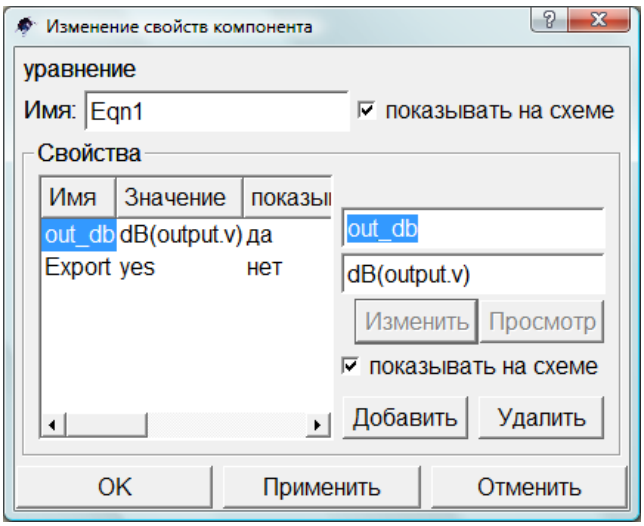


Рис. 3.17. Диалоговое окно уравнения

В данном случае полученная кривая, определяемая меткой на схеме *output*, будет определяться как ***out_db***, по формуле, где используется функция программы *dB()*. В качестве аргумента функции следует применять то написание метки, которое появляется в диалоговом окне моделирования на переменном токе, а именно: ***output.v***.

Полученная амплитудно-частотная характеристика выглядит более привычно.

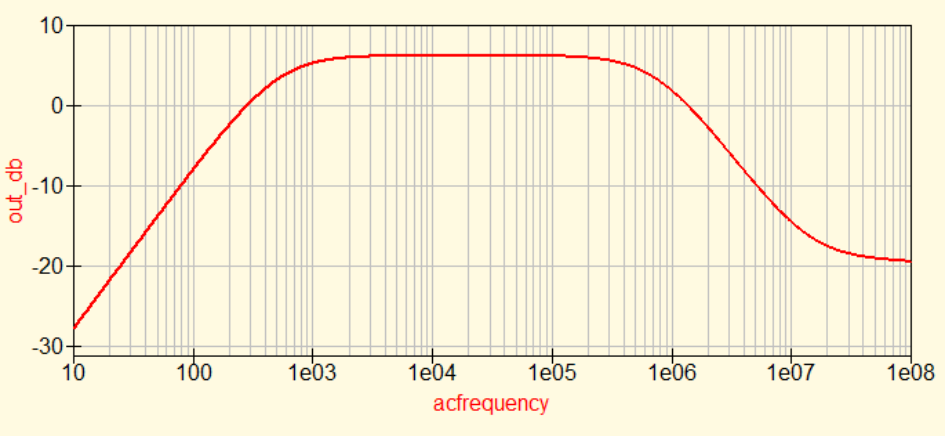
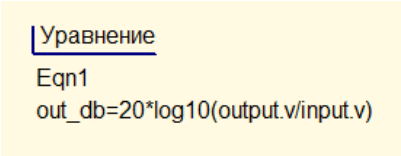


Рис. 3.18. Амплитудно-частотная характеристика каскада в децибелах

Если использовать отношение напряжения выходного сигнала к входному в децибелах, то формулу можно написать и в таком виде, что приведет к виду АЧХ:



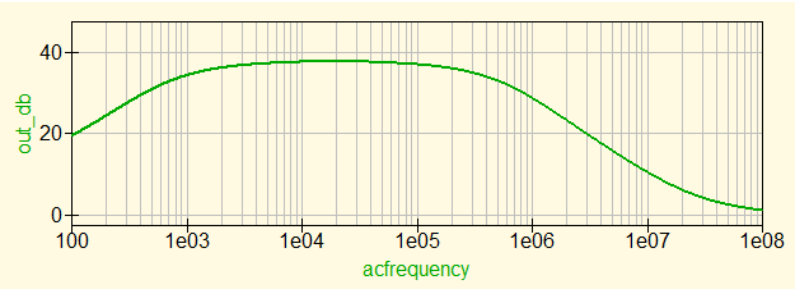
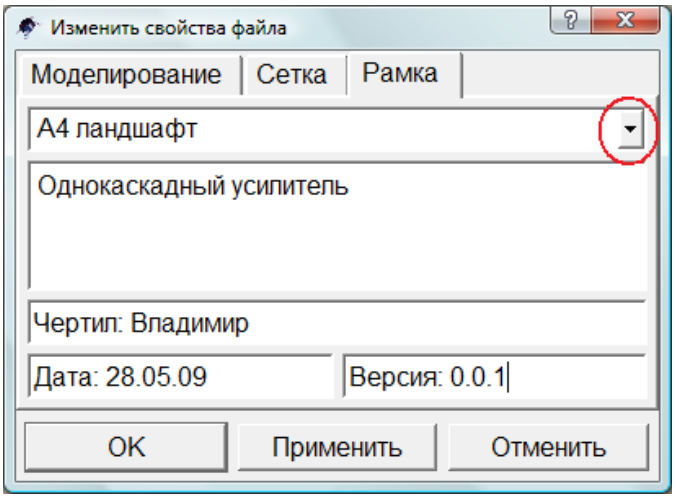


Рис. 3.20. Вид АЧХ при другом уравнении

В схему можно добавить несколько уравнений, а можно записать несколько формул в одном уравнении.

Документ

Если вернуться к пункту **Файл** основного меню, в подменю которого есть раздел **Настройки документа**, то в диалоговом окне этой настройки есть закладка **Рамка**.



Кнопка, отмеченная на рисунке, выводит список форматов от A3 до A5, включая формат конверта. Вы можете выбрать подходящий для ваших целей.

Окно ниже для ввода названия чертежа. Все остальные окна ввода обозначены и очевидны. Заполнив эти параметры, вы получите обрамление вашего чертежа и штамп.

Рис. 3.21. Диалоговое окно настроек документа

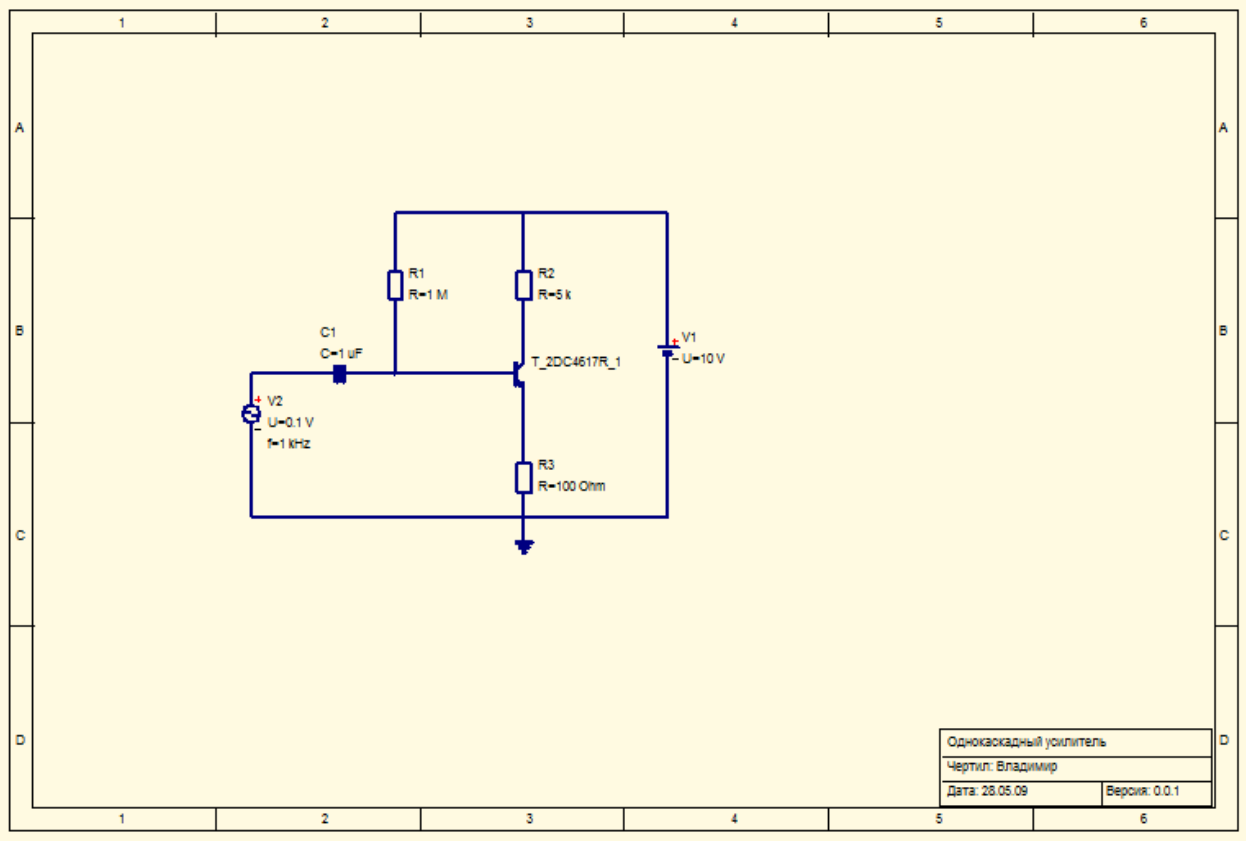


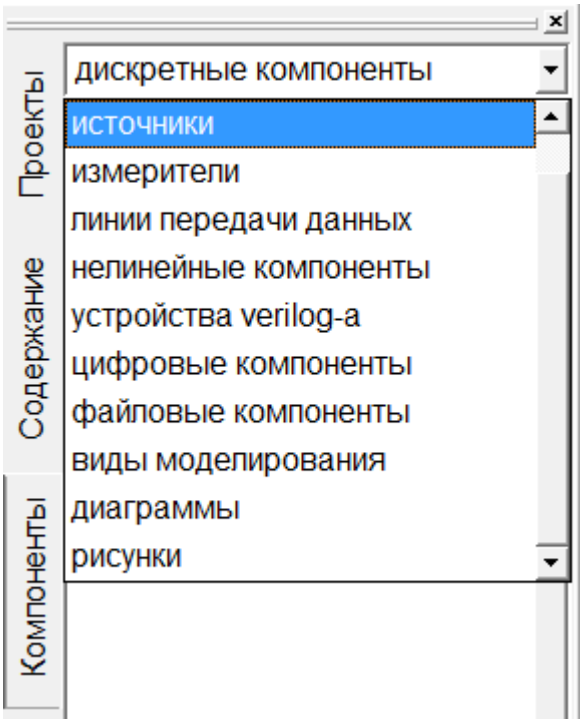
Рис. 3.22. Оформление чертежа с помощью настройки документа

Глава 3. Ввод схемы

Компоненты

Вы установили программу Qucs, создали новый проект. Что дальше?

Первое, что следует сделать, нарисовать схему. Для этого вам понадобится содержание закладки Компоненты панели навигации. Все компоненты электрических схем в программе разбиты на группы.



Каждая из групп компонентов составлена из элементов, подходящих к названию группы.

Основа группы **дискретные компоненты** – это такие элементы, как резисторы и конденсаторы. Группа **нелинейные компоненты** состоит в основном из транзисторов.

В состав группы **цифровых компонентов** включены все логические вентили и базовые цифровые устройства.

Особые группы компонентов представлены группой **виды моделирования**, **диаграммы** и **рисунки**.

Рис. 4.1. Группы компонентов в программе Qucs

Практически все компоненты перемещаются в рабочее поле одним приемом – щелкнув по нужному компоненту левой клавишей мышки, вы перемещаете курсор в нужное место, где повторный щелчок вставляет компонент. При перемещении контур компонента «привязан» к курсору.

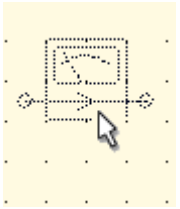


Рис. 4.2. Перемещение компонента в рабочее поле чертежа

При перемещении компонента из окна панели навигации вы можете щелчком правой клавиши мышки поворачивать объект до его вставки в схему. Вставив компонент, вы можете продолжать размещение этого типа компонента, поскольку, как правило, такие элементы схемы, как резисторы или конденсаторы, требуются в схеме не в единственном экземпляре. При такой последовательной расстановке порядковое обозначение элемента в схеме автоматически увеличивается. Чтобы выйти из режима добавления компонентов в схему, достаточно нажать на

клавиатуре клавишу ***Esc*** или щелкнуть мышкой по кнопке с изображением курсора на инструментальной панели для перехода в режим **Выделения**.

После вставки компонента в чертеж вы можете его выделить, щелкнув по нему левой клавишей мышки.

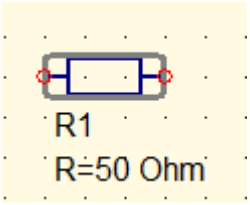
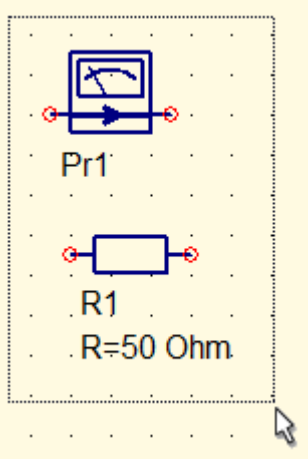


Рис. 4.3. Выделение компонента

После выделения компонента его можно перемещать либо «подцепив» мышкой, либо используя курсорные клавиши клавиатуры. После выделения компонент можно удалить, нажав клавишу ***Delete*** на клавиатуре или кнопку удаления на инструментальной панели. После выделения компонент можно размещать, используя подменю пунктов **Правка** и **Размещение**, то есть, поворачивать, отражать, привязывать к сетке и т.д. Можно щелчком правой клавиши мышки вызвать появление выпадающего меню, где есть все основные команды работы с компонентами.

Двойной щелчок левой клавишей мышки по компоненту открывает диалоговое окно его свойств, где вы можете, например, изменить величину сопротивления или емкости конденсатора.

Чтобы выделить не одиночный компонент, а группу компонентов, вы можете поместить курсор на свободном месте рабочего поля над нужной группой компонентов и, нажав и удержав левую клавишу мышки, перемещать курсор по диагонали вниз, чтобы обрисовать группу прямоугольником выделения.



Когда вы отпускаете клавишу мышки, все охваченные прямоугольником компоненты будут выделены.

Но иногда вам нужны не все выделенные в группе компоненты. Вы можете, удерживая клавишу ***Ctrl*** на клавиатуре, щелкнуть левой клавишей мышки по компоненту, чтобы снять выделение. Повторный щелчок (при удержанной клавише ***Ctrl***) вновь выделит компонент.

Рис. 4.4. Выделение группы компонентов

Выделенную группу компонентов можно перемещать, можно скопировать, используя основное меню или инструментальную панель, чтобы вставить на другую страницу схемы. При этом постарайтесь не забывать о таких компонентах, как виды моделирования, если, конечно, они вам нужны в новой схеме. Если вы передумали что-то делать с выделенной группой, достаточно щелкнуть левой клавишей мышки на свободном месте рабочего поля, чтобы снять выделение.

Начиная работу со схемой, вы представляете, какого рода схему вы собираетесь рисовать: аналоговую или цифровую. Иногда вы достаточно ясно представляете всю схему. В этом случае вы

можете располагать компоненты на тех местах, где они останутся впоследствии. Но, если схема вам не ясна, возможно, есть смысл «набросать» нужное количество компонентов в одном месте рабочего поля, чтобы постепенно «перетащить» их в нужные места.

Иногда позиционирование компонентов происходит тогда, когда вы соединили все элементы схемы проводниками. Не забывайте о проводниках.

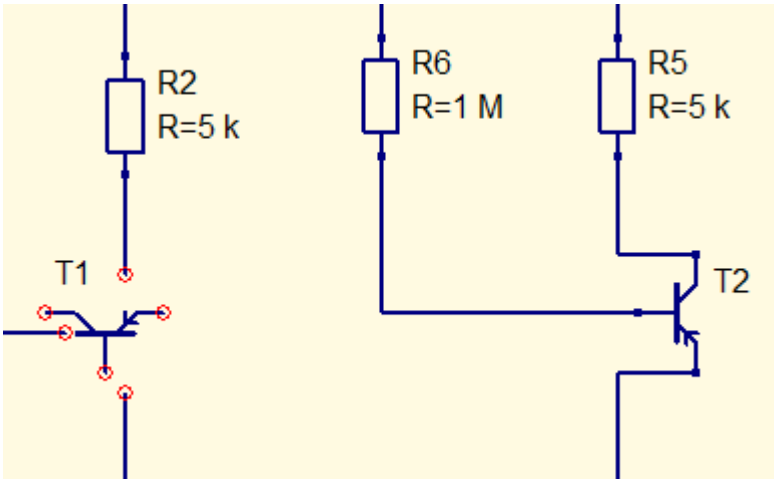
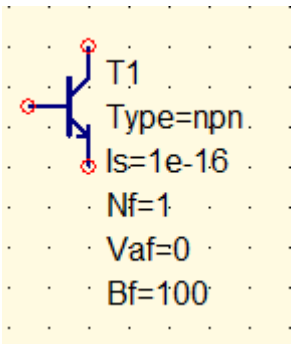


Рис. 4.5. Позиционирование компонентов схемы после соединения

Когда вам нужно вставить, скажем, резистор в цепь, где соединение проведено, вы можете сделать это, поместив резистор на проводник – соединение будет сделано автоматически. Аналогично, если вы решили заменить транзистор общего назначения конкретной моделью, вы можете выделить транзистор, удалить его, выбрать в **Библиотеке компонентов** нужный вам транзистор, перетащить его на свободное место рабочего поля из библиотеки, а затем установить на место удаленного – соединение будет восстановлено автоматически.

Видимая часть рабочей области редактора схемы — это не вся доступная вам рабочая область. Вы можете использовать полосы прокрутки или расширить ее с помощью клавиш клавиатуры влево/вправо и вверх/вниз (когда компоненты не выделены). Можно использовать колесико мышки для перемещений вверх/вниз, а при удержанной клавише ***Shift*** для перемещений влево/вправо; при удержанной клавише ***Ctrl*** вы можете увеличить и уменьшить чертеж.

Когда вы поместили компонент на чертеж, он имеет ряд параметров видимых по умолчанию. Транзистор или диод, например, имеют их довольно много.

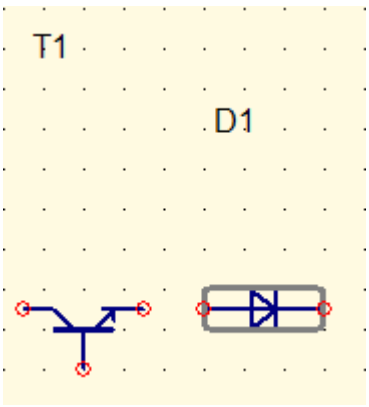


Далеко не все отображаемые параметры вам нужны при работе. Чтобы не загромождать чертеж, вы можете отключить отображение лишних параметров в диалоговом окне свойств компонента. Достаточно снять флажок ***показывать на схеме***, когда вы выделили параметр. После этого нажмите кнопку **Применить**.

Рис. 4.6. Отображаемые по умолчанию параметры транзистора

При написании статьи, как правило, вам не нужно ничего, кроме обозначений компонентов. Но полезно добавлять пояснения или таблицы, используя группу компонентов **рисунки**. Впрочем, и тогда, когда вы работаете над созданием схемы, использование пояснительных надписей и таблиц может помочь вам впоследствии, особенно в том случае, когда вы не добавляете в проект пояснительных записок или плана работы, то есть, не ведете рабочую тетрадь. Обычно это случается при работе над небольшими проектами; таким образом, пояснительные надписи на чертеже не покажутся вам лишними по прошествии некоторого времени.

Есть одна деталь, связанная со скрыванием параметров. Если при установке компонента вы его повернули, а позже решили скрыть ненужные параметры, то полученный результат может вам не понравиться.



Есть два способа исправить это: снимать флажки отображения параметров на схеме до поворота компонента и использовать команду **Переместить текст компонента** из пункта **Расположение** основного меню (клавиши ***Ctrl+K)***. После ввода команды возле курсора появится контур прямоугольника. Переместите курсор к надписи параметра, нажмите и удержите левую клавишу мышки и перетащите текст в новое место.

Рис. 4.7. Отображение обозначения после скрывания параметров повернутого компонента

При работе с аналоговыми схемами не забывайте добавлять землю, если хотите увидеть осциллограммы сигналов. И не забывайте добавить источники в любых схемах, если намерены моделировать работу схемы.

Иногда, когда вы настраиваете схему, вы можете для удобства перенести диаграмму со страницы данных на схему. Просто выделите ее, скопируйте и вставьте на страницу схемы. Вы можете видеть и схему, и результат симулирования.

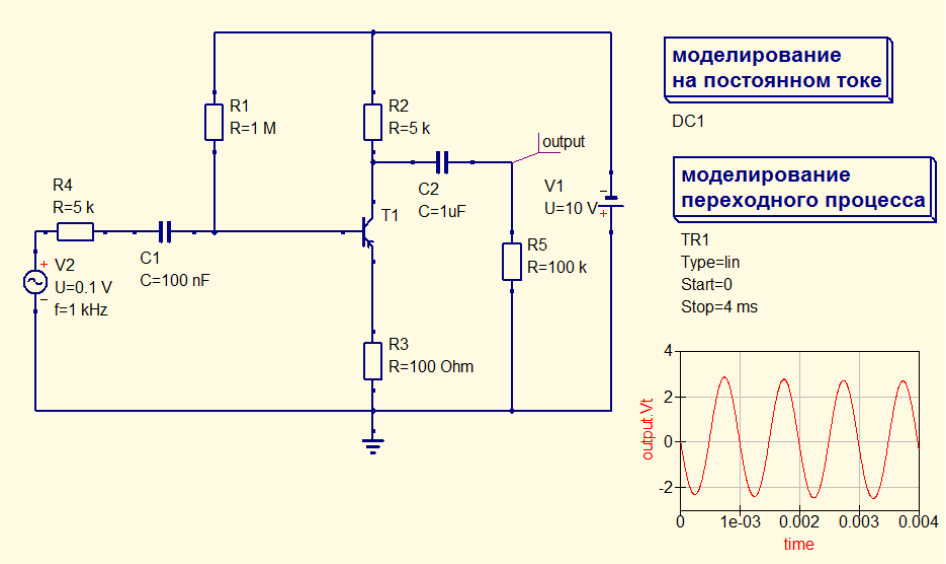
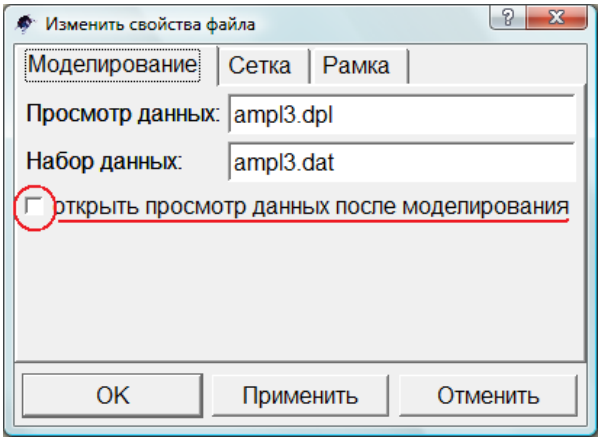


Рис. 4.8. Совмещение схемы и диаграммы

Чтобы видеть только страницу схемы после моделирования, вы можете очистить флажок ***открыть просмотр данных после моделирования*** (**Файл**, **Настройки документа**).



Чтобы снять флажок, щелкните по нему левой клавишей мышки, затем по кнопкам ***Применить*** и ***ОК***.

Рис. 4.9. Настройки документа

При работе с компонентами, когда вы рисуете схему, вам понадобятся такие операции, как перемещение, повороты, отражения, копирование и т.д.

Посмотрим, как можно было бы нарисовать схему, изображенную выше. Нам понадобится пять резисторов, которые мы должны перетащить из окна **Компоненты** панели навигации (нажав закладку Компоненты на ней) в рабочее поле чертежа. Чтобы не повторять пять раз одну операцию, мы можем подцепить резистор, и пять раз щелкнуть в рабочем поле, получив пять резисторов, обозначения которых пронумерованы автоматически.

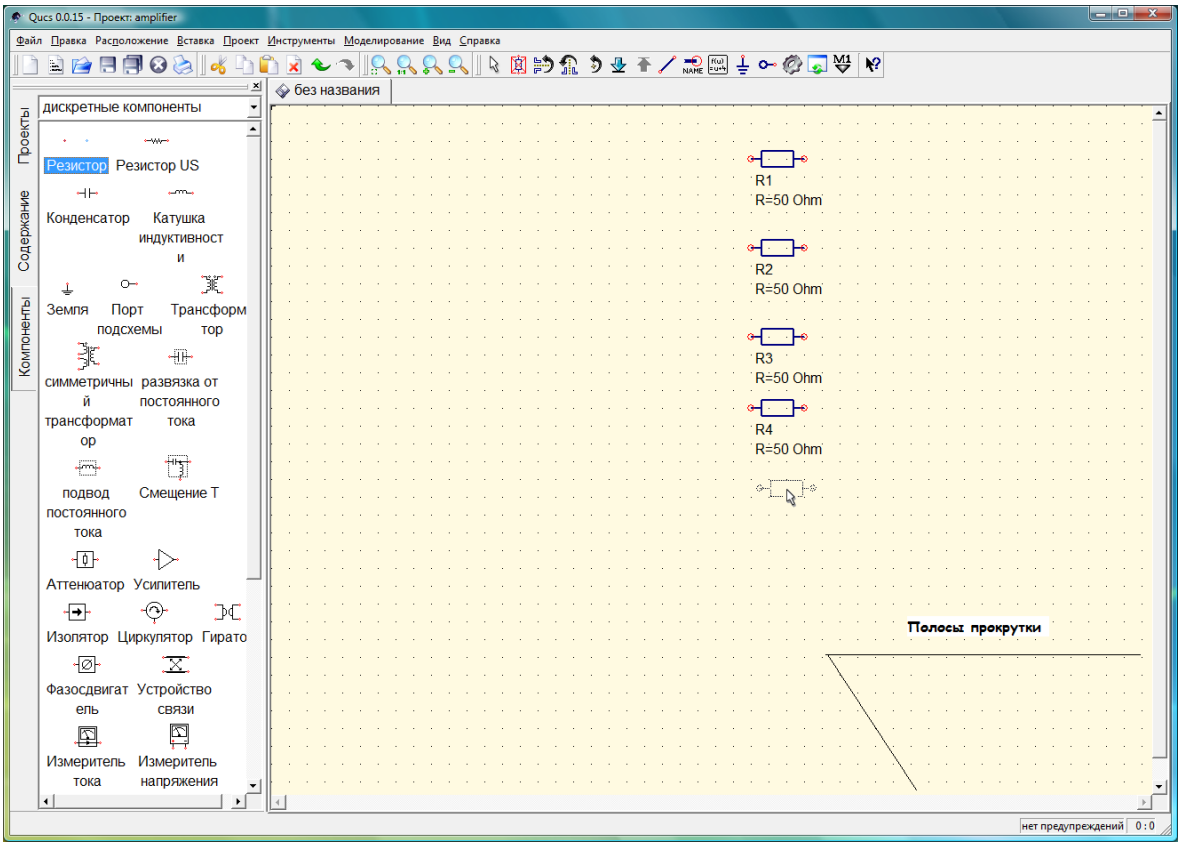
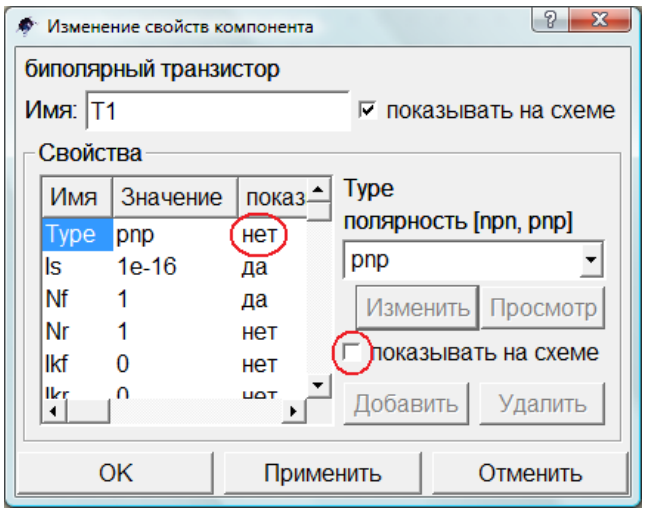


Рис. 4.10. Набор компонентов для схемы

После того, как все компоненты оказались в рабочем поле чертежа, можно расставить их по местам. Но предварительно следует скрыть все ненужные параметры, а значения заменить нужными. Еще раз вспомним, как это делается. Двойной щелчок по компоненту левой клавишей мышки открывает диалоговое окно его свойств.



Выделите в левом окне параметр, который вам нужно изменить или скрыть. Если нужно скрыть параметр, снимите флажок с ***опции показывать на схеме***. В поле *показывать* значение «да» сменится на «нет». Нажмите кнопку ***Применить***. Используйте прокрутку, чтобы переместиться к параметрам, которые не видны в окне.

Если вам нужно изменить значение параметра, введите новое значение в текстовом поле значения компонента справа.

Рис. 4.11. Диалоговое окно свойств транзистора

Применение транзистора типа p-n-p требует изменения полярности источника постоянного тока. Чтобы это сделать, можно дважды повернуть его, например, используя выпадающее меню – щелчок правой клавишей мышки после выделения источника постоянного тока – где есть команда **Повернуть**. Но разумнее в том же меню выбрать команду **Отобразить относительно оси X**.

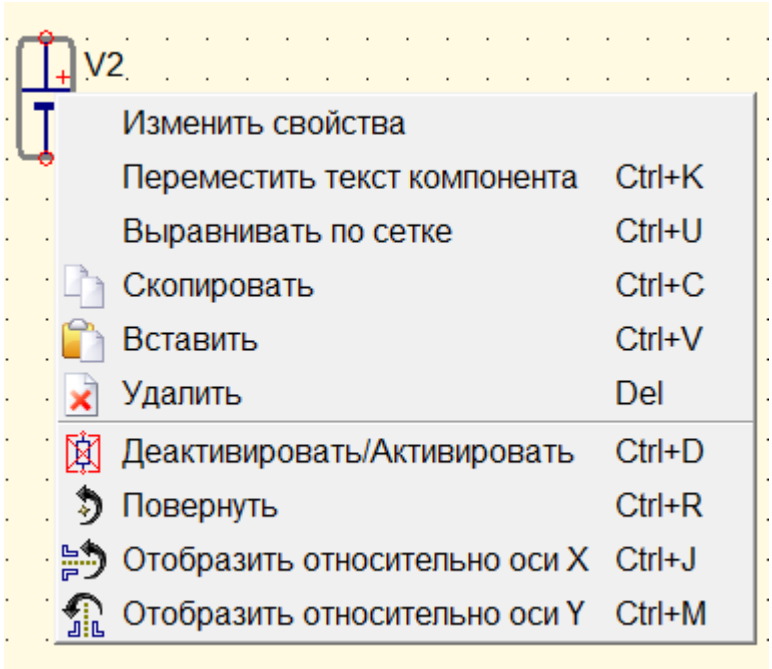
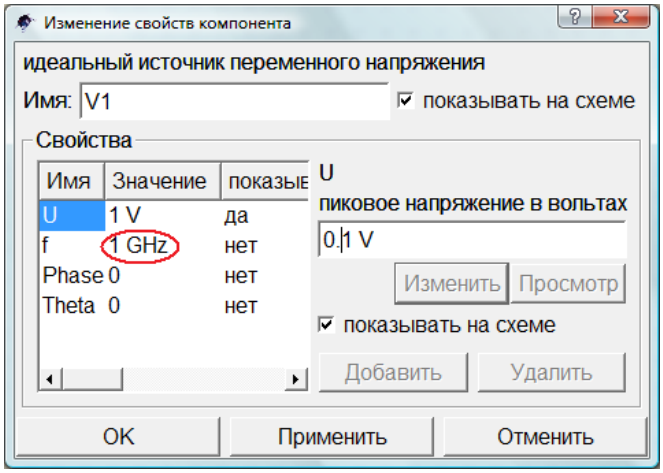


Рис. 4.12. Выпадающее меню работы с компонентом

Источник напряжения переменного тока, используемый как генератор, по умолчанию имеет частоту 1 GHz.

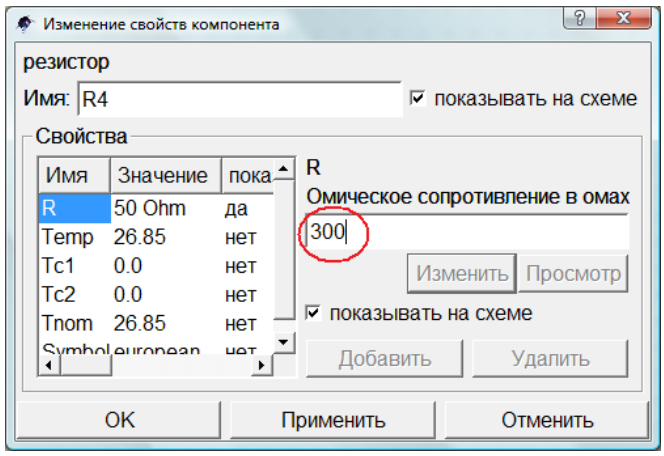


В спешке можно забыть и не изменить этот параметр. Но при попытке моделировать схему, вы можете долго ждать результата, который вам, вдобавок, и не понравится.

Рис. 4.13. Диалоговое окно свойств источника переменного напряжения

Расстановка компонентов по местам операция достаточно простая – подцепить мышкой и перенести в нужное место. Резисторы, кроме одного, должны быть повернуты. Их можно выделить все, и одной операцией поворота придать нужное положение. При этом следует пользоваться не выпадающим меню, а основным, или кнопкой инструментальной панели, или клавишами ***Ctrl+J***. Иначе, при щелчке правой клавишей мышки, выделение группы компонентов будет сброшено.

Чтобы задать нужное значение резистора, достаточно дважды щелкнуть по нему (или щелчком правой клавиши мышки вызвать выпадающее меню, где выбрать пункт **Изменить свойства**, или выделить значение компонента на схеме, как текст, и вписать новое значение, нажав на клавишу ***Enter***), открывая диалоговое окно свойств.



Новое значение вводится в текстовом поле, после чего следует нажать кнопку ***Применить***.

Часто в схеме оказывается несколько резисторов одного номинала. В этом случае удобнее использовать один компонент, значение которого изменяется должным образом, а сам компонент копируется и вставляется нужное число раз из буфера обмена.

Рис. 4.14. Диалоговое окно свойств резистора

После размещения всех компонентов схемы по местам их «дислокации» и задания их значений требуется соединить отдельные элементы в электрические цепи. О том, как это сделать, было сказано ранее, но сейчас отметим, что в свойствах программы есть одна опция, которую многие сочтут полезной – начинать соединение, когда курсор мышки находится над открытым, то есть, не соединенным с другими узлом.

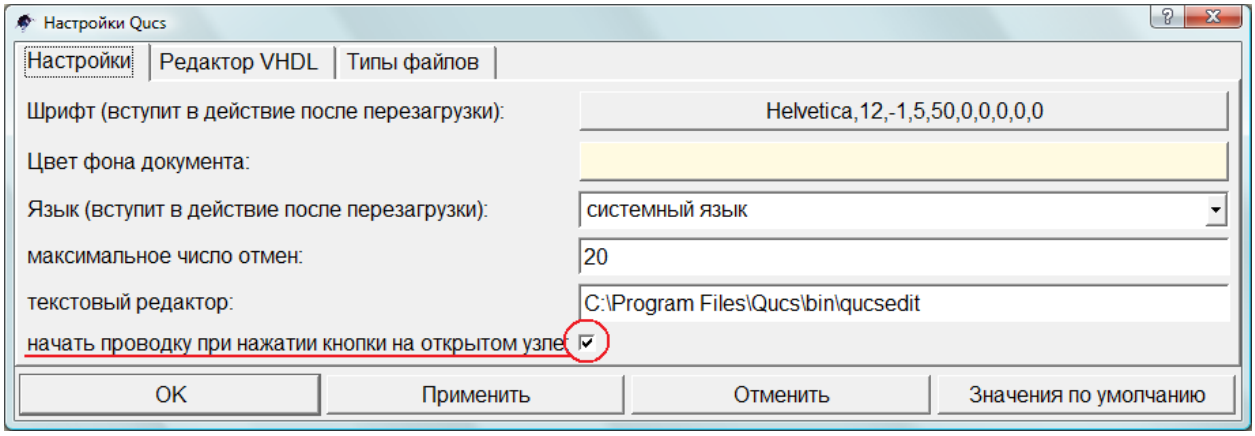
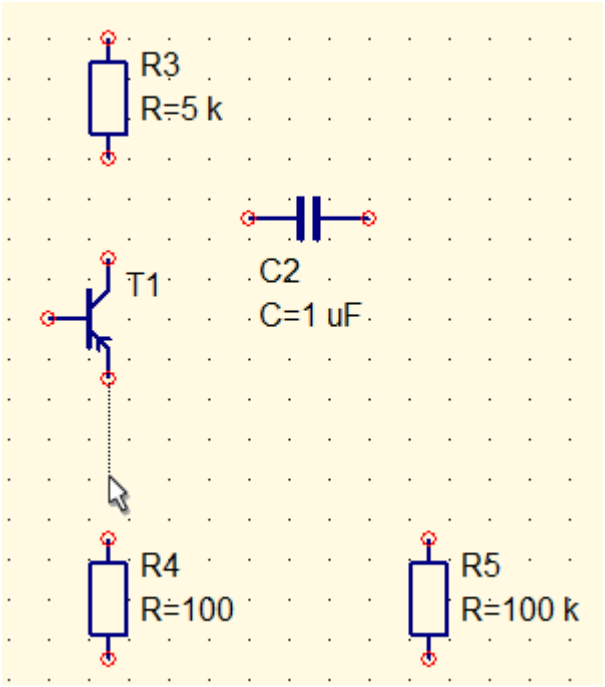


Рис. 4.15. Настройка программы для ускорения процесса соединений

Применение этой опции позволяет начинать соединение без выполнения команды **Проводник**. Когда курсор находится над открытым узлом компонента достаточно щелчка левой клавишей мышки, чтобы соединение началось. А для его завершения требуется второй щелчок левой клавишей на открытом узле другого компонента или на уже существующем соединении.



При том большом количестве соединений, что обычно приходится выполнять, работа ускоряется значительно.

Но в любой момент вы можете отключить эту опцию, если она вам стала мешать.

Соединение вы может оставить открытым, например, для того, чтобы позже добавить порт или вставить метку проводника.

Рис. 4.16. Проведение соединения с установленной опцией в настройках программы

Когда все соединения сделаны, вам остается добавить требуемый вид моделирования, чтобы начать работу со схемой. Но порой, особенно в спешке, в том состоянии нетерпения, когда вашими действиями руководят эмоции, вы и сами можете заблудиться в создаваемых версиях схемы и множестве идей, возникающих у вас «на ходу».

Дело в том, что вы можете запустить программу Qucs несколько раз, создавая несколько вариантов схемы или организуя иерархию разработки. Полезно использовать компоненты группы **рисунки** из окна закладки **Компоненты** панели навигации, чтобы оставить заметки, пояснения или таблицы с полученными данными.

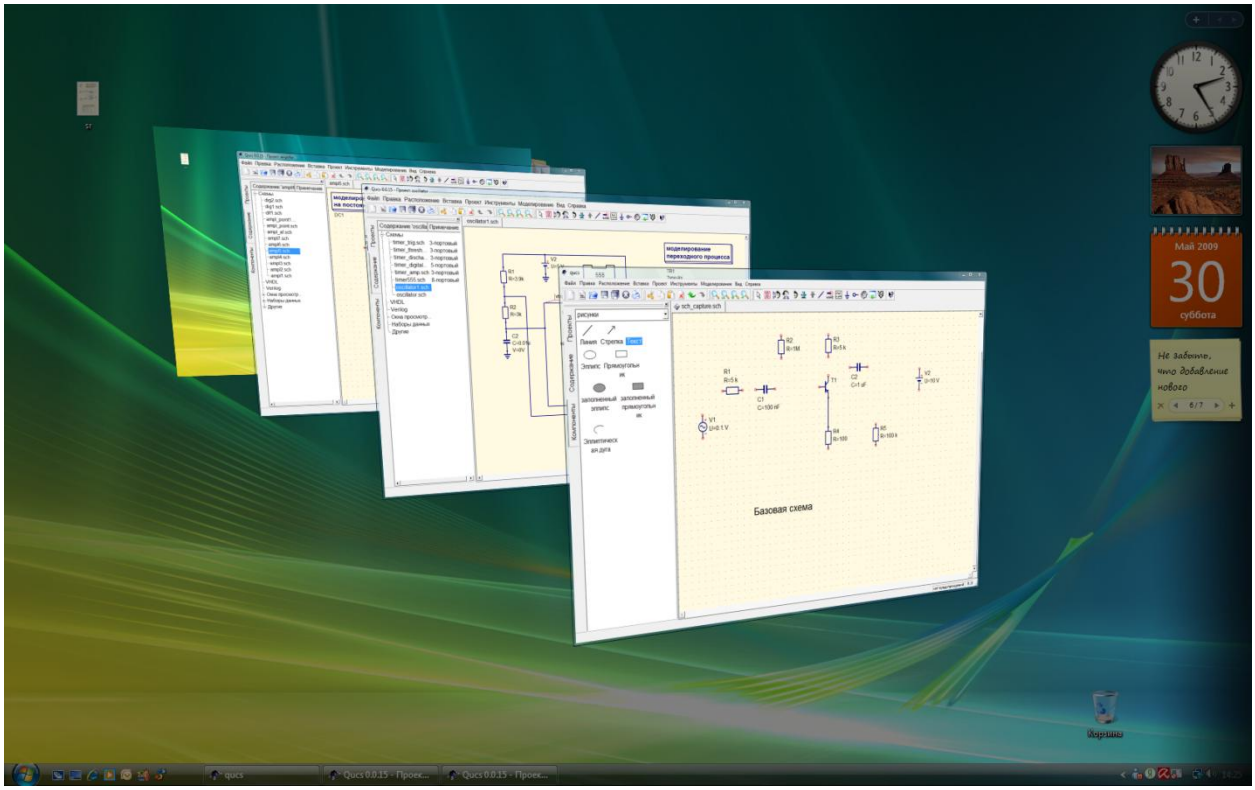


Рис. 4.17. Несколько копий программы одновременно

Позже вы оформите с помощью этих компонентов чертеж окончательного варианта схемы, используя встроенный штамп или создав свой, но по ходу работы вы можете оставлять пометки на полях, которые не помешают работе симулятора.

Окончательный или промежуточный вариант схемы вы можете распечатать на принтере, используя команды **Напечатать** или **Печать по размеру страницы** пункта **Файл** основного меню, но можете «напечатать» в файл. Выбор можно сделать в диалоге настройки печати. В файл схема (или диаграмма) будет записана в формате «.ps», что может оказаться удобнее при подготовке публикации, поскольку качество печати может быть выше.

В Windows Vista для осуществления такого вывода в файл, видимо, потребуются дополнительные усилия, поскольку опция печати в файл не активируется, но в Linux эта возможность есть без каких-либо дополнительных программ и файлов.

Для просмотра файлов в формате «ps» в Windows можно использовать программу GSView. Ее же можно использовать для преобразования файла в другой формат. Так, чтобы вставить рисунок файла в текст редактора MS Word, потребовалось преобразовать его в другой формат. К сожалению, Word не позволяет вставить файл непосредственно. Но есть редакторы, которые работают с файлами формата «ps» без каких-либо «посредников».

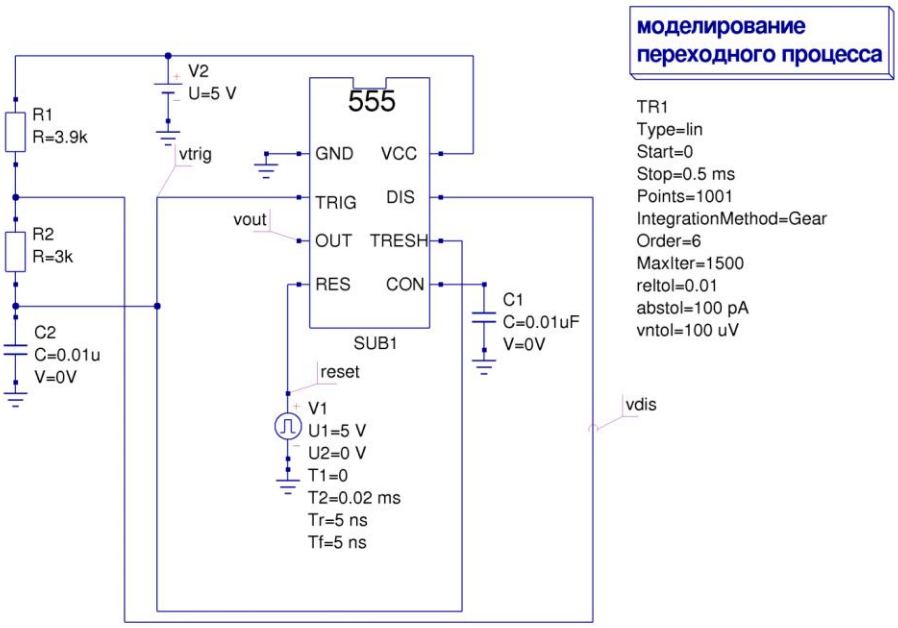


Рис. 4.18. Рисунок, преобразованный из формата «ps»

А так выглядит настройка печати в openSUSE 11.1.

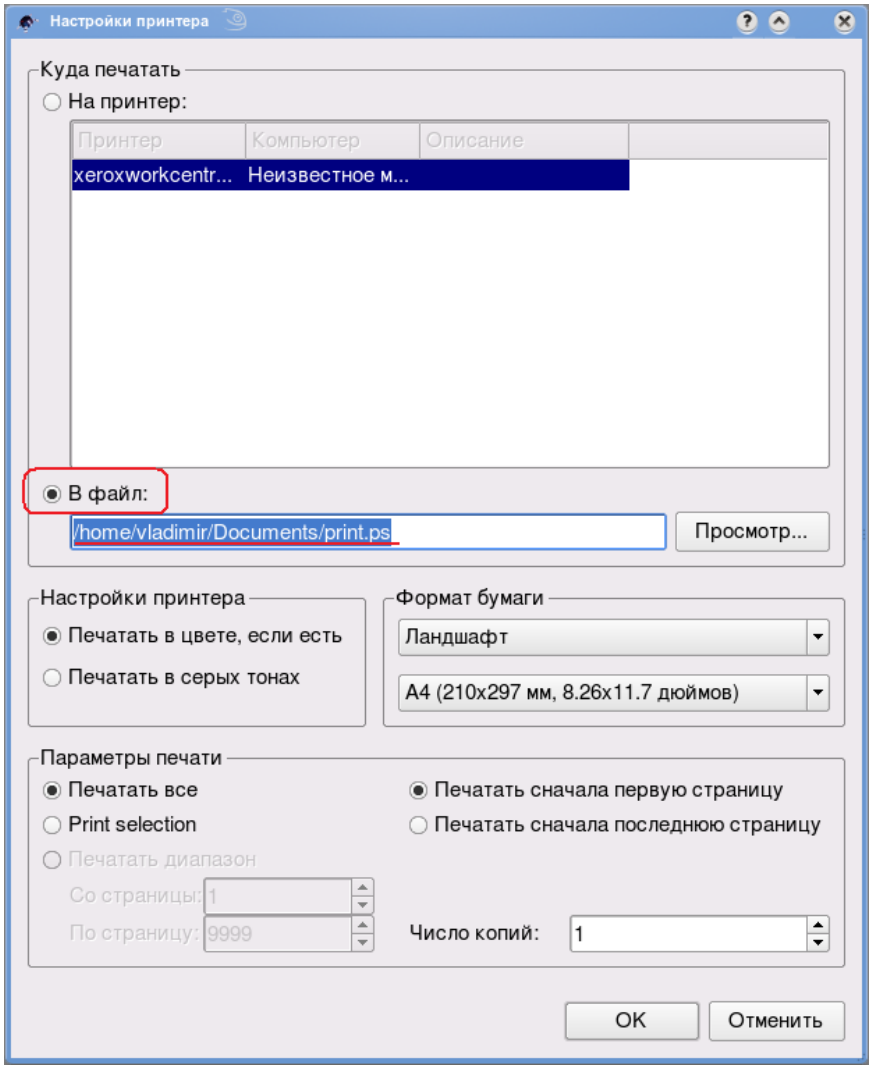
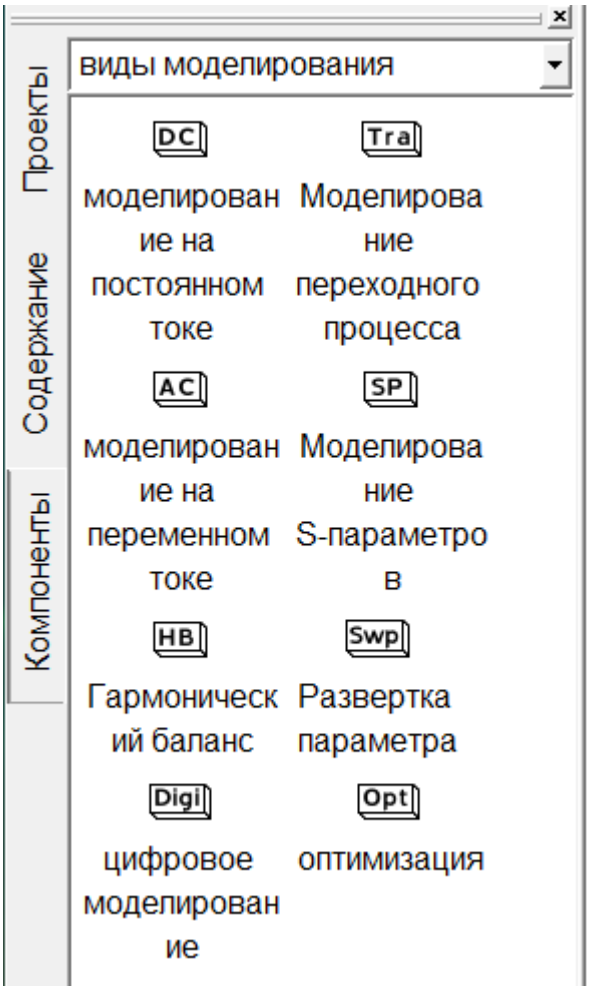


Рис. 4.19. Диалоговое окно настройки печати для вывода схемы в файл

Моделирование

Надо полагать, что вы потратили столько сил, рисуя схему, пусть и средствами Qucs, совсем не для того, чтобы полюбоваться результатами, будут ли они в формате векторной или растровой графики. Вам нужно получить информацию о работе схемы. Этому служат компоненты программы Qucs размещенные в группе **виды моделирования** закладки **Компоненты** панели навигации.



Давным-давно, лет двадцать назад, анализ работы схемы был прерогативой кабинетной работы с карандашом и бумагой с последующей проверкой на макете. Но и тогда было понятно, что нет нужды в повторении множества одинаковых операций раз за разом. Задача охотно перекладывалась на плечи калькуляторов. А с момента, когда компьютеры стали частью повседневности, как телефон или телевизор, научные калькуляторы уступили место компьютерным программам.

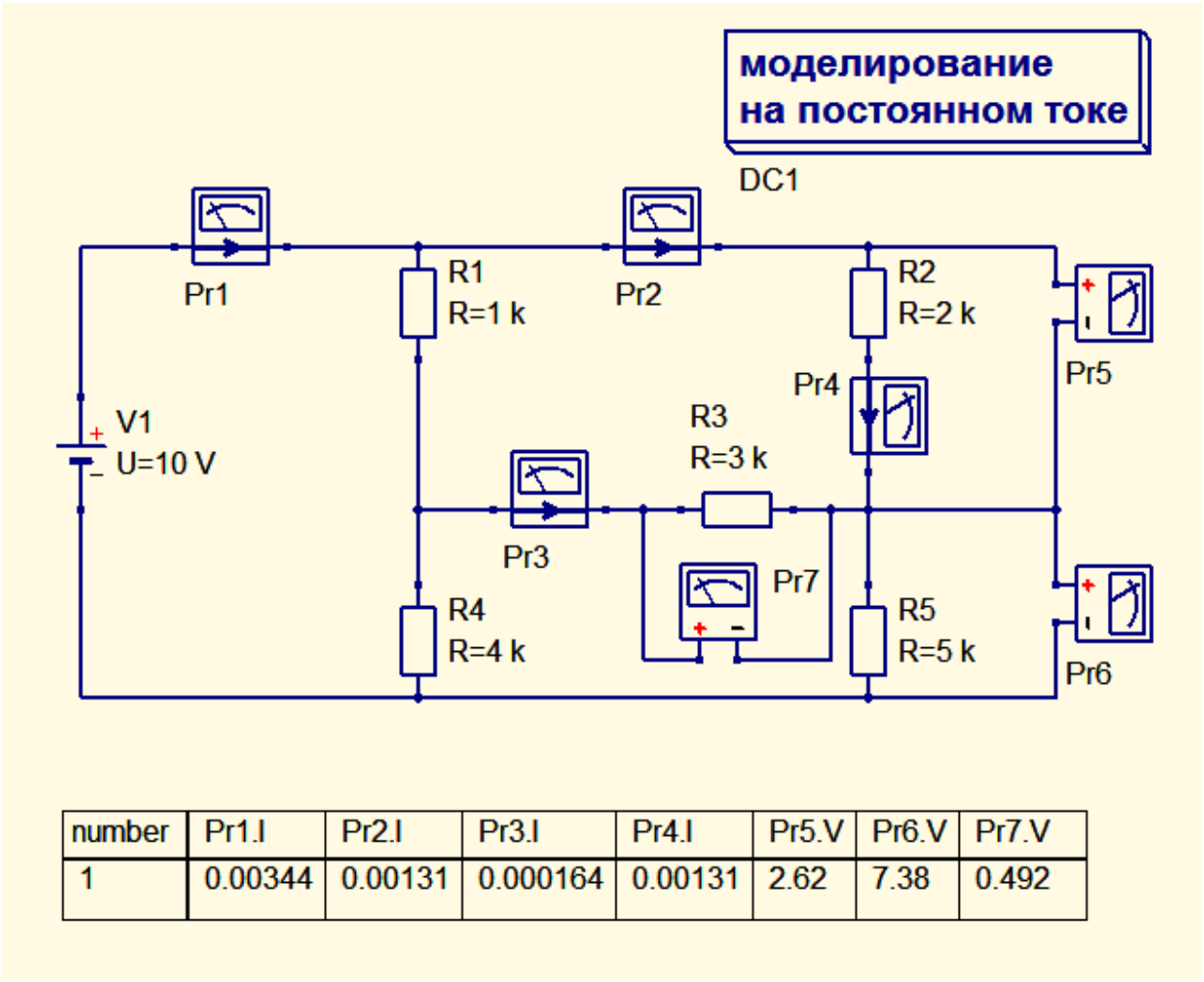
В 1960-70 годы началось создание программ и методов симуляции электрических схем в компьютерных программах. Одной из самых известных стала программа SPICE1, созданная в 1972 в Калифорнийском Университете в Беркли.

С тех пор идет непрерывная работа по совершенствованию и самих программ, и подходов к моделированию.

Рис. 4.20. Группа **виды моделирования**

Моделирование на постоянном токе, за редким исключением, обязательный этап работы. Хотя общая тенденция применения микросхем, выполняющих одну или множество функций устройства, может привести к неверному пониманию этой части работы. Но любой процесс, происходящий в электрической цепи, в каждый момент времени формально можно рассматривать как состояние схемы на постоянном токе, что делает особенно важным понимание работы устройств на постоянном токе. Да и сами микросхемы не растут на деревьях, а создаются инженерами, применяющими все классические методы разработки, подкрепленные сегодняшними инструментальными средствами, такими как программы.

Во многих случаях достаточно знать закон Ома, чтобы определить ток, напряжение или сопротивление. В более сложных случаях используются такие методы, как метод контурных токов или узловых потенциалов. Но гораздо проще использовать для решения этих задач компьютер, чем вычислять вручную. Вот простейший случай, который подтверждает это высказывание.



number	Pr1.I	Pr2.I	Pr3.I	Pr4.I	Pr5.V	Pr6.V	Pr7.V
1	0.00344	0.00131	0.000164	0.00131	2.62	7.38	0.492

Рис. 4.21. Определение токов и напряжений при моделировании на постоянном токе

При всей простоте схемы вычисления займут больше времени и отнимут больше сил, чем моделирование.

Моделирование переходного процесса, пожалуй, наиболее часто применяемый вид моделирования, когда нужно увидеть сигналы в электрических цепях. Этому виду моделирования соответствует применение осциллографа при налаживании и проверке электронных устройств. Некоторые программы, аналогичные Qucs, даже используют виртуальный осциллограф для этих целей. И этот виртуальный осциллограф в некоторых программах так похож на настоящий, что можно научиться работать с ним до того, как увидишь его в «живом» виде. На практике, однако, важнее не вид осциллографа, а вид осциллограмм. Получающие все большее распространение осциллографические приставки к компьютеру ориентированы именно на запоминание и показ осциллограмм. Эти же функции становятся обязательным атрибутом и остальных осциллографов.

Результаты моделирования переходного процесса в Qucs можно наблюдать в виде привычной диаграммы, если выбрать в группе **диаграммы Декартовский** вид отображения данных. Сигналы могут быть импульсными или синусоидальными, или любой произвольной формы. Но они соответствуют тому, что вы увидели бы на экране осциллографа, если проверяли бы работу схемы на макетной плате.

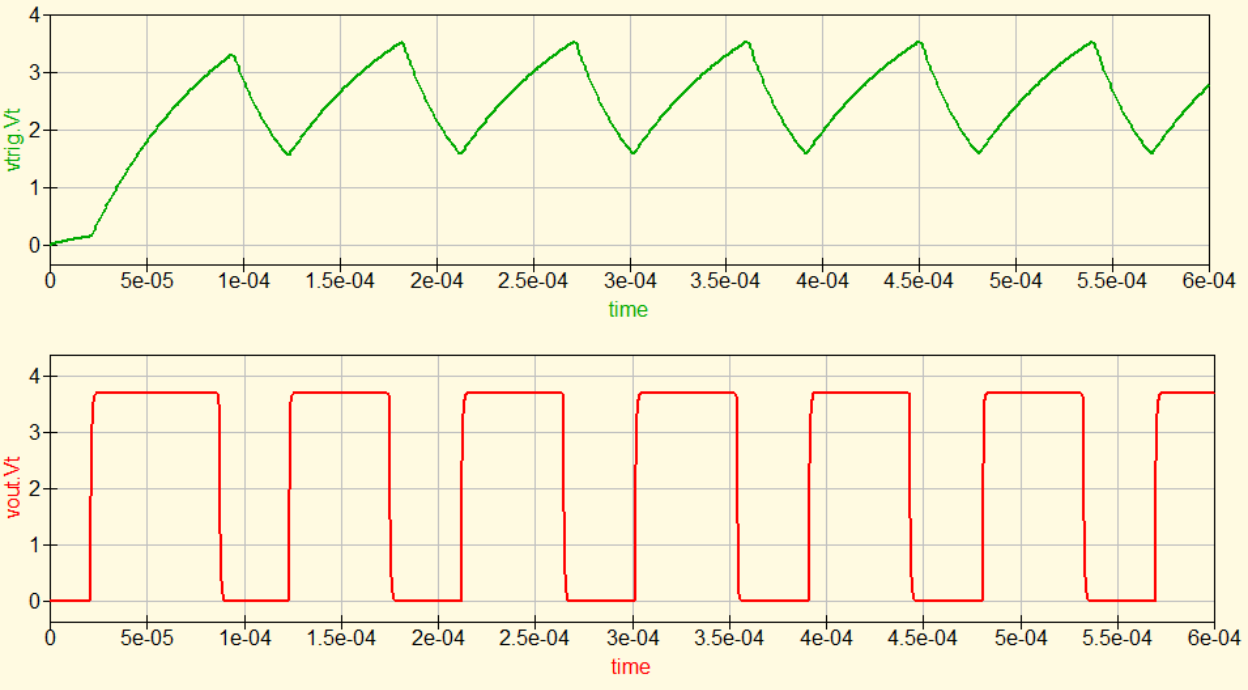


Рис. 4.22. Осциллограмма сигнала, полученного при моделировании переходного процесса

Моделирование на переменном токе позволяет вам получить амплитудно-частотную и фазочастотную характеристики электрической цепи. При работе с усилителями, а это очень значительная часть электрических схем, знание этих характеристик очень важно, например, при введении отрицательной обратной связи.

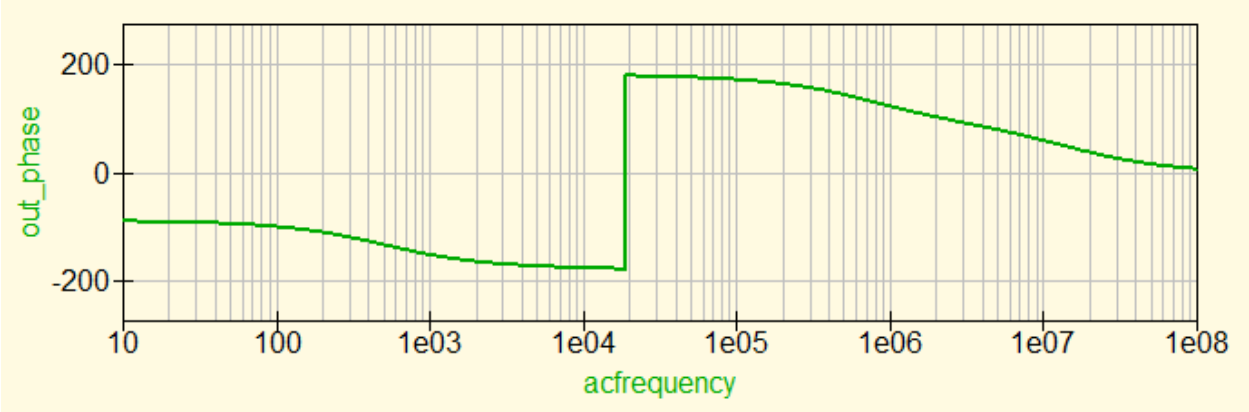


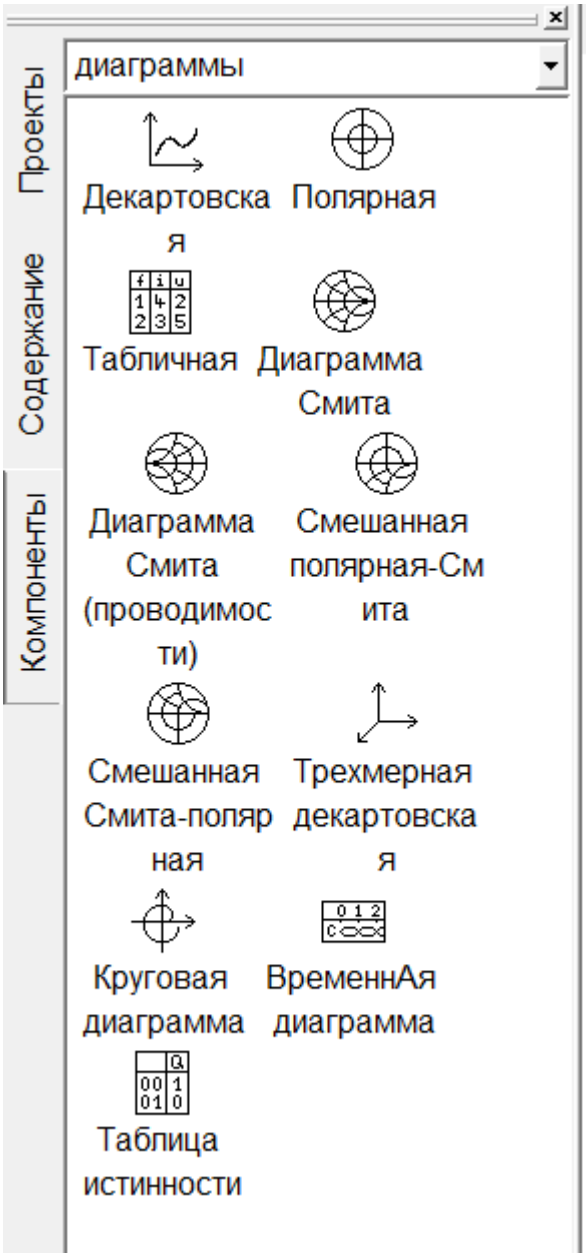
Рис. 4.23. Фазочастотная характеристика усилительного каскада

Для получения фазочастотной характеристики достаточно для выходного сигнала записать уравнение вида: `out_phase = phase(output.v)`, – а в диалоговом окне свойств диаграммы выбрать эту переменную и логарифмическую разметку оси X.

К моделированию электрических цепей мы еще вернемся в рассказе обо всех видах моделирования. Но сейчас отметим, что введя схему и выбрав вид моделирования, можно осуществить проверку работоспособности схемы и определить многие ее параметры.

Диаграммы

Группа компонентов с таким названием дает возможность выбрать, в каком виде вы хотите увидеть полученные после симуляции данные.



Некоторые виды диаграмм, как **Декартовская** или **Табличная**, мы уже использовали при рассмотрении основного меню и некоторых видов симуляции.

Другие виды диаграмм, как **Диаграмма Смита** или **Трехмерная**, мы рассмотрим позже, когда начнем рассказ о применении **моделирования S-параметров**.

И мы еще вернемся и **Декартовской**, и **Табличной** диаграммам.

Сейчас только отметим, что при цифровом моделировании используются два вида диаграмм: **Таблица истинности** и **Временная диаграмма**.

Рис. 4.24. Группа **диаграммы** закладки **Компоненты** панели навигации

Для моделирования цифровых схем используются цифровые источники сигнала, цифровое моделирование и цифровые виды отображения данных. Все это можно найти в группе цифровые компоненты.

У многих складывается неверное представление о симуляции электрических схем во многом благодаря цифровому моделированию. Считается, что логические вентили наилучшим образом можно поручить симулировать компьютеру, благодаря «родству душ», тогда как аналоговые устройства компьютер не понимает, и ему трудно объяснить, что следует делать. Это не совсем так, но цифровое моделирование во многих программах разделено с аналоговым видом симуляции.

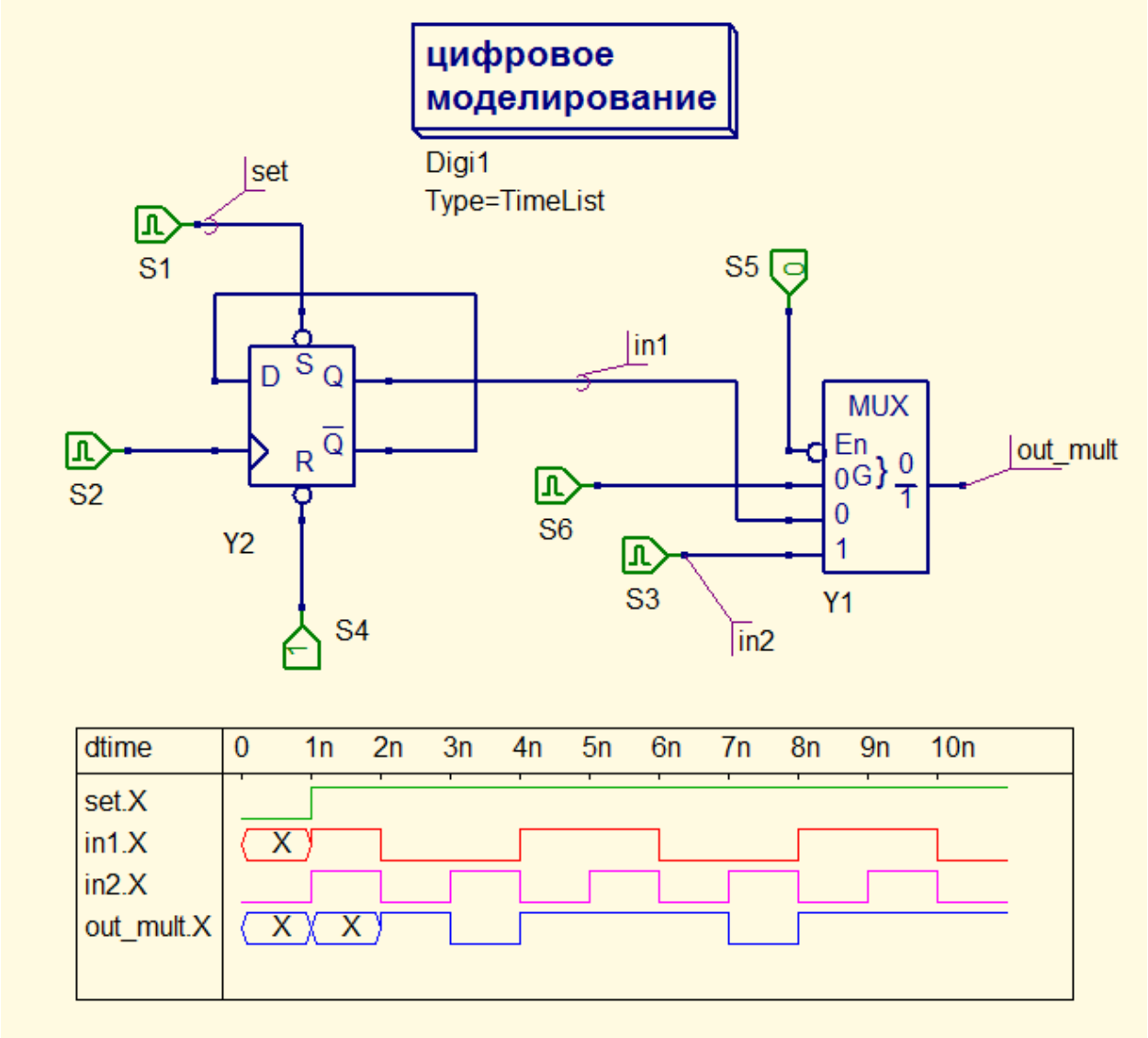


Рис. 2.25. Моделирование цифровой схемы

Чтобы развеять сомнения, показать, что не все так просто, как об этом можно подумать, если не очень задумываться, я хочу привести схему достаточно типичную для цифровых устройств – схему синхрогенератора.

Синхрогенераторы или тактовые генераторы в цифровых устройствах часто строят с использованием логических вентилей. При этом не следует забывать, что логические вентили, например, на базе ТТЛ, не более чем усилительные каскады, устроенные так, чтобы наилучшим образом работать с цифровыми сигналами, но все-таки усилители.

И понятно, что задавая определенное напряжение на входе вентиля, его можно превратить в усилитель, а, введя обратную связь, в генератор. А подобное построение, хотя оно и логично, но, скорее, следует отнести к аналоговым схемам, а не к цифровым.

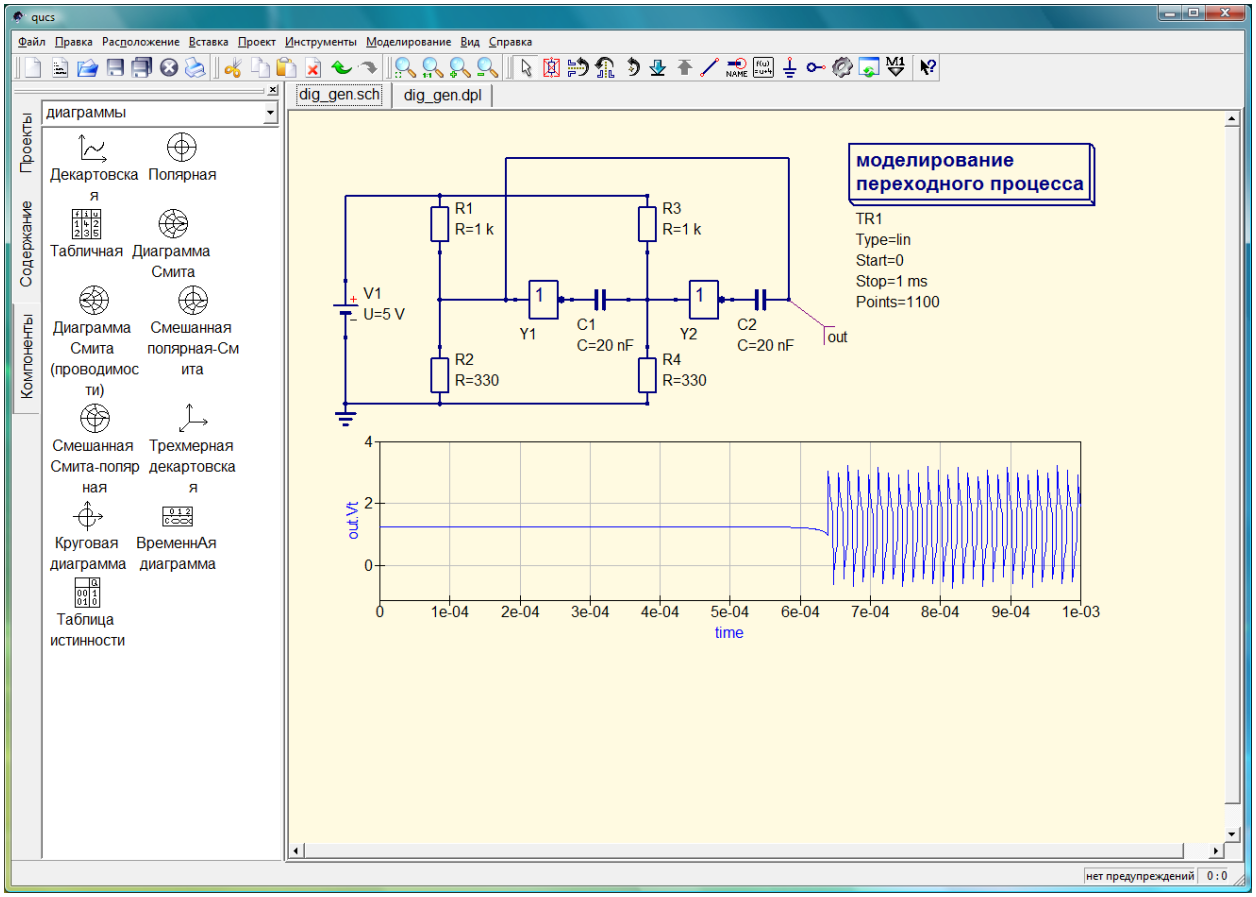
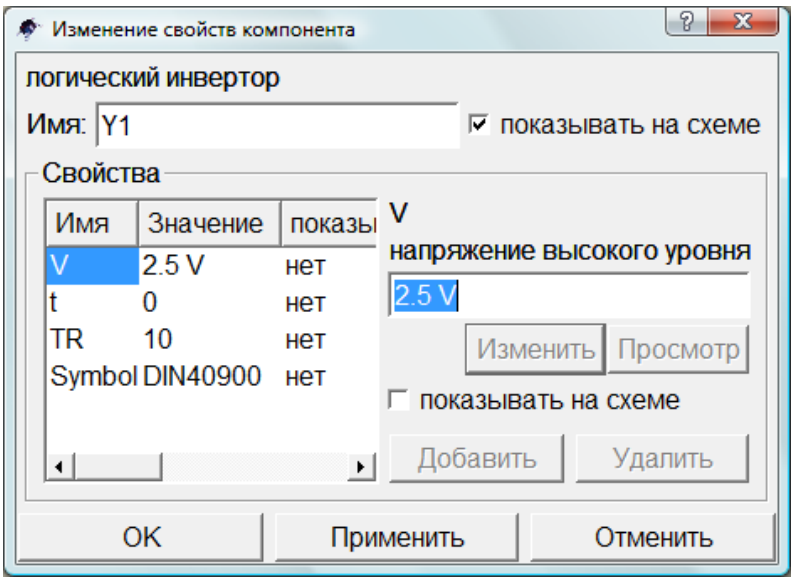


Рис. 2.26. Генератор на цифровых вентилях в Qucs

Чтобы схема заработала в программе Qucs, в свойствах инвертора (оба элемента) следует изменить напряжение логической единицы.



А если изменить время задержки до 5 ns у одного из вентилей, параметр t по умолчанию равный нулю, что больше похоже на реальное положение дел, то изменится и характер работы схемы.

Рис. 2.27. Диалоговое окно свойств цифрового инвертора

Диаграммы, как и другие компоненты в программе Qucs, имеют свои свойства, которые настраиваются в окнах диалога. Эти настройки зависят от того, что вы намерены увидеть на диаграмме. А это, в свою очередь, зависит от тех видов моделирования, что вы применяете к электрической цепи. Программа Qucs позволяет комбинировать несколько видов моделирования. Так, скажем, легко можно получить ответ на вопрос, как влияет на амплитудно-частотную характеристику цепи изменение одного из компонентов. Рассмотрим простейший пример – RC цепь.

Нам потребуется источник переменного напряжения (группа **источники** на закладке **Компоненты**), резистор и конденсатор из группы **дискретные компоненты**.

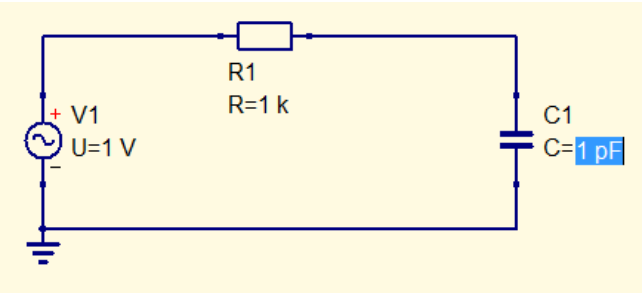


Схема столь проста, что ее можно было бы и не приводить отдельно, но хочу еще раз напомнить, что изменить значение компонента можно, выделив его щелчком левой клавиши мышки по нему, а затем впечатав новое значение.

Рис. 2.28. Простейшая интегрирующая RC цепь

В качестве значения емкости введем *cap*. Это не величина емкости, а имя переменной для вида моделирования **Развертка параметра**, который мы применим к RC цепи. А для отображения полученных данных используем **моделирование на переменном токе**. Последний вид моделирования даст нам амплитудно-частотную характеристику цепи, а развертка (или качание) параметра позволит оценить влияние меняющейся переменной «*cap*» на эту характеристику.

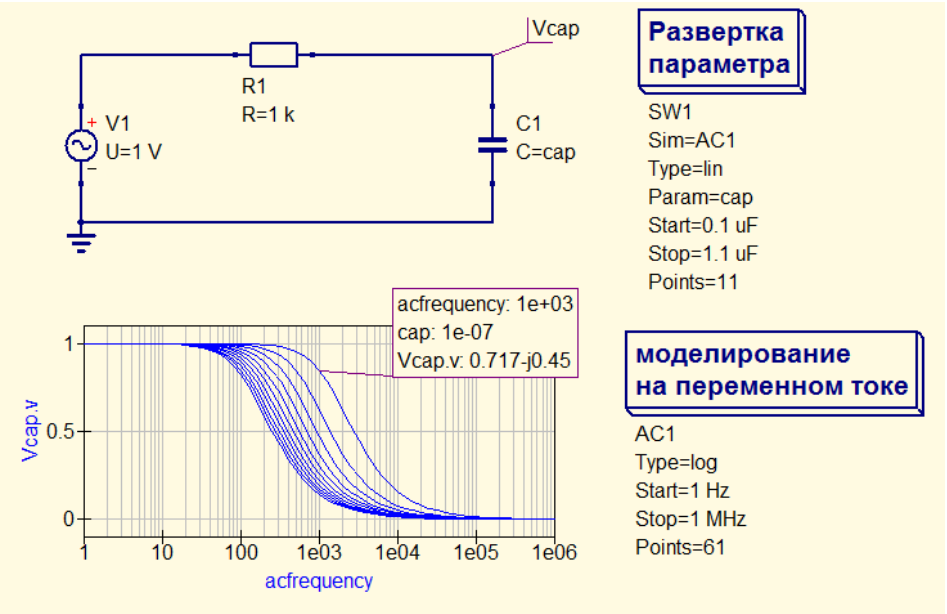


Рис. 4.29. Тестирование RC цепи в режиме изменения емкости

При настройках этой схемы, настройках видов моделирования, следует придерживаться тех параметров, которые указаны на рисунке. А для диаграммы выбрать логарифмическую разметку оси X. Иначе вид диаграммы будет совсем «невнятным».

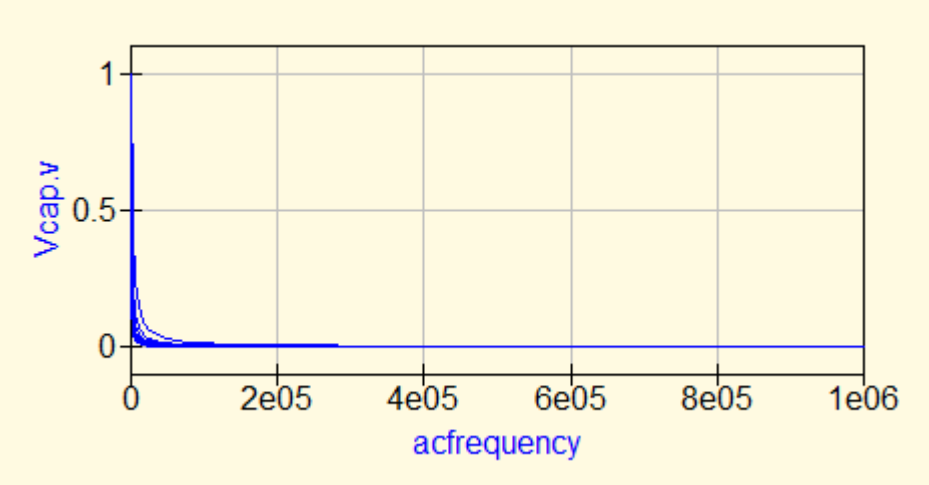


Рис. 4.30. Вид диаграммы при линейной разметке оси X

Получить нужные характеристики настраиваемой цепи можно разными способами, используя разные виды моделирования и встроенные параметры компонентов. Выбор тех или иных приемов зависит от опыта работы с программой и поставленных перед собой задач. Qucs – программа достаточно универсальная, чтобы выбор был.

Часть 2. Работа с программой

Глава 1. Компоненты

В предыдущей главе мы рассмотрели основные элементы программы, что в сочетании с хорошо написанным разделом быстрого начала работы с программой в разделе **Справка**, позволяет вам не только начать работу, но и многое к настоящему моменту успеть попробовать, если программа уже установлена на компьютере. Однако рассмотрим подробнее компоненты по их группам, отображаемым на закладке Компоненты панели навигации и в **Библиотеке компонентов** (пункт **Инструменты** основного меню).

Дискретные компоненты

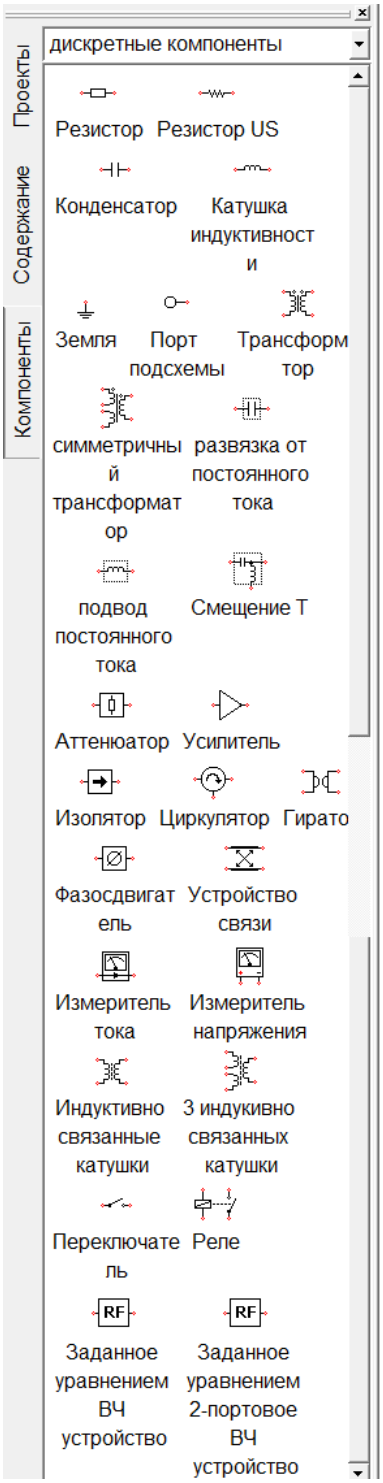


Рис. 5.1. Группа дискретных компонентов

Наиболее употребительными компонентами из этой группы, пожалуй, будут резистор, конденсатор, индуктивность. Они не только обеспечивают работу схемы, корректируют амплитудно-частотную характеристику или составляют основу любого фильтра, как «бойцы невидимого фронта» они присутствуют в самых неожиданных местах собранного устройства и могут свести на нет все усилия по созданию схемы.

Самыми загадочными для непосвященных будут, например, **Циркулятор** или **Гиратор**.

Циркулятор — это пассивный электронный компонент с тремя или более портами, в котором порты могут достигаться следующим образом: когда сигнал подается в любой порт, он передается только в следующий; первый порт считается следующим за последним. Когда один порт трехпортового гиратора подключен к согласованной нагрузке, он может использоваться как изолятор, поскольку сигнал может переходить только в одном направлении между оставшимися портами.

Гиратор или ***инвертор пассивного импеданса*** — электрическая цепь, которая инвертирует полное сопротивление. Другими словами, она может сделать емкостную цепь индуктивной, полосовой фильтр станет режекторным, и т.д. Часто используется при разработке активных фильтров и миниатюризации.

Фазосдвигатель — это микроволновая цепь, которая поддерживает управляемый фазовый сдвиг RF сигнала. Фазосдвигатели используются в фазированных решетках.

Устройство связи — это небольшое устройство, позволяющее вам соединить два или больше сетевых кабеля вместе.

Обычно используются для расширения сети.

Многие компоненты из этой группы предназначены к экспериментам с радиочастотными цепями. Это и **развязка от постоянного тока**, используемая, скажем при получении S-параметров цепи, как и **Смещение Т**.

Вот пример использования этих компонентов.

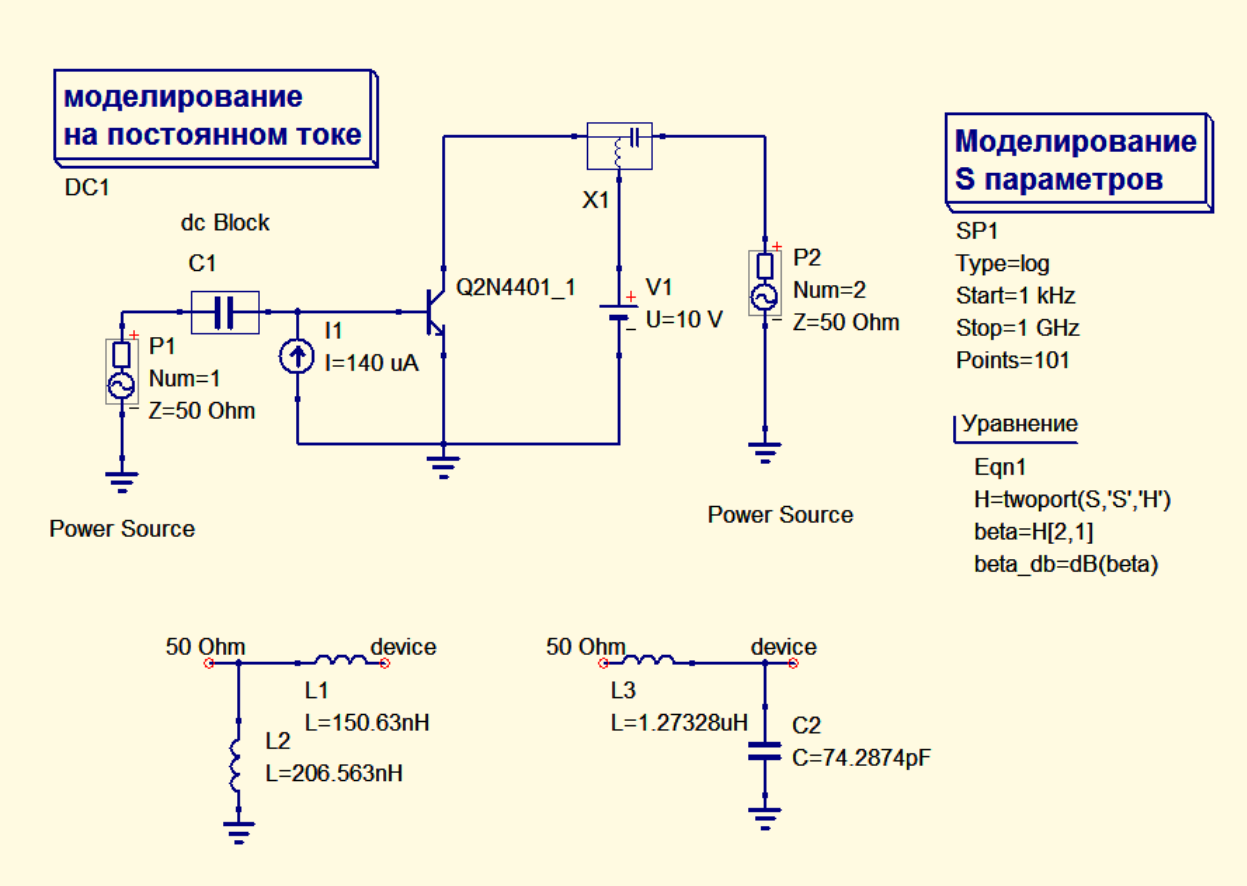
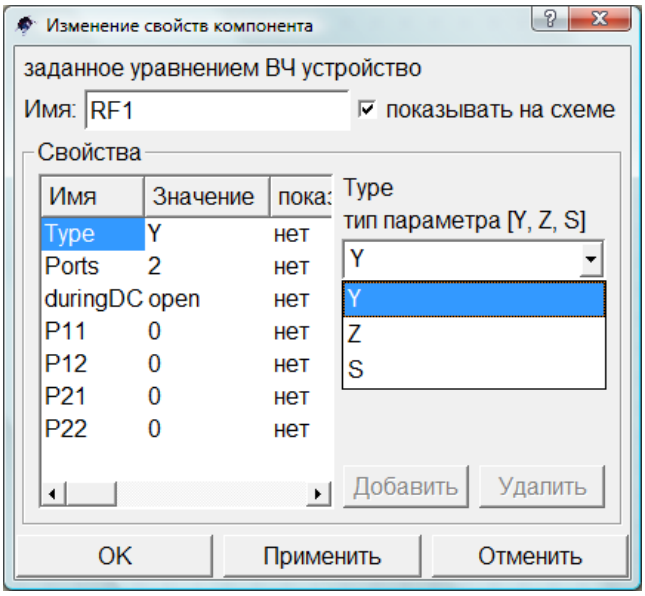


Рис. 5.2. Пример моделирования S-параметров

В схеме C1 – **развязка от постоянного тока**, а X1 **Смещение Т**. В таком виде их назначение становится яснее.

Очевидно, что к исследованиям радиочастотных цепей предназначены и компоненты заданных уравнениями ВЧ устройств. В диалоговом окне свойств компонентов вы можете выбрать тип параметров, которые будут описывать это устройство. Для двухпортового устройства список длиннее.

Малосигнальные параметры достаточно хорошо описывают поведение радиочастотных цепей, поэтому часто применяются при проектировании. Есть методики применения этих параметров и для низкочастотных устройств. Программа Qucs позволяет вам получить малосигнальные параметры и использовать их в ваших проектах.



При необходимости вы можете перевести один тип параметров в другой.

Программа Qucs больше ориентирована в этом плане на ВЧ устройства, на работу с S-параметрами, но позволяет вам получить набор, например, Y-параметров.

Рис. 5.3. Диалоговое окно заданного уравнением ВЧ устройства

Измеритель тока и **Измеритель напряжения** включены в группу, скорее, для удобства. Они имеют свою группу – **измерители**. Оба виртуальных прибора можно использовать в тех случаях, когда используются амперметр и вольтметр (различия между постоянным током и переменным нет), а также использовать их в качестве переменных для получения кривых в осциллограммах.

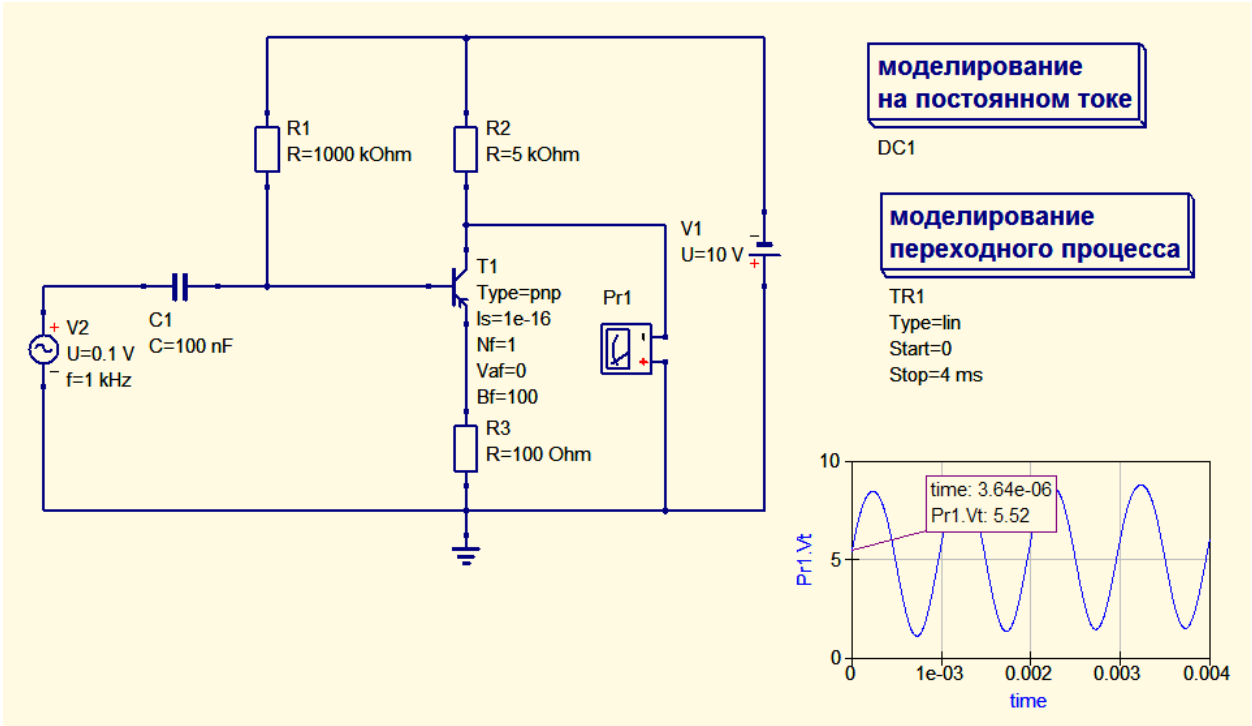
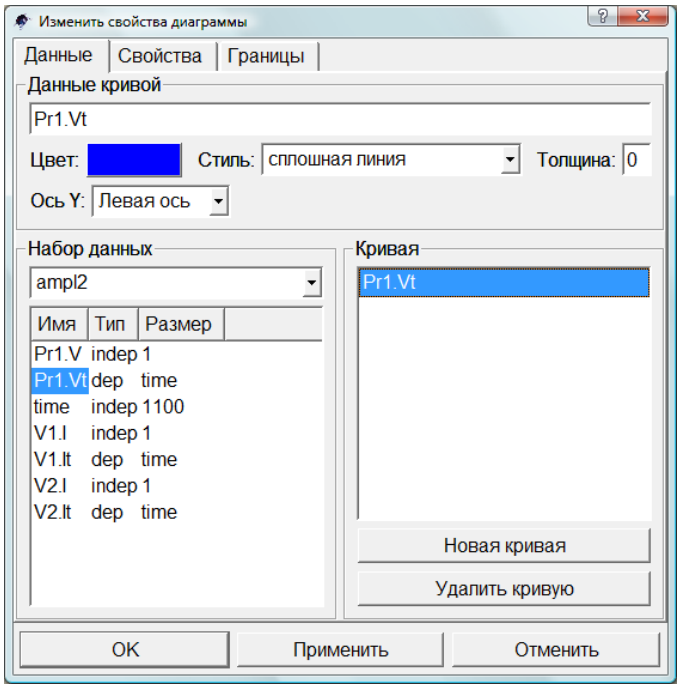


Рис. 5.4. Использование измерителя напряжения для получения осциллограммы

В диалоговом окне декартовской диаграммы достаточно выбрать прибор Pr1, чтобы получить осциллограмму.



Удобство использования измерителей проявляется еще и в том, что кроме осциллограммы можно получить табличные значения напряжений или токов.

С помощью маркеров, как показано на рисунке, легко определить величину напряжения в любой точке осциллограммы, но с той степенью достоверности, с которой удается рассмотреть график. Табличная форма лишена этого недостатка, хотя длинный список измеренных значений может свести на нет все преимущества.

Рис. 5.5. Диалоговое окно декартовской диаграммы для предыдущей схемы

Аттенюатор удобен тем, что вы задаете искомое ослабление мощности сигнала в децибелах, а в свойствах компонента можете задать необходимое полное сопротивление.

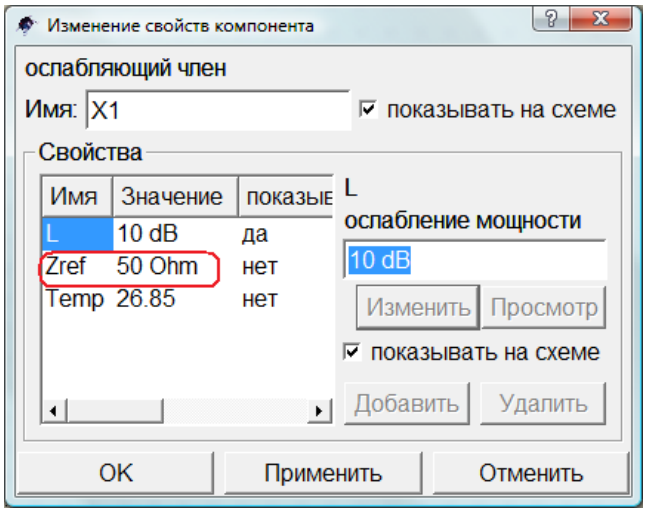
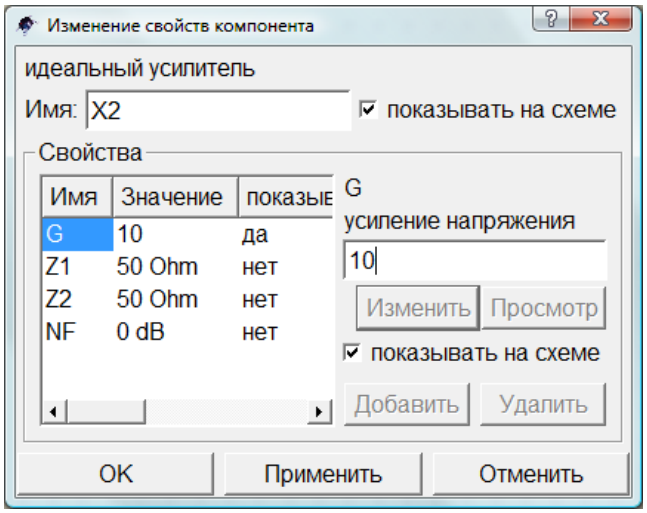


Рис. 5.6. Диалоговое окно свойств аттенюатора

В программе Qucs есть целая группа, которая называется **линии передачи данных**. Вы можете оценить влияние линии на работу устройств. Но иногда вас может интересовать только ослабление сигнала, вносимое линией. В этом случае удобно рассматривать линию, как аттенюатор.

Кроме того, в сочетании с моделированием развертки параметра вы может использовать аттенюатор, как устройство регулировки.

Усилитель – это идеальный усилитель.



Очень часто, особенно в сложных устройствах, вас не интересует, как устроен усилитель. Так происходит при использовании микросхем или в иерархическом построении схемы. Позже этот идеальный усилитель будет замен реальной схемой, а пока вы используете его для получения нужного усиления. Вы можете задать входное и выходное сопротивление. А параметр NF – фактор шума.

Рис. 5.7. Диалоговое окно свойств усилителя

Трансформатор можно использовать в экспериментах по созданию блоков питания, например.

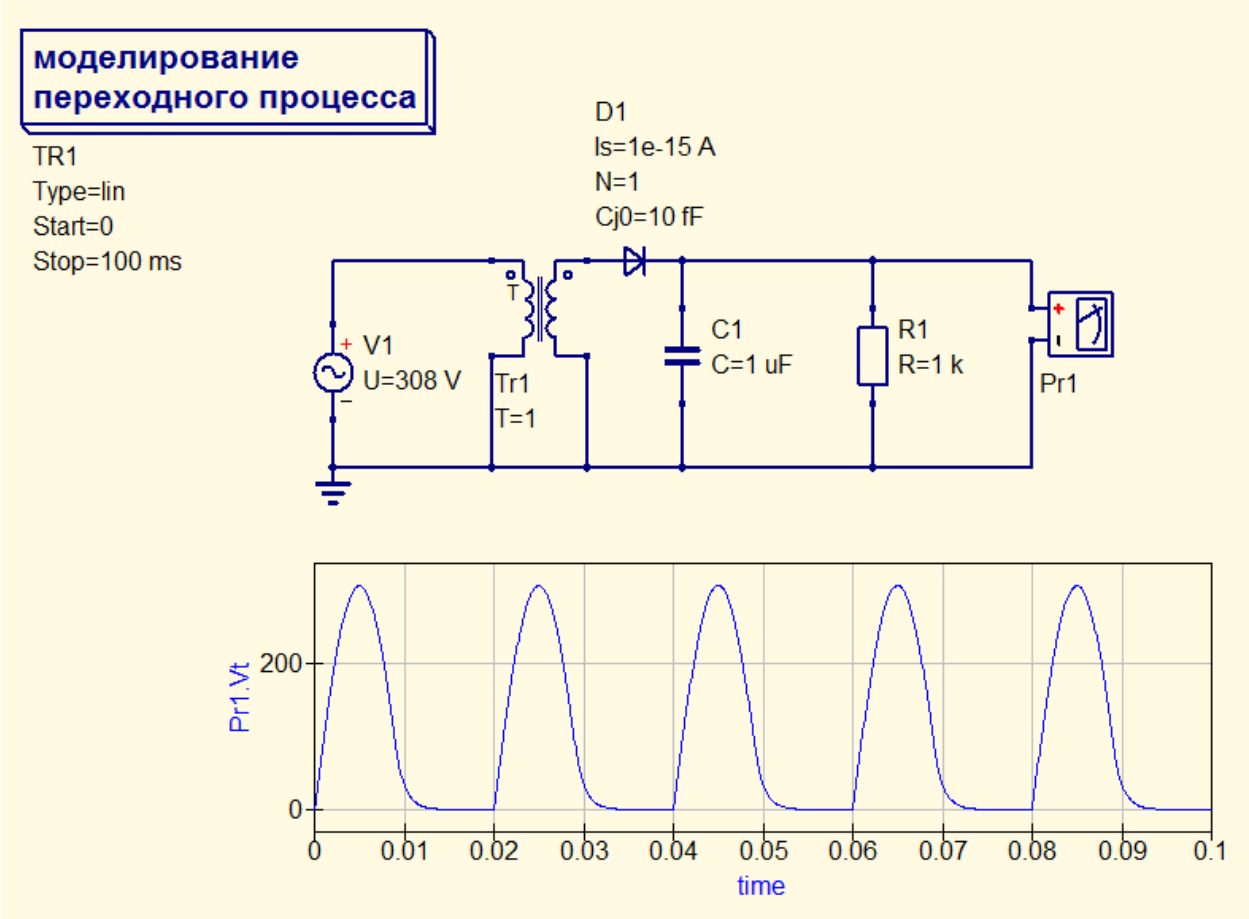


Рис. 5.8. Выпрямитель промышленного однофазного напряжения

В диалоговом окне трансформатора вы можете изменить только один параметр – коэффициент трансформации. Аналогично можно использовать и **симметричный трансформатор**.

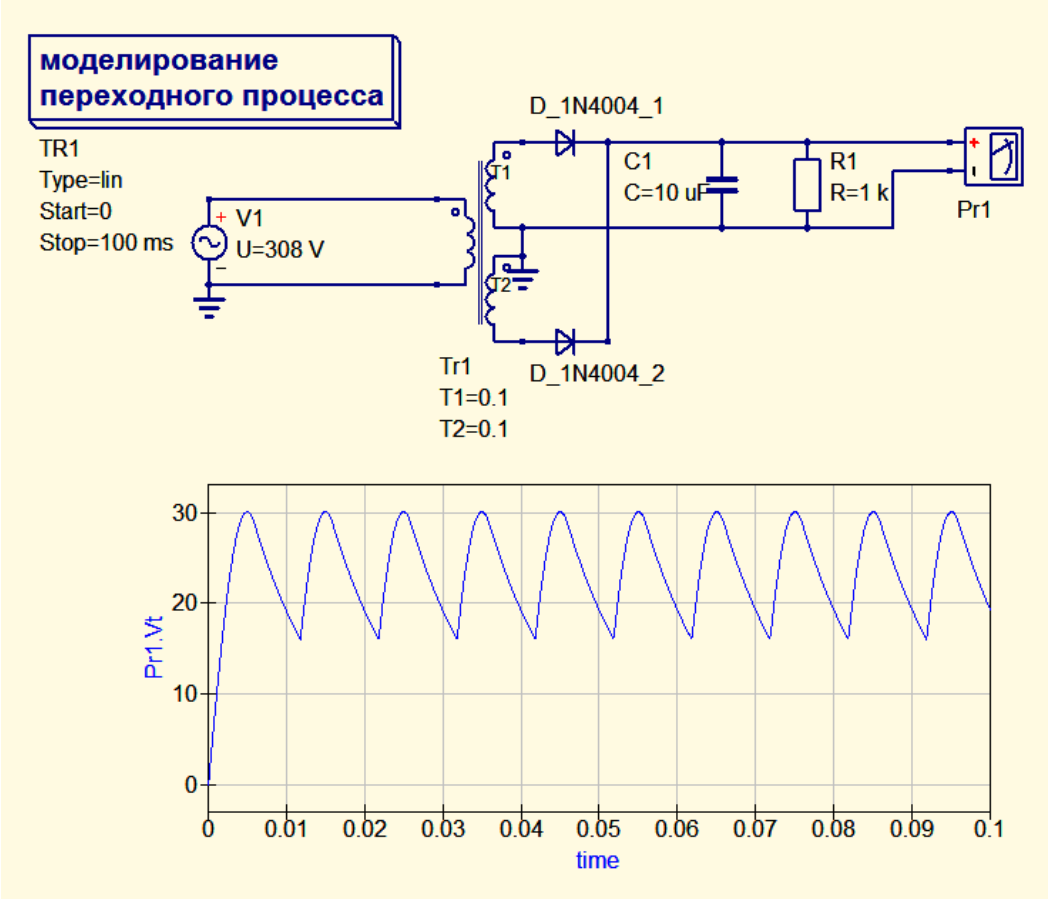


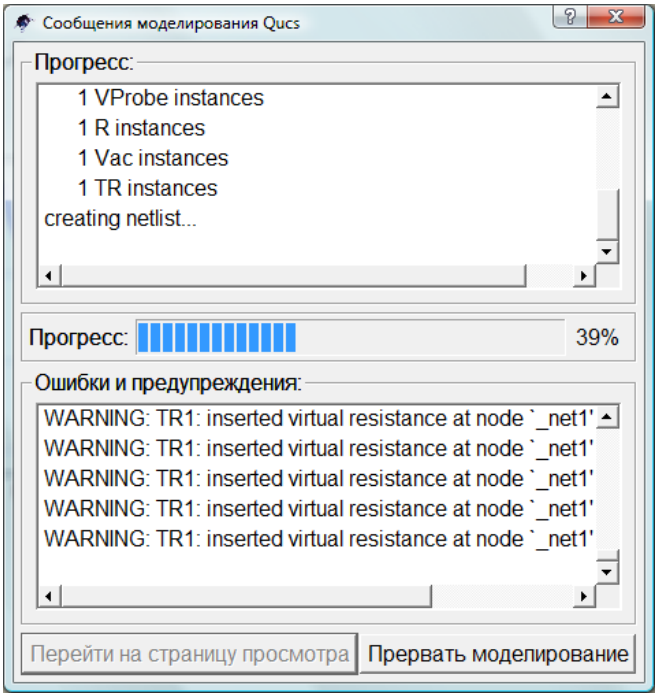
Рис. 5.9. Выпрямитель с симметричным трансформатором

Напряжение источника переменного напряжения задается пиковое, поэтому вместо 220 В задано 308 В, 50 Гц.

> И еще одно замечание: в реальном устройстве вы не будете соединять входную обмотку трансформатора с общим проводом, но для моделирования это лучше сделать. Иначе даже на достаточно современном компьютере процесс вычислений становится столь затяжным, что можно и не дождаться его окончания.

Конечно, пользователю совсем не обязательно знать, на каких математических принципах построена работа программы. Но следует иметь в виду, что каждое моделирование электрической цепи связано с огромной вычислительной работой компьютера. Во многих построениях схем следует обращать внимание на время, задаваемое для отображения результатов и количество расчетных точек. Слишком большие значения могут приводить к торможению процесса симуляции, не давая ничего нового в конечном результате.

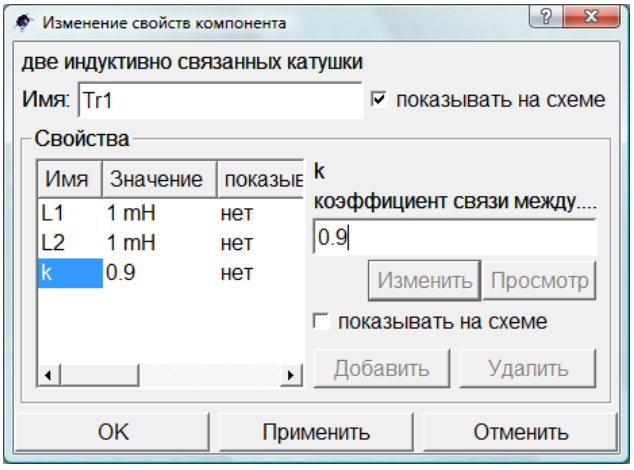
Иногда процесс моделирования удачно проходит со значениями, заданными по умолчанию, а при следующем повторе с нужными параметрами походит быстрее, чем тогда, когда нужные параметры заданы для первого моделирования.



Сообщение, отображаемое в окне прогресса симуляции, показывает, что программа добавляет виртуальные резисторы в сеть трансформатора. Как правило, это резисторы очень большого сопротивления, но они нужны программе для расчетов.

Рис. 5.10. Прогресс моделирования предыдущей схемы без земли на первичной обмотке

Если вас интересует нечто большее, чем коэффициент трансформации, вы можете использовать такой компонент, как **Индуктивно связанные катушки**.



Помимо дополнительной возможности задать индуктивность первичной и вторичной обмоток, есть возможность изменить и коэффициент связи между обмотками.

Но и то, и другое нужно знать.

Рис. 5.11. Диалог свойств компонента **Индуктивно связанные катушки**

Если в схеме выпрямителя заменить трансформатор индуктивно связанными катушками, то процесс симуляции идет очень долго, а результаты, отображенные, как и в первом случае на осциллограмме, мало чем различаются. Конечно, применение катушек индуктивности вместо трансформатора позволяют глубже понять работу сетевого адаптера, но подобная замена, все-таки, больше подходит импульсному блоку питания, а не сетевому адаптеру.

Тоже относится и к следующему компоненту – **3 индуктивно связанным катушкам**. Импульсный блок питания или генератор, вот типовое место применения этого компонента.

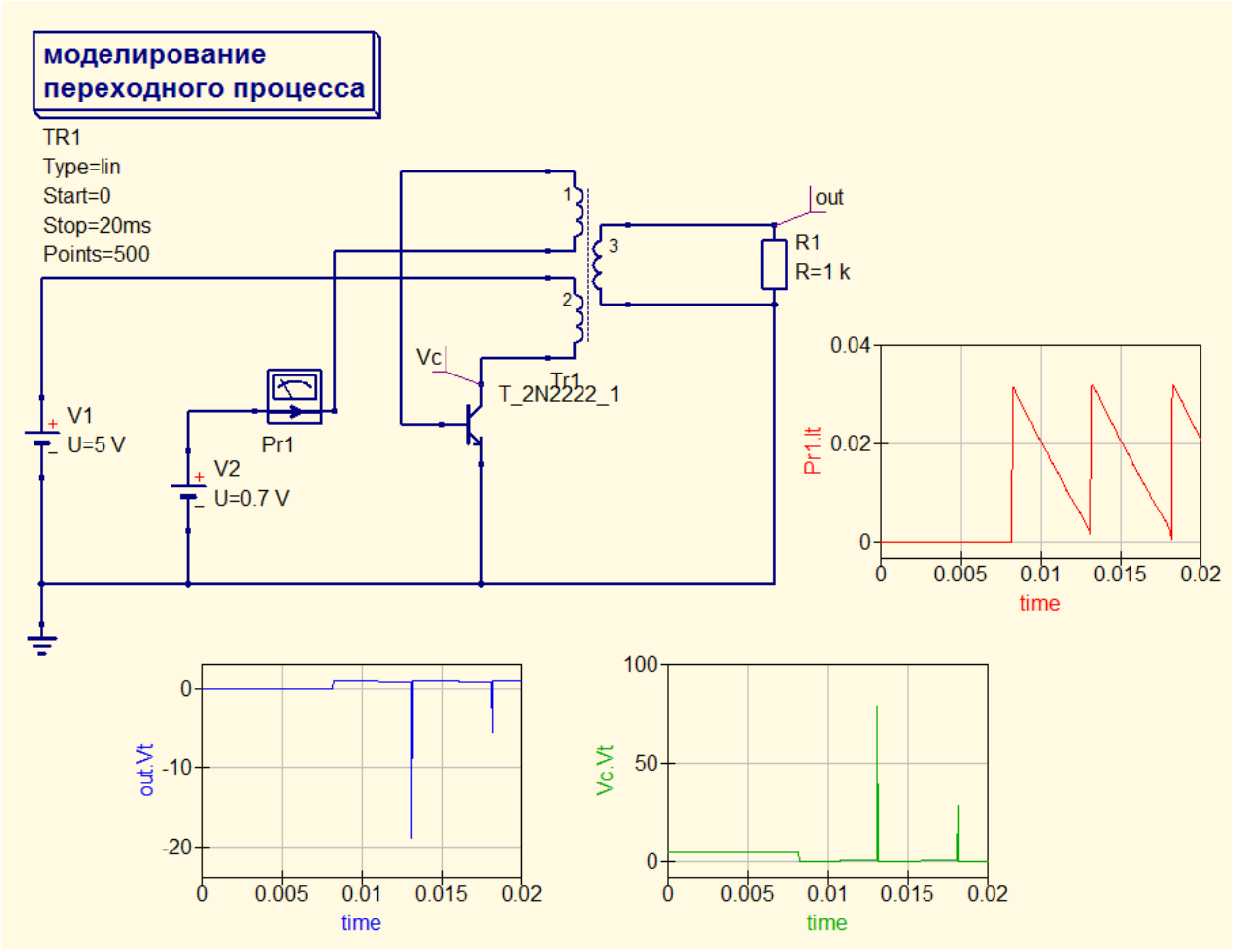
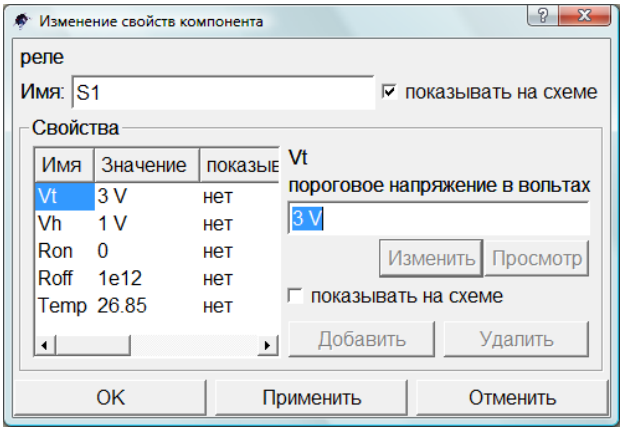


Рис. 5.12. Применение индуктивно связанных катушек

Реле и **Переключатель**, входящие в состав группы дискретных компонентов, можно было бы опустить из рассмотрения. Если бы ни некоторые детали. Вот диалоговое окно свойств реле.



Реле имеет два основных «входных» параметра – пороговое напряжение и величину гистерезиса. Но сопротивления обмотки реле нет. Поэтому в схеме, скажем, при питании реле через транзистор параллельно реле следует включить резистор.

Рис. 5.13. Диалоговое окно свойств реле

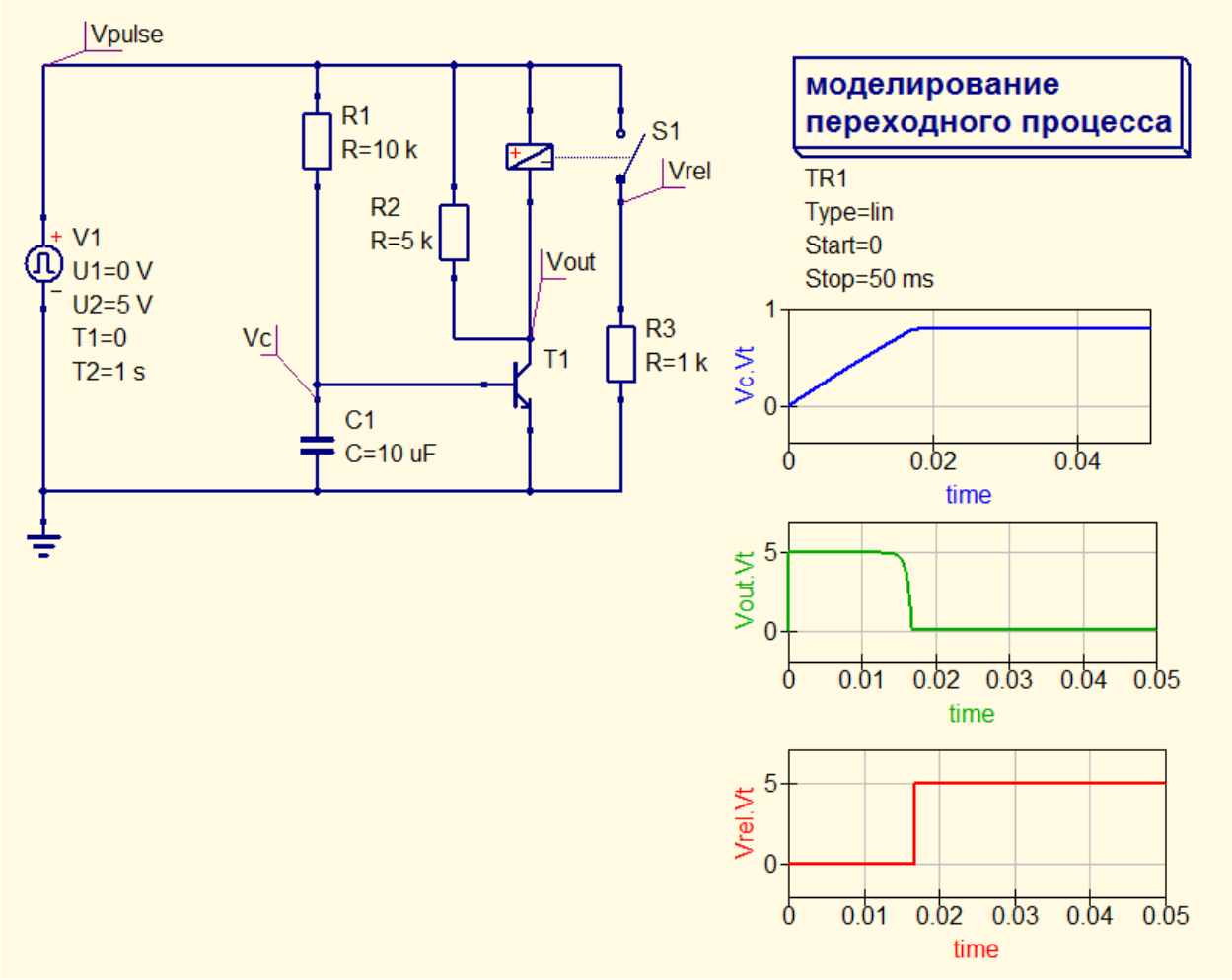
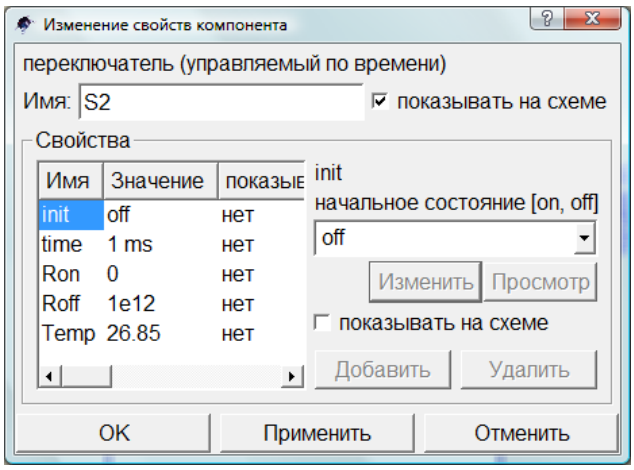


Рис. 5.14. Схема с использованием реле

Переключатель имеет свое диалоговое окно свойств.



Можно задать начальное состояние переключателя и время, когда его состояние изменится. Можно изменить переходное сопротивление контактов переключателя во включенном состоянии и сопротивление утечки разомкнутых контактов.

Рис. 5.15. Диалоговое окно свойств **Переключателя**

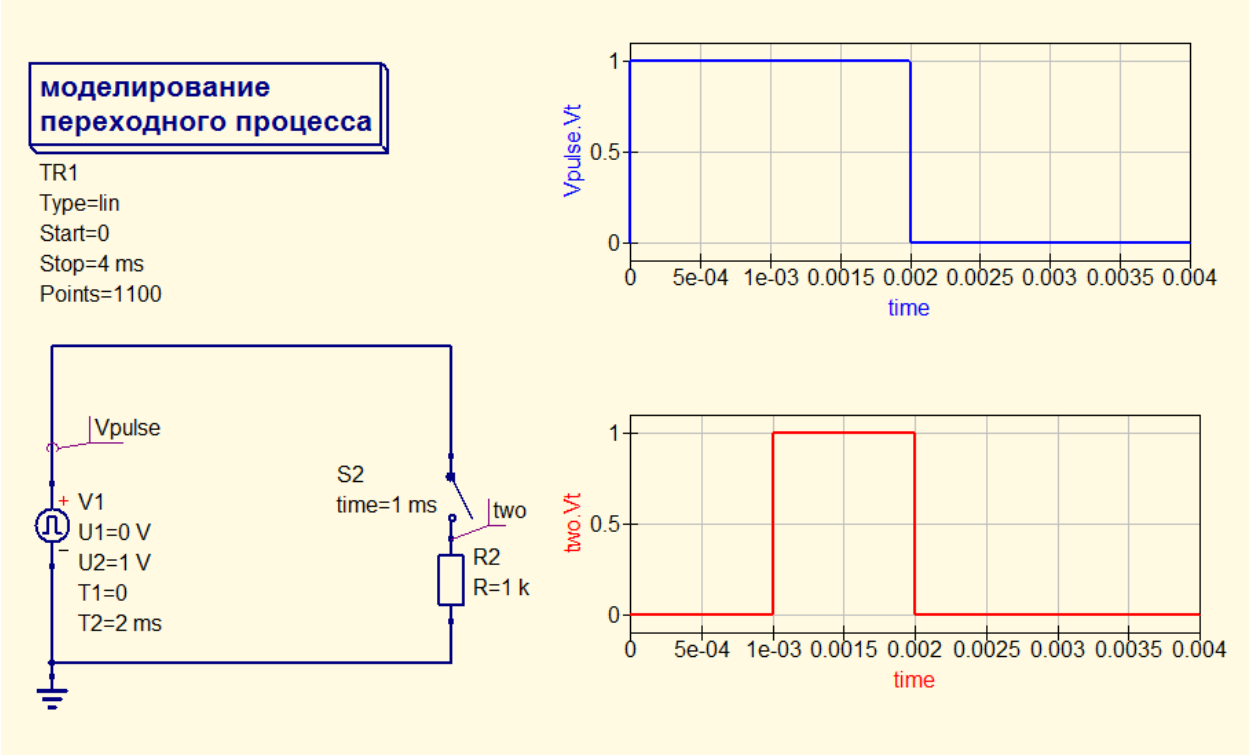


Рис. 5.16. Работа компонента Переключатель

Как и в случае с реле, рассматривать работу **Переключателя** удобнее при подключении его к источнику импульсного напряжения.

И есть еще один компонент, **Порт подсхемы**. Как и следует из его названия, этот компонент удобен для реализации подсхем. Подсхему легче отладить отдельно от основной схемы, при необходимости ее легче видоизменить отдельно от основной схемы. А для соединения с основной схемой подсхема должна быть снабжена портами.

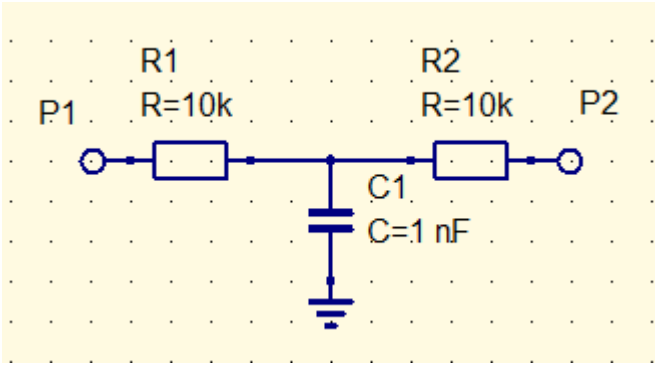


Рис. 5.17. Применение портов P1 и P2 для создания подсхемы

К портам и подсхемам мы еще вернемся.

Источники

По количеству доступных источников раздел **Компоненты** панели навигации не уступает группе дискретных компонентов.

Чаще всего, вероятно, используются такие источники, как источник напряжения постоянного тока, источник напряжения переменного тока и источник напряжения прямоугольной формы.

Источник напряжения постоянного тока – это и батарейка, это и блок питания, присоединяемый к устройству.

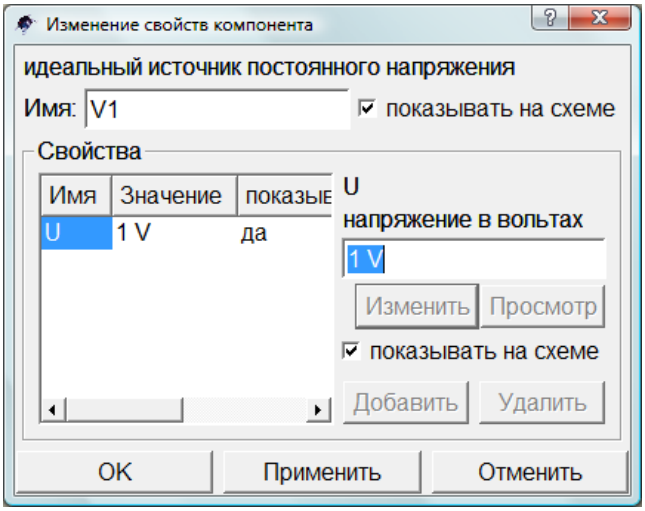


Рис. 6.2. Диалоговое окно источника постоянного тока

Единственный параметр источника – напряжение. Оно может быть и единицами вольта, и тысячами вольт. Реальные источники имеют внутреннее сопротивление; чтобы приблизить источник постоянного напряжения к реальному аккумулятору или батарейке достаточно включить последовательно с ним резистор, соответствующего сопротивления.

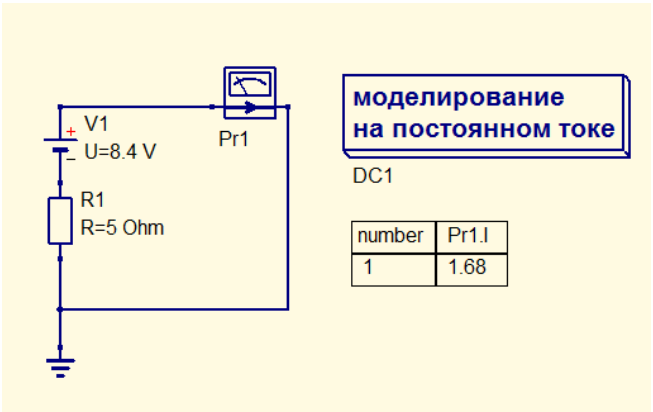


Рис. 6.3. Источник постоянного напряжения, «как настоящий»

Источник постоянного тока, не источник напряжения постоянного тока, имеет бесконечно большое внутреннее сопротивление, а его единственный параметр – это ток. Видоизменим предыдущую схему.

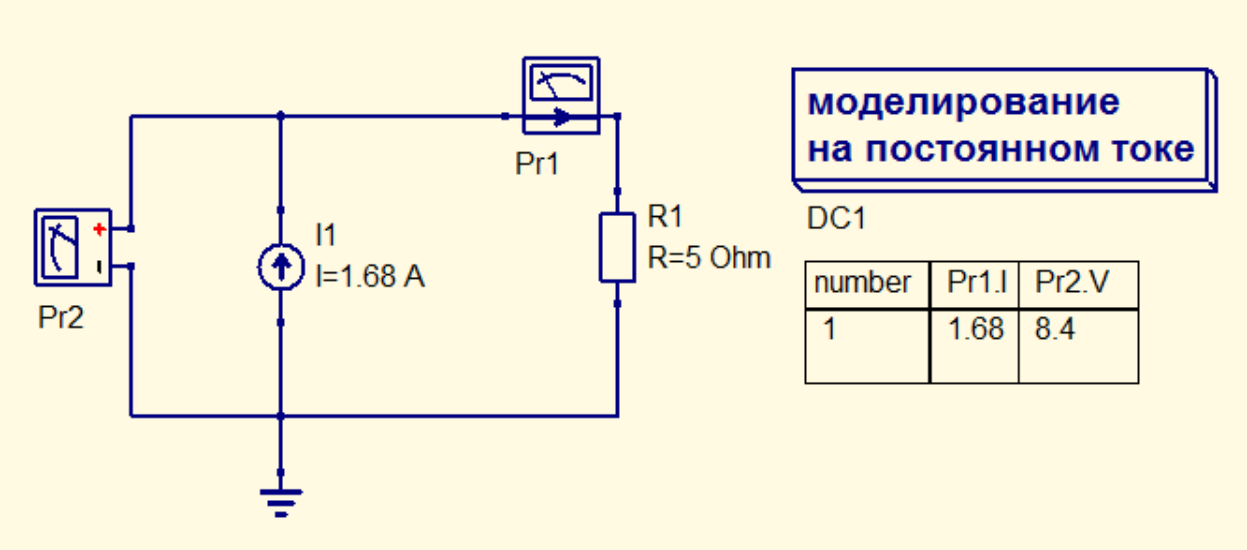
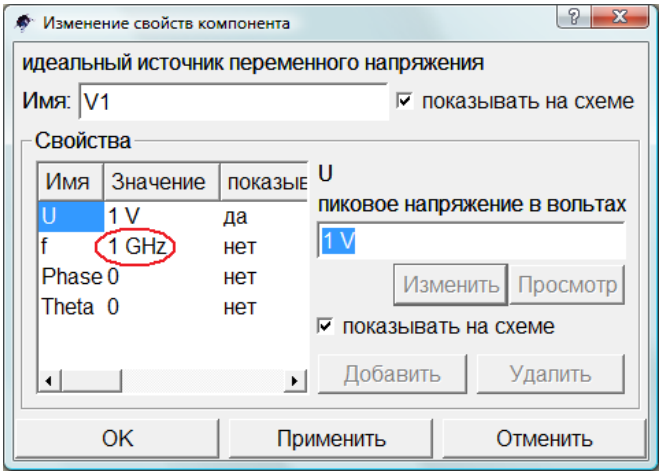


Рис. 6.4. Источник постоянного тока

Измеритель тока Pr1 показывает заданный в настройках источника ток 1.68 А, а измеритель напряжения Pr2 показывает напряжение в цепи 8.4 В. Схожесть показаний измерителей лишний раз показывает, как важно, особенно начинающим изучать электротехнику, различать два источника – источник напряжения и источник тока. Идеальный источник напряжения поддерживает заданное напряжение вне зависимости от сопротивления нагрузки. Другими словами, вне зависимости от потребляемого тока. Идеальный источник тока поддерживает заданный ток вне зависимости от сопротивления нагрузки. Другими словами, его напряжение станет таким, чтобы ток в цепи был равен заданному. Если в последней схеме задать ток, скажем, в 10 А, то напряжение пропорционально изменится.

Источник напряжения используется в программе Qucs для питания любых электрических цепей, требующих питающего напряжения. Если требуется несколько источников, как при двухполярном питании микросхем, можно использовать несколько. Источник тока чаще используют тогда, когда нужно задать фиксированный ток, например ток смещения на входе идеального операционного усилителя.

Источник напряжения переменного тока можно использовать при исследовании устройств, подключаемых к сети 220 В, 50 Гц. Но чаще он используется, как генератор синусоидального напряжения. При этом не следует забывать, что по умолчанию частота «генератора» 1 ГГц.



Прежде, чем включить моделирование, следует изменить частоту, если только частота 1 ГГц не рабочая частота.

И не следует забывать, что напряжение задается амплитудное (или пиковое).

Можно изменить значение фазы, что важно для исследования трехфазовых цепей.

Рис. 6.5. Диалоговое окно параметров источника переменного напряжения

Параметр Theta – коэффициент затухания.

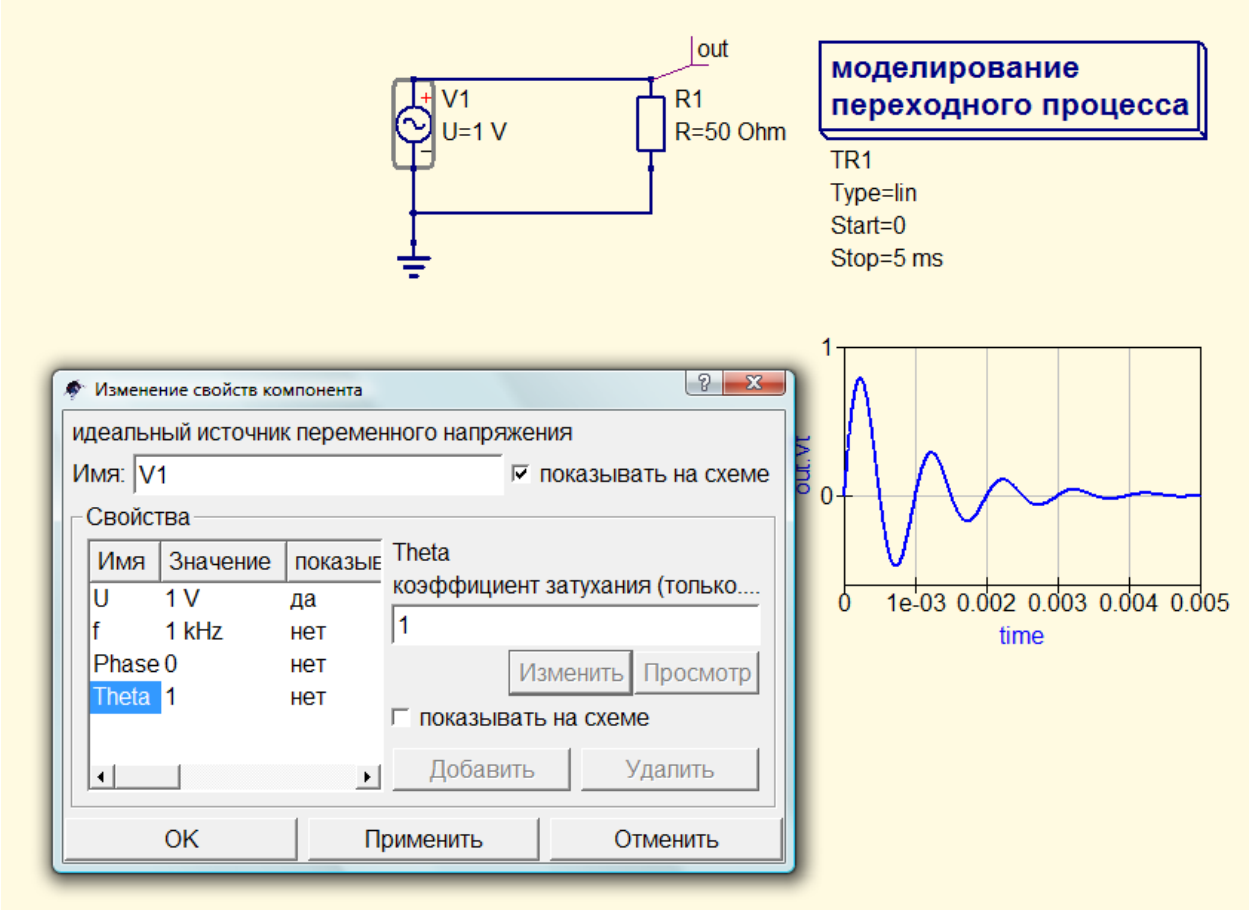


Рис. 6.6. Источник переменного напряжения с коэффициентом затухания единица

Источник переменного тока, как и в случае с постоянным током, чаще применим тогда, когда нужен фиксированный переменный ток, как в эквивалентных схемах или схемах замещения.

Источник питания (переменного тока) предназначен к питанию разного рода фильтров при их разработке, может использоваться при испытании линий.

Источники импульсного напряжения и тока – пара, дающая импульс прямоугольного напряжения первый или тока второй. Параметры, задаваемые в диалоговом окне свойств достаточно очевидны, но можно сделать одно замечание. Иногда удобно использовать импульс треугольной формы. Чтобы получить его из прямоугольного достаточно изменить времена переднего и заднего фронта прямоугольного импульса.

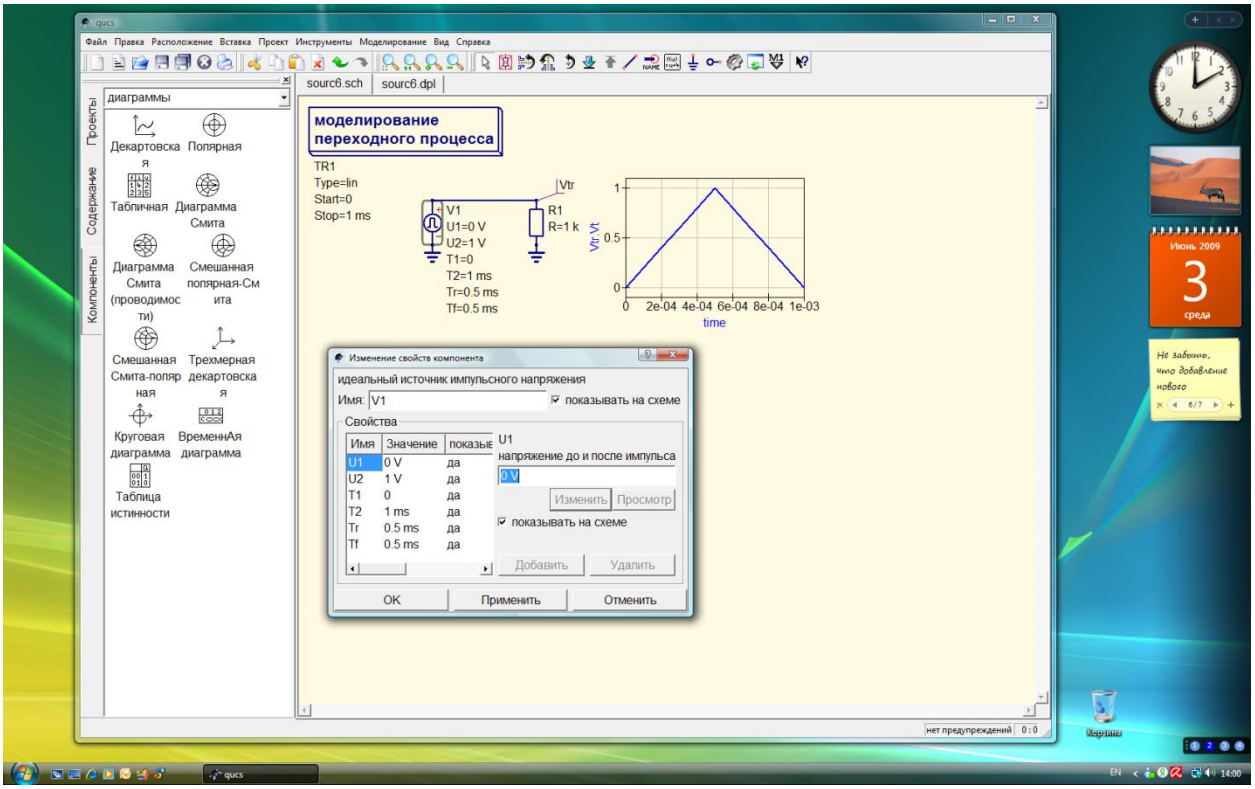


Рис. 6.7. Настройки для получения треугольного импульса

Источник шумового напряжения и **Источник шумового тока** образуют пару, позволяющую оценить работоспособность разрабатываемого устройства в условиях, приближенных к реальным. Шумы в реальных устройствах имеют разную природу – это и тепловой шум, и дробный, и наводки. Диалоговые окна источников позволяют подобрать нужные параметры шума.

К этой же подгруппе источников можно отнести источники коррелированных шумов.

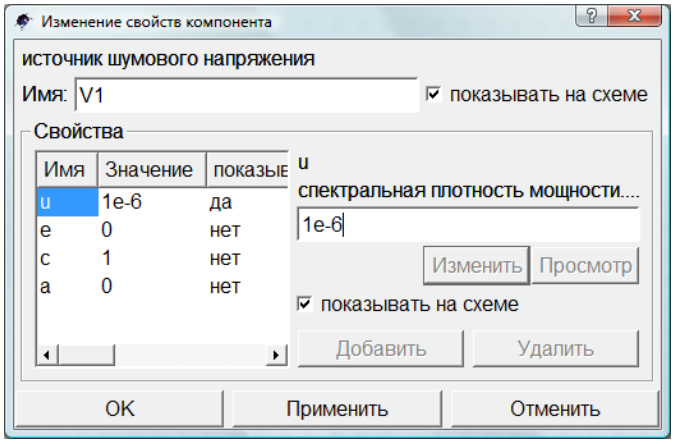


Рис. 6.8. Диалоговое окно настройки параметров шума

Несколько источников: **Источник тока, управляемый напряжением**, **Источник тока, управляемый током**, **Источник напряжения, управляемый напряжением** и **Источник напряжения, управляемый током**, – все они удобны при создании моделей устройств, при рассмотрении принципов работы устройств или в тех случаях, когда нужно проверить какие-то параметры устройства.

Источники напряжения и тока прямоугольной формы можно рассматривать как генераторы прямоугольного напряжения и тока. Настраивая времена фронтов их тоже можно превратить в генераторы треугольных или пилообразных импульсов, как и источники импульсного напряжения и тока.

Следующие два источника: **Источник с АМ-модуляцией** и **Источник с импульсной модуляцией**, – показывают примеры амплитудной и фазовой модуляции, и могут найти применение при работе со схемами приемников.

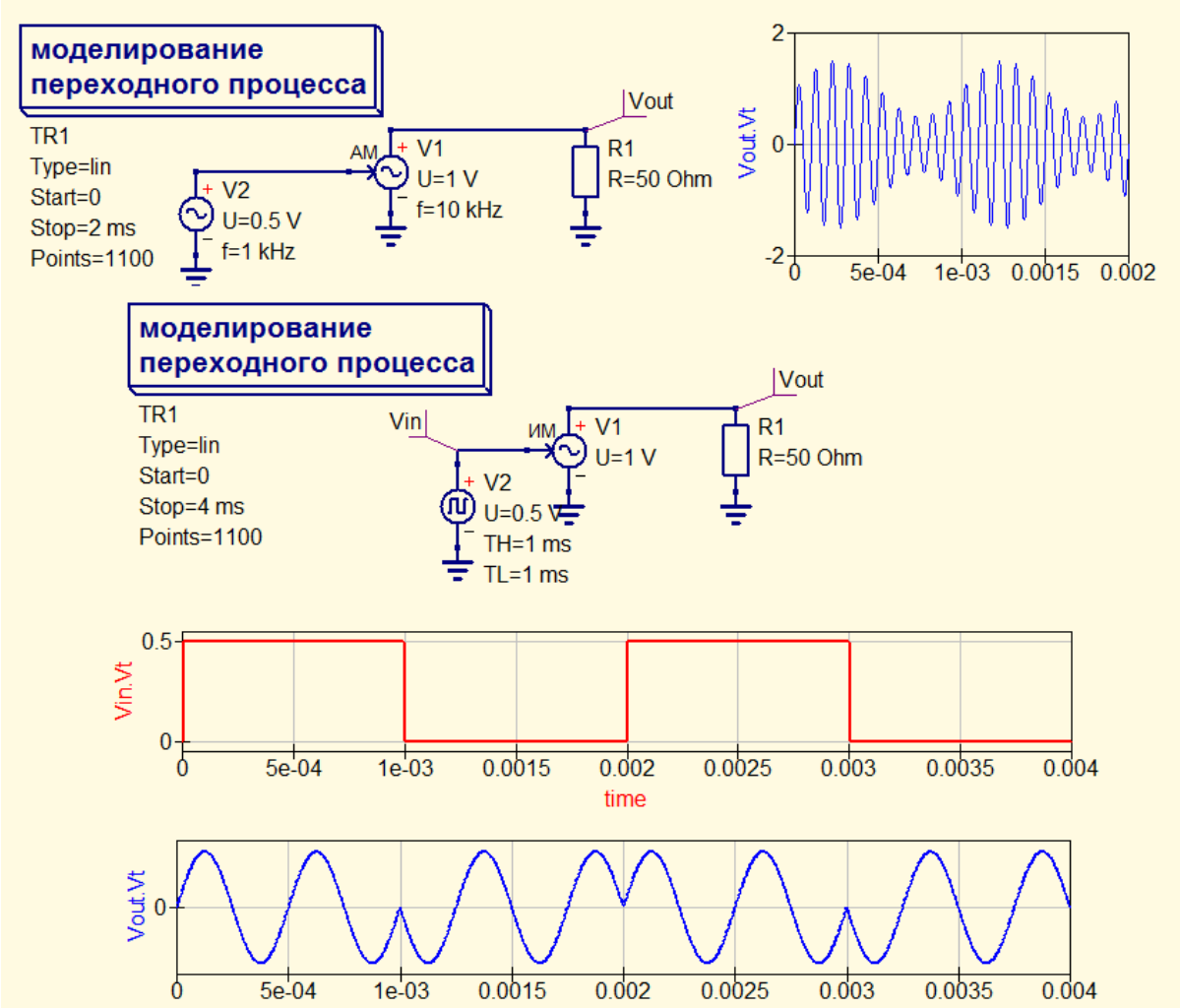
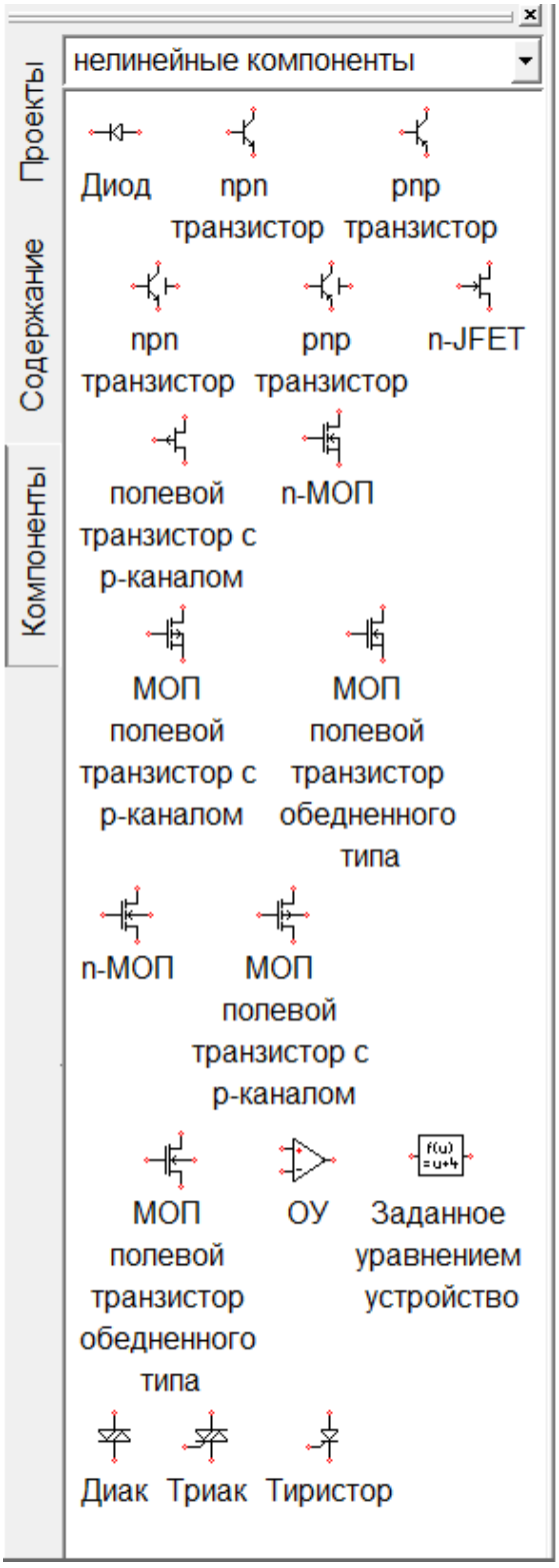


Рис. 6.9. Примеры работы источников модулированных сигналов

При необходимости вы можете создать собственные источники напряжения и тока, используя такие компоненты, как **источники на основе файла**.

Нелинейные компоненты

Широкое применение микросхем в устройствах электроники подчас маскирует значение таких элементов схемы, как транзисторы или диоды. Но пока почти ни одно устройство не может обходиться без них. Группа **нелинейные компоненты** предоставляет большой выбор этих компонентов электрических цепей.



Особенно важно использовать нелинейные компоненты при изучении электрических цепей и сигналов. Именно нелинейные компоненты являются активными элементами любой схемы.

Все компоненты этой группы – это обезличенные или близкие к идеальным прототипы реальных конструкций.

Вместе с тем большое количество параметров, а их достаточно много даже для диода, позволяют вам при необходимости превратить прототип в нужный вам элемент.

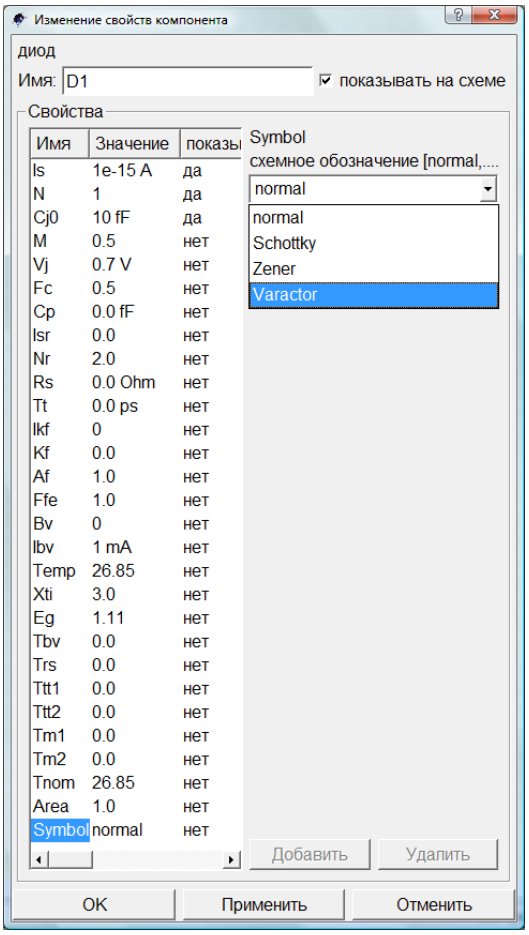


Рис. 7.1. Группа нелинейных компонентов и диалоговое окно свойств диода

Диод и **транзисторы n-p-n** и **p-n-p** присутствуют явно или скрыто почти в каждом электронном изделии. Поэтому легче выбрать схемы, где они отсутствовали бы, например, ламповые, чем привести все примеры их использования.

Между тем, выше уже упоминалось, что, меняя параметры, скажем, транзистора, его поведение можно приблизить к поведению реального транзистора. Вот АЧХ однокаскадного усилителя с n-p-n транзистором.

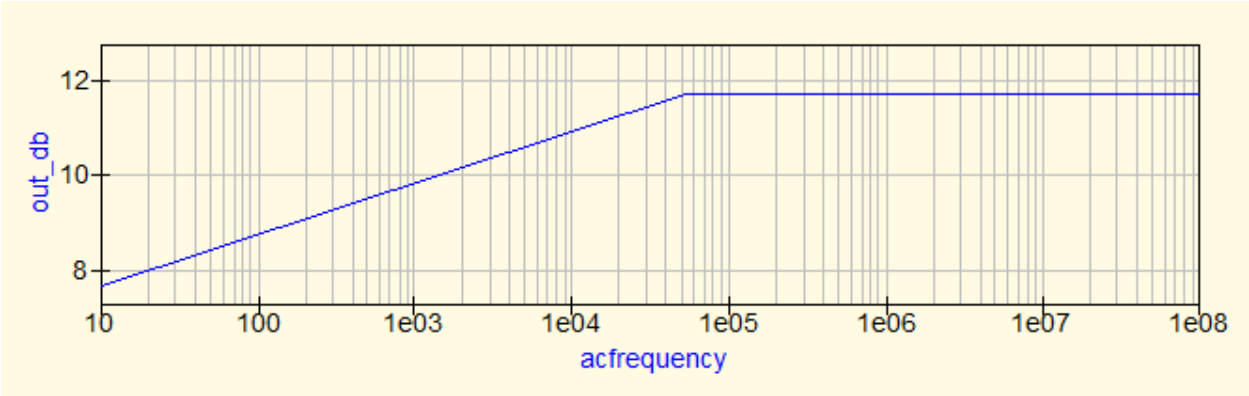


Рис. 7.3. Аплитудно-частотная характеристика каскада усиления

Изменим значение емкости коллекторного перехода с 0 на 10 пФ.

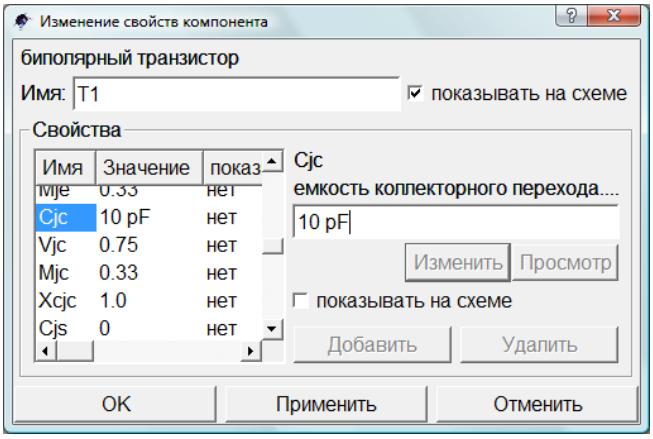


Рис. 7.4. Диалоговое окно свойств транзистора

Посмотрим, как изменится АЧХ.

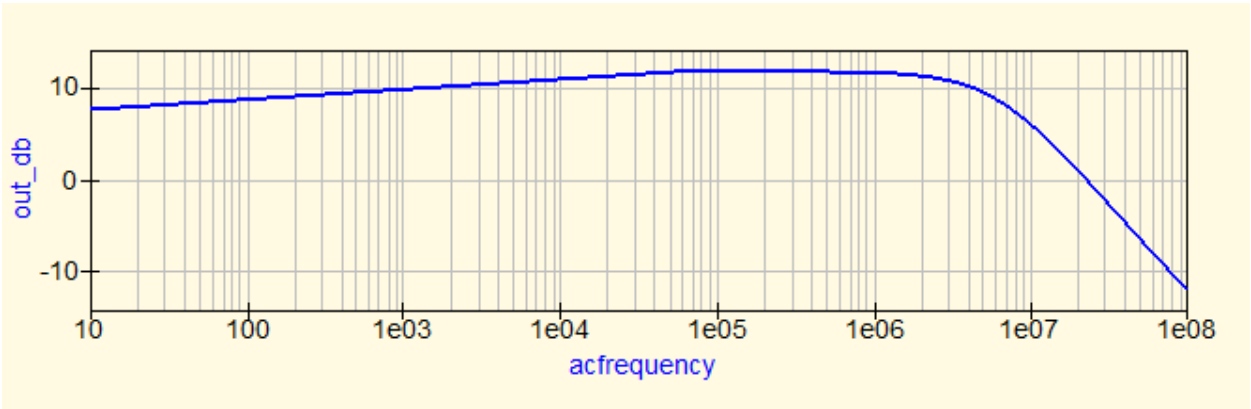


Рис. 7.5. АЧХ усилительного каскада после изменения свойств транзистора

Полевые транзисторы во включенном состоянии очень похожи на резисторы, величина которых будет зависеть от приложенного напряжения к затвору транзистора.

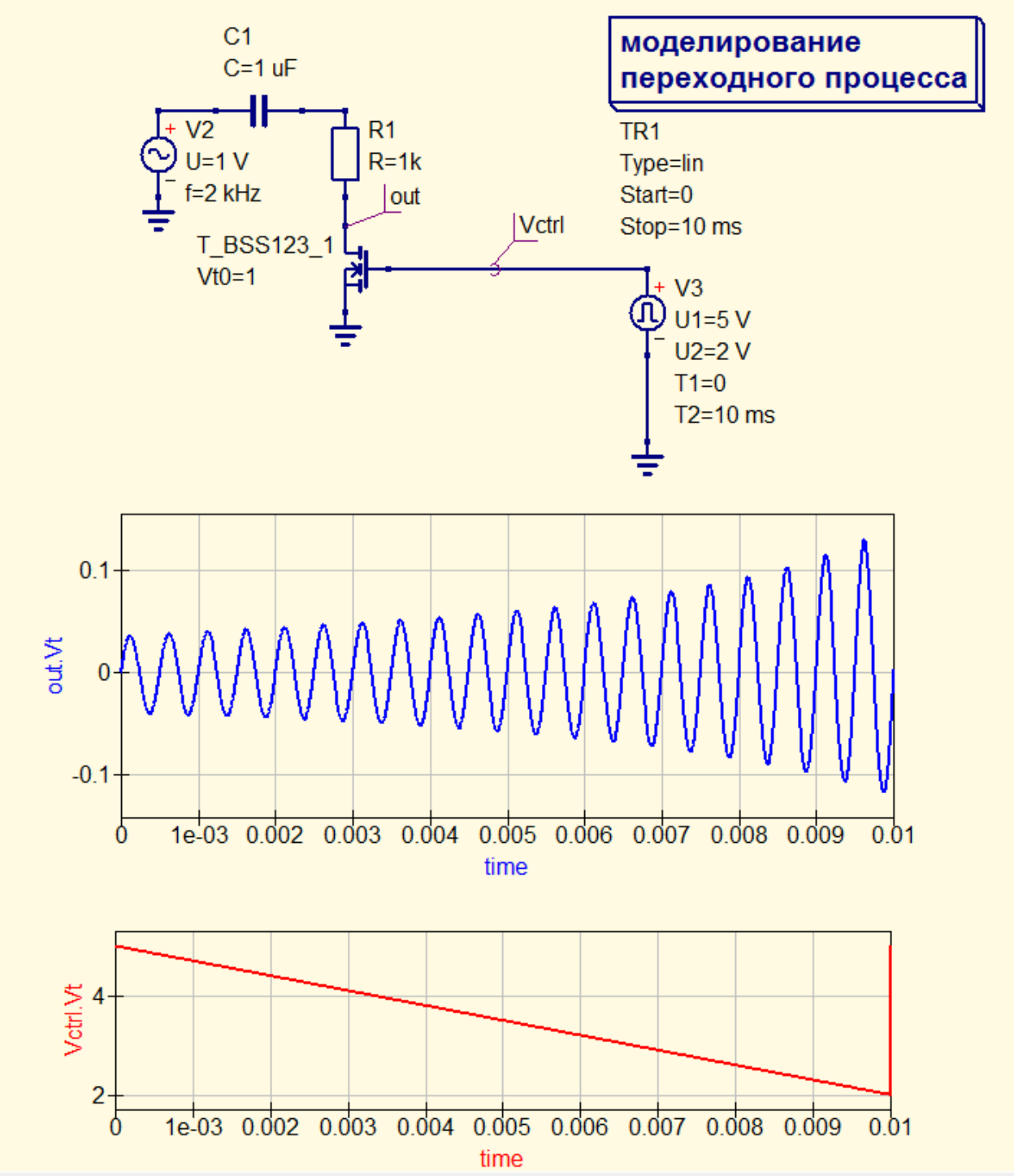


Рис. 7.6. Опыт с полевым транзистором

Свойства полевых транзисторов, высокое входное сопротивление и управляемое проходное сопротивление, находят широкое применение в схемотехнике. А совершенствование технологии изготовления полупроводниковых приборов приводит к тому, что полевые транзисторы находят новое применение, очень сильно изменяя параметры электронных устройств, как это происходит в силовой электронике или цифровых устройствах.

Кроме большого количества модификаций транзисторов в группу входят многослойные полупроводниковые приборы – динистор, тиристор, триак. Последние два во многом схожи – это управляемые приборы, но триак позволяет включать его в цепь переменного напряжения.

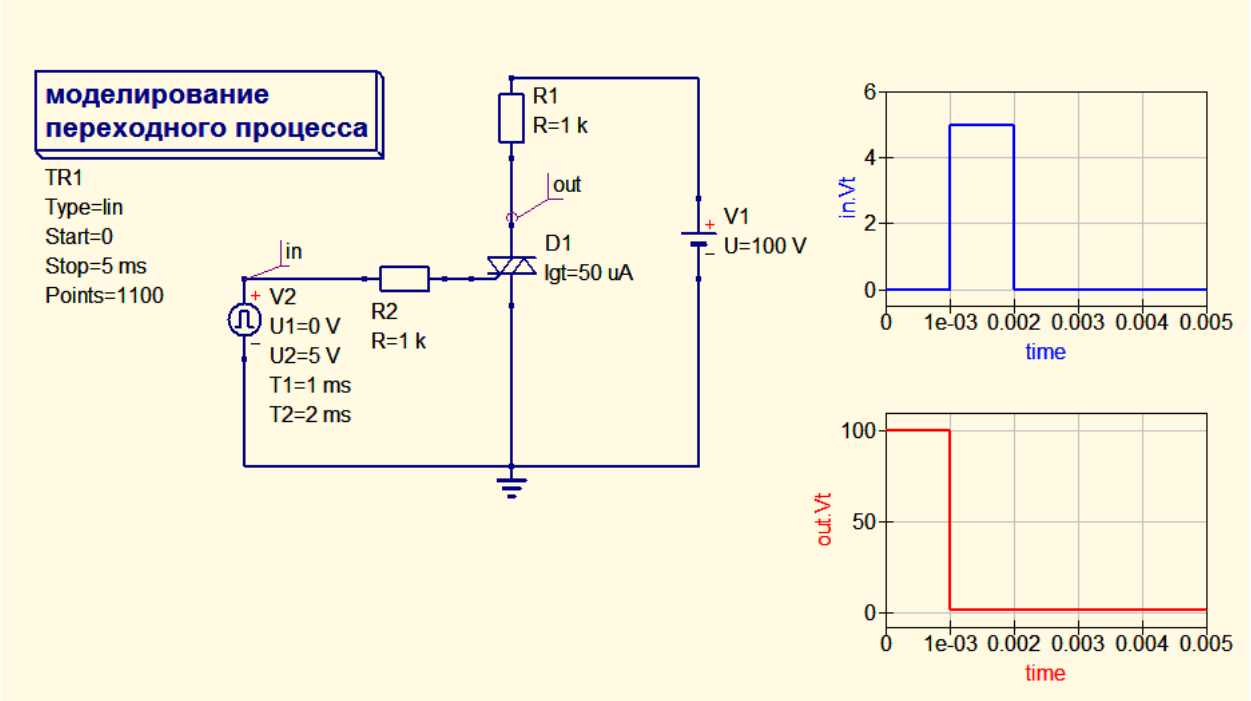
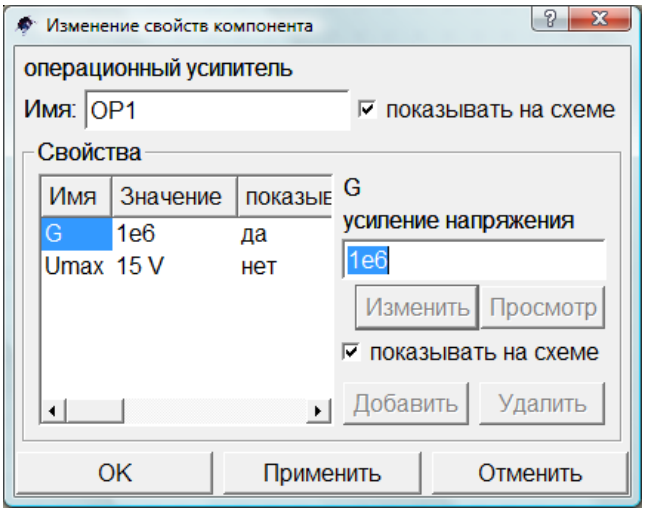


Рис. 7.7. Триак в цепи постоянного напряжения

Если в этой схеме поменять полярность источника V1 на обратную, то триак опять включится с приходом управляющего импульса, а выключится он тогда, когда напряжение V1 будет снято.

Операционный усилитель в составе группы нелинейных компонентов – это идеализированный, обобщенный элемент, имеющий только два параметра в перечне свойств.



Во многих отношениях, в области применения операционных усилителей, этого более чем достаточно.

Рис. 7.8. Диалоговое окно свойств операционного усилителя

Виды моделирования и диаграммы

Многие программы, схожего с Qucs назначения, различают компоненты и виды анализа. В Qucs все виды анализа находятся на закладке Компоненты панели навигации. В Qucs все виды моделирования (анализа, симуляции) добавляются на схему, как компоненты схемы. Это позволяет проще обращаться с параметрами моделирования, как свойствами компонента, и делает очевидными намерения по работе со схемой. Все компоненты, так или иначе, связаны между собой, если они присутствуют на схеме, но между видами моделирования и диаграммами, пожалуй, самая тесная связь.

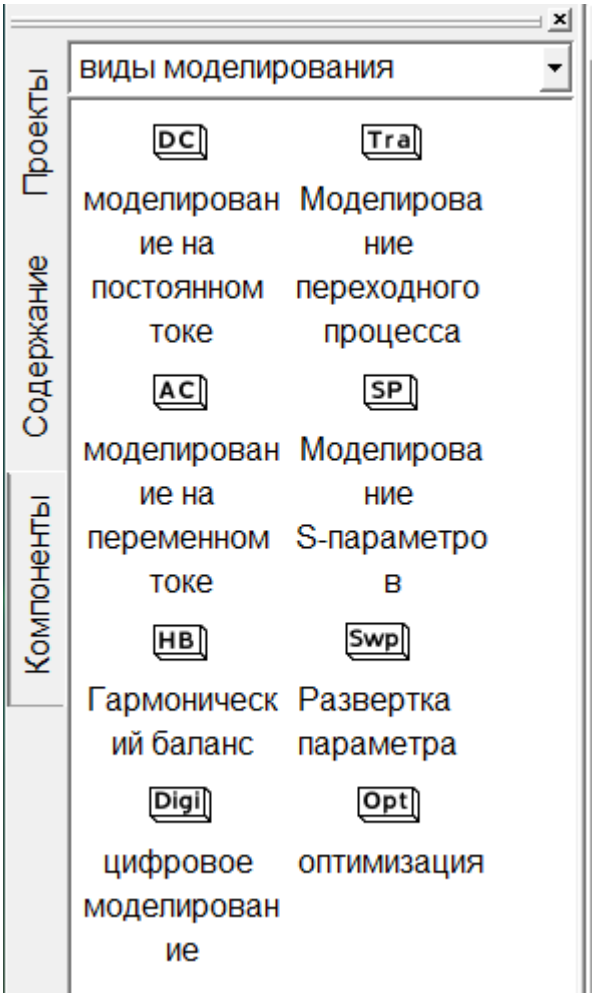
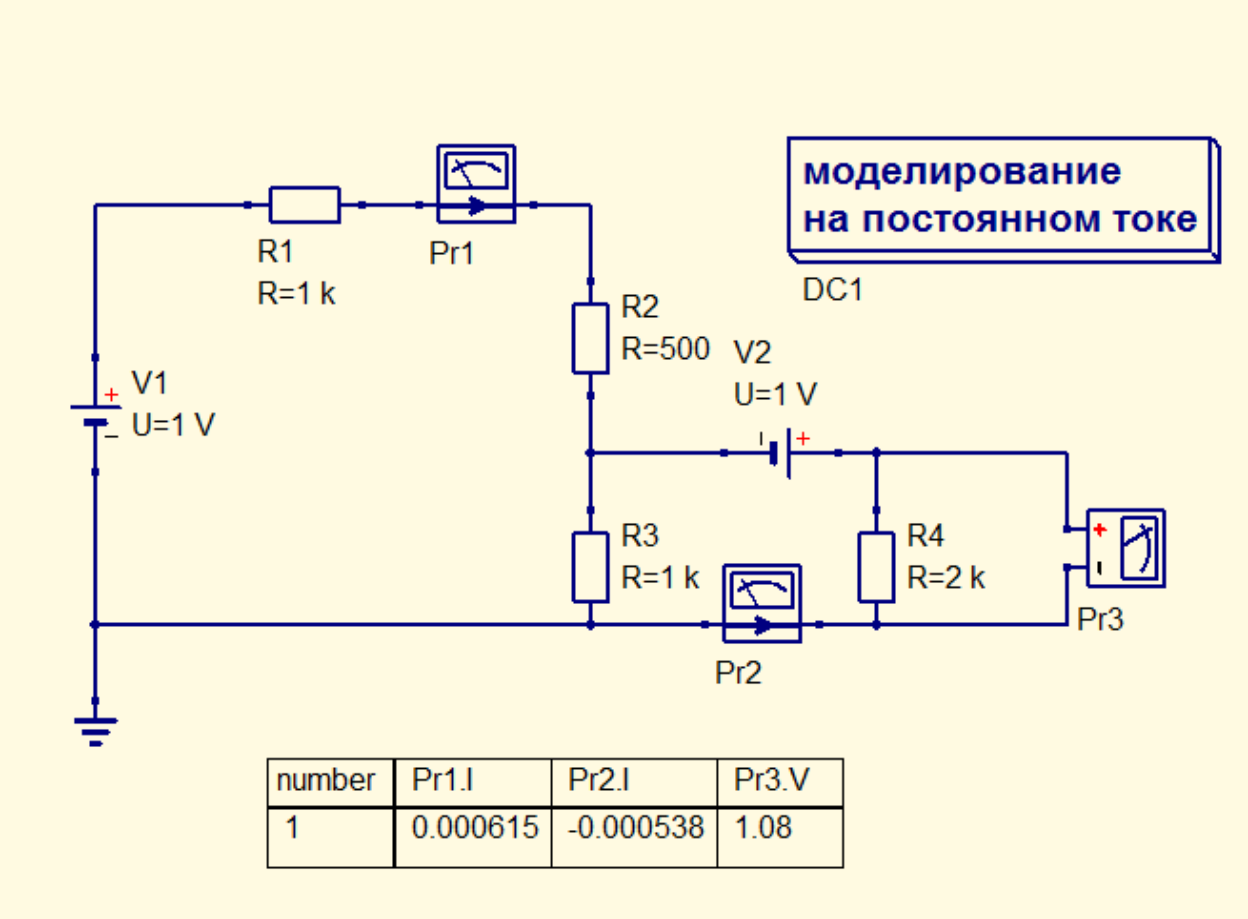


Рис. 7.10. Две группы компонентов

Моделирование на постоянном токе, первый вид моделирования в группе, вполне отвечает **Табличной** форме диаграммы. А моделирование переходного процесса хорошо отобразится на **Декартовской** диаграмме.

Особенно это относится к **цифровому моделированию**: **Таблица истинности** и **Временная диаграмма**, – вот то, что позволит решить основные вопросы работы цифровой схемы. **Цифровое моделирование**, как компонент, входит в группу **цифровых компонентов**.

Моделирование, об этом уже говорилось, основа работы программы. Именно ради этого она и создавалась. **Моделирование на постоянном токе**, как важная составляющая при работе с аналоговыми устройствами, может входить обязательным компонентом, но может быть опущено (как добавляемый компонент) в других случаях. Вместе с тем, этот вид моделирования важен сам по себе. Расчеты схемы на постоянном токе достаточно трудоемки. Если можно измерить параметры схемы (в основном, положим, токи и напряжения), то это существенно ускорит работу со схемой.



number	Pr1.I	Pr2.I	Pr3.V
1	0.000615	-0.000538	1.08

Рис. 7.11. Измерение токов и напряжения на постоянном токе

Достаточно сравнить время, потраченное на расчет с применением калькулятора, со временем моделирования такой, в общем-то, простой схемы, чтобы убедиться в преимуществах программного подхода. В таблице измерение тока прибором Pr2 показывает, что прибор включен «в обратную сторону» – ток получился отрицательным. В этом эксперименте можно было бы применить в качестве диаграммы Декартовскую, но что она может отобразить? Основной аргумент, время, в данном случае отсутствует. Табличная форма диаграммы – самый удобный способ отобразить ответ на вопрос о токах и напряжении в цепи, который мы задали, поместив три измерительных прибора: два амперметра Pr1 и Pr2 и вольтметр Pr3.

Другое дело, когда мы используем **моделирование переходного процесса**. Само слово процесс подразумевает, что время обязательно присутствует в нашем эксперименте. Тогда-то **Декартовской** диаграмме самое применение.

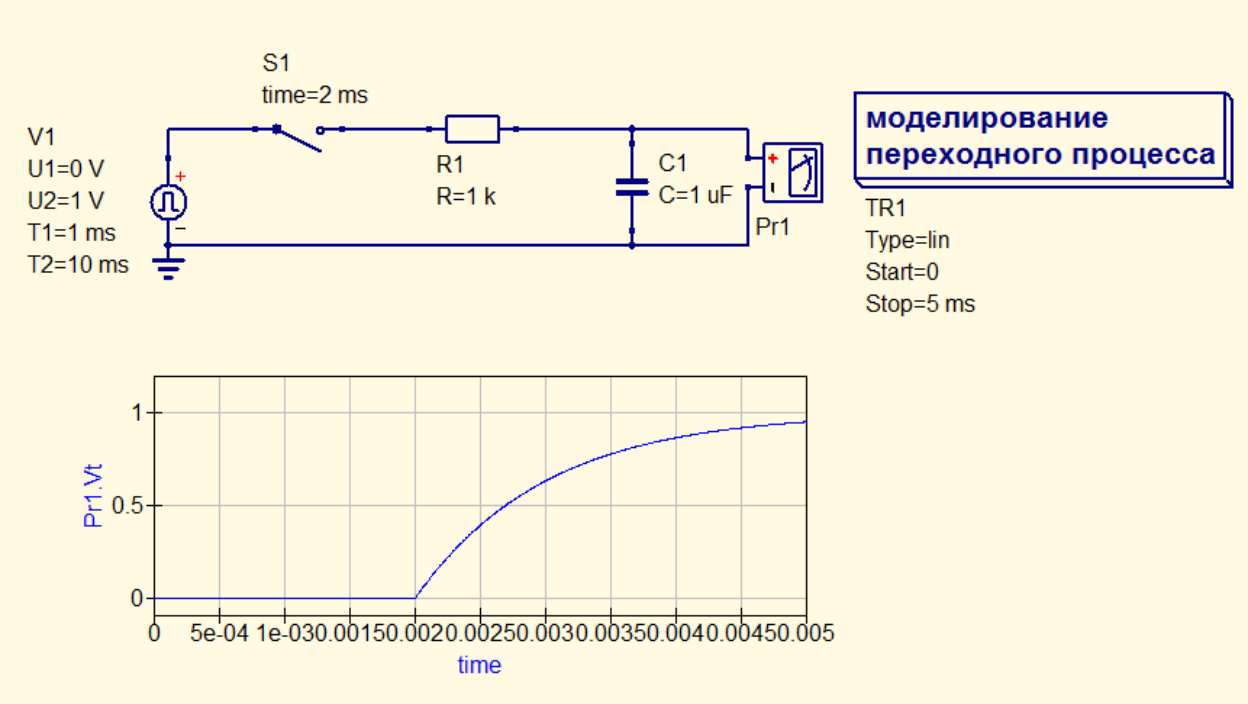


Рис. 7.12. Исследование переходного процесса

Разные виды моделирования могут дополнять друг друга, как моделирование на постоянном или переменном токе дополняет развертку параметра; а могут мешать друг другу, в этом случае можно применить команду деактивирования компонента (вида моделирования).

Группы виды моделирования и диаграммы включают много компонентов, но с ними лучше знакомиться не по формальному описанию, а по применению к разным электрическим цепям.

Кроме того, в состав компонентов входит еще ряд групп, о которых пока мало было рассказано.

Попробуем прервать плавный пересказ руководства к программе с тем, чтобы посмотреть на конкретных примерах, как можно распорядиться компонентами программы Qucs для своих целей.

Глава 2. Моделирование

Моделирование на постоянном токе

Несколько примеров моделирования на постоянном токе приводилось выше. Понятно, что любое сочетание компонентов, проводящих постоянный электрический ток, с любым сочетанием источников постоянного тока, объединенные в цепь, становятся предметом этого вида моделирования.

Рассмотрим еще одну схему измерений.

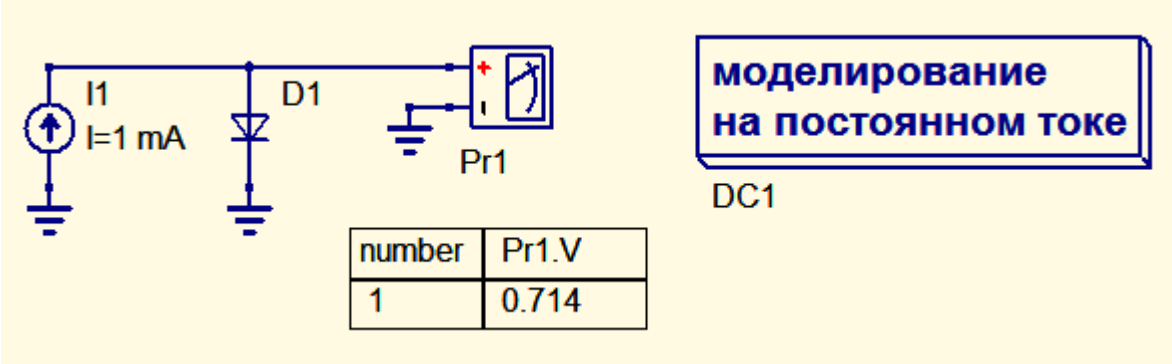
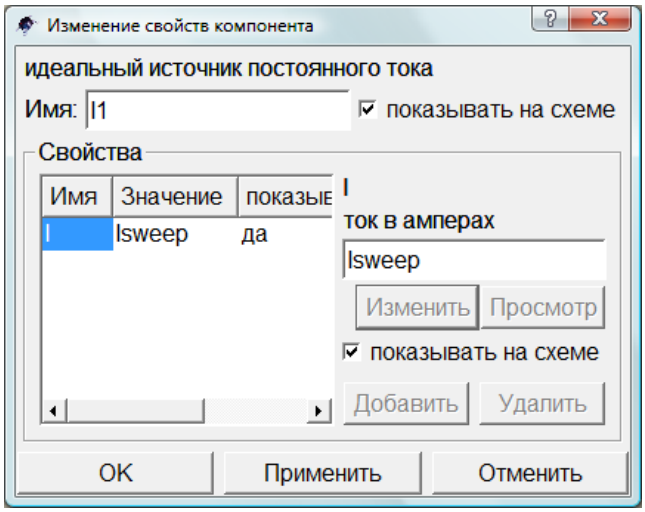


Рис. 8.1. Измерение тока и напряжения на диоде

В данном случае ток задан параметром источника тока, а напряжение измерено вольтметром Pr1. Чаще нас интересует не только значение напряжения при одном токе через диод, а зависимость между напряжением и током. В этом случае **моделирование на постоянном токе** следует дополнить **Разверткой параметра**. Этот компонент переносится из группы **виды моделирования** закладки **Компоненты** панели навигации на схему. Для развертки параметра его следует задать, что достигается в диалоговом окне свойств источника I1.



По умолчанию источник тока появляется с заданным параметром – током. Если заменить этот параметр переменной, в данном случае *Isweep*, то эту переменную можно использовать, как параметр развертки.

Рис. 8.2. Использование источника постоянного тока в качестве «свип-генератора»

Теперь можно настроить компонент Развертка параметра, открыв двойным щелчком по нему диалоговое окно.

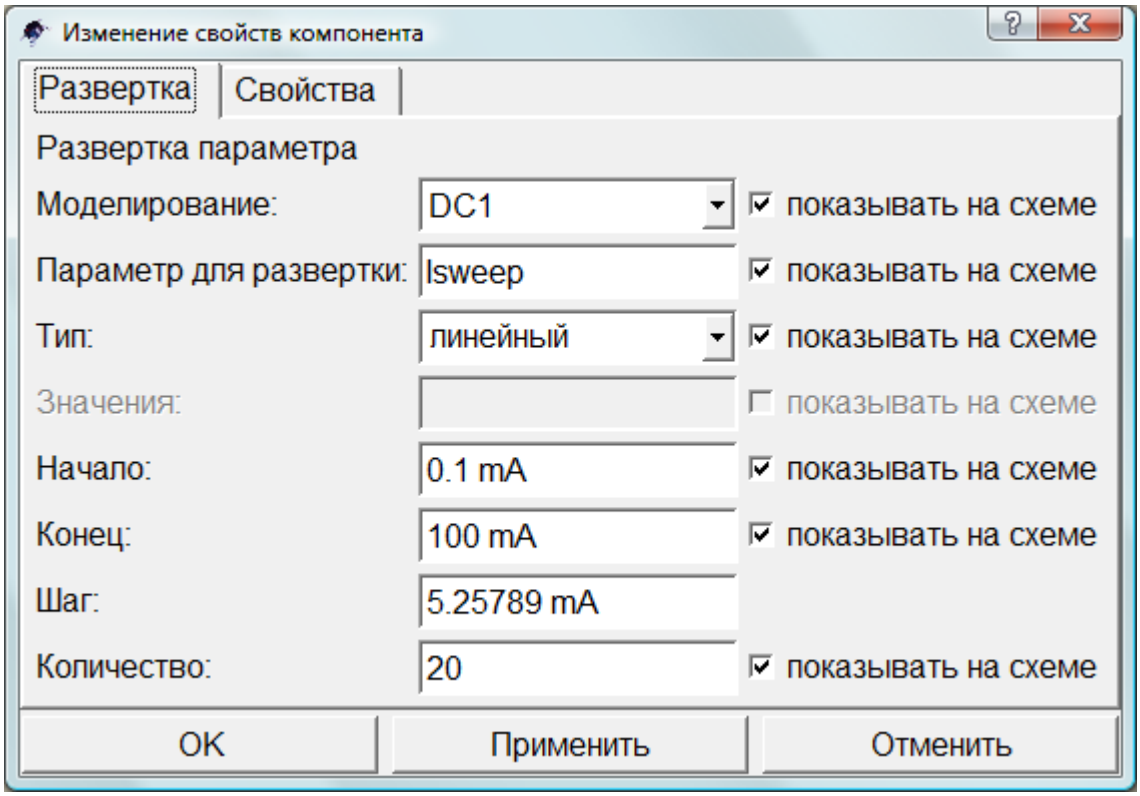


Рис. 8.3. Диалоговое окно свойств **Развертки параметра**

Моделирование выбирается из выпадающего списка (кнопка справа от окна), параметр для развертки и такие параметры, как *Начало, Конец* и *Количество*, вводятся. В итоге схема и результат работы программы оказываются такими.

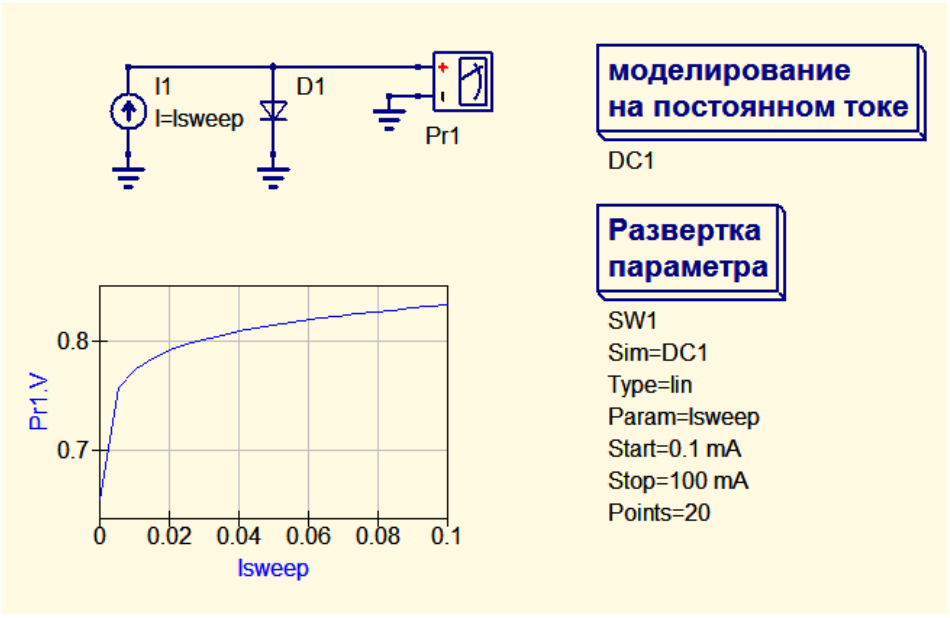


Рис. 8.4. Зависимость напряжения на диоде от тока через него

Диод использовался обобщенный, но его легко заменить конкретной моделью, скажем, 1N4001 из **Библиотеки компонентов** (**Инструменты** основного меню). В этом случае график выглядит несколько иначе.

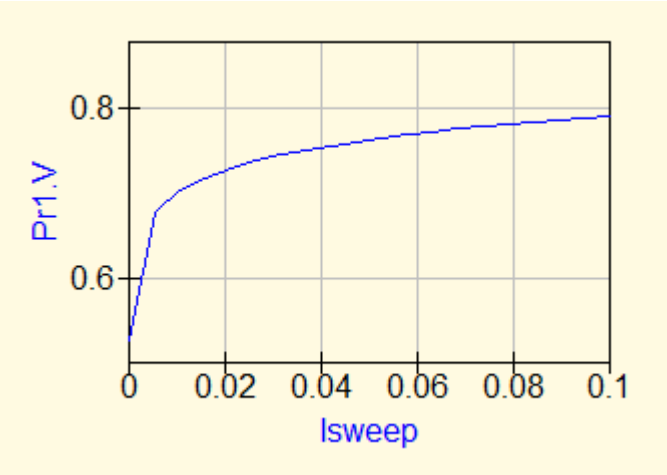
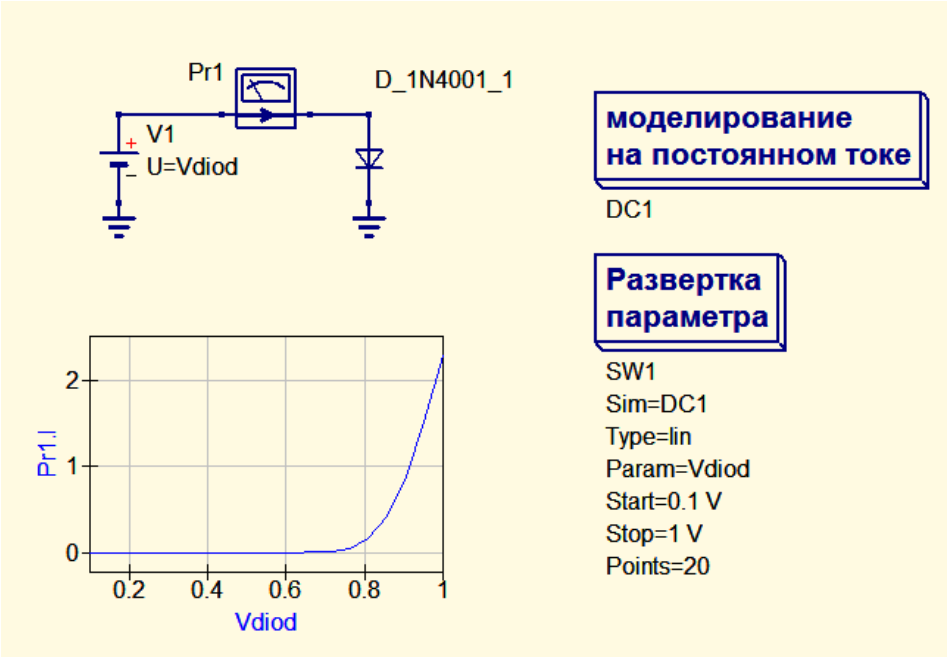


Рис. 8.5. Та же зависимость для модели диода 1N4001

Связь между током и напряжением для диода привычнее выглядит тогда, когда в качестве аргумента выступает напряжение. Что ж, можно провести и такое моделирование.



В данном случае в качестве переменной выбрана *Vdiod* (как напряжение V1).

Источник тока заменен источником напряжения. Pr1 – амперметр.

Рис. 8.6. Изменение схемы эксперимента с диодом

Еще более интересный эксперимент можно проделать с транзистором, получив его статическую выходную характеристику. Для схемы включения транзистора с общим эмиттером выходная характеристика – это зависимость тока коллектора от напряжения на коллекторе при фиксированном токе базы.

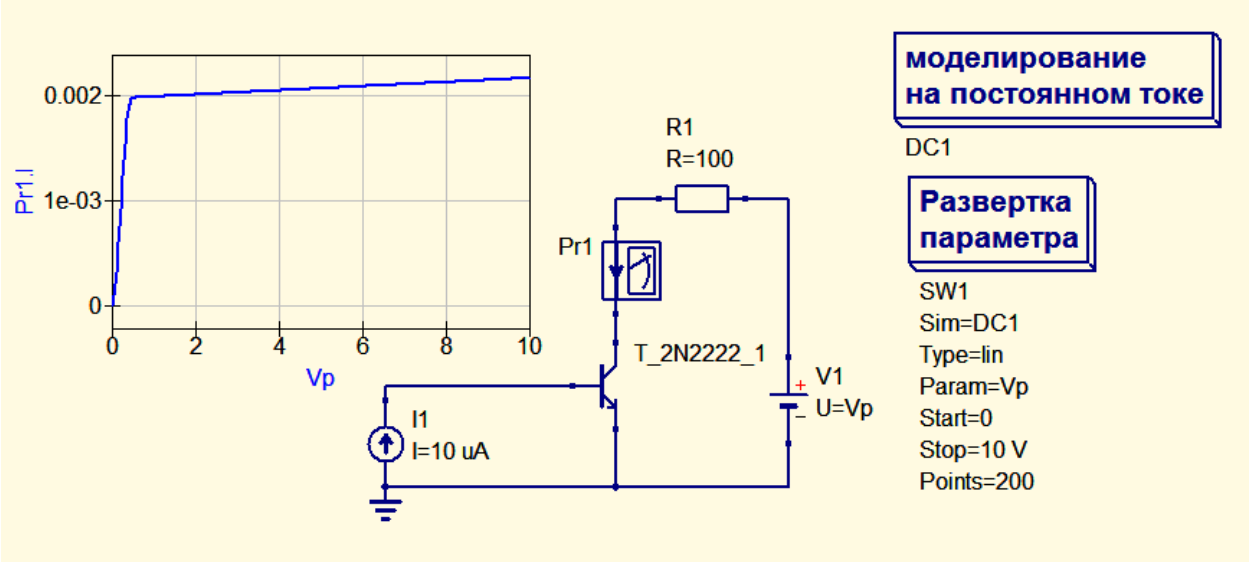


Рис. 8.7. Измерение выходной характеристики транзистора в схеме с ОЭ

Обычно работают с семейством таких характеристик, например, при расчете выходного каскада усилителя мощности, при разных токах базы. Небольшая модификация схемы позволяет получить эти характеристики.

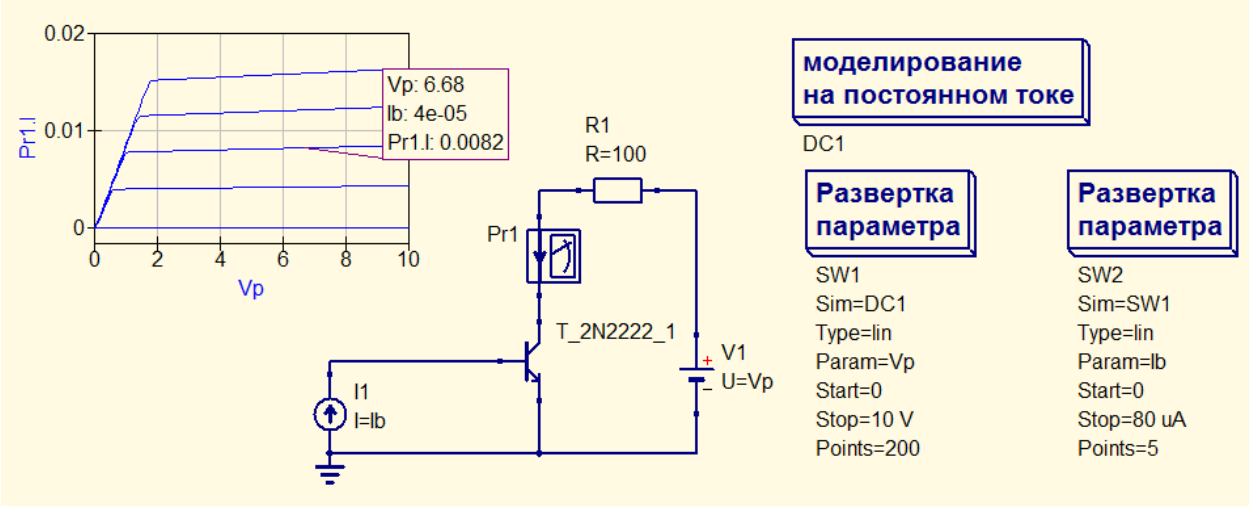


Рис. 8.8. Семейство выходных характеристик

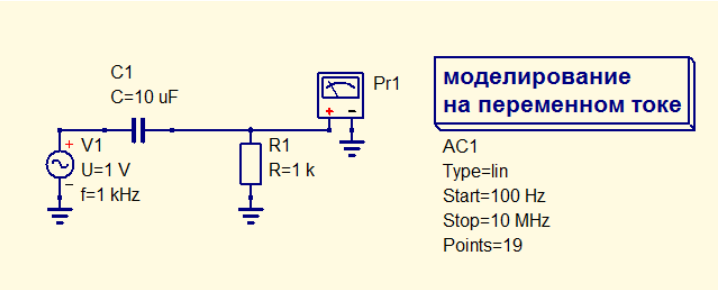
Достаточно добавить к предыдущей схеме еще одну развертку параметра, где в качестве параметра выступает источник тока базы I1. Вместо фиксированного тока (10 мкА в предыдущем случае) мы задаем переменную *Ib*, которая выступает в качестве параметра развертки для SW2. Параметр для развертки SW1 – это напряжение (переменная *Vp*). Маркер на одной из кривых показывает, что она построена при токе базы в 40 мкА. Маркеры можно установить на все кривые, чтобы знать, при каких токах базы получена эта кривая.

Моделирование на постоянном токе, дополненное **Разверткой параметра**, может дать много интересной информации о свойствах и компонентов, и электрических цепей.

Моделирование на переменном токе

Этот вид анализа позволяет получить амплитудно- и фазочастотные характеристики электрических цепей. Они важны, скажем, при исследовании фильтров, при создании усилителей с обратной связью, где позволяют определить параметры и устойчивость усилителя после введения обратной связи. Да и исследование усилителей без обратной связи не менее важная задача.

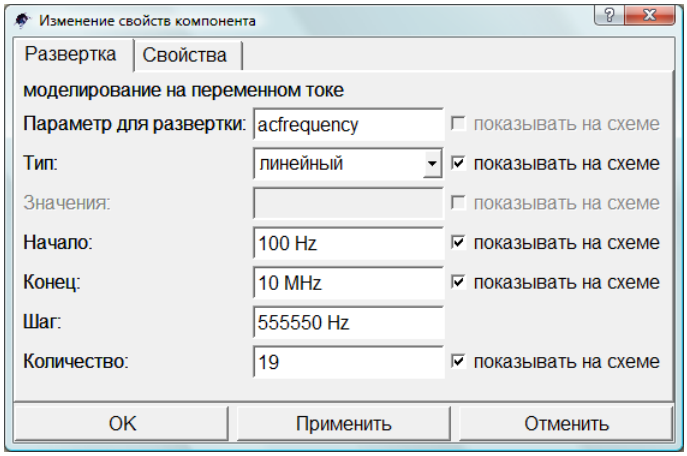
Самый простой опыт по работе с **моделированием на переменном токе** можно провести по схеме на рисунке ниже.



Электрическая цепь, состоящая из конденсатора и резистора, подключена к источнику переменного напряжения V1.

Параметры моделирования показаны на рисунке, а задаются эти параметры в диалоговом окне свойств этого вида моделирования.

Рис. 8.9. Анализ электрической цепи на переменном токе

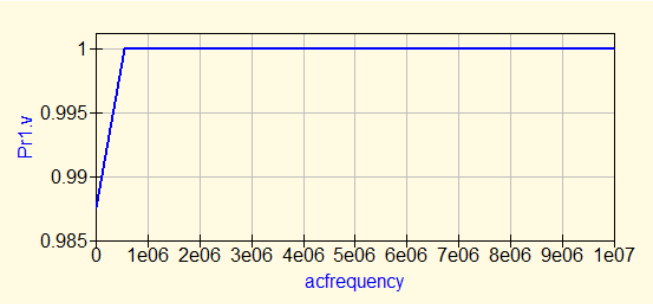


Одна из закладок, ***Развертка***, позволяет вам выбрать тип развертки, линейный или логарифмический, с помощью кнопки справа от окна. Позволяет задать начальное и конечное значение параметра и количество расчетных точек.

Вторая закладка, ***Свойства***, позволяет вам добавить к расчету шумы, если есть в этом необходимость.

Рис. 8.10. Диалоговое окно параметров моделирования на переменном токе

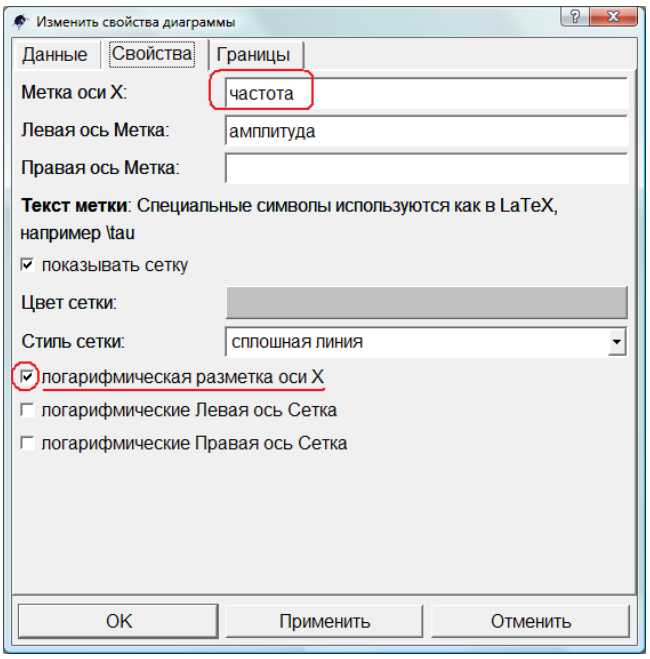
Если схему на предыдущем рисунке симулировать, то результат будет выглядеть так:



Ранее уже упоминалось, что более привычный вид этой кривой может быть получен, если в диалоге свойств диаграммы выбрать логарифмическое отображение по оси X.

Рис. 8.11. Результат симуляции электрической цепи на переменном токе

Для получения диаграммы использовался измеритель напряжения Pr1. Как видно, он может применяться и при работе с постоянным током, и при работе с переменным током. Но вместо него можно поставить метку проводника. Заменим измеритель меткой, и изменим свойства диаграммы (опция на рисунке ниже).



Помимо вида разметки оси X можно задать метку оси X, обозначить левую и правую оси.

На диаграмме можно показывать, а можно отключить показ сетки. Можно выбрать цвет сетки и стиль сетки.

Дополнительные опции позволяют выполнить логарифмическую разметку сетки по осям.

Рис. 8.12. Изменение свойств диаграммы в ее диалоговом окне, вызываемом двойным щелчком по диаграмме

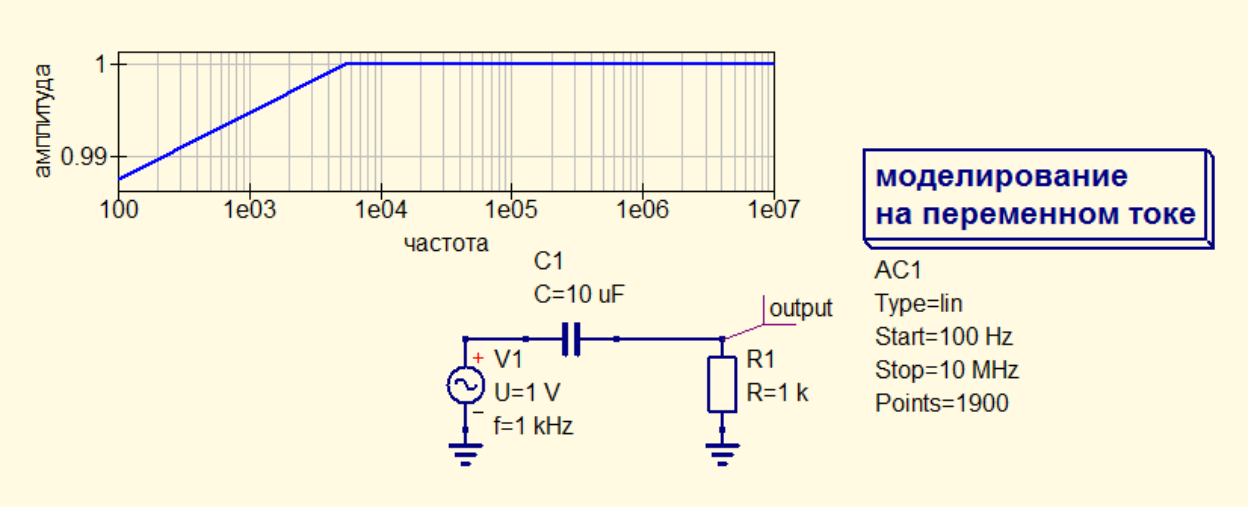


Рис. 8.13. Видоизменение схемы для моделирования на переменном токе

А, если добавить уравнение, то можно получить и фазочастотную характеристику цепи.

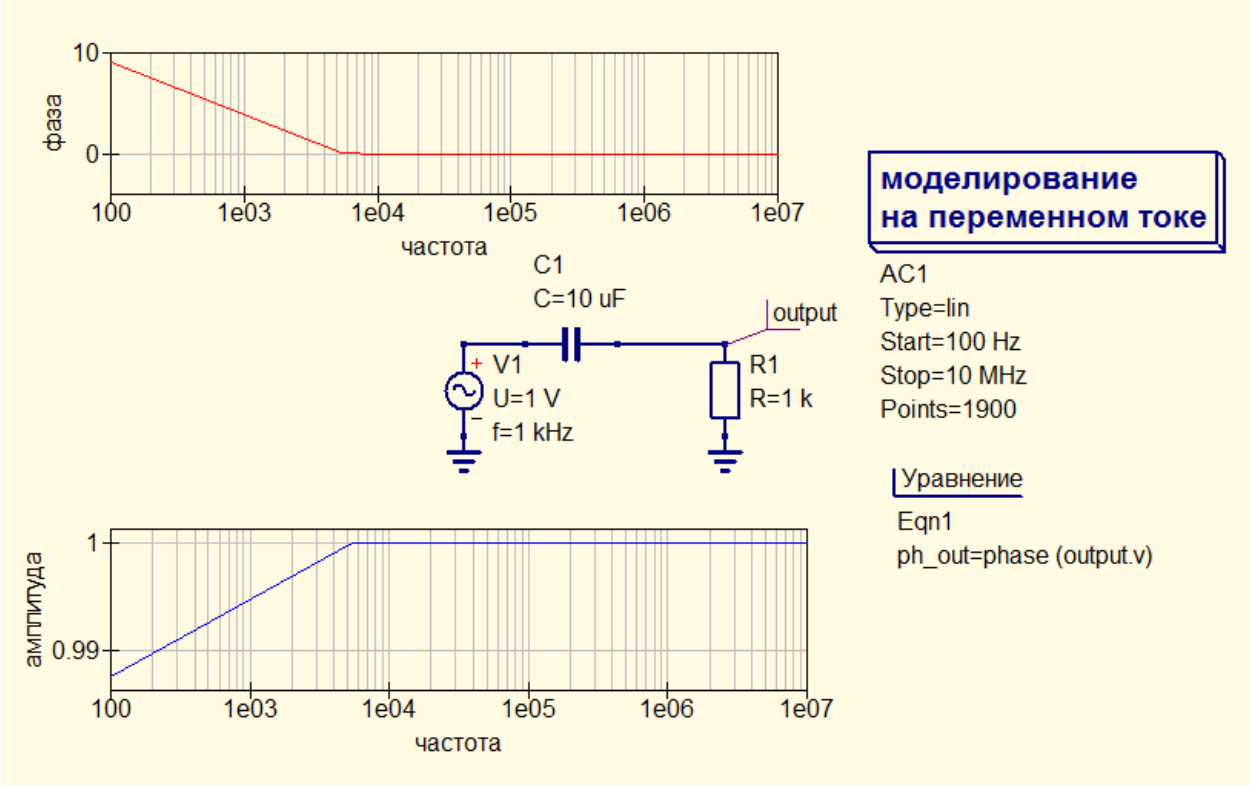
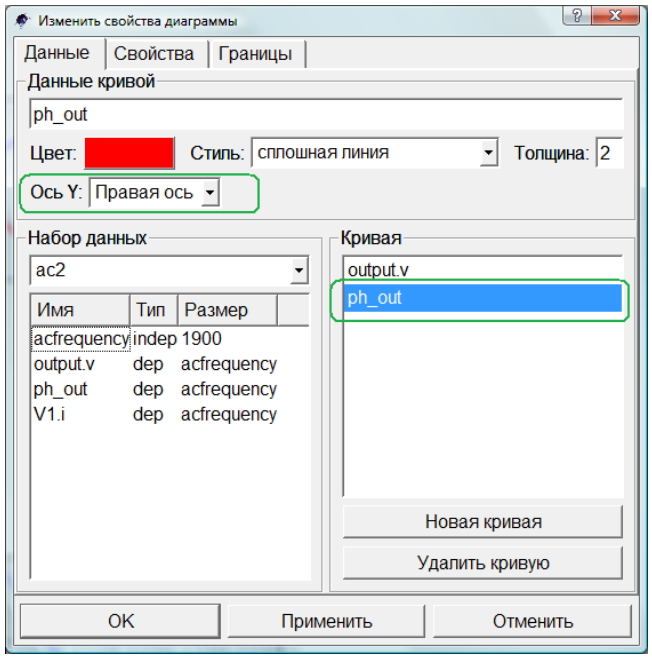


Рис. 8.14. Получение фазочастотной характеристики цепи на переменном токе

Обе характеристики, АЧХ и ФЧХ, можно совместить на одной диаграмме, но для получения более ясной картины следует выбрать для одной из кривых левую ось Y, для другой правую. Это можно сделать в диалоговом окне свойств диаграммы на закладке ***Данные***.



Для амплитуды выбрана левая ось, а для фазы правая ось Y. После добавления меток для осей получается такой вид кривых.

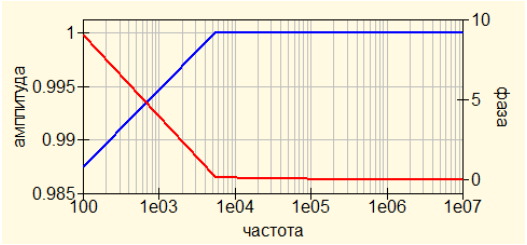


Рис. 8.16. Задание двух осей для отображения двух кривых

На закладке ***Границы*** диалогового окна свойств диаграммы можно вручную «прописать» границы осей, если вас интересует поведение функции в каком-то интервале. И, как всегда, вы можете использовать маркеры для получения уточненных данных.

И, как и при моделировании на постоянном токе, добавление уравнений и развертки параметра может приумножить возможности получения информации о свойствах электрической цепи.

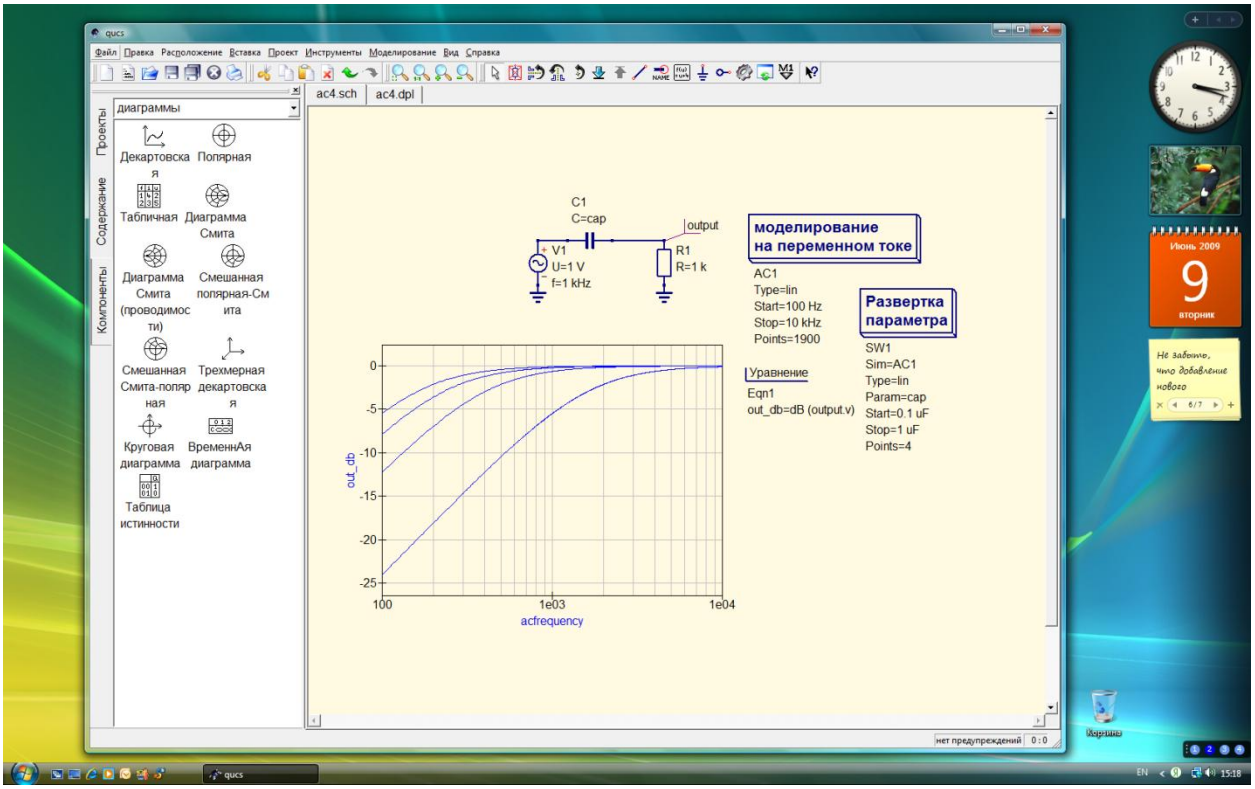
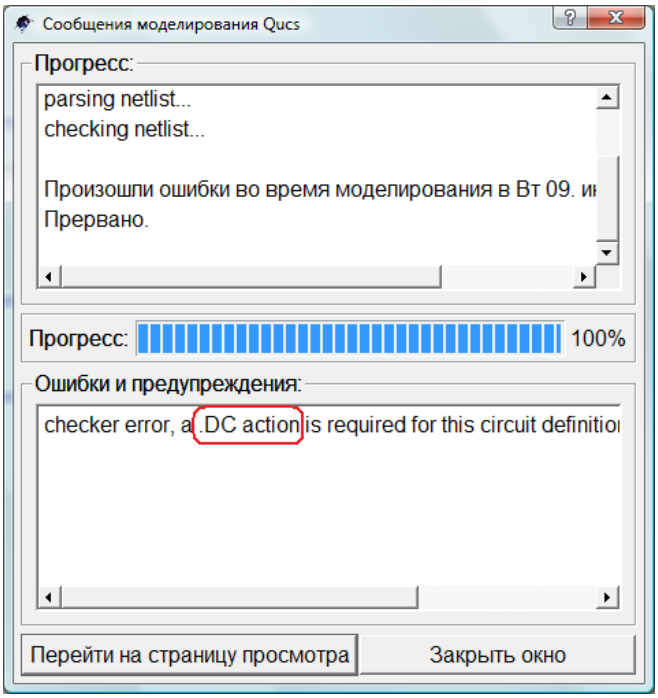


Рис. 8.17. Применение развертки параметра *cap* (изменяющейся емкости конденсатора)

Иногда **моделирование на переменном токе** требует добавления **моделирования на постоянном токе**. Об этом вы получите сообщение, запустив симуляцию.



В этом нет ничего удивительного – параметры схемы на переменном токе во многом определяются выбором рабочей точки на постоянном токе.

В этом случае достаточно добавить еще один компонент из группы виды моделирования – моделирование на постоянном токе, где нет нужды что-либо менять в свойствах. Только добавить.

Рис. 8.18. Сообщение о необходимости добавить моделирование на постоянном токе

Моделирование на переменном токе можно использовать для того, чтобы получить представление о мощности, рассеиваемой на коллекторе транзистора, в зависимости от частоты. Ниже показаны функции мощности и амплитуды от частоты.

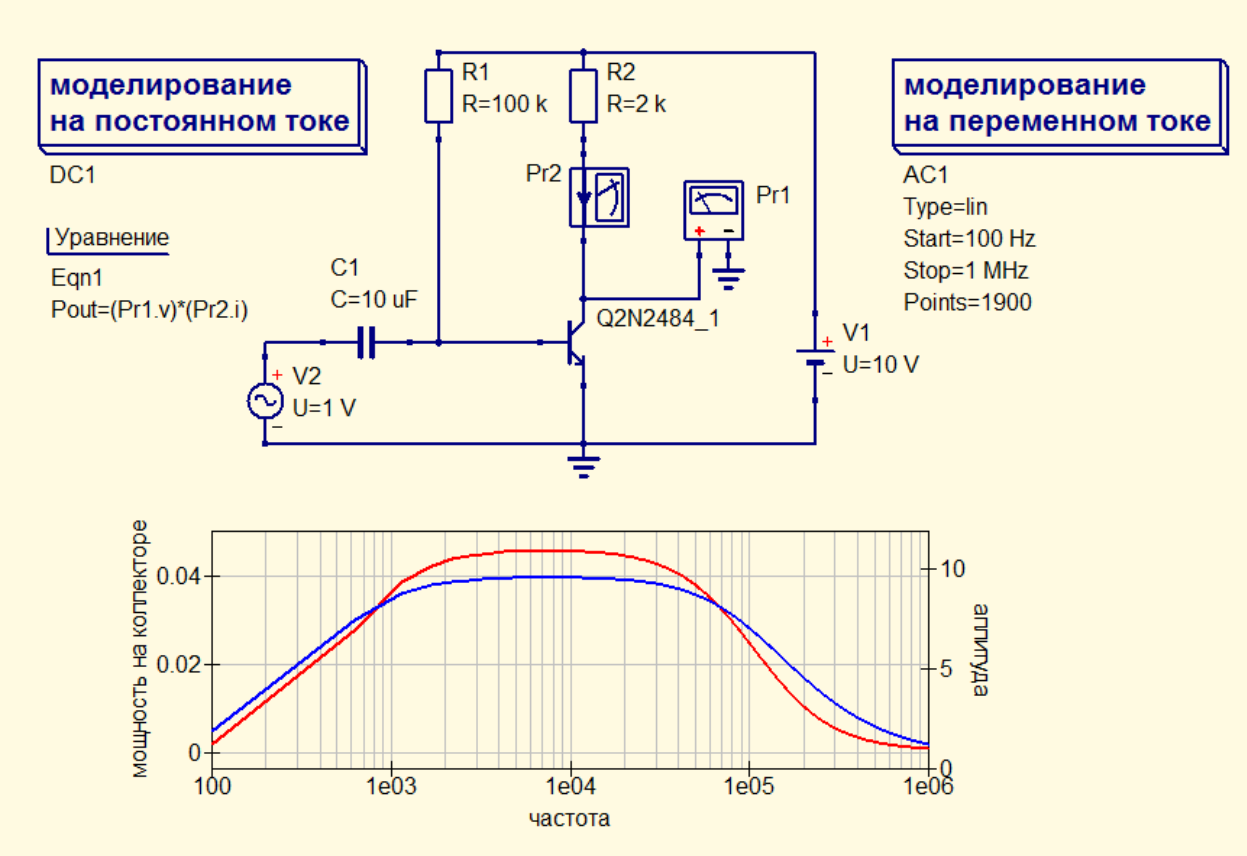
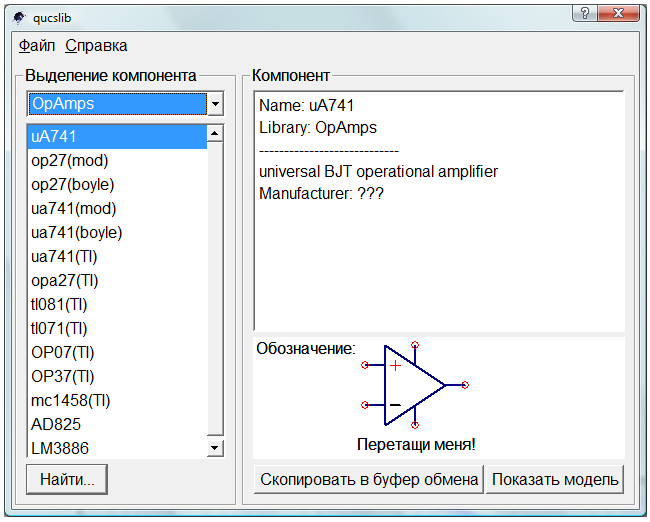


Рис. 8.19. График зависимости мощности (верхняя кривая) на коллекторе от частоты

Процесс моделирования на переменном токе подразумевает моделирование с применением малосигнальных параметров, то есть, без учета влияния нелинейных процессов, которые могут иметь место в реальной схеме. Когда к схеме добавляется источник переменного напряжения, его параметры можно оставить такими, какие они есть по умолчанию, но… Рассмотрим конкретный пример – амплитудно-частотную характеристику операционного усилителя.

Из набора **нелинейных компонентов** закладки **Компоненты** можно взять операционный усилитель, но это будет идеализированный компонент. Полученная АЧХ может простираться до значений, существенно превосходящей наши интересы. Воспользуемся реальной моделью, которую можно найти в **Библиотеке компонентов** (группа ***OpAmps***).



Операционный усилитель uA741 один из первых, появившихся в широком пользовании.

Сегодня его можно считать устаревшим, но он прекрасно подходит для разного рода экспериментов, целью которых выбрано знакомство с операционными усилителями.

Рис. 8.20. Выбор операционного усилителя из Библиотеки компонентов

Для добавления компонента в схему достаточно, как и советуют, перетащить его в схему: «подцепить» мышкой рисунок в правом окне диалога, переместить курсор с контуром символа в нужное место при удержанной левой клавише, а в нужном месте отпустить клавишу мышки.

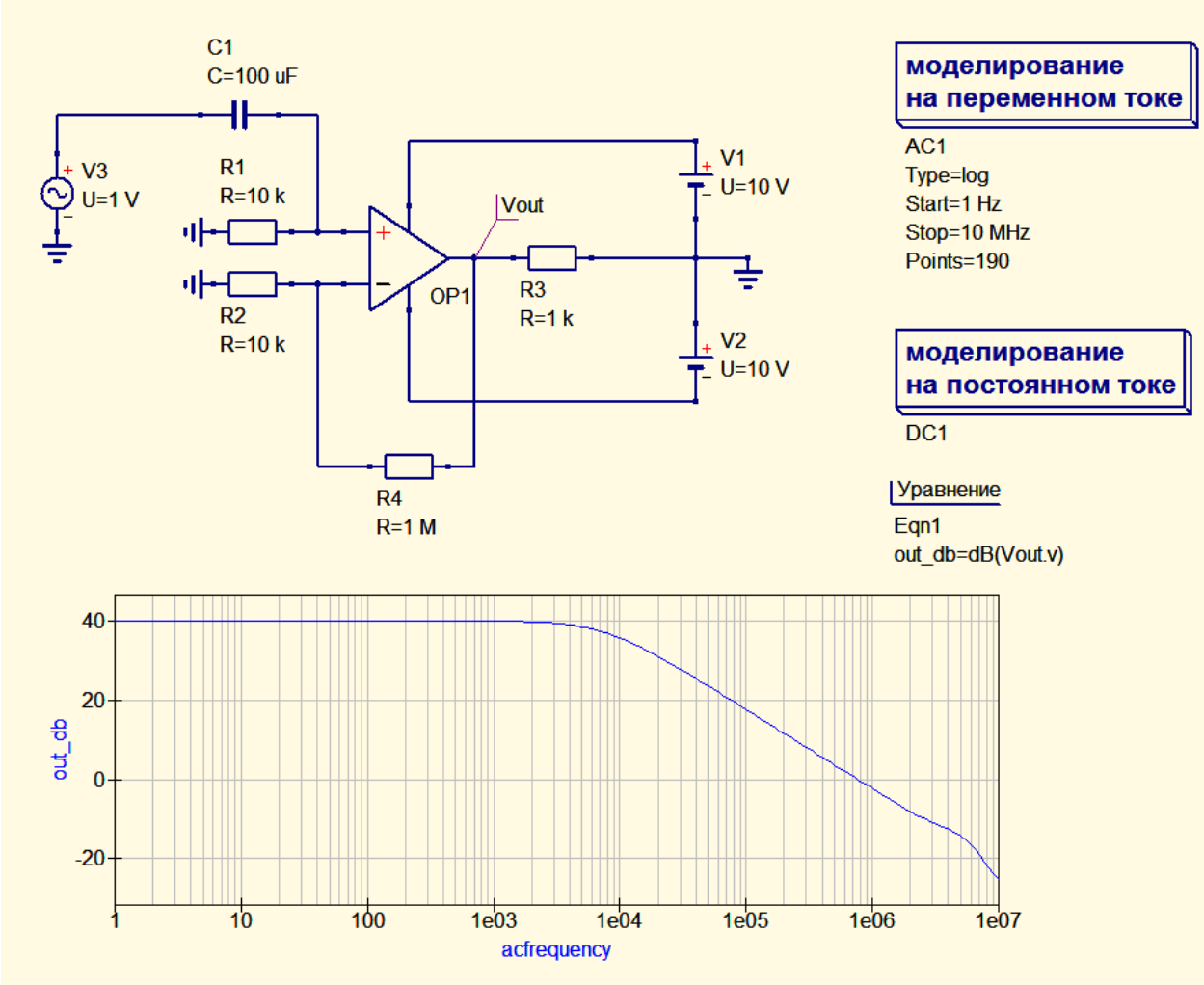


Рис. 8.21. Моделирование на переменном токе операционного усилителя

Добавленное к схеме уравнение позволяет получить АЧХ в децибелах. Расчет коэффициента усиления ОУ по напряжению «навскидку» - это отношение величины резистора R4 к R2, что в децибелах отвечает значению 40 дБ. Параметры источника сигнала V3 оставались неизменными. Но, если бы мы имели дело с реальным операционным усилителем, мы выбрали бы величину входного сигнала порядка 1 мВ. Проверим, что получается при моделировании, если изменить значение напряжения источника сигнала.

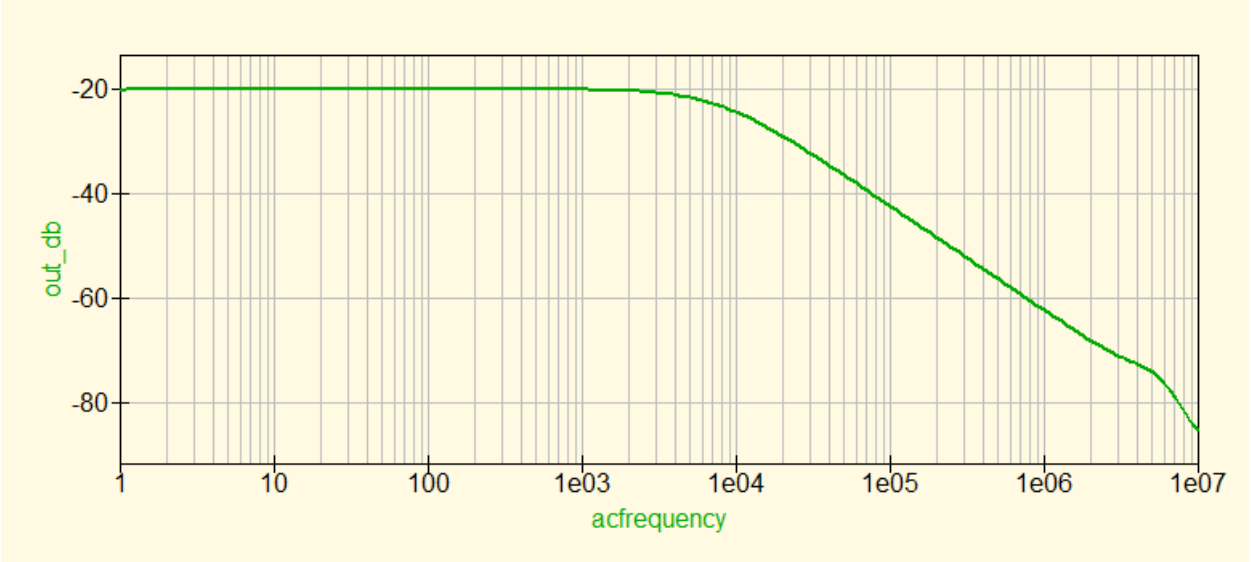


Рис. 8.22. Амплитудно-частотная характеристика ОУ при изменении входного напряжения

То, что АЧХ не меняет своего характера, позволяет определить многие интересующие параметры, но изменение величины сигнала (с 40 дБ до – 20 дБ) следует учитывать, не забывать об этом.

С помощью маркеров можно определить параметры.

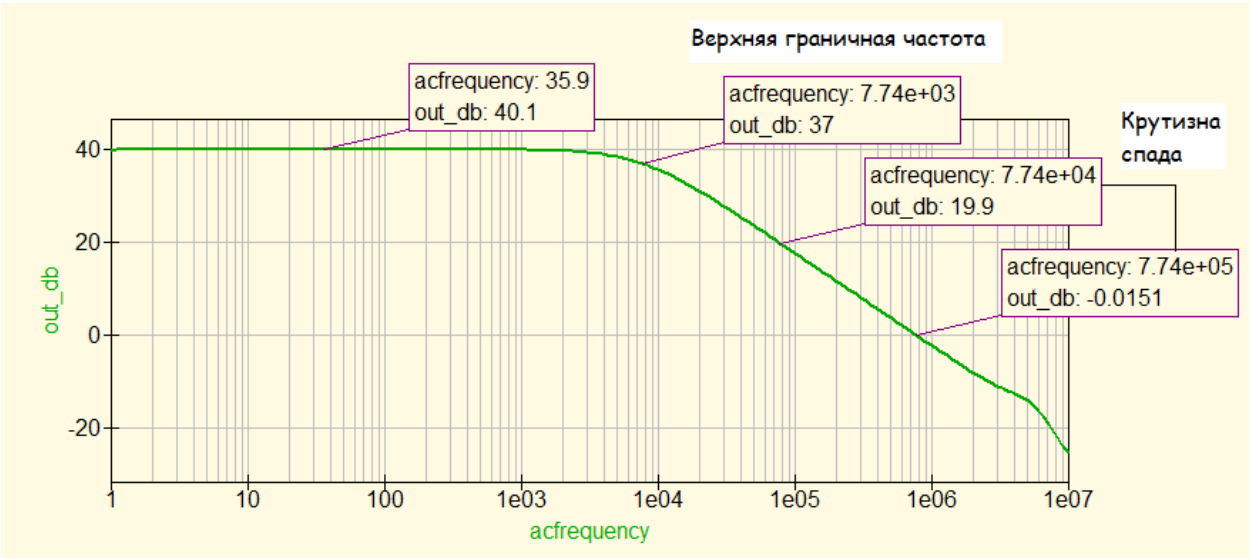


Рис. 8.23. Определение параметров АЧХ с помощью маркеров

Моделирование переходного процесса

При моделировании на переменном токе предыдущей схемы, если бы был использован выходной сигнал *Vout*, то полученный график показал бы выходное напряжение в 100 В. Конечно, реальный операционный усилитель ограничит выходное напряжение значительно раньше.

Чтобы увидеть это ограничение, следует использовать моделирование переходного процесса.

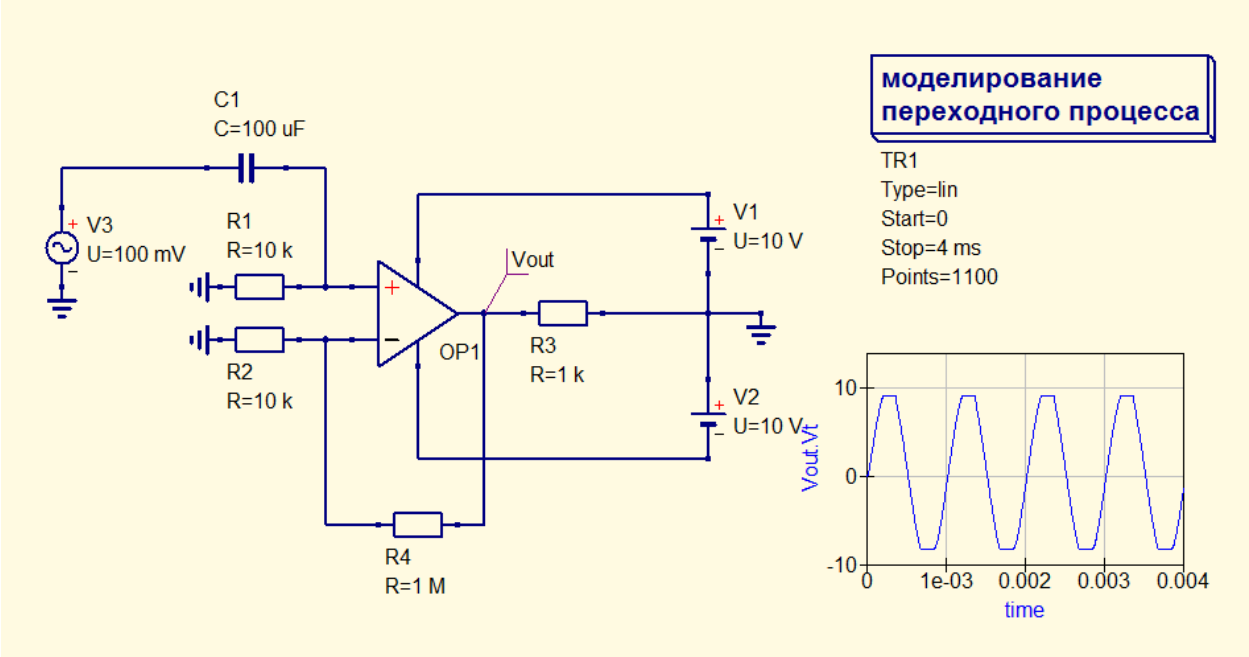


Рис. 8.24. Моделирование переходного процесса

Если при моделировании на переменном токе свойства источника сигнала V3 можно было оставить такими, какие они есть по умолчанию, а в свойствах моделирования следовало выбрать логарифмический тип, то при **моделировании переходного процесса** следует не забыть и изменить частоту с 1 ГГц на 1 кГц.

Как видно из осциллограммы, ограничение выходного сигнала начинается при значении чуть меньшем 10 В.

Моделирование переходного процесса очень важный и нужный аспект программы, в реализации которого есть свои трудности. И проблемы могут быть связаны не только со сложностью моделируемой электрической цепи. Рассмотрим простой вопрос о токе, протекающем в цепи, при заряде конденсатора от источника постоянного тока.

Если мы говорим о заряде конденсатора, как о переходном процессе, то есть, о процессе, занимающем некоторое время, начинающемся при одном состоянии цепи, а заканчивающемся при другом, то нам следует озаботиться в первую очередь о создании подходящих начальных условий процесса. Скажем, бесполезно пытаться получить моделирование переходного процесса от схемы, показанной ниже.

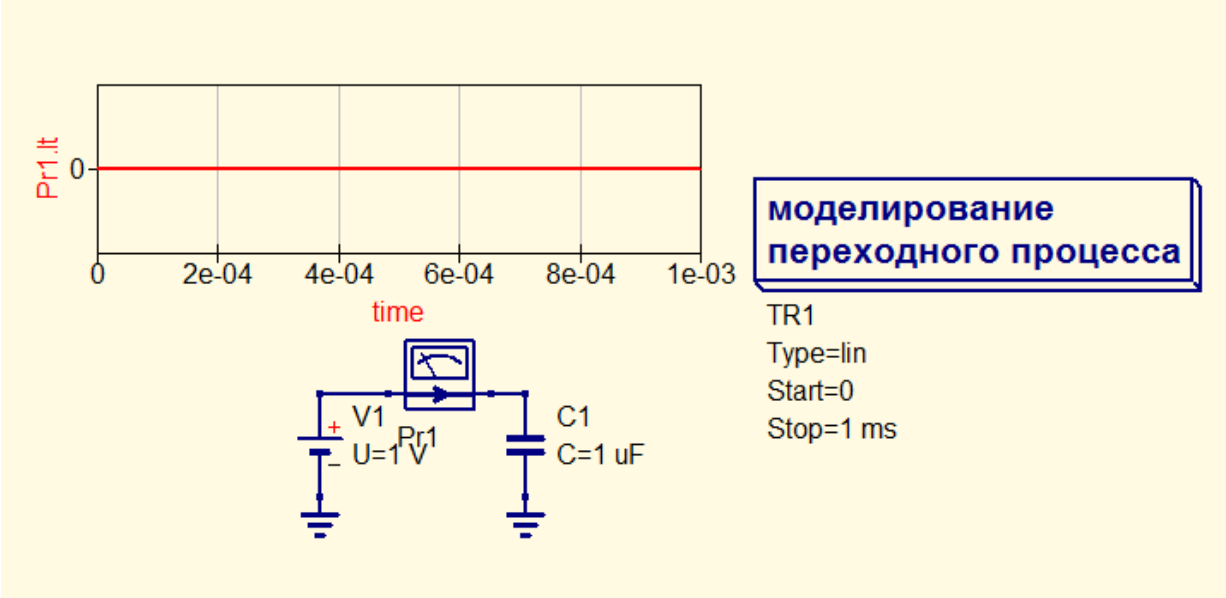
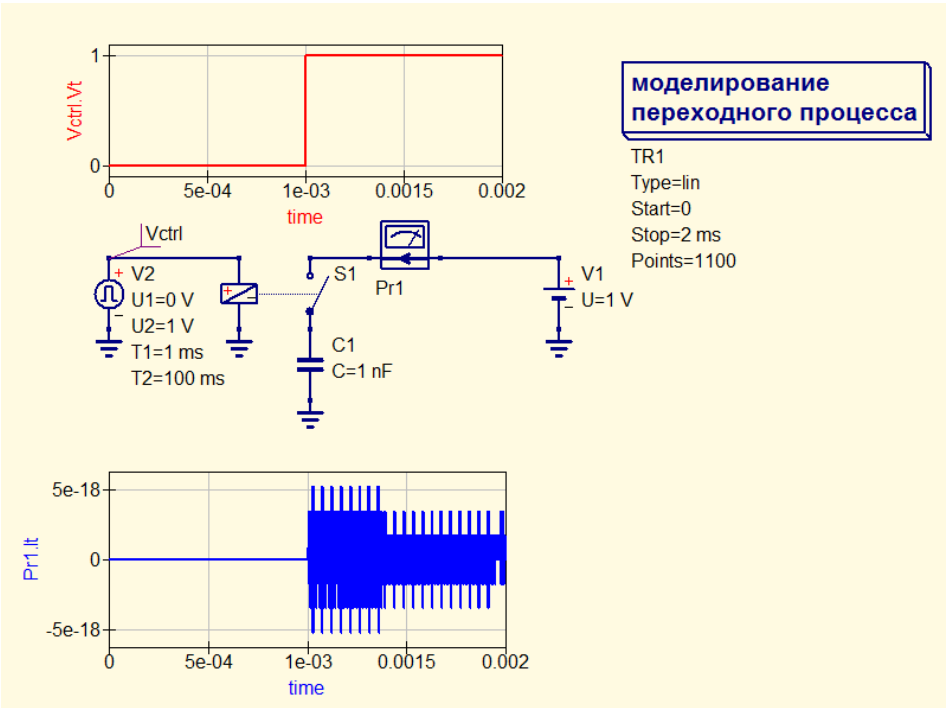


Рис. 8.25. Заряд конденсатора от батарейки

Модель «честно» показывает отсутствие тока через конденсатор. Постоянный ток не протекает через идеальный конденсатор.

Создадим «подходящие» условия для наличия переходного процесса, добавив к схеме реле, которое будет включаться от источника импульсного напряжения, а импульс задержим на 1 мс.

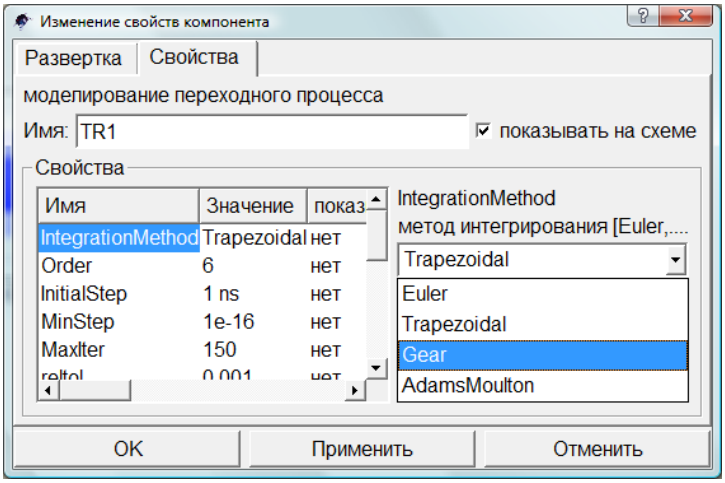


Верхняя диаграмма показывает импульс, управляющий работой реле.

Нижняя диаграмма показывает… что-то непонятное, которое начинается после включения реле.

Рис. 8.26. Модификация схемы заряда конденсатора

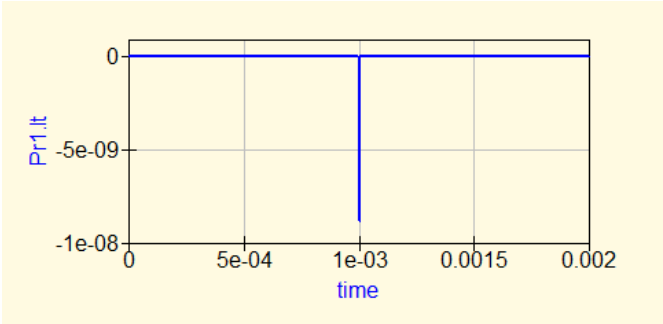
Обратимся к свойствам компонента **моделирование переходного процесса**.



На закладке ***Свойства*** диалогового окна заменим трапецеидальный метод интегрирования методом Гира с порядком 6.

Рис. 8.27. Изменение метода интегрирования

Полученная диаграмма тока через конденсатор изменится.

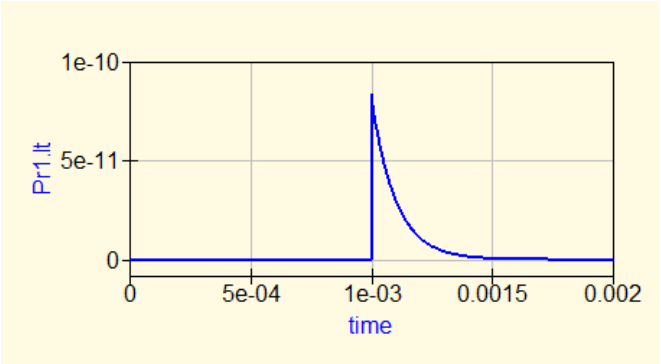


Сказать, что улучшение радикально, не скажешь. Но… и источник V1, и конденсатор – элементы идеальные.

Первый может отдавать бесконечно большой ток, второй пропускать его. А модель не любит играть с бесконечностями.

Рис. 8.28. Новый вид осциллограммы тока через конденсатор

Добавим резистор последовательно с измерителем тока Pr1 величиной 100 кОм.



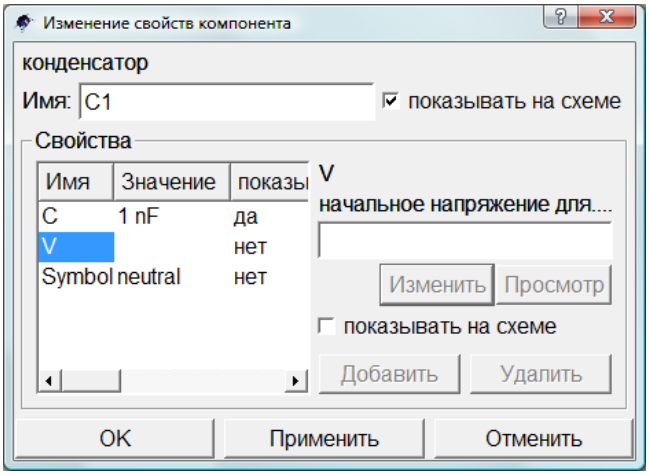
Ток через конденсатор, как ему и положено, в начальный момент времени большой, постепенно спадая к концу процесса. Это очень похоже на реальный или воображаемый нами процесс заряда конденсатора через резистор. Но…

Рис. 8.29. Диаграмма заряда конденсатора через резистор

Но всегда есть что-то при работе с программами симуляции электрических цепей, что нас не устраивает. В данном случае, если разделить 1 В на 100 кОм, то значение будет отлично от того, что мы видим на диаграмме. А если рассудить, то в начальный момент времени ток должен

определяться именно величиной напряжения источника и величиной сопротивления резистора. Что не так?

Заглянем в свойства конденсатора, отображаемые в диалоговом окне.



В свойствах конденсатора предусмотрен такой параметр, как начальное напряжение для моделирования переходных процессов. По умолчанию оно не определено. Сделаем его равным нулю.

Рис. 8.30. Диалоговое окно свойств конденсатора

Теперь результат моделирования не вызывает вопросов. А вывод, который можно сделать из работы со столь простой схемой – не следует во всем полагаться на удачу. Нужно и подумать, и попробовать, и подправить…

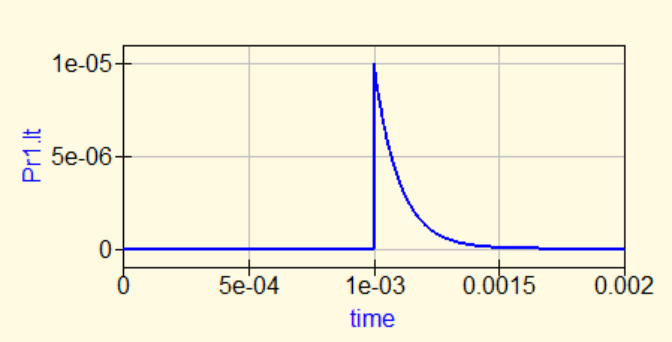


Рис. 8.31. Окончательный вид диаграммы

Очень часто при моделировании переходного процесса, а это один из самых распространенных видов моделирования, возникают проблемы с генерацией. То есть, проблемы при исследовании схем генераторов. Вот одна из распространенных схем низкочастотного генератора.

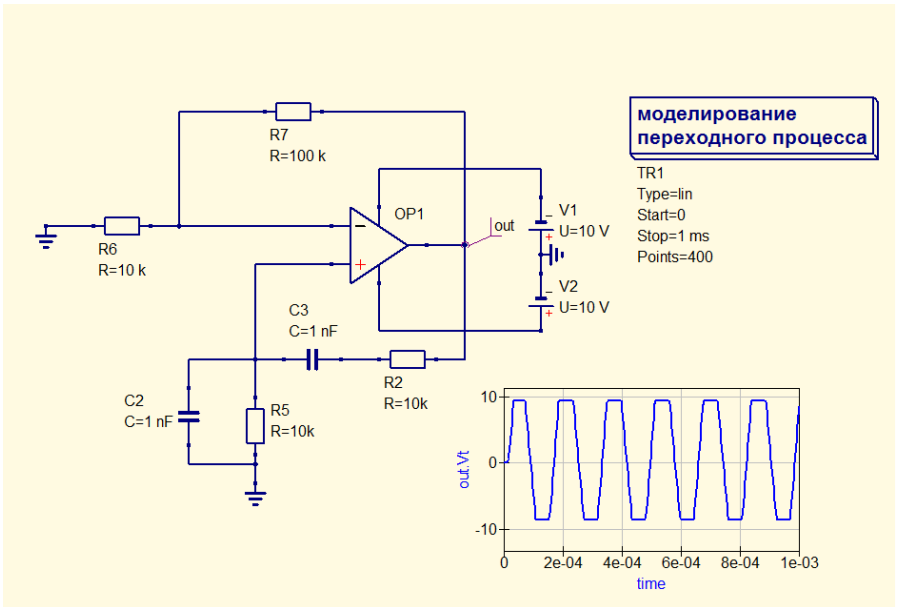


Рис. 8.32. Схема генератора с мостом Вина

И в этом случае моделирование проходит успешнее, если задать начальное напряжение конденсаторов равное нулю. В качестве операционного усилителя из Библиотеки компонентов (раздел Инструменты) выбран uA741. Чтобы приблизить вид сигнала к синусоидальному, в цепь отрицательной обратной связи включают нелинейные элементы (или некоторые типы терморезисторов).

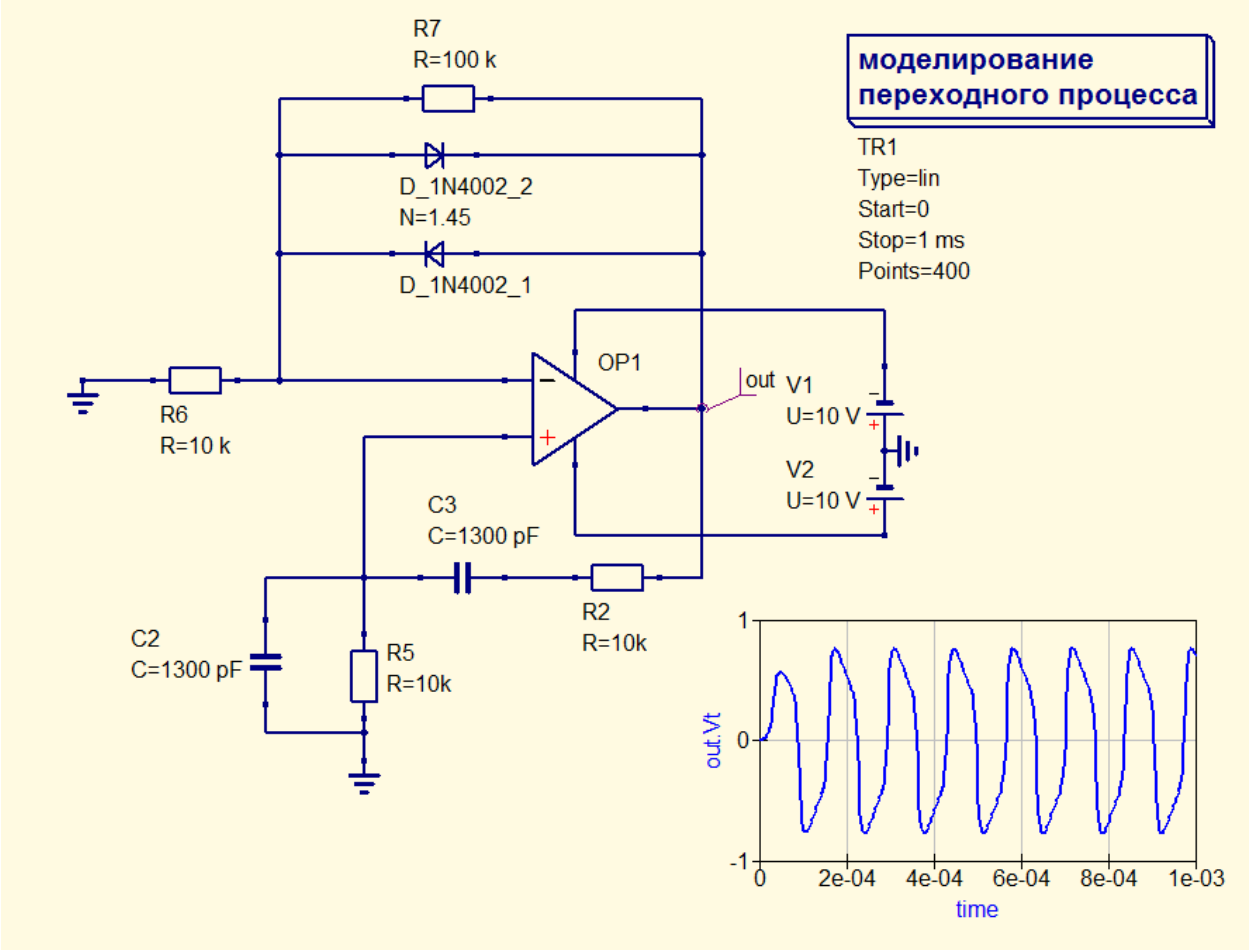
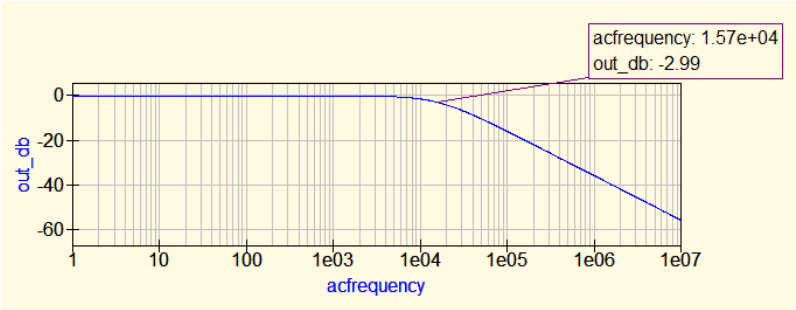


Рис. 8.33. Генератор с диодами в цепи отрицательной обратной связи

Все знают, что однокаскадный усилитель при проверке его амплитудно-частотной характеристики, ведет себя подобно интегрирующей RC-цепи. Можно посмотреть АЧХ такой цепи, используя **моделирование на переменном токе**.



Маркер показывает, что верхняя граничная частота около 16 кГц.

Рис. 8.34. Амплитудно-частотная характеристика RC-цепи

А теперь подвергнем эту же цепь анализу переходного процесса, используя в качестве источника сигнала генератор прямоугольных импульсов.

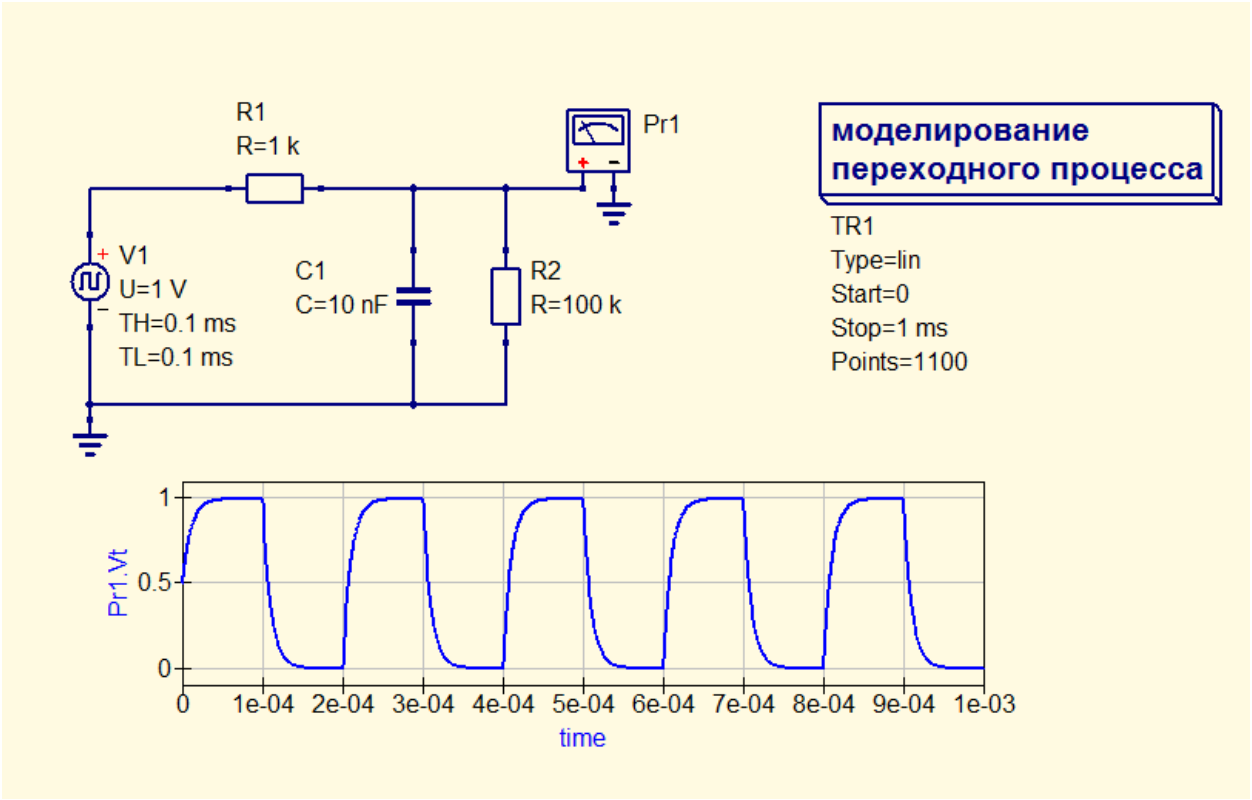


Рис. 8.35. Испытание RC-цепи прямоугольными импульсами

Затяжки фронтов импульсов свидетельствуют о недостаточно высокой верхней частоте среза цепи. Если уменьшить величину конденсатора C1, то форма импульсов изменится.

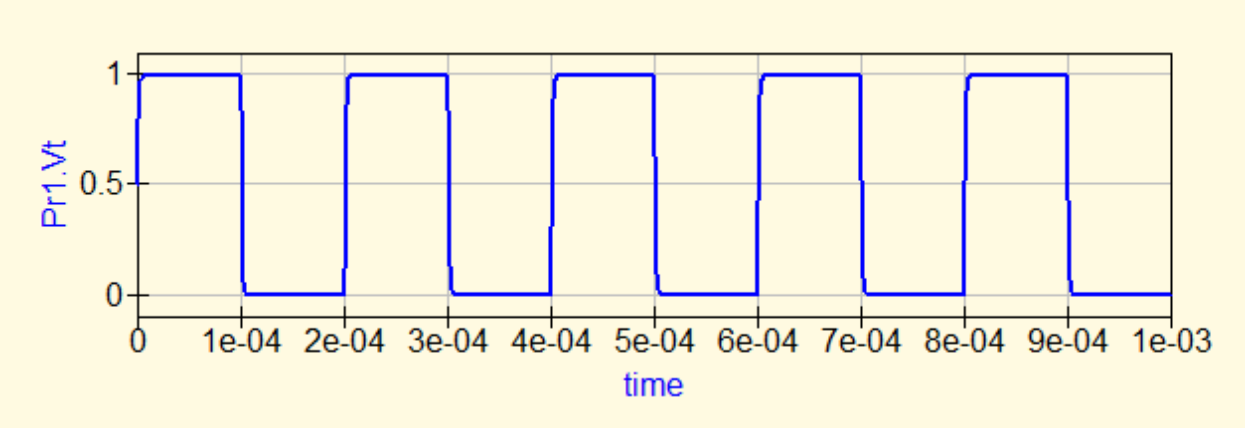


Рис. 8.36. Форма импульсов при емкости C1 = 1 нФ

Проведя ряд подобных экспериментов, можно получить практический навык быстрой оценки полосы пропускания усилителя при использовании генератора прямоугольных импульсов.

Если поменять местами в RC-цепи резистор R1 и конденсатор C1, удалив R2 и изменив времена **источника прямоугольного напряжения** до 1 мс, то мы получим следующую осциллограмму.

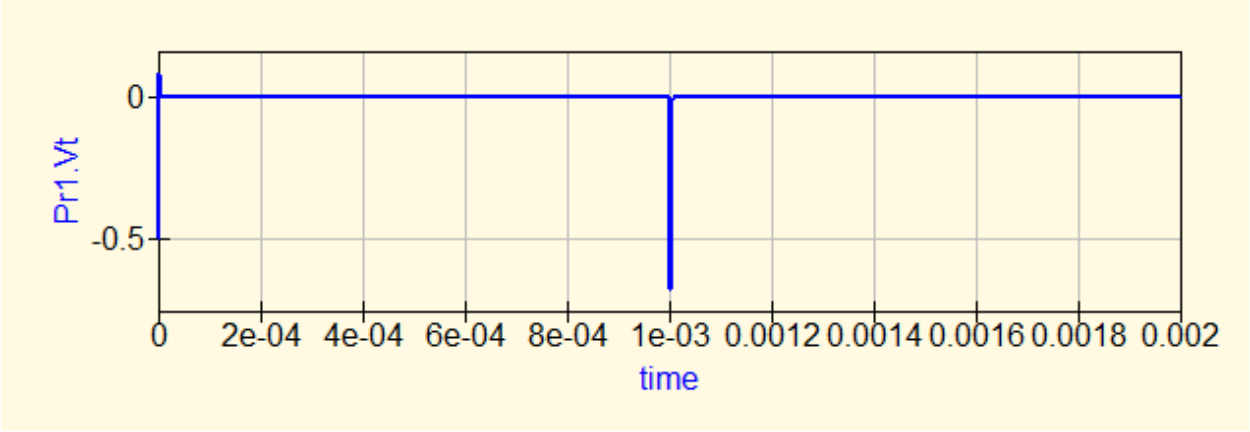


Рис. 8.37. Вид осциллограммы для дифференцирующей RC-цепи

Не знаю, как вам, мне она очень напоминает рисунок 8.28. Ускорим процесс, изменив временные настройки **источника прямоугольного напряжения** и **моделирования переходного процесса**.

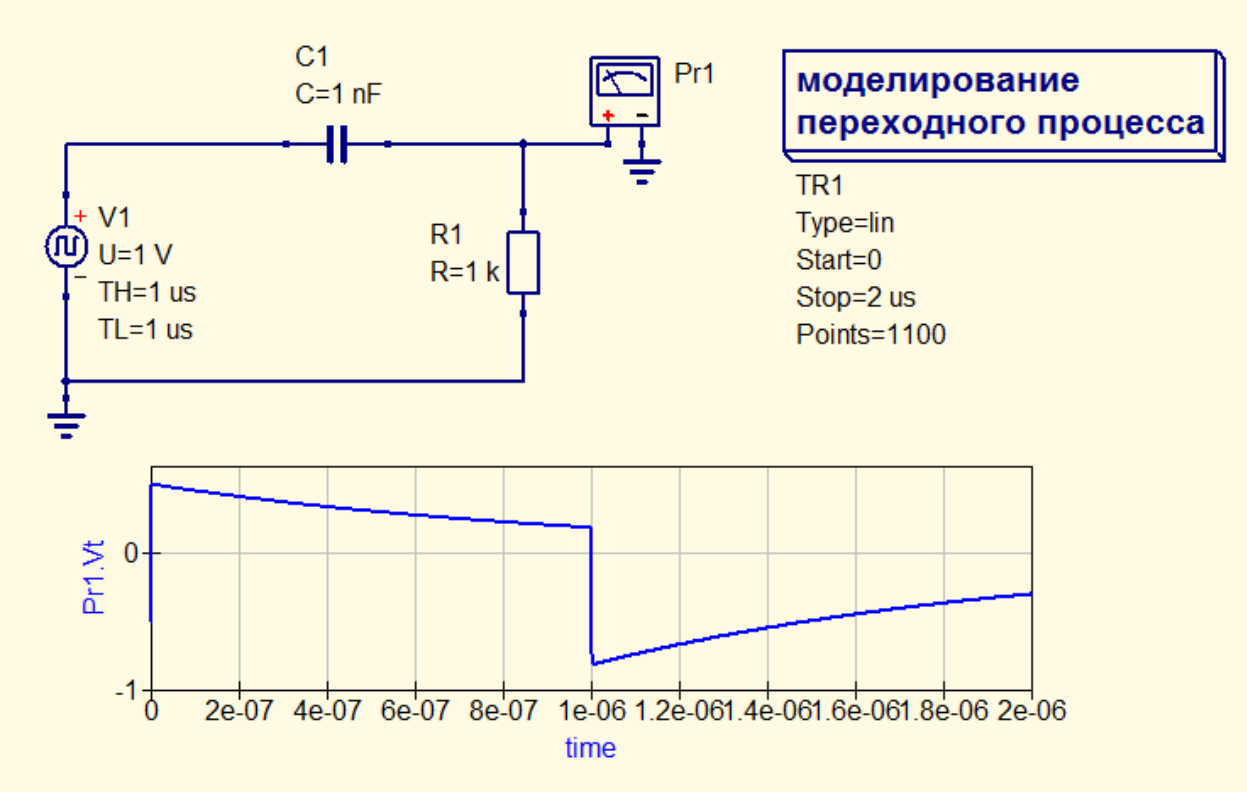


Рис. 8.38. Реакция дифференцирующей RC-цепи на прямоугольные импульсы

По реакции на прямоугольные импульсы можно судить о нижней граничной частоте усилителя. Чем ярче выражен «скат» плоской вершины импульса, тем выше эта частота. Заменим конденсатор C1 на 1 мкФ и получим следующую осциллограмму импульса.

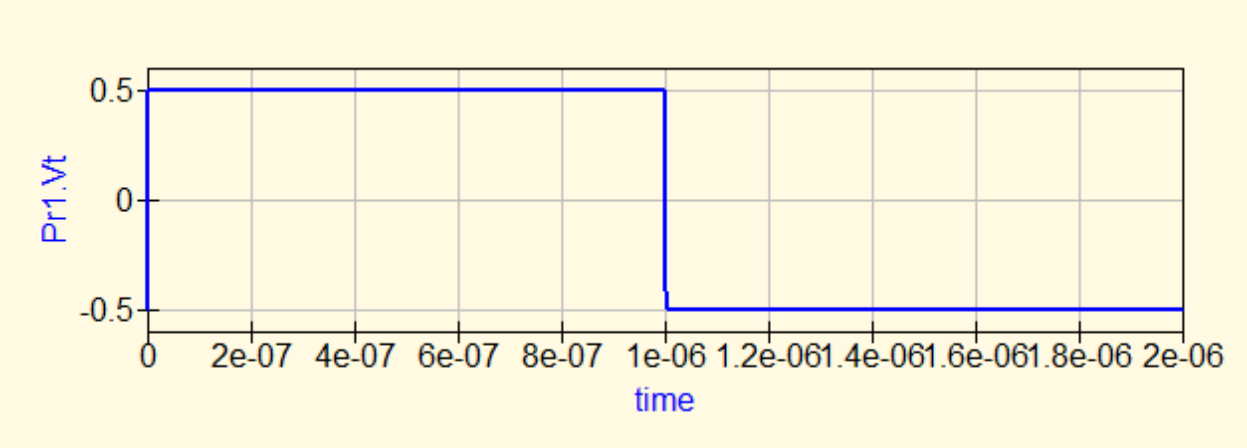


Рис. 8.39. Вид импульса при увеличении емкости конденсатора

Все эти эксперименты можно провести и с помощью реальных приборов. Но даже получение АЧХ любой электрической цепи без специализированного прибора задача достаточно трудоемкая. Наблюдение редких и коротких импульсов, как при опытах с зарядом конденсатора, тоже потребует специализированных приборов. Тогда как моделирование переходного процесса в программе позволяет наблюдать поведение токов и напряжений в различных электрических цепях быстро и комфортно, узнавая многое о свойствах различных электронных схем.

При моделировании переходного процесса применяются достаточно сложные математические операции. И этот вид моделирования оказывается очень чувствителен к параметрам, задаваемым в диалоговом окне его свойств. Вот пример работы с простейшей электрической цепью.

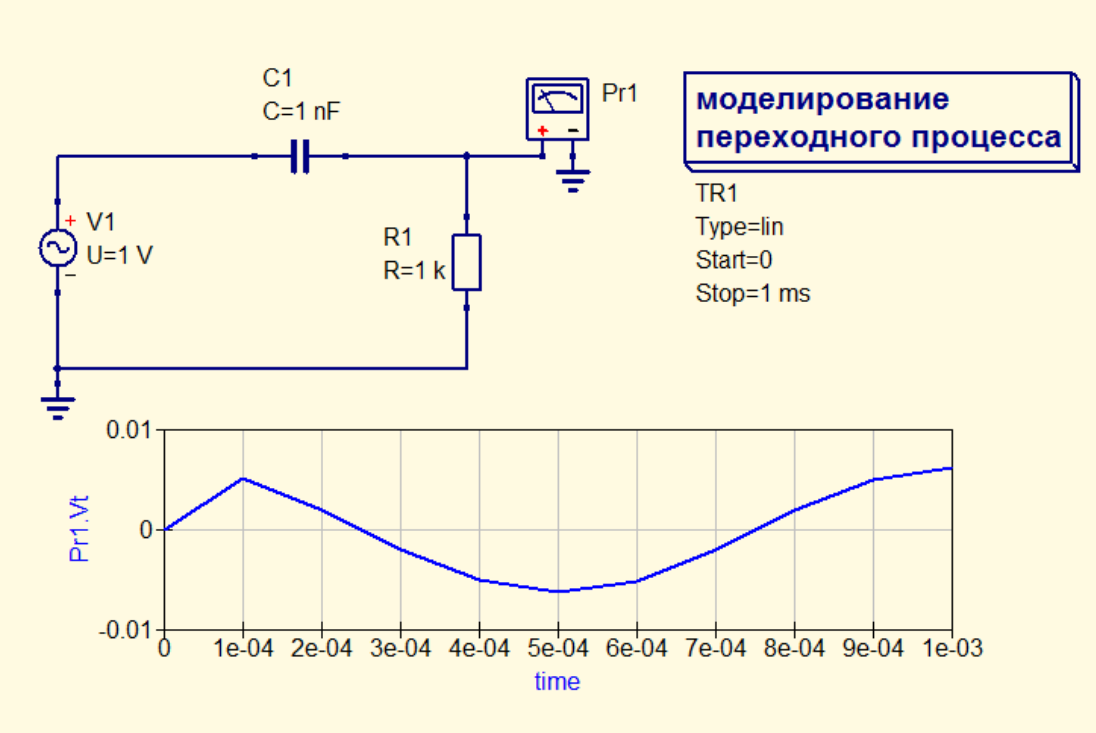
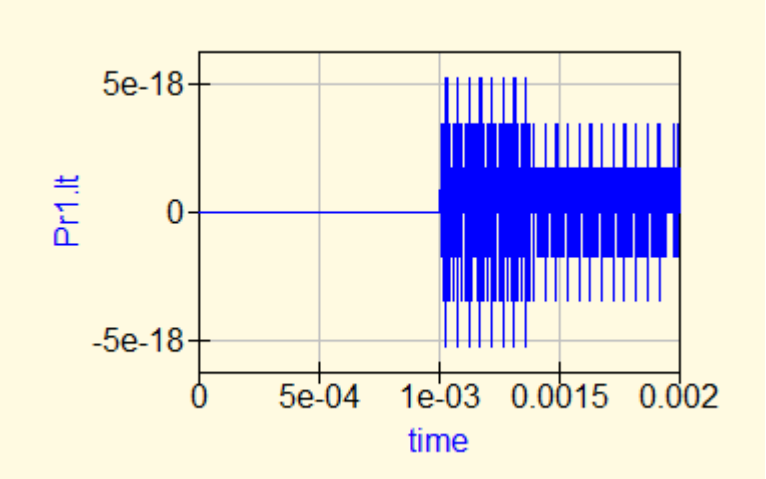


Рис. 8.40. Прохождение синусоидального сигнала через конденсатор

Каждому понятно, что конденсатор не должен так искажать вид синусоиды. В чем же причина?

В элементарной небрежности. Перенося на схему компонент **моделирование переходного процесса**, мы не озаботились увеличить количество расчетных точек. По умолчанию их всего 11. Даже увеличение их до 100 существенно исправит ситуацию. Но иногда увеличение количества точек вызывает замедление процесса симуляции. Это может происходить в сложной схеме из-за увеличения количества операций. А в простой (по виду) схеме это может быть связано с тем, что среди расчетных точек появляются такие, где решение системы уравнений для токов и напряжений становятся расходящимися. Появившиеся сингулярности либо срывают процесс симуляции, либо так его замедляют, что дождаться конца операции не у всякого хватит терпения.

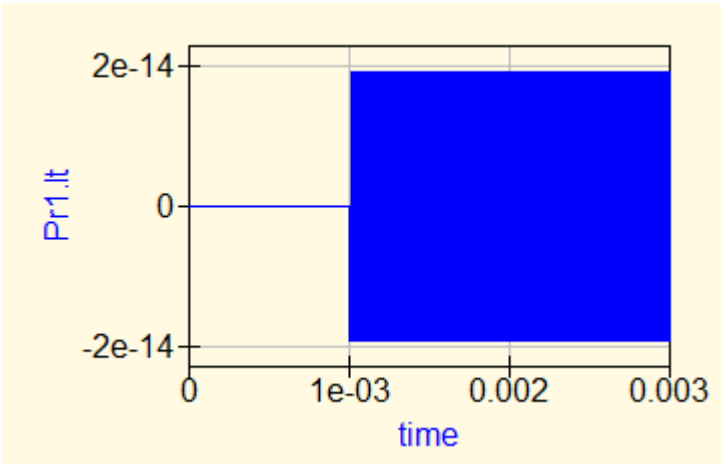
Изменяя время переходного процесса можно получать разные диаграммы. Их вид при этом определяется не столько происходящими в цепи процессами, сколько отображает процесс расчетов. Вернемся к рисунку 8.26. Диаграмма отображает некий «сложный процесс».



Диапазон изменения тока уже вызывает сомнения. А сам процесс… остается непонятен.

Рис. 8.41. Диаграмма рисунка 8.26

Но, изменив время наблюдения переходного процесса с 2 мс на 3 мс, мы получим другую диаграмму.



Можно продолжать изменение времени процесса, получая новые виды диаграммы.

Однако они отображают не процесс заряда конденсатора, а процесс попыток рассчитать электрическую цепь.

Рис. 8.42. Предыдущая диаграмма при изменении времени процесса

При работе с переходными процессами, а это один из самых удобных способов исследования электрических цепей, следует понимать, что моделирование электрической цепи – сложная

математическая обработка заданных параметров. Прежде, чем довериться полученным результатам, следует задуматься, а что отображает диаграмма?

Развертка параметра

С этим видом моделирования мы уже знакомились выше, где он с успехом выступал как дополнение к другим видам моделирования. Основное назначение этого компонента программы – показать, как изменяются параметры электрической цепи, когда происходят изменения параметров одного или нескольких компонентов.

В области низких частот наиболее употребительна схема включения транзистора с общим эмиттером. При этом большое внимание при проектировании уделяется выбору рабочей точки. А при реализации схемы не меньшее внимание уделяется вопросу стабилизации выбранной рабочей точки. Связано это в первую очередь с тем, что полупроводниковые приборы очень чувствительны к изменениям температуры. Правда, это их свойство используется и во благо – терморезисторы и датчики температуры на основе полупроводников используют чувствительность полупроводников к температуре.

Посмотрим, как влияет температура на напряжение на коллекторе транзистора в простейшей схеме. Поможет в этом **развертка параметра**.

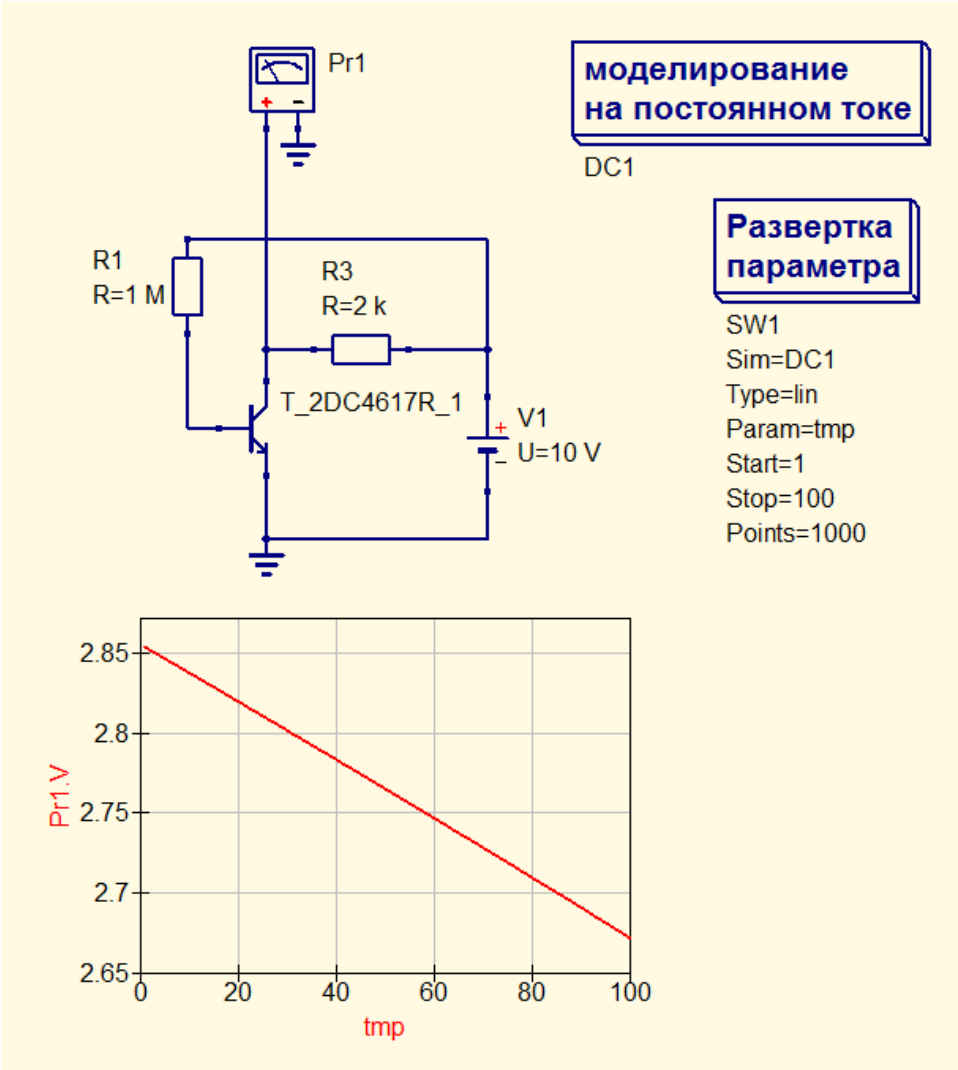


Рис. 8.43. Схема для исследования температурного влияния на рабочую точку

Транзистор для эксперимента выбран в **Библиотеке компонентов** (раздел **Инструменты**).

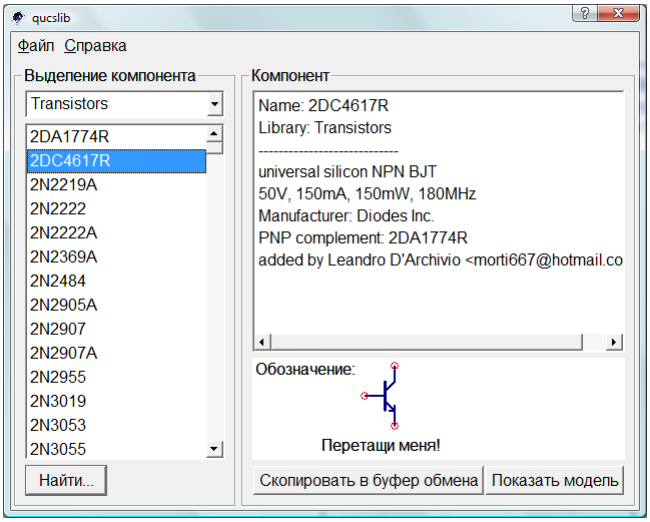


Рис. 8.44. Выбор транзистора из группы *Transistors* **Библиотеки компонентов**

В свойствах транзистора (двойной щелчок по транзистору в схеме открывает диалоговое окно его свойств) есть параметр *Temp*, температура. По умолчанию задано 26.85 градусов. Заменим его переменной ***tmp***.

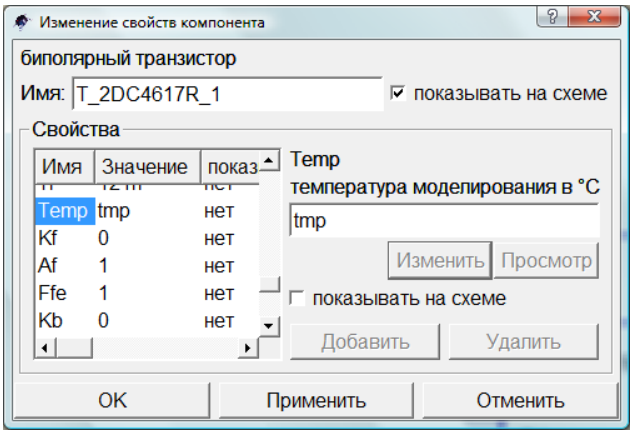


Рис. 8.45. Задание переменной в свойствах транзистора

Эту переменную мы используем в качестве параметра «качания» для моделирования **развертки параметра**. Для этого откроем диалоговое окно этого вида моделирования и произведем нужные настройки.

Обратите внимание на кнопки со стрелкой вниз. Обычно такие кнопки в программе означают, что нажатие на них выводит список возможных значений. Рядом с окном, отображающим ***Моделирование:*** есть такая стрелка. Если у вас на схеме есть моделирование на постоянном токе, как на рисунке, то в этом окне отобразится именно оно. Если есть несколько видов моделирования, то отобразятся они все.

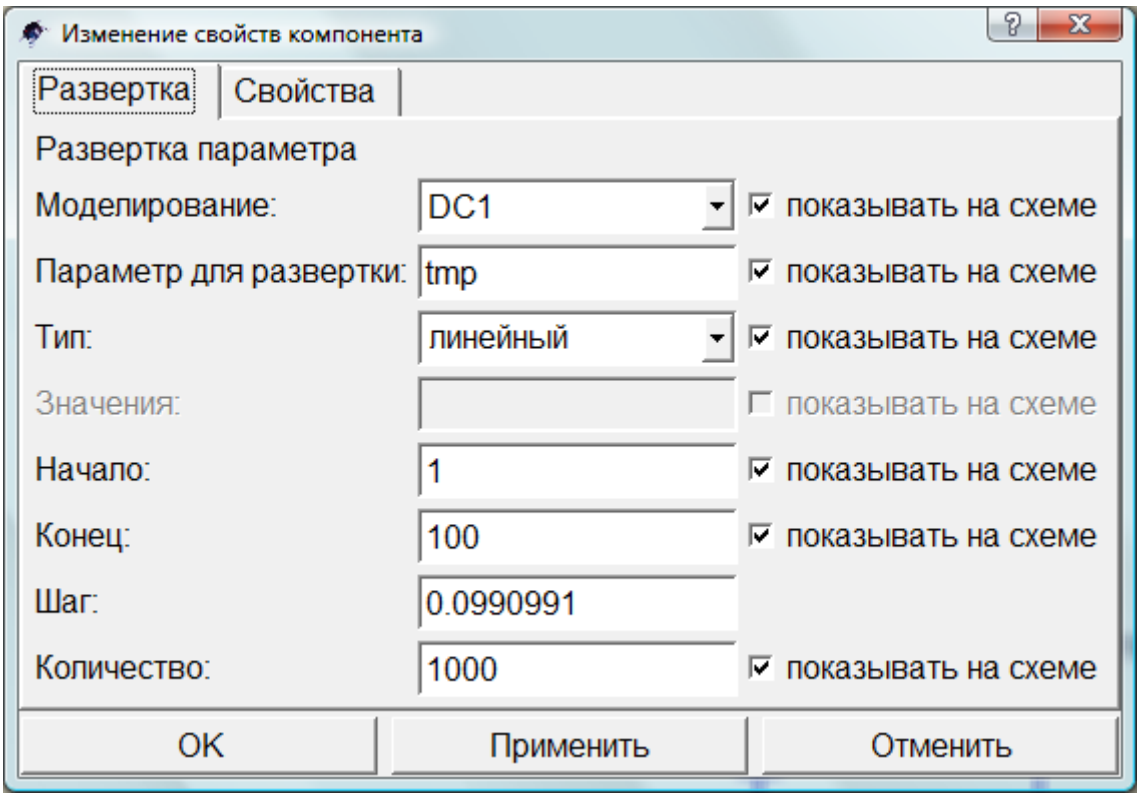
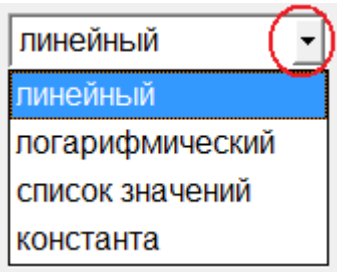


Рис. 8.46. Диалоговое окно развертки параметра

Параметр для развертки мы вписываем тот, что задали в свойствах транзистора: ***tmp***. Тип развертки можно оставить линейный, но симуляция этого вида позволяет использовать еще несколько типов.



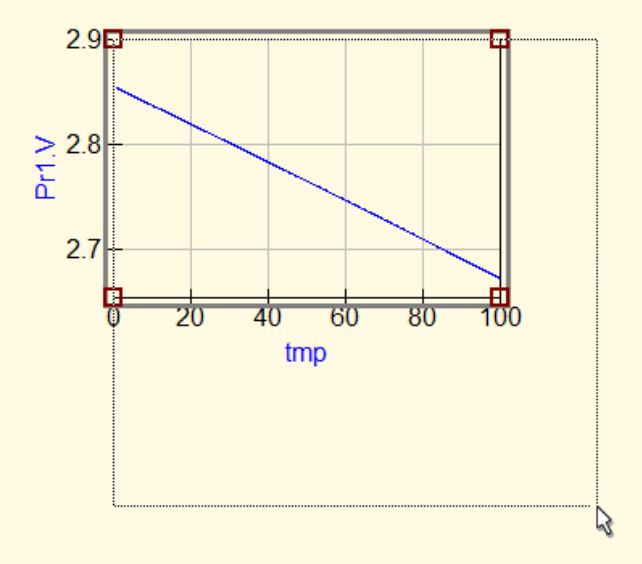
Кнопка со стрелкой выделена!

Обратите внимание на такой вид, как ***список значений***. Если выбрать его, то активизируется окно ***Значения***, где можно вписать интересующие вас значения параметра, а вас может интересовать поведение схемы при определенных значениях параметра.

Рис. 8.47. Разные типы развертки параметра

Задав начальное и конечное значение параметра, задав количество расчетных точек (или выбрав шаг) мы автоматически получим шаг развертки. После чего готовы приступить к исследованию. Запустив симуляцию, например, нажав клавишу F2 на клавиатуре, мы на странице отображения диаграмм получим возможность перенести Декартовскую диаграмму, выбрав в ее свойствах отображение кривой Pr1. Вид полученной кривой можно скопировать со страницы отображения диаграмм на схему, что и сделано на рисунке 8.43.

Нас вполне может удовлетворить полученный результат, диаграмма. Но, если нет, мы можем использовать маркеры (кнопка M1 на инструментальной панели) в нужном количестве. А можем «растянуть» диаграмму, как нам будет удобнее. Для этого достаточно выделить диаграмму, зацепиться мышкой за один из появившихся квадратиков (навести на него курсор мышки, нажать и удержать левую клавишу мышки) и растянуть диаграмму.



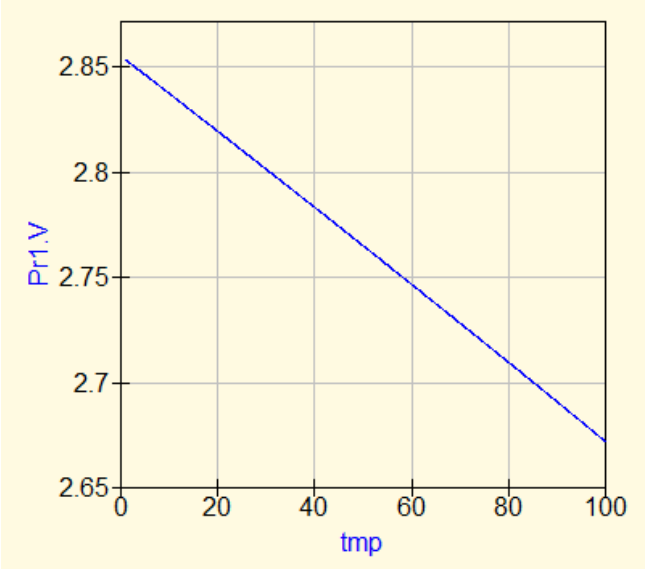
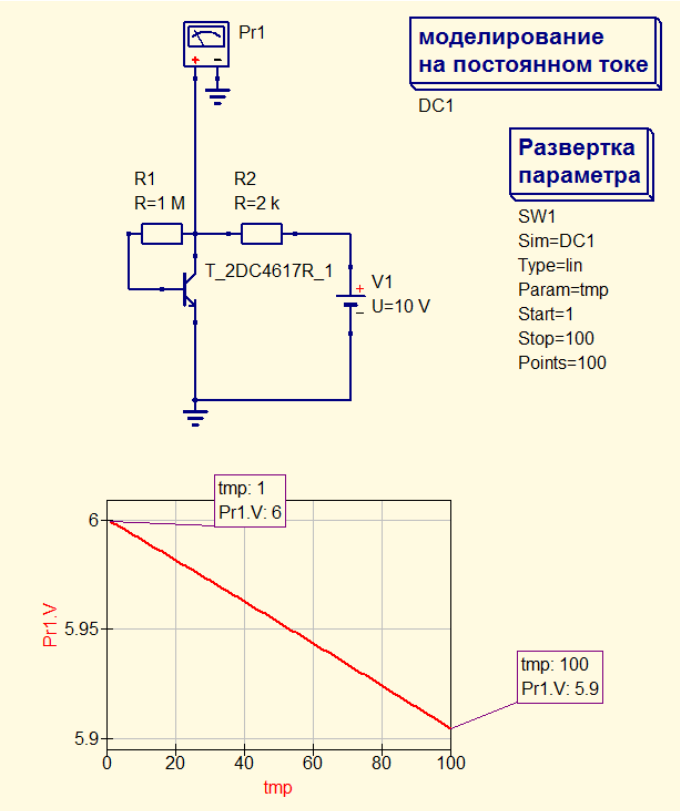


Рис. 8.49. Растягивание диаграммы

Значения по осям будут меняться автоматически, чтобы соответствовать новому масштабу изображения.

Изменение напряжения на коллекторе при заданном нами изменении температуры составляет около 0.2 В. Много ли это, мало ли, решать приходится разработчику. Если проектируемое устройство будет применено в автомобиле рядом с двигателем, то диапазон изменения температуры будет больше. Чтобы избежать влияния температуры на рабочую точку (уменьшить это влияние) принимают ряд схемных решений. Посмотрим, как влияет одно из решений на стабильность рабочей точки.



При тех же изменениях температуры напряжение меняется вдвое меньше.

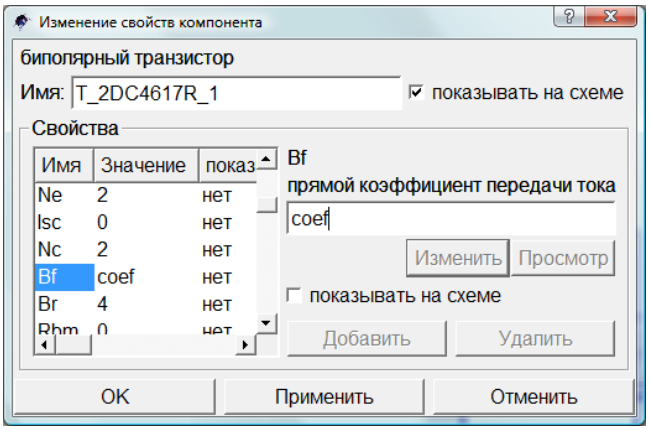
Влияние температуры не одиноко в своем неприглядном усложнении жизни проектировщиков электроники.

Не меньше неприятностей приносит разброс параметров.

Так многие типы транзисторов имеют разброс коэффициента усиления по току в схеме с общим эмиттером, скажем, от 40 до 100.

Рис. 8.50. Введение обратной связи для стабилизации рабочей точки

Посмотрим, как повлияют оба параметра на рабочую точку. Добавим еще одну развертку параметра и еще один параметр для развертки.

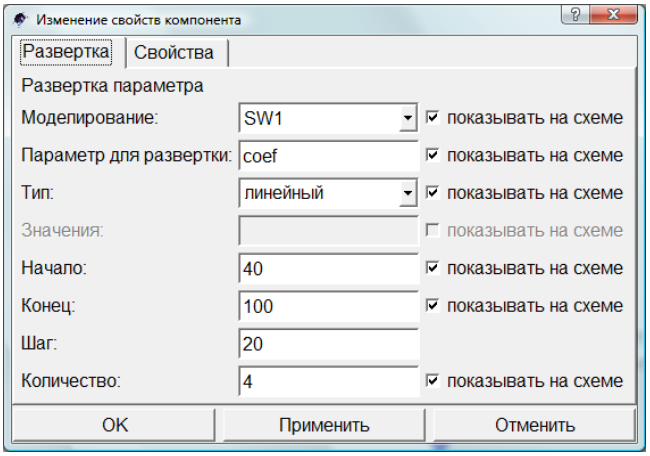


Этот параметр ***coef*** заменит заданное по умолчанию усиление.

Этот же параметр можно использовать во втором моделировании развертки.

Рис. 8.51. Задание усиления транзистора, как параметра развертки

Во втором моделировании в диалоге свойств можно задать первое моделирование как предмет моделирования с параметром моделирования ***coef***.



Выбор разброса параметра 40-100 в данном случае произволен. Но для конкретной модели транзистора его можно получить из справочника.

Рис. 8.52. Настройка второй развертки параметра

Полученная диаграмма позволит лучше понять, влияние какого из параметров должно стать заботой «номер один».

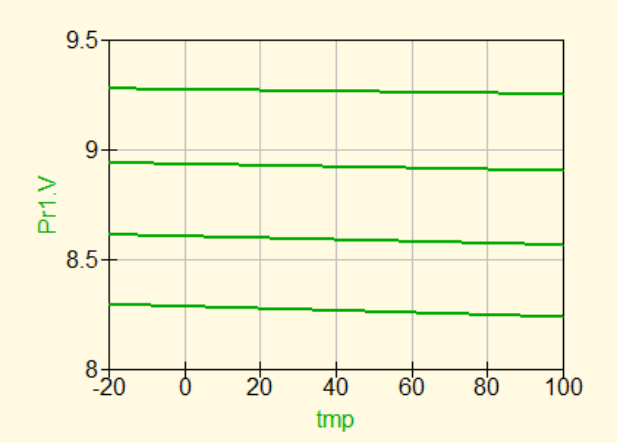


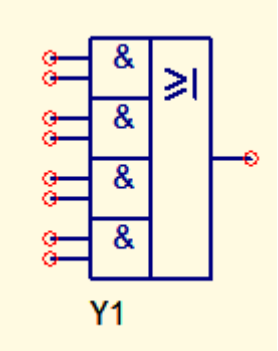
Рис. 8.53. Диаграмма результата развертки двух параметров

Цифровое моделирование

Данный вид моделирования, как это следует из его названия, предназначен для работы с цифровыми схемами. Компонент есть в группе **видов моделирования**, есть и в группе **цифровых компонентов**.

Современная схемотехника позволяет активно использовать функциональные микросхемы, как аналоговые, так и цифровые. А микроконтроллеры все чаще вытесняют «старые, добрые цифровые микросхемы». Следует ли из этого, что нет нужды в таком компоненте, как цифровое моделирование; не лучше ли его заменить моделированием работы микроконтроллера?

Но как быть с самими микроконтроллерами? Кто-то должен их делать, совершенствовать, то есть, разбираться в цифровом моделировании. Среди цифровых микросхем различных серий много таких, назначение которых далеко не очевидно. Что можно сказать, например, о назначении подобного компонента?



Однако компоненты, похожие на этот, входят, например, в серию цифровых микросхем 555 (или в серию SN74).

Рассмотрим работу микросхемы 555ЛР11 (или SN7451).

Рис. 8.54. Цифровой компонент Qucs

Схему этого элемента мы рассмотрим из составляющих. Для нее запишем функцию y(x1,x2,x3,x4).

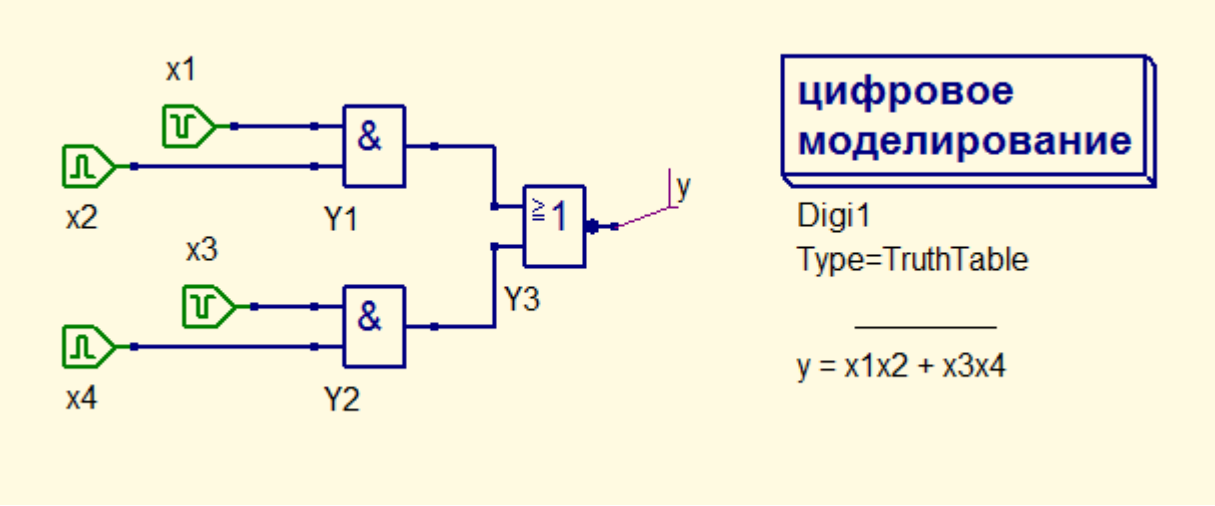


Рис. 8.55. Изображение половинки схемы 555ЛР11

Цифровое моделирование позволяет получить два вида отображения данных: в виде таблицы истинности и в виде временной диаграммы. Выбор осуществляется в диалоговом окне свойств цифрового моделирования.

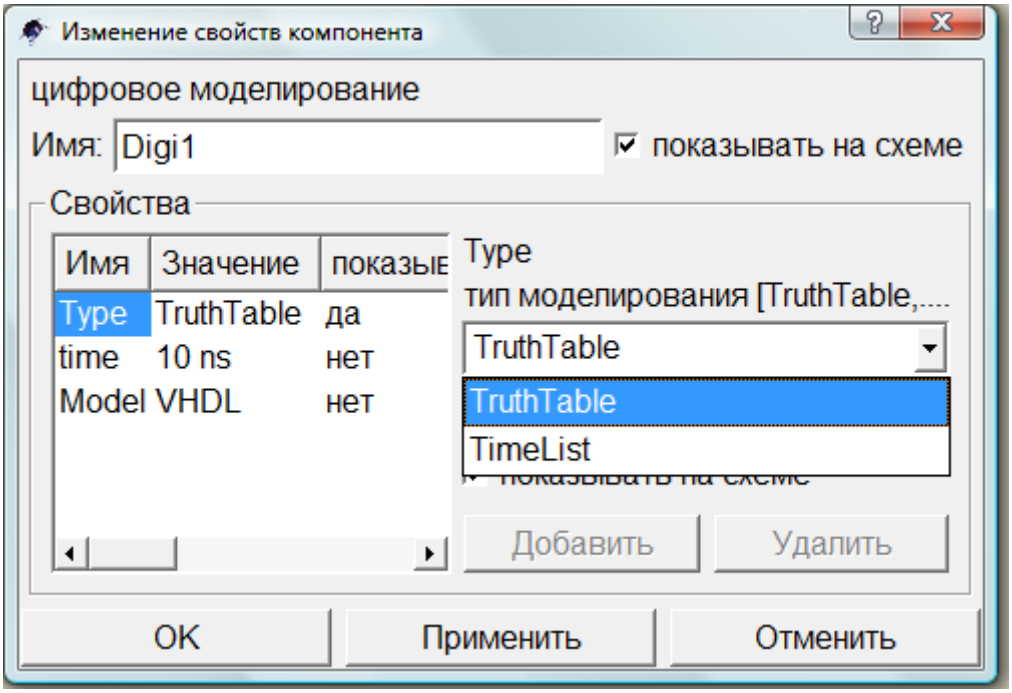


Рис. 8.56. Диалоговое окно свойств цифрового моделирования

При изучении логических свойств цифровых цепей удобно использовать, что мы и сделаем, **таблицу истинности** (***Truth Table***). Запустив моделирование (удобно использовать клавишу ***F2*** на клавиатуре), мы получим таблицу истинности.

	y.X
0000	1
0001	1
0010	1
0011	0
0100	1
0101	1
0110	1
0111	0
1000	1
1001	1
1010	1
1011	0
1100	0
1101	0
1110	0
1111	0

Как и положено в математике, логические функции можно преобразовать. Qucs позволяет проверить подобные трансформации.

Используя правила (законы) де Моргана, преобразуем исходную функцию y = *NOT* (x1x2 + x3x4).

Первое преобразование выглядит так:

Y = *NOT* (x1x2) **NOT* (x3x4)

Рис. 8.57. Таблица истинности для схемы на рисунке 8.55

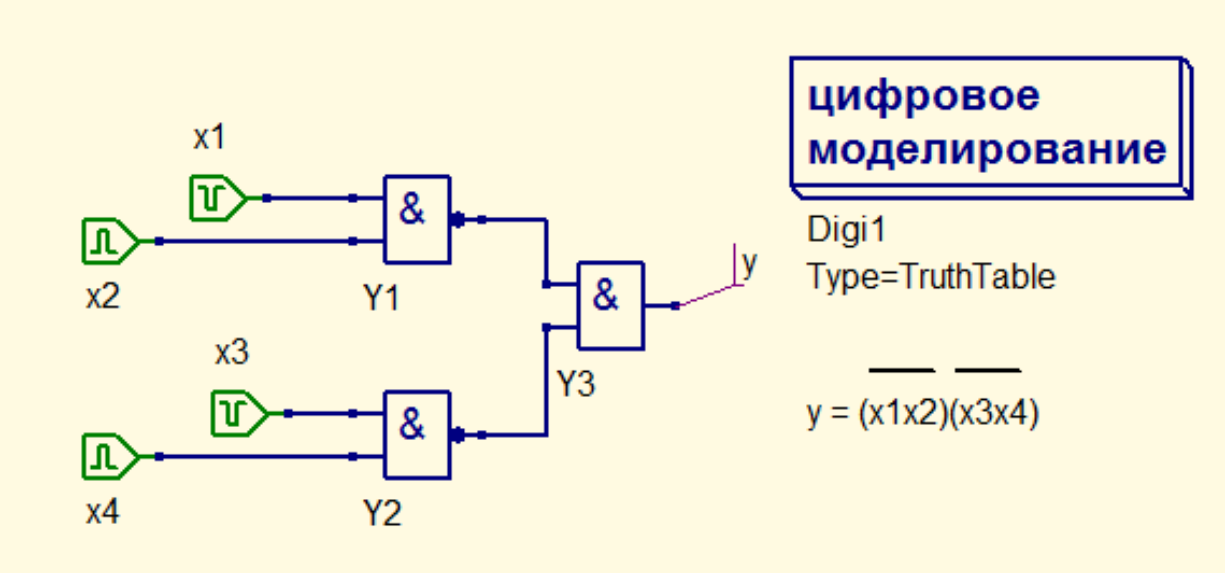


Рис. 8.58. Первое преобразование схемы

Моделирование дает возможность получить таблицу истинности и сравнить с исходной. Продолжив преобразование y = (*NOT* x1 + *NOT* x2)(*NOT* x3 + *NOT* x4), можно получить следующую схему.

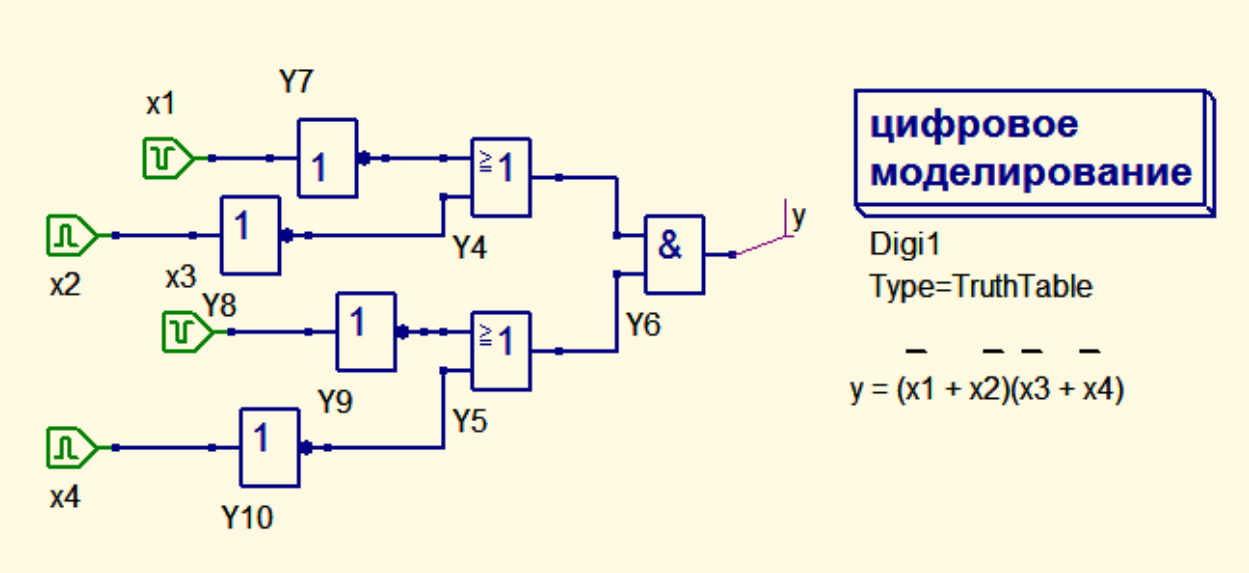
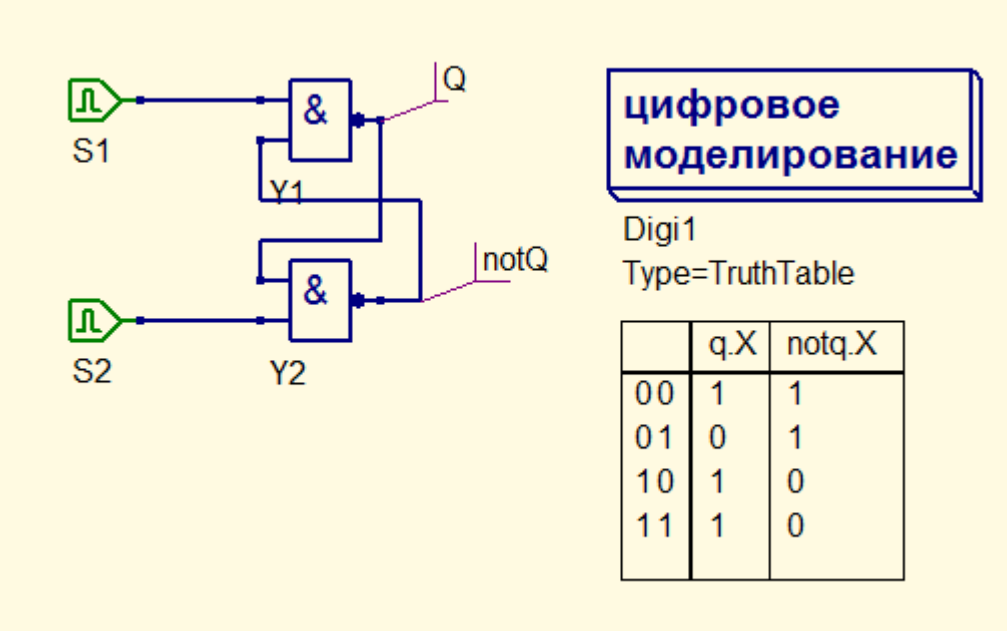


Рис. 8.59. Второе преобразование схемы

Таблица истинности для этой схемы, как можно убедиться, ничем не отличается от приведенной ранее. Это не значит, что, не доверяя де Моргану, мы проверили его законы. Мы проверили свое понимание того, как применить эти законы к элементам цифровой техники.

Используя базовый набор цифровых вентилей, можно создать любую (или, во всяком случае, многие) из микросхем набора серии 555.

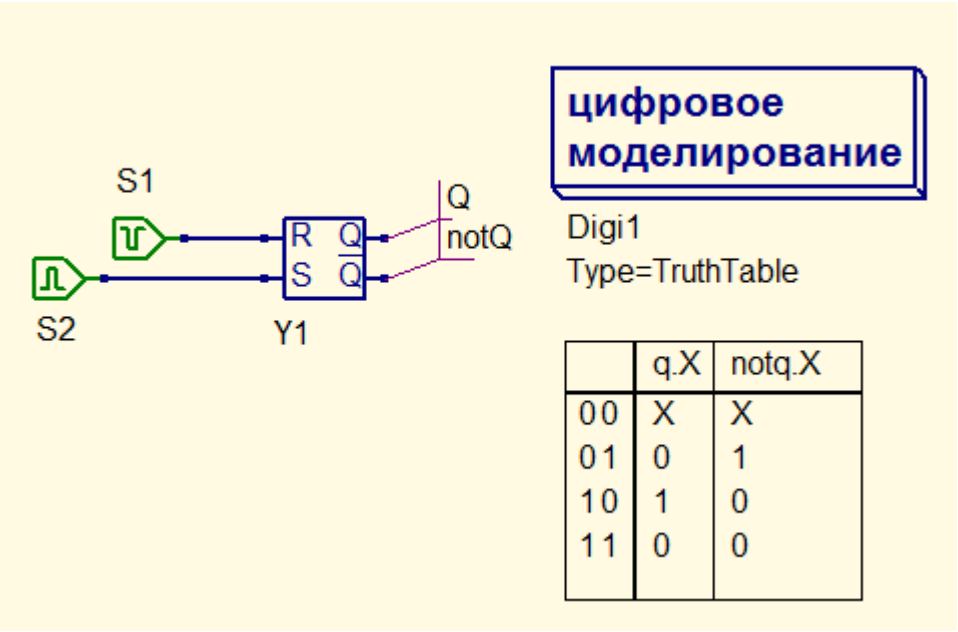
Вот простой пример такого построения.



	q.X	notq.X
0 0	1	1
0 1	0	1
1 0	1	0
1 1	1	0

Рис. 8.60. Создание RS-триггера из двух вентилей 2И-НЕ

Для разных типов цифровых микросхем, TTL или KMOS, существую RS-триггеры, работа которых определяется типом базовых элементов. Отличаются они возможностью одновременной подачи двух активных уровней на входы, и поведением, когда это событие происходит. Вот пример работы RS-триггера.



	q.X	notq.X
0 0	X	X
0 1	0	1
1 0	1	0
1 1	0	0

Рис. 8.61. Таблица истинности RS-триггера

Активный уровень, как это следует из вида символа и таблицы истинности, это уровень логической «1», а крестики в первой строке отображают тот факт, что до подачи одного из сигналов с активным уровнем состояние триггера не определено.

Для цифровых устройств важно не только определить логику работы, но и видеть динамику процессов в цепях. Этим целям служит временная диаграмма.

В Qucs пока нет счетчиков, достаточно часто использующихся в устройствах: это и разного рода таймеры, и электронные часы и делители частоты. Во второй главе на рисунке 2.51 приводилась схема двоичного счетчика. Давайте еще раз присмотримся к его диаграмме, используя либо созданный нами счетчик для **Библиотеки компонентов**, либо повторив схему еще раз.

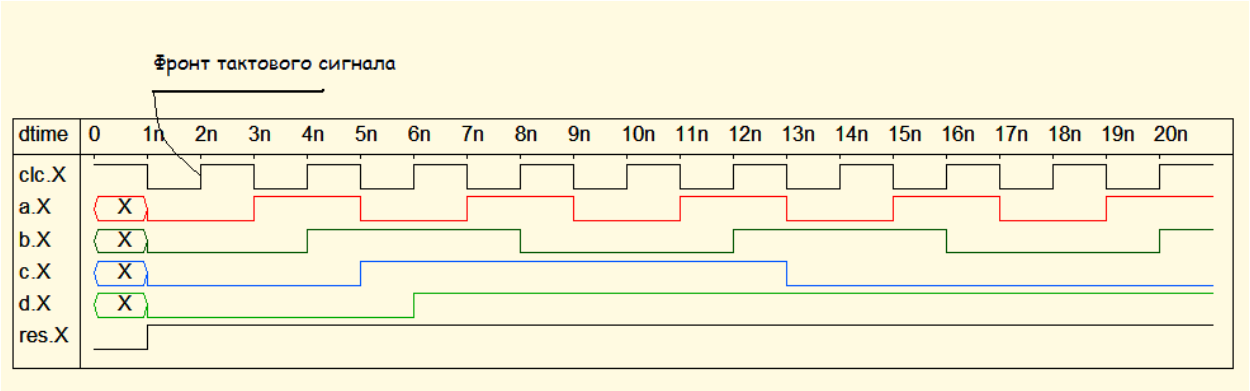


Рис. 8.62. Временная диаграмма счетчика на рисунке 2.51

Частота тактового сигнала делится на два на каждом из выходов, относительно предыдущего, но меня настораживает тот факт, что первый выход не переключается в высокое состояние по переднему фронту тактового сигнала, тогда как из рисунка D-триггера следовало ожидать именно такое поведение компонента.

В цифровых компонентах есть предыдущая версия D-триггера с прямым выходом. Используем этот компонент, добавив инвертор для получения инверсного выхода.

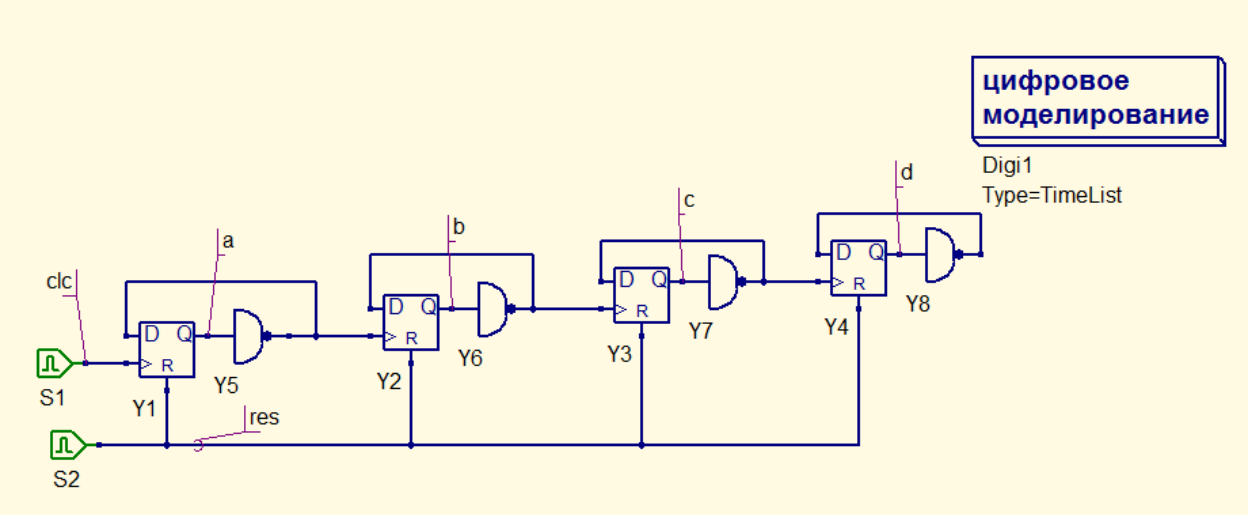


Рис. 8.63. Вторая схема счетчика

И посмотрим на вторую временную диаграмму.

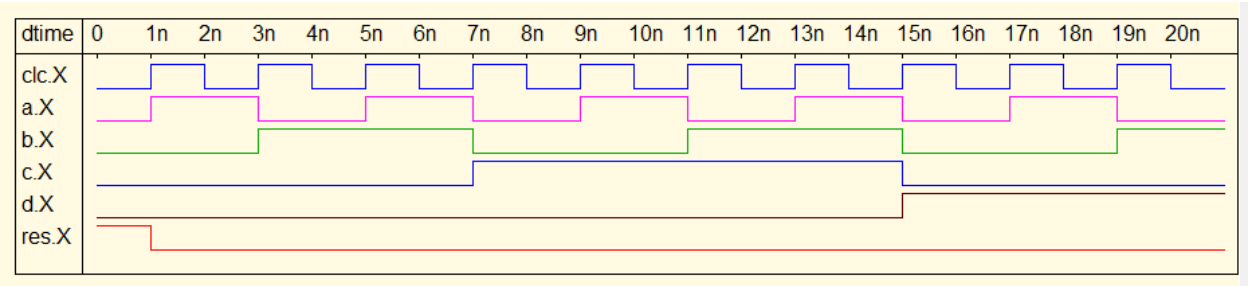


Рис. 8.64. Временная диаграмма предыдущей схемы

Теперь выход меняет свое состояние с приходом переднего фронта тактового импульса.

Так временная диаграмма позволяет найти «нестыковки» и отыскивать ошибки при цифровом моделировании.

Глава 3. Применение Qucs

Обучение

В профессиональной подготовке специалистов компьютерные программы давно заняли надлежащее место. Первопроходцами, видимо, были программы, позволяющие писать программы. Для этого достаточно было простейшего текстового редактора, интерпретатора языка программирования, или компилятора и компоновщика, и компьютера. Обучение программированию без компьютера, как обучение плаванию без водоема: возможно, но не эффективно.

Почти одновременно с обучением программированию по специальности программист, программированию стали обучать специалистов других, связанных с наукой, областей знаний.

По мере развития компьютерной техники все больше и больше специалистов разных дисциплин оказывалось связано с работой за компьютером. И процесс обучения требовал не только обучения предмету, но обучению работе с компьютером, особенно с разного рода специализированными программами. Каждое учебное заведение стремилось выпускать специалистов, которые, покинув стены аудиторий, готовы были начать работу на современном уровне.

Если специализированные программы, применяемые на профессиональном уровне, не могут быть заменены другими, то сам учебный предмет никак не зависит от компьютерных программ. Было время, когда компьютеров еще не было, но это не мешало учебным заведениям каждый год выпускать специалистов, трудом которых появились и компьютеры, и программы.

На разных уровнях обучения предъявляются разные требования к глубине знаний; задачи, предлагаемые для закрепления теоретического материала в школе, отличны от институтских. Но и в высших учебных заведениях, где больше внимания уделяют математическому аппарату для описания электрических процессов и их физическому обоснованию, и в школе учащиеся имеют дело с одними и теми же объектами. А вопрос представления этих сущностей, определяется ранее полученными знаниями. В этом смысле Qucs - программа вполне пригодная для обучения в школе основам работы с электрическими цепями.

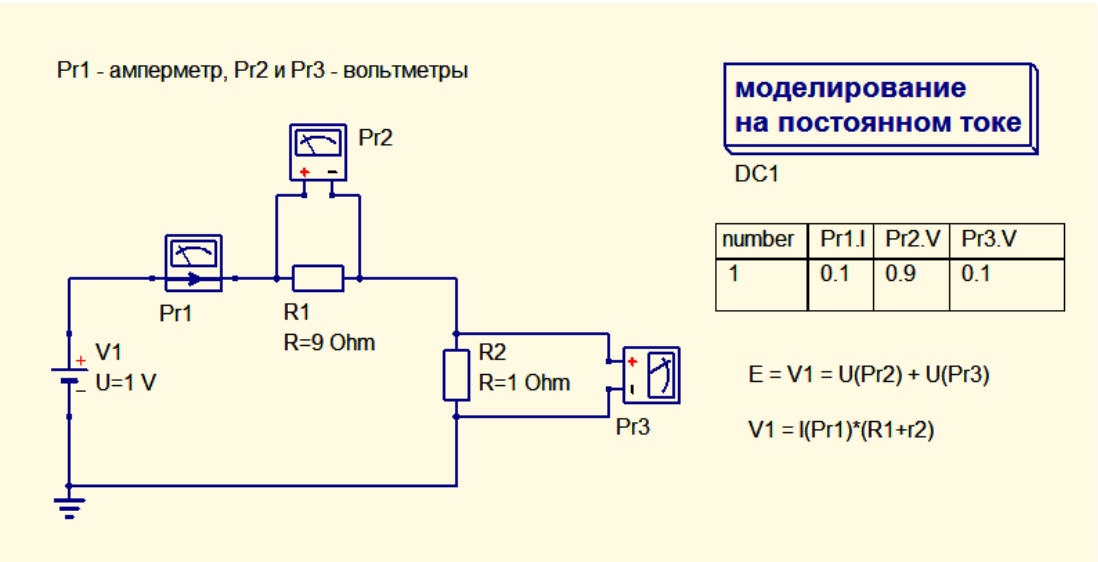


Рис. 9.1. Иллюстрация закона Кирхгоффа и закона Ома в Qucs

А в высшем учебном заведении вполне уместно сравнить расчетные данные с теми, что получены в программе Qucs.

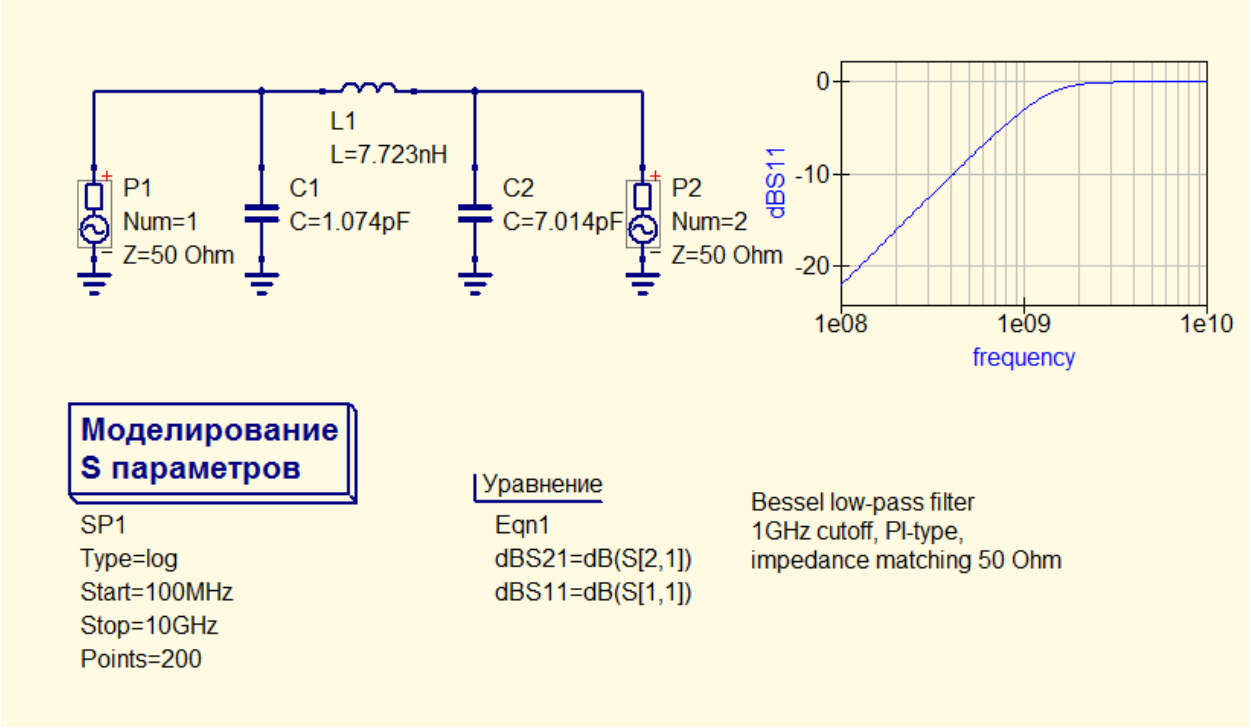


Рис. 9.2. Синтез фильтра в программе Qucs

Между этими двумя работами лежит не пропасть, а мост, который называется «Обучение». Учителю физики в школе трудно придумать опыт, используя доступные приборы, чтобы показать, что хотя конденсатор не пропускает постоянный ток, есть небольшой отрезок времени при подключении конденсатора к источнику постоянного тока, который называют переходным процессом, когда ток через конденсатор все-таки протекает. Много проще это сделать с помощью программы Qucs. При этом можно объяснить и разницу между идеальными элементами, как ниже, где и источник питания, и провода, и контакты выключателя – вся цепь имеет нулевое сопротивление, и ток в начальный момент времени, когда контакты замыкаются, должен быть равен бесконечности, а идеальный источник питания, по определению, может обеспечить бесконечный ток.

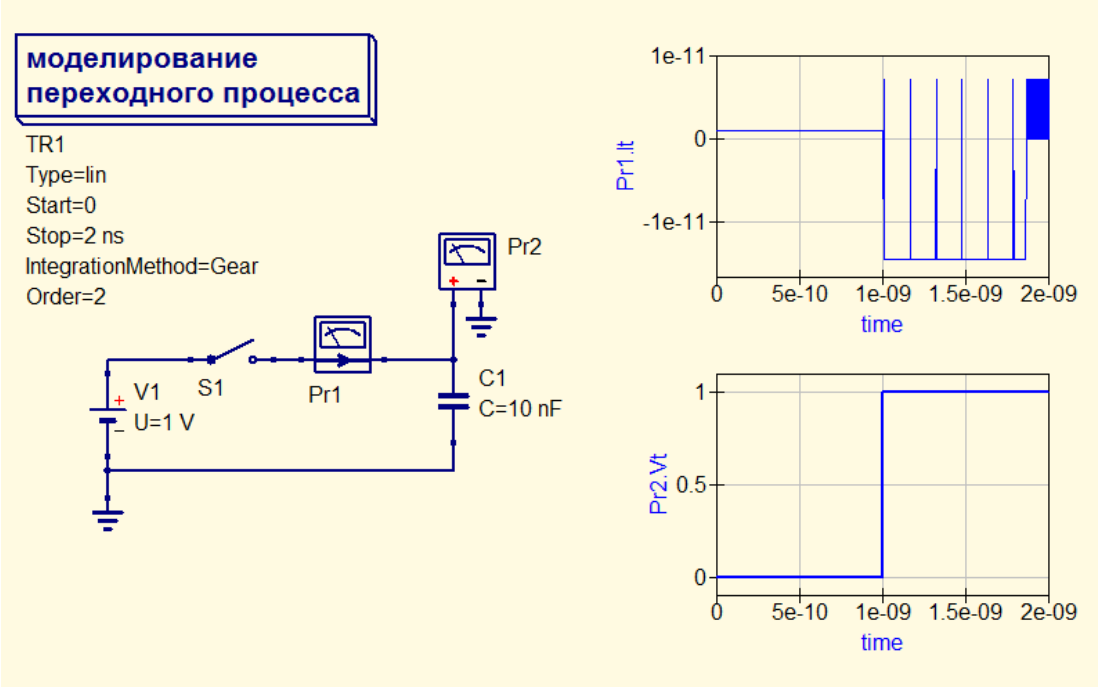
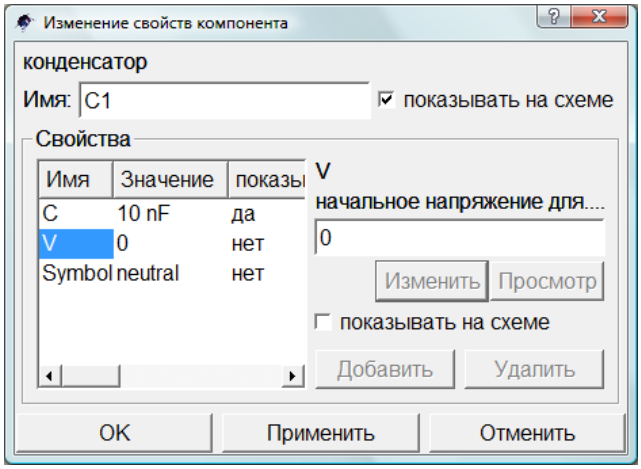


Рис. 9.3. Подключение конденсатора к идеальному источнику постоянного тока

Чтобы получить в Qucs результат моделирования, следует в свойствах конденсатора задать начальное напряжение равное нулю.



Без задания начального напряжения при моделировании не включается выключатель S1.

Рис. 9.4. Диалоговое окно свойств конденсатора

Изменив цепь добавлением резистора, который может отображать и внутреннее сопротивление источника, и сопротивление проводов, можно получить результат, который ближе к реальности.

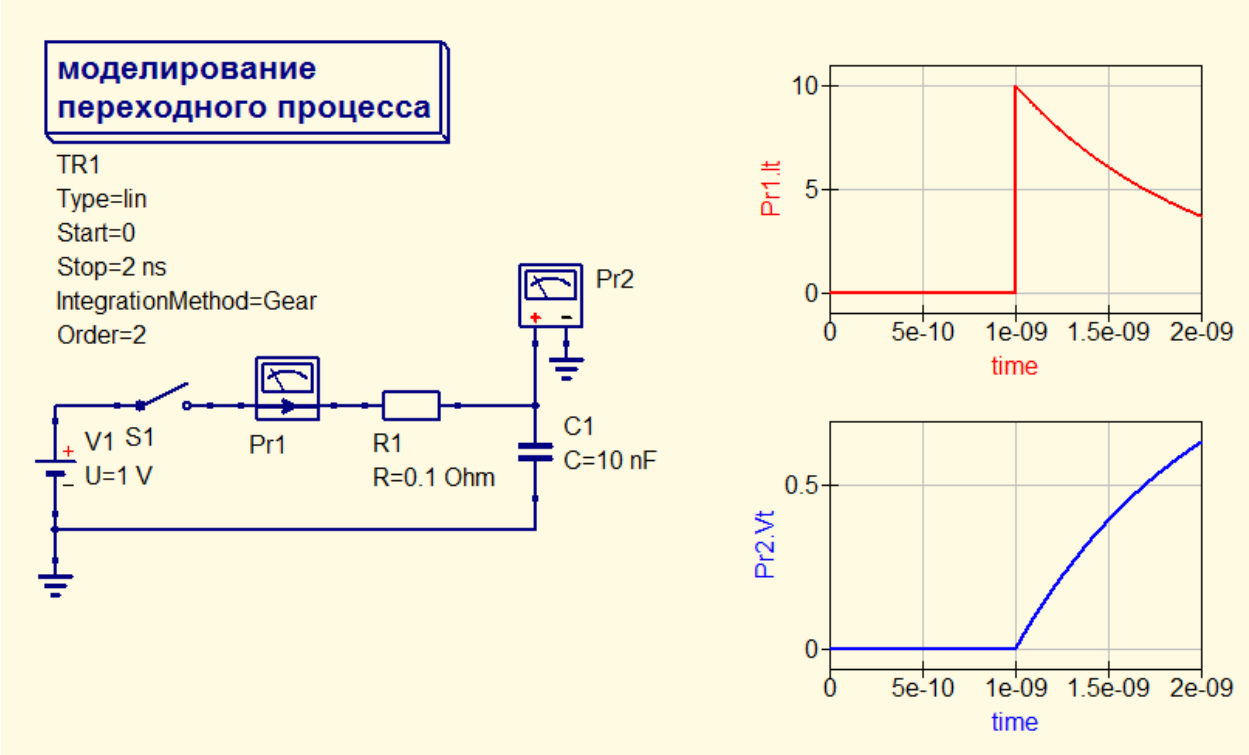


Рис. 9.5. Подключение конденсатора к источнику тока при наличии сопротивления

Удаление бесконечностей приводит к результатам, которые были бы получены при использовании реальных компонентов. Если еще больше приближать электрическую цепь к реальности, то нужно учесть, что и провода, и конденсатор могут иметь некоторую индуктивность, которая может изменить характер заряда конденсатора.

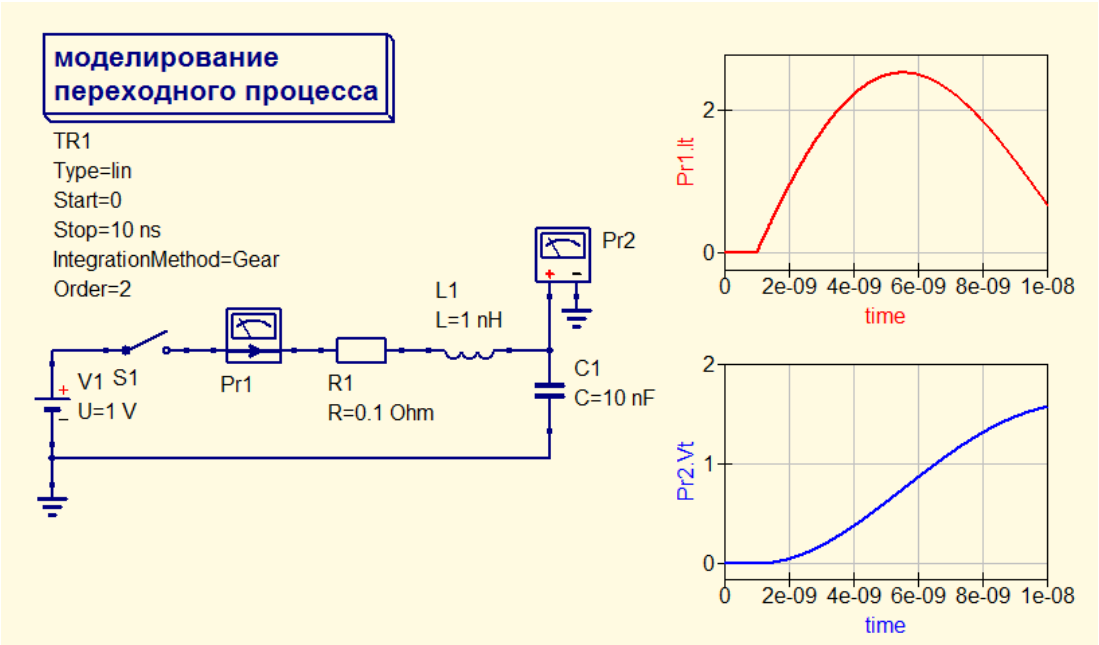


Рис. 9.6. Подключение конденсатора к источнику тока при наличии индуктивности

А если индуктивность ввести искусственно? То можно говорить о другом использовании Qucs.

Исследование

Далеко не всегда исследование – это организованный процесс, происходящий в научно-исследовательских лабораториях и институтах. Каждый раз, когда вы задаетесь вопросом, на который у вас нет готового ответа, вы приступаете к исследованию. Так что же относительно индуктивности?

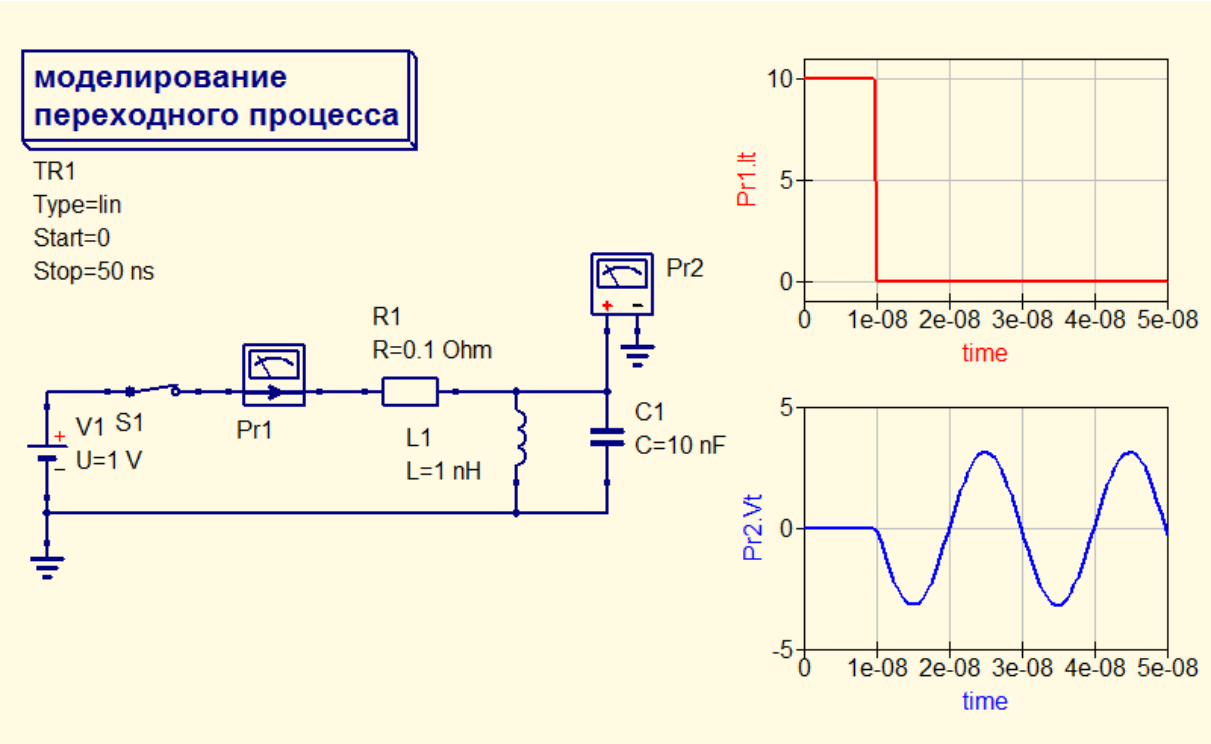


Рис. 9.7. Подключение LC цепи к источнику постоянного тока

Изменяя характер включения новой электрической цепи, заменив включение выключением, можно перейти к вопросу появления колебаний в LC-цепи.

А если катушка индуктивности намотана проводом, то она должна иметь некоторое сопротивление. Введение сопротивления в идеальную цепь при первом эксперименте изменило характер протекающих в ней процессов. Не произойдет ли этого и теперь?

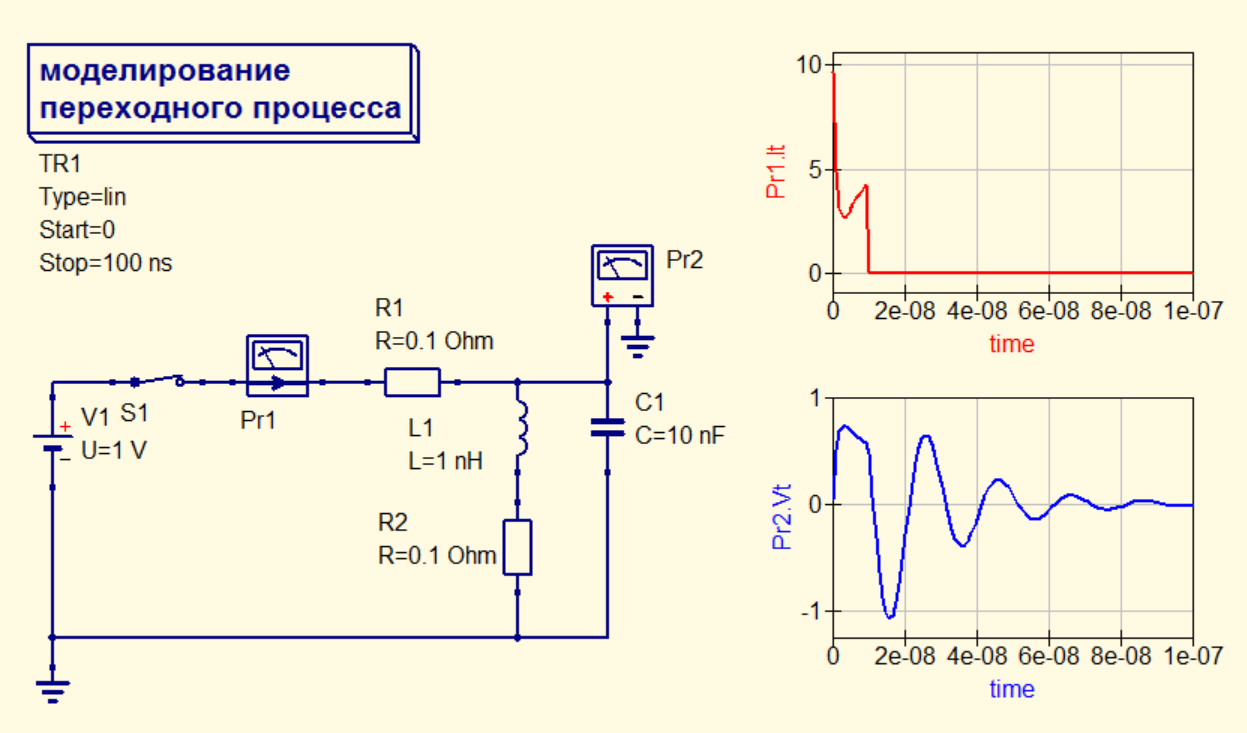


Рис. 9.8. Изменение характера колебаний с учетом сопротивления намоточного провода

Хорошо, а если заменить выключатель и источник таким источником, который периодически будет включать и выключать электрическую цепь?

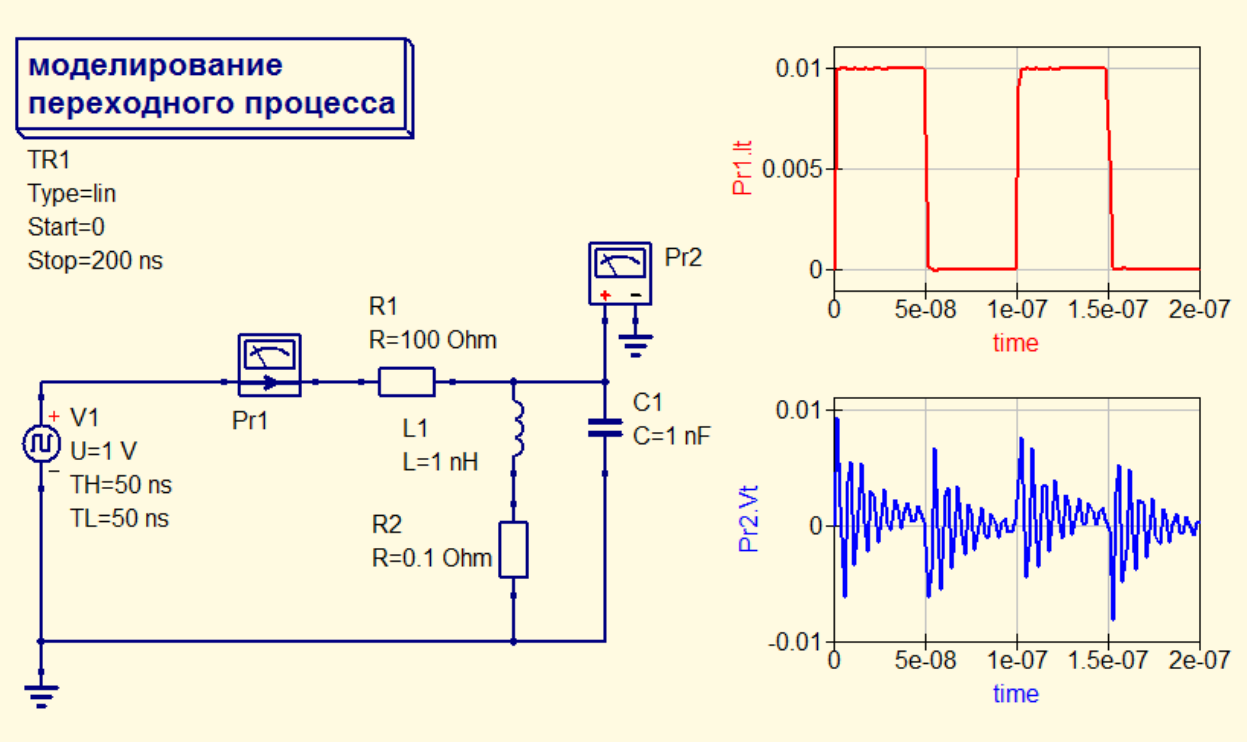


Рис. 9.9. Замена источника постоянного тока импульсным

И не следует думать, что исследования дают ответы только на «детские» вопросы.

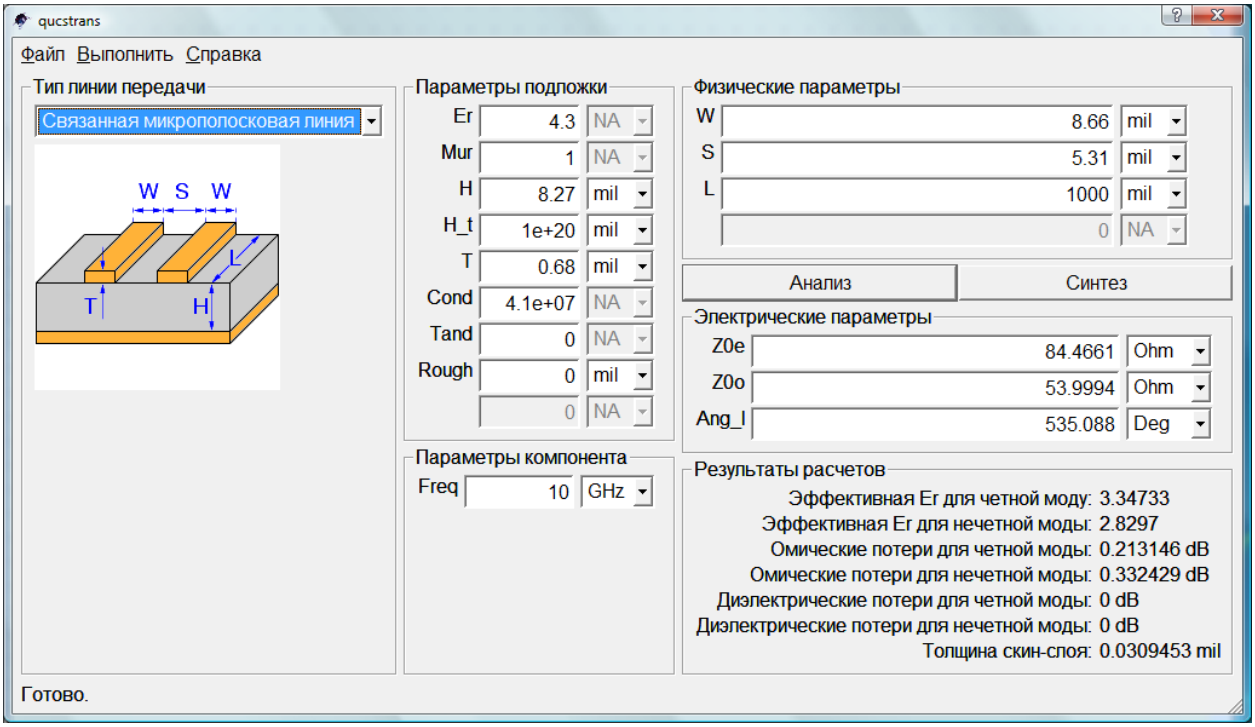


Рис. 9.10. Встроенный в программу Qucs расчет линий

Программа, например, имеет достаточно большой арсенал компонентов для подготовки специалистов в области радиоэлектроники.

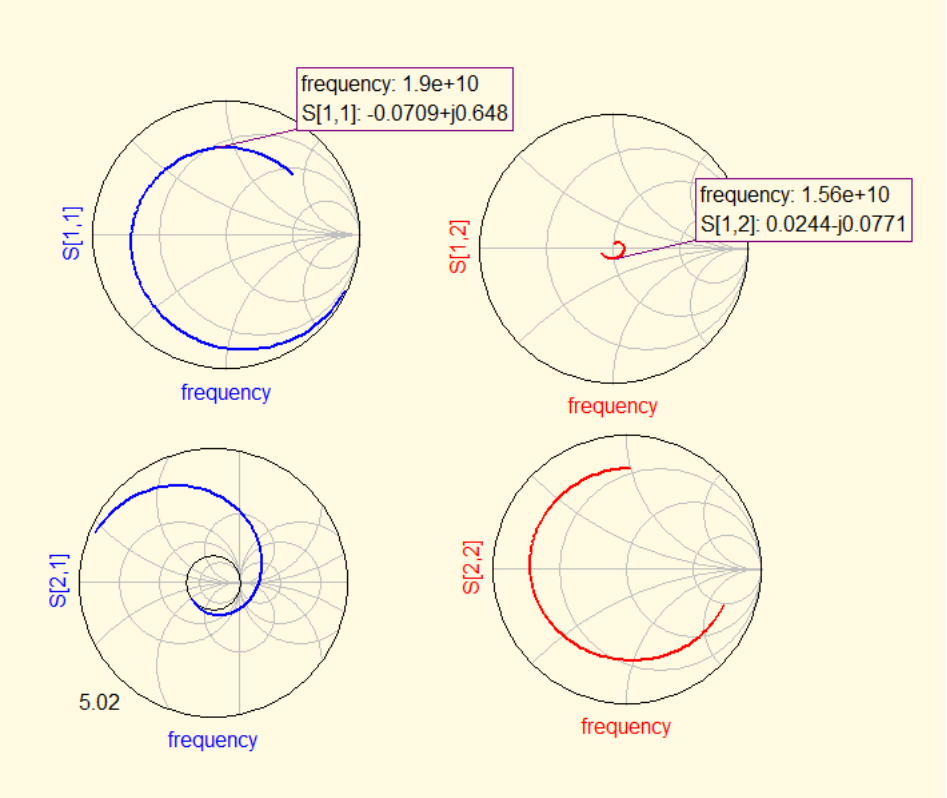


Рис. 9.11. Пример получения S-параметров

И схема, параметры которой приведены выше.

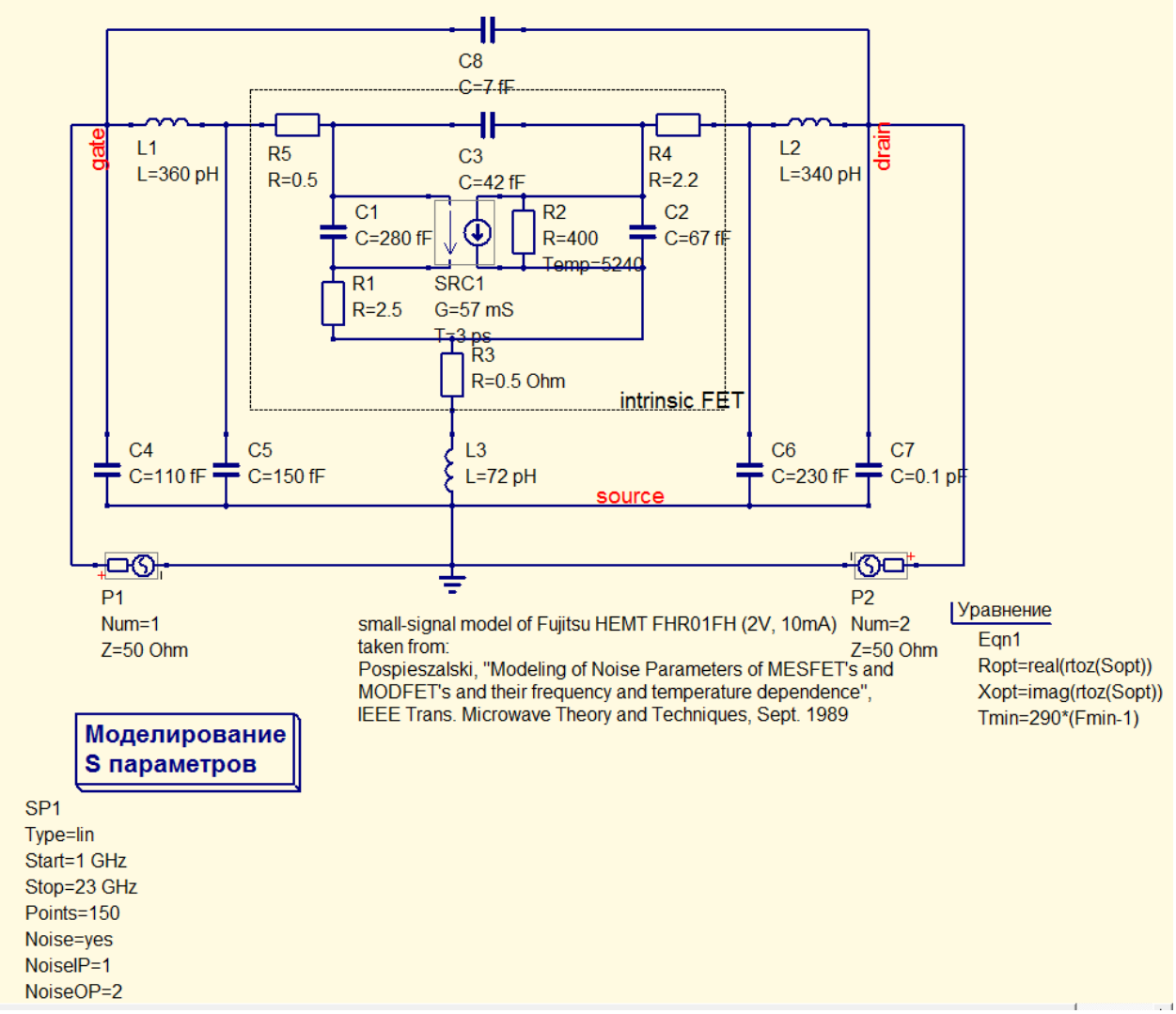


Рис. 9.12. Эквивалентная схема полевого транзистора

Для удобства ведения документации: отчетов, написания статей, где интенсивно используются формулы и математические выражения, часто добавляются графики – программа Qucs предлагает поддержку формата, разработанного специально для этой цели.

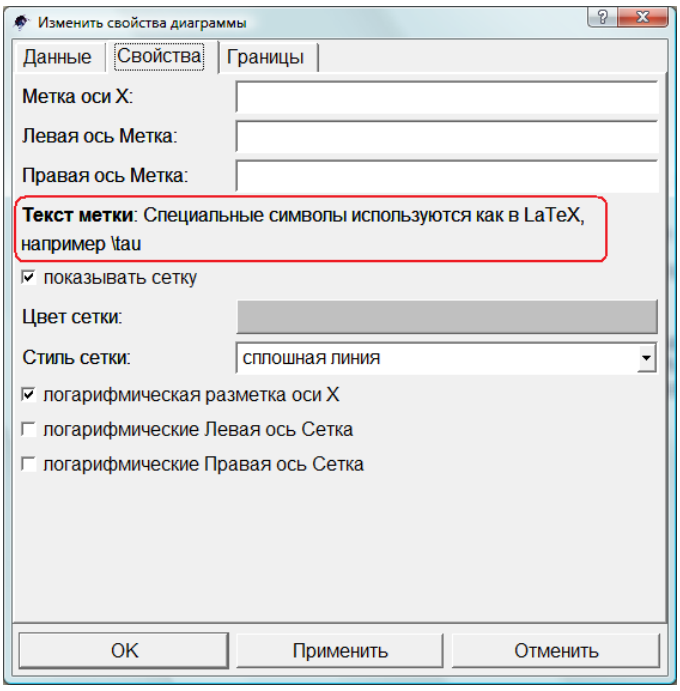


Рис. 9.13. Поддержка формата LaTeX

А диаграммы можно вывести в файл, как рисунок.

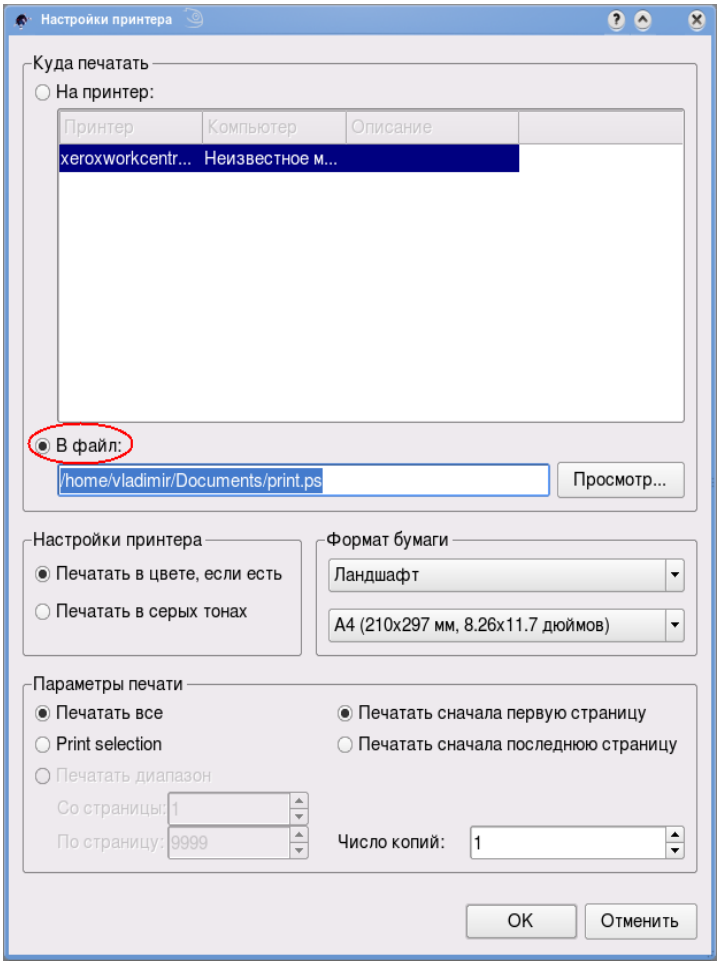


Рис. 9.14. Вывод диаграммы в файл

Практика

Специалисты, занимающиеся разработкой электронных устройств, едва ли согласятся сменить привычное программное обеспечение на нечто, что предстоит осваивать, проверять и т.д.

Но с электроникой связаны не только разработчики. При эксплуатации оборудования часто возникает необходимость в его налаживании или выявлении возникающих проблем. Иногда для проверки можно снять функциональный модуль, чтобы проверить его работу «на столе». Но не всегда. Проверка «на месте» всегда опасна даже при соблюдении и правил электрической безопасности, и правил эксплуатации – любая ошибка, даже не очевидная, может повлечь выход из строя компонентов устройства. Вместе с тем, покупка дорогостоящих программ, которые могли бы помочь в этих случаях, для предприятия явно не покажется обоснованным решением.

Программа Qucs дает возможность получить ответы на многие вопросы, возникающие, например, при ремонте электронных устройств. Например, есть подозрение, что в функциональном узле, показанном ниже, изменился режим работы, что повлекло за собой общую ошибку в работе устройства.

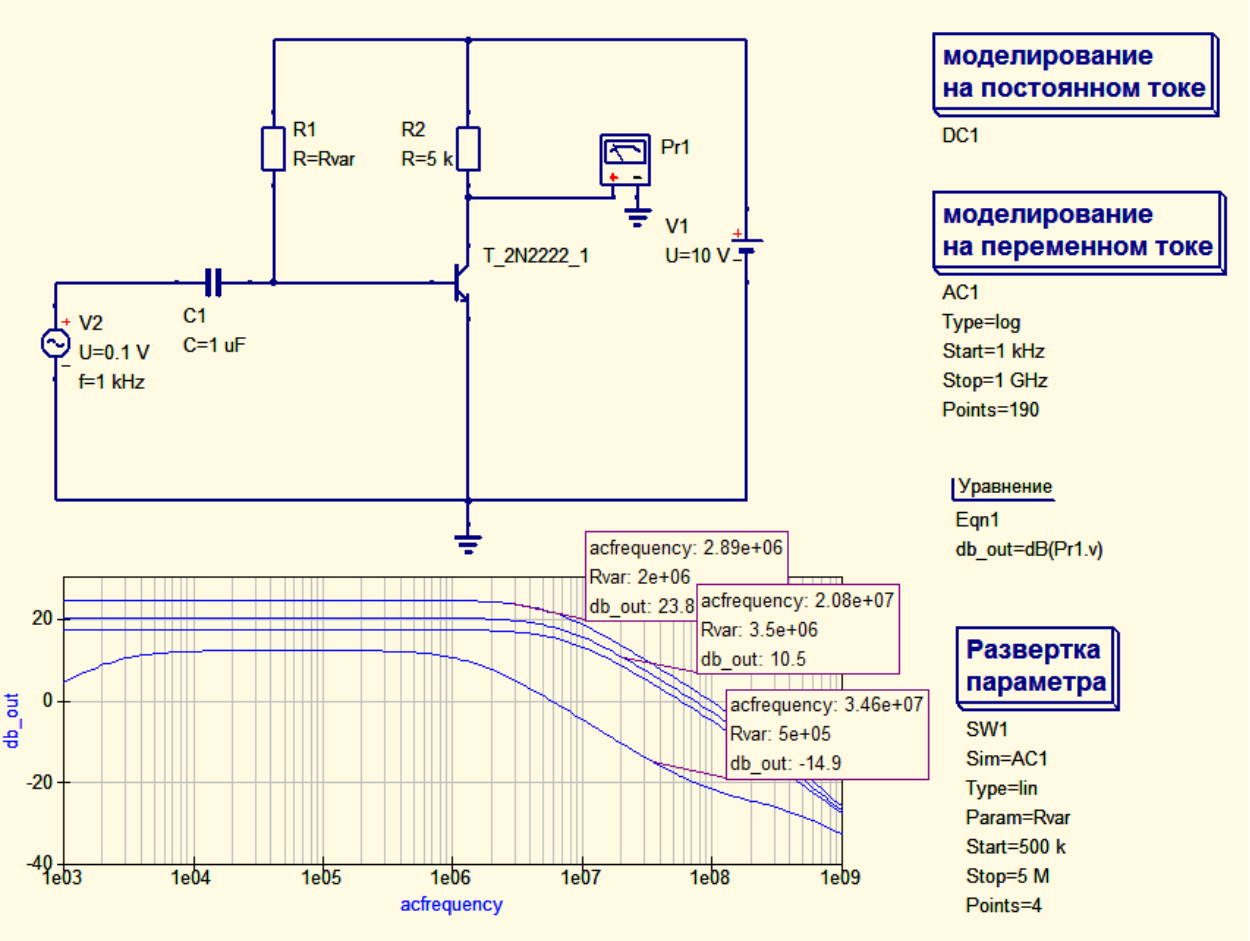


Рис. 9.15. Эксперимент с функциональным узлом

Резисторы, запаянные на плате, не всегда при выходе из строя позволяют увидеть дефект даже при тщательном осмотре. А оборудование не всегда имеет полную документацию, дающую ответы на любые вопросы, это относится и к схеме, и к спецификации. Измерение величины резистора в базовой цепи транзистора дает оценочную возможность для определения рабочей

точки, но не покажет, как выбор рабочей точки скажется на амплитудно-частотной характеристике каскада усиления.

Применяя возможности программы Qucs, можно оценить влияние, например, величины резистора в базовой цепи транзистора на АЧХ. Аналогичная работа, даже если ее можно проделать без опасности выхода из строя оборудования, займет много времени и потребует измерительных приборов, которые могут отсутствовать.

Еще больше трудностей в практике эксплуатации возникает при длительном использовании устройств. Со временем достать запасные части становится все труднее. А для решения задачи модернизации следует продумать очень многое в рамках ограниченных возможностей по проверке каждого из вариантов. И в этом случае программное моделирование ситуации окажет хорошую поддержку начинаниям. Программа не только покажет, насколько новая схема по своим параметрам совпадает с прежней, но и даст ответы на вопросы: «А что произойдет, если…».

Например, при замене старых транзисторов, особенно германиевых, возникает необходимость в настройке для восстановления рабочей точки. Но замена может коснуться и других параметров. При использовании более высокочастотного транзистора может расшириться полоса пропускания каскада. Чаще всего это не сказывается отрицательно на работе устройства, но при этом меняется характер шумов. Если со старым транзистором схема была не чувствительна к наводкам из-за «естественного» среза на верхней рабочей частоте, то теперь эти наводки могут мешать работе устройства. Сравним частотные свойства каскада, схема которого приведена выше, при использовании разных транзисторов. Вы заменили старый транзистор современным 2N2222.

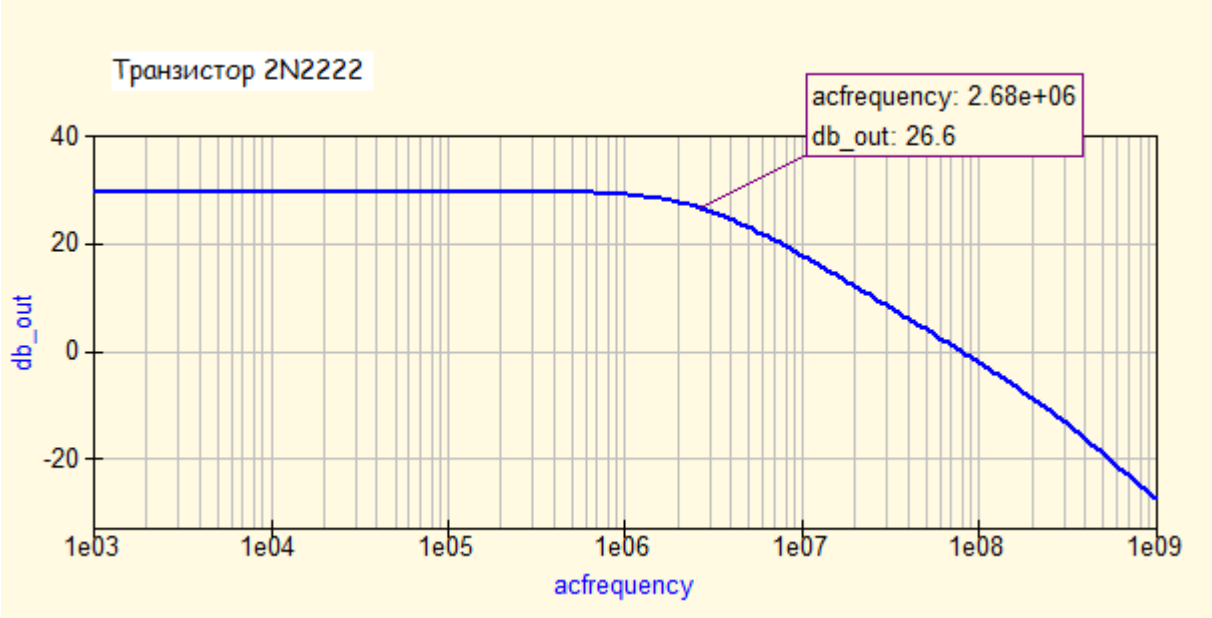


Рис. 9.16. АЧХ каскада с транзистором 2N2222

Верхняя граничная частота каскада 2,68 МГц. С прежним транзистором каскад имел полосу пропускания уже.

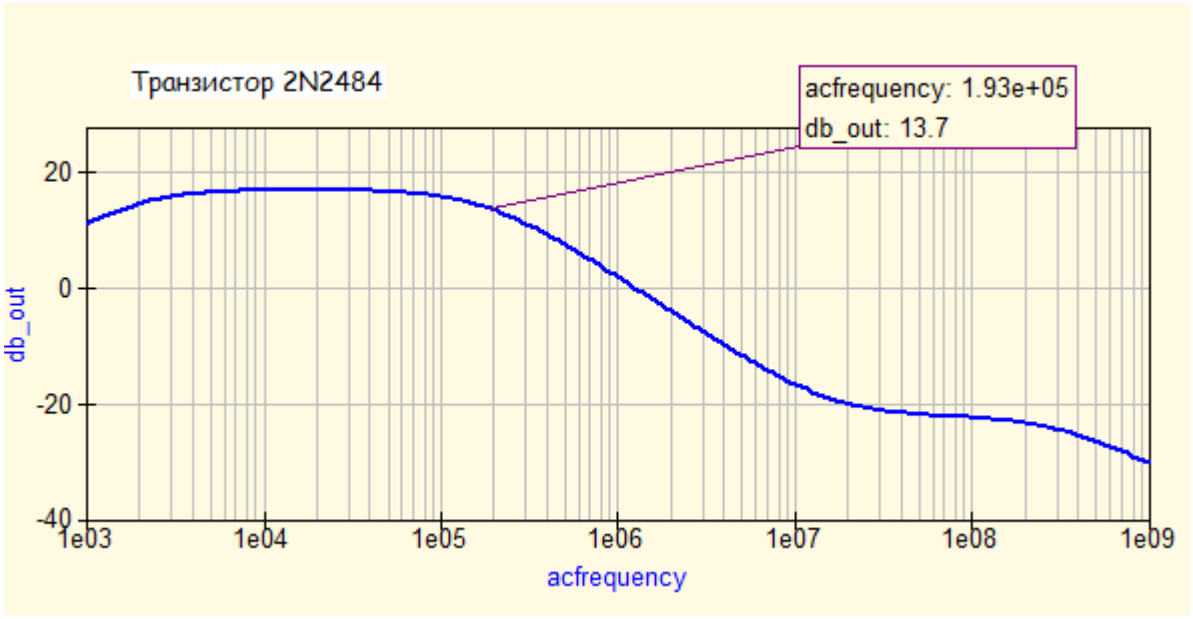


Рис. 9.17. АЧХ каскада с транзистором 2N2484

Если, зная конкретное устройство, вы понимаете, что расширение полосы пропускания может отрицательно сказаться на работе устройства, вы можете скорректировать АЧХ.

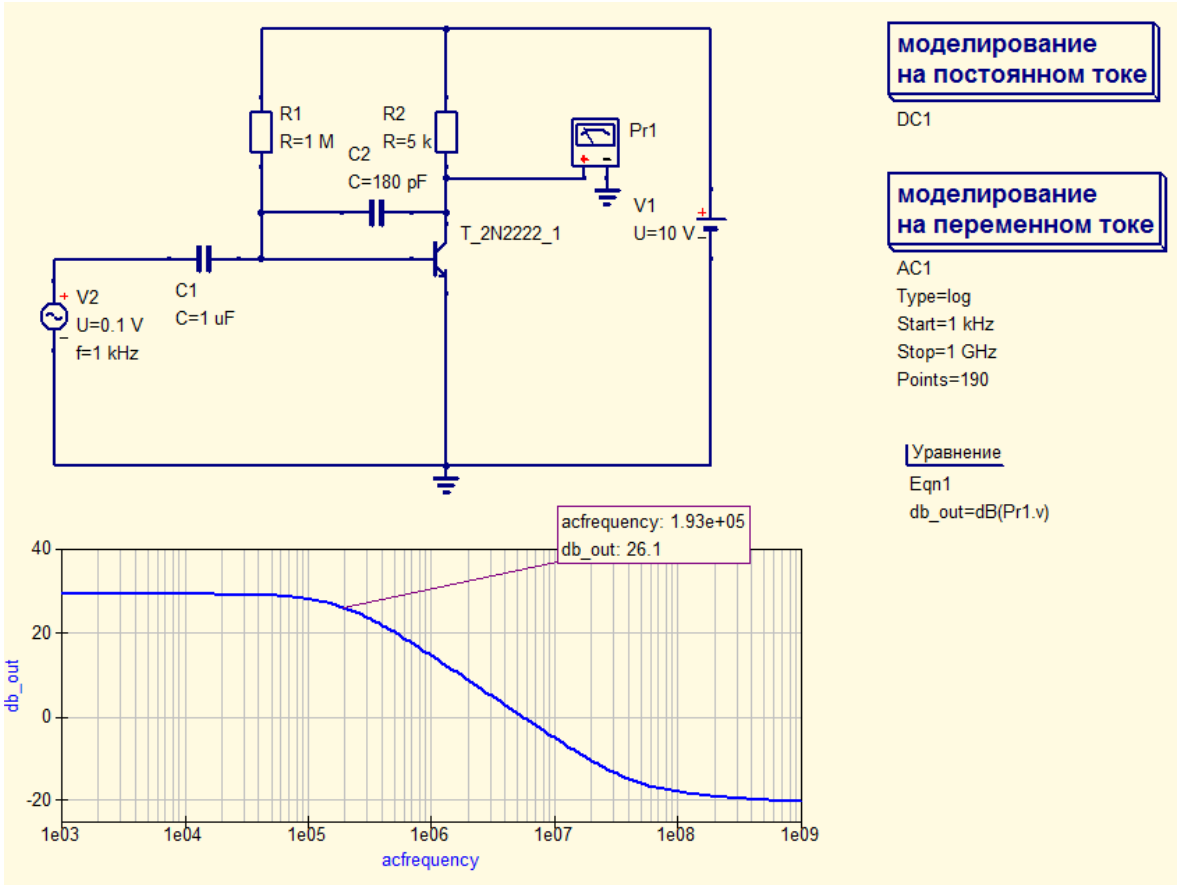


Рис. 9.18. Коррекция АЧХ

Без опыта каждодневных расчетов использование программы избавит и от лишних ошибок.

Часть 3. Возможности Qucs

Глава 1. Некоторые расширения программы

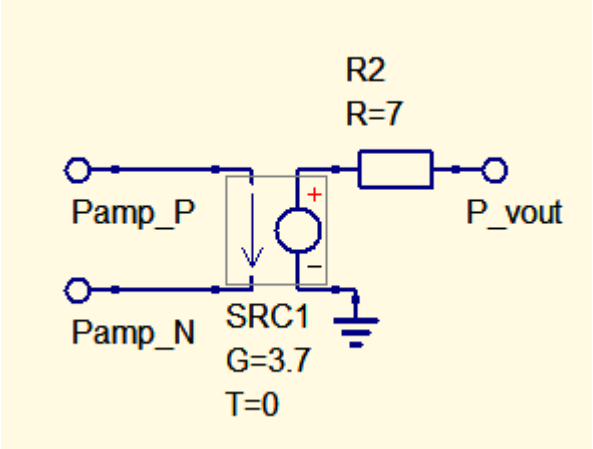
В предыдущих главах мы рассмотрели, конечно, не все компоненты и не все возможности программы. Начнем с компонентов.

В разделе **источники** закладки **Компоненты** панели навигации есть ряд источников, о которых ранее только упоминалось. Остановимся на них подробнее.

Источники

В группе источников есть четыре управляемых источника: источник тока, управляемый напряжением; источник тока, управляемый током; источник напряжения, управляемый напряжением; источник напряжения, управляемый током.

Начнем с примера. При создании модели таймера 555, достаточно сложного устройства для моделирования в силу того, что используются смешанные компоненты, цифровые и аналоговые, в качестве усилителя в составе микросхемы используется управляемый источник.



Применение такого решения значительно упрощает симуляцию схемы.

Рис. 10.1. Использование управляемого источника

Что же такое, управляемые источники?

Они рассматриваются в теории электрических цепей и представляют результат идеализации реальных транзисторных или ламповых усилителей. При работе этих усилителей в линейном режиме напряжение на выходе усилителя пропорционально входному напряжению или току.

Для управляемых источников важно помнить, что входное сопротивление управляемых напряжением источников бесконечно велико (идеализация, к которой могут приближаться, например, входное напряжение каскада на полевом транзисторе), а у источника, управляемого током, равно нулю.

Управляемый источник можно рассматривать как разновидность линейного четырехполюсника, который может быть пассивным или активным. В режиме малого сигнала и в линейной области любой четырехполюсник можно представить в следующем виде:

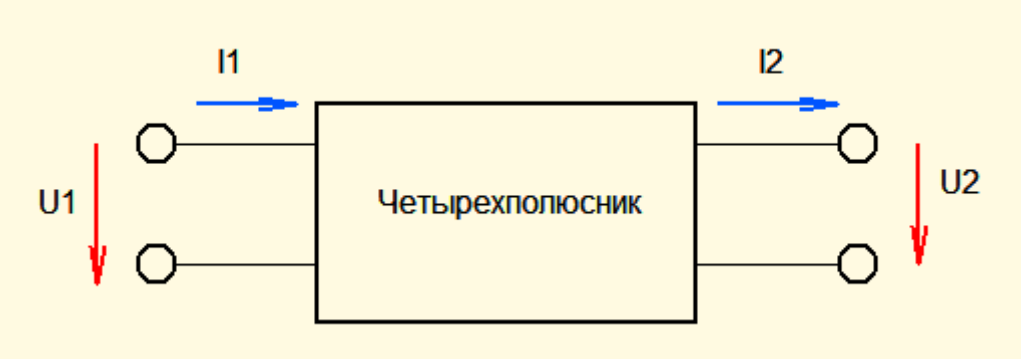


Рис. 10.2. Изображение четырехполюсника

Нас в первую очередь интересуют токи и напряжения. На рисунке I1 и U1 – входной ток и напряжение, в общем случае комплексные величины; I2 и U2 – выходные ток и напряжение. Для линейного преобразования можно составить системы уравнений, где одна пара будет считаться известной, а вторая неизвестной. Коэффициенты при неизвестных в уравнениях образуют четыре параметра, которые будут называться h-параметрами, y-параметрами и т.д., в зависимости от выбора системы уравнений. При матричном подходе к решению уравнений эти параметры образуют коэффициенты матрицы преобразования. Например, можно записать систему уравнений:

U1 = h11*I1 + h12*U2

I2 = h21*I1 + h22*U2

Параметр h11 характеризует входное сопротивление, h12 – обратную связь, h21 – усиление, а h22 – выходное сопротивление.

На рисунке 5.2 показан пример моделирования S-параметров. Но в свойствах диаграммы можно выбрать отображение H-параметров.

Использование маркеров на диаграммах дает возможность определить параметры для заданной частоты. Обычно получение параметров малого сигнала, скажем, для эквивалентной схемы транзистора сопряжено с рядом трудностей. В учебном процессе использование программы в этом случае дает серьезный выигрыш во времени и усилиях.

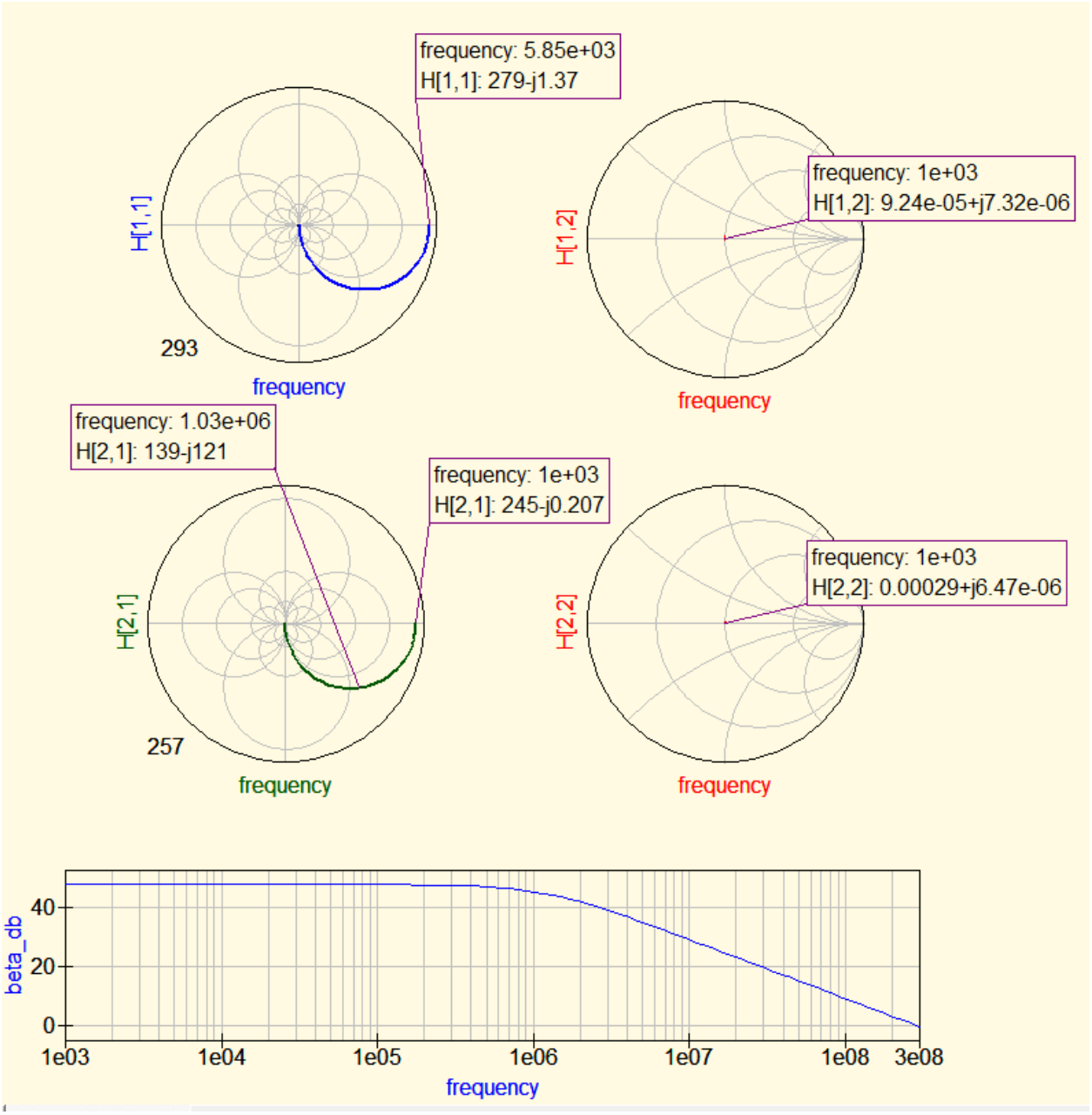


Рис. 10.3. Получение h-параметров с помощью диаграмм Смита

Между компонентами группы **дискретные** есть один, о котором уже упоминалось. Это порт подсхемы. На инструментальной панели есть кнопка добавления вывода. Оба варианта предназначены к добавлению компонента, который помогает организовать схему так, чтобы ее легче было прочитать, а, значит, понять, чтобы ее легче было отладить.

Подсхемы

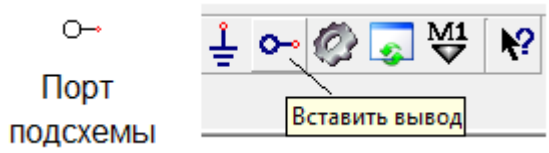


Рис. 10.4. Порт подсхемы

Рассмотрим простой пример, когда создание подсхемы удобно для работы. Исследуем частотные свойства схемы, составленной из резисторов и конденсаторов, применив **анализ на переменном токе**.

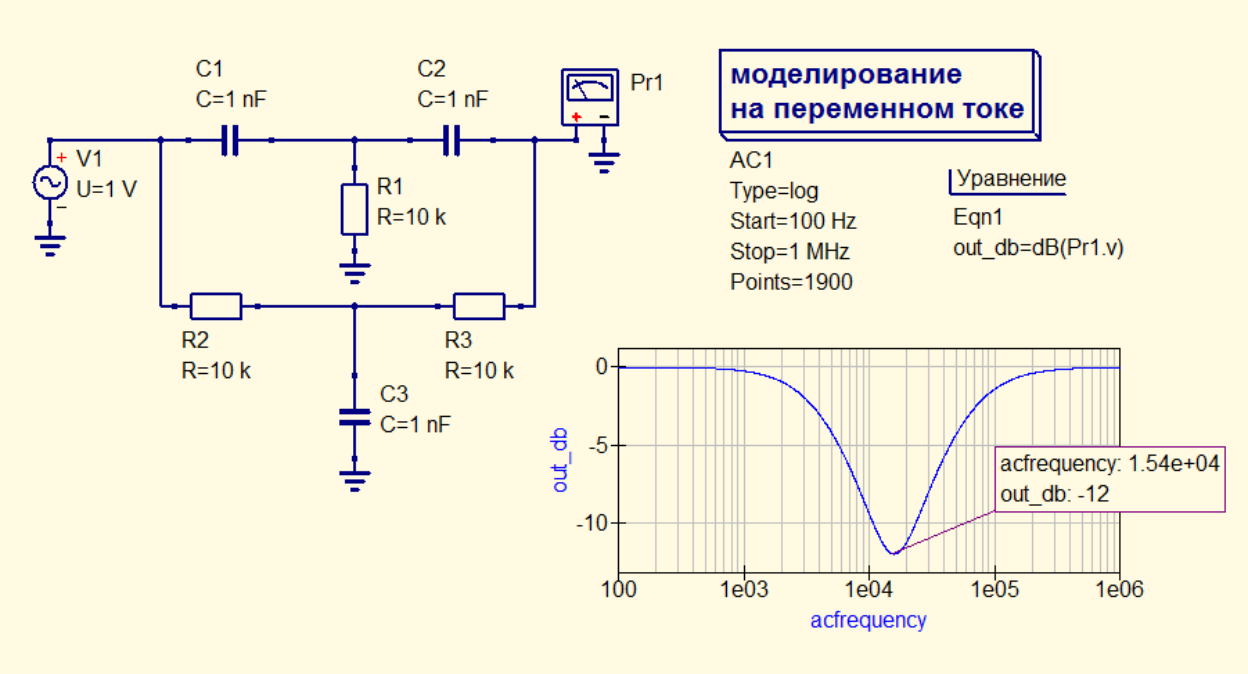


Рис. 10.5. Исследование частотных свойств электрической цепи

Как видно из амплитудно-частотной характеристики электрической цепи – это режекторный фильтр. А маркер показывает частоту, равную 15 кГц. Уравнение, добавленное в рабочее поле, позволяет отобразить кривую в относительных единицах, децибелах.

Заменим источник напряжения и измерительный прибор портами.

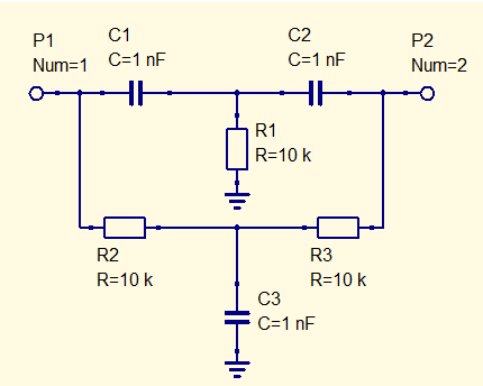


Рис. 10.6. Подключение портов к электрической цепи

Открыв диалоговое окно порта – двойной щелчок по нему левой клавишей мышки – мы можем изменить свойства порта. Хотя схема симметрична, обозначим левый порт как входной, а правый как выходной. На рисунке отмечена кнопка, выводящая список возможных вариантов.

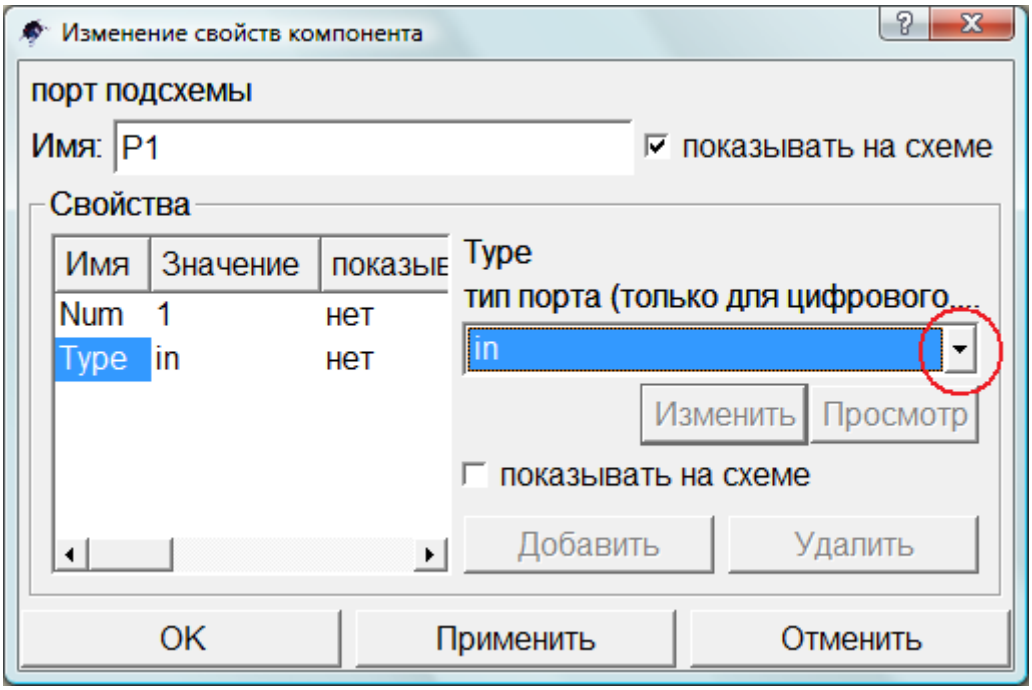


Рис. 10.7. Диалоговое окно свойств порта

Вид портов изменился. Левый обозначен как входной, правый – выходной.

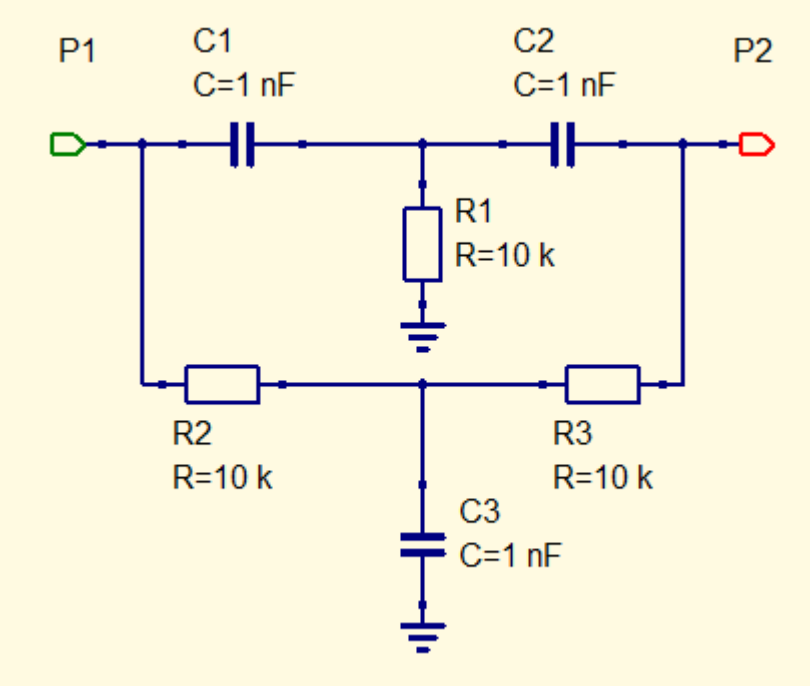


Рис. 10.8. Изменение вида портов

Теперь выберем в основном меню в разделе **Файл** пункт ***Изменить обозначение схемы*** (можно использовать и функциональную клавишу на клавиатуре F9).

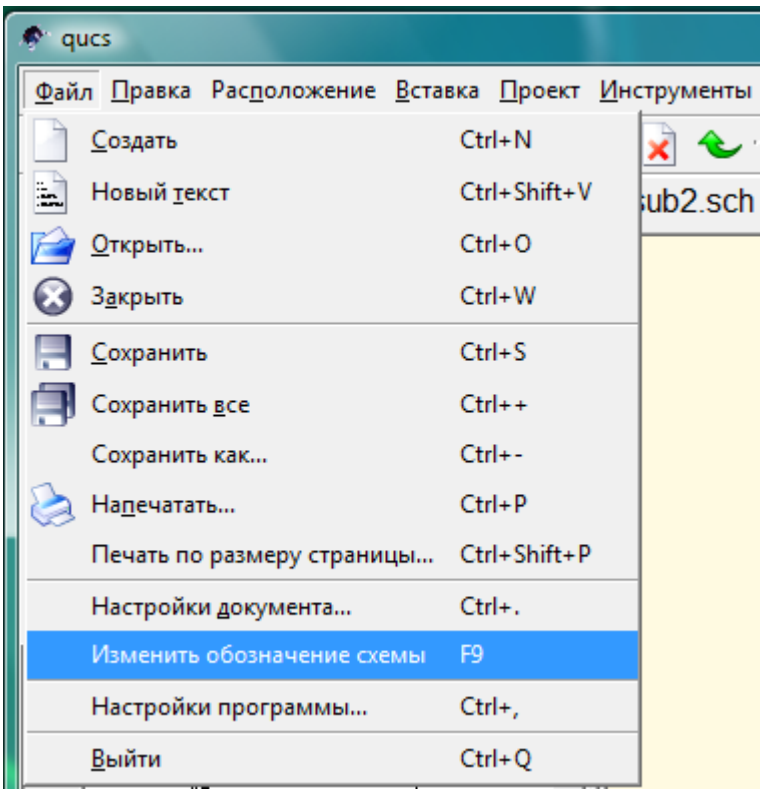
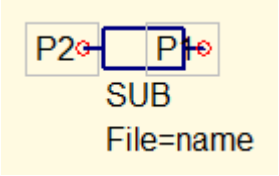


Рис. 10.9. Команда изменения обозначения схемы

Появившийся вид схемы близок к тому, что мы хотели бы видеть.



Однако порт P1… Подцепим его мышкой (подведем курсор к элементу, нажмем и удержим левую клавишу мышки) и «оттащим» в сторону.

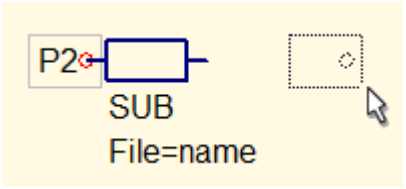


Рис. 10.11. Изменение вида порта P1

Оставив его в новом месте, он будет выделен, щелкнем по нему правой клавишей мышки, чтобы из выпадающего меню выбрать команду **Отобразить относительно оси Y**.

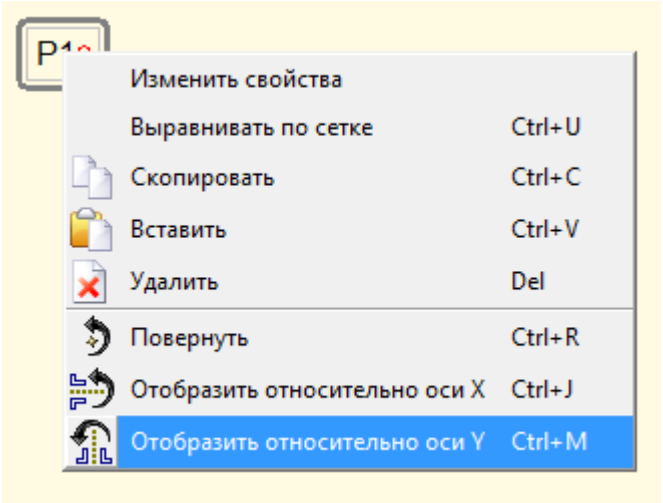


Рис. 10.12. Выпадающее меню работы с компонентами

После отражения объекта мы вновь «подцепим» его мышкой и вернем к выводу, где порт был до этого, но уже в нужном нам виде.

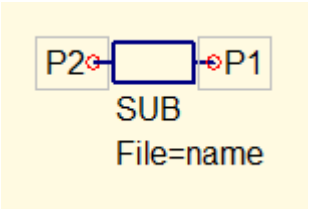


Рис. 10.13. Окончательный вид фильтра

Приведя схему к окончательному виду, который нам потребуется в дальнейшем, мы можем приступить к следующему эксперименту.

Создадим новый проект, добавим операционный усилитель и исследуем его частотные свойства, добавив в цепь отрицательной обратной связи ранее собранный фильтр.

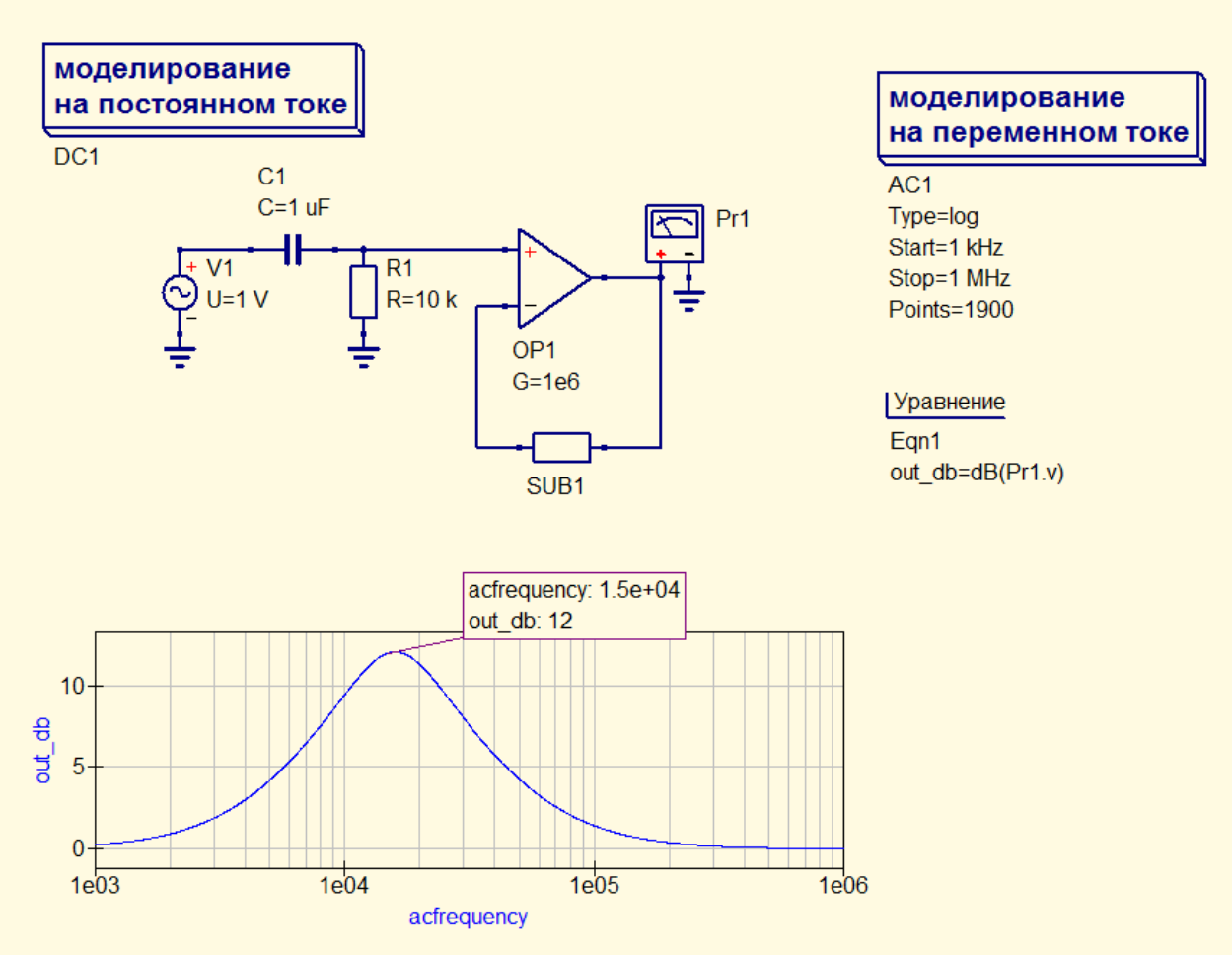
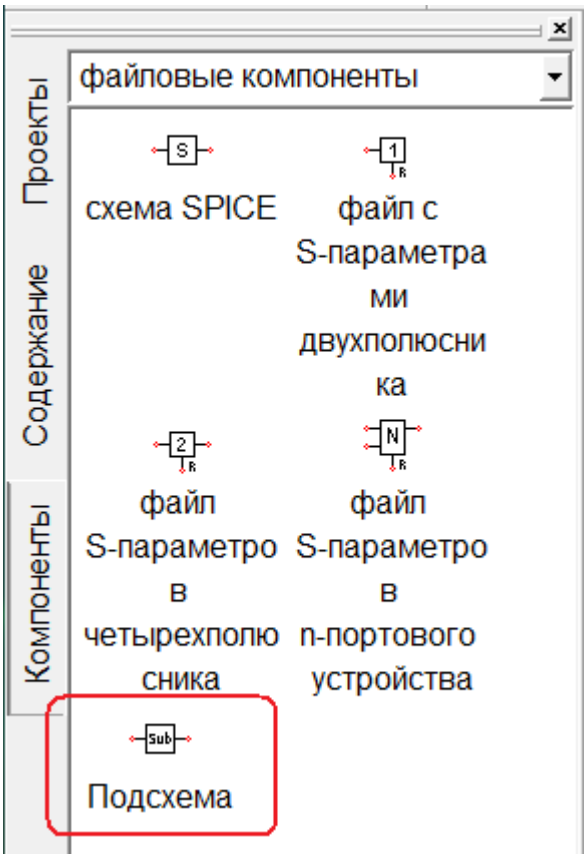


Рис. 10.14. Исследование частотных свойств ОУ с фильтром

Как мы включили фильтр в схему?

В группе компонентов **файловые компоненты** на закладке **Компоненты** панели навигации есть такой компонент, который называется **Подсхема**.



Достаточно перенести его в рабочее поле чертежа и соединить с нужными нам узлами схемы.

Конечно, пока мы не указали файл подсхемы, мы не сможем работать с этим компонентом.

Двойной щелчок мышкой по нему открывает диалоговое окно, где главный элемент – это указатель файла подсхемы.

Рис. 10.15. Компонент **Подсхема**

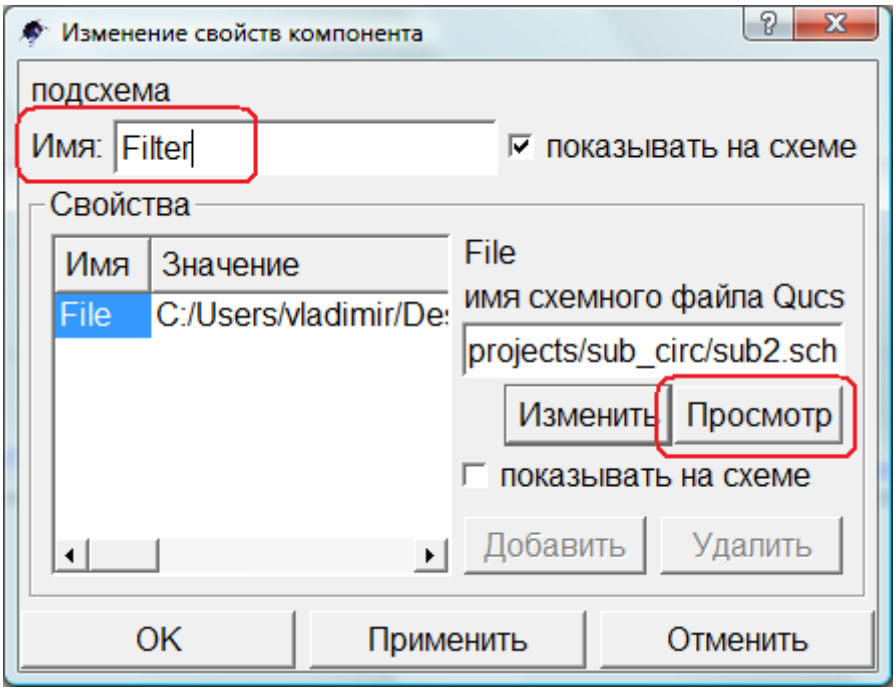


Рис. 10.16. Диалоговое окно компонента Подсхема

Кстати, в диалоговом окне можно изменить имя подсхемы. А для указания электрической цепи, используемой в качестве подсхемы, используется кнопка ***Просмотр***, которая вызывает проводник в Windows и файловый менеджер в Linux. Указав путь к нужному файлу, достаточно нажать кнопку ***OK***.

Эксперимент с операционным усилителем не был целью проекта. Чтобы осуществить последний эксперимент, хотелось бы изменить частоту фильтра. Для этого достаточно изменить значения конденсаторов в исходной схеме. А чтобы это сделать с наибольшим комфортом, можно использовать основное меню, выделив подсхему (щелчком левой клавиши по ней).

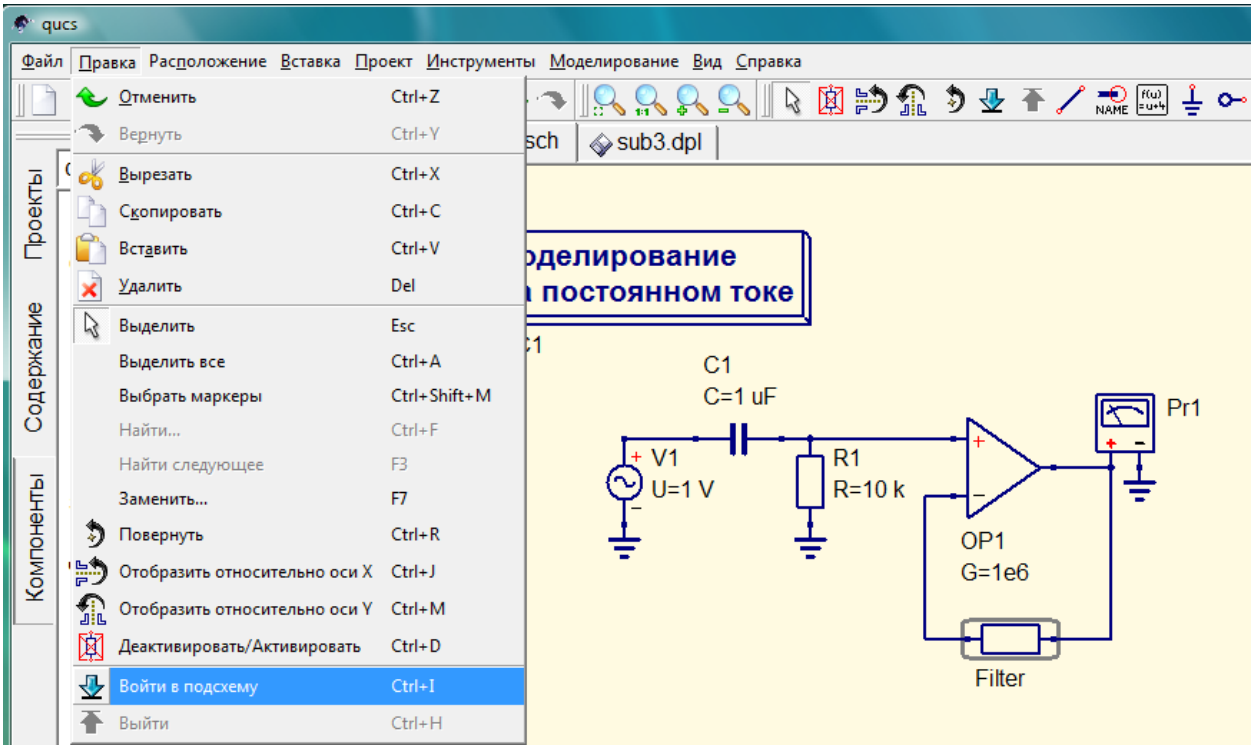


Рис. 10.17. Вход в подсхему для внесения изменений

После входа в подсхему открывается тот файл, где создавался фильтр. Заменив значения конденсаторов, чтобы частота стала 1 кГц, можно вернуться к исходной схеме, используя, скажем, инструментальную панель (хотя можно и следующий пункт раздела **Правка** основного меню).

Рис. 10.18. Кнопки входа в подсхему (левая) и выхода из подсхемы (правая)

Мы могли бы изменить описание подсхемы, выделив (на рисунке 10.13) этот атрибут и открыв его свойства.

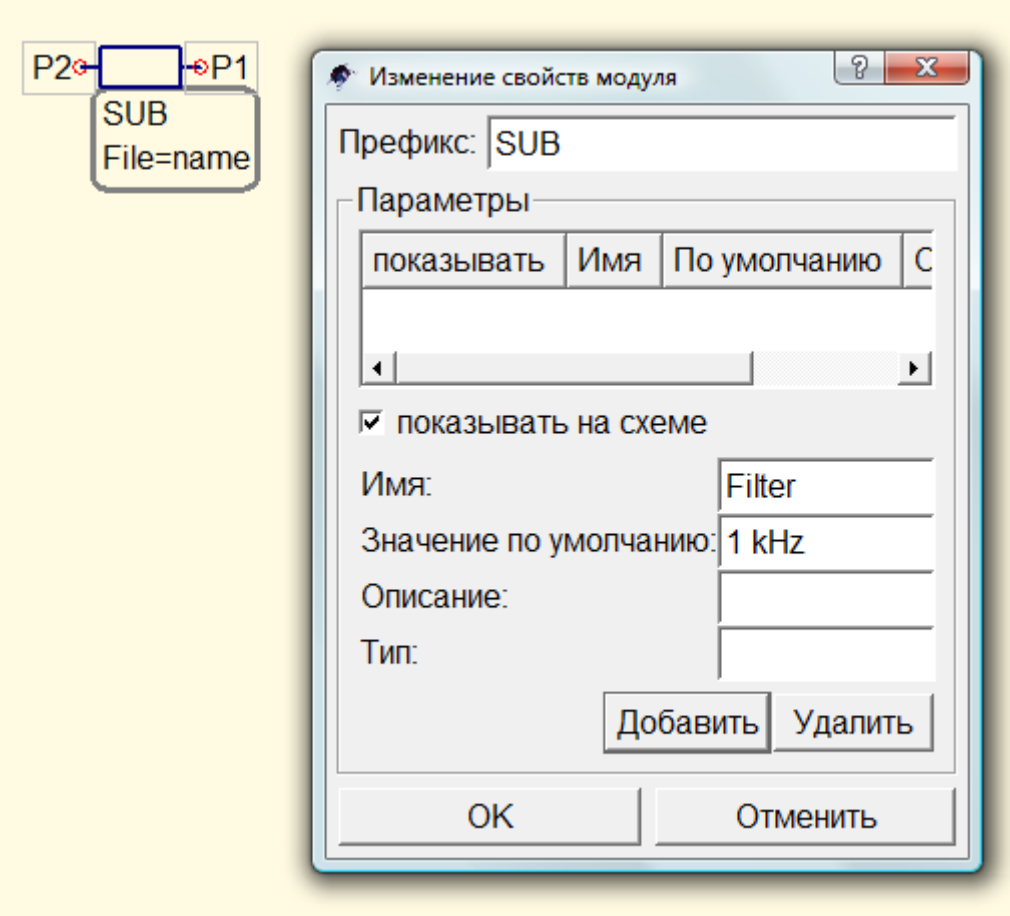


Рис. 10.19. Свойства тестового атрибута компонента **Подсхема**

В свойствах можно изменить имя, показать значение по умолчанию и дать описание. Нажав кнопку ***Добавить***, мы добавим это в текстовый атрибут компонента.

При добавлении подсхемы к основной схеме эти изменения будут видны.

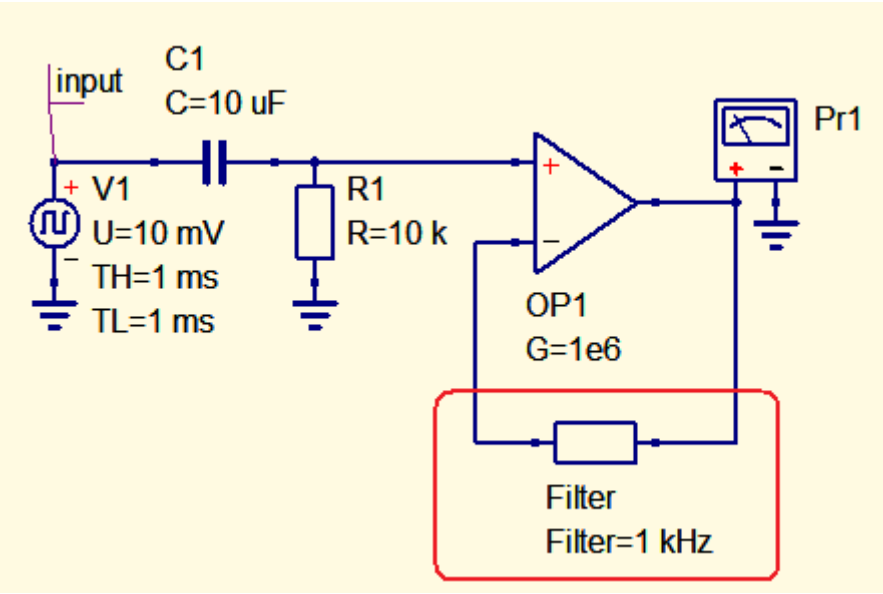


Рис. 10.20. Изменение описания компонента **Подсхема**

Внеся все правки в значения элементов фильтра, изменим схему для следующего эксперимента.

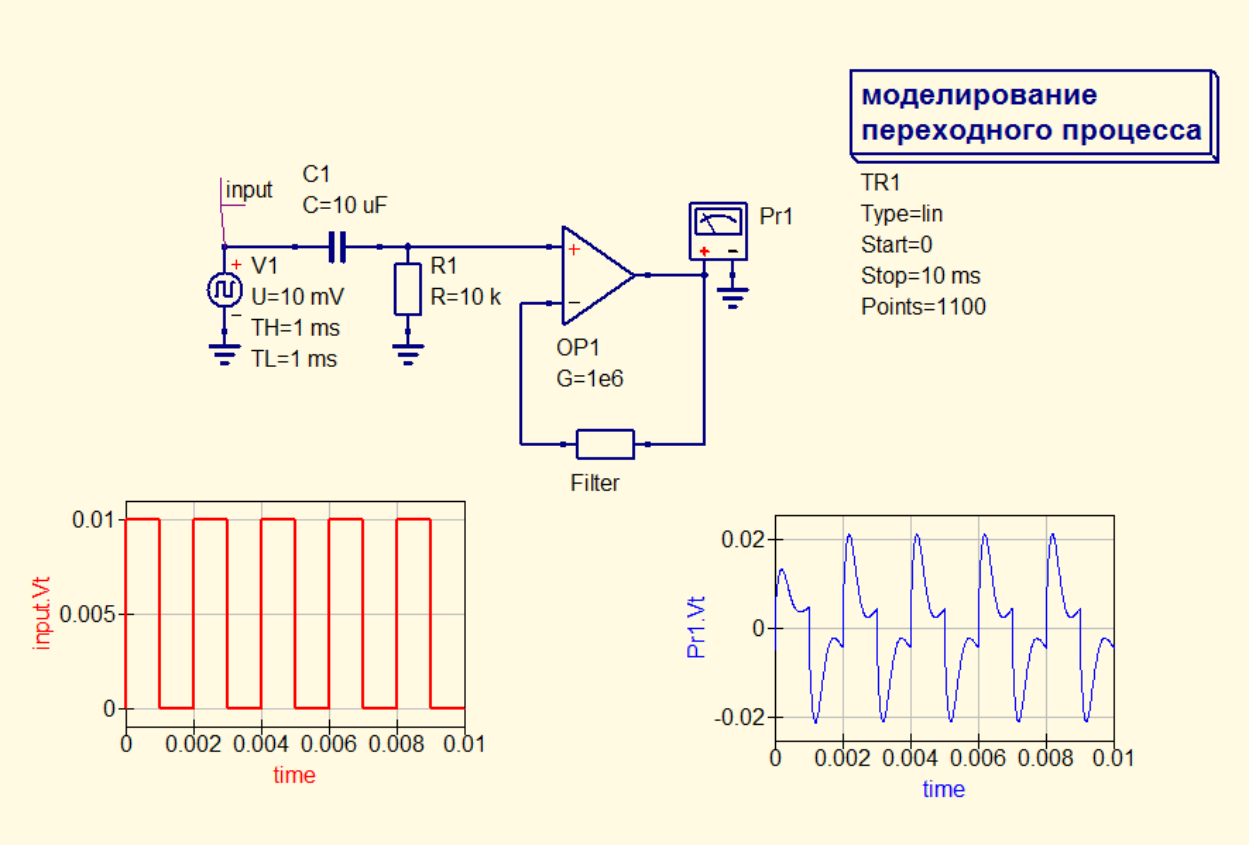


Рис. 10.21. Эксперимент с прямоугольными импульсами

Заменив источник переменного напряжения источником прямоугольных импульсов, мы на выходе получим сигнал, который достаточно близок к синусоиде. Возможно, если бы добротность фильтра была выше, сигнал в большей мере напоминал бы синусоидальный.

Смысл этого эксперимента напомнить, что прямоугольные периодические сигналы могу быть представлены совокупностью гармонических составляющих. Если выделить первую гармонику с помощью фильтра, то можно получить синусоидальный сигнал из прямоугольного.

На практике бывает проще построить генератор прямоугольных импульсов, чем синусоидальных. А испытание усилителя, где важно использовать генератор синусоидального сигнала, можно проводить на нескольких частотах, не обязательно во всем диапазоне частот. Используя хороший фильтр можно обойтись генератором прямоугольных импульсов.

Впрочем, речь шла о подсхемах. А из предыдущего рассказа достаточно ясно, если бы мы нарисовали фильтр на основной схеме, то читалась бы она труднее.

Файловые компоненты

Если для работы вам не хватает базовых компонентов, если в **Библиотеке компонентов** вы не находите нужного, то в группе **файловых компонентов** есть **схема SPICE**. Этот файловый компонент предназначен для добавления нового элемента при наличии SPICE-модели. Многие производители снабжают свои изделия подобными моделями. Вот как выглядит модель высокочастотного транзистора MBR951:

```
********************
.SUBCKT XMBR951 1 2 3
CBPAD 2 1 9E-14
CEPAD 1 3 9E-14
Q1 1 2 3 DMBR951
.ENDS
********************

.MODEL DMBR951 NPN
+ IS = 8.633E-16
+ BF = 105
+ NF = 0.9909
+ VAF = 26
+ IKF = 100
+ ISE = 1E-15
+ NE = 1.356
+ BR = 60
+ NR = 0.9983
+ VAR = 5
+ IKR = 0.022
+ ISC = 1.673E-16
+ NC = 1.076
+ RB = 5
+ IRB = 8E-05
+ RBM = 5
+ RE = 0.33
+ RC = 10
+ XTB = 0
+ EG = 1.11
+ XTI = 3
+ CJE = 2.9E-12
+ VJE = 1.061
+ MJE = 0.5089
+ CJC = 7.3E-13
+ VJC = 0.25
+ MJC = 0.27
+ XCJC = 0.5
+ TF = 9E-12
+ XTF = 500
+ VTF = 3
+ ITF = 1.1
+ PTF = 85
+ TR = 1E-09
+ FC = 0.5
*$
```

Это простой текстовый файл, который следует сохранить с расширением .cir.

Добавив в рабочее поле чертежа компонент **схема SPICE** из группы **файловые компоненты** закладки **Компоненты** панели навигации, можно двойным щелчком левой клавиши мышки открыть диалоговое окно свойств этого компонента.

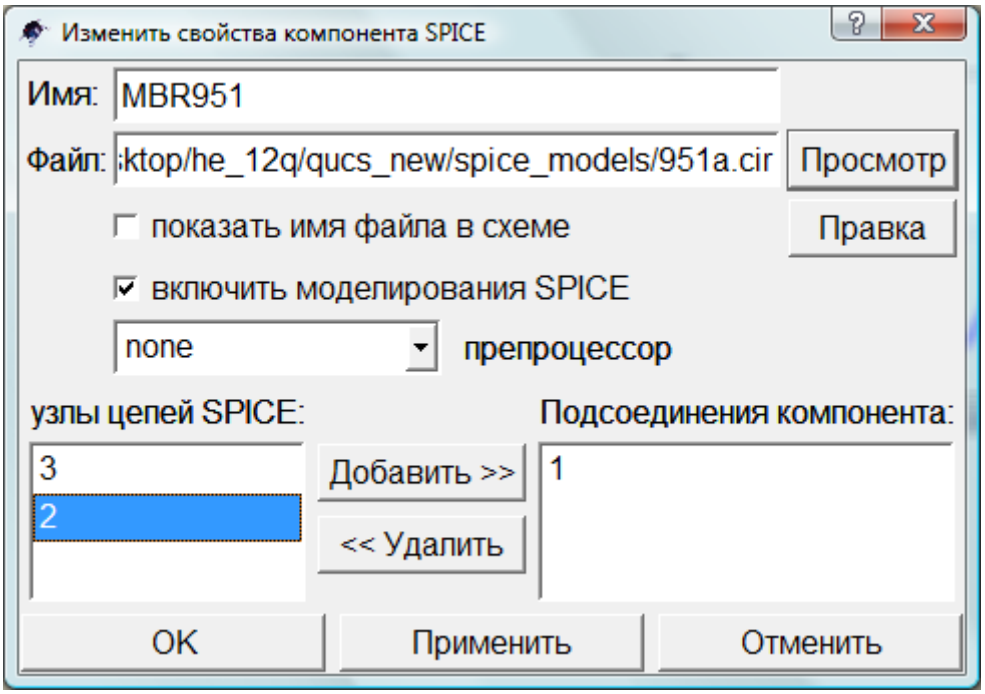


Рис. 10.22. Диалоговое окно свойств компонента **схема SPICE**

Используя клавишу ***Просмотр*** можно указать путь к нужному файлу, а в текстовом поле *Имя:* ввести нужное вам имя компонента.

При указании пути к файлу, после выбора, появляется сообщение:

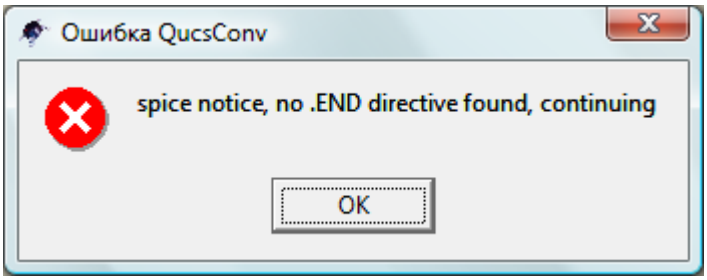


Рис. 10.23. Сообщение при открывании файла модели

Но ничего страшного не происходит, можно нажать кнопку ***OK***. В окне *узлы цепей SPICE:* будут отображены узлы, заданные строкой `.SUBCKT XMBR951 1 2 3` в файле модели. Выделяя узлы, с помощью кнопки Добавить>> их следует перенести в правое окно *Подсоединения компонента:*, где находятся узлы, к которым будут присоединяться к компоненту остальные элементы схемы. Можно снять флажок ***показать имя файла в схеме***, который будет отображать путь к файлу модели. Хотя вид транзистора будет несколько непривычен, с ним можно работать, как с другими транзисторами. На рисунке ниже вывод 1 обозначен дополнительно буквой «К» (коллектор), буквами «Б» и «Э» обозначены база и эмиттер транзистора. Но это не обязательно. В остальном с компонентом можно поступать, как с любым другим транзистором.

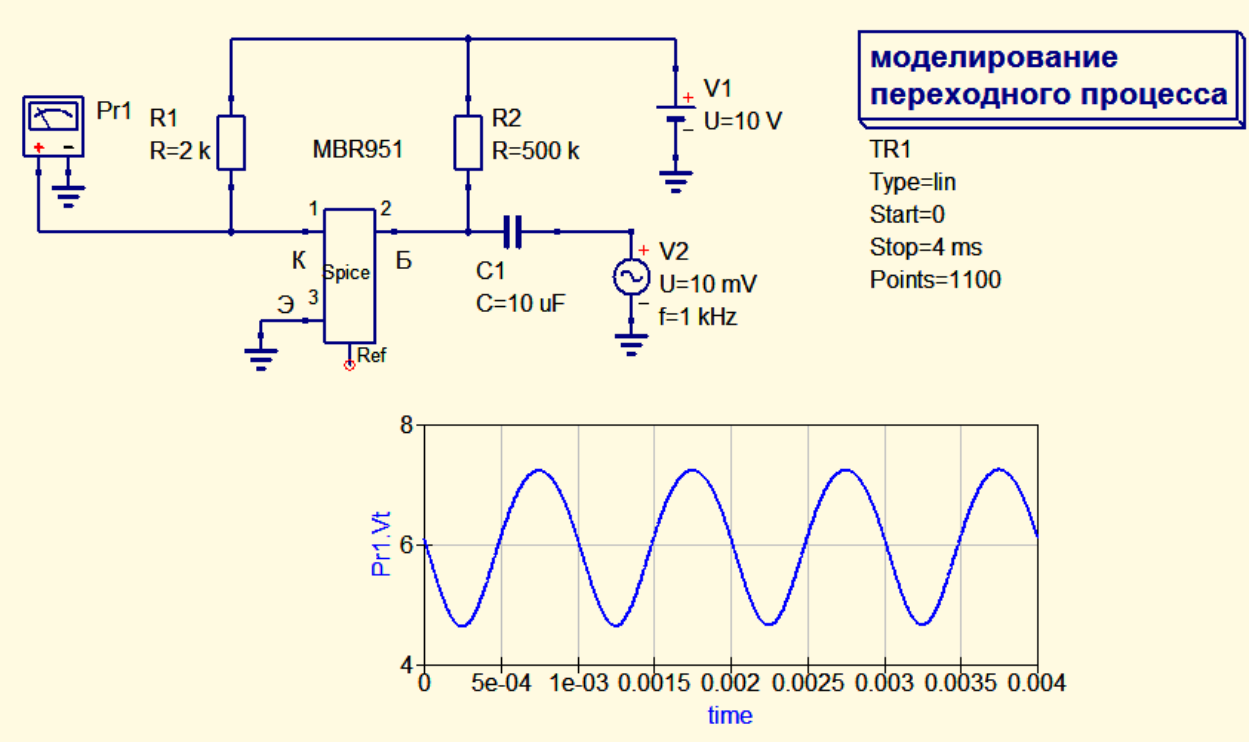


Рис. 10.24. Работа с транзистором, заданным моделью SPICE

Используемая модель может быть и макромоделью.

Для операционных усилителей, когда в корпусе микросхемы может быть четыре усилителя, модель представлена одним элементом, как на рисунке ниже.

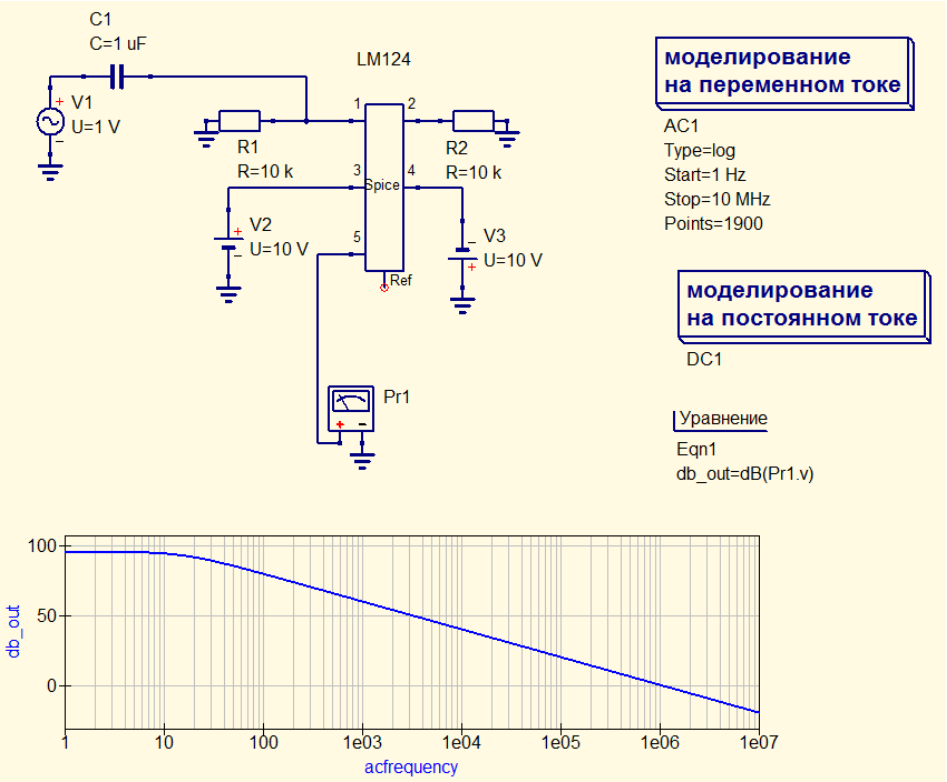


Рис. 10.25. Использование модели ОУ LM124

Процесс симуляции при использовании внешних моделей может идти дольше, чем при использовании компонентов из Библиотеки компонентов. Следует дождаться его завершения.

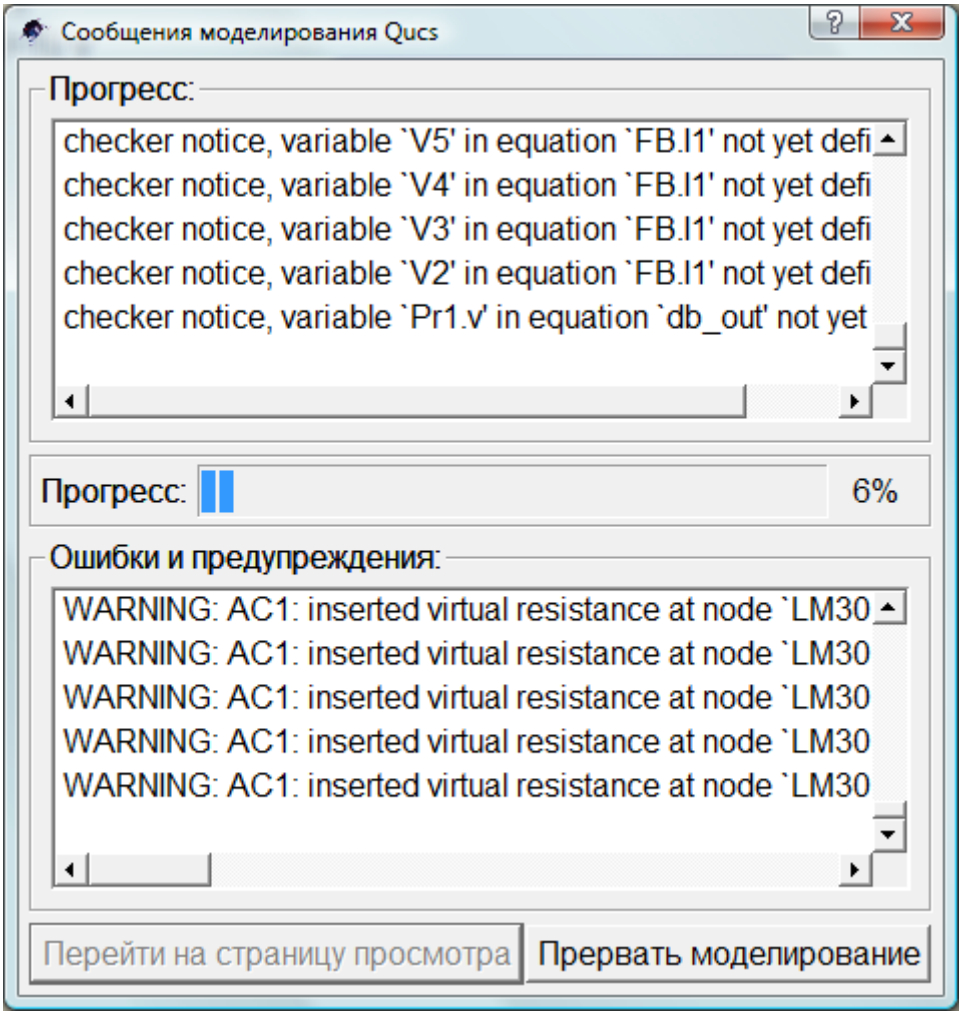


Рис. 10.26. Процесс симуляции при использовании внешней модели

Файловые компоненты есть и в группе цифровых компонентов: это Файл VHDL и Файл Verilog. Их использование подразумевает, что эти файлы уже написаны, первый на языке VHDL, второй на Verilog. Вот пример простого файла на языке Verilog:

```
// Verilog code for AND3 gate
module and3gate (a, b, c, d);
input a, b, c;
output d;

assign d = a & b & c;
endmodule
```

Этот файл можно написать в блокноте, он, как и файл модели SPICE текстовый. Сохранив его с расширением .v, его можно указать в качестве целевого для компонента **Файл Verilog**. Добавим этот компонент в новом проекте, откроем диалоговое окно двойным щелчком левой клавиши мышки по нему, где укажем путь к файлу с текстом.

Для проверки работы нам потребуются три цифровых источника S1-S3 и цифровое моделирование, где выберем *TruthTable* в качестве типа моделирования. И не забывайте в диалоговом окне **цифрового моделирования** указать, что используется формат verilog.

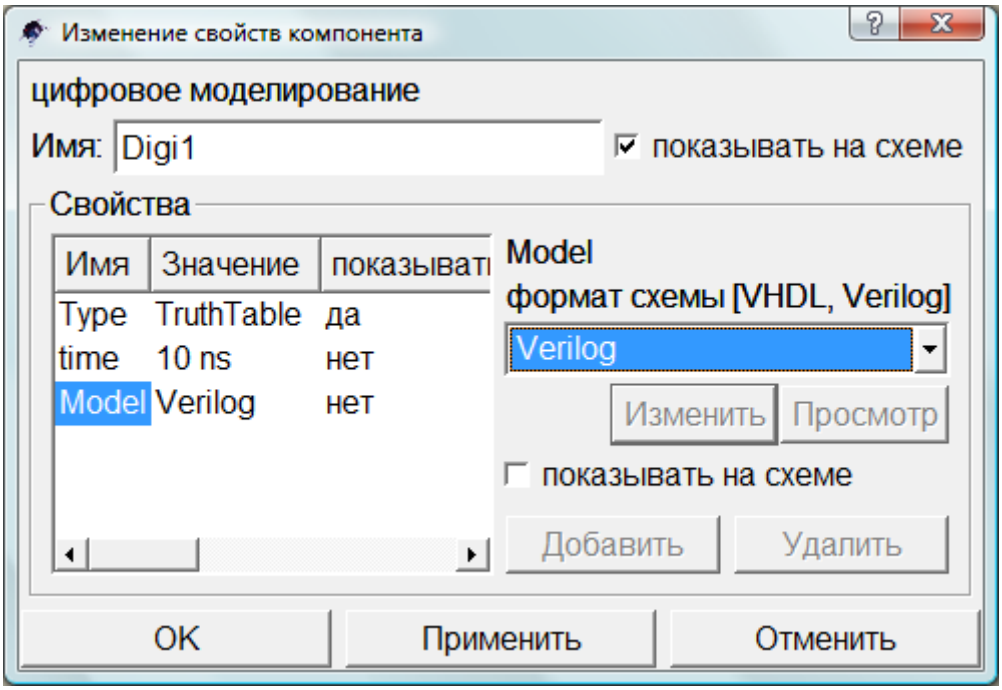


Рис. 10.27. Настройка цифрового моделирования

Иначе все выходные значения оказываются неопределенными.

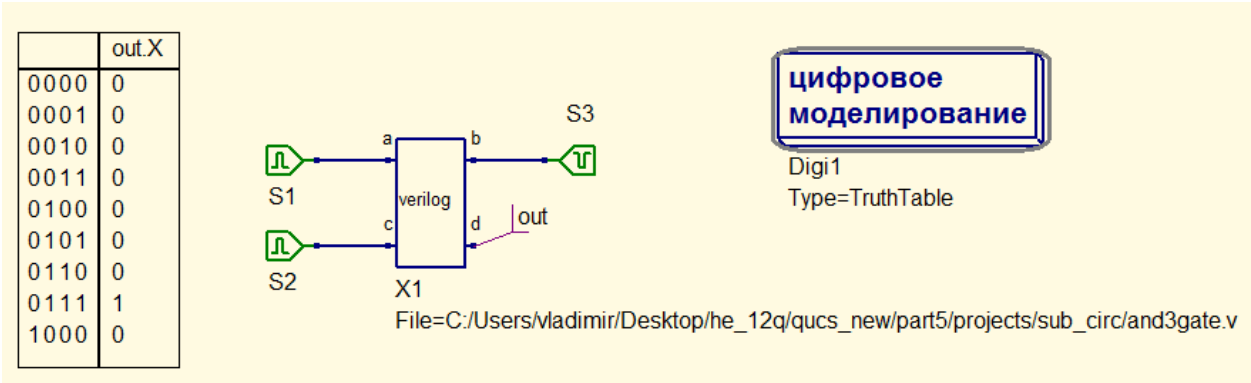


Рис. 10.28. Испытания файлового компонента Verilog

Несколько слов о Verilog.

В полупроводниковой и электронной индустрии Verilog — это язык описания оборудования (HDL), используемый для моделирования электронных систем.

Verilog HDL, не следует путать с VHDL, наиболее часто используется при разработке, проверке и реализации цифровых логических микросхем на уровне регистровых передач (RTL) абстракции. Он также используется при проверке аналоговых и смешанных цепей.

Verilog-AMS (Verilog Analog Mixed-Signal Simulation) помогает пользователям, позволяя описать и симулировать аналоговые и со смешанными сигналами разработки, с помощью методологии разработки верхнего уровня столь же успешно, что и традиционные нижнего

уровня подходы. Verilog-AMS расширяет возможности языка цифрового моделирования (IEEE 1364, Verilog-D) и Verilog-A, предлагая единый унифицированный язык с обеими, аналоговой и цифровой, семантиками и обратной совместимостью. Стандарт Verilog-AMS поддерживает разработку аналоговых и смешанных цепей на трех уровнях: транзистор/вентиль, транзистор/вентиль-rtl/поведение и смешанном транзистор/вентиль-rtl/поведение цепи. Более того, Verilog-AMS предоставляет мощные структурные и поведенческие модельные возможности для систем, в которых есть внешние эффекты и внутренние взаимоотношения, где различные проявления, электрические, механические и температурные, очень важны.

Verilog-AMS — это язык описания оборудования. Он может быть полезен для спецификации поведения аналоговых моделей компактных устройств. Обычно это C или C++ реализации в аналоговых симуляторах. Попытки реализовать современные компактные модели на C/C++ достаточно хорошо сравнимы с описанием на Verilog-AMS.

Программа ADMS (см. http://mot-adms.sourceforge.net) позволяет Verilog-AMS описания транслировать на любые другие языки программирования. Она генерирует структурированное XML дерево, представляющее описание модели компактного устройства.

Внутреннее XML дерево используется для генерации готового к компиляции C/C++ кода, который специфицирован для симуляторов API. Генератор кода способен производить:

- *вычисления уравнений устройства (ток и заряд), включая их производные;*
- *связующий код для симуляторов API;*
- *документацию и*
- *любые другие данные, описанные в оригинальном Verilog-AMS входном файле.*

Языковое преобразование использует язык, названный admst. Это, собственно, XML описание.

В плане создания admst скриптов для симулятора необходимо понимать и специфику симулятора API, и то, как разделы дерева данных ADMS — которые базируются на исходном файле Verilog-AMS при описании модели — соотносятся с API.

Язык admst используется для прохождения по внутреннему дереву. Корень дерева определен в определении модуля Verilog-AMS.

```
module d e v i c e ( node1 , node2 , . . . )
// module d e f i n i t i o n s and code
endmodule
```

Тот факт, что компонент **Файл Verilog** находится в группе цифровых компонентов, подразумевает, что используется только цифровое представление модели.

Описание для файловых компонентов цифровой группы в Qucs несколько отличается от оригинального языка. Повторим моделирование трехходового вентиля И, используя штатный компонент программы. После моделирования в разделе основного меню **Моделирование** откроем пункт **Показать последнюю схему**. Текстовый файл будет файлом на языке VHDL.

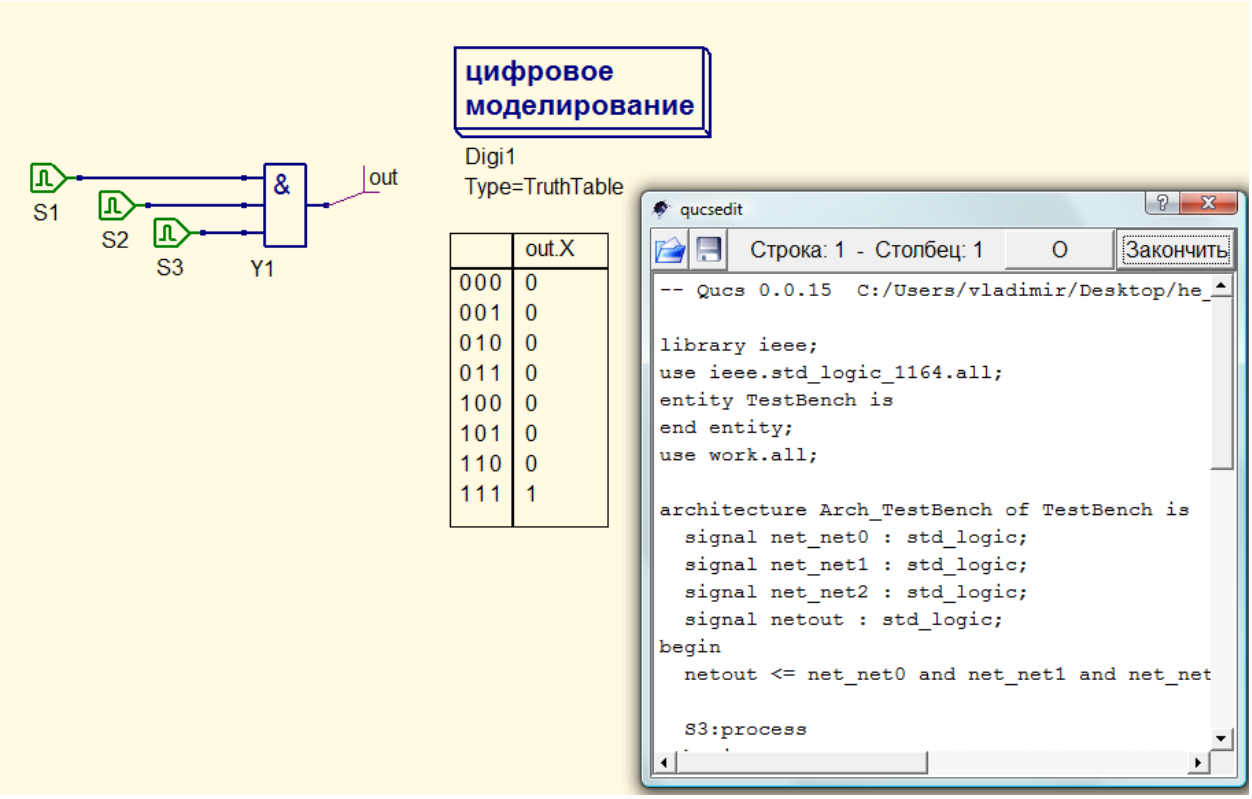


Рис. 10.29. Моделирование вентиля И

Текстовый файл с расширением .vhdl для этого эксперимента выглядел бы следующим образом:

```
-- VHDL code for AND3 gate

library ieee;
use ieee.std_logic_1164.all;
entity and3gate is port (a, b, c : in std_logic; d : out std_logic);
end and3gate;
architecture
and3gate_arch of and3gate is
begin
d <= a and b and c;
end and3gate_arch;
```

Сохранив этот файл, можно провести эксперимент с файловым компонентом **Файл VHDL**.

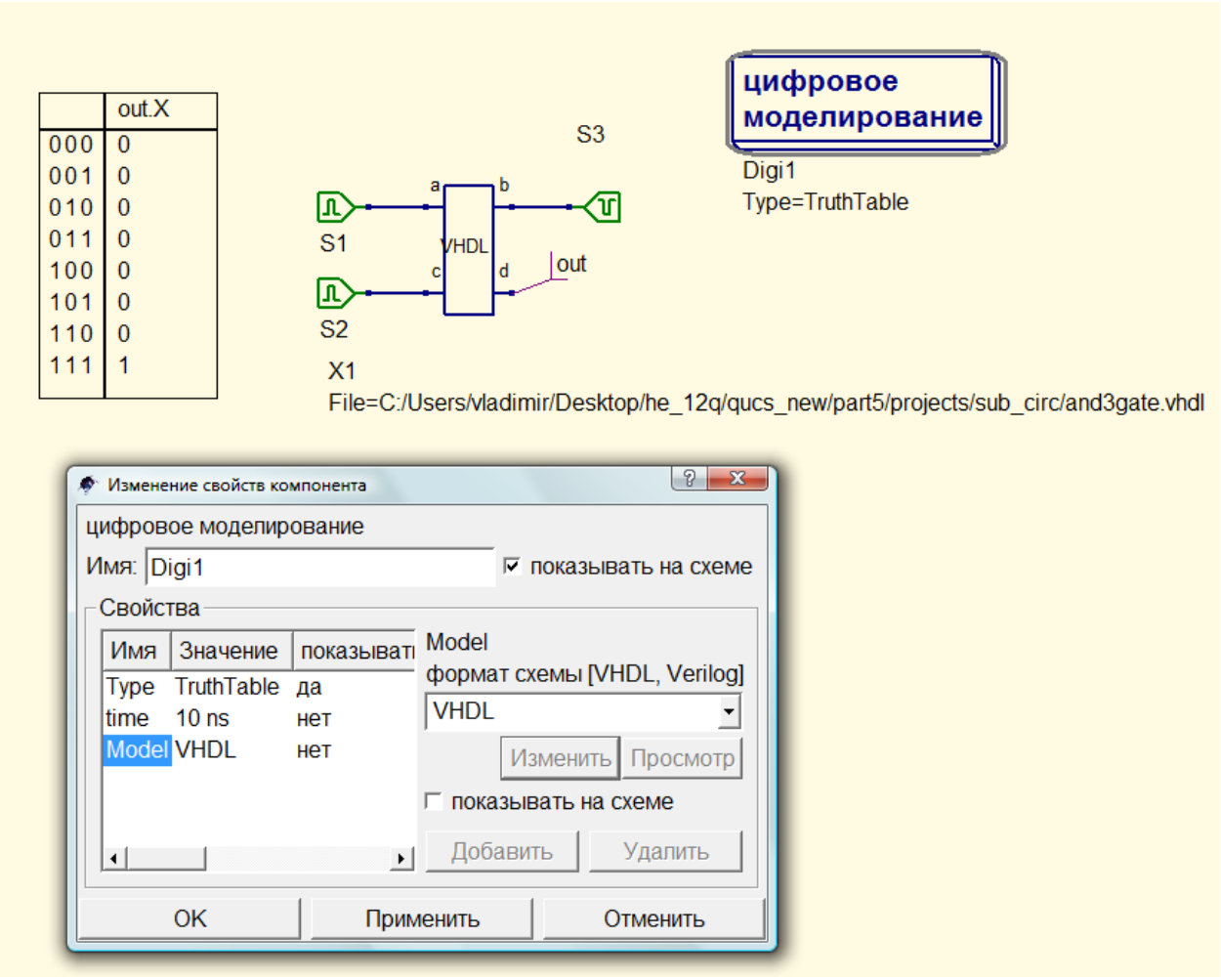


Рис. 10.30. Симуляция предыдущего файла в Qucs

Сравнивая таблицы истинности проверки всех трех моделей вентиля «И» можно со всей определенностью отметить, что все три модели идентичны. Использование файловых компонентов дает возможность расширить модельный ряд электронных компонентов.

Виды моделирования

Кроме тех видов моделирования, с которыми мы познакомились, в группе **виды моделирования** есть еще два вида: **гармонический баланс** и **оптимизация**.

Что касается первого вида моделирования, то он применим к анализу установившегося режима в нелинейных электрических цепях, рассматриваемого в некоторой области частот. Рассмотрим простейшую нелинейную цепь, состоящую из двух встречно включенных диодов.

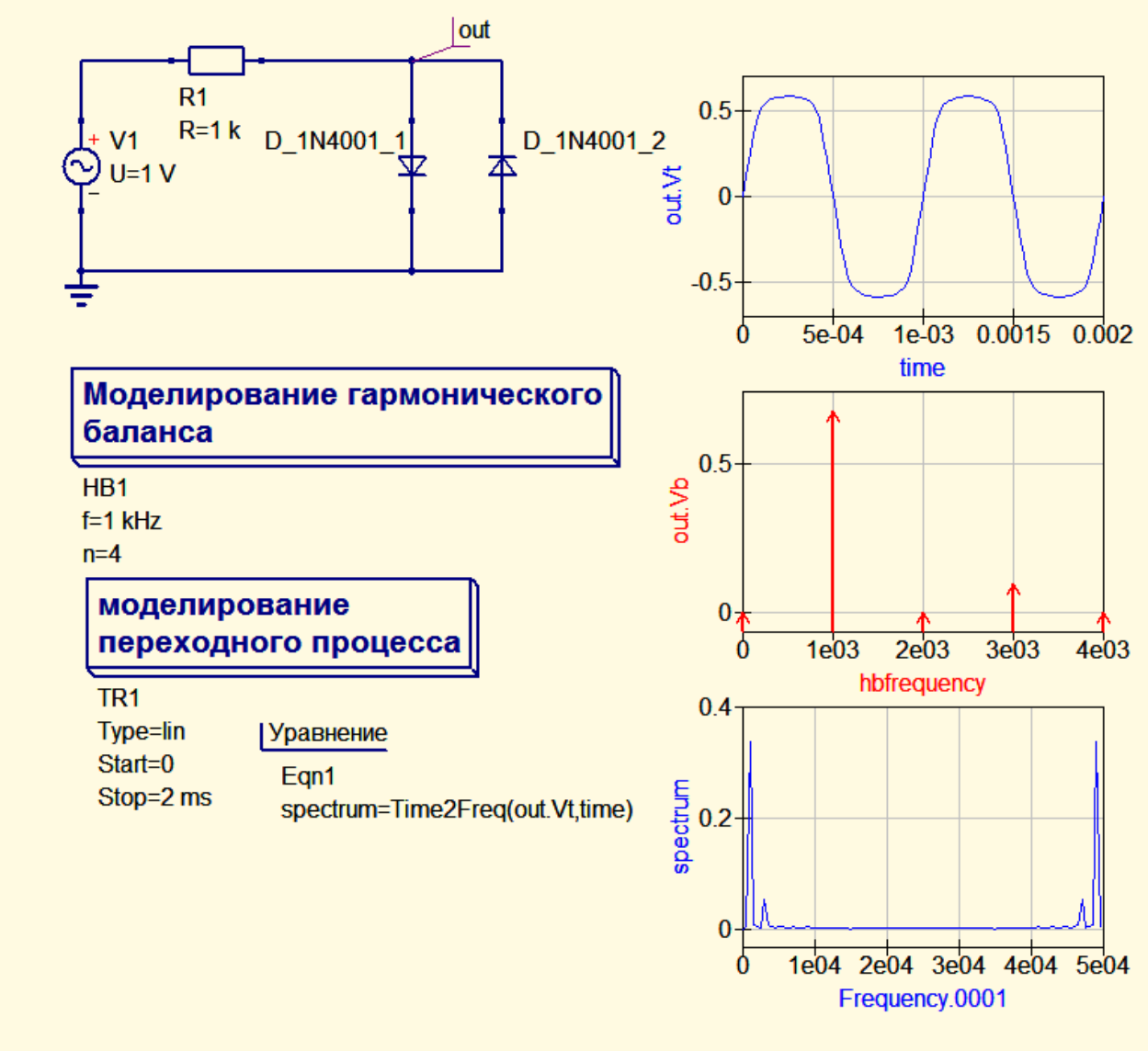
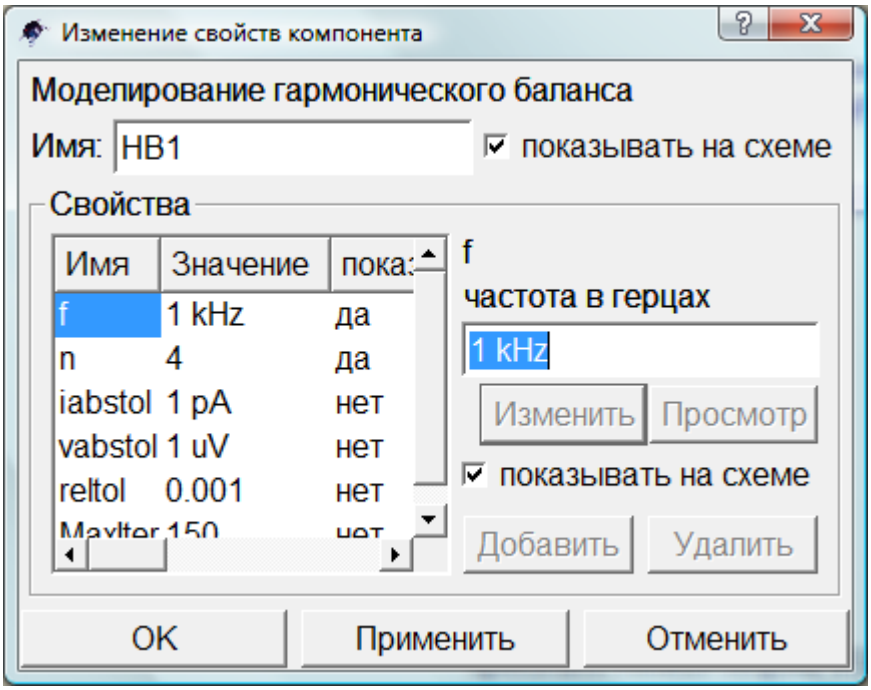


Рис. 10.31. Гармонический баланс

Диалоговое окно гармонического баланса позволяет задать частоту и количество наблюдаемых гармоник.



Уравнение: spectrum = Time2Freq(out.Vt,time), на рисунке выше означает дискретное преобразование Фурье функции out(t), и интерпретирует ее физически.

Рис. 10.32. Диалоговое окно гармонического баланса

Вид спектральной диаграммы (spectrum на рисунке выше) можно изменить, используя свойства диаграммы.

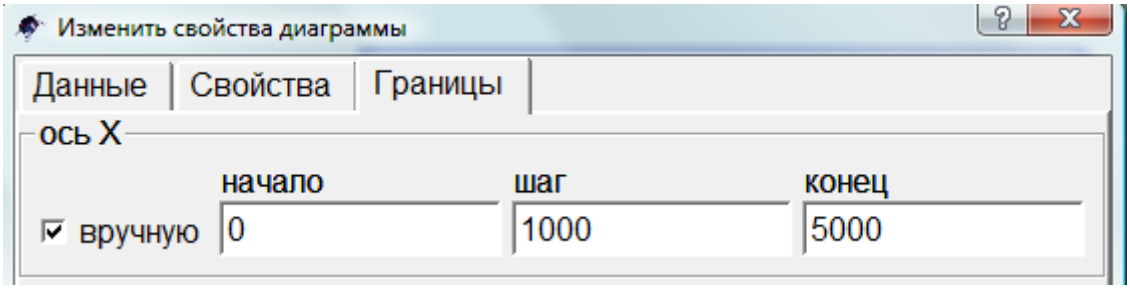


Рис. 10.33. Изменение свойств диаграммы спектра

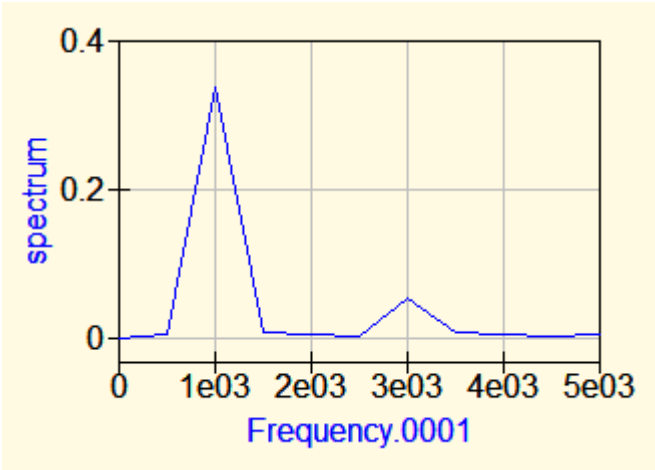


Рис. 10.34. Измененный вид диаграммы

А как вам такой эксперимент?

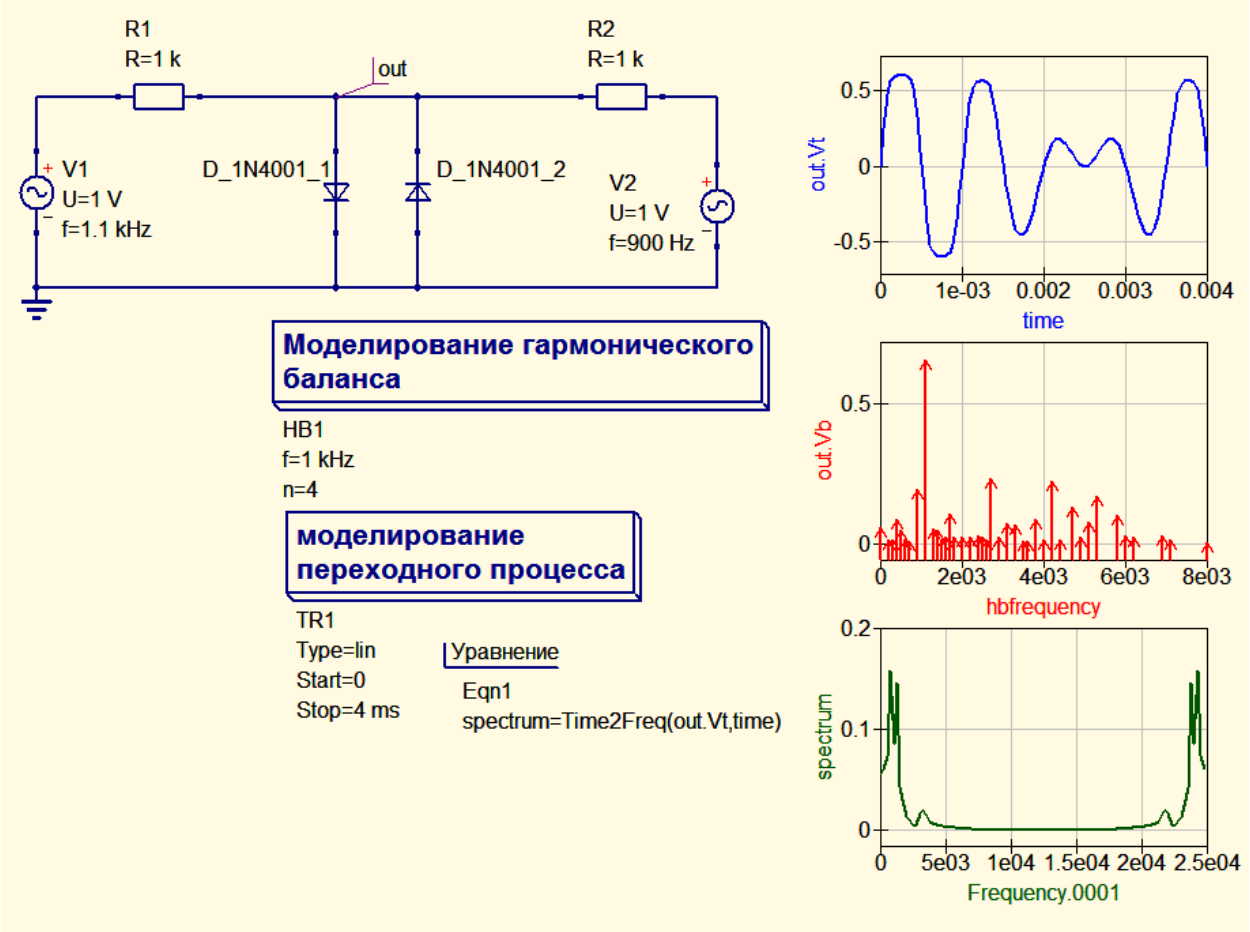


Рис. 10.35. Модификация предыдущего анализа

Что же до **оптимизации**, это не что иное, как минимизация функции стоимости. Это могут быть либо время задержки или фронта для цифровой схемы, мощность или усиление для аналоговой. Другая возможность — определение проблем оптимизации, как композиции функций, главное в этом случае определение добротности.

Для оптимизации цепей Qucs использует программное средство ASCO (http://asco.sourceforge.net/). Краткое описание того, как подготовить схему, выполнить и интерпретировать результаты, дано ниже. Перед выполнением этой функции следует установить на компьютер ASCO.

В Linux все необходимое в виде исходных кодов устанавливается, об этом можно прочитать в файле Readme, с помощью типовых команд компиляции программы из исходных файлов. Подобную же компиляцию можно сделать в Windows. Но можно использовать и готовую версию программы ASCO, которую сегодня можно найти на сайте разработки:

http://asco.sourceforge.net/downloads_win32.html

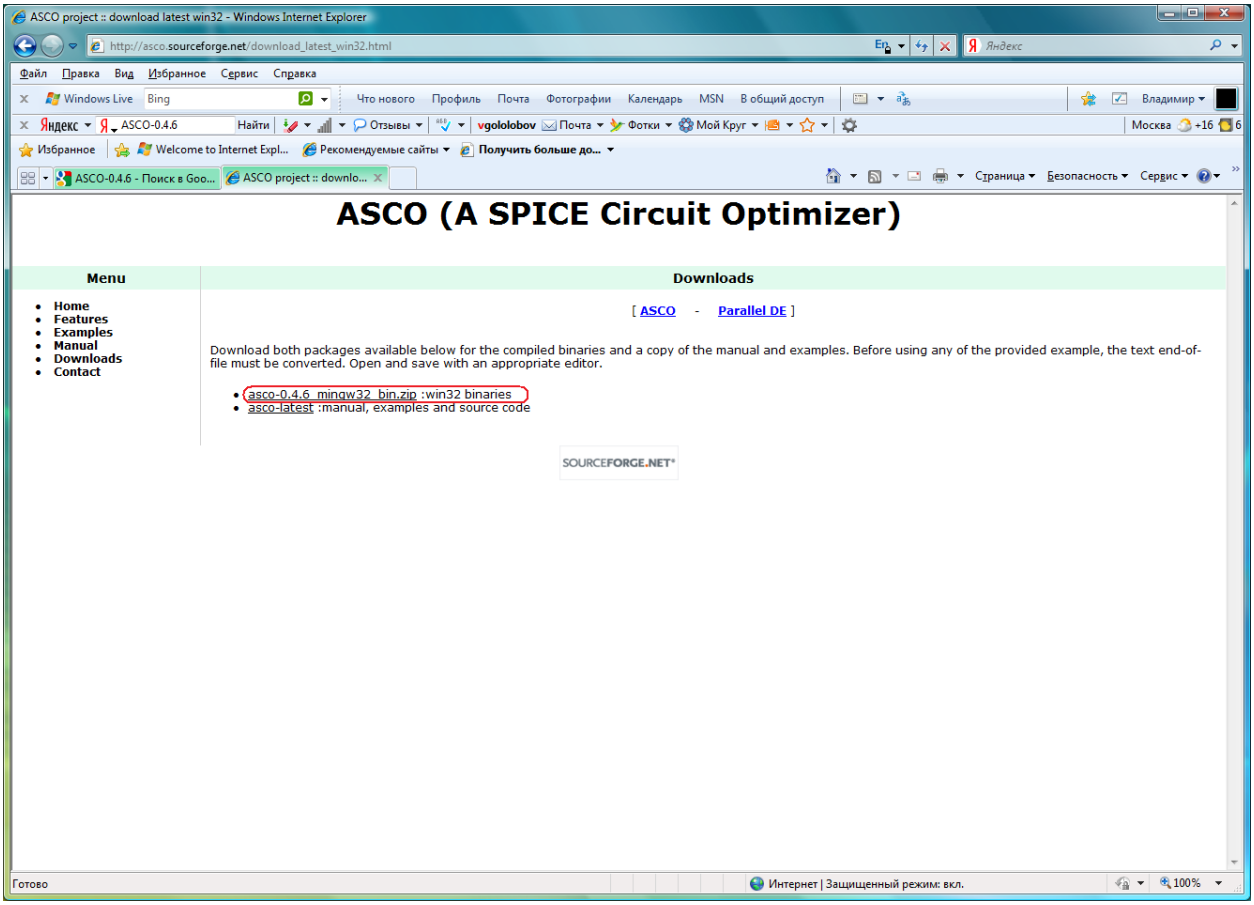


Рис. 10.36. Сайт разработки программного средства ASCO

Вот, что написано об установке пакета после его скачивания и распаковке в файле Readme для пользователей Windows:

Скопируйте оба исполняемых файла, «asco» и «asco-test» в директорию одного из предложенных примеров для вашего симулятора. А о поддерживаемых симуляторах ниже сказано: под win32 поддерживаются симуляторы LTspice, HSPICE и Qucs. Каждый из них должен быть в искомом пути к ltspice, hspice и qucsator.

Последуем этим советам, перенеся файлы в директорию программы Qucs.

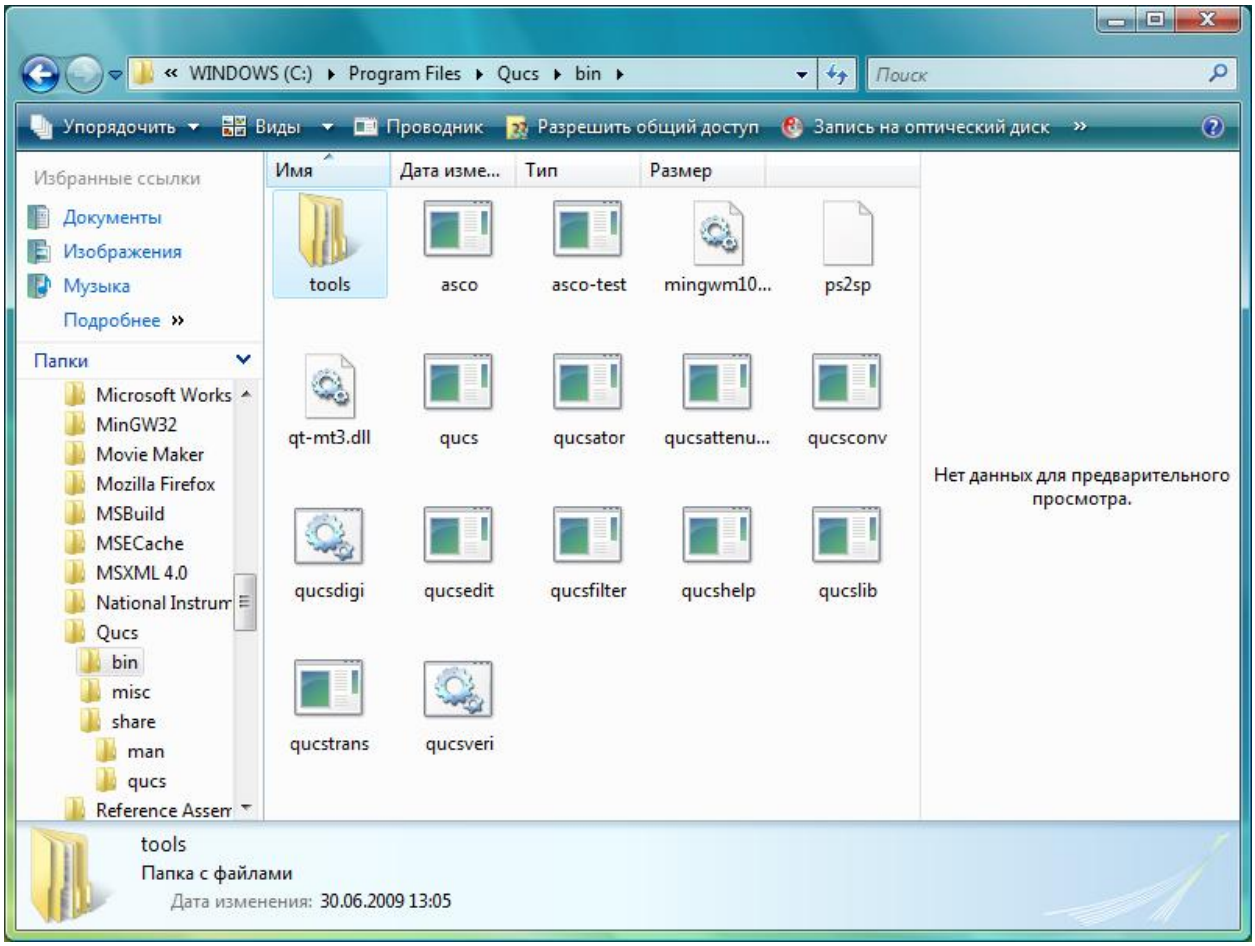


Рис. 10.37. Путь установки файлов ASCO

Для установок оптимизации должны быть добавлены две вещи к уже существующему файлу netlist: вставлено уравнение(ия) и компонент — блок оптимизации.

Конечно, чтобы файл netlist существовал, после добавления уравнений и вида моделирования **оптимизация** следует выполнить моделирование.

Кстати, прежде чем устанавливать ASCO, проверьте, ваша версия Qucs уже может содержать необходимые для работы файлы.

Возьмем схему на картинке ниже (вы может найти ее в папке «Examples» пакета ASCO).

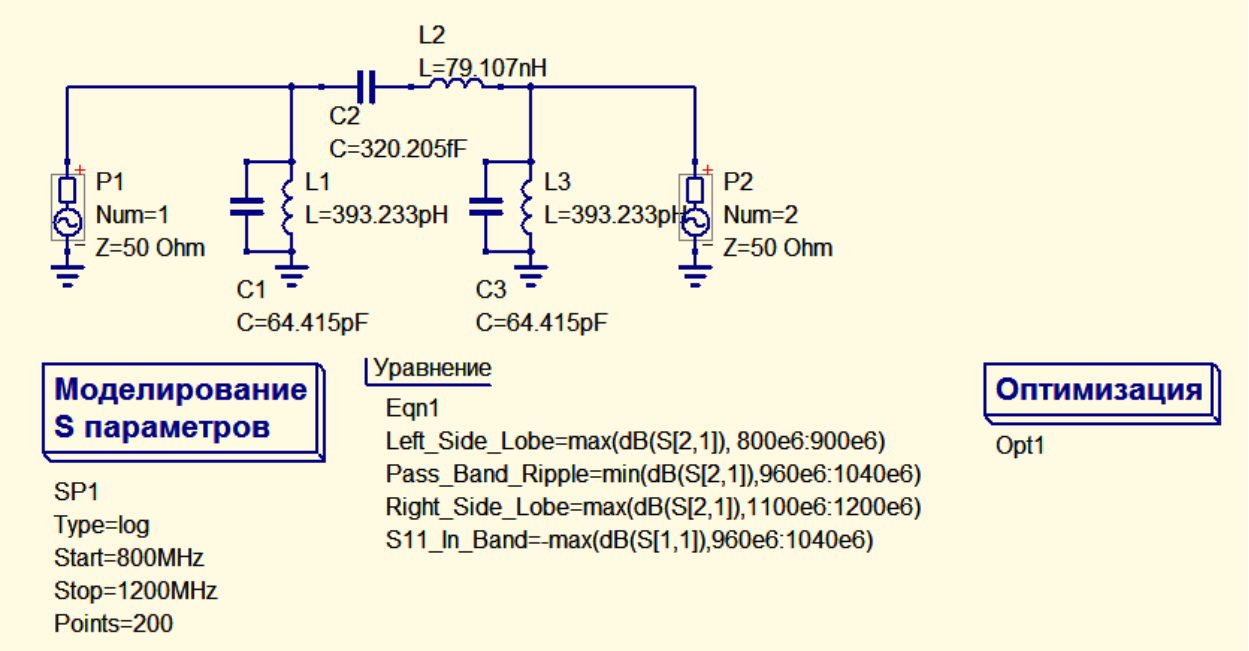
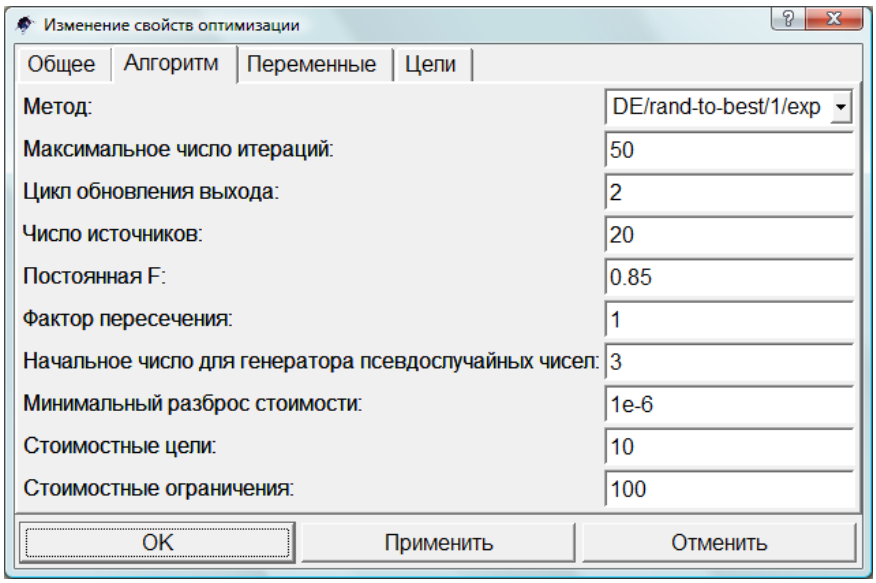


Рис. 10.38. Выполнение оптимизации схемы bandpass.sch из набора примеров ASCO

Теперь откроем компонент **Оптимизация** и выберем закладку ***Алгоритм***. Из существующих параметров особое внимание следует уделить «*Максимальному числу итераций*», «*Постоянной F*» и «*Фактору пересечения*». Чрезмерные или недостаточные значения могут привести к преждевременной сходимости оптимизации на локальном минимуме или к очень большому времени процесса оптимизации.



Закладка (ниже) ***Переменные*** определяет, какие элементы цепи будут выбраны из допустимого диапазона. Имена переменных относятся к идентификаторам, размещенным в свойствах компонентов, а не к именам компонентов.

Рис. 10.39. Диалоговое окно вида моделирования **Оптимизация**

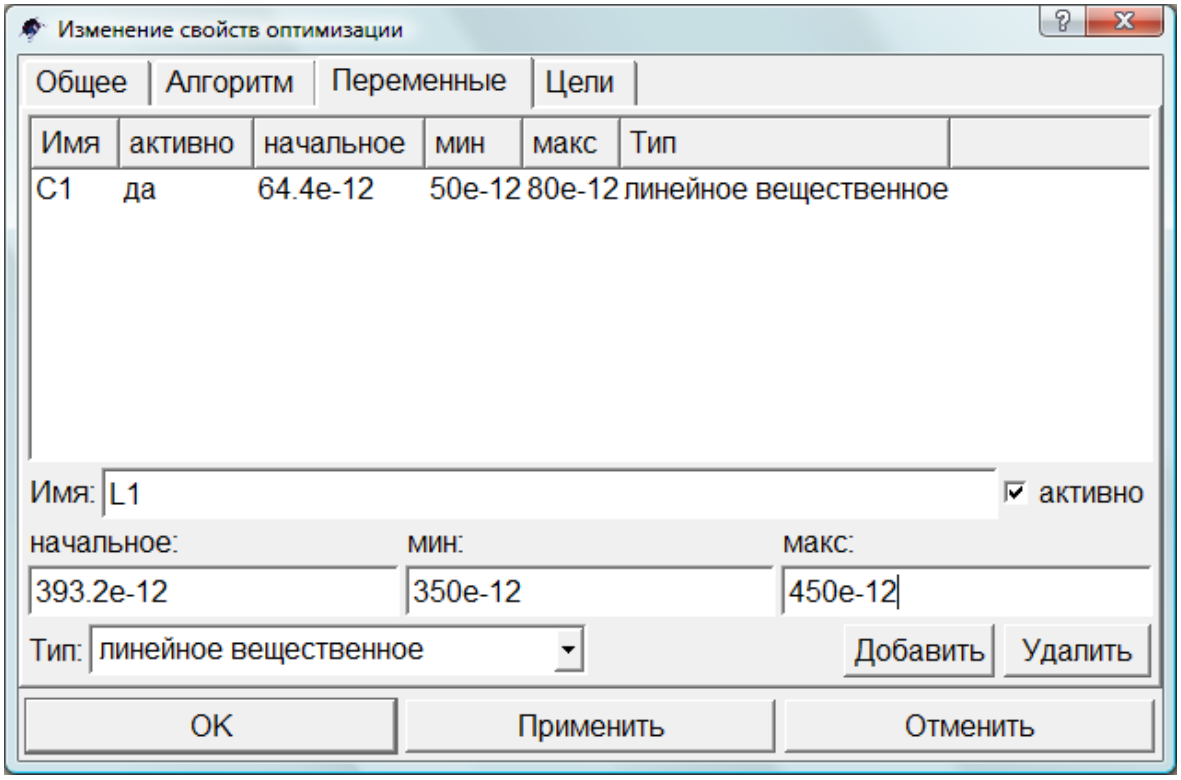


Рис. 10.40. Диалоговое окно **Оптимизация** на закладке ***Переменные***

Порядок работы с этой закладкой простой – ввести в текстовые окна нужные значения, выбрав нужный тип (в данном случае *линейное вещественное*) и нажать клавишу ***Добавить***. Все значения добавляются для каждой из переменных. В результате верхнее окно должно быть заполнено следующим образом:

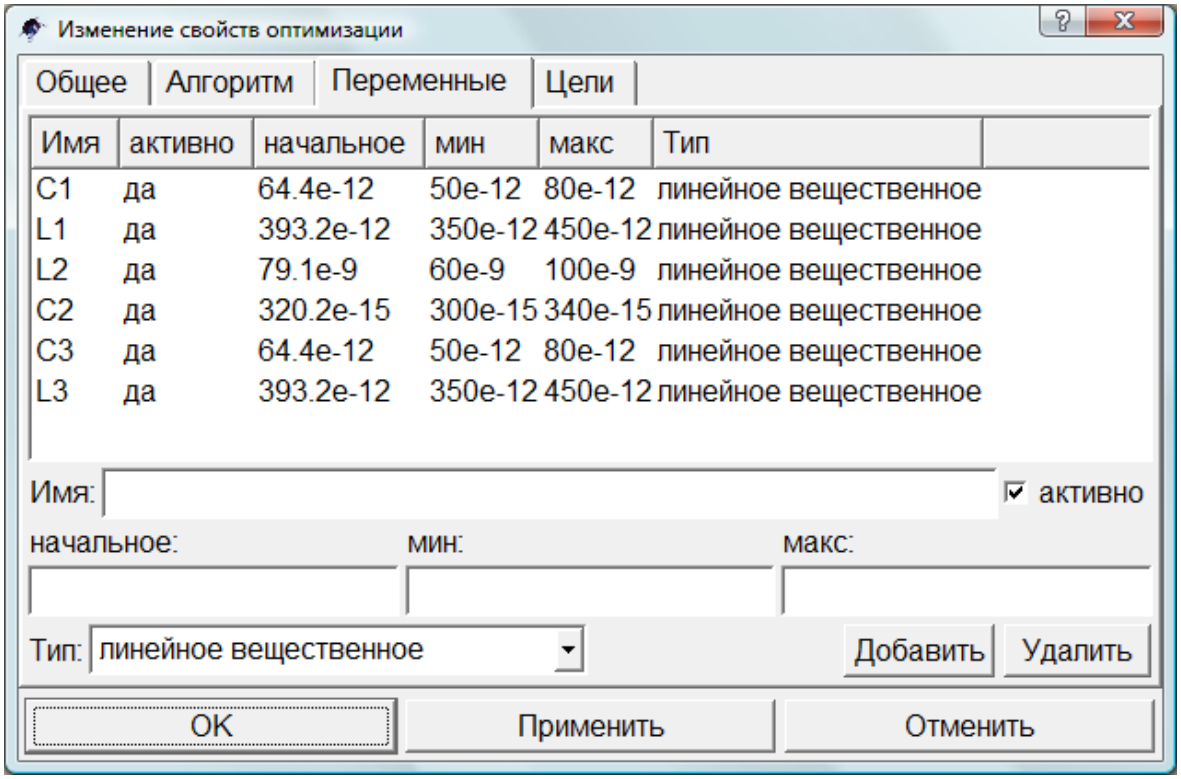


Рис. 10.41. Заполненное окно значений переменных для оптимизации рис. 10.38

Заполнив значения переменных, можно нажать кнопки ***Применить*** и ***ОК***., и перейти к закладке ***Цели***, где определяются цели (максимизировать, минимизировать) и ограничения (меньше, больше, равно).

Затем ASCO автоматически скомбинирует их в единую функцию стоимости, которая и будет минимизирована.

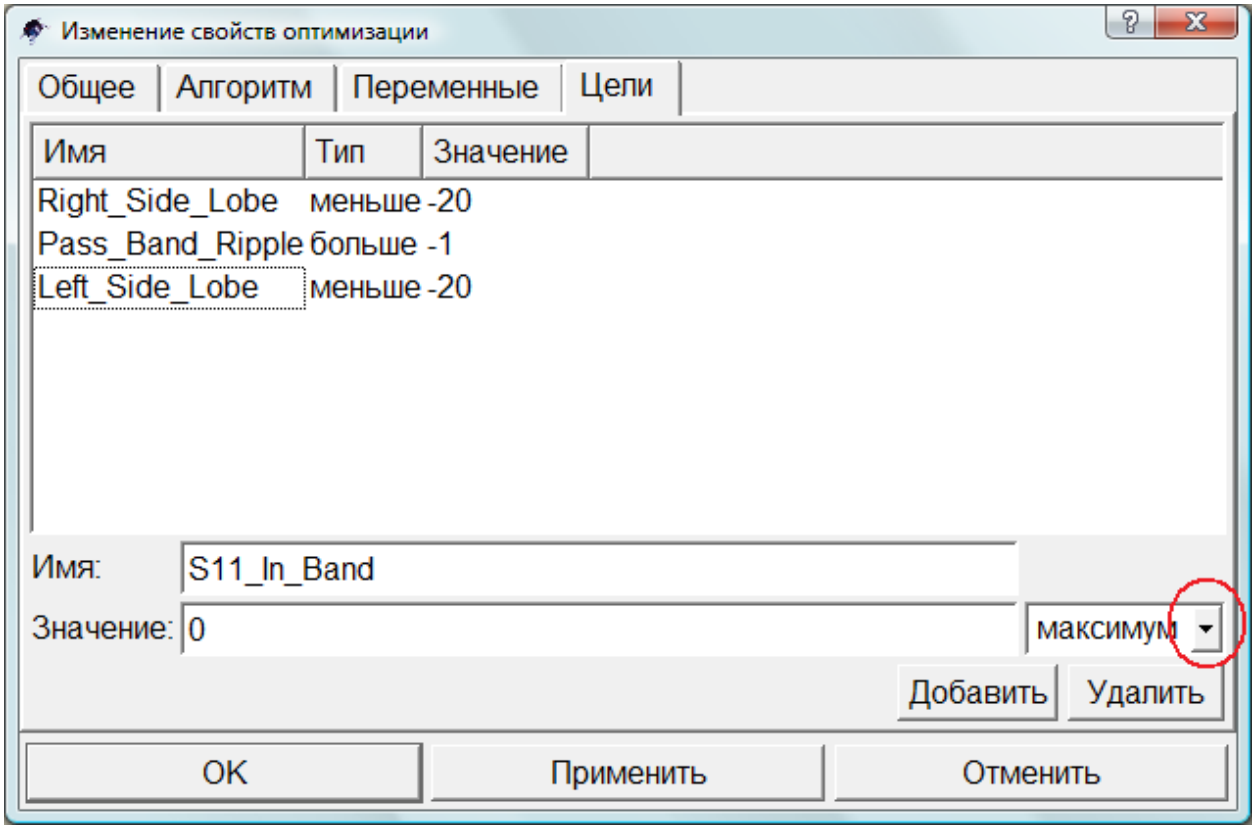


Рис. 10.42. Заполнение закладки ***Цели***

Здесь, как и в предыдущем случае заполняются поля, условие выбирается из выпадающего списка, даваемого отмеченной на рисунке кнопкой, и с помощью кнопки ***Добавить*** заполненные значения переносятся в верхнее окно. Если при заполнении произошла ошибка, можно выделить строку с ошибкой и использовать кнопку ***Удалить*** для удаления неверного значения.

Далее необходимо изменить схему (рисунок 10.38), как показано ниже. Подготовка, собственно, сводится к замене всех значений компонент, которые мы уже задали в диалоговом окне **Оптимизация**, именами переменных.

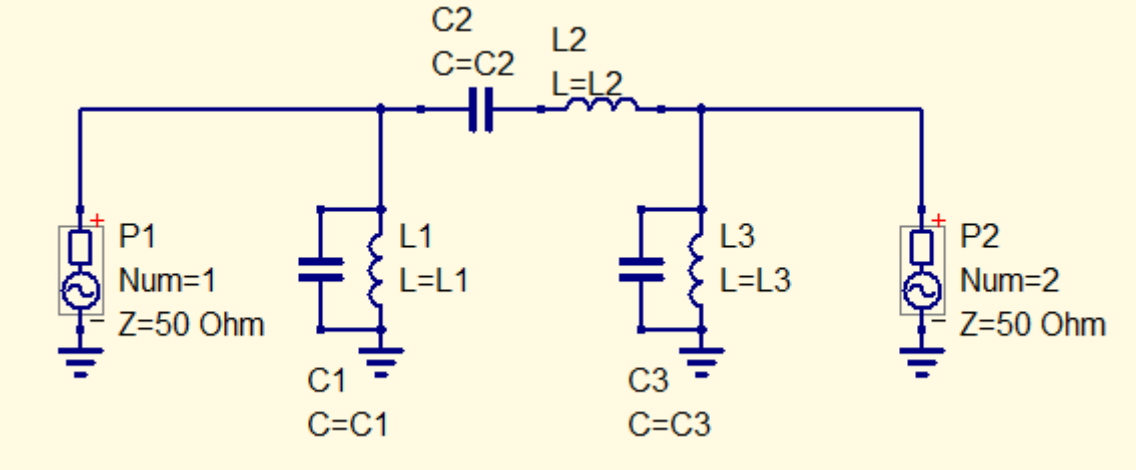
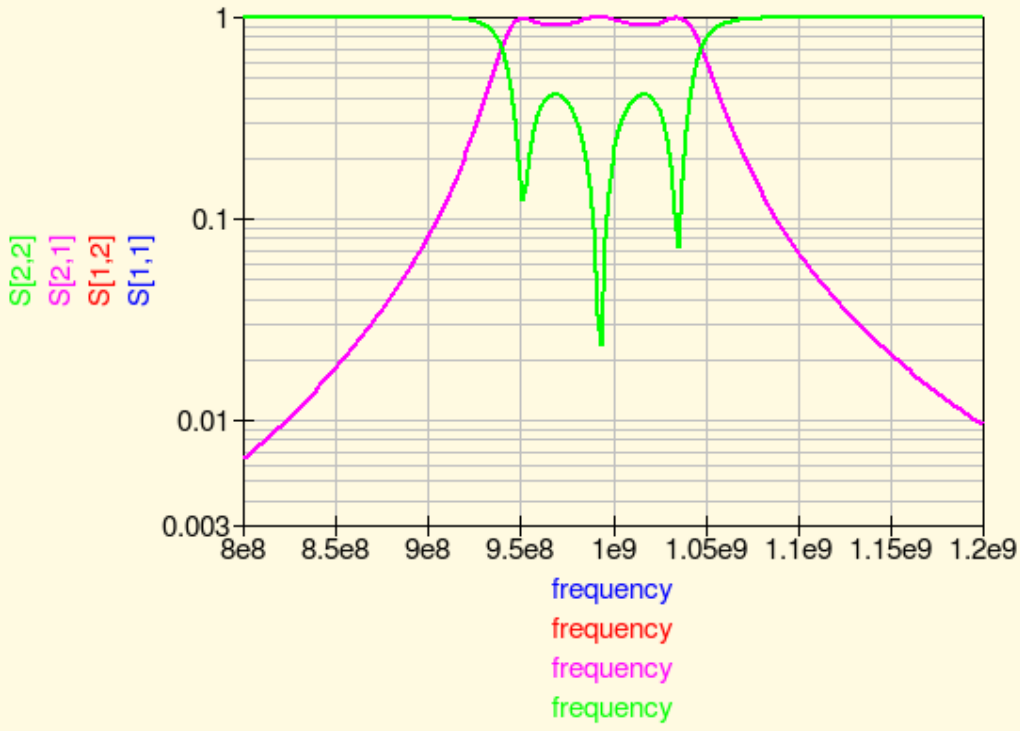


Рис. 10.43. Изменение схемы при подготовке к оптимизации

Теперь осталось только выполнить оптимизацию, например, нажав клавишу ***F2*** на клавиатуре.

Когда процесс завершится, что займет некоторое время (а на Windows Vista, возможно пока, процесс и не идет), можно получить графики и таблицу в окне отображения данных.



number	Left_Side_Lobe	Pass_Band_Ripple	Right_Side_Lobe	S11_In_Band
1	-22.1	-0.836	-23.7	7.57

Рис. 10.44. Отображение результатов оптимизации

Наилучшие параметры цепи можно найти в диалоге оптимизации на закладке ***Переменные***. Они имеют начальные значения для каждой представленной переменной.

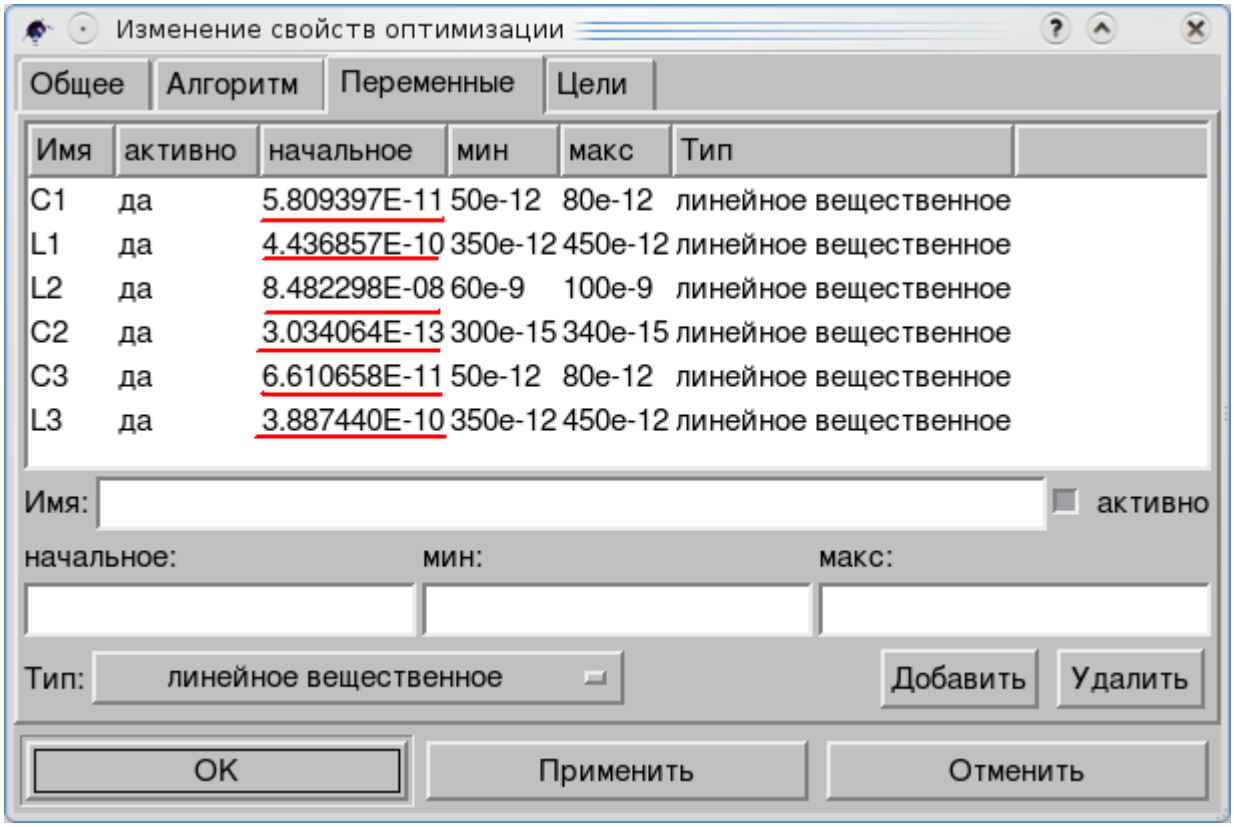


Рис. 10.45. Полученные оптимизированные значения

О том, почему процесс оптимизации не проходит на Windows Vista можно получить некоторое представление, если выбрать в основном меню раздел **Моделирование** и в нем пункт **Показать последние сообщения**.

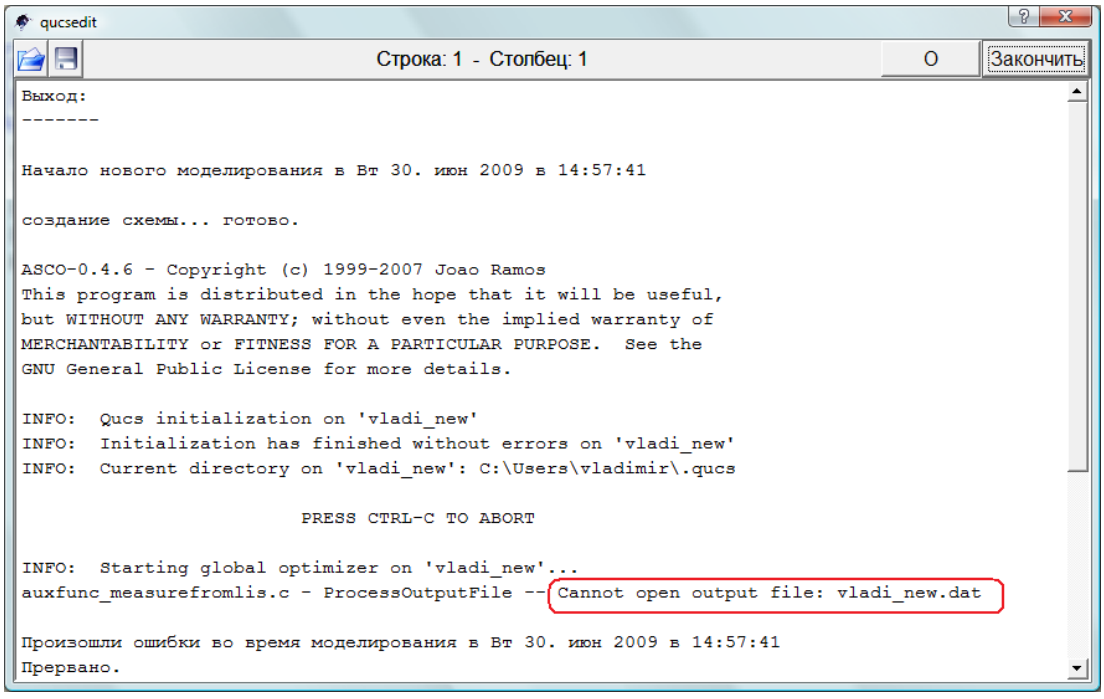


Рис. 10.46. Отображение сообщений в Qucs

Похоже, операционная система не позволяет создать файл с данными, что, видимо, как-то исправляется, но…

И еще раз о моделировании S-параметров. Мы уже говорили о малосигнальных параметрах. На высоких частотах (т.е., микроволновых) использование переменных мощности и энергии подходит больше, а двухпортовое ток-напряжение приближение заменяется приближением, базирующимся на параметрах матрицы рассеяния (S-параметрах). Повторим схему рисунка 5.2.

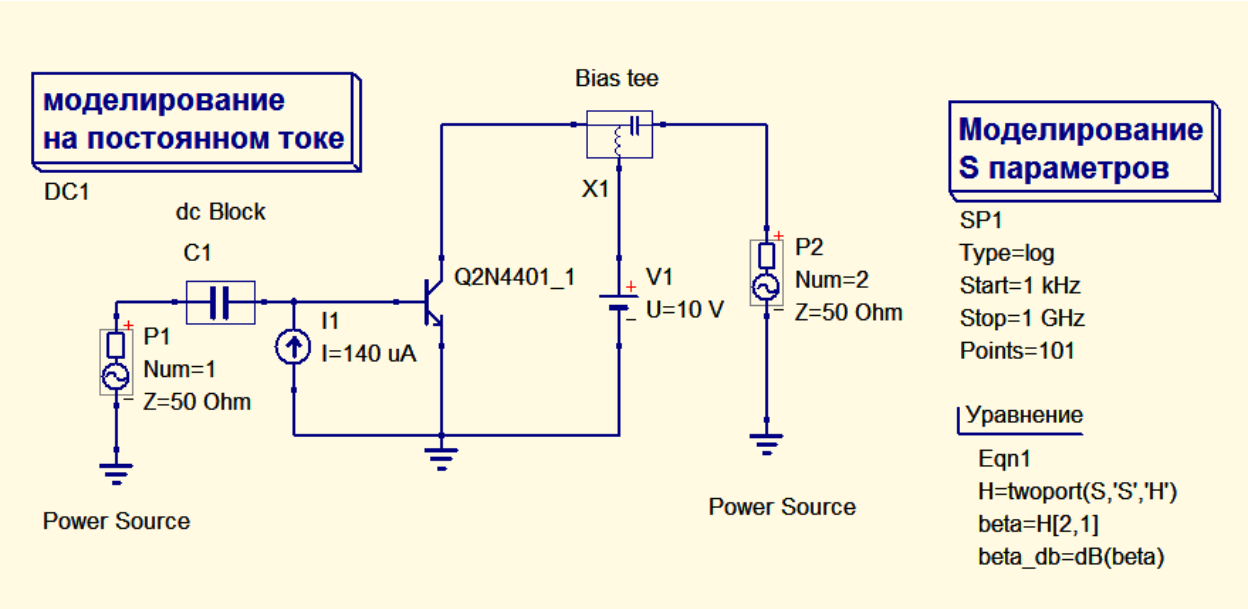


Рис. 10.47. Получение S-параметров

Подобно установкам для **моделирования на переменном токе**, установки **S-параметров** на картинке выше используют те же смещения. Установки будут использоваться для определения частоты единичного усиления биполярного транзистора (схема была описана выше, но есть резон повторить еще раз).

Два **Источника питания** P1 и P2 требуются для двухпортового **Моделирования S-параметров**.

Двухпортовая сеть (или четырехполюсник) — это электрическая цепь или устройство с двумя парами выводов. Два вывода образуют порт, если они удовлетворяют основным требованиям, известным как условия порта: тот же ток должен входить и выходить из порта. Примеры включают малосигнальные модели транзисторов, фильтры и согласующие схемы. Анализ пассивных четырехполюсников — следствие обратимости теорем впервые выведенных Лоренцем. Четырехполюсник делает возможным выделение либо всей цепи или ее части и замену на характеристические параметры. Когда это сделано, выделенная часть становится «черным ящиком» с набором характерных свойств, позволяющих нам абстрагироваться от физического содержания, что упрощает анализ. Любая линейная цепь с четырьмя выводами может быть преобразована в двухпортовую сеть при условии, что она не содержит независимых источников и удовлетворяет условиям порта.

Источники могут быть найдены на закладке **Компоненты** в группе **источники**. В зависимости от числа этого рода источников выполняется однопостовое, двухпортовое или многопортовое моделирование. Свойство *Num* источников определяет размещение входов в результирующей

матрице S-параметров. Свойство *Z* определяет базовый импеданс (опорное полное сопротивление) S-параметров.

Дополнительная **развязка от постоянного тока** (группа дискретные компоненты) *C1* в базовой цепи и **Смещение Т** (дискретные компоненты) *X1* в коллекторной цепи используются для развязки между источниками постоянного тока, задающими смещение, и внутренним импедансом **Источников питания**. Также компонент ***Смещение Т*** гарантирует, что сигнал от источника P2 не будет закорочен через **Источник напряжения постоянного тока** V1. Такой же результат достигается для **источника постоянного тока** I1 в базовой цепи. Это представлено идеальным открытым источником переменного тока.

Само **Моделирование S-параметров** определяется размещением блока SP1 на схеме. Уравнения содержат функцию двухпортового преобразования, которое преобразует результирующий параметр S в соответствующий H-параметр H. И теперь усиление по переменному току h21 вычисляется и преобразуется в dB.

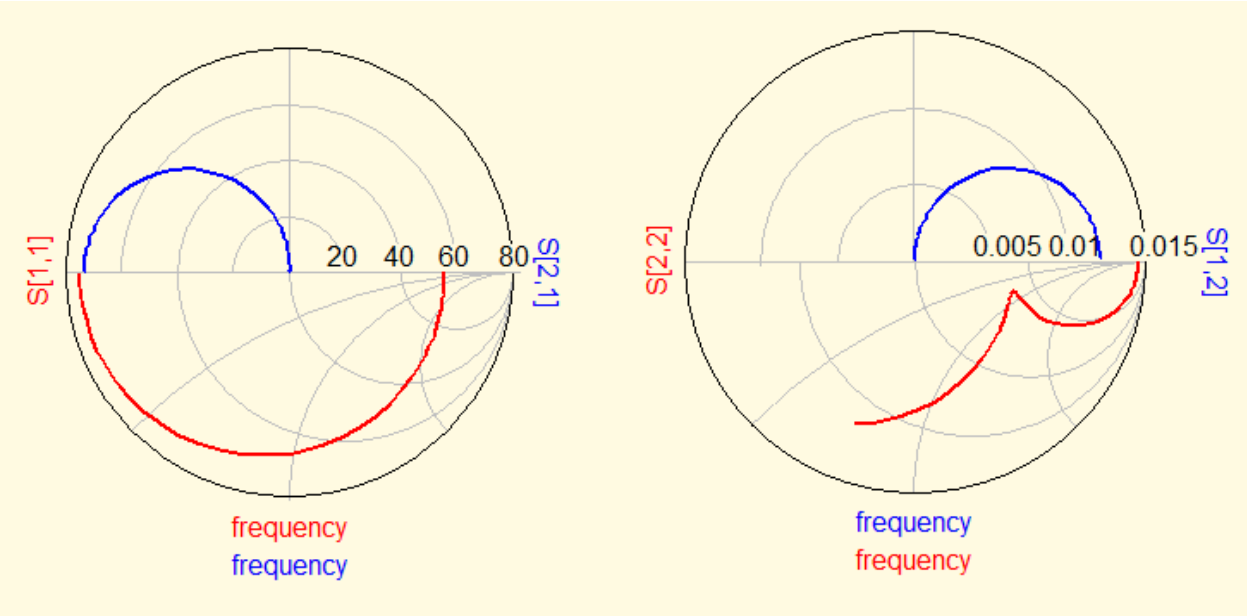


Рис. 10.48. Отображение данных в виде диаграмм Смита

Четыре комплексных S-параметра отображены на двух **Полярных смешанных диаграммах Смита** (закладка **Компоненты** панели навигации, группа **диаграммы**), которые показывают, что можно ожидать от типичного биполярного транзистора.

Для получения S-параметров биполярного транзистора в таком виде вы должны использовать свойства диаграмм. Для первой диаграммы в диалоговом окне выбираем, как обычно двойным щелчком в левом окне S11 и S21. После их перемещение в правое окно для первой кривой оставляем свойства, показанные ниже.

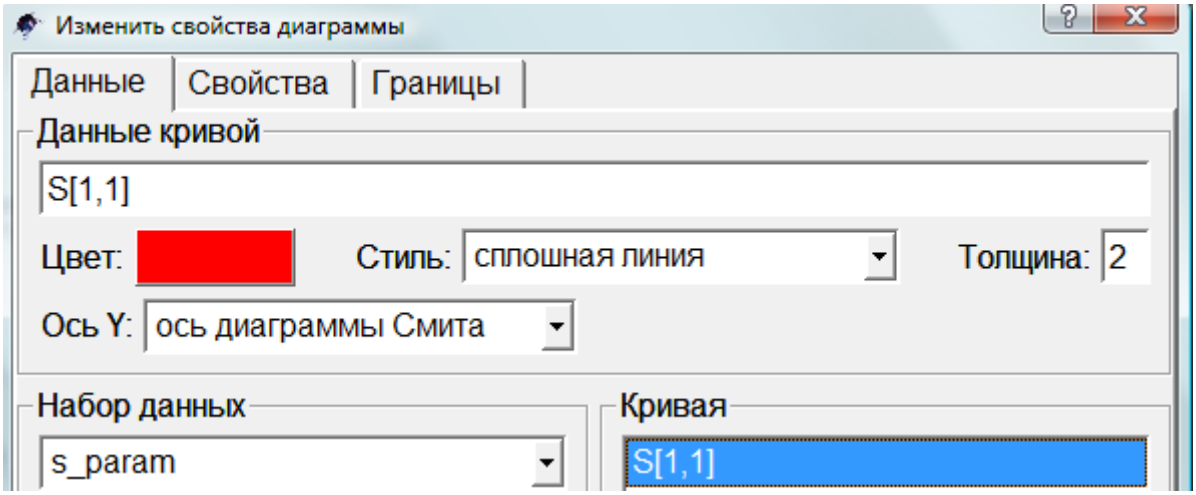


Рис. 10.49. Выбор свойств кривой S11

А для второй кривой выбираем новый вариант отображения оси Y.

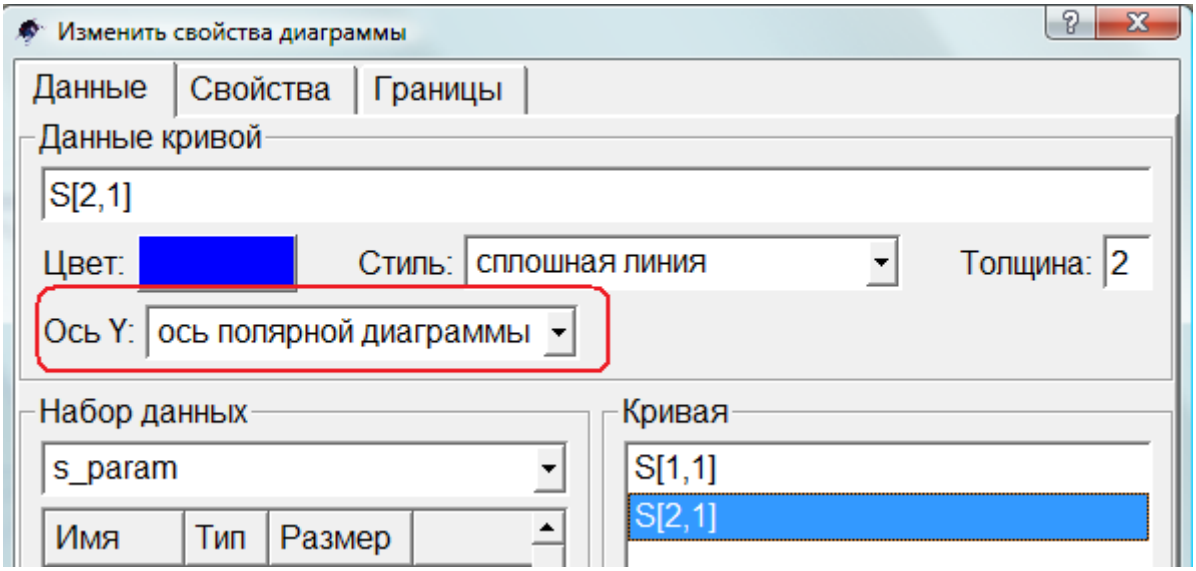


Рис. 10.50. Выбор свойств кривой S21

Используя вычисленные H-параметры, мы можем сравнить результаты **Моделирования S-параметров** с теми, что получены в **Моделировании на переменном токе** (оба файла должны быть в папке одного проекта).

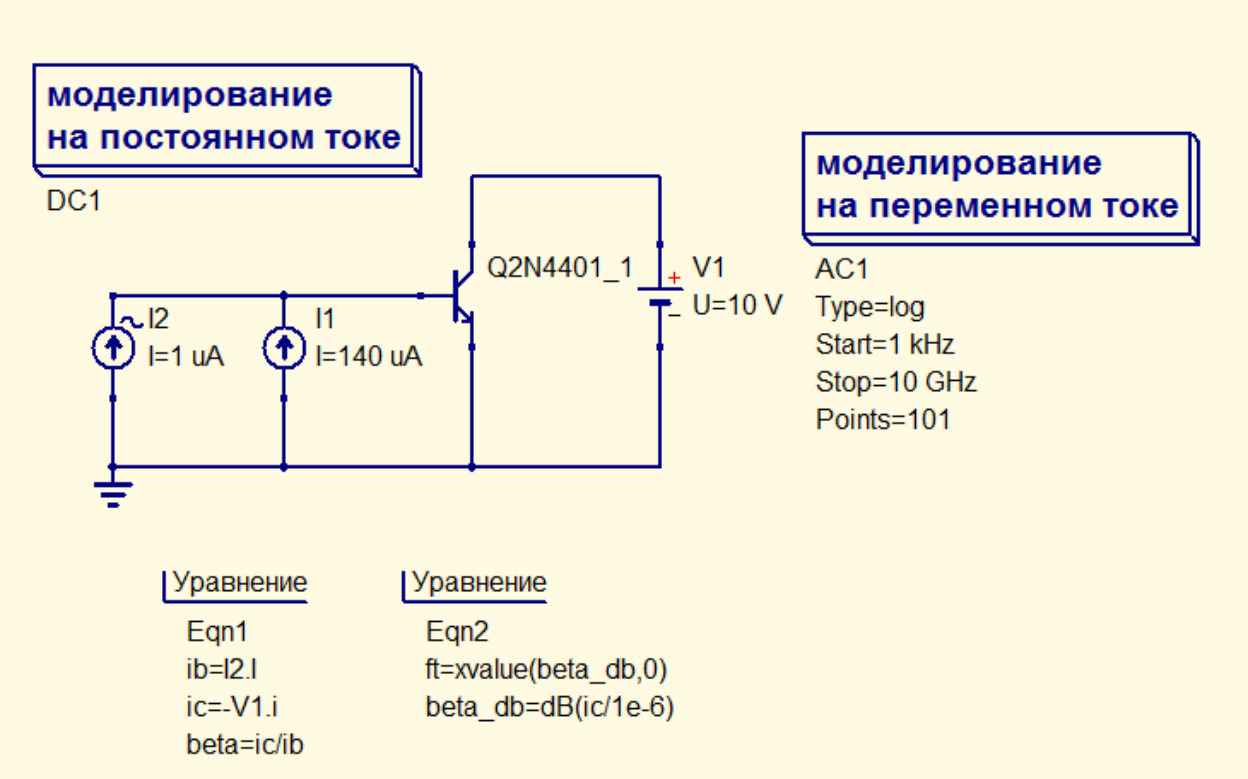


Рис. 10.51. Моделирование на переменном токе

После моделирования этой схемы на переменном токе добавляем **Декартовскую** диаграмму в поле отображения данных. А в свойствах диаграммы (не забыв на вкладке Свойства установить опцию: *логарифмическая разметка оси X*, как мы не забыли выбрать свойствах моделирования тип: *логарифмический*) задаем обычные параметры, выбрав в качестве данных bjtac.

Отображение кривой выбрано сплошной линией, а сама кривая, определяемая уравнением, это beta_db.

На той же диаграмме мы отобразим результат другого моделирования.

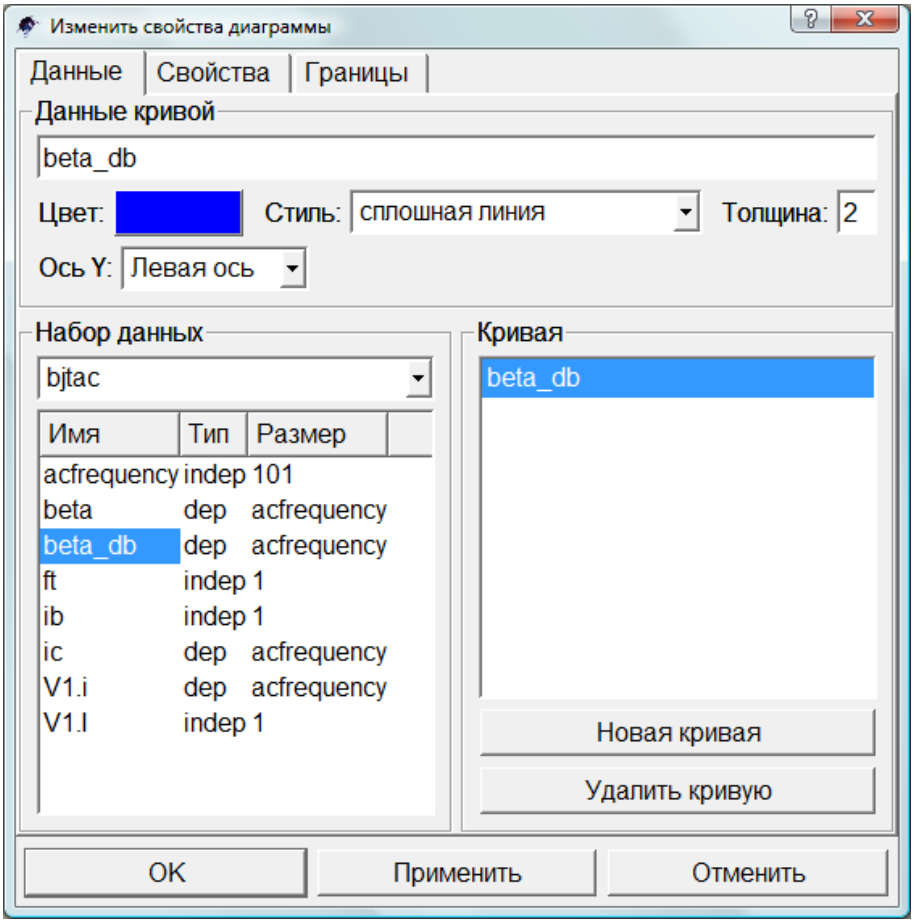


Рис. 10.52. Задание свойств первой кривой

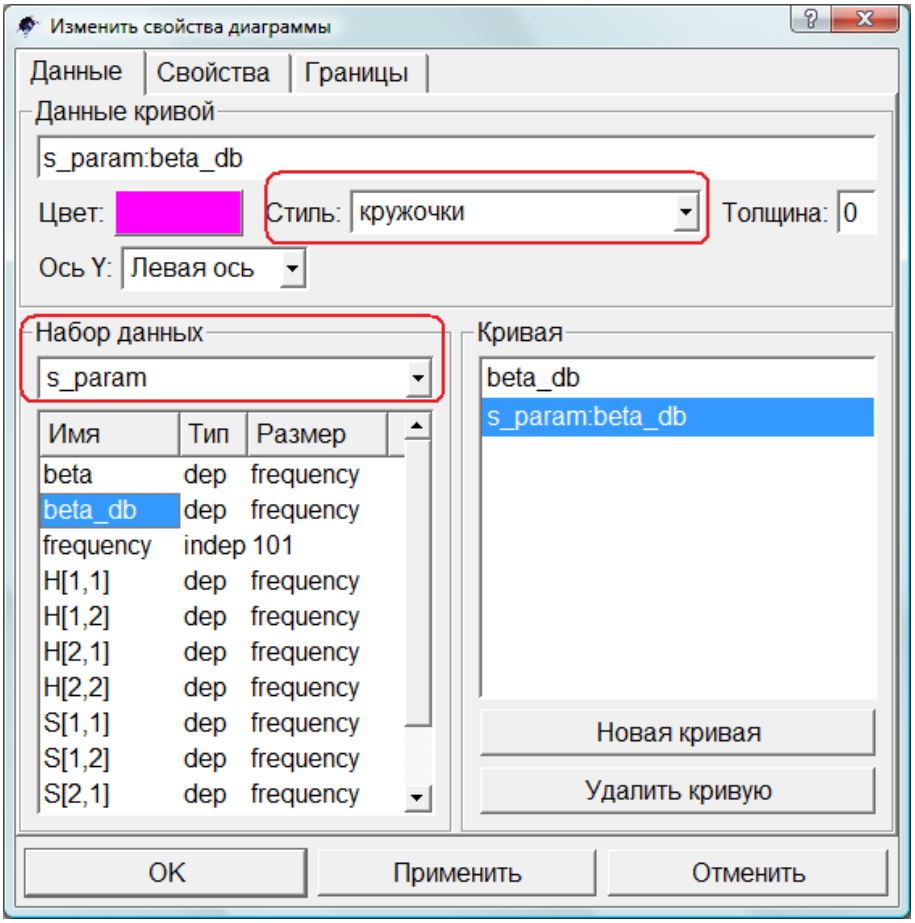


Рис. 10.53. Задание свойств второй кривой

Для второй кривой из набора данных выбраны данные, полученные при первом моделировании. И тогда обе кривые, совпадающие полностью, могут отобразиться на одном графике.

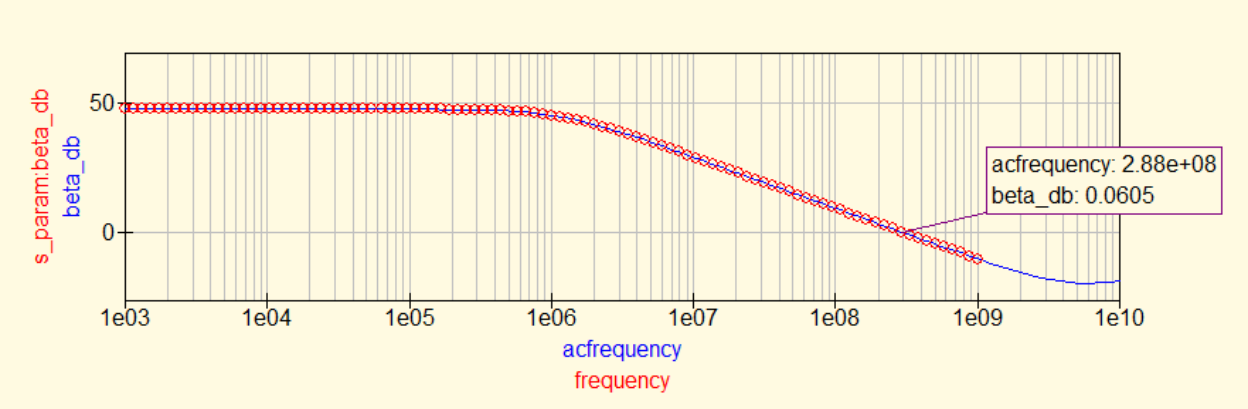


Рис. 10.54. Отображение обеих кривых, полученных разным способом

Частота единичного усиления, как выясняется, для этого транзистора 288 МГц.

Пополнение компонентов

Выше уже рассказывалось о пополнении **Библиотеки компонентов** за счет создания пользовательской библиотеки, рассказывалось о создании компонентов с помощью **файла SPICE**.

Множество основных компонентов схем, найденных для SPICE, пригодны и в Qucs. За последние три десятилетия язык netlist симулятора цепей SPICE стал стандартом для описания, обмена и публикаций моделей полупроводниковых устройств и данных электрических цепей. Сегодня большинство производителей полупроводников предоставляют SPICE модели или netlists подсхем своих дискретных компонентов или интегрированных схем.

Одна область, где Qucs и SPICE различаются очень существенно, это в их форматах файла netlist электрической цепи. Qucs не может непосредственно симулировать стандартные цепи SPICE netlist, необходимо конвертировать их в Qucs эквивалент до симуляции. Цель этого раздела, во-первых, ввести читателя в технику, позволяющую симулировать SPICE netlist в Qucs, во-вторых, показать те ограничения, что накладывает процесс конвертации, и, наконец, показать, как, вероятно, Qucs будет доработан в будущем для совместимости со SPICE netlist.

Входные данные симуляции SPICE — это текстовые файлы, которые описывают структуру цепи, данные компонентов и требуемые задачи симуляции для цепи, чьи характеристики подлежат симулированию. Подобные текстовые файлы формируют основные входные данные для механизма SPICE симулятора, и обычно включают:

- Заглавное предписание
- Имена узлов схемы
- Значения элементов схемы
- Описания источников напряжения и тока
- Команды анализа
- Предписания выходных данных
- Описания остальных команд

В SPICE 2 имена узлов цепей (сетей) идентифицированы целыми числами от 0 до 9999. SPICE 3 позволяет смешивать буквы и цифры для имен узлов. Все узлы схемы должны иметь путь протекания тока к земле. Узел земли всегда node 0 и рассматривается глобальным.

Значения элементов схемы выражаются целыми или действительными числами в научной нотации, например, 5, 0.5e1 5.0 или в инженерной нотации, использующей суффиксы. Допустимы в SPICE суффиксы: f =1e-15 (femto), p =1e-12 (pico), n = 1e-9 (nano), u = 1e-6 (micro), mil = 25e-6, m = 1e-3 (milli), k =1e3 (kilo), meg = 1e6 (mega), g = 1e9 (giga) и t = 1e12 (tera). Аббревиатуры единиц компонентов допустимы в описании значений схемы. Однако она не должна быть отделена от присвоенных им значений пробелами. Аббревиатуры наиболее общих единиц: V = Volt, A = Amps. Hz = Hertz, ohm = Ohm(Ω), H = Henry, F = Farad и deg =Degree. Файлы входных данных SPICE имеют следующий формат:

1. Заголовок, начинающийся со звездочки (*), знака строки комментария
2. Описание схемы
3. Директивы симуляции
4. Директивы выходных данных .end

Типичный файл входных данных SPICE для цепи с дискретными компонентами показан ниже:

```
* A two-stage BJT amplifier.
*
* Input node 2, output node 9
* Power supply Vcc connected to node 10
*
c1 2 3 10uf
r1 3 10 200k
r2 3 0 50k
r5 10 4 12k
q1 4 3 5 qmod
r6 5 0 3.6k
c2 4 6 10uf
c4 5 0 15uf
r3 10 6 120k
r4 6 0 30k
r7 10 7 6.8k
q2 7 6 8 qmod
r8 8 0 3.6k
c5 8 0 25uf
c3 7 9 10uf
*
.model qmod npn (is=0.2fa bf=50 br=1 rb=5 rc=1 re=0
+ cje=0.4pf vje=0.8 me=0.4 cjc=0.5pf vjc=0.8 ccs=1pf va=100)
*
.end
```

В этом netlist все узлы показаны числами, следуя соглашению об именах узлов в SPICE 2. Также источник питания, генератор сигнала переменного напряжения и выходная нагрузка здесь не включены. По существу, netlist, представленный выше описывает усилитель без внешних компонентов, подключенных к нему. Хотя Qucs не может непосредственно симулировать SPICE netlists, программа имеет часть, названную QUCSCONV, для конвертации из SPICE в Qucs netlist. Эта программа берет в качестве входного файл SPICE netlist и выдает эквивалентный файл формата Qucs netlist. Qucs netlist файл может читаться и симулироваться механизмом Qucs симулятора.

Чтобы сделать процесс прозрачным и действительно понятным пользователю, этап конвертирования симулируемого файла SPICE netlist был автоматизирован через Qucs GUI команду моделирования (клавиша ***F2***).

Файлы SPICE netlist могут быть привязаны к Qucs SPICE netlist схемному символу, который, в свой черед, может на схеме быть соединен с любым другим подходящим символом компонента Qucs или с символом, определенным пользователем. Рисунок ниже показывает результирующую схему для двухкаскадной BJT цепи.

На этой диаграмме внешние источники напряжения и нагрузка усилителя были добавлены, как обычные иконки Qucs для моделирования цепи и на постоянном, и на переменном токе. В процессе симуляции Qucs интерпретирует компоненты SPICE netlist, как подсхему и генерирует соответствующий Qucs netlist код. Например, netlist, показанный далее, иллюстрирует стиль Qucs netlist кода для двухкаскадного BJT усилителя.

Симуляция усилителя дает выходные осциллограммы, показанные под схемой.

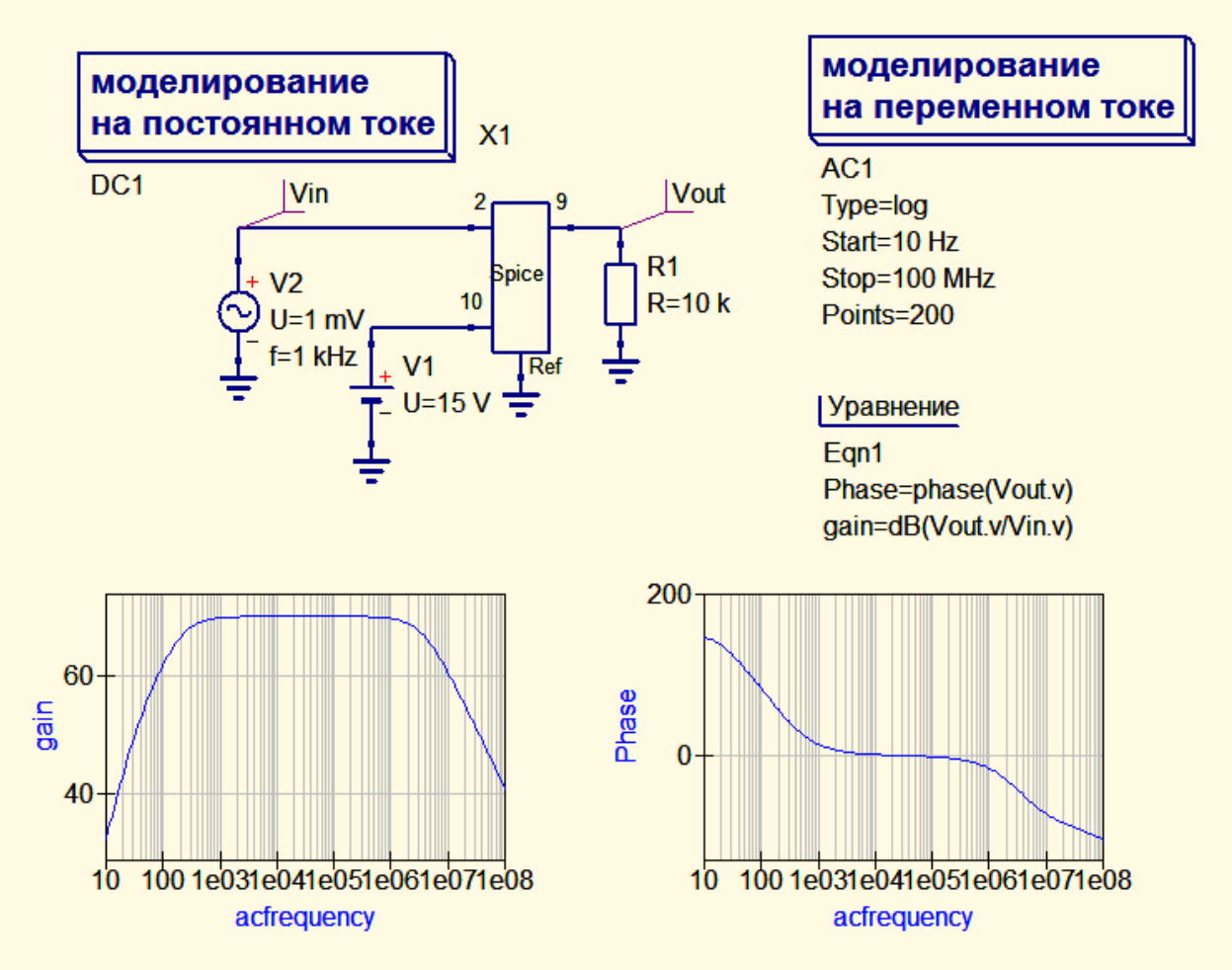


Рис. 10.55. Симуляция схемы, заданной SPICE-моделью

Вид netlist Qucs, к которому приведен текст схемы, можно получить, используя в основном меню раздел **Моделирование**, пункт которого **Показать последнюю схему** выводит текст в формате Qucs. И не забудьте, что если вы берете текст spice-файла из другого текстового файла, будьте внимательны: «100 k» и «100k» — это разное написание.

Глава 2. Теория и практика

Встретив на одном из форумов сетования по поводу неработающей в программе Multisim 10 схемы мультивибратора, я очень удивился. Да, было время, когда подобная проблема существовала, но это время давным-давно прошло. Схема, прикрепленная к сообщению, содержала ряд ошибок, исправив и повторив ее в предыдущей версии программы Multisim, я убедился, что схема симулируется, как ей и положено.

Разные симулирующие программы предлагают разные варианты отображения работающей схемы. Так программа Proteus позволяет создать анимацию работающей схемы. Такой вариант, конечно, будет очень полезен для демонстрации в школе на уроке физики. Однако обычной практикой является использование приборов. Один из наиболее удобных для понимания того, как создаются, трансформируются и «путешествуют» по схеме электрические сигналы, приборов – это осциллограф. Поэтому программы симуляции чаще всего показывают осциллограммы переменных напряжений и токов.

Вот как выглядит и как работает мультивибратор, о котором речь шла выше, в Qucs.

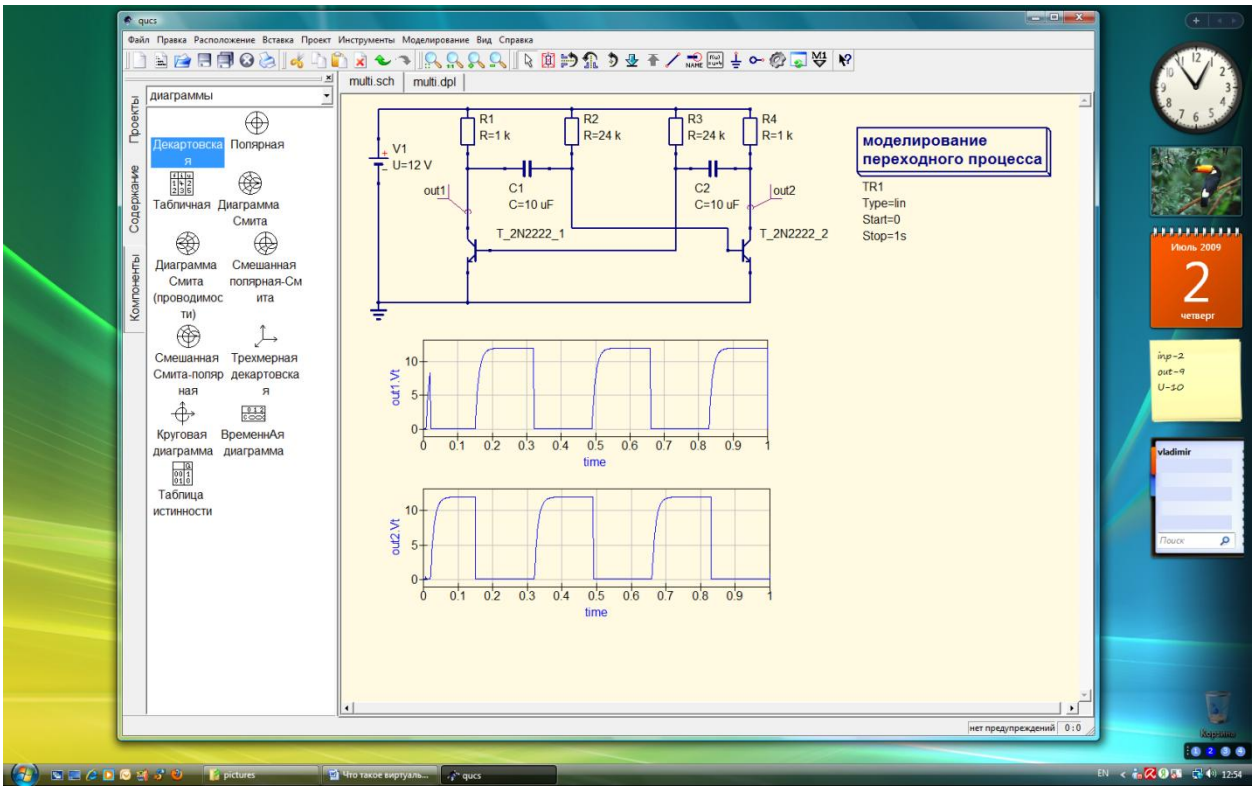


Рис. 11.1. Работа мультивибратора в Qucs

Очень многие, и не только любители, весьма скептически относятся к использованию программ САПР в освоении электроники. Мне кажется, они очень неправы. Хотя мне не довелось использовать, скажем, программу Qucs в профессиональной разработке, чтобы безапелляционно утверждать ее полную пригодность к этому, я решил проделать несколько простых опытов – сравнить результаты, полученные в программе с теми, что получены повторением схемы на макетной плате.

Диод

Без претензии на общность я хочу провести сравнение вольтамперных характеристик диода 1N4004 в программе Qucs и на макете. Отчего именно этот диод? Его можно сегодня купить в магазине, а программы редко поддерживают отечественную элементную базу.

Есть два способа в программе Qucs проделать опыт с диодом. Первый способ ближе к тому, что предстоит воплотить в жизнь на макетной плате.

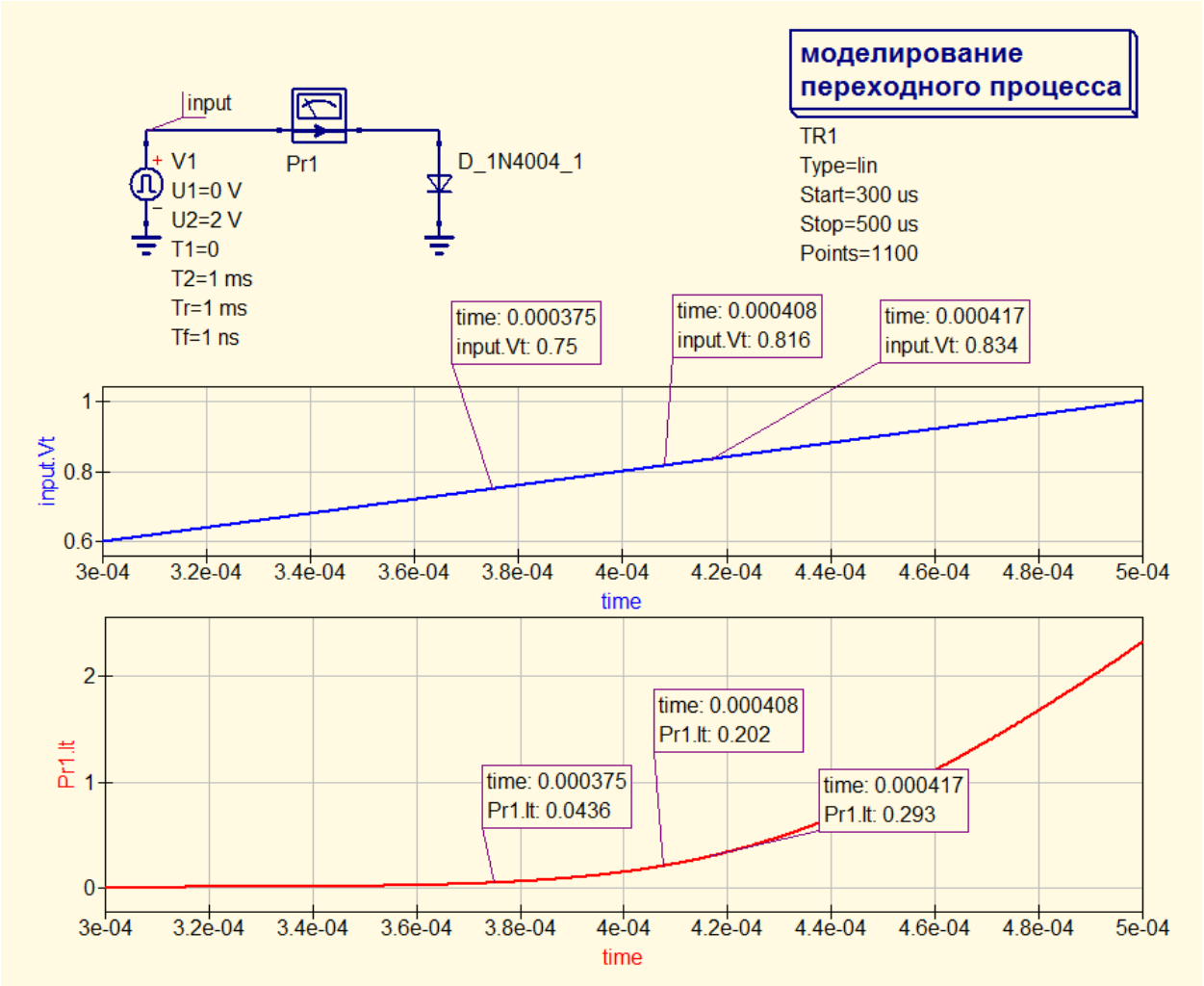


Рис. 11.2. Первый вариант получения вольтамперной характеристики диода

В качестве источника тока можно применить **источник импульсного напряжения**, параметры которого настроены так, чтобы получился источник линейно нарастающего напряжения (метка *input*). Настройки должны быть ясны из рисунка. Измеритель тока Pr1 даст зависимость тока от времени. Используя маркеры, расставленные в точках с одинаковым временем на обеих диаграммах, мы получим несколько значений тока при выбранных напряжениях.

Второй способ получения ВАХ (вольтамперной характеристики) диода ближе к тому, что мы получили бы, если бы использовали специализированный прибор для снятия характеристик полупроводниковых приборов.

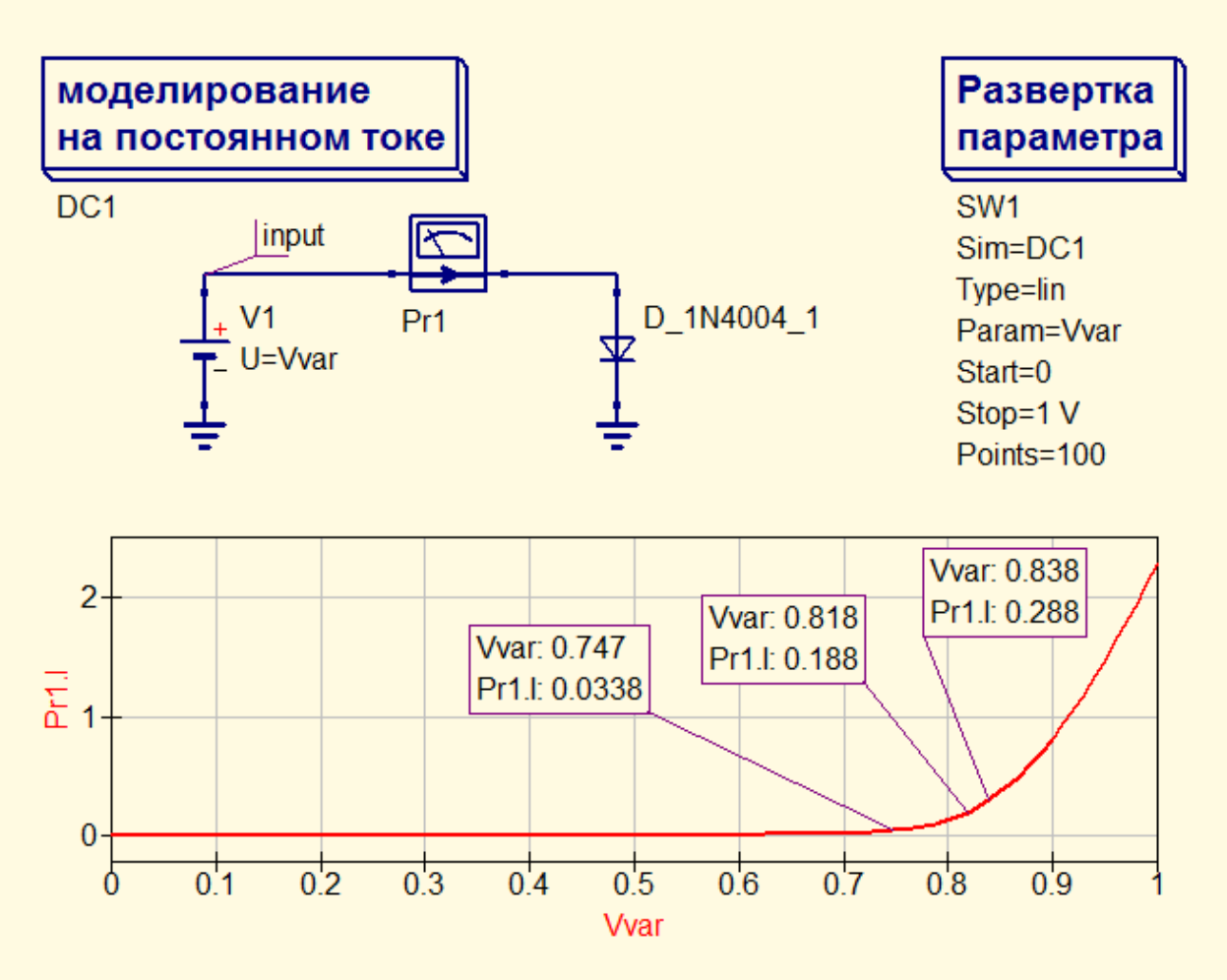


Рис. 11.3. Второй вариант построения ВАХ диода

В этом случае использован вид моделирования – **Развертка параметра**. Заменив напряжение источника питания переменной *Vvar*, мы используем ее для получения развертки параметра. Впрочем, мы могли бы использовать не источник напряжения, а источник постоянного тока.

Метод развертки параметра дает еще одно преимущество – можно получить табличные значения, задав разумное количество точек измерения и пределы напряжения.

Ниже на картинке отмечены точки соответствующие тем, что отмечены маркерами выше.

Vvar	Pr1.I
0.75	0.0361
0.761	0.0479
0.772	0.0634
0.783	0.0833
0.794	0.109
0.806	0.141
0.817	0.182
0.828	0.231
0.839	0.291
0.85	0.361

Для вывода таблицы в диалоговом окне Развертки параметров заданы начальная точка 0.75 V, конечная точка 0.85 V, а количество точек выбрано равное 10.

Рис. 11.4. Данные в виде таблицы

А ниже показана таблица результатов, полученных на макетной плате.

Результаты «прямого» эксперимента	
Напряжение, В	Ток, А
0.75	0.04
0.816	0.18
0.834	0.27

Разброс параметров, видимый в полученных данных, свидетельство в первую очередь того факта, что эксперименты проводились недостаточно строго. Так мощность, рассеиваемая на диоде при напряжении 0.85 В, близка к 0.25 Вт. А при такой рассеиваемой мощности температура диода будет отлична от комнатной, что вызовет изменение параметров. Но для практических целей точности опыта вполне достаточно.

Притом прямая выгода от использования программы – возможность оценить параметры, например, при токах в 1 и более ампер, что на макетной плате трудно повторить, и, скорее всего, приведет к выходу из строя диода.

Стабилитрон (или диод Зенера)

Любой диод имеет такой параметр, как предельно допустимое обратное напряжение. Если обратное напряжение (напряжение обратной полярности) превышает это значение, то возникает лавинообразно нарастающий обратный ток, который, скорее всего, приведет к выходу диода из строя.

Разные конструкции диодов имеют разное допустимое значение этого напряжения. А сам эффект – быстрое нарастание тока при незначительных изменениях напряжения выше допустимого – сам эффект используется в современной электронике с большой пользой. Специально разработанные диоды, которые называются стабилитроны, даже в самом простом случае позволяют создать устройства, называемые стабилизаторами напряжения. Назначение такого стабилизатора – поддерживать питающее напряжение постоянным (в заданных пределах) при изменении напряжения на входе стабилизатора в широких пределах.

В качестве примера рассмотрим диод 1N751 (на макетной плате используем близкий по назначению стабилитрон BZX5V1).

На рисунке ниже в окне графики выделен участок «перегиба» вольтамперной характеристики диода при обратной полярности включения. Маркерами отмечены два участка, где напряжение равно 5.08 В, а ток через стабилитрон 3 мА, и второй при напряжении 5.14 В с током через диод около 7 мА. Нет нужды пояснять, что малейшее изменение напряжения приводит к существенному росту тока через стабилитрон.

Обычно простейший стабилизатор напряжения с диодом Зенера дополняется добавочным резистором. При увеличении тока через стабилитрон, вызванного увеличением напряжения на входе стабилизатора, увеличивается падение напряжения на резисторе, тогда как на диоде напряжение остается в заданных пределах.

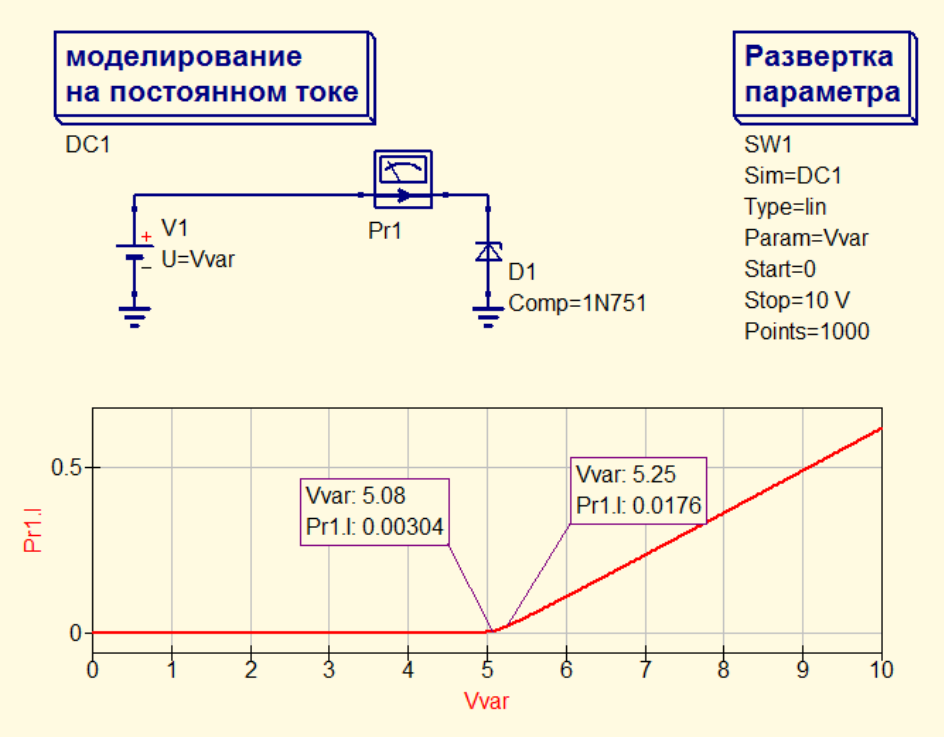


Рис. 11.5. Опыт с диодом Зенера

Проверка реального диода Зенера	
Напряжение, В	Ток, мА
4.38	2.6
4.88	18

Разброс значений получается значительно больше, чем в предыдущем опыте. И сказывается не только различие в моделях, но и возможный разброс значений даже у разных представителей одной и той же модели. Паспортное напряжение стабилизации проверенного стабилитрона 5.1 В, тогда как при токе 18 мА оно не достигает и 5 В.

Однако моделирование явно показывает, что при незначительном изменении напряжения в зоне стабилизации происходит существенное возрастание тока. Чтобы приблизить модель в Qucs к реальному стабилитрону, хотя такой модели нет в Библиотеке компонентов, можно воспользоваться SPICE-моделью:

```
**********************************
* Model created by *
* Uni.Dipl.-Ing. Arpad Buermen *
* arpad.burmen@ieee.org *
* Copyright: *
* Thomatronik GmbH, Germany *
* info@thomatronik.de *
**********************************
* February 2001
* SPICE3
* anode cathode
* Reverse direction: node 1 <- node 9
.SUBCKT bzx55c5v1 1 9
DF 1 9 DFMOD
.MODEL DFMOD D N = 2 IS = 1.9E-009 RS = 0.36459
+ EG = 1.11 XTI = 3
+ CJO = 1.71713E-010 VJ = 0.490432 M = 0.5 FC = 0.5
+ TT = 1E-008 TNOM = 25
* Leakage
RL 1 9 1.05263E+007 RLMOD
.MODEL RLMOD R TC1 = 0 TC2 = 0 TNOM = 25
* Breakdown
ES 9 90 10 20 1
DR1 90 31 DREV1 TEMP = 25
.MODEL DREV1 D IS = 1E-015 N = 3.95147 RS = 0.00225008 TNOM = 25
VR1 31 1 0.00879249
DR2 90 32 DREV2 TEMP = 25
.MODEL DREV2 D IS = 1E-015 N = 2.92114 RS = 695.888 TNOM = 25
VR2 32 1 0.0793202
VTRIM 20 0 2.95119
IBVC 0 10 0.001
RBVC 10 0 5100 RBVCMOD
.MODEL RBVCMOD R TC1 = 0.00015 TNOM = 25
.ENDS
```

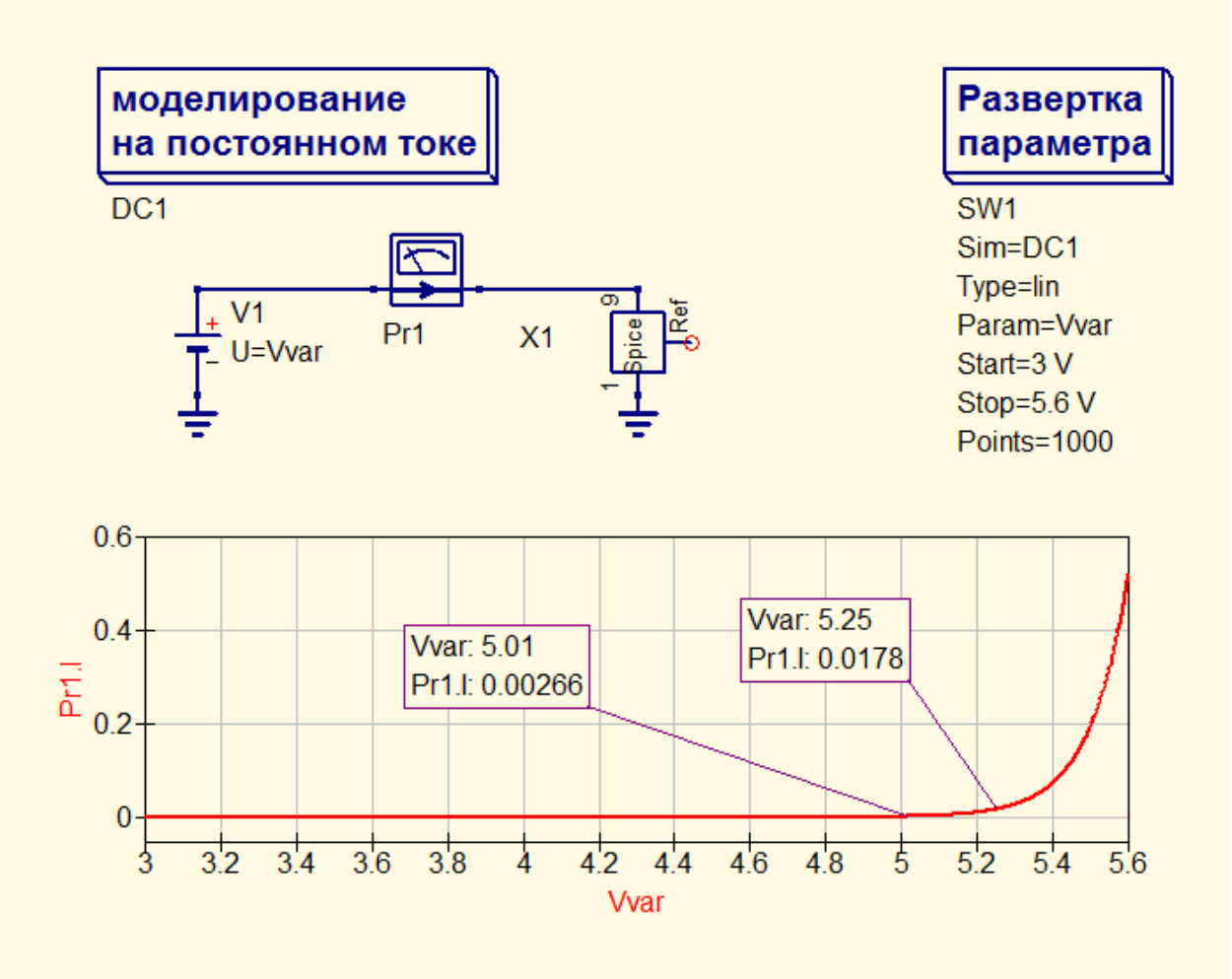


Рис. 11.6. Использование SPICE-модели диода BZX55C5V1

В данном случае моделирование мне кажется более убедительным, чем проверка на макете. Скорее всего, реальный диод не отличается качеством выполнения. Тем не менее, использование стабилитрона, скажем, в устройстве, выполненном на микросхемах ТТЛ, вполне даст допустимое напряжение при токе стабилизации в 15-20 мА.

Транзистор

Для биполярного транзистора можно построить разные характеристики. При проектных расчетах часто пользуются статическими входной (зависимость напряжения эмиттер-база от тока базы) и выходной (зависимость тока коллектора от напряжения коллектор-эмиттер) характеристиками для схемы включения с общим эмиттером.

Как и в случае с диодом, мы можем провести опыты с транзистором.

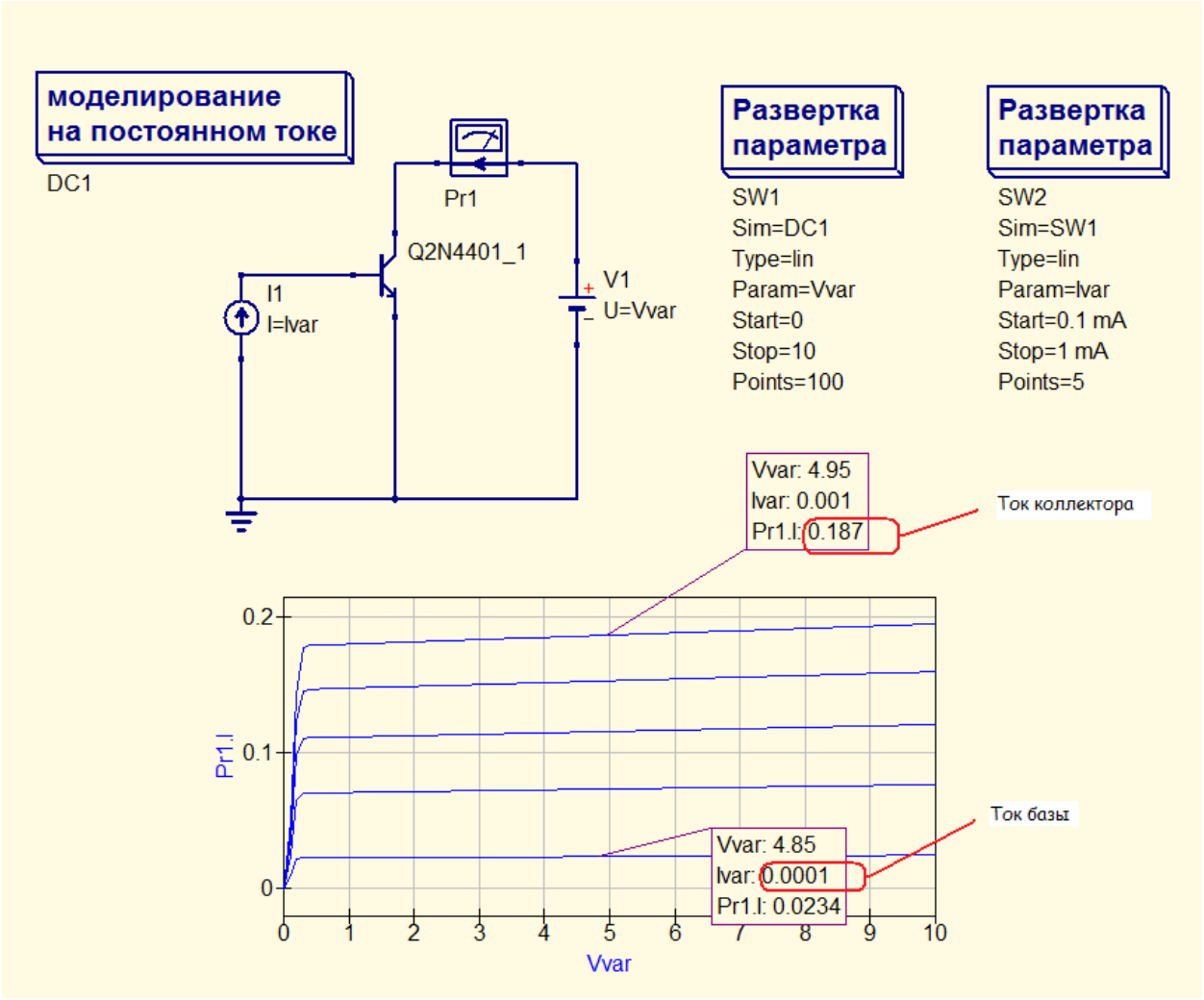


Рис. 11.7. Опыт по построению выходной характеристики биполярного транзистора

Но в практической деятельности чаще интерес представляет амплитудно-частотная характеристика транзистора. Из нее, в частности, можно сделать вывод о возможности применить транзистор в усилителе, предназначенном для работы в определенной полосе частот.

На рисунке ниже верхний маркер указывает верхнюю граничную частоту, а нижний частоту единичного усиления, что важно для тех случаев, когда транзистор используется как активный элемент генератора.

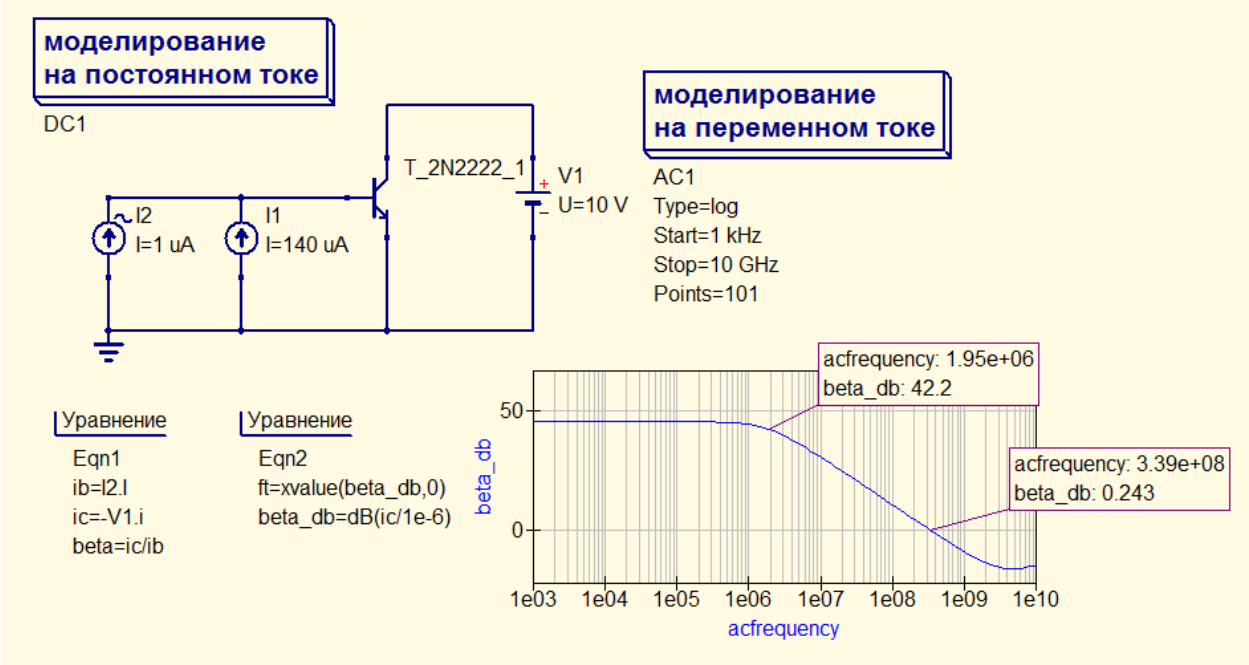


Рис. 11.8. Амплитудно-частотная характеристика транзистора 2N2222A

Можно, конечно, снять АЧХ усилительного каскада, построенного на транзисторе 2N2222A, но можно поступить иначе. Протестируем простейший усилительный каскад прямоугольными импульсами двух частот 100 кГц и 1 МГц.

На рисунке ниже верхняя диаграмма относится к эксперименту с частотой 100 кГц, нижняя, когда частота прямоугольных импульсов 1 МГц. Если на первой диаграмме прямоугольные импульсы достаточно «прямоугольны», то на нижней диаграмме заметны «затяжки» на фронтах.

Чем шире частотный диапазон усилителя, выше его верхняя частота среза, тем больше прямоугольные импульсы похожи на «прямоугольные». С приближением частоты прямоугольных импульсов к верхней рабочей частоте усилителя форма сигнала меняется – затягиваются фронты.

Если использовать генератор прямоугольных импульсов, то можно оценить полосу пропускания усилителя, конечно, «на глаз», визуально. В некоторых случаях этого оказывается достаточно. Но если вы предполагаете работать с усилителем, предназначенным для работы на радиочастотах, следует иметь в своем распоряжении соответствующий генератор.

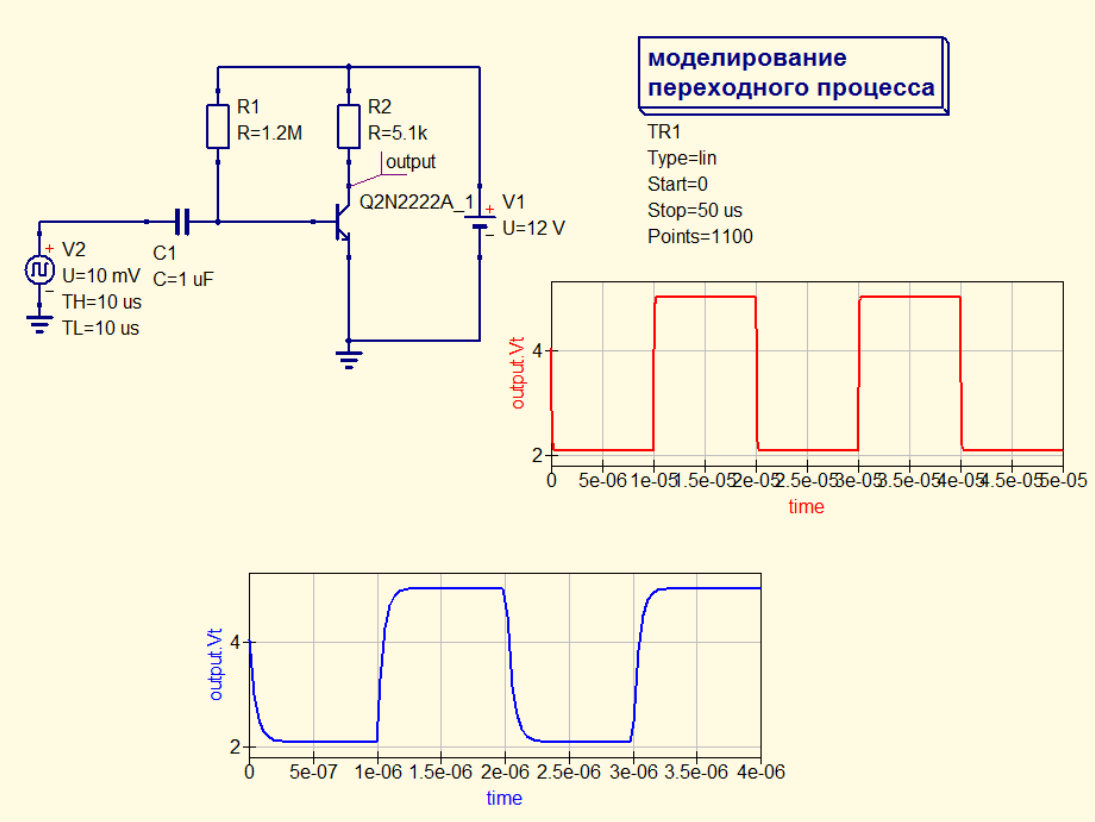


Рис. 11.9. Тестирование усилителя прямоугольными импульсами

А теперь посмотрим, что происходит на макете.

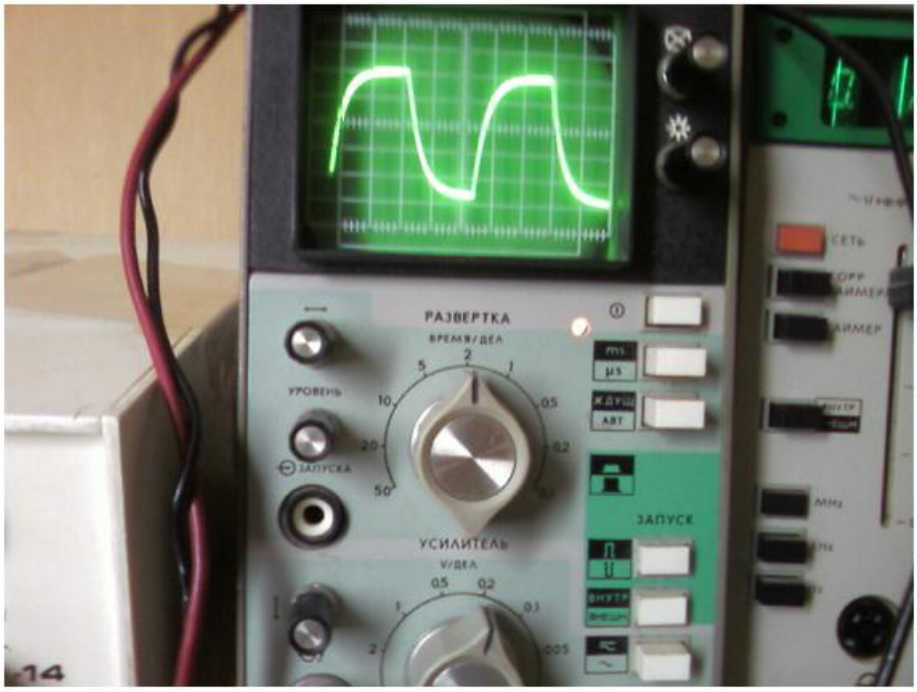


Рис. 11.10. Испытание усилителя прямоугольными импульсами с частотой 100 кГц

Форма импульсов, конечно, отличается от той, что представлена верхней осциллограммой рисунка 11.9. Однако причина не в плохой работе программы моделирования. Причина в том, что мы не учли всех особенностей реальной работы. На вход усилителя сигнал попадает от генератора, имеющего выходное внутреннее сопротивление, через кабель, имеющий емкость, а в программе идеальный источник. На вход осциллографа сигнал попадает через щуп осциллографа, имеющий входную емкость порядка нескольких десятков пикофарад.

Попробуем добавить в схему моделирования эти компоненты.

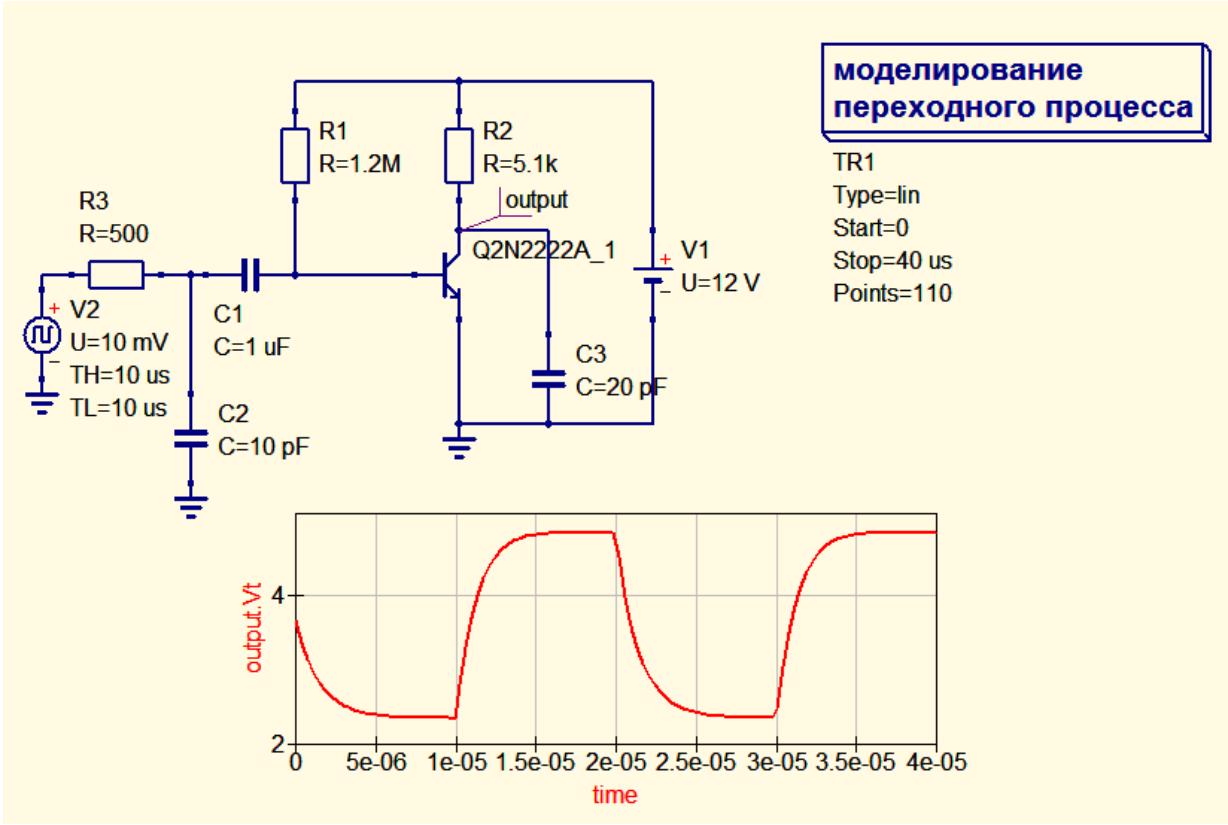


Рис. 11.11. Моделирование с учетом не идеальных условий

Кроме амплитудно-частотных искажений реальный усилительный каскад вносит и нелинейные искажения. Обычно нелинейные искажения измеряют на выходе усилителя мощности (или каскадов усиления мощности). Но при неправильном выборе рабочей точки нелинейные искажения могут появиться и в предварительных каскадах усиления. Для измерения нелинейных искажений используют специальные приборы, которых у меня (к сожалению, или естественно) нет. Но при больших искажениях они заметны визуально на экране осциллографа. Например, увеличивая сигнал от генератора синусоидального сигнала V2 (на рисунке выше), можно добиться вполне «видимых искажений».

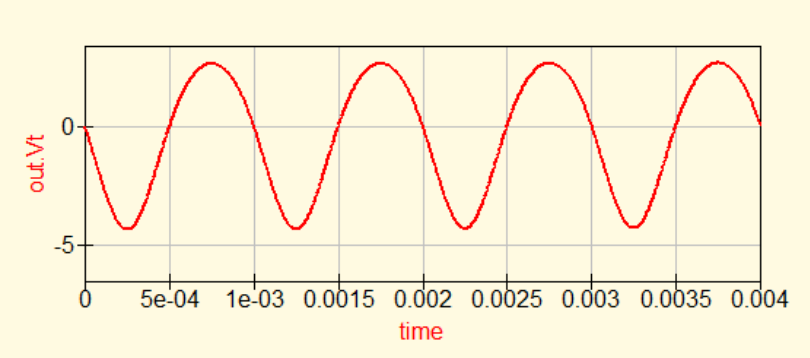


Рис. 11.12. Появление нелинейных искажений на осциллограмме

На макетной плате усилитель ведет себя схожим образом.

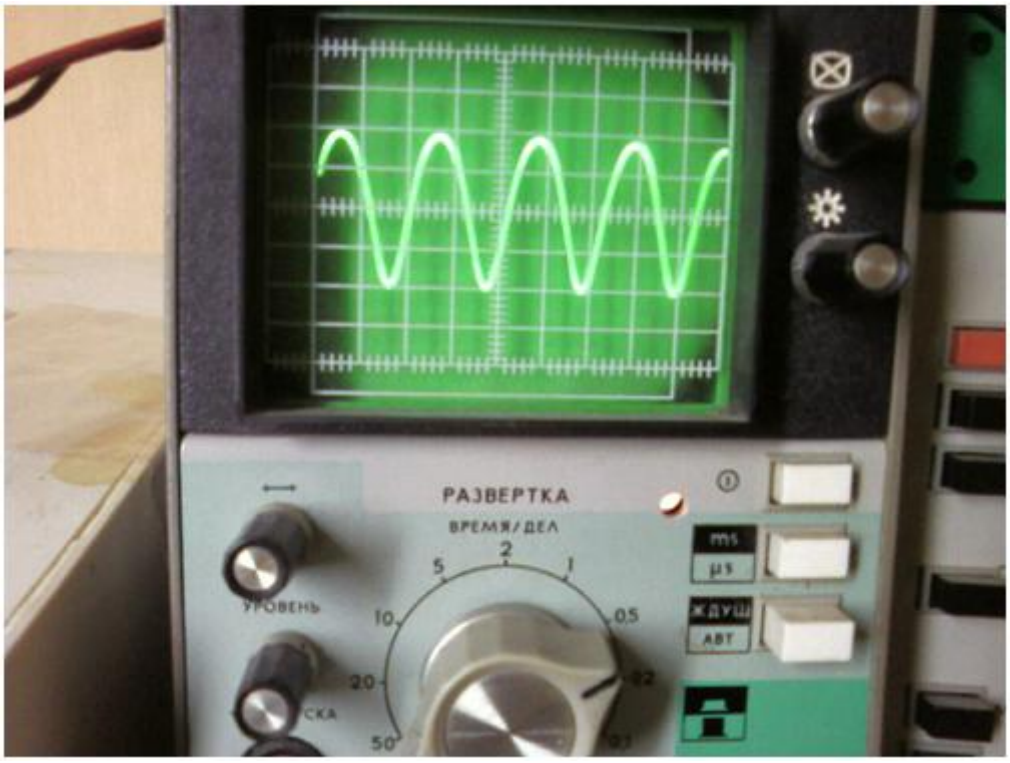


Рис. 11.13. Нелинейные искажения при перегрузке усилителя

Еще одна область успешной работы транзисторов – генераторы. Классический генератор – емкостная трехточка. Для успешной работы генератора необходимо выполнение условий самовозбуждения генератора.

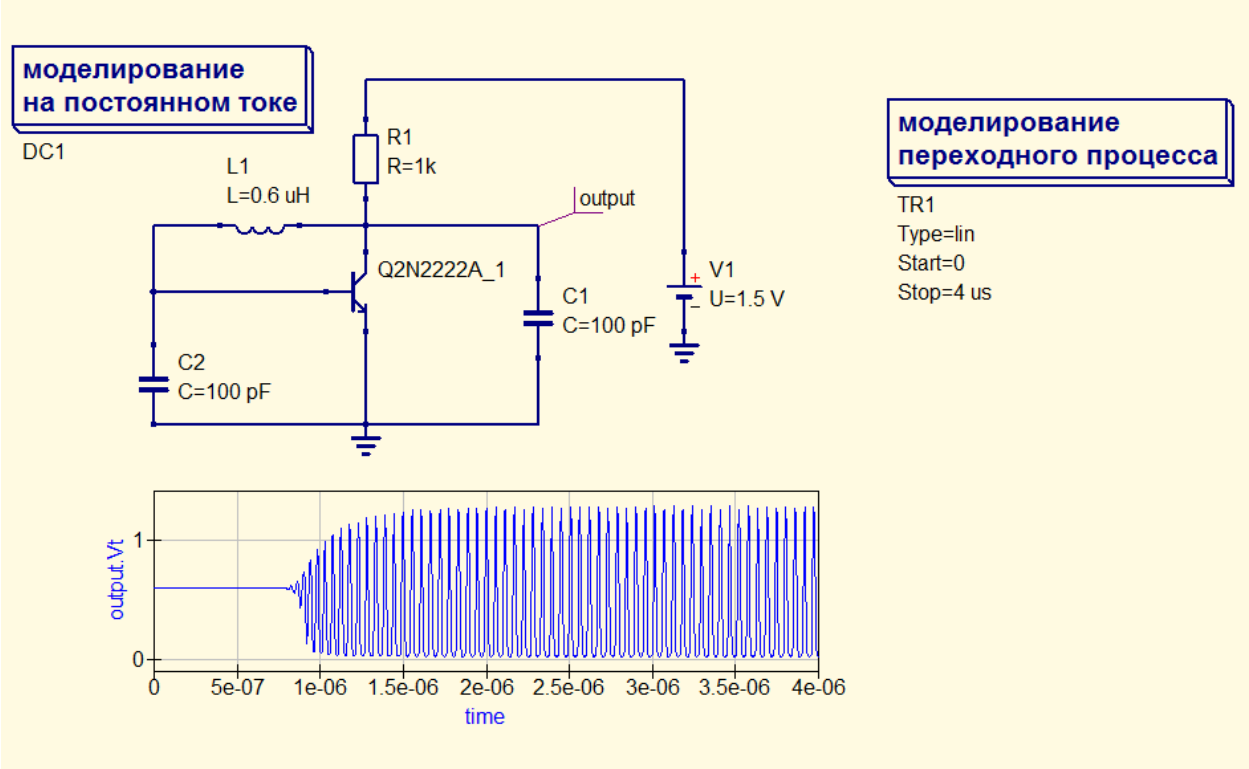


Рис. 11.14. Генератор по схеме емкостной трехточки

Повторив схему на макетной плате, получаем работающий генератор.

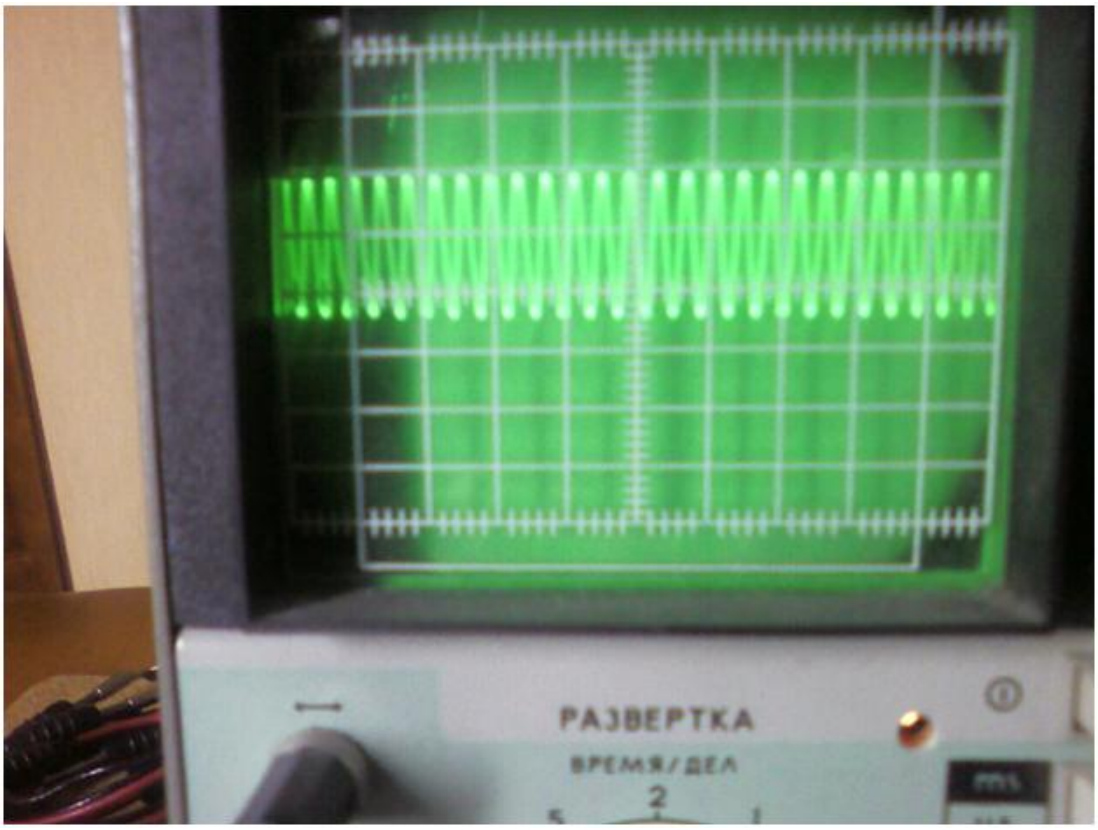


Рис. 11.15. Осциллограмма работающего генератора по схеме выше

Глава 3. Читая учебник

Любое овладение предметом, любительское ли, профессиональное, базируется на чтении учебника, а подкрепляется практической работой. Но, согласитесь, что, например, любитель, который открыл книгу, рассказывающую об электронике, готов после прочтения нескольких страниц обратиться к макету и паяльнику. Далеко не всегда есть подходящие детали, да и макетирование отнимает много времени. В еще большей мере это относится к студентам, которым и без того трудно выкроить время, чтобы сдать все зачеты.

В этом плане компьютер оказывает неоценимую услугу, позволяя повторить прочитанное (или многое из прочитанного) почти так же, как при проведении лабораторных работ.

Откроем учебник Джеймса Нильссона и Сьюзен Райэдел «Электрические цепи» и попробуем читать его за компьютером. Определения и пояснения используем те, которыми пользуются авторы.

Напряжение, ток, мощность

Итак. *Напряжение*:

U = dw/dq, где напряжение U в вольтах, энергия w в джоулях, а заряд q в кулонах.

Ток:

I = dq/dt, где ток I в амперах, заряд q в кулонах, время t в секундах.

Мощность:

P = dw/dt, где мощность P в ваттах, энергия w в джоулях, время t в секундах.

Мощность можно выразить и иначе: P = dw/dt = (dw/dq)(dq/dt) = U*I.

Далее авторы приводят ряд упражнений, одно из которых выглядит похожим на это:

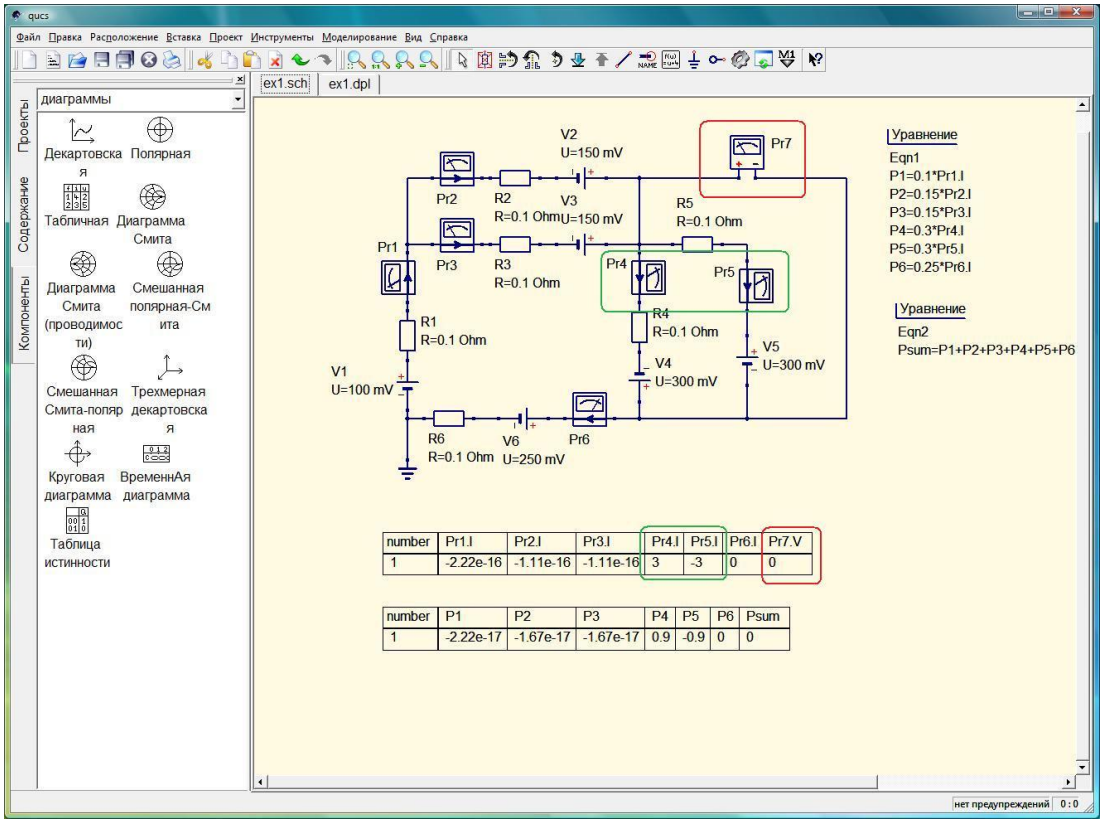


Рис. 12.1. Напряжения, токи, мощность

Правда, в авторском варианте эти упражнения сводятся к подсчету мощности с учетом полярности напряжения и направлений токов. Но мне показалось интересным изменить оригинал, чтобы получить отмеченные на рисунке особенности.

Возможность в программе одним щелчком мышки отразить источник относительно оси x или y, изменить значения, дает множество новых реализаций электрической цепи. Таким образом, кроме практики в том, что предлагают авторы, вы получаете практику в создании своих задач и их решении, а выглядит это, скорее, как увлекательная игра.

Элементы электрической цепи

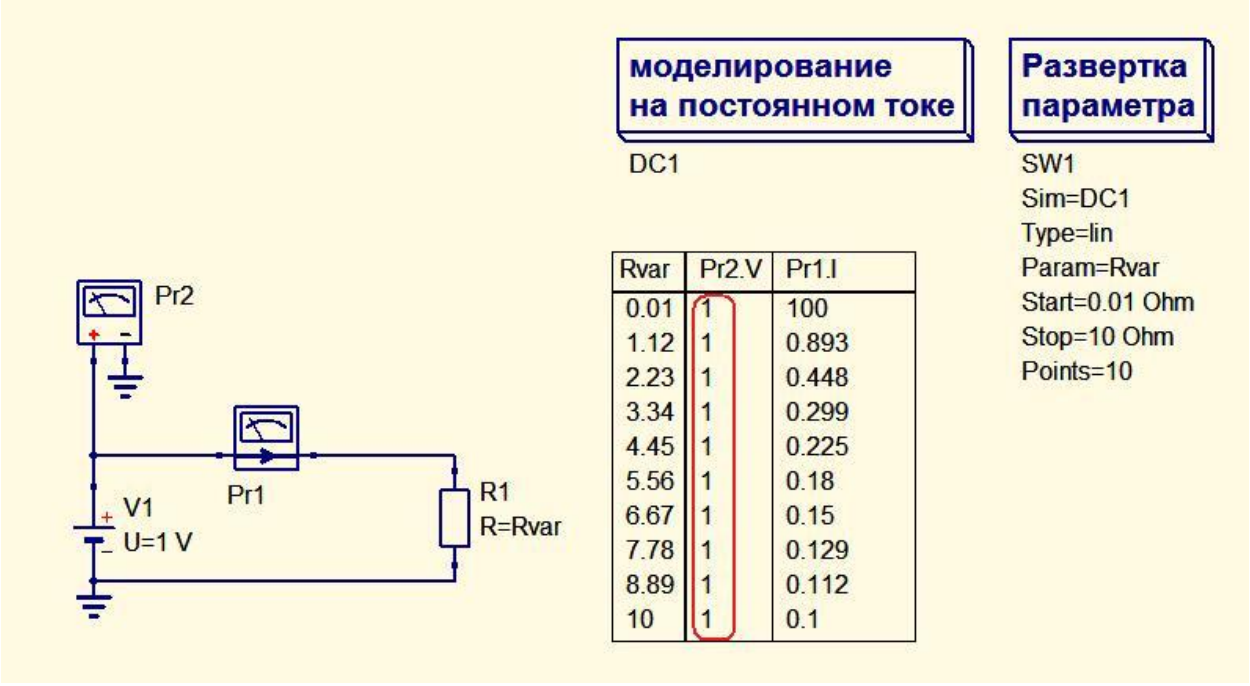
Авторы определяют пять базовых идеальных элементов: источники напряжения, источники тока, резисторы, индуктивности и конденсаторы.

Далее авторы пишут, что хотя может показаться, что этого слишком мало для начала анализа электрических цепей, но многие практические системы можно моделировать с помощью этих элементов. Кроме того, по мнению авторов, это прекрасная стартовая точка, благодаря относительной простоте: взаимосвязь между напряжениями, токами и сопротивлениями алгебраическая.

Идеальный источник напряжения поддерживает напряжение на своих выводах вне зависимости от тока, проходящего через эти выводы.

Соответственно, **идеальный источник тока** поддерживает ток на своих выводах вне зависимости от напряжения на них.

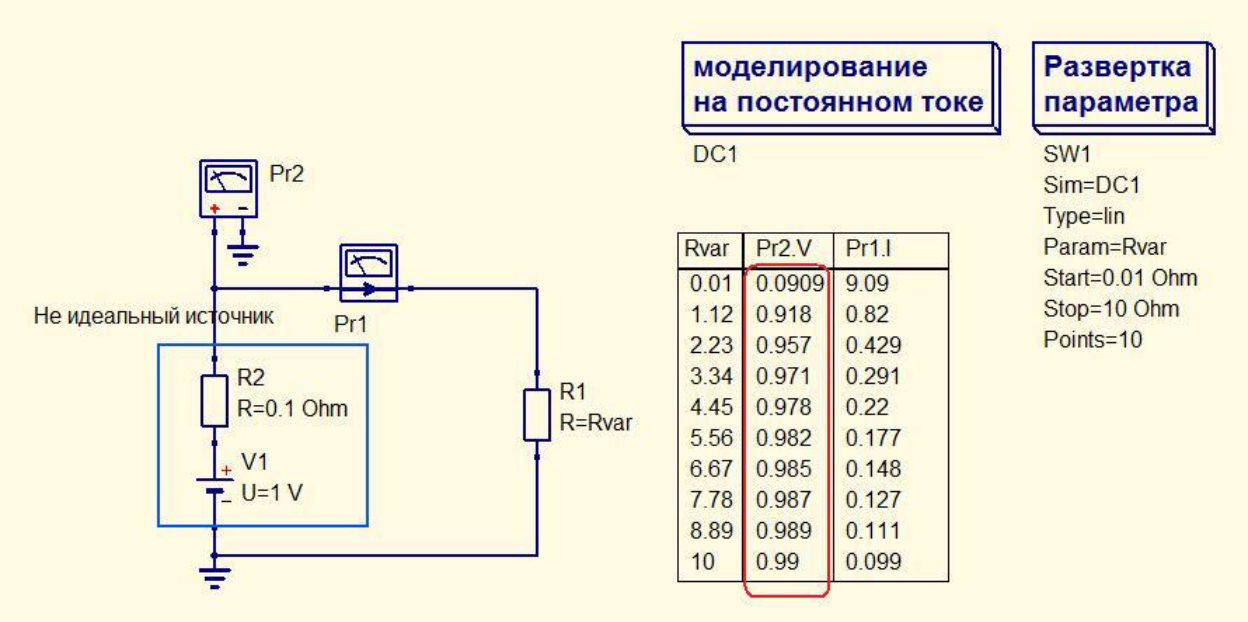
Программа Qucs позволяет очень наглядно увидеть, что имеют в виду авторы, когда говорят об идеальном источнике напряжения.



Rvar	Pr2.V	Pr1.I
0.01	1	100
1.12	1	0.893
2.23	1	0.448
3.34	1	0.299
4.45	1	0.225
5.56	1	0.18
6.67	1	0.15
7.78	1	0.129
8.89	1	0.112
10	1	0.1

Рис. 12.2. Идеальный источник напряжения

И сравнить его с не идеальным источником напряжения.



Rvar	Pr2.V	Pr1.I
0.01	0.0909	9.09
1.12	0.918	0.82
2.23	0.957	0.429
3.34	0.971	0.291
4.45	0.978	0.22
5.56	0.982	0.177
6.67	0.985	0.148
7.78	0.987	0.127
8.89	0.989	0.111
10	0.99	0.099

Рис. 12.3. Не идеальный источник напряжения

Авторы рассматривают **зависимые и независимые источники** напряжения и тока, зависимые источники иногда называют **управляемыми источниками**.

Такие источники в программе можно найти на закладке **Компоненты** в разделе **источники**.

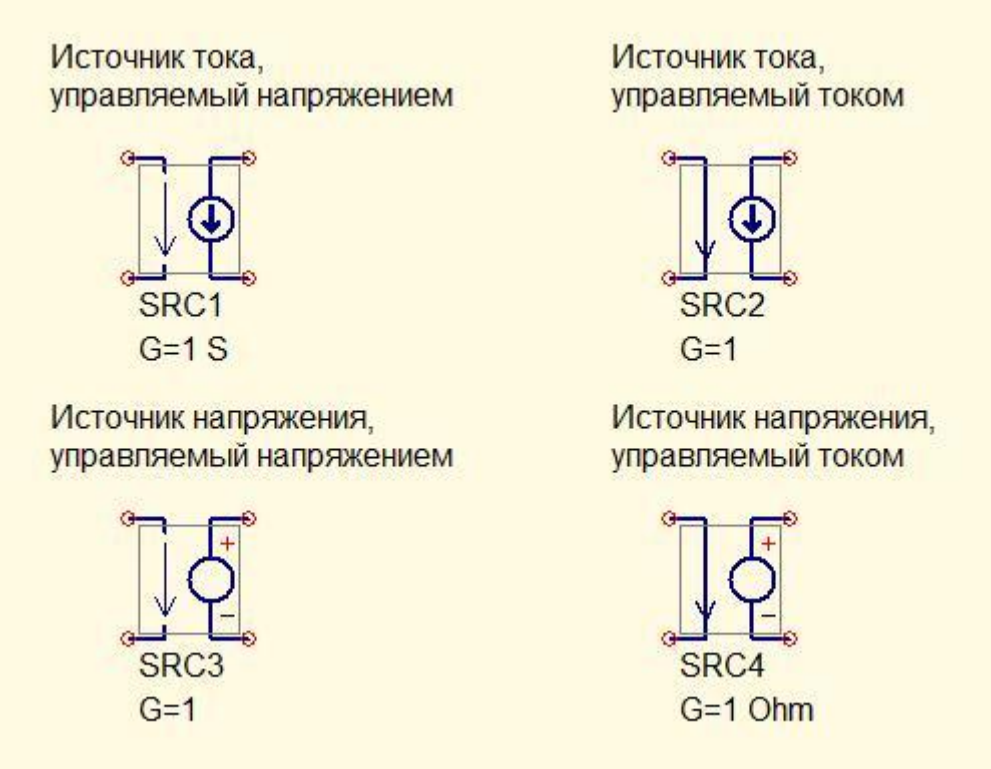


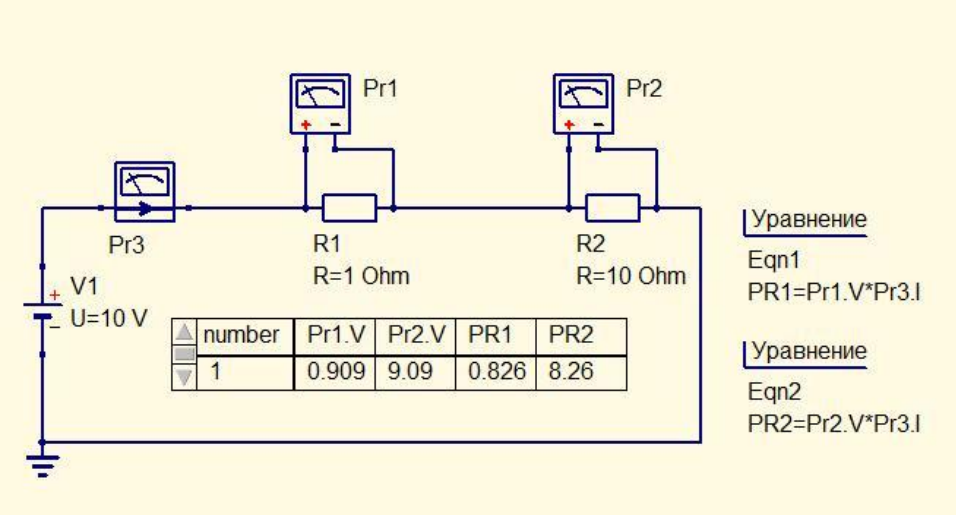
Рис. 12.4. Зависимые источники напряжения и тока

Резистор авторы справедливо определяют, как способность материала препятствовать протеканию электрического тока или, точнее, перемещению электрических зарядов.

Законы Ома и Кирхгоффа

Конечно, **Закон Ома**:

v = iR, где напряжение v в вольтах, ток i в амперах, сопротивление R в омах.



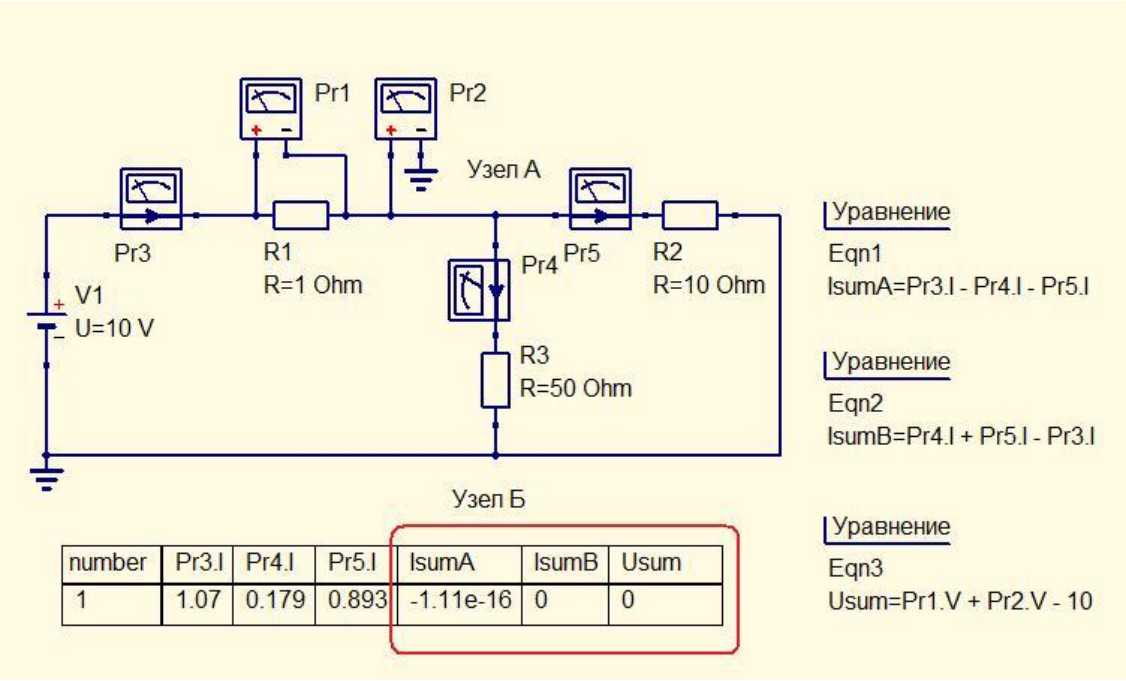
number	Pr1.V	Pr2.V	PR1	PR2
1	0.909	9.09	0.826	8.26

Рис. 12.5. Закон Ома

И, конечно, Законы Кирхгоффа:

Алгебраическая сумма всех токов в любом узле электрической цепи равна нулю.

Алгебраическая сумма всех напряжений вдоль замкнутой цепи равна нулю.

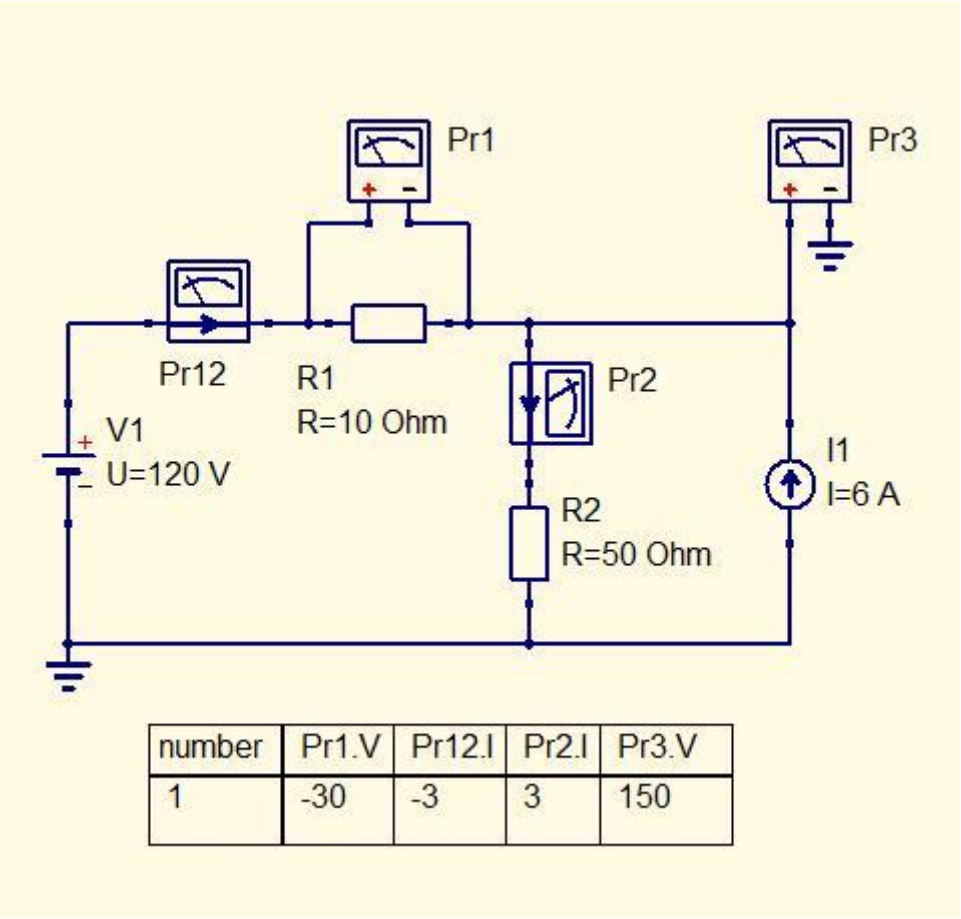


number	Pr3.I	Pr4.I	Pr5.I	IsumA	IsumB	Usum
1	1.07	0.179	0.893	-1.11e-16	0	0

Рис. 12.6. Законы Кирхгоффа

Надеюсь, вас не смущает, что ток в узле А не точно равен нулю – это обычная ошибка счета, ведь программа считает, а не принимает все на веру.

Очень интересный пример приведен дальше для исследования незнакомой электрической цепи с использованием законов Ома и Кирхгоффа.



number	Pr1.V	Pr12.I	Pr2.I	Pr3.V
1	-30	-3	3	150

Рис. 12.7. Исследование электрической цепи

Практика преподавания авторов книги позволяет им привести большое количество примеров и задач. Используя программу, можно проверить правильность решений, даже не обращаясь к ответам в конце книги. Но повторение задач дает еще одно преимущество – навык создания электрических схем в редакторе программы. Позже, если вы займетесь предметом профессионально, эти навыки вам пригодятся.

Умение использовать, казалось бы, очень простые законы Ома и Кирхгоффа, когда вы познакомитесь на практике с реальными устройствами, не только поможет вам в работе, но, скорее всего, будет одним из основных теоретических инструментов. Повторяя раз за разом «простые» схемы, комбинируя «простые» идеальные элементы схем, вы научитесь лучше понимать реальные компоненты любого электрического устройства.

Последовательное и параллельное соединение резисторов

Для понимания результирующего эффекта от соединения резисторов достаточно использовать законы Ома и Кирхгоффа.

При **последовательном** соединении сопротивлений

Rпосл = R1 + R2 +… + Rn, общее сопротивление – сумма всех резисторов

Величина, обратная к сопротивлению, называется проводимостью *G = 1/R* и измеряется в сименсах.

При **параллельном** соединении сопротивлений

Gпарал = G1 + G2 +…+ Gn, сумма всех проводимостей даст результирующую проводимость

1/R(парал) = 1/R1 + 1/R2 +…+ 1/Rn

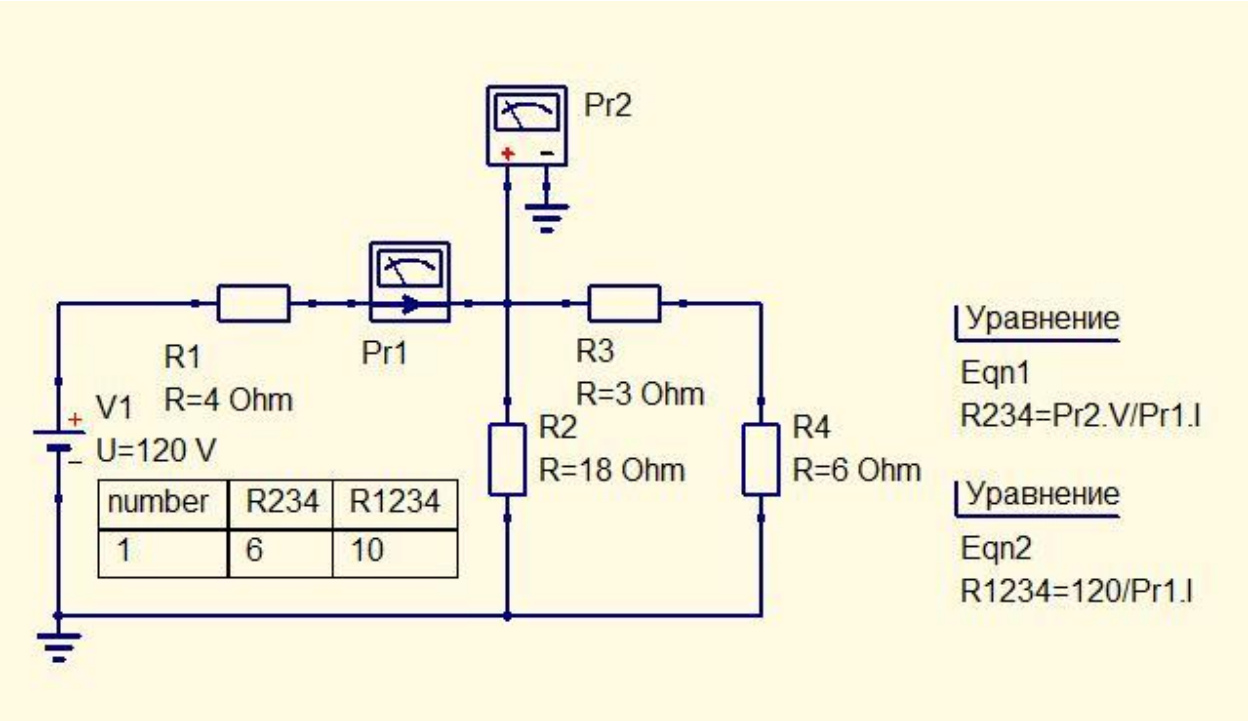


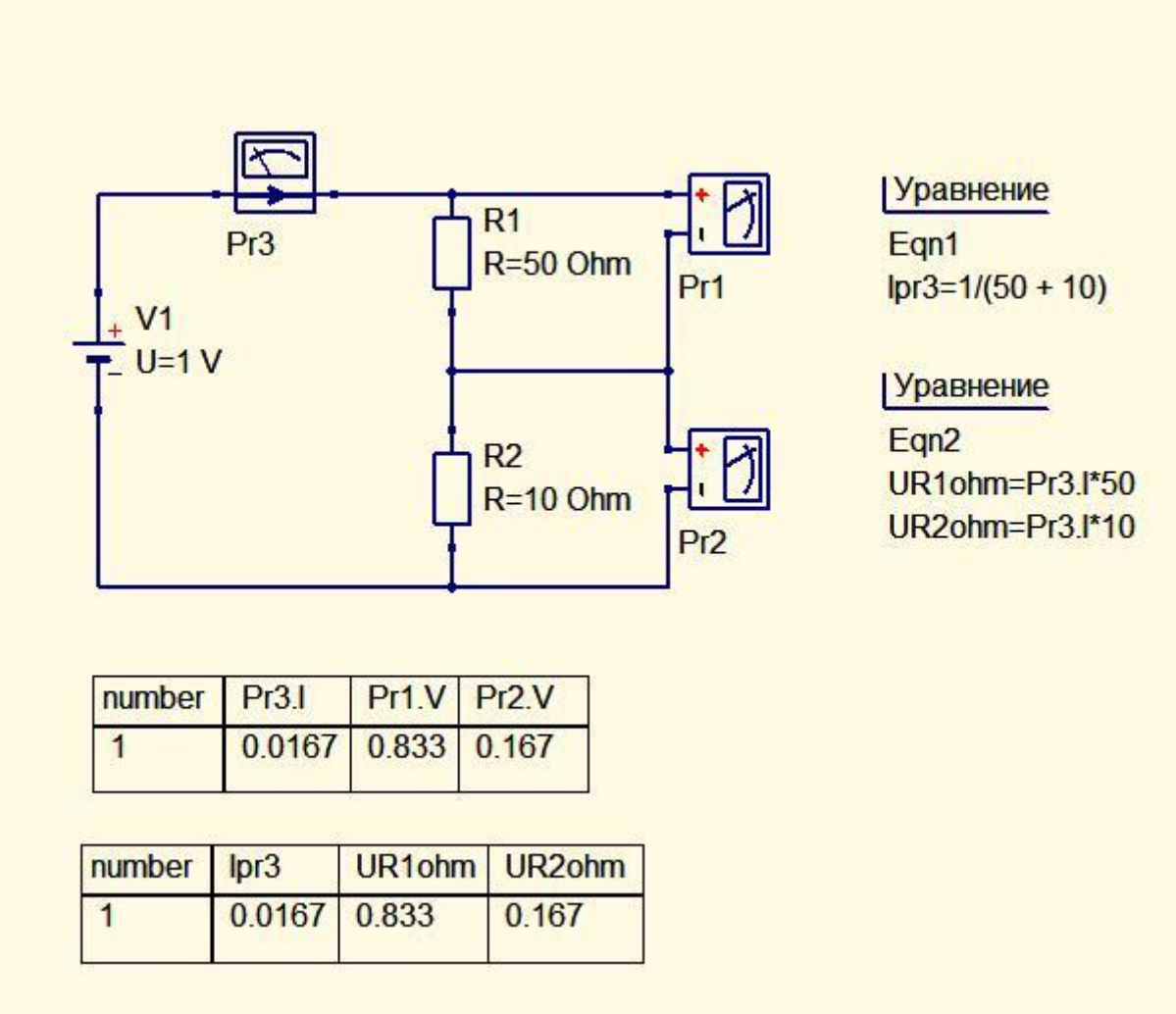
Рис. 12.8. Соединение резисторов

Интересно отметить, забегая вперед, что при измерении амперметром Pr1 и вольтметром Pr2 значений тока и напряжения, если бы источник V1 был источником переменного синусоидального напряжения, а приборы предназначены для измерения переменного напряжения, мы получили бы такие же результаты, как свидетельство того, что сопротивление свойство материала и в линейной области не зависит от характера тока.

Резистивные делители

Делитель напряжения – конструкция, довольно часто встречаемая в схемах. Необходимость в делителе возникает тогда, когда нужно использовать только часть напряжения источника. Регулировку громкости в старых усилителях всегда выполнял переменный резистор (или потенциометр), делитель с механически изменяемым коэффициентом деления напряжения.

Чтобы понять, как работает делитель напряжения, достаточно знания законов Ома и Кирхгоффа.



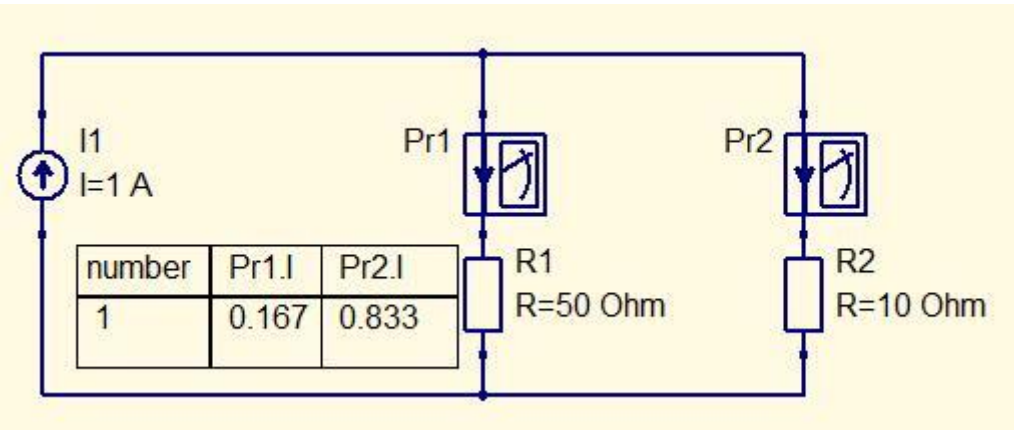
number	Pr3.I	Pr1.V	Pr2.V
1	0.0167	0.833	0.167

number	Ipr3	UR1ohm	UR2ohm
1	0.0167	0.833	0.167

Рис. 12.9. Резистивный делитель напряжения

Приборы Pr1 и Pr2 – вольтметры, Pr3 – амперметр.

Необходимость в резистивном делителе тока возникает тогда, когда нужно «ответвить» часть тока. Примерно по этому принципу устроены шунты в приборах.



number	Pr1.I	Pr2.I
1	0.167	0.833

Рис. 12.10. Резистивный делитель тока

Два делителя напряжения могут делить напряжение источника в одинаковой пропорции и тогда, когда их суммарные сопротивления различны. На этом принципе работает метод измерения сопротивления, называемый мостовым, где используются и делитель напряжения, и делитель тока.

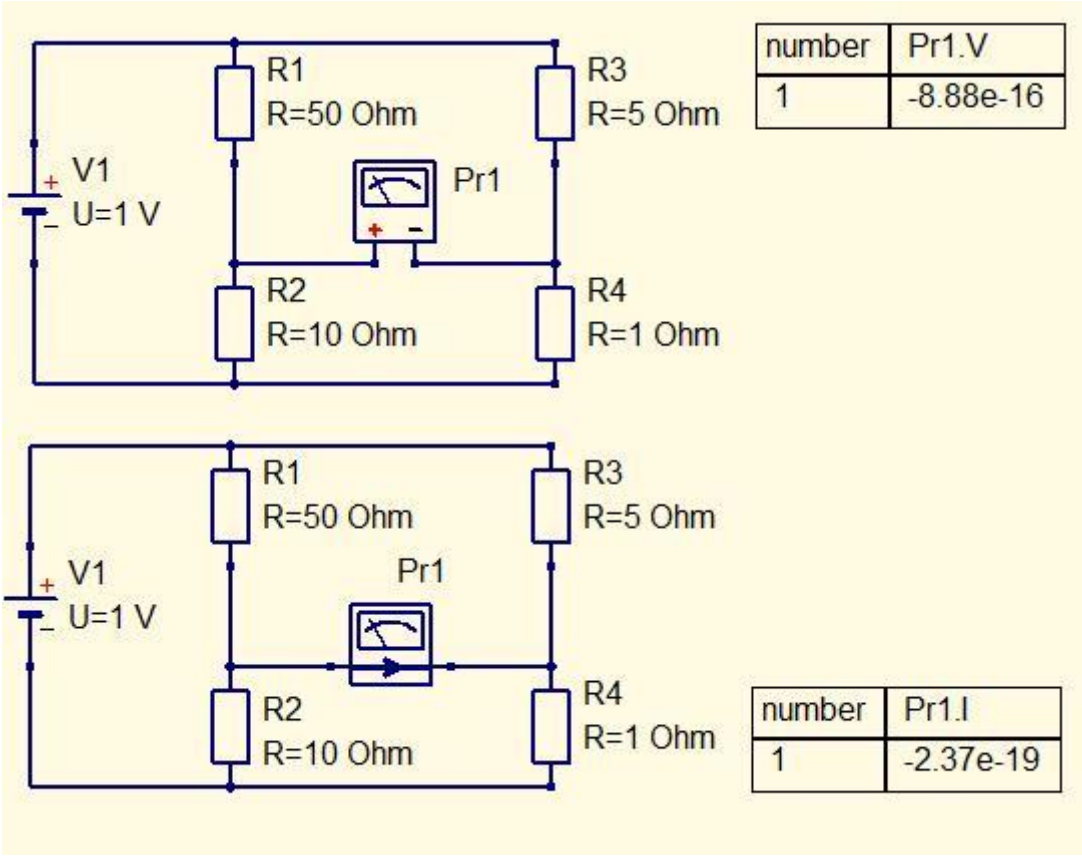


Рис. 12.11. Мостовой метод измерения сопротивления

На рисунке в первом случае Pr1 – вольтметр, во втором амперметр.

Если значения, например, резисторов R1-R3 известны, можно определить значение резистора R4 по минимуму тока через амперметр.

Авторы отмечают, что если измерительный амперметр в схеме выше заменить резистором, эквивалентным сопротивлению прибора, то полученную схему будет трудно рассчитать при произвольных значениях резисторов, используя законы Ома и Кирхгоффа. Эта трудность связана с внутренним соединением элементов.

Решением этих затруднений является метод преобразования «треугольника в звезду» и наоборот. Соединение резисторов на схеме рисунка 12.11, если присмотреться, очень напоминает два треугольника из резисторов. Если один из них преобразовать в эквивалентную схему соединения «звездой», то схема легко поддается расчетам. Под эквивалентностью понимаются такие значения резисторов, при которых схема, помещенная в черный ящик, не позволяет определить, как включены резисторы, если измерять сопротивления, используя три точки вне черного ящика. Метод преобразования соединения треугольником в эквивалентное соединение звездой один из важных методов в теории электрических цепей.

Кроме законов Ома и Кирхгоффа, применяемых к относительно простым электрическим резисторным цепям, существуют более продуктивные методы расчетов сложных цепей.

Метод контурных токов и узловых потенциалов

Признаться, что начиная этот раздел рассказа, я так увлекся чтением учебника, что с трудом нашел место, где остановился в своем рассказе. Авторы учебника так тщательно определяют все понятия, которыми пользуются в дальнейшем, что не успеваешь оглянуться, как ты переместился страниц на двадцать вперед, не сделав ни одной «зарисовки» на компьютере.

Однако вернемся к методам расчета сложных электрических цепей, которые, скорее всего, лежат в основе если не всех, то многих, компьютерных программ, симулирующих работу электрических цепей.

Несколько определений.

Узел – точка, где соединяются два (или более) элемента электрической цепи.
Основной узел – точка, где соединяются три (или более) элемента электрической цепи.
Путь – путь прохождения основных элементов без включения более одного раза.
Ветвь – путь, соединяющий два узла.
Основная ветвь – путь, соединяющий два основных узла, но без прохождения через основной узел.
Контур – путь, где конечный узел совпадает с начальным.
Петля – контур, не содержащий других контуров.
Планарная цепь – цепь, которая может быть нарисована на плоскости без пересекающихся ветвей.

И иллюстрация.

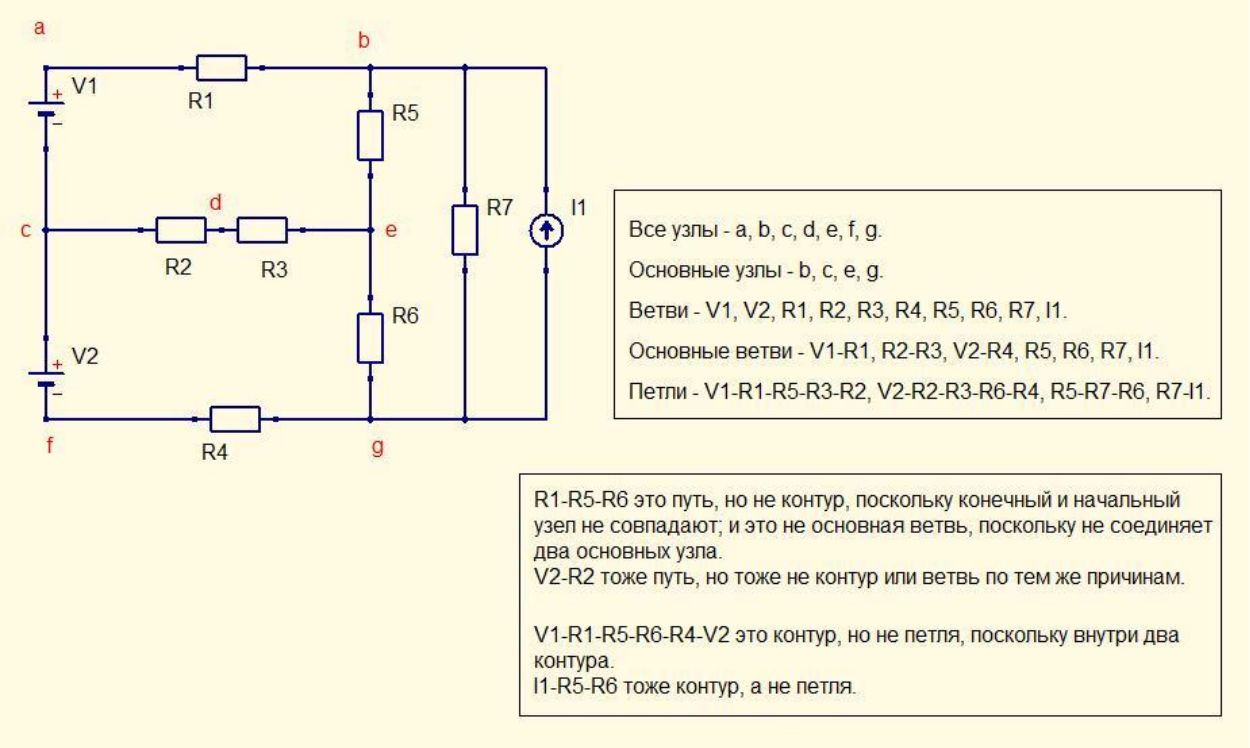


Рис. 12.12. Иллюстрация определений

Рассматривая вопрос о том, сколько нужно совместных уравнений для неизвестных токов, авторы используют схему, изображенную выше. Их вывод – столько уравнений, сколько ветвей с неизвестными токами. Для схемы выше это девять уравнений (ток в ветви I1 задан!) для девяти ветвей. Уравнения можно составить, используя оба закона Кирхгоффа. Но лучшие результаты даст метод узловых потенциалов. Рассмотрим схему.

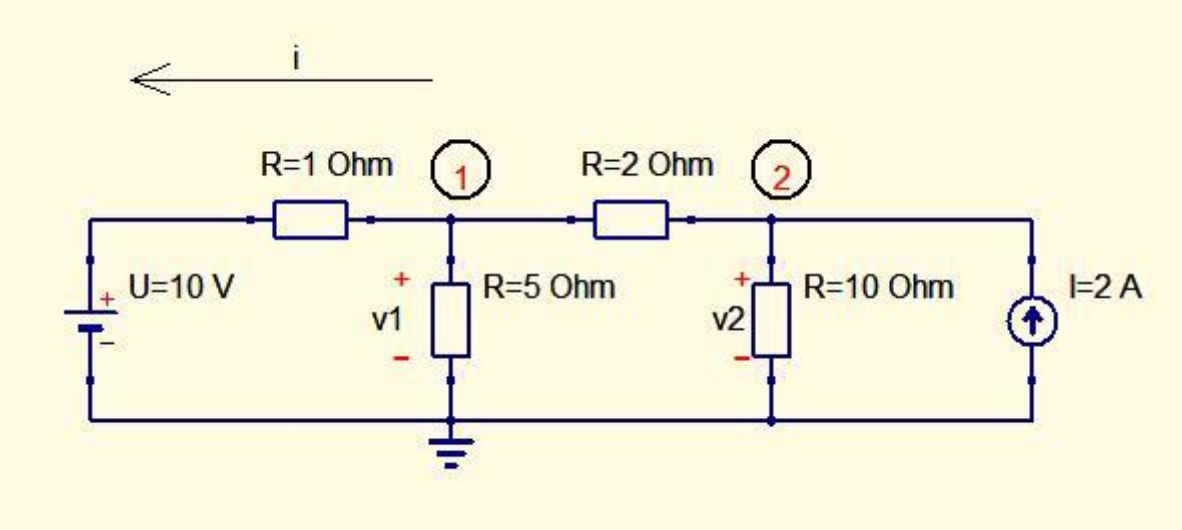


Рис. 12.13. Схема для поиска решения методом узловых потенциалов

Количество уравнений должно быть на единицу меньше, чем количество основных узлов (их три). Один из основных узлов выбирается в качестве опорного, что на схеме изображается с помощью символа «земля». Затем определяются узловые напряжения, как напряжения от опорного узла к искомому (v1 и v2 на рисунке выше). Для составления уравнений вначале записываются токи, протекающие по каждой ветви, присоединенной к не опорному узлу, которые затем суммируются и, согласно закону Кирхгоффа приравниваются нулю. Так для резистора в один Ом ток ***i*** равен разности напряжений ***v1 -10***, деленной на величину этого сопротивления (то есть, (v1-10)/1). Аналогично записываются токи для двух оставшихся ветвей. Результат:

(v1-10)/1 + v1/5 + (v1-v2)/2 = 0

И уравнение для узла 2:

(v2-v1)/2 + v2/10 – 2 = 0

Получив систему из двух уравнений для двух неизвестных, ее можно решить:

v1 = 9.09 V; v2 = 10.91 V

Конечно, все примеры, приведенные в учебнике, необходимо решить с карандашом и бумагой. Для того они и приведены. Но проверить решение можно за компьютером.

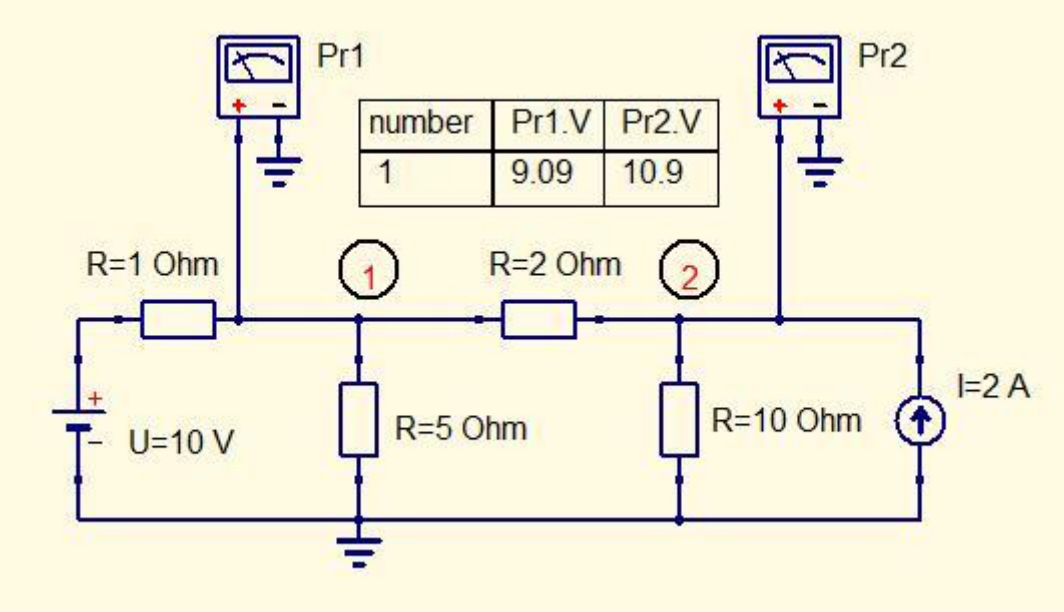


Рис. 12.14. Проверка решения

Зная напряжения v1 и v2, легко определить все токи в ветвях.

Сущность второго метода, метода контурных токов (в оригинале метод петлевых токов), в выделении на схеме петель (контур, не содержащий внутри другого контура), по которым протекают токи, как показано на рисунке ниже (метод применяется только к планарным цепям).

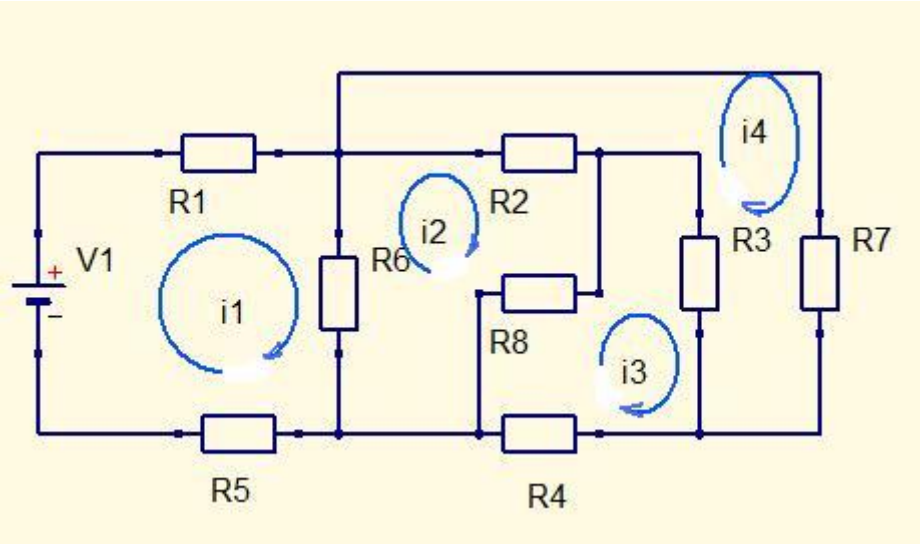


Рис. 2.15. Иллюстрация метода решения с помощью контурных токов

Контурным токам не всегда соответствуют токи в ветвях. Например, току i2 не соответствует ни один из токов в ветвях, тогда как токам i1, i3 и i4 такое соответствие находится.

Рассмотрим использование метода на другом примере, где легко увидеть, как связаны между собой токи в ветвях и петлях.

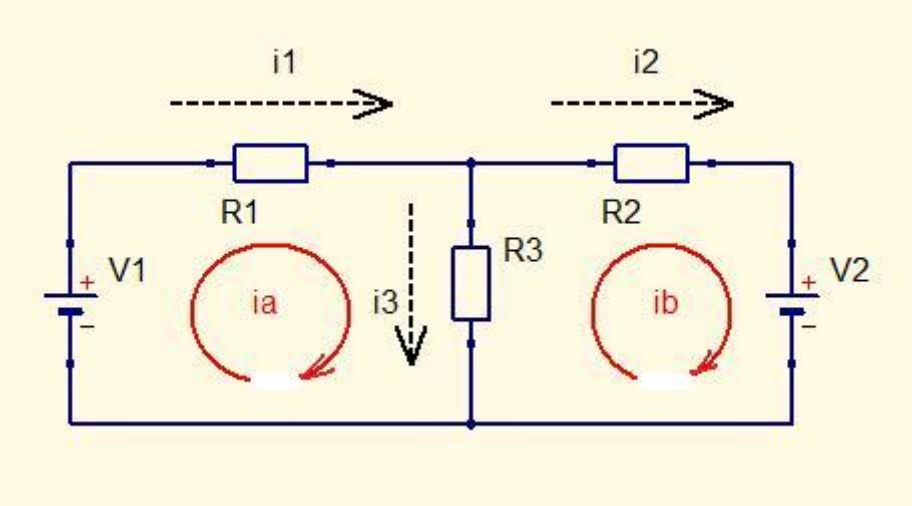


Рис. 12.16. Поиск решения методом контурных токов

Следуя закону Кирхгоффа для токов, можно записать:

i1 = i2 + i3

С другой стороны, закон Кирхгоффа для напряжений дает еще два уравнения:

v1 = i1R1 + i3R3

-v2 = i2R2 – i3R3

Выразив ток i3 в первом уравнении через токи i1 и i2, мы можем подставить его в два уравнения для напряжений, получив два связанных уравнения с двумя неизвестными. Решение этой системы даст величину токов i1 и i2, а найти ток i3 можно из первого уравнения.

Но мы можем воспользоваться контурными (петлевыми) токами ia и ib для получения системы из двух уравнений с двумя (относительно контурных токов) неизвестными:

v1 = iaR1 + (ia – ib)R3

-v2 = (ib – ia)R3 + ibR2

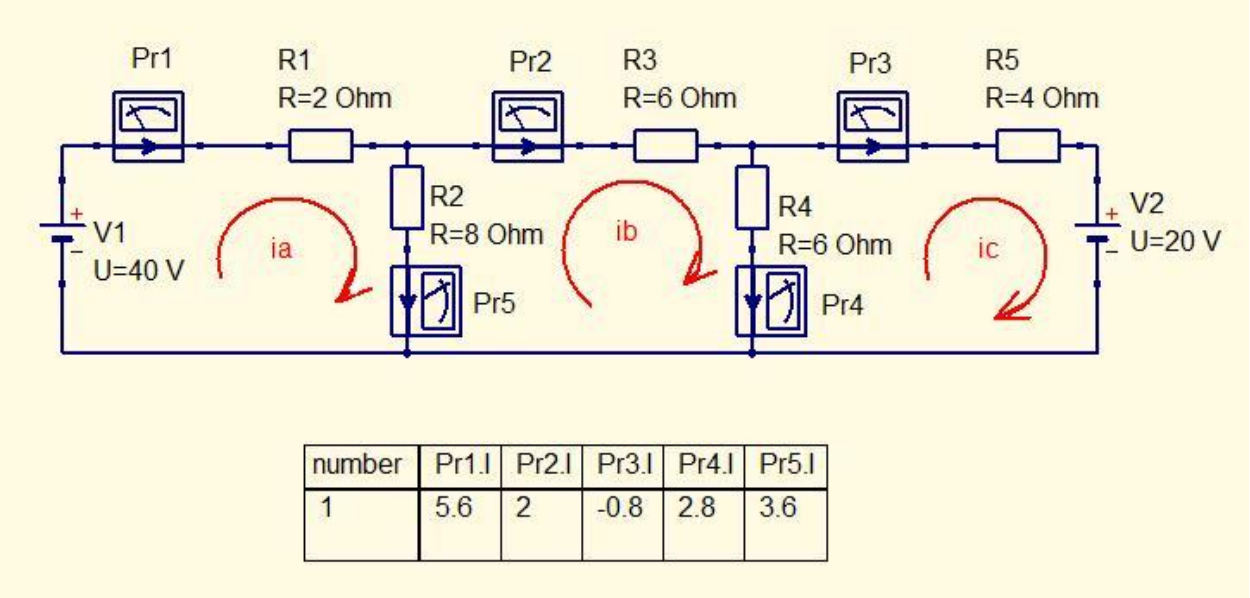
Решение этих уравнений даст два тока ia и ib, через которые легко находятся искомые токи:

i1 = ia

i2 = ib

i3 = ia – ib

Вот конкретный пример:



number	Pr1.I	Pr2.I	Pr3.I	Pr4.I	Pr5.I
1	5.6	2	-0.8	2.8	3.6

Рис.12 17. Конкретный пример с проверкой в Qucs

Решение уравнений для токов ia, ib, ic дает значения: 5.6 A, 2.0 A, -0.8 A.

Преобразование источников

Методы контурных токов и узловых потенциалов могут быть дополнены еще одним полезным приемом. И, не смотря на то, что цепь, показанная на рисунке ниже, очень проста, прием может быть полезен во многих случаях при анализе схем.

Речь идет о преобразовании источника напряжения в источник тока и наоборот. Пусть оба источника имеют свойства приближенные к реальным, то есть, имеют внутреннее сопротивление.

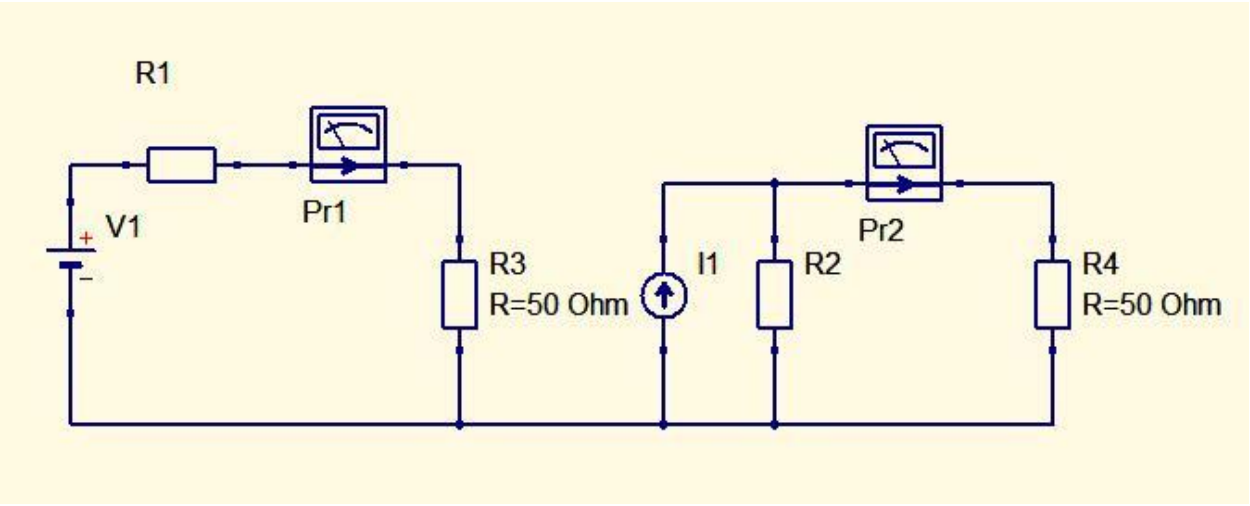


Рис. 12.18. К преобразованию источников

Под эквивалентностью источника напряжения V1 с внутренним сопротивлением R1 и источника тока I1 с внутренним сопротивлением R2 будем понимать такие значения их параметров, при которых ток через нагрузку (резисторы R3 и R4) будет одинаков, если величина сопротивления одинакова. То есть, амперметры Pr1 и Pr2 должны показать одинаковый ток.

Для первой схемы с источником напряжения можно записать выражение для тока через нагрузку R3:

I = V1/(R1 + R3)

Для второй схемы с источником тока можно приравнять напряжение на R2 (равное R(I1 – I), закон Кирхгоффа для токов) и на R4 (равное IR4), откуда получается уравнение для тока через нагрузку R4:

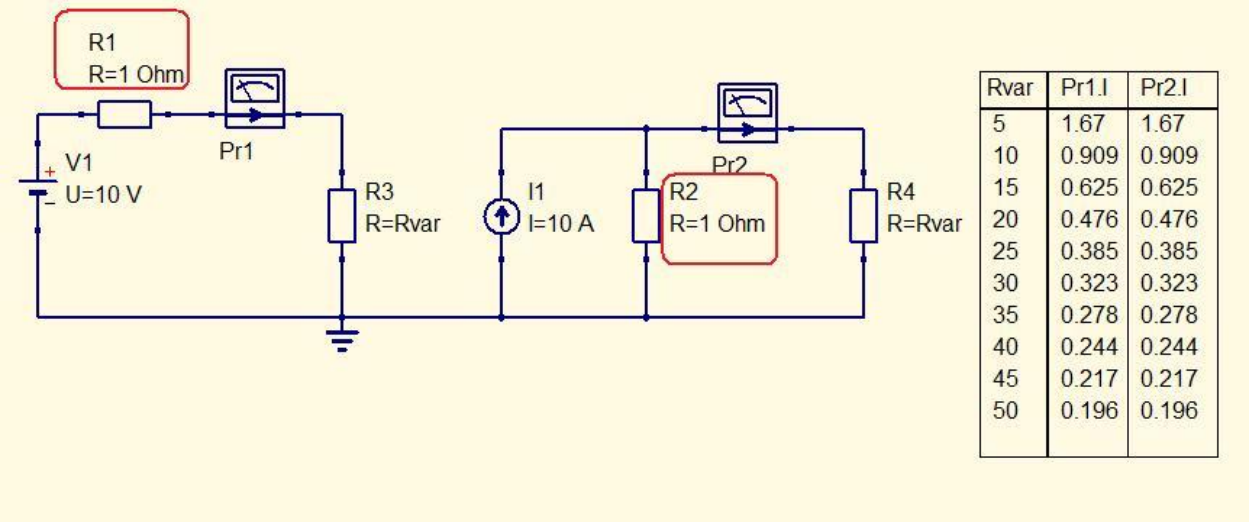
I = I1R2/(R2 + R4)

Приравнивая правые части этих уравнений, и накладывая дополнительное условие: R1 = R2 = R, можно получить связь между током источника тока и напряжением источника напряжения.

I1 = V1/R

При этом, как видно из выражения, ток в нагрузке не зависит от величины нагрузочного сопротивления.

Проверим это в программе Qucs, используя развертку параметра R (сопротивления нагрузки):



Rvar	Pr1.I	Pr2.I
5	1.67	1.67
10	0.909	0.909
15	0.625	0.625
20	0.476	0.476
25	0.385	0.385
30	0.323	0.323
35	0.278	0.278
40	0.244	0.244
45	0.217	0.217
50	0.196	0.196

Рис. 12.19. Проверка независимости эквивалентных источников от нагрузки

При анализе схемы бывает удобно использовать этот «принцип эквивалентности».

Операционный усилитель

Если рассматривать операционный усилитель, как целый блок, не вдаваясь в детали внутреннего устройства, то интерес представляют его выводы и его поведение.

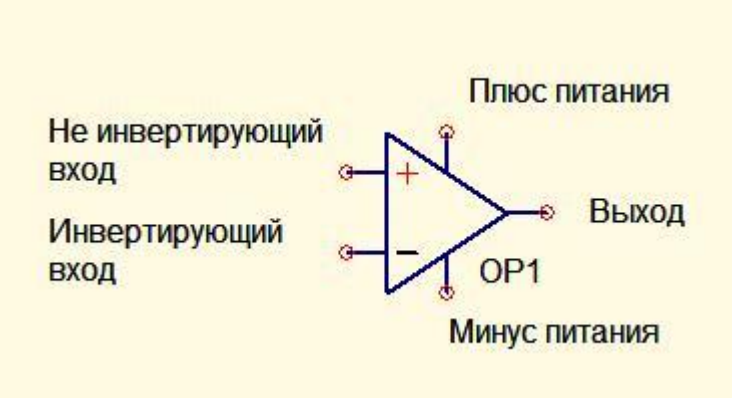


Рис. 12.20. Выводы операционного усилителя

При напряжении питания Vcc и усилении по напряжению A можно выделить следующие области работы операционного усилителя:

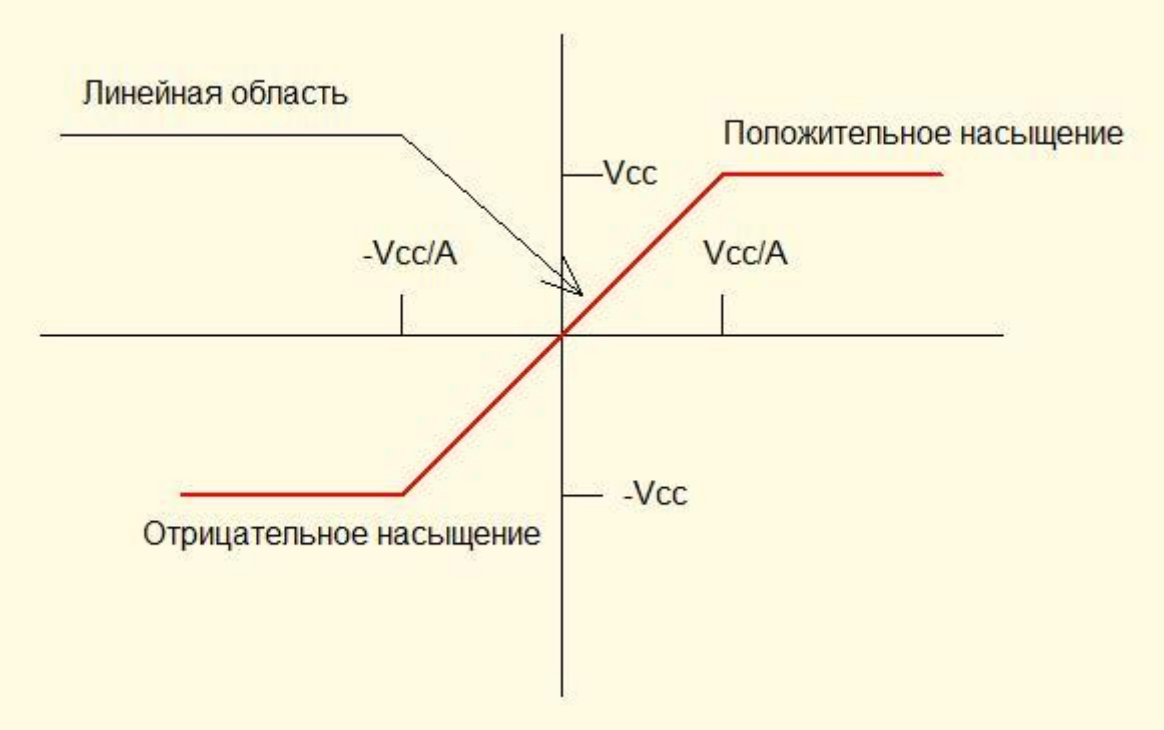


Рис. 12.21. Передаточная характеристика напряжения операционного усилителя

И, чтобы не пересказывать всю книгу, несколько Qucs-иллюстраций примеров, приведенных в книге.

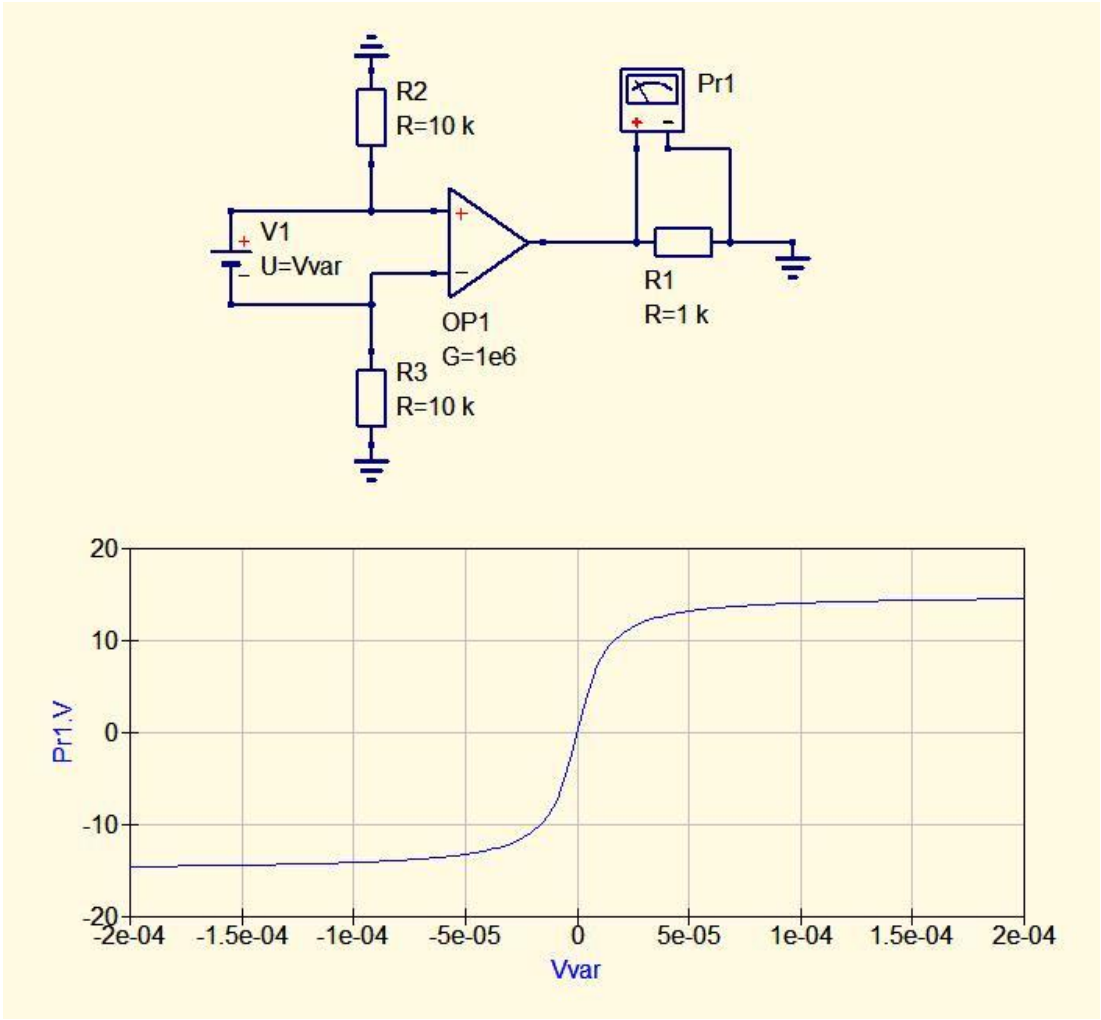


Рис. 12.22. Зависимость между изменением разностного входного напряжения и выходным

Влияние отрицательной обратной связи на усиление операционного усилителя.

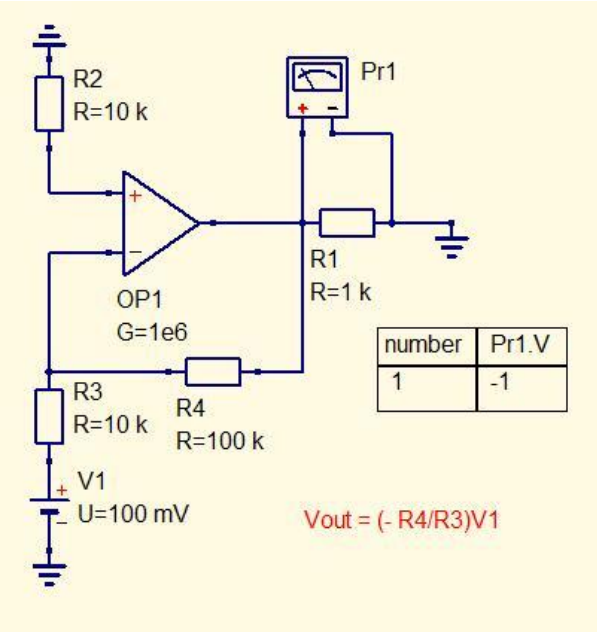


Рис. 12.23. Инвертирующий операционный усилитель с отрицательной обратной связью

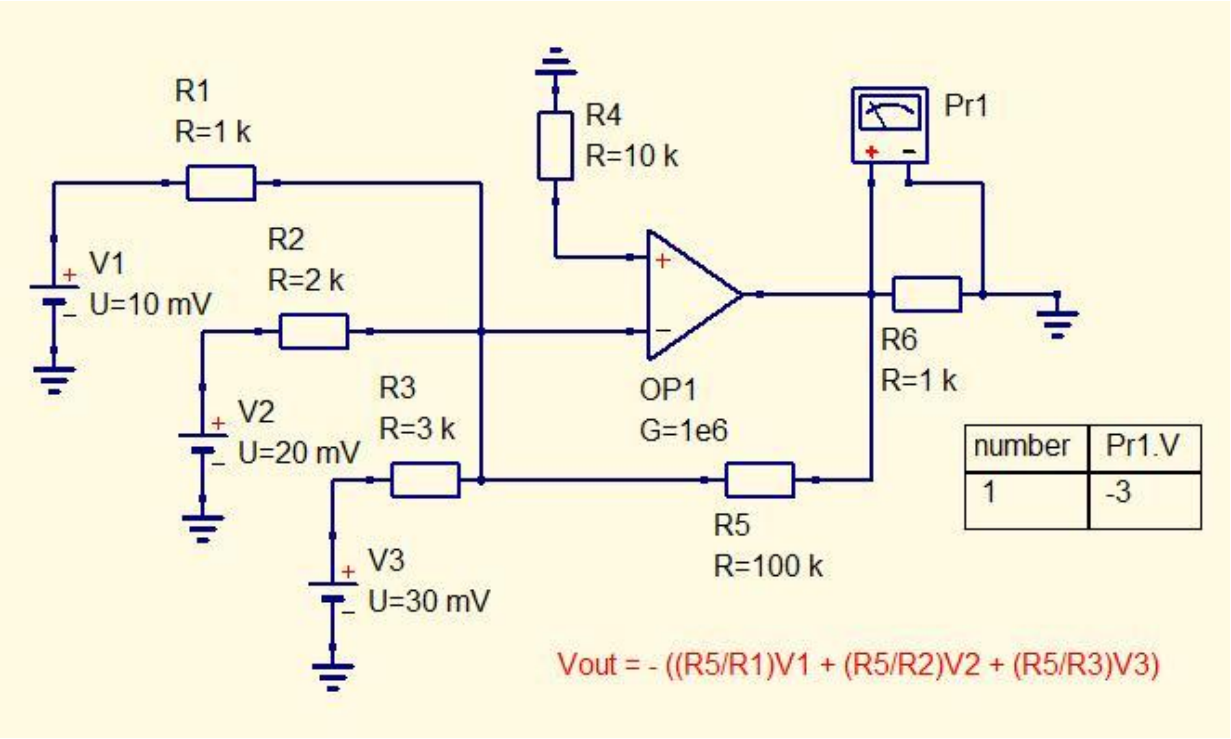


Рис. 12.24. Инвертирующий суммирующий операционный усилитель

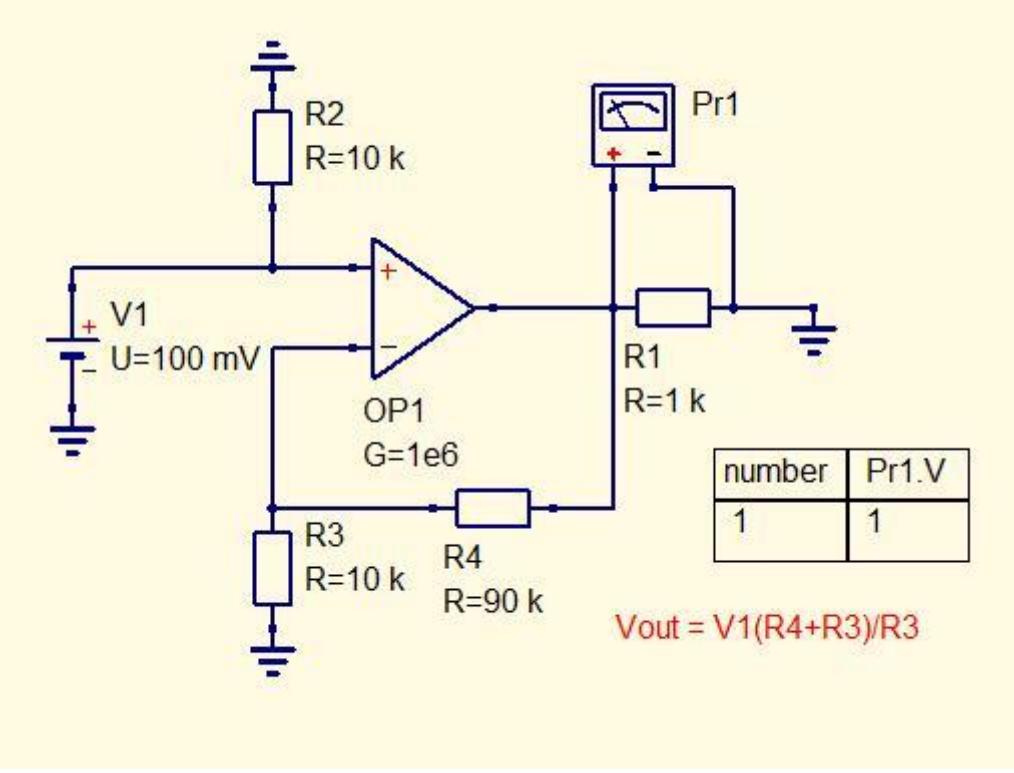


Рис. 12.25. Не инвертирующий операционный усилитель

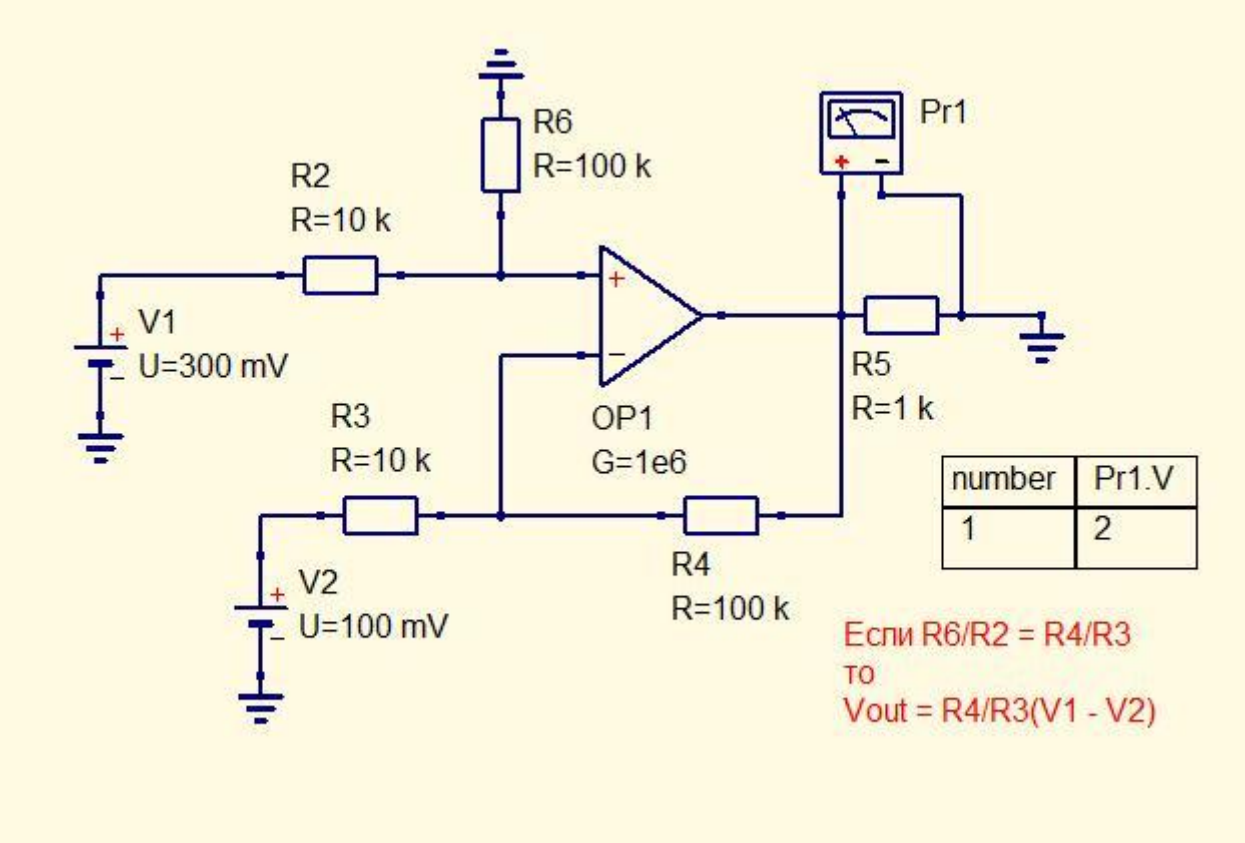


Рис. 12.26. Дифференцирующий усилитель

Индуктивность, емкость

Не как сопротивление, индуктивность и емкость больше интересны реакцией на изменение тока, чем в цепях постоянного тока. Поэтому, когда говорят об их сопротивлении, то называют его реактивным. Для напряжения на индуктивности справедливо выражение:

v = L (di/dt)

где L – индуктивность в генри, v – напряжение в вольтах, ток в амперах и время в секундах, а di/dt характеризует скорость изменения тока в цепи.

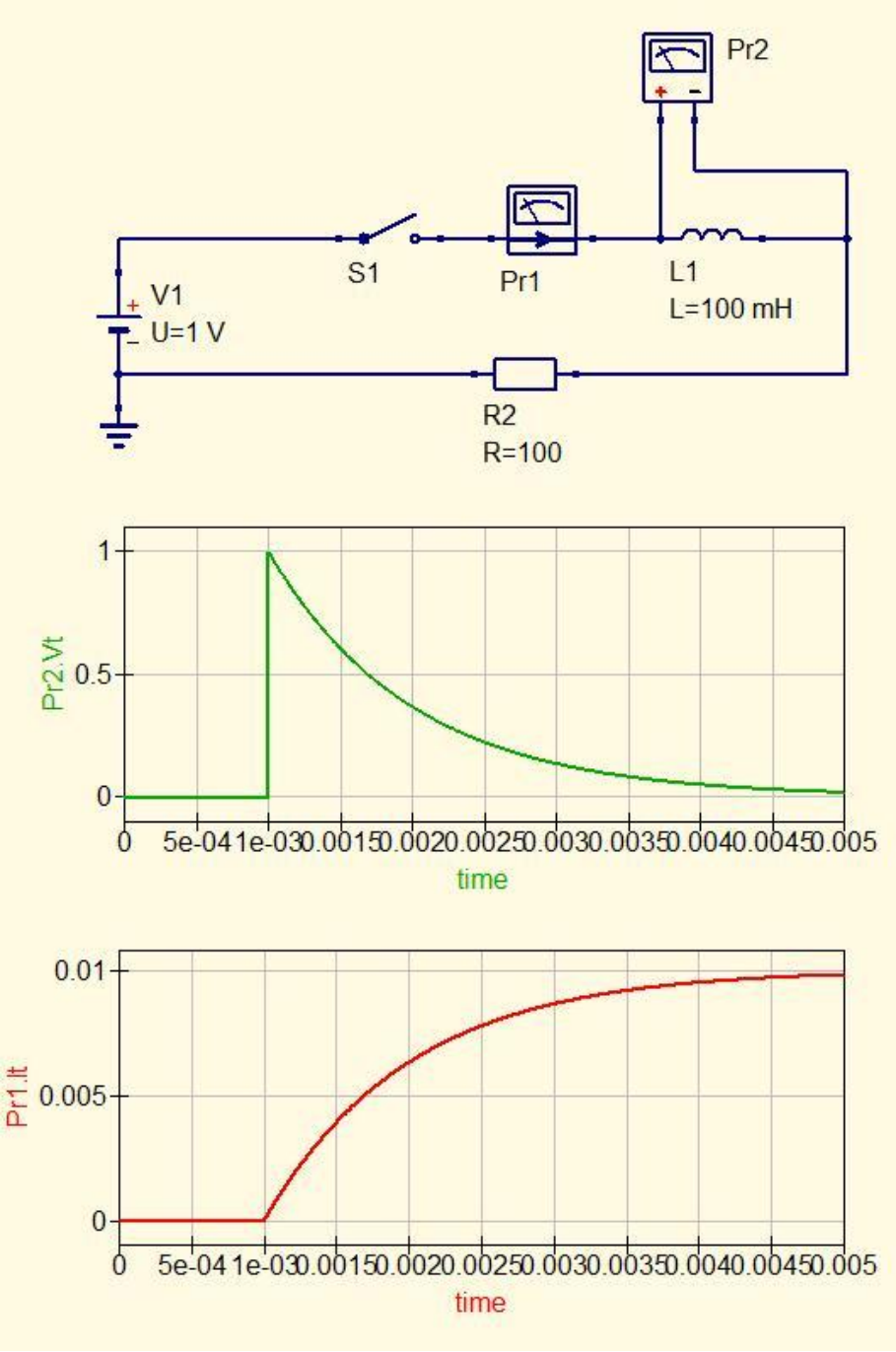


Рис. 12.27. Реакция индуктивности на изменение напряжения

Чтобы увидеть эту реакцию, в схему добавлен выключатель S1, который включается через 1 мс, а резистор R2 превращает идеальные источник напряжения и индуктивность в реальные –

катушка индуктивности мотается проводом, имеющим сопротивление, на сердечнике или без него, но имеет активное сопротивление не равное нулю.

Выражение для тока через конденсатор можно записать в таком виде:

i = C(dv/dt), где ток в амперах, емкость в фарадах, а напряжение в вольтах.

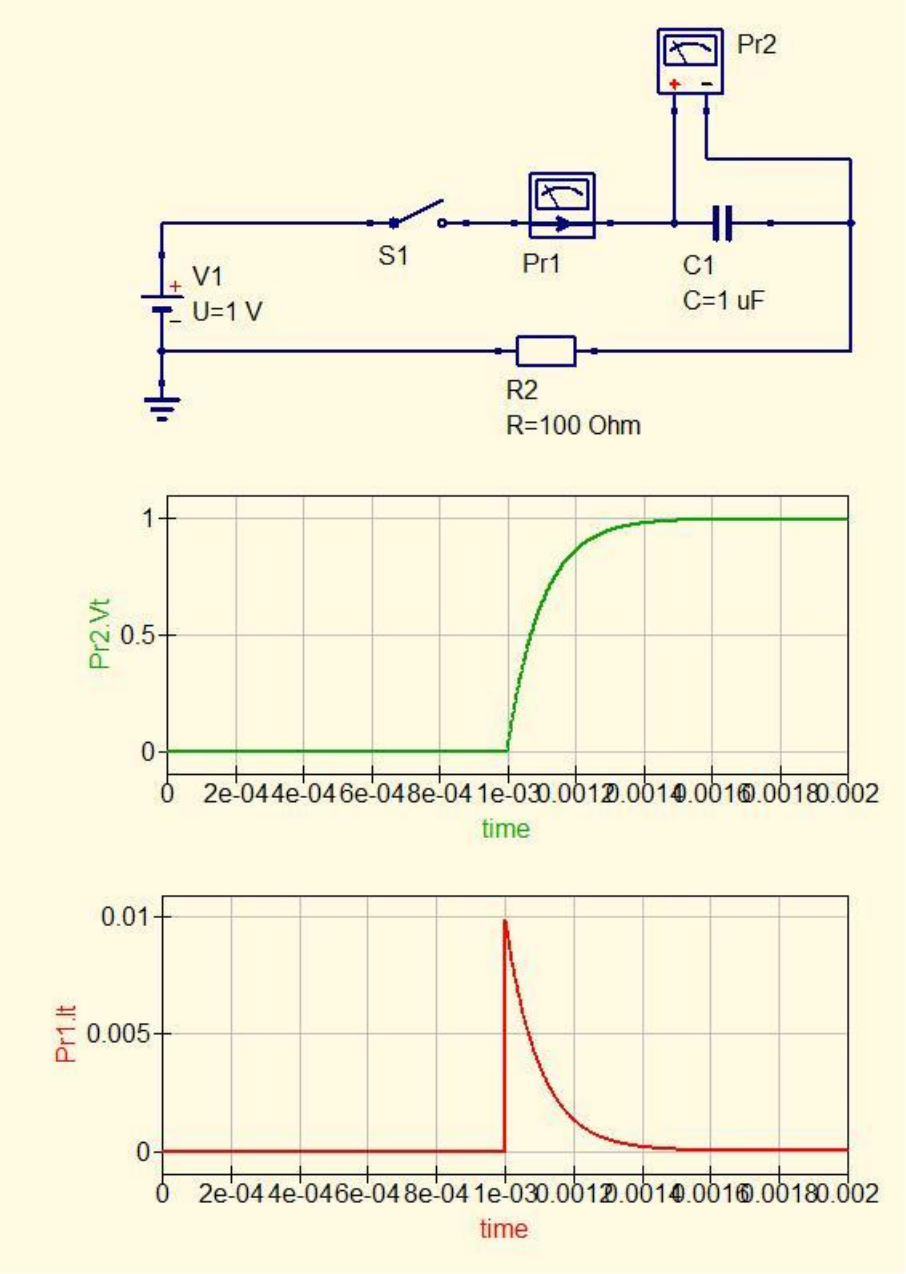
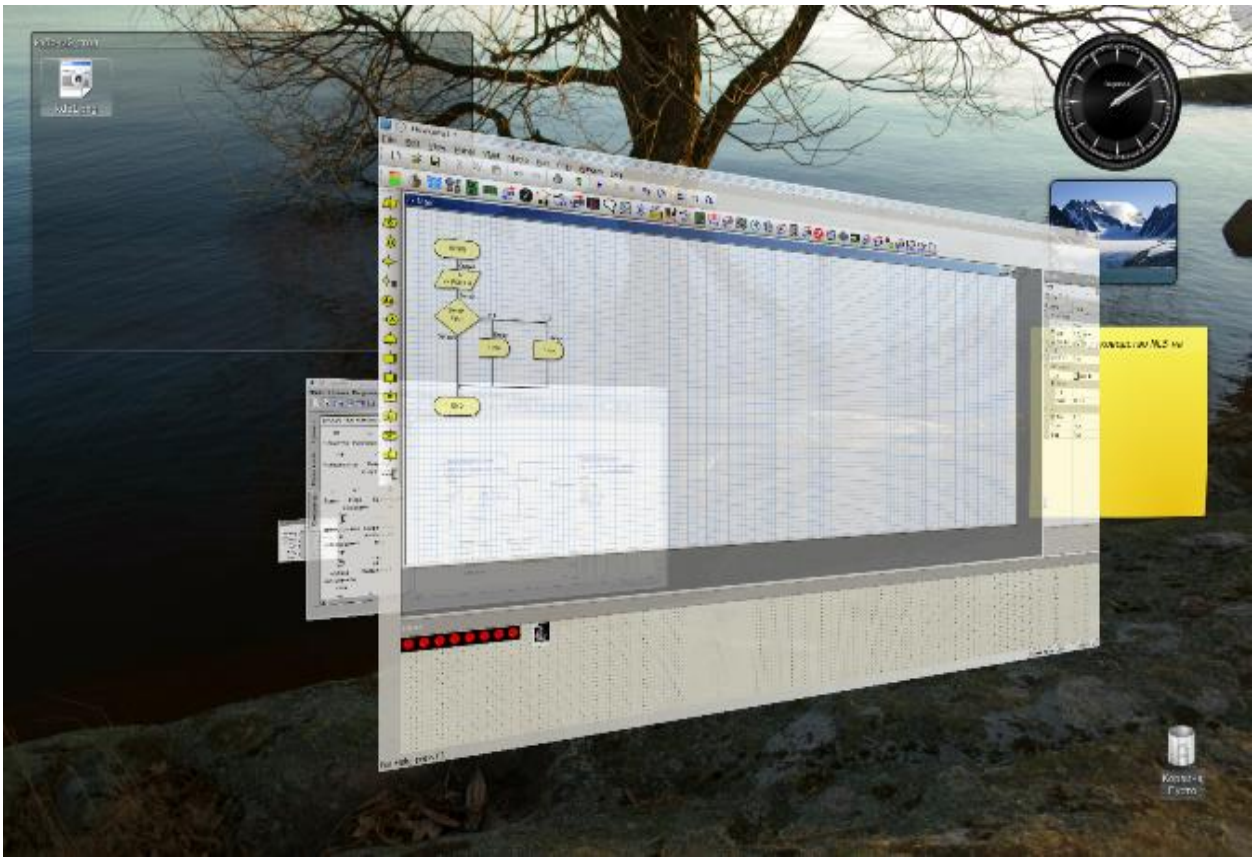


Рис. 12.28. Реакция конденсатора на изменение напряжения

Заключение

В книге Джеймса Нильссона и Сьюзен Райэдел еще много интересного. Но пересказывать ее всю нет необходимости. И совсем, конечно, не обязательно читать именно эту книгу, можно читать другой учебник, важно только, что читая учебник, можно с помощью программы Qucs проверить не только ответы, но яснее понять происходящее. Обычно эта работа проделывается в лаборатории, где преподаватели подготовили все оборудование для проведения экспериментов. Но отчего бы не иметь лабораторию под рукой, когда читаешь учебник?

Книга 2. Микроконтроллер и FlowCode



Книга 2. Микроконтроллер и FlowCode

Предисловие

Микроконтроллер – это контроллер, регулирующее или управляющее устройство в электронике, в миниатюрном исполнении в виде одной микросхемы. Основу работы микроконтроллера составляет взаимодополнение процессора и программы. Поэтому многие книги о микроконтроллерах начинаются с описания устройства процессора и продолжаются описанием языка ассемблера или языка высокого уровня, чаще языка Си, как наиболее употребительного в профессиональной практике.

По всей видимости, такой подход мешает начинающим осваивать программирование микроконтроллеров в программе FlowCode. Вольно или невольно, они пытаются найти в программе FlowCode те регистры, о которых идет речь в книгах, или те банки памяти, куда они могли бы записать операторы. Отдельный предмет их беспокойства и первых действий в освоении микроконтроллеров – создание программатора и поиск программы для обслуживания этого программатора.

Бесспорно, и знание устройства микроконтроллера, и программатор – все это нужно. Но не следует забывать, что применение микроконтроллеров – это в первую очередь умение создавать программы для них, а овладение искусством программирования, как и любым другим, требует достаточно много времени; в перерывах всегда найдется несколько свободных часов и для пайки программатора, и для чтения описания конкретной модели микроконтроллера. Найдется время и для изучения тех составляющих микроконтроллера, которые называются модулями ШИМ, АЦП или USART.

Программа FlowCode – это среда разработки программ для микроконтроллеров нескольких популярных видов PIC, AVR и ARM. В этом смысле есть программы, которые выглядят одинаково, но работают с выбранными типами микроконтроллеров, хотя есть возможность, например, программу, написанную для PIC-контроллера, импортировать в программу для работы с AVR-контроллерами и наоборот.

Как среда разработки программы, FlowCode в качестве основных компонентов предлагает наиболее употребительные языковые конструкции, которые можно найти, практически, в любом из языков высокого уровня: ветвление программы, цикл и т.п. Именно языковые конструкции и есть то, что следует искать в программе, из чего, кирпич к кирпичу, возводится здание программы. В отличие от других сред программирования, как MPLAB для PIC-контроллеров и AVRStudio для AVR, программа FlowCode в качестве основного языка программирования использует графический язык. И, как в объектно-ориентированном программировании, объекты FlowCode выполняют ряд операций и наделены набором свойств. Графическое программирование – отличительная черта и главное достоинство программы FlowCode, особенно для начинающих.

Демонстрационная версия программы, распространяемая бесплатно, имеет ряд серьезных ограничений, что может создать неверное представление о программе, как только о предназначенной для обучения. Это не так, даже в отношении демо-версии, в рамках которой вполне можно создать устройства весьма полезные.

Но, если говорить об обучении, то достоинство программы FlowCode еще и в том, что в качестве промежуточных результатов она записывает программу на языках Си и ассемблере. Что

позволяет получить наиболее часто используемые коды фрагментов программы для этих языков, и наверняка облегчит их освоение.

План этого рассказа: краткое описание программы FlowCode, краткое введение в программирование, пример создания простых программ в FlowCode, получение в программе FlowCode программных фрагментов, относящихся к наиболее употребительным операциям, на языке Си, пример переноса этих фрагментов в другую среду разработки (на языке Си).

Под кратким описанием программы FlowCode подразумевается не то, что следует изучать, но только то, что полезно иметь под рукой, когда работаешь в FlowCode. Многим начинающим не нравится, что первые примеры создания программы для микроконтроллера слишком примитивны: «Подумаешь, помигать светодиодом!». Но не следует забывать, что основное, что делает микроконтроллер – это «мигает» своими выводами, используете вы USART или ШИМ, работаете с индикатором или дисплеем. Да, можно написать программу решения тензорного уравнения, которая будет вызывать почтение, но будет ли она вам нужна? Да и каким образом это поможет вам разобраться и в без того насыщенном новыми элементами мире программирования? Не мудрствуя лукаво, используйте то учебное пособие, что есть в пакете программы FlowCode.

В этом рассказе будут использованы версия FlowCode для PIC-контроллеров и среда разработки MPLAB с компилятором PICC Lite (HI-TECH). Вы можете использовать аналогичную пару для AVR-контроллера.

Найти все необходимое можно на следующих сайтах:

http://www.matrixmultimedia.com/ (программа FlowCode)

http://www.microchip.com/ (среда программирования MPLAB)

http://www.htsoft.com/products/compilers/PICClite.php (компилятор Си HI-TECH)

http://sdcc.sourceforge.net/ (компилятор Си SDCC)

Последние версии программы MPLAB при установке позволяют загрузить и установить компилятор PICC Lite (бесплатная версия имеет ряд ограничений). Предыдущие версии достаточно легко позволяют использовать полнофункциональный и бесплатный компилятор SDCC.

Если у вас есть возможность приобрести (или собрать) программатор, работающий в среде FlowCode (или MPLAB, или AVRStudio), вы избавитесь от необходимости применять программу для обслуживания программатора.

Не ищите в этом рассказе решение вашей задачи. Здесь будет то, что необходимо вам в начале пути к той цели, которую можно было бы обозначить, как освоение работы с микроконтроллерами. Весь путь вам придется проделать самому. И если вы не хотите, чтобы он превратился в пытку, в изнуряющую необходимость идти, превозмогая себя, то не пытайтесь сразу найти самое таинственное место программы, которое превратит невозможное в возможное. Не пытайтесь сразу найти в одном месте ответы на все вопросы, которые у вас возникают, а наберитесь терпения и находите удовольствие в тех небольших победах, которые у вас обязательно будут, когда из непонимания вырастает знание, а из ошибок и неполадок работающее устройство.

Знакомство с интерфейсом программы FlowCode

Свободно распространяемая версия при первом запуске покажет следующее:

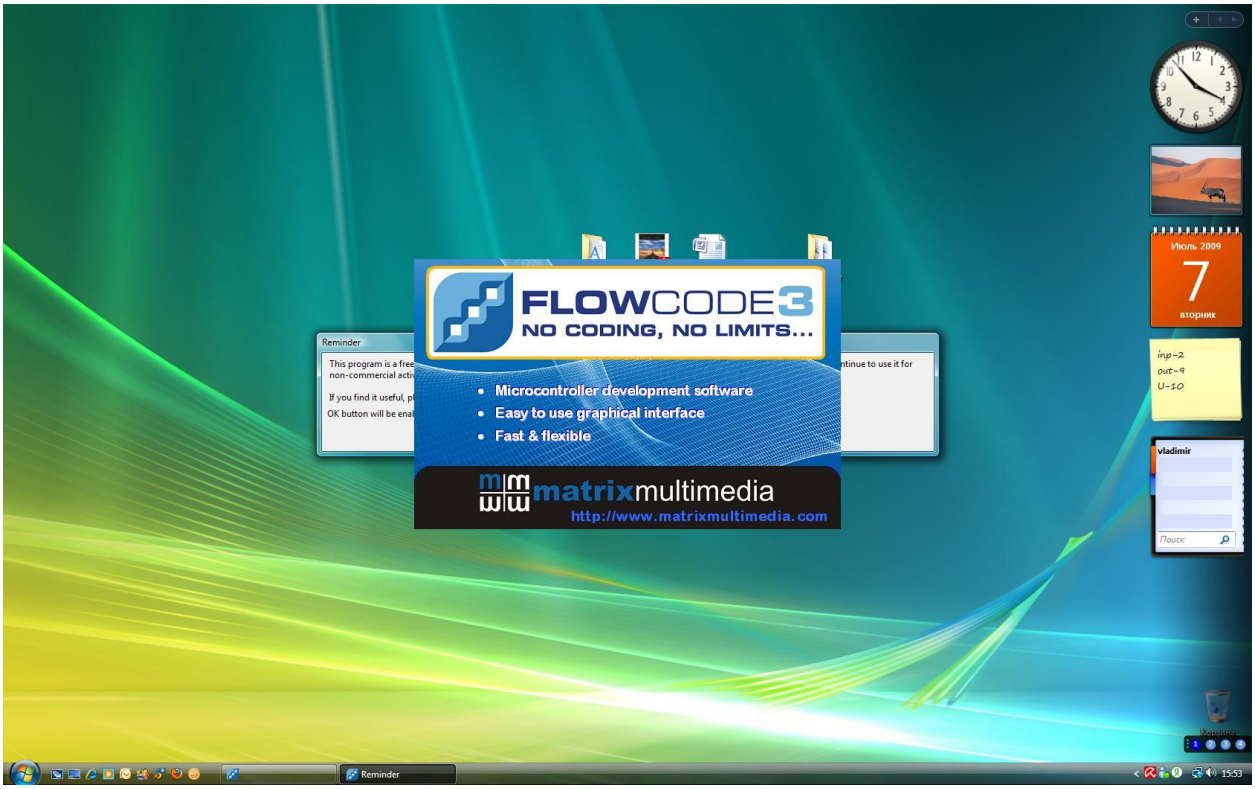


Рис. 1.1. Запуск демо-версии FlowCode

О том, что это демонстрационная версия постоянно напоминает заставка.

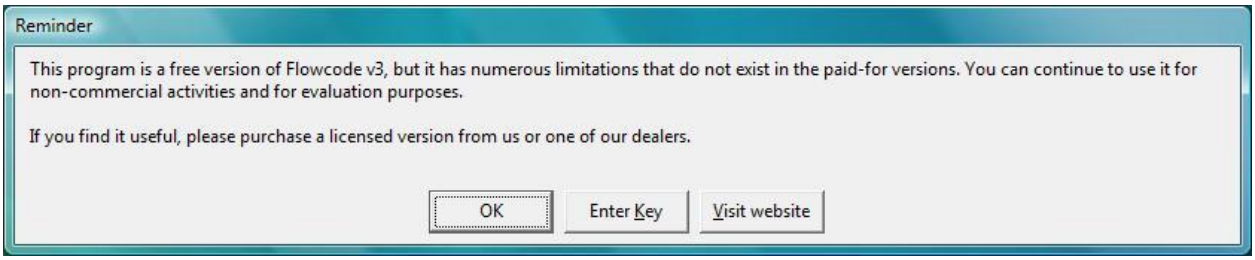


Рис. 1.2. Заставка демо-версии

Она же появляется и тогда, когда вы выходите из программы. Если не обращать внимания на эти мелочи и на то, что есть ряд существенных ограничений, то можно продолжить работу с программой. Следующий диалог предлагает вам создать новый проект или открыть уже существующий. Существующий проект можно открыть двойным щелчком в этом окне.

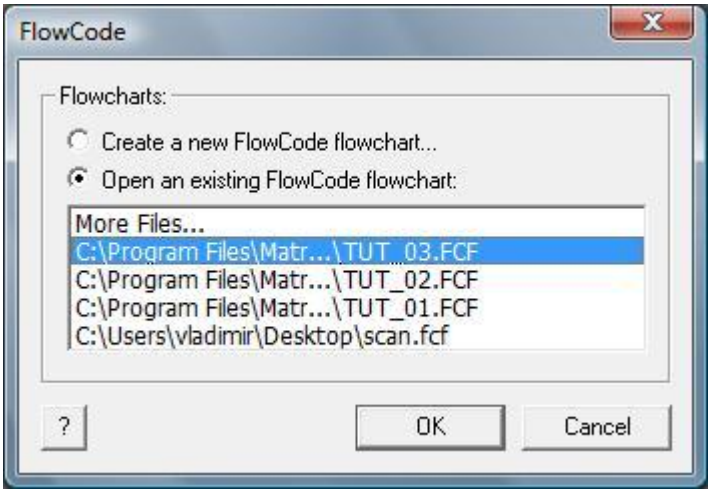


Рис. 1.3. Выбор продолжения работы

Если вы выбираете создание новой программы (опция *Create a new FlowCode flowchart*), то в следующем диалоговом окне будет предложено выбрать модель микроконтроллера (в демо-версии Flowcode_AVR только одна модель).

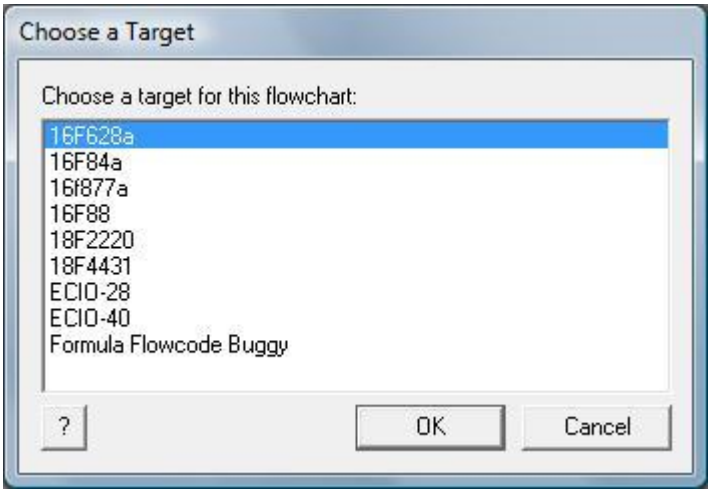


Рис. 1.4. Выбор модели контроллера

В полной версии программы этот список гораздо длиннее. Но этого более чем достаточно, если вы начинаете осваивать программирование микроконтроллеров. Конечно, вам захочется не только посидеть за компьютером, устанавливая программы и разбираясь в работе с ними, вам захочется запрограммировать микросхему, собрать что-то на макетной плате и посмотреть, как «оно живьем». Очень советую, по этой же причине и рассказываю, как правило, об одной единственной модели PIC16F628A, купить эту микросхему. В ней нет ничего, что так бы ее выделяло среди других, но она есть в демоверсии программы FlowCode, она достаточно хорошо (как PIC16F627A) поддерживается компилятором PICC Lite, программатор для нее можно быстро спаять и саму микросхему, если ее нет в ближайшем магазине радиотоваров, купить в почтовом агентстве «Десси». Стоит она не дорого. Позже, когда вы вполне освоитесь, у вас появятся критерии выбора микроконтроллера. Тогда и список поддерживаемых моделей вам понадобится длинный. А, может быть, и не понадобится. Микроконтроллер PIC16F628A не столь плох, чтобы не использовать его и дальше.

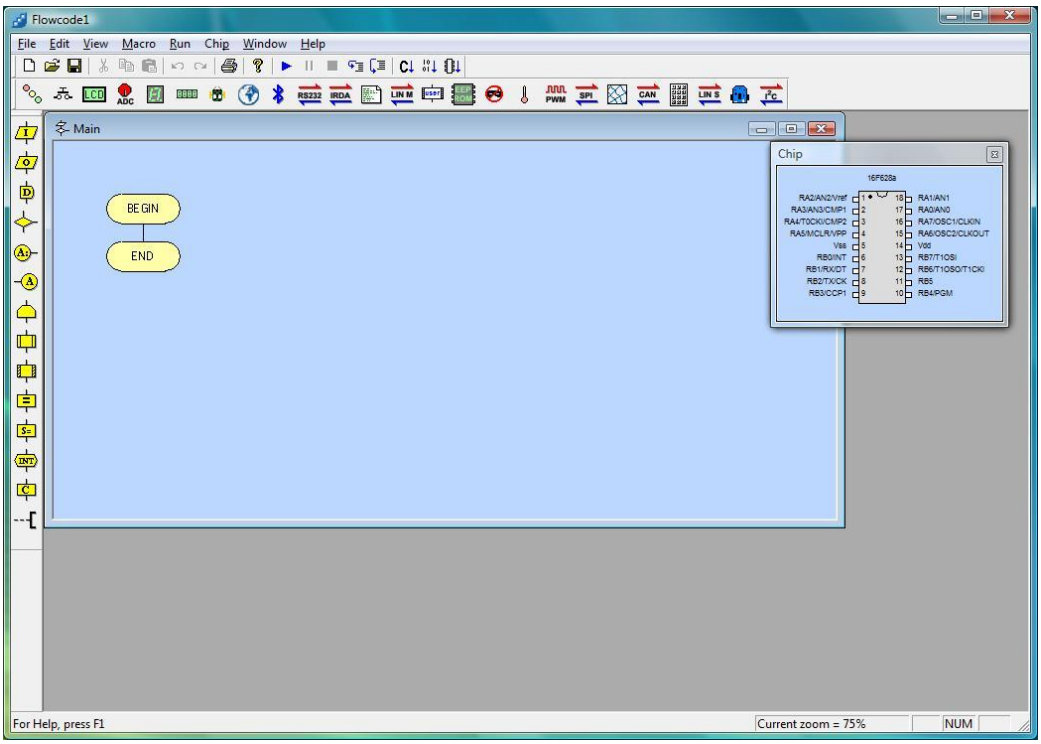


Рис. 1.5. Вид программы после выбора модели контроллера

Итак. Окно с надписью *Main* – это рабочее поле программы. Маленькое окошко с надписью *Chip* – это информационное окошко, где показана цоколевка выбранной модели. Программа, как любая другая, имеет основное меню и несколько инструментальных панелей. Каждая из них может быть перенесена в любое удобное для вас место – подцепите ее мышкой и перетащите.

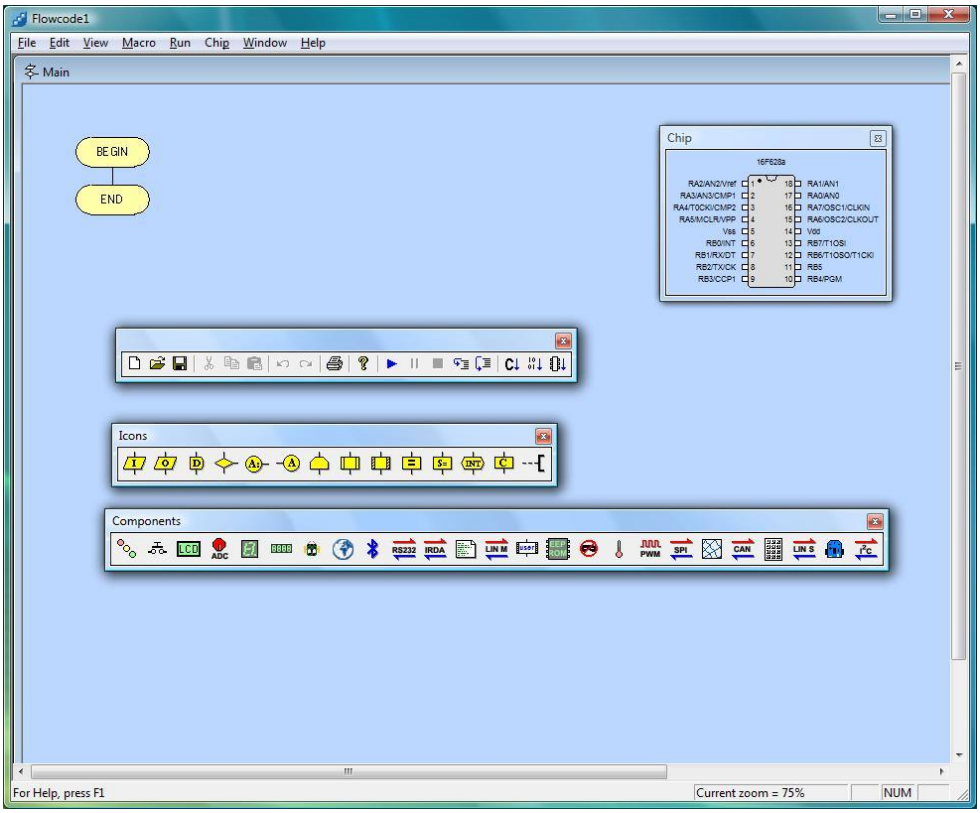
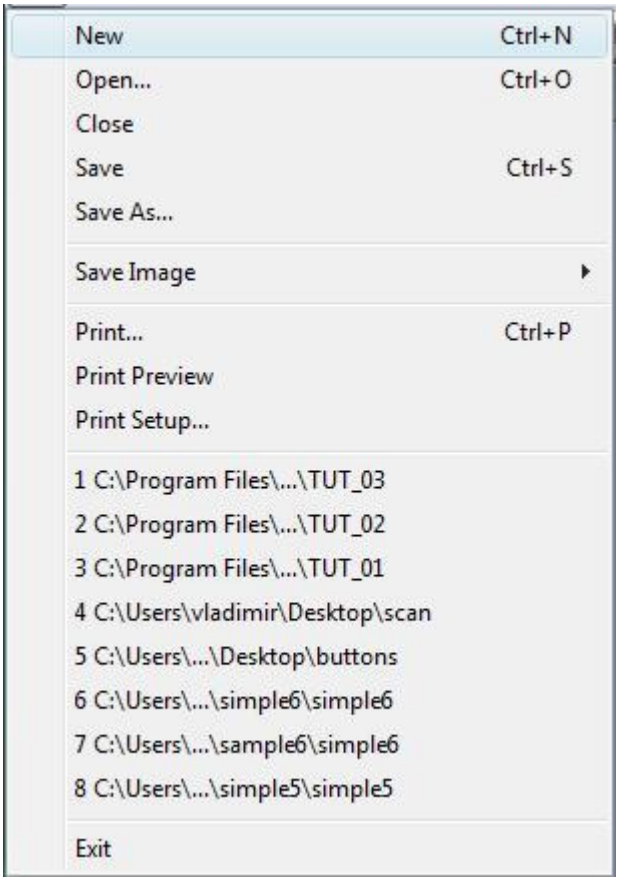


Рис. 1.6. Возможное расположение инструментальных панелей в FlowCode

Основное меню программы

Основное меню начинается, редко встречается другой вариант, с команд работы с файлами.



В полновесной версии в этом меню есть еще один полезный раздел – **Import**.

Его использование позволяет импортировать программу, написанную для AVR-контроллера, чтобы использовать ее с PIC-контроллером.

Рис. 1.7. Меню **File** основного меню программы

Меню содержит стандартный набор операций: ***New*** — создать новый файл; ***Open***... — открыть уже существующий; ***Close*** — закрыть файл; ***Save*** — сохранить файл; ***Save As***... - сохранить как..., удобно на тот случай, если вы хотите сохранить файл под другим именем или в другой папке; следующий пункт выпадающего меню ***Save Image***, позволяет сохранить вам программу в виде картинки в двух графических форматах, которые вы выбираете в открывающемся меню, если курсор наведен на этот пункт; пункты меню, начиная с ***Print***..., относятся к выводу на печать, последний из них позволяет настроить принтер, а ***Print Preview*** предварительно увидеть, как будет выглядеть вывод на печать. Завершают этот список перечень недавно открывавшихся файлов и выход из программы ***Exit***.

Почти все выпадающие подменю основного меню программы контекстно-зависимые: если нет объекта для выполнения операции, раздел меню не активен, надпись блеклая. Только тогда, когда есть, что делать конкретной команде, раздел меню активизируется, надпись становится четкой.

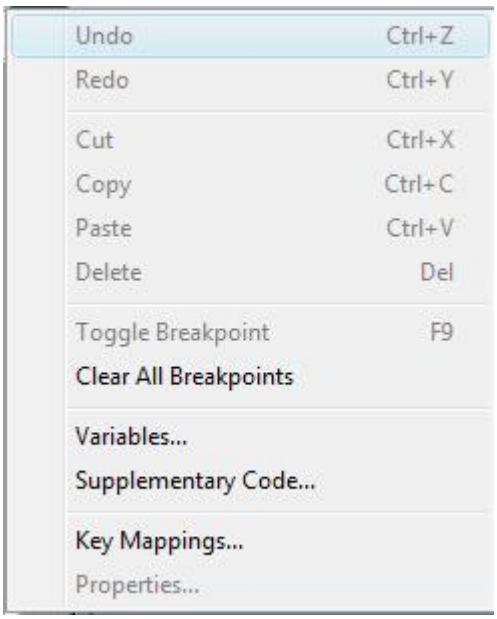


Рис. 1.8. Меню **Edit** основного меню

Я хочу подчеркнуть, что это вид выпадающего меню из списка основного меню, поскольку, щелкнув правой клавишей мышки в окне графического редактора, в зависимости от места, по которому вы щелкните, вы получите выпадающие меню для работы с элементами графики.

В этом меню первая половина относится к обычным операциям с объектами любого редактора: ***Undo*** и ***Redo*** — две взаимно противоположные команды по отмене последнего действия и возвращения его; ***Cut*** — вырезать; ***Copy*** — копировать; ***Paste*** — вставить; ***Delete*** — удалить. Два следующих пункта относятся к режиму отладки программы: ***Toggle Breakpoint*** — переключение точки останова (точка останова позволяет вам остановить выполнение программы при отладке, чтобы, например, посмотреть значения переменных), если точка останова была включена (задана), то она выключится, и наоборот; ***Clear All Breakpoints*** — очистит вашу программу от всех точек останова, которые вы задавали.

Следующий пункт в меню ***Variables...*** открывает диалоговое окно, в котором вы можете определить переменные вашей программы, если они вам нужны. Программы всегда полны переменных, с которыми они и работают.

Следующий пункт ***Supplementary Code...*** (дополнительный код) позволяет при желании (или необходимости) добавить код к графической программе, открывая окно диалога, где есть поле для объявления функции и поле для реализации этой дополнительной функции. Это соответствует языковым конструкциям с разделением ***Definitions and function declarations*** (определение и объявление функции) и ***Function implementations*** (реализация функции).

В соответствующих окнах мини-редакторов можно ввести текст, который после нажатия клавиши **OK** позволит использовать эту дополнительную функцию. Возможность добавить нужные коды в программу значительно расширяет функциональность FlowCode для профессионалов, но и любителям, уже освоившимся в программе, достаточно хорошо владеющим языком программирования, это дополнение не покажется лишним.

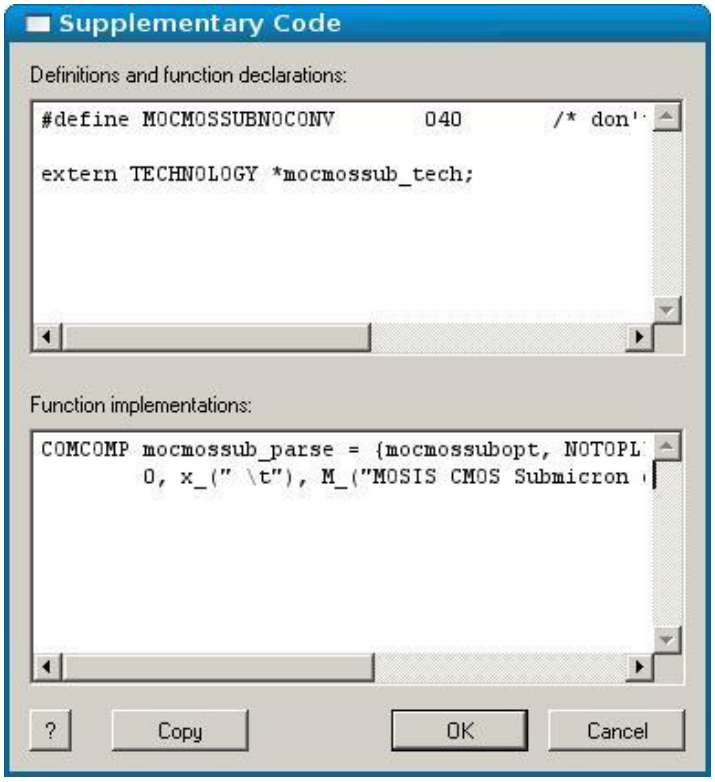


Рис. 1.9. Диалоговое окно добавления функции

Как я и говорил, все выпадающие меню в программе «чувствительны» к происходящему в активном окне редактирования. И пока вы не добавите для отладки в программу клавиатуру (**KeyPad**) из набора дополнительных элементов, следующий пункт ***Key Mappings***... (карта соответствий) не будет активен. Но с появлением клавиатуры в проекте вы можете изменить свойства этого дополнительного элемента.

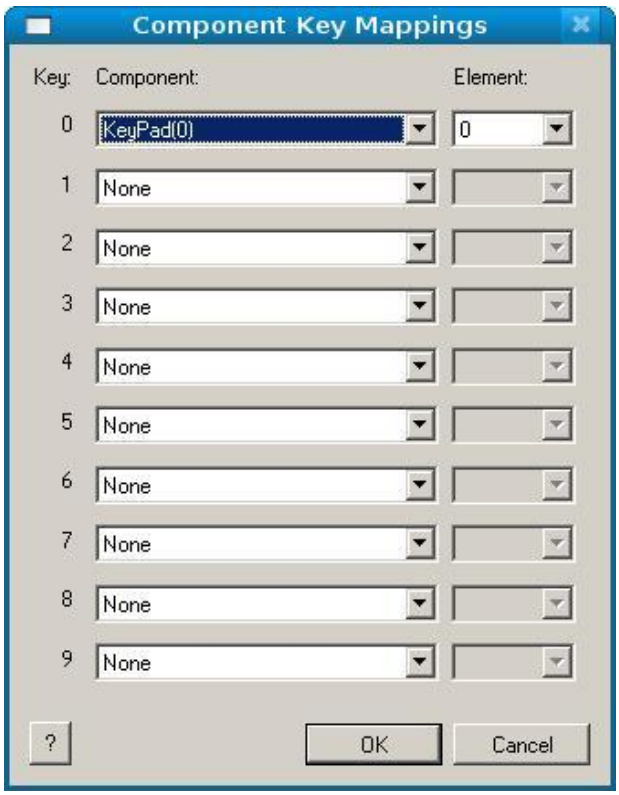


Рис. 1.10. Окно ***Key Mappings***

Аналогично поведение и последнего элемента списка из набора возможностей редактирования — ***Properties***... (свойства). Чтобы «оживить» его, вам следует выделить любой элемент программы. Тогда, щелкнув по этому пункту меню, вы попадете в окно диалога свойств выбранного элемента. Вид этого диалога и предоставляемые пользователю возможности зависят от выделенного элемента. Так для элемента **Input** (вход) можно задать привязку к переменной и к порту, выбрать работу с одним битом или всем портом, задать маску, поставив галочку в поле *Use Masking*.

Каждый элемент программы в FlowCode имеет свой набор свойств, назначение и смысл которых яснее всего проявляются тогда, когда вы начинаете работать над программой. У вас обязательно появятся вопросы вида — как сделать так, чтобы в конкретном месте программы проверить состояние только одного бита порта, а не всех вводов? Например, когда вы хотите использовать ветвление программы по условию, зависящему от состояния одного бита. Очень часто это проверка состояния «флага» (одного бита переменной или регистра, меняющего свое значение по окончании процесса или по результату операции).

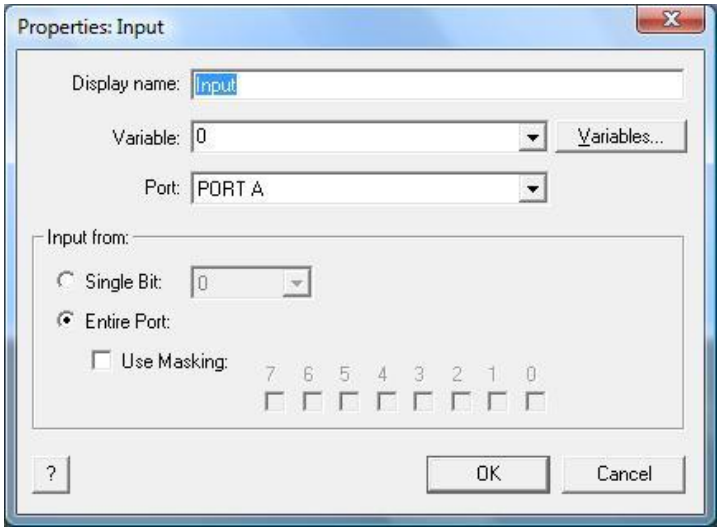


Рис. 1.11. Диалоговое окно свойств программного элемента **Input**

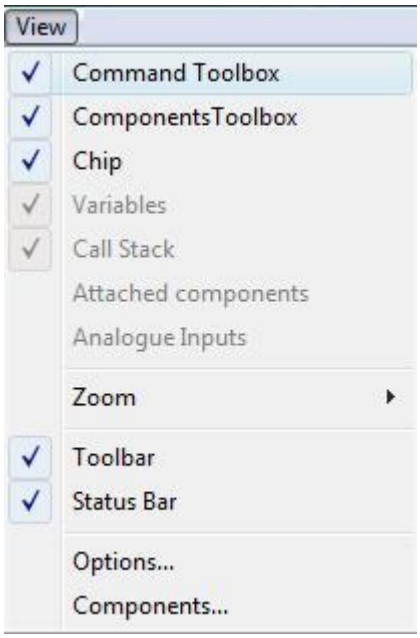


Рис. 1.12. Подменю **View** (вид)

«Галочки» слева показывают, какие компоненты вы желаете видеть при работе: ***Command Toolbox*** (инструментальное меню команд), ***Components Toolbox*** (инструментальное меню компонент), ***Chip*** (контроллер, с которым вы работаете), ***Variables*** (переменные, если они есть), ***Call Stack*** (стек вызовов подпрограмм), *Attached components* (присоединенные компоненты, если вы добавили клавиатуру или светодиоды для отладки), ***Analogue Inputs*** (аналоговые входы).

Присоединенные компоненты появляются в виде дополнительного списка, на что указывает стрелка справа от пункта меню. Как и дополнительный список масштаба изображения ***Zoom***, который вы можете выбрать, увеличить или уменьшить, привести к заполнению экрана или ширине окна редактирования.

Вы можете также включить или выключить ***Toolbar*** (основное инструментальное меню) и ***Status Bar*** (строку состояния). Вы можете изменить свойства проекта, такие как цвет рисунка и фона, шрифт, используя раздел ***Options...,*** или выбрать из предлагаемого списка, какие компоненты вам нужны в настоящее время для работы — ***Components...*** Все это позволяет вам выбрать комфортный в вашей работе вид программы: только нужные компоненты, цветовую гамму, возможность легко читать все надписи в элементах программы и т.д. Не забывайте только, что, выключив из списка компонентов что-то, вы не увидите этот компонент при следующем запуске программы, и, если сегодня он вам не нужен, то завтра может понадобиться.

Следующие пункты основного меню я опишу кратко. Тому несколько причин — многое из этого вам понадобится не сразу, а, когда понадобится, вы будете разбираться в среде программирования микроконтроллеров FlowCode не хуже меня; рассказ о перечне пунктов меню к тому времени, когда вам понадобится воспользоваться чем-то, забудется, и легче самому сообразить, чем найти в тексте описания; да и не интересно начинать работу с программой, если ты о ней все уже знаешь, гораздо приятнее делать «свои маленькие открытия».

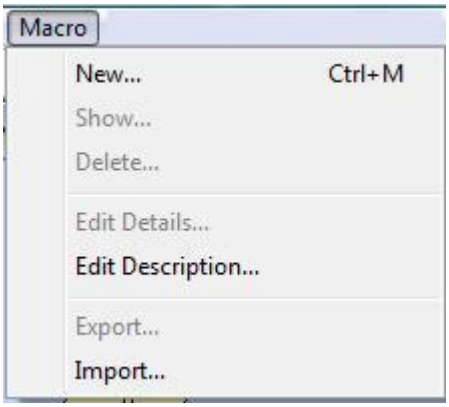
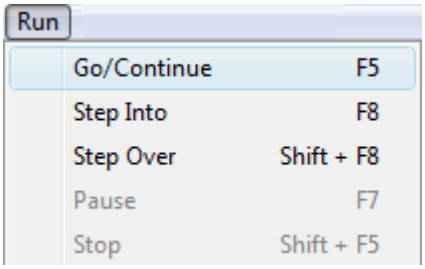


Рис. 1.13. Подменю **Macro** (макрос)

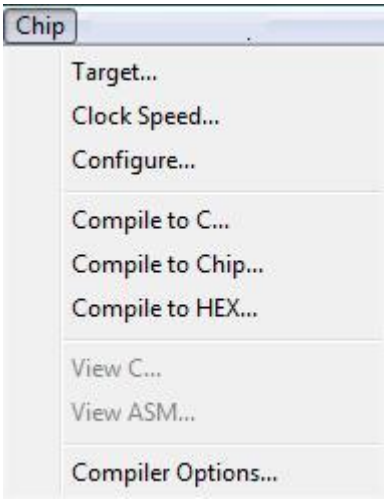
New... (новый макрос), ***Show...*** (показать макрос), ***Delete...*** (удалить), ***Edit Details...*** (редактировать детали), ***Edit Description...*** (редактировать описание, которое очень полезно делать), ***Export...*** (экспортировать), ***Import...*** (импортировать) предварительно экспортированную подпрограмму (макрос).



Go/Continue (запуск/продолжение), ***Step Into*** (шаг внутрь, например, функции), ***Step Over*** (шаг через, например, условие), ***Pause*** (пауза), ***Stop*** (стоп).

Рис. 1.14. Подменю работы с отладчиком программы

Отладка написанной программы – это очень важный компонент любой среды разработки программы. Этому больше внимания будет уделено в следующих главах.



Target... (выбор модели), ***Clock Speed***... (частота тактового генератора), ***Configure***... (слово конфигурации), ***Compile to C***... (транслировать на Си), ***Compile to Chip***... (транслировать все и загрузить), ***Compile to Hex***... (получить hex-файл загрузки), ***View C***... (просмотр Си кода), ***View ASM***... (просмотр ассемблерного кода), ***Compiler Options***... (опции компиляции).

При установке программы есть один момент, имеющий отношение к пункту ***Configure***.

Рис. 1.15. Подменю раздела **Chip**

При установке программы появляется следующее окно диалога.

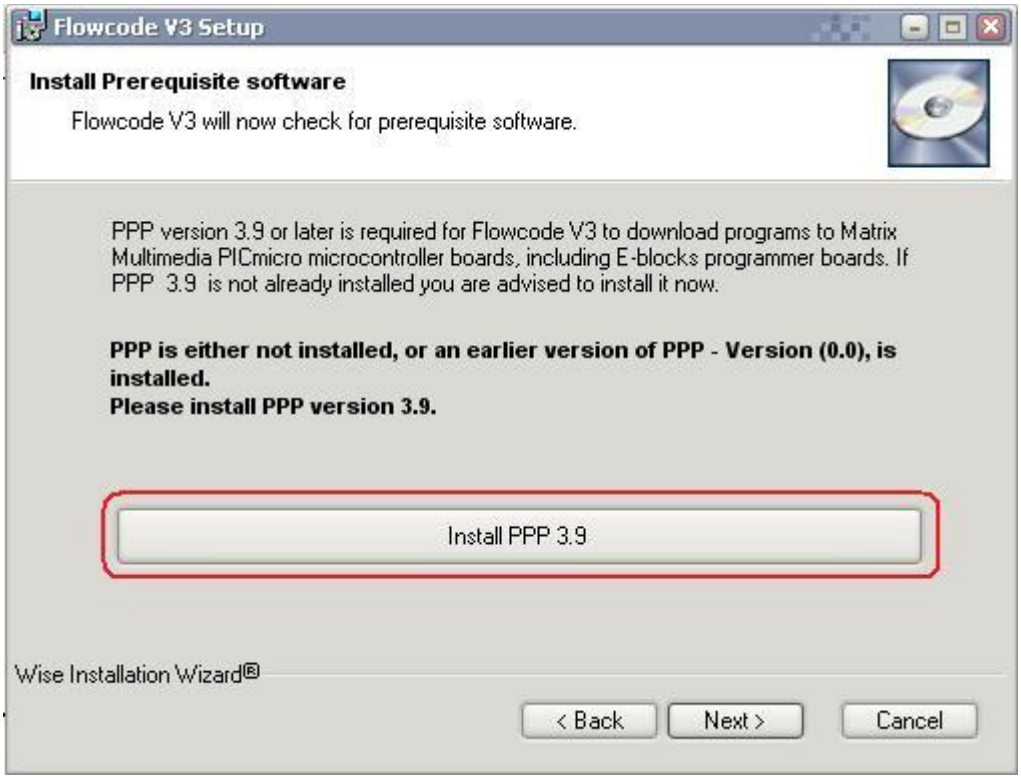
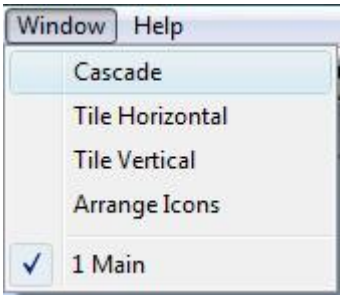


Рис. 1.16. Один из моментов установки программы FlowCode

Обязательно нажмите отмеченную на рисунке кнопку **Install PPP 3.9** до того, как нажмете кнопку **Next >**. При установке этой части программы в папке, где расположена программа FlowCode появляется папка с именем *Common*. В ней располагается то, что позволит вам ввести слово конфигурации для микроконтроллера.



Черепицей, горизонтально, вертикально, упорядочить иконки — вот, что можно сделать с окнами.

Рис. 1.17. Подменю **Window**

Последний пункт основного меню **Help**, на тот случай, если есть желание или необходимость обратиться к руководству по работе с программой.

Инструментальная панель программных компонентов

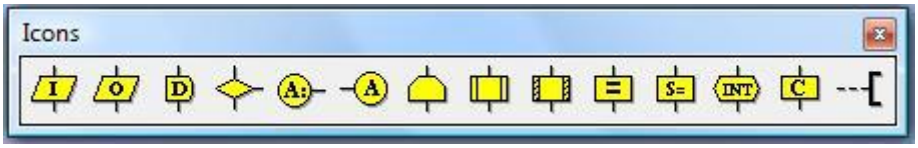


Рис. 1.18. Программные компоненты

Перечень представленных компонентов (слева-направо на рисунке, сверху-вниз, когда панель справа):

Input (ввод), ***Output*** (вывод), ***Delay*** (пауза), ***Decision*** (ветвление), ***Connection Point*** (две точки соединения), ***Loop*** (цикл), ***Macro*** (макрос), ***Component Macro*** (макрос компонента, добавленного в программу), ***Calculation*** (вычисление), ***String Manipulation*** (строковые операции), ***Interrupt*** (прерывание), ***C Code*** (блок кода на языке Си), ***Comment*** (комментарий).

Все эти названия появляются в виде подсказки, когда курсор мышки наведен на компонент.

Рассмотрим некоторые из программных компонентов подробнее. Так «взаимодополняющие» или «взаимоисключающие» компоненты **Input** и **Output** имеют очень похожие диалоговые окна свойств. Известно, что, как правило, любой из выводов порта микроконтроллера можно назначить для входа или для выхода. Можно это сделать и для всего порта. Тогда весь порт будет работать, как входной или выходной. Эти свойства выбираются в диалоговом окне (после добавления компонента к программе достаточно дважды щелкнуть по нему левой клавишей мышки или выделить одинарным щелчком и воспользоваться основным меню ***Edit->Properties***).

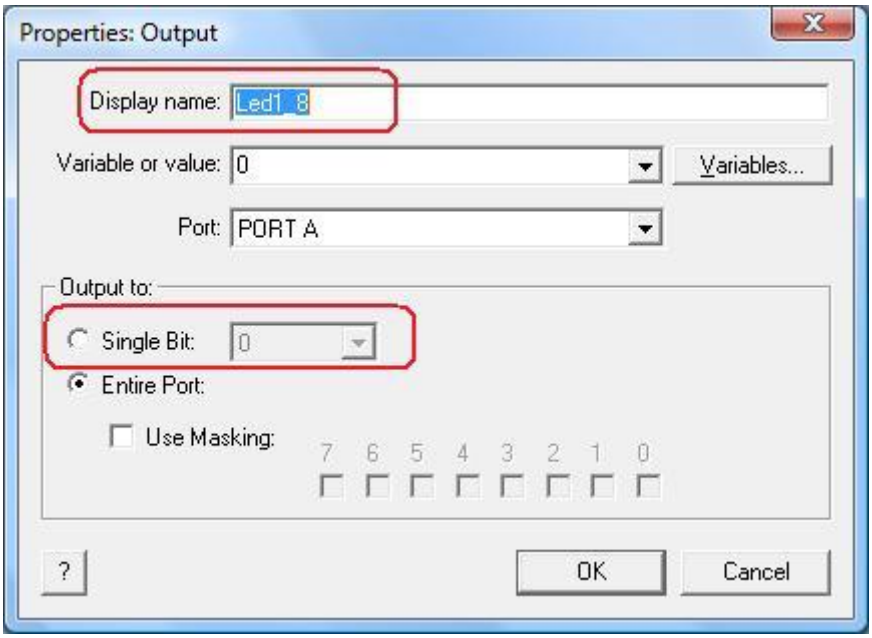


Рис. 1.19. Диалоговое окно программного компонента **Output**

По умолчанию в окне *Display name:* (отображаемое имя) появляется *Output*. Его можно оставить, но можно заменить более понятным, скажем, для вас. Работая над программой, легче понять ее, а «разрастается» она быстро, когда вы описываете назначение всех элементов

программы. FlowCode позволяет вписать название кириллицей. Но я не советую это делать. Лучше написать что-то в роде: svetodiody, – если вы, как и я, не слишком дружите с английским языком.

На рисунке отмечено, что вы можете использовать единственный бит (единственный вывод) порта. По умолчанию используется весь порт. Кнопка справа от окошка со стрелкой вниз позволяет выбрать из выпадающего списка нужный бит. А кнопка справа от окошка с именем порта предлагает список портов.

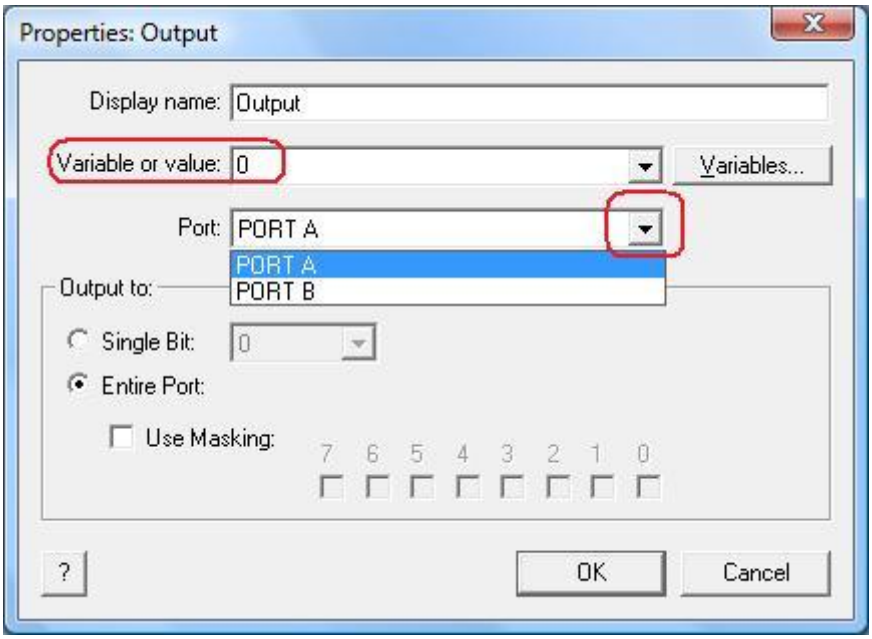


Рис. 1.20. Выбора нужного порта для выхода

Чуть выше вы задаете значение (*Variable or value:*), которое нужно вывести в порт или задать биту порта. Используя кнопку ***Variables*** можно выбрать существующую переменную для вывода в порт. Закончив настройку выхода, вы можете подтвердить сделанные изменения, нажав на кнопку ***OK***, или можете отменить все с помощью кнопки ***Cancel***.

Следующий программный компонент – **Delay** (пауза). Нужный компонент, особенно когда вы формируете выходной импульс, а формирование импульсов очень часто используется. После добавление компонента в его диалоговом окне задается значение.

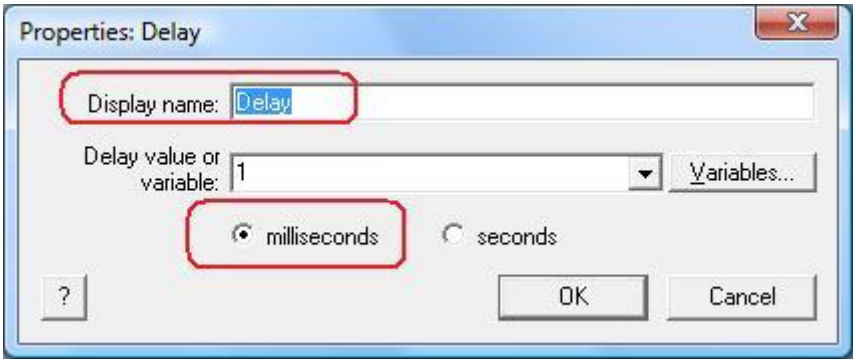


Рис. 1.21. Диалоговое окно компонента **Delay**

Как и в других окнах диалога, можно этому компоненту дать имя, которое отражало бы его назначение. По умолчанию пауза задается числом миллисекунд. Изменив опцию на *seconds*,

можно задать время в секундах. Кроме того, используя клавишу ***Variables***, можно выбрать переменную, если она была создана для этого случая, которая задаст значение для паузы.

Есть некоторое неудобство в том, что программа FlowCode оперирует с целыми числами, и нельзя задать время паузы, скажем, 0.1 мс. Эту проблему можно обойти, если использовать вставку на языке Си, ассемблере или задавая счетный цикл с количеством нужных проходов. Но об этом позже.

Следующий программный компонент – ***Decision*** (ветвление). Только очень простые программы обходятся линейным написанием, когда выполняется какое-то количество операций подряд. Да и такие программы, как правило, выполняются в цикле. Обычно контроллер отслеживает события, происходящие на его входах (или входе), по результатам опроса входов программа проходит по одной или другой ветке программы. Например, такой фрагмент программы:

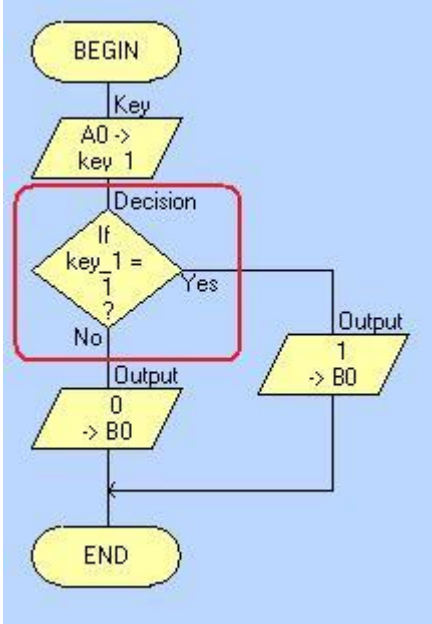


Рис. 1.22. Фрагмент программы с компонентом **Decision**

В этом фрагменте нулевой вывод порта А назначен на вход. Если его состояние (он связан с переменной *key_1*) равно единице, то программа проходит по ветке ***Yes***, если нулю, то по ветке ***No***. В первом случает нулевой вывод порта В (назначенный на выход) принимает высокий уровень, то есть, единицу, во втором низкий, то есть, ноль. Если этот фрагмент заключить в бесконечный цикл, то можно посмотреть в отладчике FlowCode, как программа реагирует на нажатие кнопки.

Условие ветвления вводится в диалоговом окне.

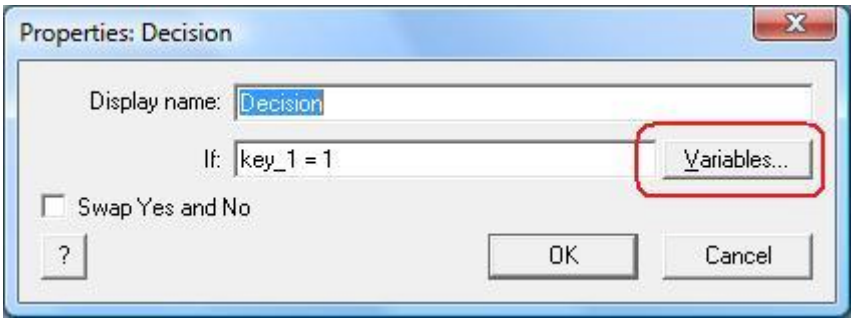
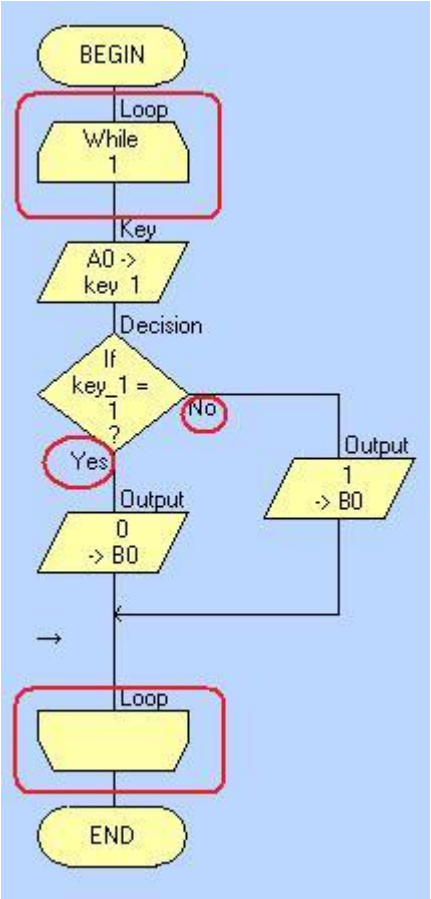


Рис. 1.23. Диалоговое окно компонента **Decision**

С помощью кнопки ***Variables***, отмеченной на рисунке, я выбрал ранее созданную (когда оформлялся ввод) переменную, продолжив запись условия ветвления. Это условие может быть простым, как выше, но может быть и сложным, использующим, например, логические операции и переменные.

Опция *Swap Yes and No* меняет местами эти ветки. Иногда так удобнее.

О следующих двух (они используются парой) компонентах мы поговорим позже, а сейчас остановим внимание на очень важном элементе программы – **Loop** (цикл). Приведенный выше фрагмент программы вполне «правильный». Логически. Но «нежизненный». Даже проверить его работу с помощью отладчика можно только в пошаговом режиме. А, если скомпилировать программу и загрузить в микросхему, то даже проверить ее не получится. Программа выполнится так быстро, что нажать кнопку не успеешь; правда, можно один раз включить схему при отжатой кнопке, а второй раз при нажатой. Но ценность такой программы равна нулю. Другое дело, если мы добавим бесконечный цикл. Заодно я использую опцию *Swap Yes and No*, о которой говорил выше.



Обратите внимание на ветки (отмеченные) *Yes* и *No*. И сравните с предыдущим фрагментом.

Цикл «окаймляет» программу, начинаясь с условия выхода из цикла. В данном случае *While 1*, поскольку никаких условий выхода из цикла я не задавал. Это не заданное условие, как и заведомо не выполнимое условие, приведут к тому, что программа, заключенная в цикл, будет повторяться бесконечно.

Такие бесконечные циклы опасны, если созданы по ошибке, ошибка при написании программы или сбой программы, и опасны тем, что программа «повисает». Она перестает реагировать на внешний мир или реагирует только на часть нужных команд.

С другой стороны, когда это сделано осознанно, так работает почти каждый контроллер: он «крутится» в бесконечном цикле.

Рис. 1.24. Фрагмент вышеприведенной программы с циклом

Свойства цикла, как и других программных компонентов, задаются в его диалоговом окне.

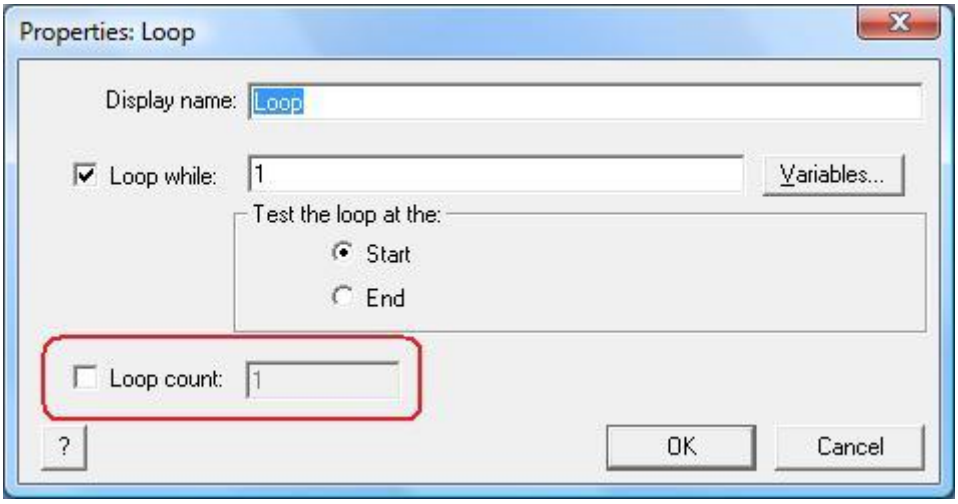


Рис. 1.25. Диалоговое окно компонента **Loop**

С помощью этого диалога можно задать условие выхода из цикла, используя кнопку ***Variables*** для выбора или создания переменной. Как правило, эта переменная должна меняться внутри цикла. Если в вышеприведенном фрагменте использовать переменную, связанную с кнопкой входа порта A, то можно при нажатии, скажем, кнопки выходить из цикла. Иногда переменную создают внутри цикла, которая меняется при работе фрагмента программы, заключенного в цикл, а когда достигает заданного значения, цикл прекращает работу.

Проверка условия завершения цикла может происходить в начале цикла, если установлена опция *Start*. При этом если условие выполняется до выполнения программы, заключенной в цикл, то программа не будет выполнена ни разу. Если установить опцию *End*, то проверка выполнения условия будет происходить в конце программы, заключенной в цикл. При этом программа будет выполнена хотя бы один раз.

В этом же диалоге можно сменить вид цикла. Ранее мы говорили об условном цикле, то есть, таком, когда программа внутри цикла выполняется до тех пор, пока не будет выполнено условие выхода из цикла. Если установить опцию *Loop count*, задав в окне число, то цикл будет выполняться до того момента, когда количество проходов программы в цикле станет равным заданному числу.

Следующий программный компонент – ***Macro*** (макрос). Или подпрограмма. Такой же смысл имеют процедуры и функции в языках программирования высокого уровня. Обычно нужда в этом элементе программы обусловлена тем, что к нему обращаются много раз за время работы программы, а выполняемые им операции одни и те же. В языке Паскаль, например, различают процедуры – фрагменты программы, в которых выполняются операции – и функции, когда производятся вычисления. В языке Си оба вида объединены под именем «функция». В FlowCode для этого введен компонент **Macro**.

После добавления компонента в программу мы можем перенести в него ранее созданную процедуру. Для этого дважды щелкнем по компоненту **Macro**, открывая диалоговое окно.

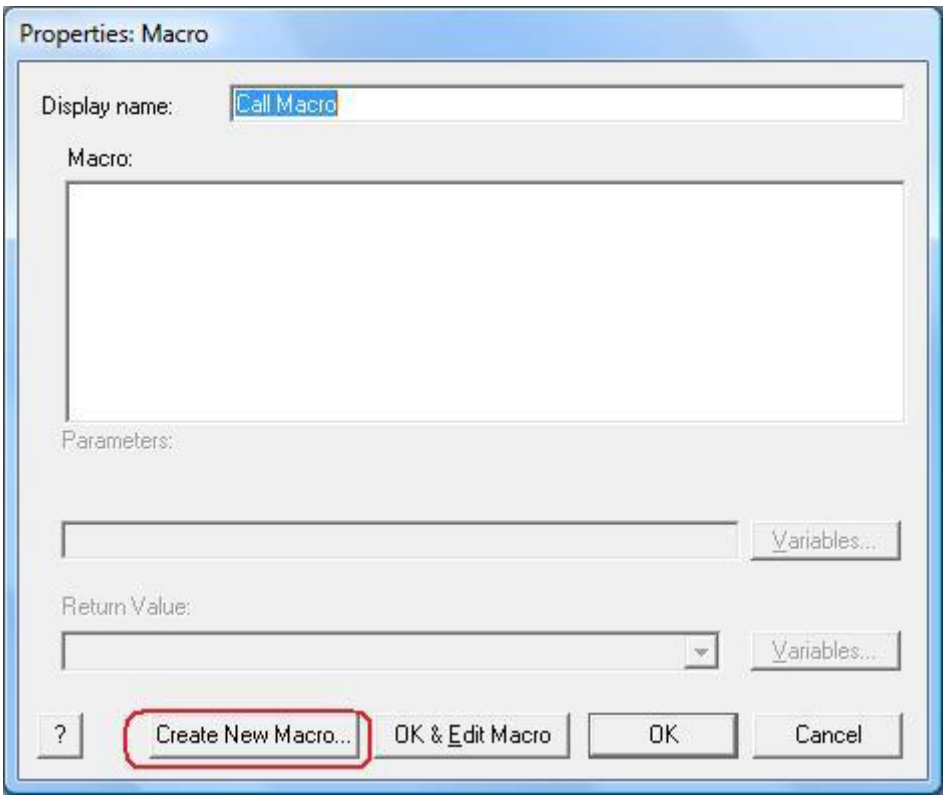


Рис. 1.26. Диалоговое окно компонента **Macro**

И вновь, имя компонента можно изменить. Если подпрограмма еще не создана, а мы только что добавили компонент, следует использовать кнопку ***Create New Macro***.

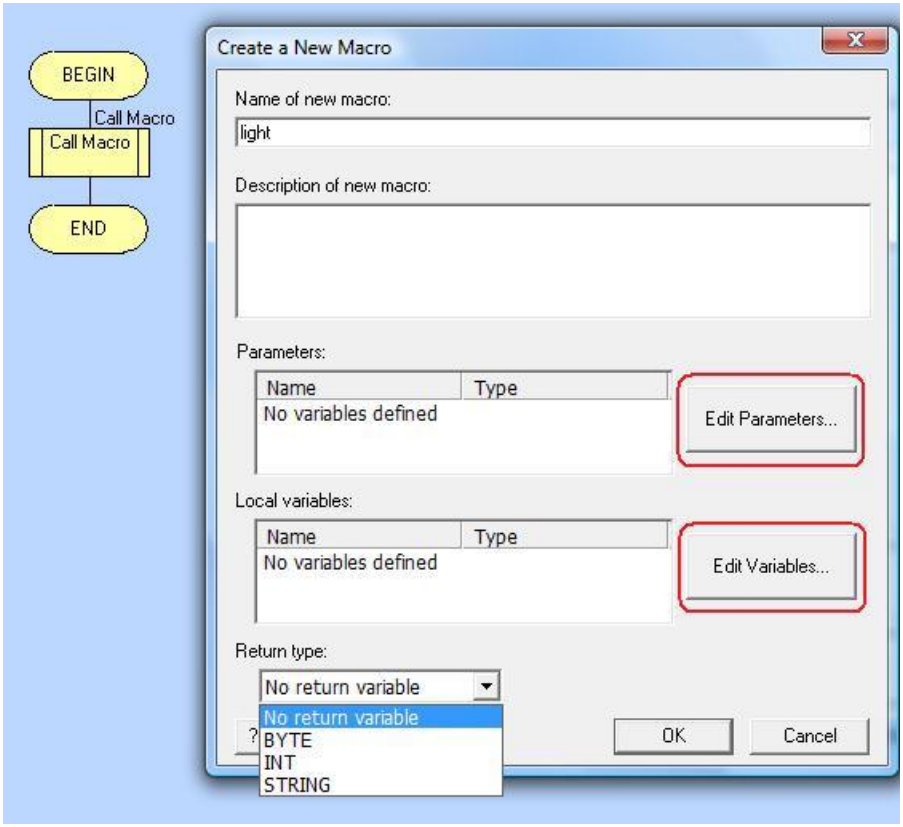


Рис. 1.27. Диалоговое окно создания нового макроса

Кроме имени новой подпрограммы, как и для функции или процедуры можно задать параметры (кнопка ***Edit Parameters***), можно создать локальные переменные (кнопка ***Edit Variables***), можно задать тип возвращаемой переменной (если подпрограмма возвращает переменную), который выбирается из выпадающего списка. Нажав кнопку ***OK***, вы возвратитесь в предыдущий диалог. Если нажать кнопку ***OK & Edit Macro***, то программа открывает второе рабочее окно для создания подпрограммы.

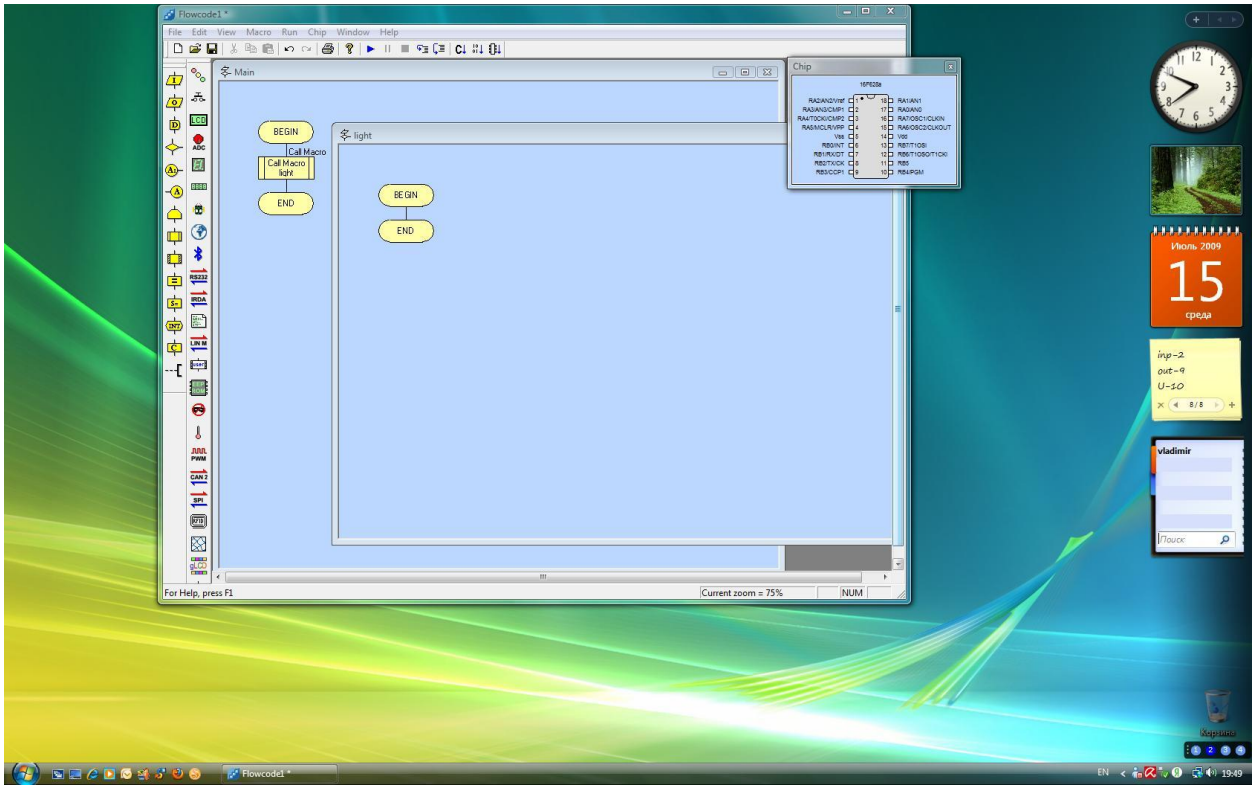


Рис. 1.28. Второе рабочее окно редактора для создания подпрограммы

К этому мы вернемся позже, а сейчас отметим, что между созданием программы и созданием подпрограммы в FlowCode в техническом плане нет разницы.

Опустим ряд очень полезных программных компонентов, назначение которых, свойства и применение удобнее рассмотреть на конкретных примерах, и рассмотрим два компонента: ***Calculation*** (вычисление) и ***Comment*** (комментарий).

Необходимость в компоненте ***Calculation*** возникает уже тогда, когда необходимо произвести присваивание значения переменной. В разных языках программирования этот оператор может иметь разный вид. В FlowCode он совпадает со знаком равенства.

Используемый для расчетов компонент может содержать одно или несколько выражений, результаты операций можно использовать дальше в программе.

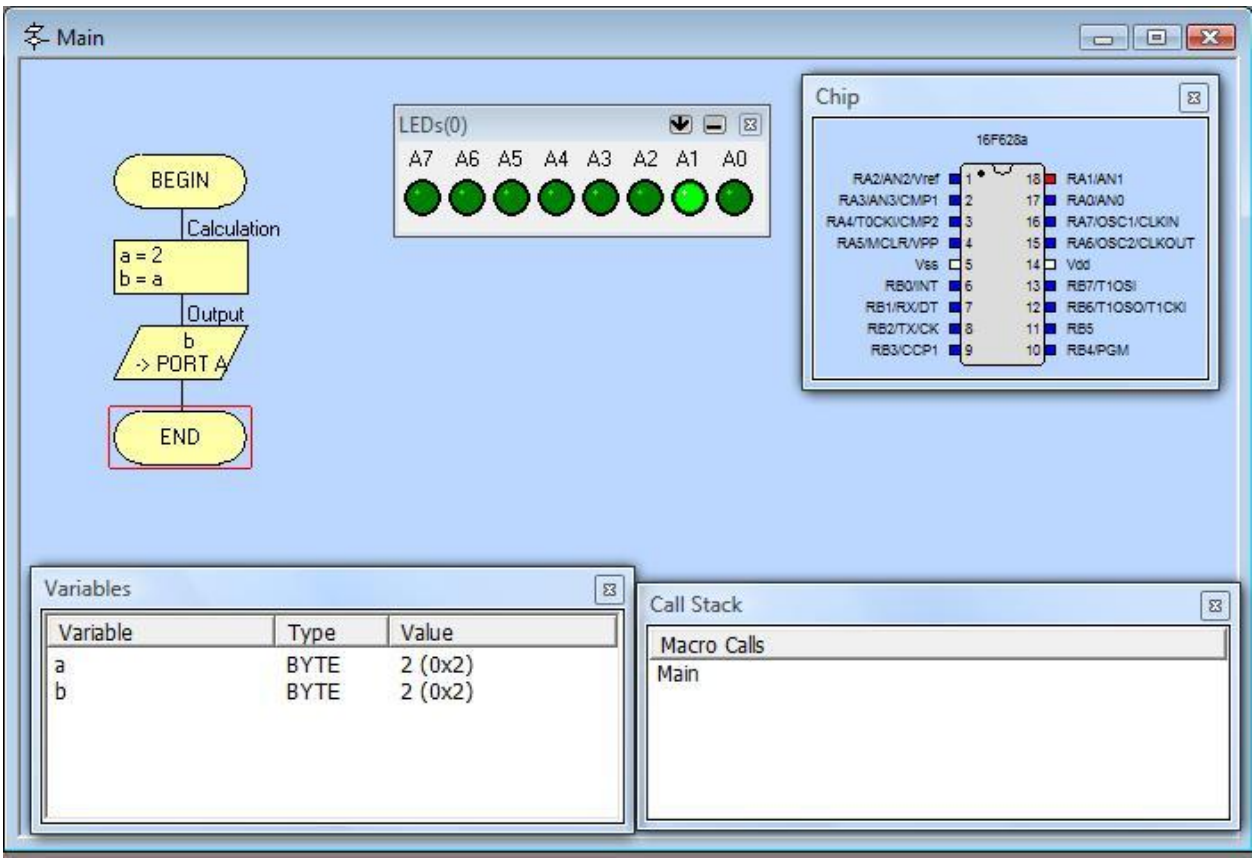


Рис. 1.29. Использование компонента ***Calculation***

Все необходимые операции записываются в окне диалога.

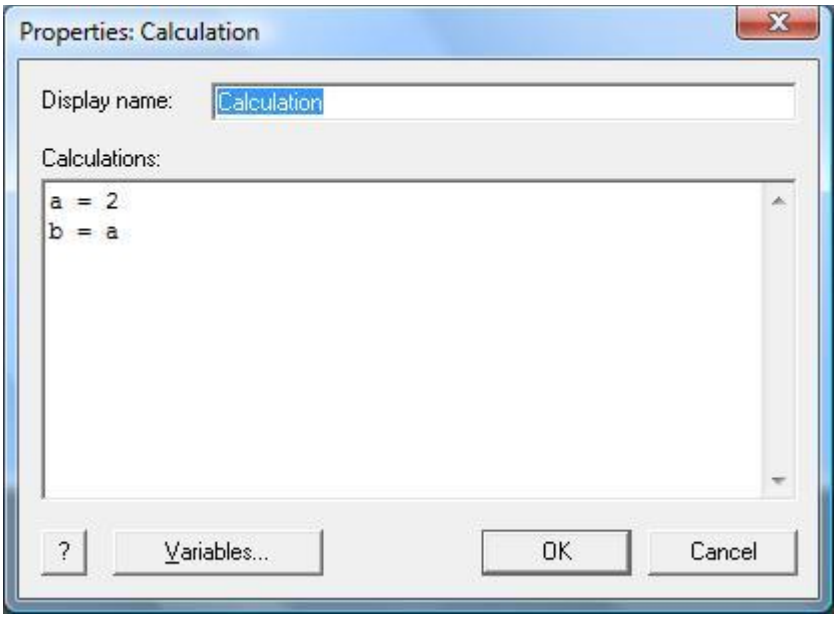


Рис. 1.30. Диалоговое окно компонента ***Calculation*** (вычисление)

Каждое новое выражение следует писать с новой строки, если в выражении допущена синтаксическая ошибка, появляется сообщение об ошибке, а строка, где допущена ошибка, подсвечивается. Кнопка ***Variables*** служит для выбора переменных из списка всех переменных программы.

Программный компонент ***Comment*** (комментарий) используется для добавления к программе пояснений. Хотя формально все, что стоит за знаком комментария (или внутри скобок комментария), компилятор должен игнорировать, лучше писать комментарии латиницей, иначе могут возникать проблемы.

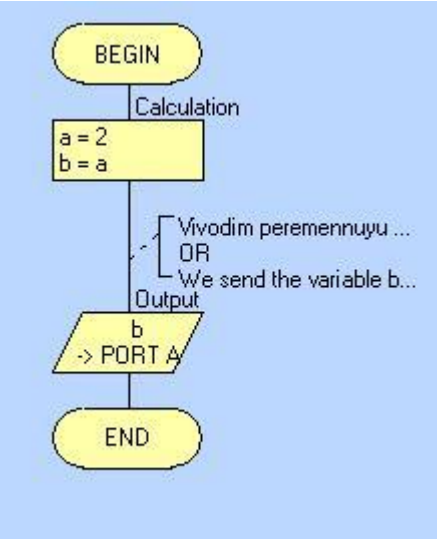


Рис. 1.31. Использование компонента ***Comment***

Для ввода комментария служит диалоговое окно свойств компонента ***Comment***.

Не менее важна для создания программ другая инструментальная панель – панель дополнительных компонентов. Она содержит элементы, необходимые при отладке программы. Набор кнопок, используемый для изменения состояния входов, или линейка светодиодов могут использоваться не только тогда, когда устройство рассчитано на подключение, скажем, светодиодов. Светодиоды линейки можно использовать в качестве программных «заглушек», на месте которых позже появятся подпрограммы или прерывания; их можно использовать для отслеживания событий при тестировании программы.

Кроме внешних составляющих схемы, как семисегментный индикатор или линейка таких индикаторов, на панели есть модули, часто встраиваемые в микроконтроллеры. Если выбранная вами модель контроллера имеет, скажем, встроенный модуль USART, вы сможете проверить работу контроллера с помощью компонента RS232.

О назначении дополнительных модулей и их применении удобнее рассказать тогда, когда будут рассматриваться примеры программ.

Основная инструментальная панель

Рис. 1.32. Группа работы с файлами основной инструментальной панели

Основная инструментальная панель содержит несколько групп кнопок, повторяющих операции основного меню, начинаясь с группы работы с файлами: **New** (создание нового файла), **Open** (открытие существующего файла), **Save** (сохранение файла).

Диалоговые окна всех операций типовые. И здесь можно обратить внимание на то, что в папке Examples есть много примеров – обучающие программы.

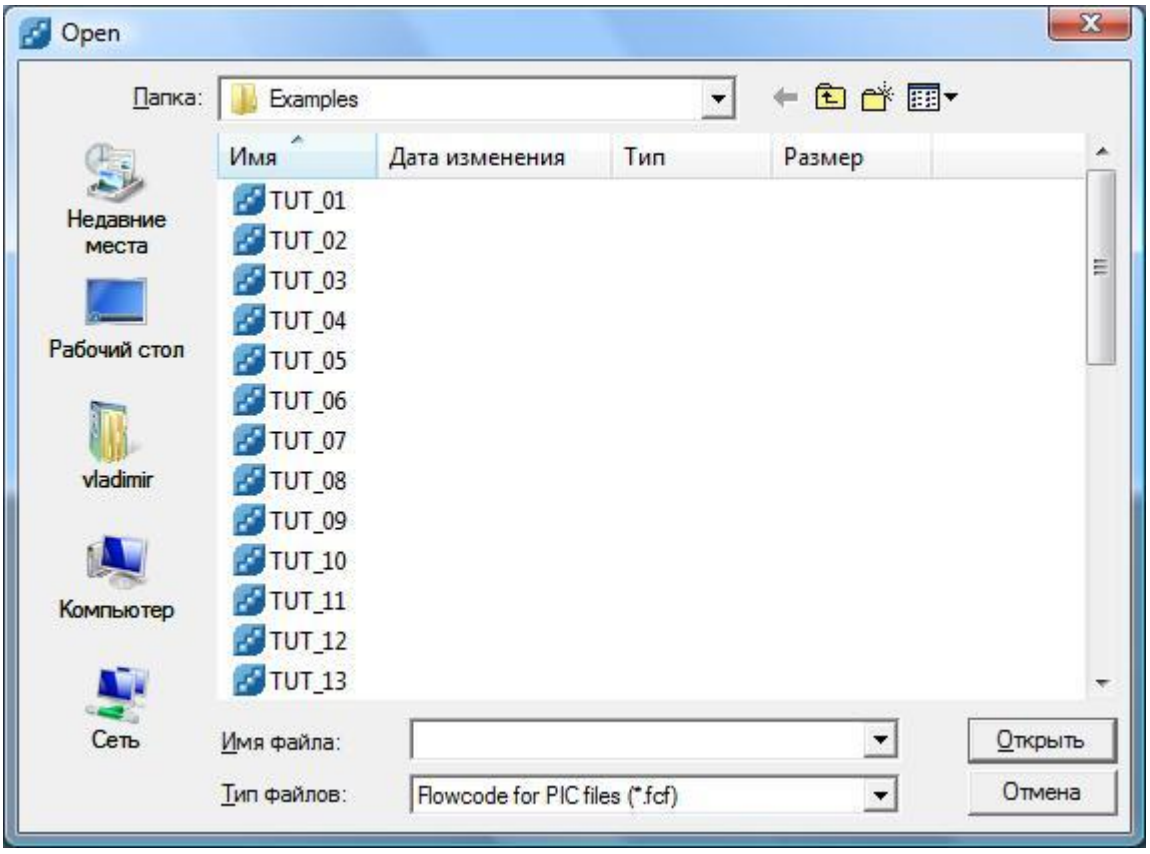


Рис. 1.33. Файлы примеров программы FlowCode

Следующая группа кнопок используется при редактировании.

Рис. 1.34. Группа редактирования основной панели

Cut (вырезать), **Copy** (копировать), **Paste** (вставить), **Undo** (отменить), **Redo** (вернуть) – обычный набор любого редактора. Пусть текст программы состоит не из букв, а из слов и фраз, но к ним применимы все операции редактирования, как к любому тексту.

Прежде, чем применять эти операции следует выделить элемент программы – достаточно щелкнуть по нему – или выделить фрагмент программы: поместить курсор на свободном месте рабочего поля над нужным фрагментом, нажать левую клавишу мышки и, удерживая ее, обвести прямоугольником выделения фрагмент программы.

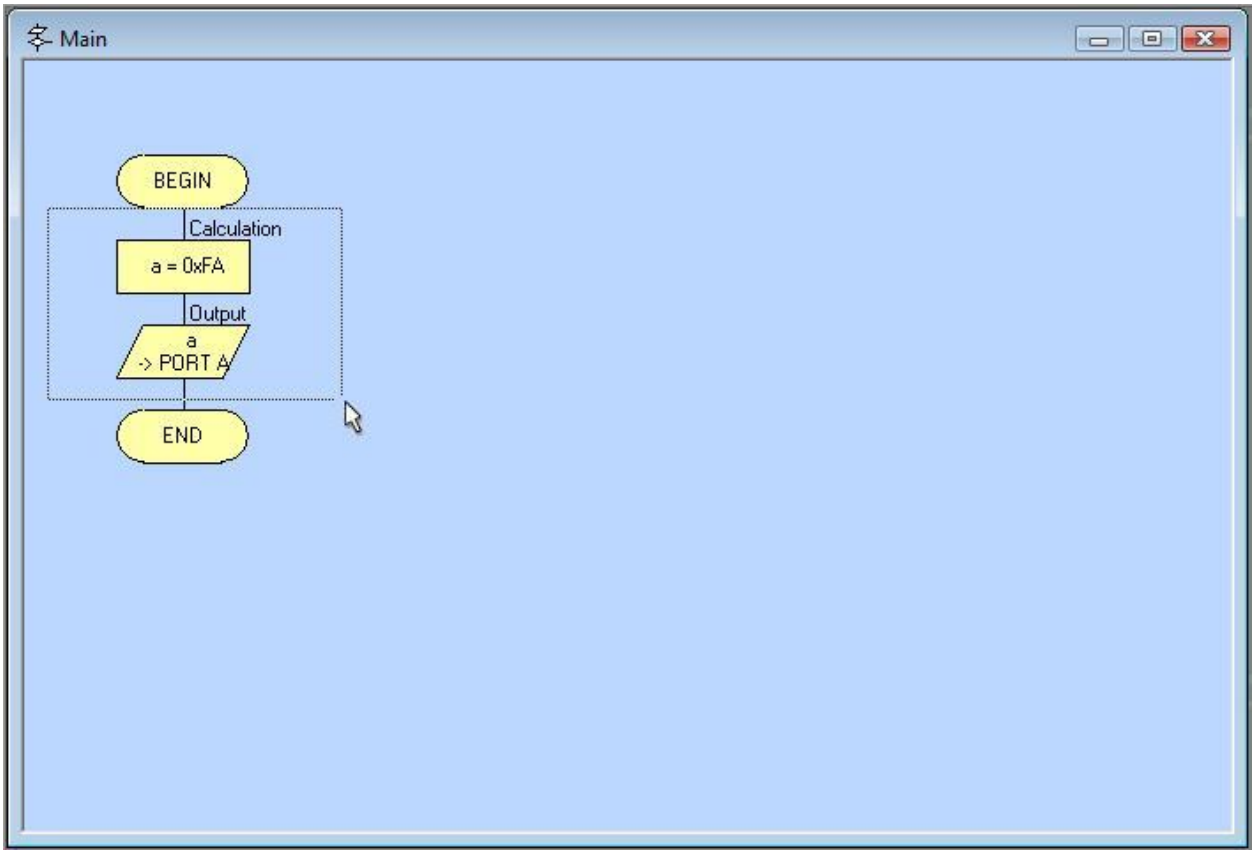


Рис. 1.35. Выделение фрагмента программы

После того, как вы отпустите клавишу, выделенный фрагмент меняет цвет. Теперь его можно вырезать, копировать, вставлять. Создав подпрограмму, можно перенести уже отлаженный фрагмент программы в подпрограмму, просто скопировав его.

Следом за командами редактирования на инструментальной панели расположены команды печати, команда распечатает программу на принтере, и команда вывода информации о версии программы.

Следующая группа команд относится к отладке программы.

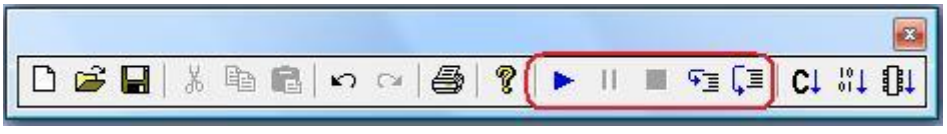


Рис. 1.36. Группа команд отладки

Первая кнопка «запускает» программу. Если для отладки программы добавлены светодиоды, группа выключателей, то, «нажимая» мышкой выключатели, можно увидеть, как реагируют светодиоды на эти нажатия (если программа поддерживает это). После запуска программы активизируются две следующие кнопки – пауза и стоп.

Отлаживая программу, бывает полезно посмотреть, как меняются переменные. Когда вы «запускаете» программу кнопкой ***Run***, вы можете и не увидеть изменения переменных. Есть два способа посмотреть значения переменных – установить точки останова. Для этого достаточно выделить элемент программы, щелкнуть правой клавишей мышки и выбрать из выпадающего меню **Toggle Breakpoint**.

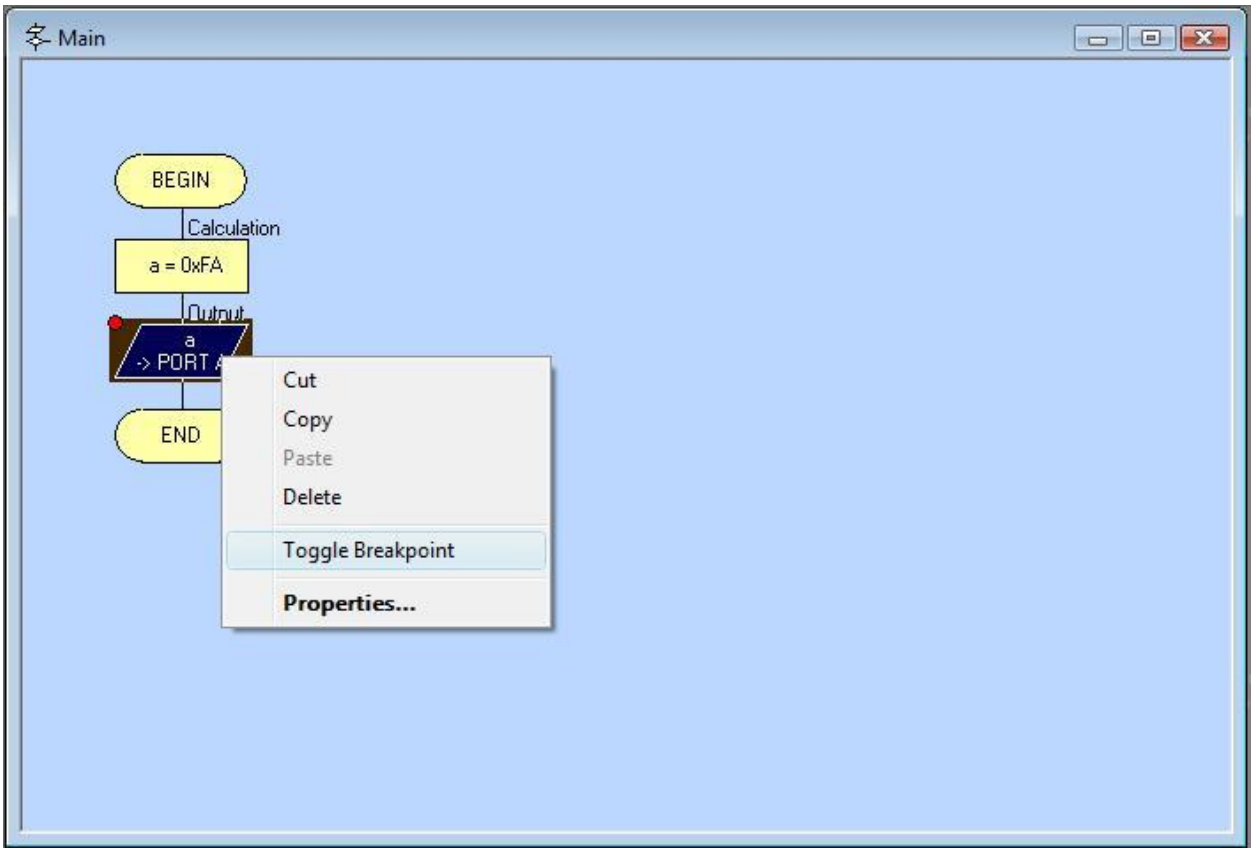


Рис. 1.37. Задание точки останова

Слева от элемента появится точка, точка останова программы, и, когда программа дойдет до этого элемента, отладчик остановит выполнение программы.

Точек останова можно задать несколько в тех местах программы, где вас интересуют значения переменных. Они отображены в окне переменных. Второе окно показывает состояние стека, если в вашей программе есть подпрограммы.

Чтобы удалить точку останова можно повторить операцию задания точки останова. Повторное нажатие на команду **Toggle Breakpoint** снимет точку останова. Чтобы снять все точки останова, достаточно воспользоваться разделом **Edit** основного меню, выбрав пункт **Clear All Breakpoints**.

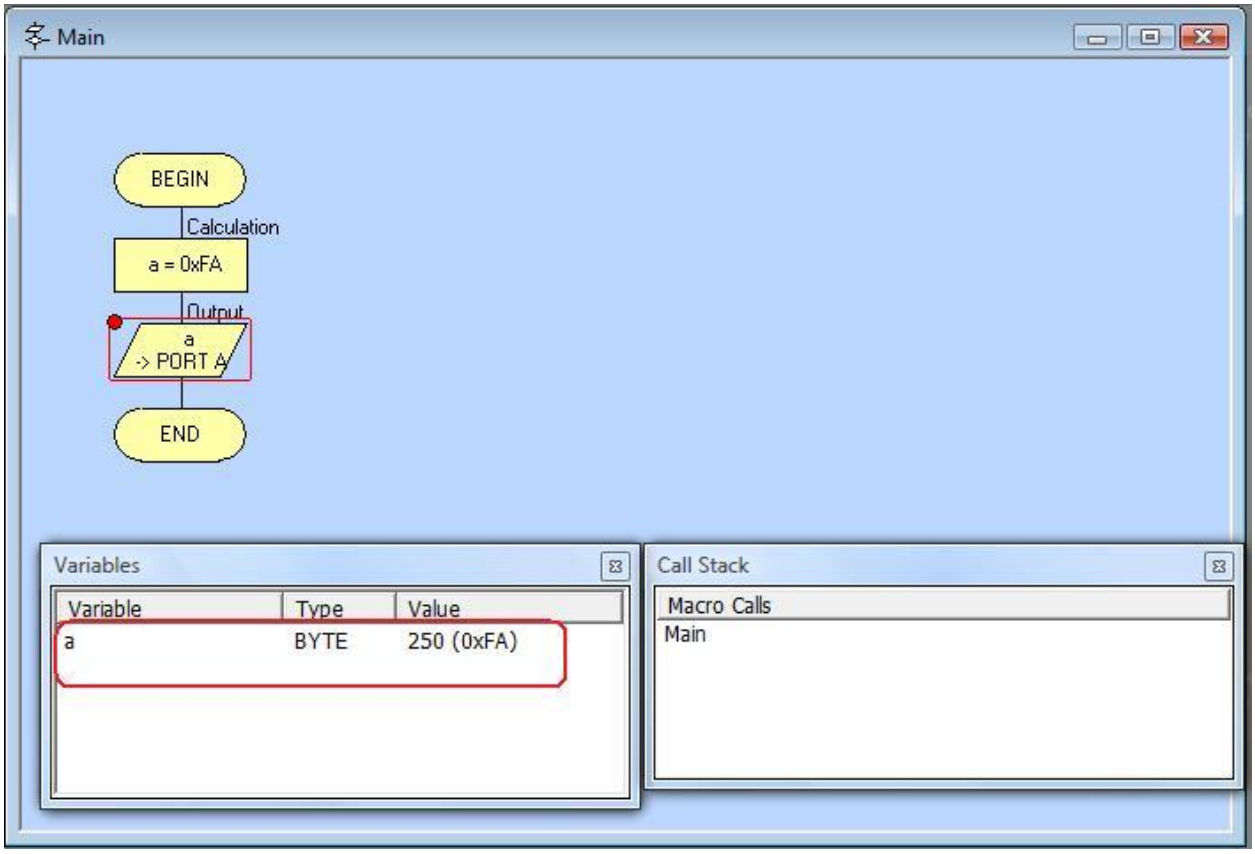


Рис. 1.38. Значение переменной *a*, отображаемое в точке останова

Другой способ увидеть, как меняются переменные, это использовать пошаговый режим отладки. Следом за кнопкой ***Stop*** на инструментальной панели расположена кнопка ***Step Into*** (шаг внутрь). При пошаговом выполнении программы значения переменных отображаются так же, как показано выше. Кнопка ***Step Over*** позволяет «перешагнуть» через выполнение, например, долго выполняемых команд.

Следующая группа относится к трансляции программы.

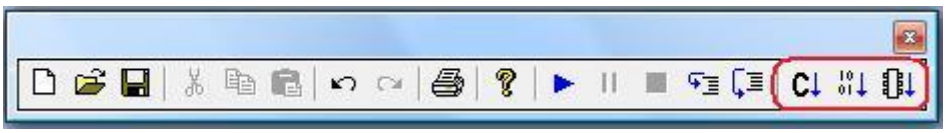


Рис. 1.39. Группа команд трансляции программы

Первая кнопка позволяет транслировать программу в код на языке Си. Полученный код можно увидеть во встроенном текстовом редакторе. Для этого достаточно использовать раздел основного меню **Chip**, где есть пункт **View C**. Если трансляция прошла удачно, то появится сообщение:

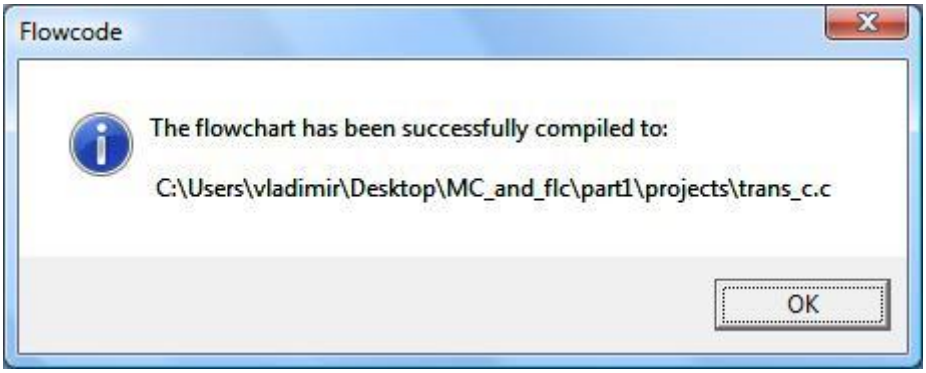


Рис. 1.40. Сообщение об удачной трансляции программы

И полученный код можно открыть, можно копировать и использовать как код на языке Си, о чем мы поговорим позже.

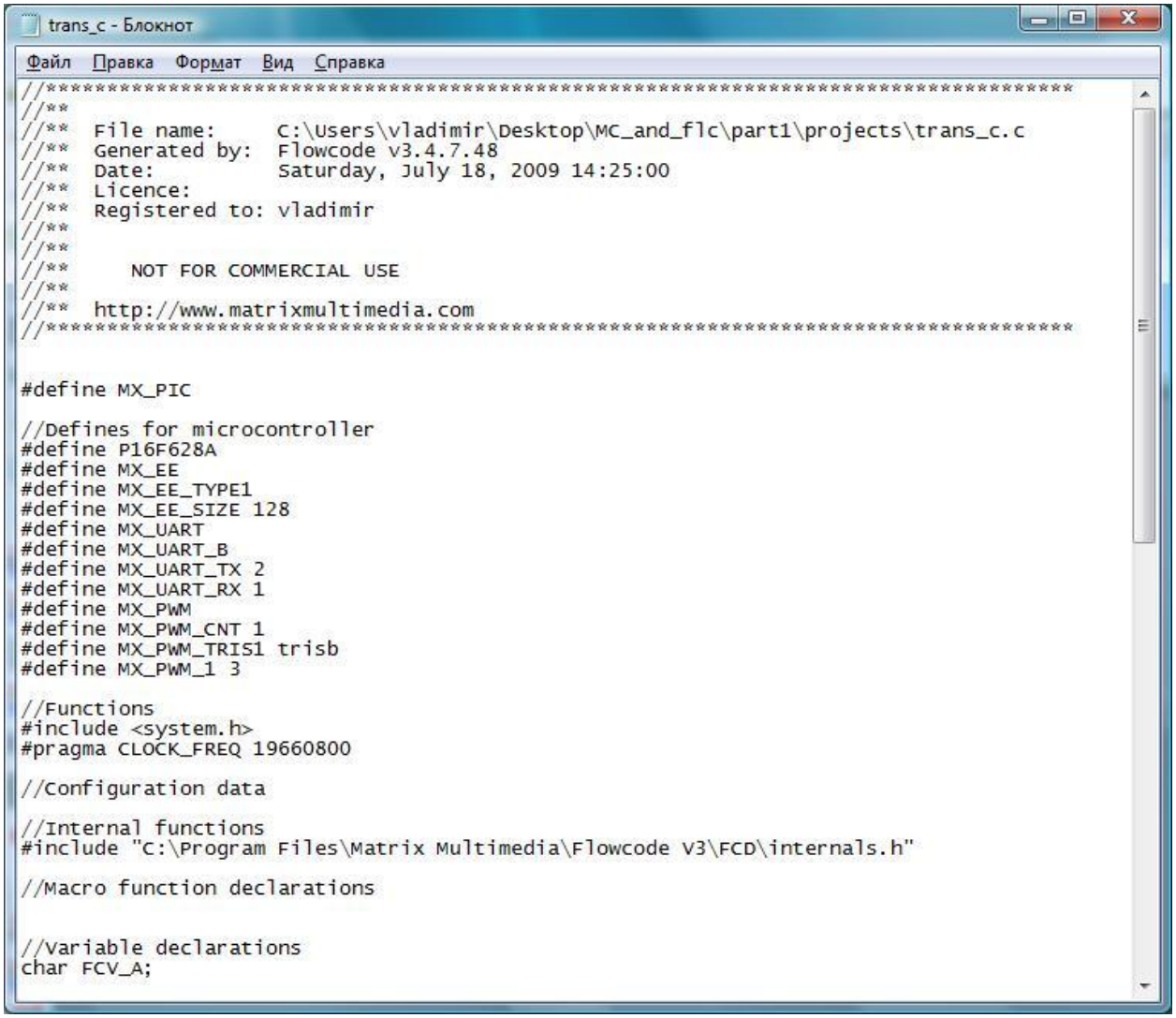


Рис. 1.41. Полученный после трансляции код

Следующая кнопка позволяет транслировать программу в hex-файл для загрузки в микросхему. Если сразу нажать эту кнопку, то программа будет сначала оттранслирована на Си, затем на ассемблер (вы можете посмотреть и этот код, используя раздел **View ASM** пункта **Chip** основного меню), и только после этого будет получен hex-файл.

Последняя кнопка позволяет вам загрузить полученный hex-файл с помощью программатора, который работает с программой, в микросхему. Если у вас нет такого программатора, то вы включите программу обслуживания программатора, который у вас есть, откроете полученный hex-файл, настроите, если это нужно, слово конфигурации (фузы) и загрузите программу в микроконтроллер.

Тот факт, что программа FlowCode создает код на языке Си и ассемблере, может оказать поддержку при изучении этих языков, о чем мы поговорим позже. Все файлы, на языке Си, ассемблере и hex-файл вы найдете в папке, где сохранили свой проект.

Использование программного компонента Calculation

К использованию этого программного компонента приходится прибегать достаточно часто. Вот некоторые операции, поддерживаемые этим компонентом:

( )	- Скобки
= <>	- Равно, НЕ равно
+ - * / MOD	- Прибавить, вычесть, умножить, разделить, модуль
< <= > >=	- Меньше, чем; меньше или равно; больше, чем; больше или равно
>> <<	- Сдвиг вправо, сдвиг влево
NOT AND OR XOR	- NOT(инверсия), И, ИЛИ, Исключающее ИЛИ

и математические функции:

abs(), round(), floor(), ceil()	- абсолютное значение и округление
fround(,)	- округление числа с плавающей точкой
mod(), fmod()	- модуль целого и числа с плавающей точкой
sqrt(), cbrt()	- квадратный и кубический корни
log(), log10()	- логарифмы (с основанием e и 10)
exp(), pow(,)	- функции экспоненты и степенная
sin(), cos(), tan()	- тригонометрические функции
asin(), acos(), atan(), atan2()	- инверсия тригонометрических функций
sinh(), cosh(), tanh()	- гиперболические функции
asinh(), acosh(), atanh()	- инверсия гиперболических функций
fact()	- факториал
rand()	- случайные числа
isnan(), isinf()	- проверка, число ли это, и бесконечности

Увидеть это и многое другое можно в разделе подсказки **Help** основного меню программы FlowCode.

Знакомство с программированием в FlowCode

Простые обучающие программы

Рассмотрим обучающие примеры, которые были получены вместе с программой.

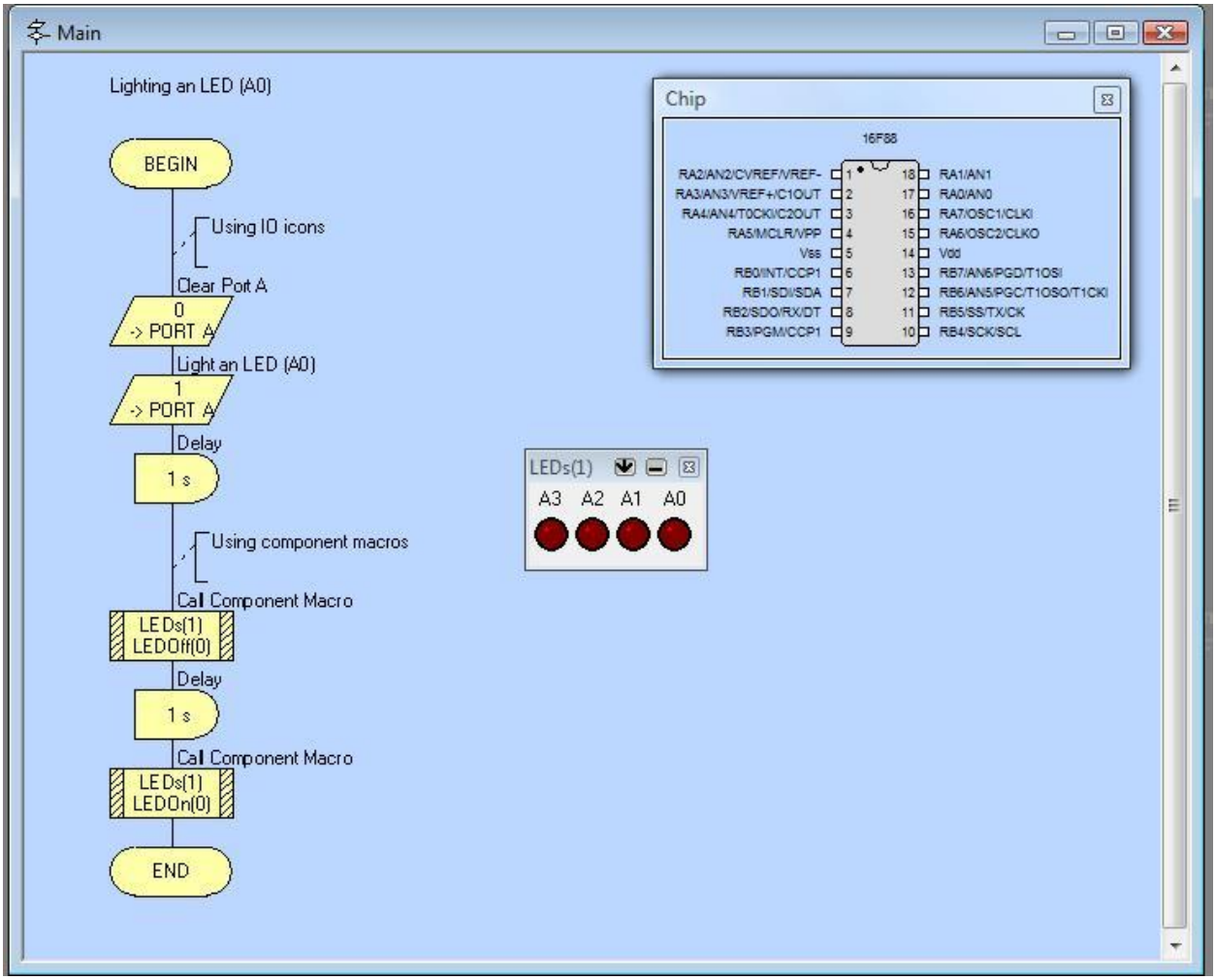


Рис. 2.1. Первый пример из папки Examples

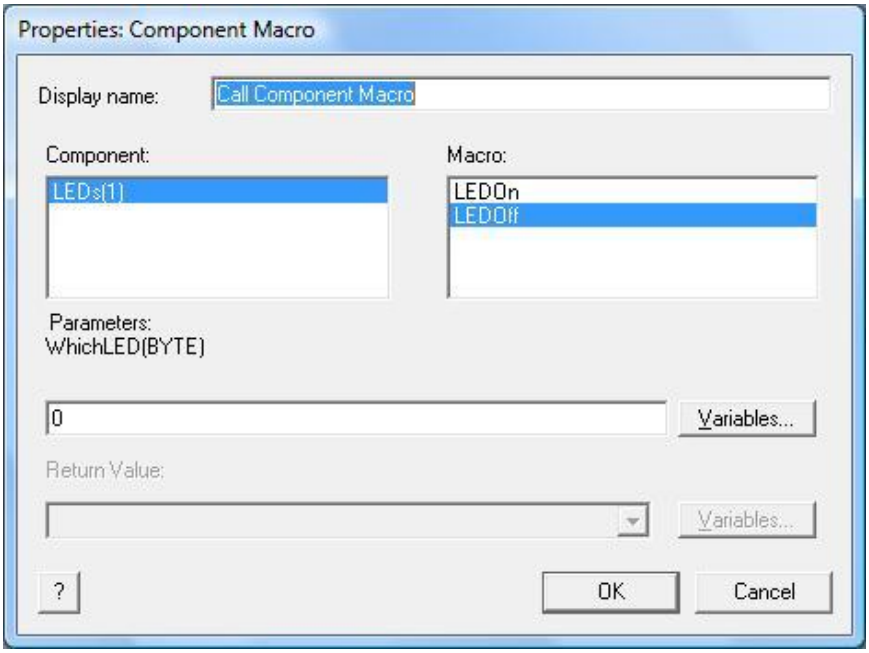
В этом примере для отладки использована линейка светодиодов. Если программные компоненты нужно «подцепить» мышкой и перенести в рабочее поле в нужное место, то для дополнительных компонентов достаточно нажать соответствующую кнопку на инструментальной панели дополнительных компонентов.

Рис. 2.2. Инструментальная панель дополнительных компонентов, отмечены светодиоды

Как в обычном разговорном языке можно написать заявление, а можно написать сонет, так в программе FlowCode можно создать простую программу, а можно очень сложную. При создании

программы можно использовать для одной и той же цели разные программные компоненты, что и показывает первый урок по созданию программы, зажигающей светодиод.

Верхняя часть программы использует для этой цели программный компонент **Output**. А нижняя использует компонент, о котором мы не говорили, когда шла речь об инструментальной панели программных компонентов. Этот элемент называется **Component Macro**. Все элементы панели дополнительных компонентов имеют соответствующие подпрограммы, которые уже написаны, имеют необходимый набор функций и свойств. Связь между дополнительными компонентами, как линейкой светодиодов, и их свойствами осуществляет **Component Macro**. Откройте двойным щелчком по этому компоненту в первом примере диалоговое окно его свойств.



В левом окне отображено, какой компонент, а их может быть несколько, используется.

В правом окне выбор доступных функций. В данном случае *LEDOn* (включить) и *LEDOff* (выключить).

Рис. 2.3. Диалоговое окно свойств линейки светодиодов

В качестве параметра (*Parameters*) вы можете указать, какой из светодиодов следует использовать (на рисунке нулевой).

Обратите внимание, что комментарий (как программный компонент) добавляется к «пути следования» программы. Когда вы, подцепив мышкой программный компонент, перемещаете курсор на рабочее поле, он, курсор, принимает вид того программного компонента, который вы добавляете, а слева появляется стрелка, указывающая в то место программы, куда будет вставлен компонент.

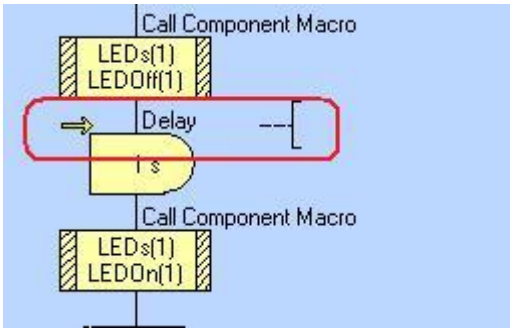
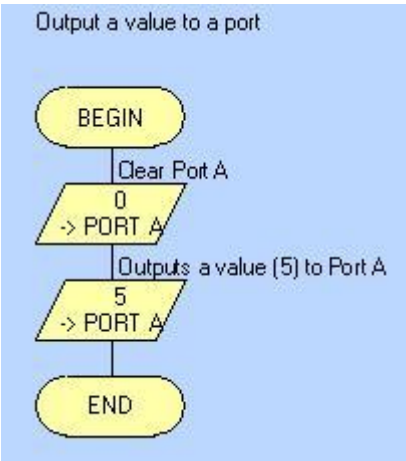


Рис. 2.4. Добавление комментария к программе

Следующий пример (TUT_02) показывает, как вывести число в порт.



В следующем примере, TUT_03, показано, как можно управлять выводами порта «побитово».

Очень полезно открыть диалоговые окна и посмотреть, как организован побитовый вывод, и по шагам пройти всю программу.

Рис. 2.5. Вывод числа 5 в порт A

Пример TUT_05 показывает основные операции: присваивание, сложение, вычитание, деление и умножение, выполняемые с помощью программного компонента **Calculation**.

А пример TUT_06 показывает, как можно «напрямую» связать входы и выходы портов, используя программные компоненты **Input** и **Output** и дополнительные компоненты.

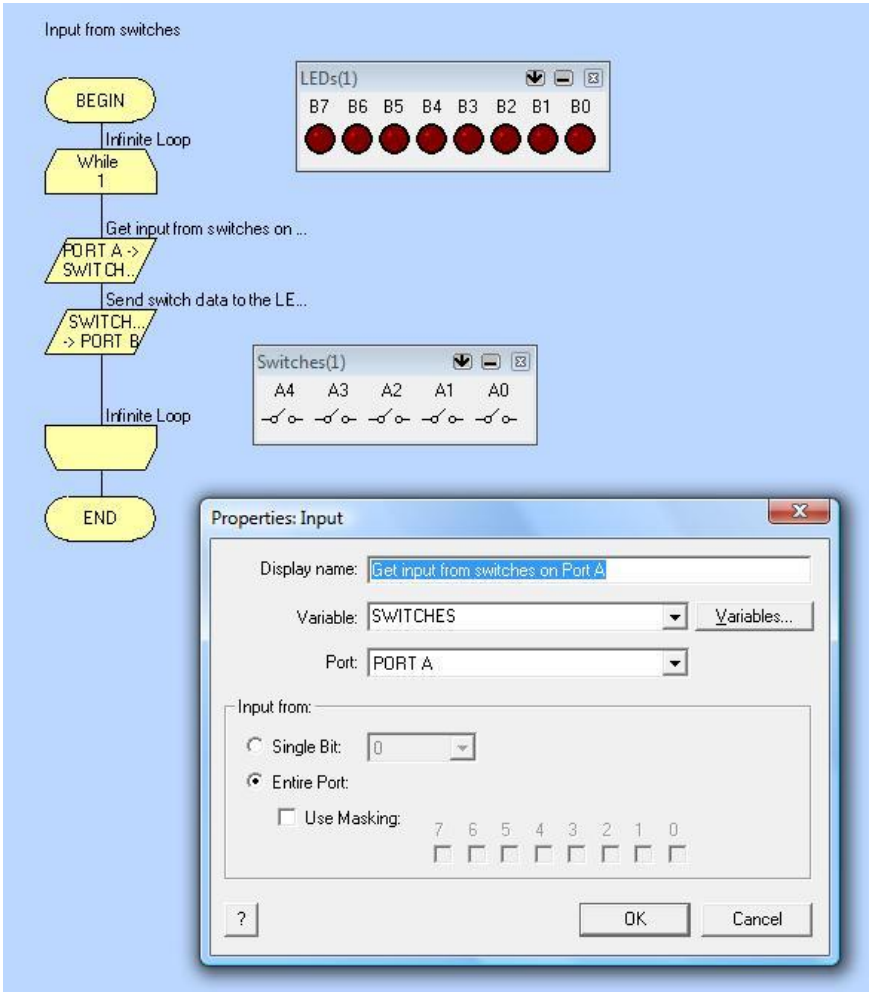


Рис. 2.6. Использование «прямой» связи выключателей и светодиодов

Рис. 2.7. Выключатели на инструментальной панели дополнительных компонентов

Создав одну переменную *SWITCHES*, к которой будет «привязан» компонент **Input**, можно вывести эту переменную, с помощью компонента **Output** в порт.

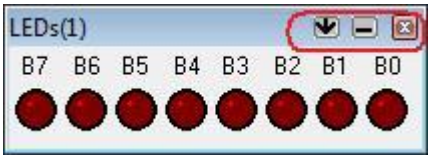


Рис. 2.8. Кнопки управления на панели дополнительных компонентов

Попутно отметим, что компоненты инструментальной панели дополнительных компонентов, имеют три кнопки, первая из которых открывает выпадающее меню, вторая сворачивает компонент, а третья «убирает» его с рабочего поля. Выпадающее меню линейки светодиодов выглядит следующим образом:

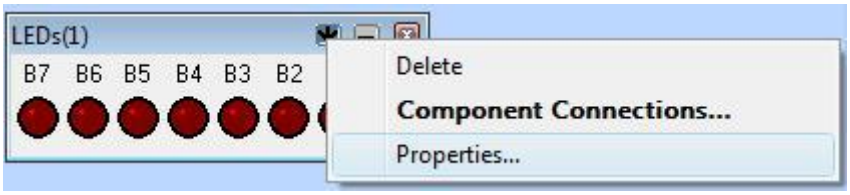


Рис. 2.9. Выпадающее меню управления свойствами дополнительных компонентов

И очень интересно диалоговое окно свойств (*Properties*).

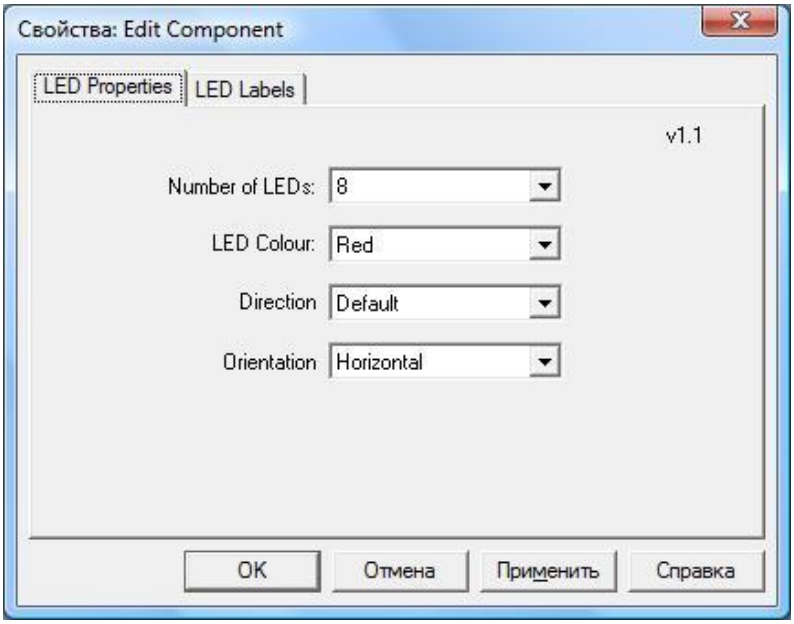


Рис. 2.10. Диалоговое окно свойств линейки светодиодов

В первом окне можно выбрать количество светодиодов, в следующем цвет, дальше можно изменить направление отсчета слева-направо или наоборот, и, наконец, последнее в свойствах, что можно изменить, это ориентация, горизонтальная или вертикальная.

Вторая закладка LED Labels позволяет добавить подписи к элементам.

Если, скажем, вы хотите, чтобы микроконтроллер «покомандовал» двумя светофорами на перекрестке, то можно добавить нужное число линеек светодиодов, выбрать цвет и подпись для каждого и проверить работу светофора.

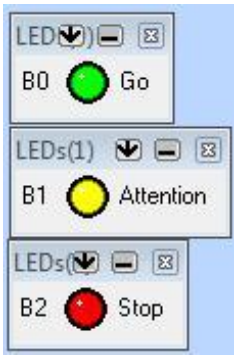


Рис. 2.11. Создание светофора из нескольких линеек светодиодов

Выключатели управляются (нажимаются) мышкой. Иногда, запустив программу, достаточно «пощелкать» выключателем, но бывает так, что мышкой нужно управлять выключателем и еще чем-то одновременно. В этом случае можно изменить свойства выключателей в их диалоговом окне.

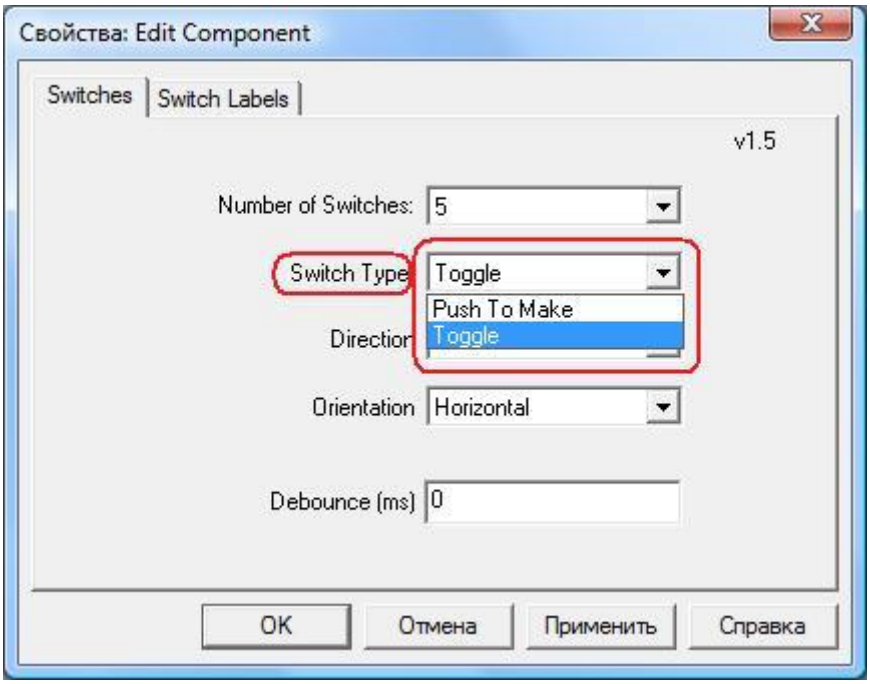


Рис. 2.12. Выбор характера включения выключателей

Пример TUT_08 показывает, как применить маску для выделения нужных входных битов.

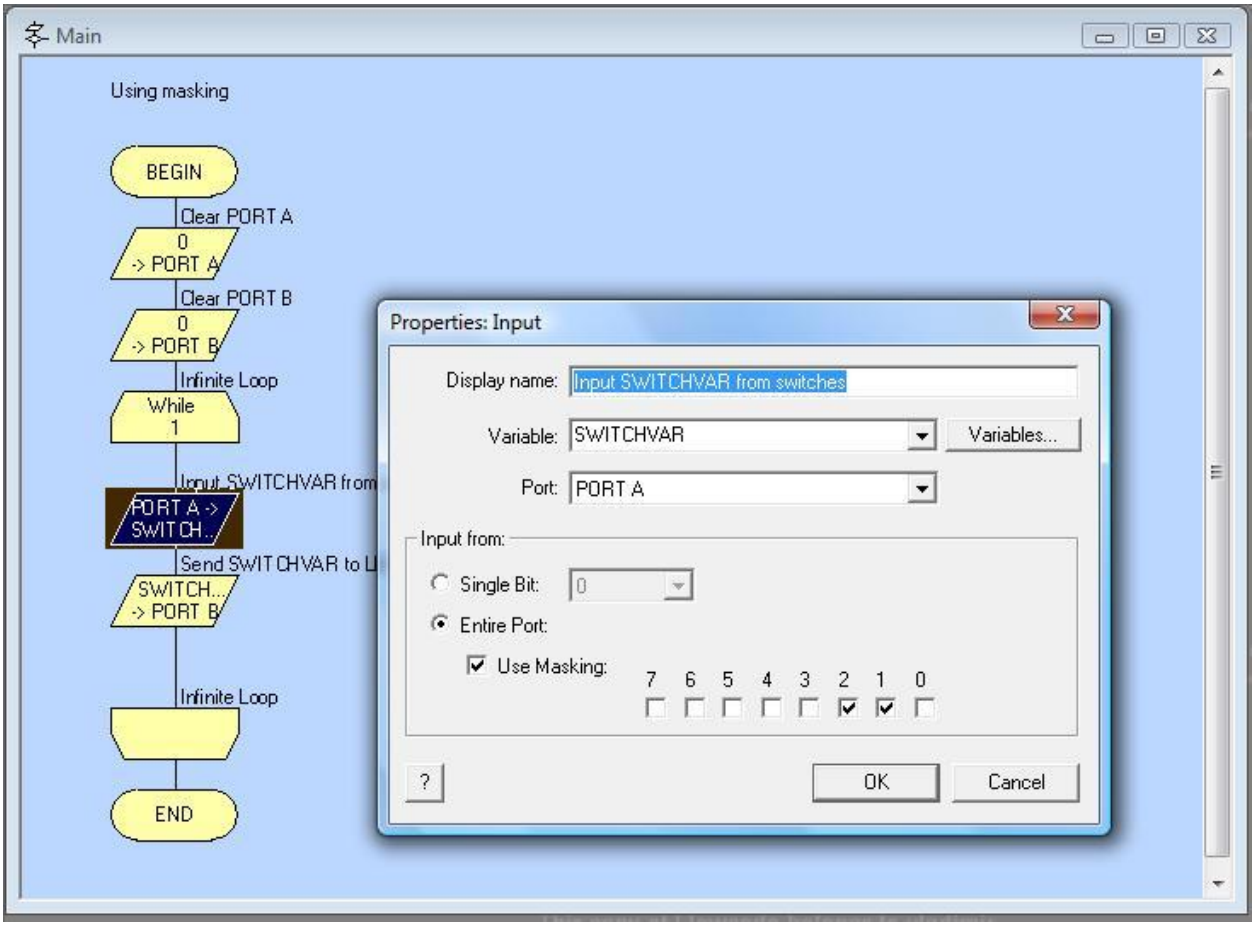
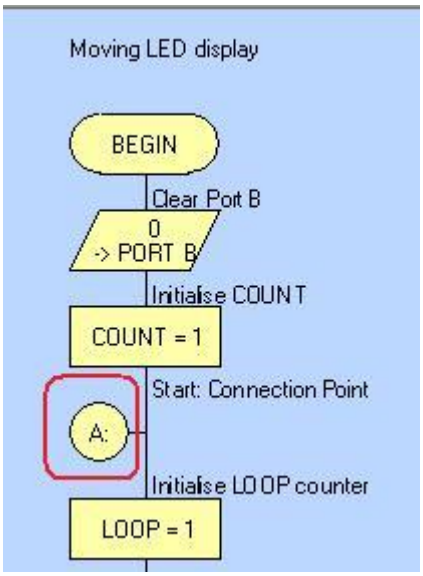


Рис. 2.13. Применение маски

Ряд примеров, показывающих, как можно создать счетчик, как использовать циклы, как применить компонент **Calculation** для сдвига содержимого регистра вправо или влево, дают возможность приобрести навыки программирования, значительно расширяющие кругозор, вдобавок эти программы могут впоследствии использоваться в качестве подпрограмм.

Пример TUT_15 показывает, зачем нужен такой программный компонент, как пара точек **Connection Point**.

Метки в программе, особенно на ассемблере, применяются достаточно часто. Первая из пары точек **Connection Point** и есть метка, а вторая – переход к метке.



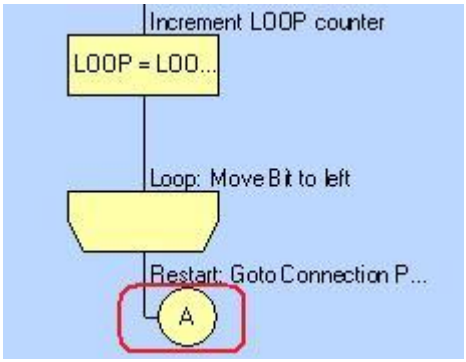


Рис. 2.15. Точки соединения

В диалоговом окне свойств второй точки соединения указывается переход:

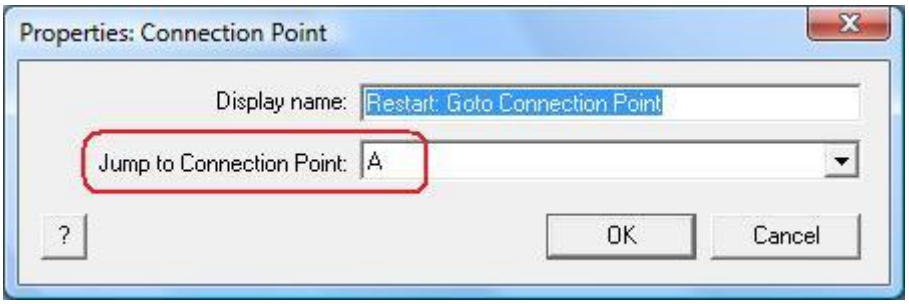


Рис. 2.16. Диалоговое окно свойств точки соединения

Еще один пример полезного использования программного компонента **Component Macro** приведен в программе TUT_18.

Этот компонент используется тогда, когда в программе есть соответствующий дополнительный компонент.

Рис. 2.17. Семисегментный индикатор на инструментальной панели

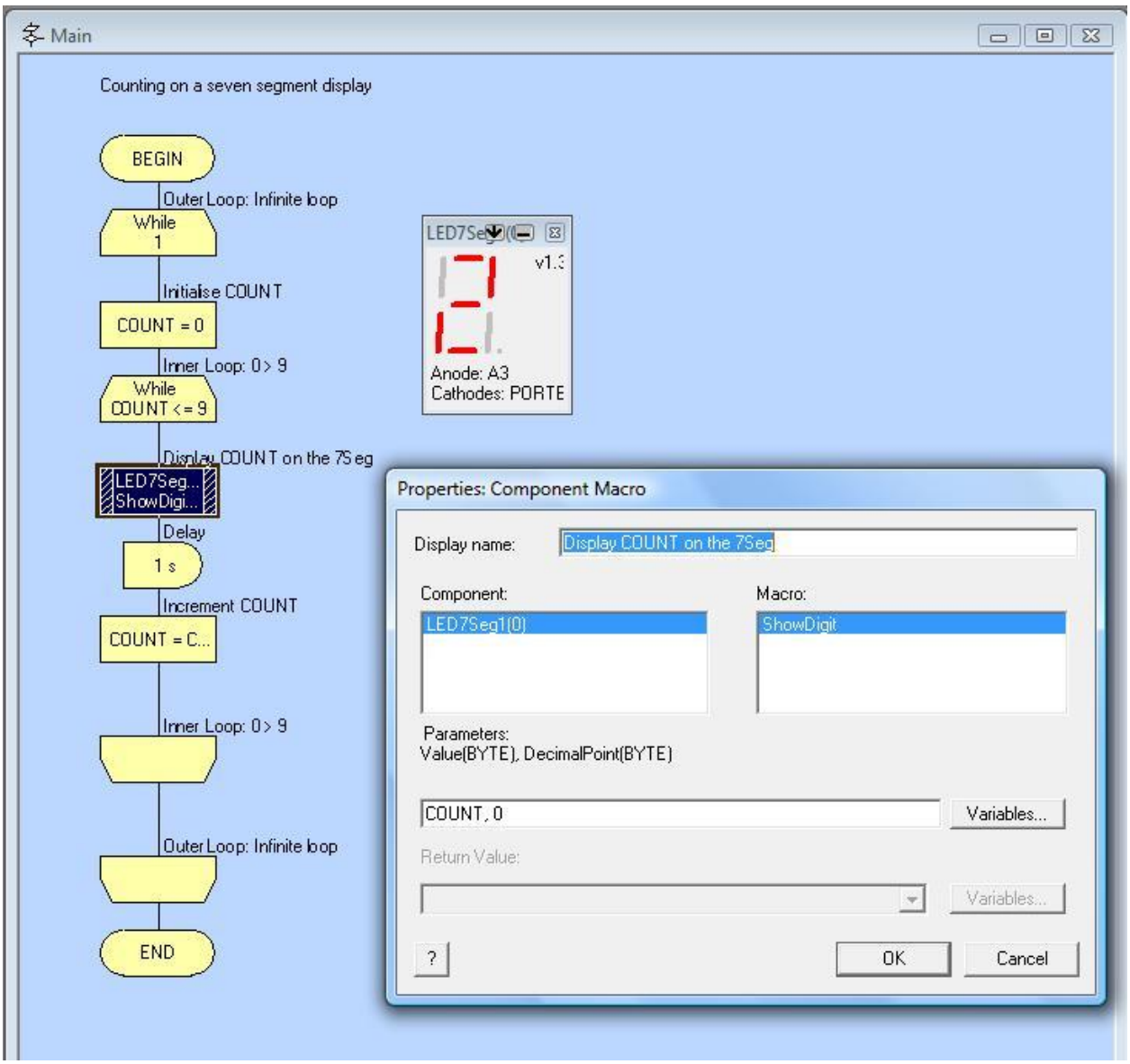


Рис. 2.18. Использование индикатора

В качестве параметра (*Parameters*) используется переменная COUNT. Это позволяет вывести на индикатор любую цифру (от 0 до 9), которое присвоено переменной.

Примеры более сложных приемов программирования

Прерывание используется при программировании достаточно часто. Необходимость в этом элементе программирования возникла в связи с обслуживанием процессором внешних устройств, работающих гораздо медленнее процессора. Обслуживание таких внешних устройств возможно по опросу, когда процессор, передав часть задания, постоянно опрашивает внешнее устройство, готово ли оно продолжить работу, и ждет. Ожидание не позволяет процессору заняться чем-то полезным, в результате бесцельно пропадает процессорное время.

Ситуацию исправляет механизм прерывания. Передав часть задания внешнему устройству, процессор разрешает ему прервать свою работу, когда устройство будет готово продолжить выполнение задания. Пока внешнее устройство «трудится», процессор может заняться выполнением другой задачи. А, получив запрос на прерывание, поместить в стек, специально отведенную для этой цели область памяти, адрес программы, где его застал запрос на прерывание, и данные, с которыми процессор работал. После этого процессор может перейти к подпрограмме обслуживания прерывания, взяв из стека необходимые данные для продолжения работы с внешним устройством.

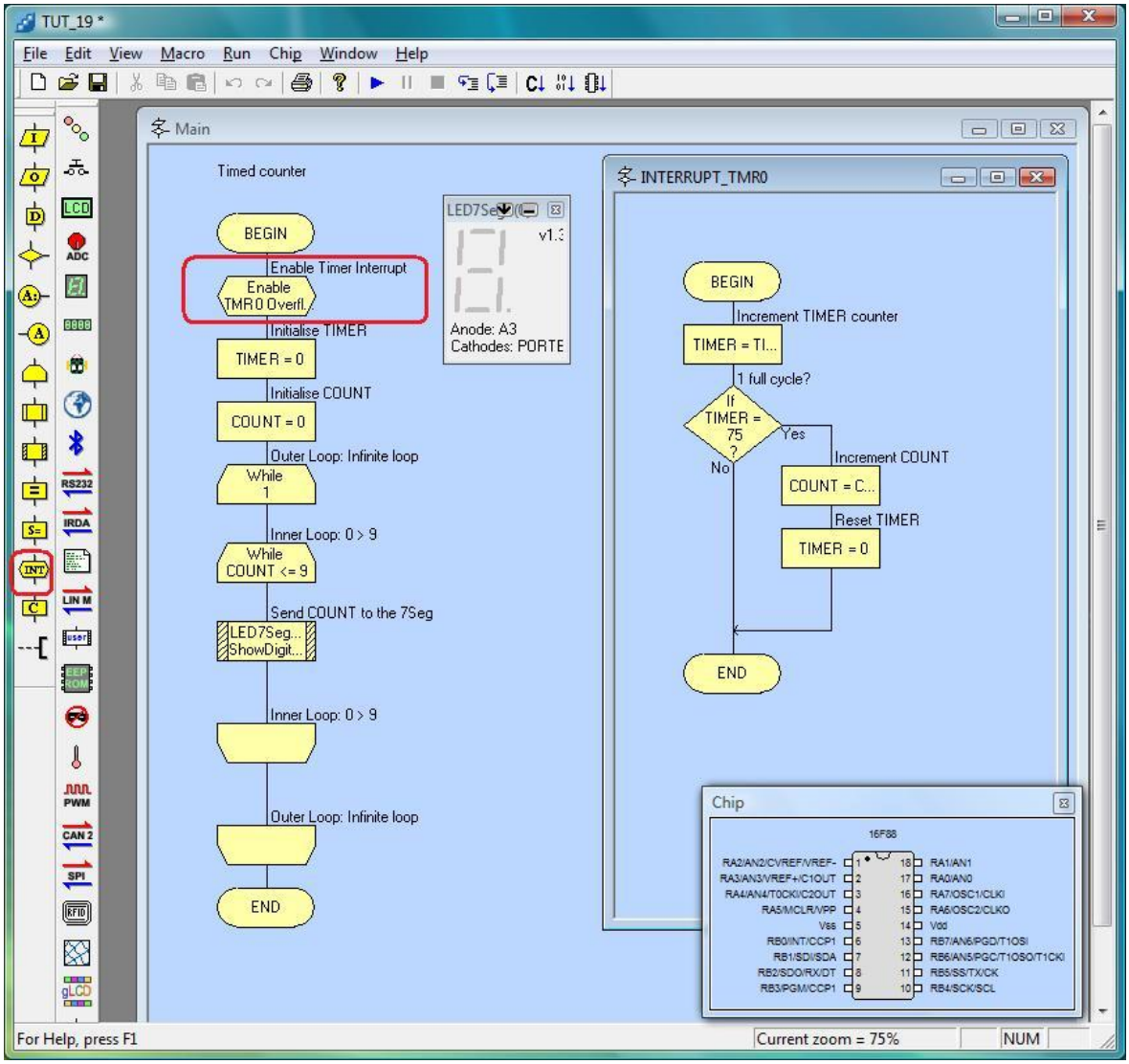


Рис. 2.19. Использование программного компонента **Interrupt** (прерываниее)

На рисунке кроме основного окна программы (Main) видно второе окно подпрограммы INTERRUPT_TMR0. Кроме прерываний по таймеру, как в показанной выше программе, диалоговое окно свойств программного компонента **Interrupt** дает возможность выбрать из выпадающего списка еще несколько видов прерываний. Используемые прерывания могут зависеть от модели микроконтроллера. Так вместо опроса в цикле входных выводов, к которым может быть подключена клавиатура, можно использовать прерывание по изменению этих входов (RB Port Change).

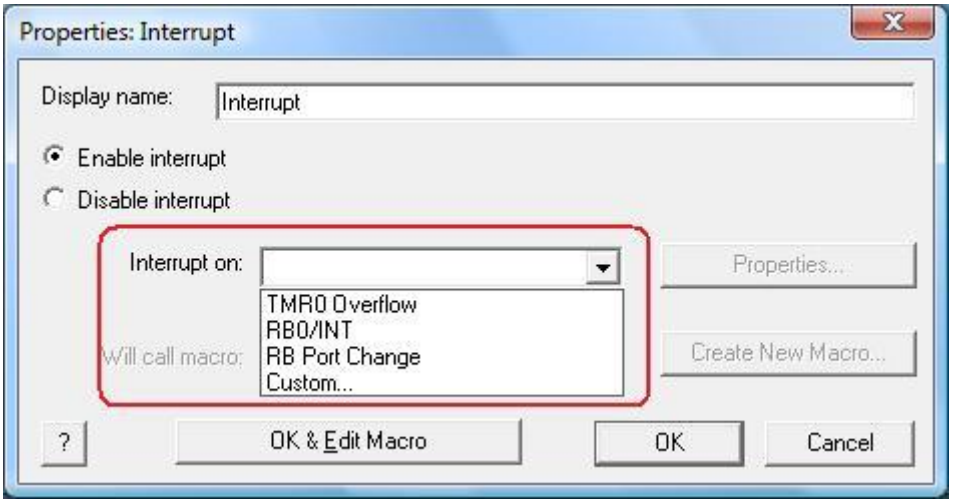


Рис. 2.20. Допустимые виды прерывания для модели PIC16F88

На рисунке выше выбрано прерывание по таймеру *TMR0 Overflow* (переполнение таймера 0). Если нажать кнопку ***Properties*** рядом с окном выбора вида прерывания, то можно задать ряд свойств прерывания. Все свойства выбираются из выпадающих списков.

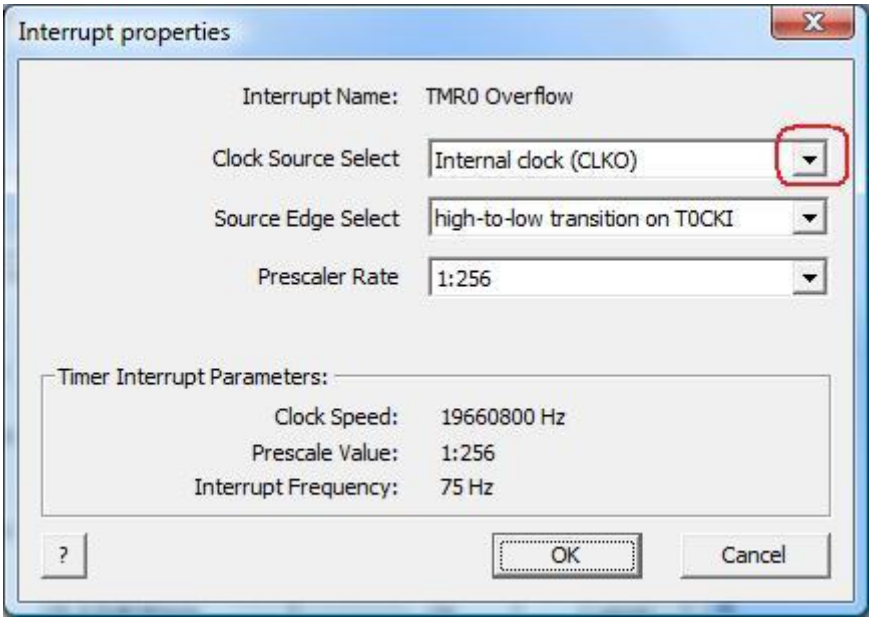


Рис. 2.21. Свойства прерывания, заданные для программы выше

Можно выбрать источник тактового сигнала (Clock Source Select), выбрать каким из двух фронтов, передним или задним, импульса будет инициировано прерывание, выбрать

предварительное деление тактовой частоты для отсчета времени. Ниже показана полученная частота при заданной частоте тактового генератора и выбранного коэффициента деления.

После задания свойств прерывания необходимо написать подпрограмму обслуживания прерывания. Это обычная, в сущности, подпрограмма, и вначале нужно создать макрос с помощью кнопки ***Create New Macro*** (рис. 2.20), затем с помощью кнопки ***OK & Edit Macro*** открыть новое рабочее поле для подпрограммы и начать писать подпрограмму.

Рассмотрим подпрограмму вышеприведенного примера.

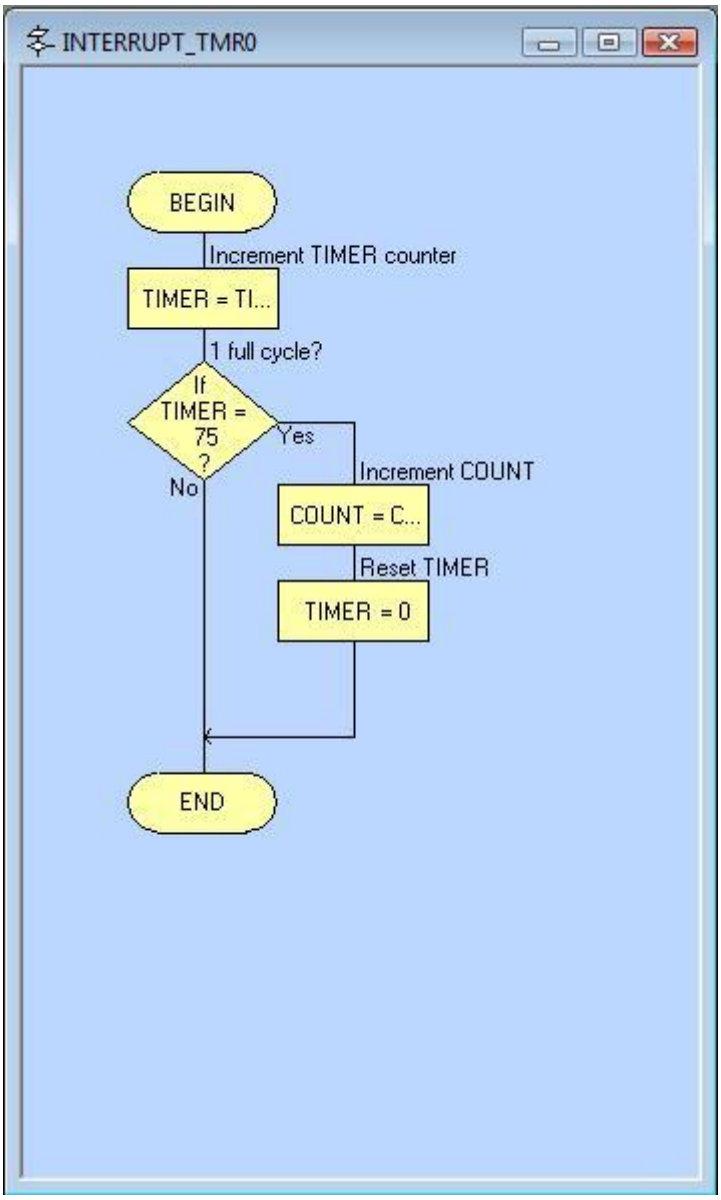


Рис. 2.22. Подпрограмма обслуживания прерывания по таймеру

Откроем диалоговое окно первого элемента программы *Increment TIMER counter* (наращивание счетчика таймера).

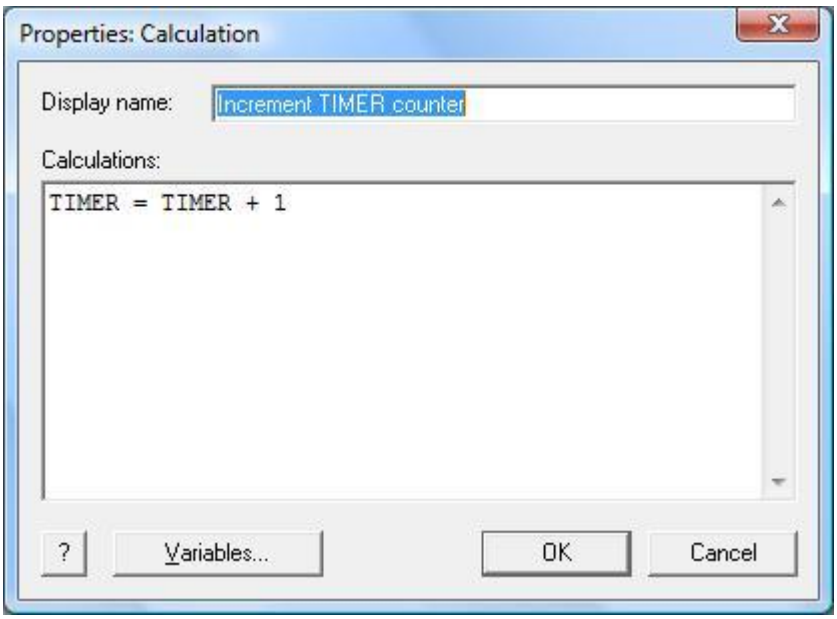


Рис. 2.23. Первый элемент подпрограммы

Как видно, это программный компонент **Calculation**, где значение переменной *TIMER* увеличивается на единицу. Затем следует проверка значения переменной (программный компонент **Decision** – ветвление).

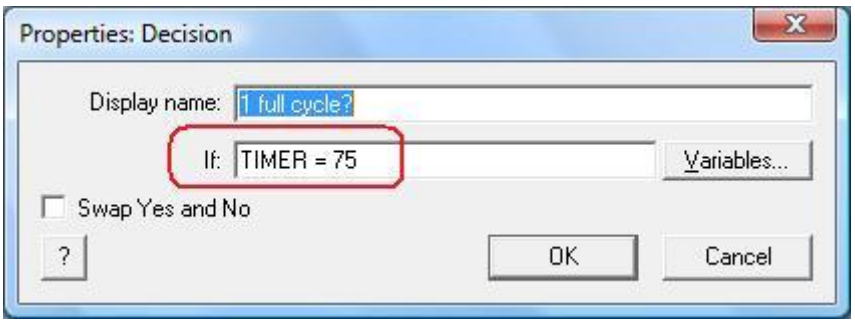


Рис. 2.24. Проверка значения переменной

Если значение достигло заданной величины, то с помощью двух программных компонентов **Calculation** увеличивается на единицу значение переменной *COUNT* и обнуляется переменная *TIMER*, обуславливая завершение одного полного цикла, как и обозначено выше в имени ветвления *1 full cycle*.

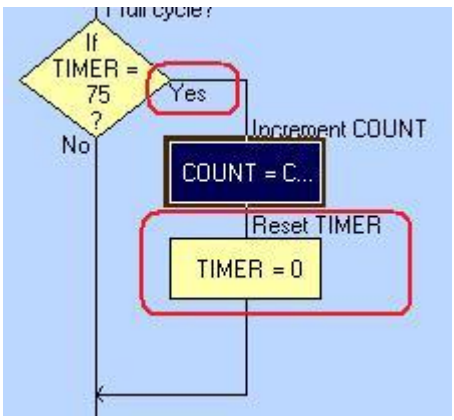


Рис. 2.25. Ветвление программы

Если запустить эту программу, используя кнопку ***Run*** инструментальной панели или команду **Go/Continue** раздела **Run** основного меню, то… можно ничего интересного и не увидеть. Достаточно долго ничего не меняется. Это достаточно часто встречающаяся ситуация при отладке программы. Внесем некоторые изменения (только для отладки): уменьшим значение, скажем, до 5 в условии ветвления подпрограммы. И поставим точку останова в подпрограмме на элемент *Increment TIMER counter*.

Если теперь запустить программу, то она будет останавливаться каждый раз, когда начинается работа подпрограммы. Перезапуская программу (без остановки кнопкой ***Stop*** при каждом прогоне) несколько раз, можно увидеть изменения.

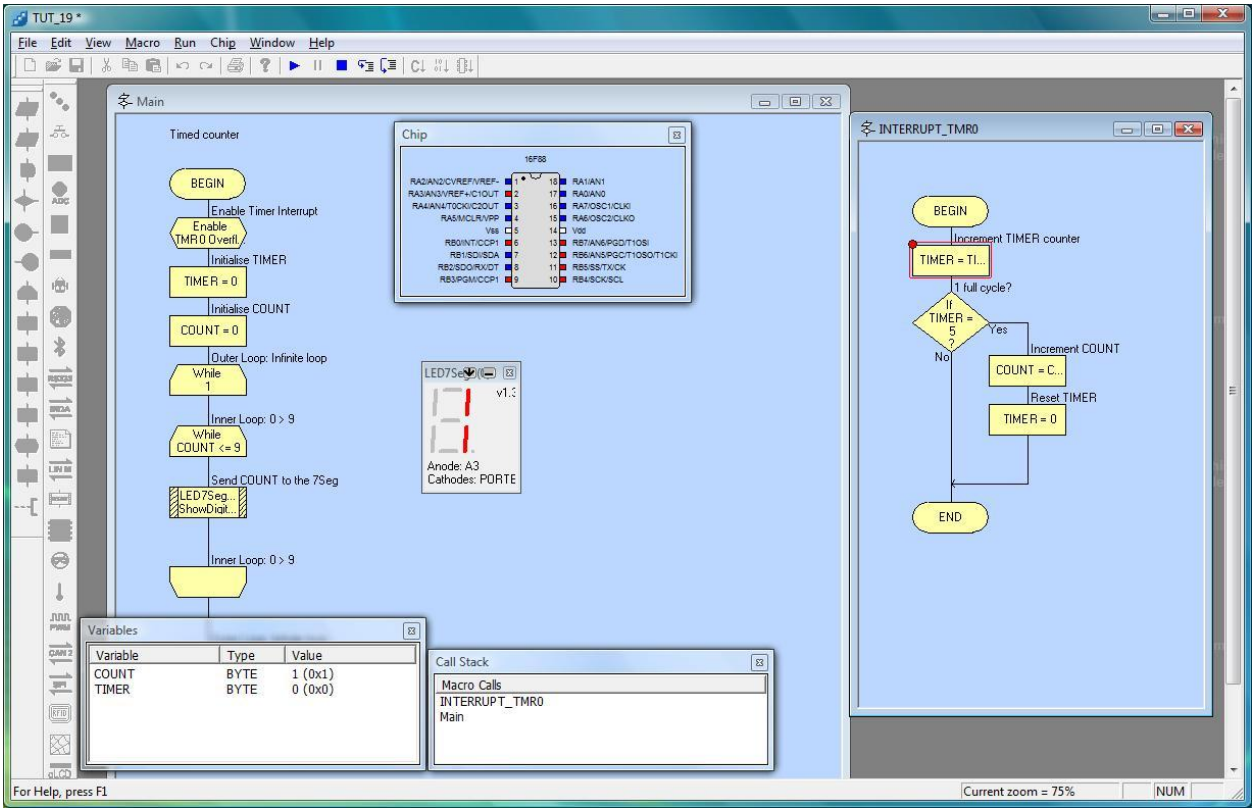


Рис. 2.26. Отладочный запуск программы

Прерывание полезный прием, но не следует забывать, что каждое прерывание использует стек, а стек имеет ограниченное пространство в памяти – обилие не оправданных прерываний может приводить к переполнению стека, вызывая сбой программы.

В примере TUT_20 используется семисегментный индикатор с четырьмя секциями. Действительно, если вам захочется сделать часы или таймер на базе микроконтроллера, то вам нужно будет отображать четыре цифры. Понятно, что когда мы используем одну секцию семисегментного индикатора, выводов порта достаточно для обслуживания индикатора. Но как быть, когда нам нужно в четыре раза больше выводов. Не менять же микроконтроллер.

В первую очередь, где на инструментальной панели дополнительных компонентов находится подобный индикатор?

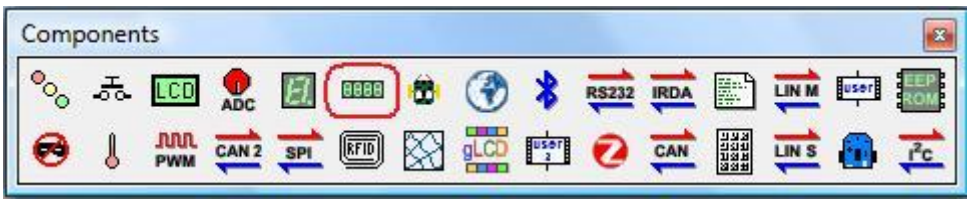


Рис. 2.27. Семисегментный индикатор с четырьмя секциями

Открыв пример TUT_20 обратите внимание на то, как подключен индикатор. Для этого используйте кнопку, открывающую выпадающее меню.

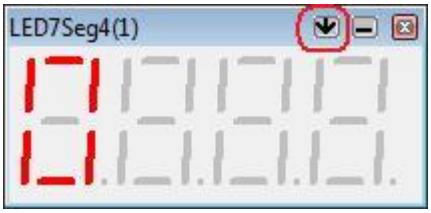


Рис. 2.28. Кнопка выпадающего меню дополнительных компонентов

А в выпадающем меню выберите пункт **Component Connections**.

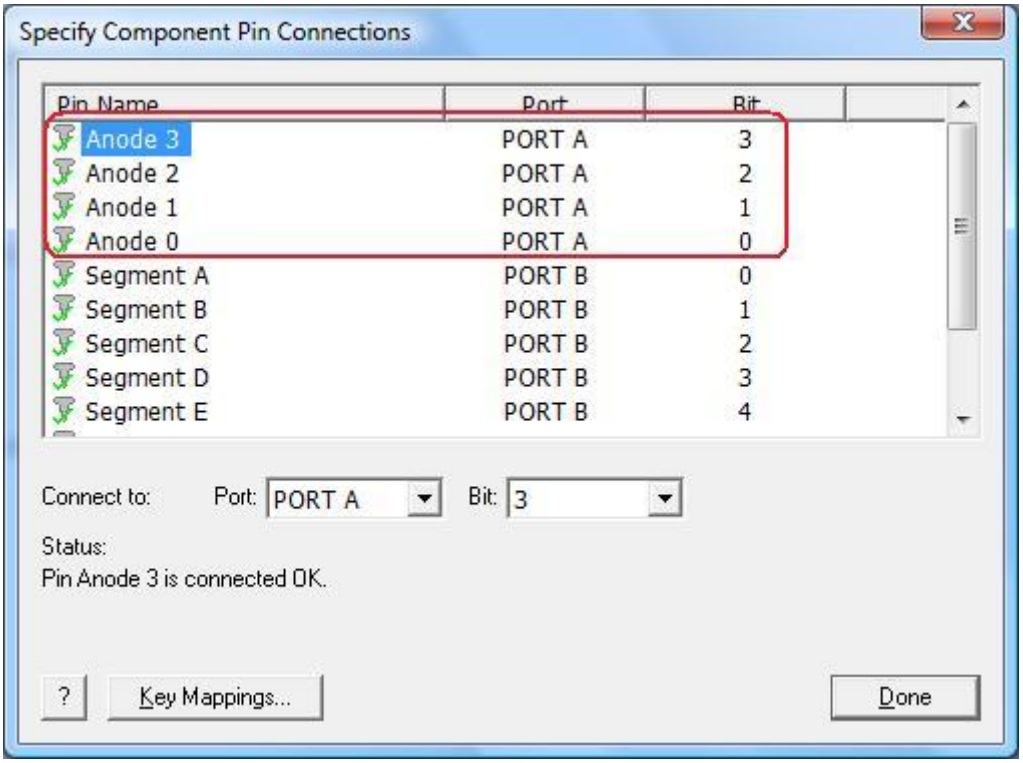


Рис. 2.29. Подключение индикатора к портам контроллера

Таким образом, сегменты всех секций подключены к выводам одного порта, а аноды этих сегментов подключены к выводам другого порта. Пример TUT_20 показывает, как использовать динамическую индикацию.

Развитие индикаторов со временем привело к появлению не слишком дорогих даже для любителей жидкокристаллических дисплеев. Теперь микроконтроллер, снабженный подобным

дисплеем, может стать основой целой серии полезных устройств. На дисплей можно вывести, например, фразу. Как это сделать, показано в примере TUT_21.

Найти дисплей можно на инструментальной панели дополнительных компонентов.

Рис. 2.30. Дисплей на инструментальной панели

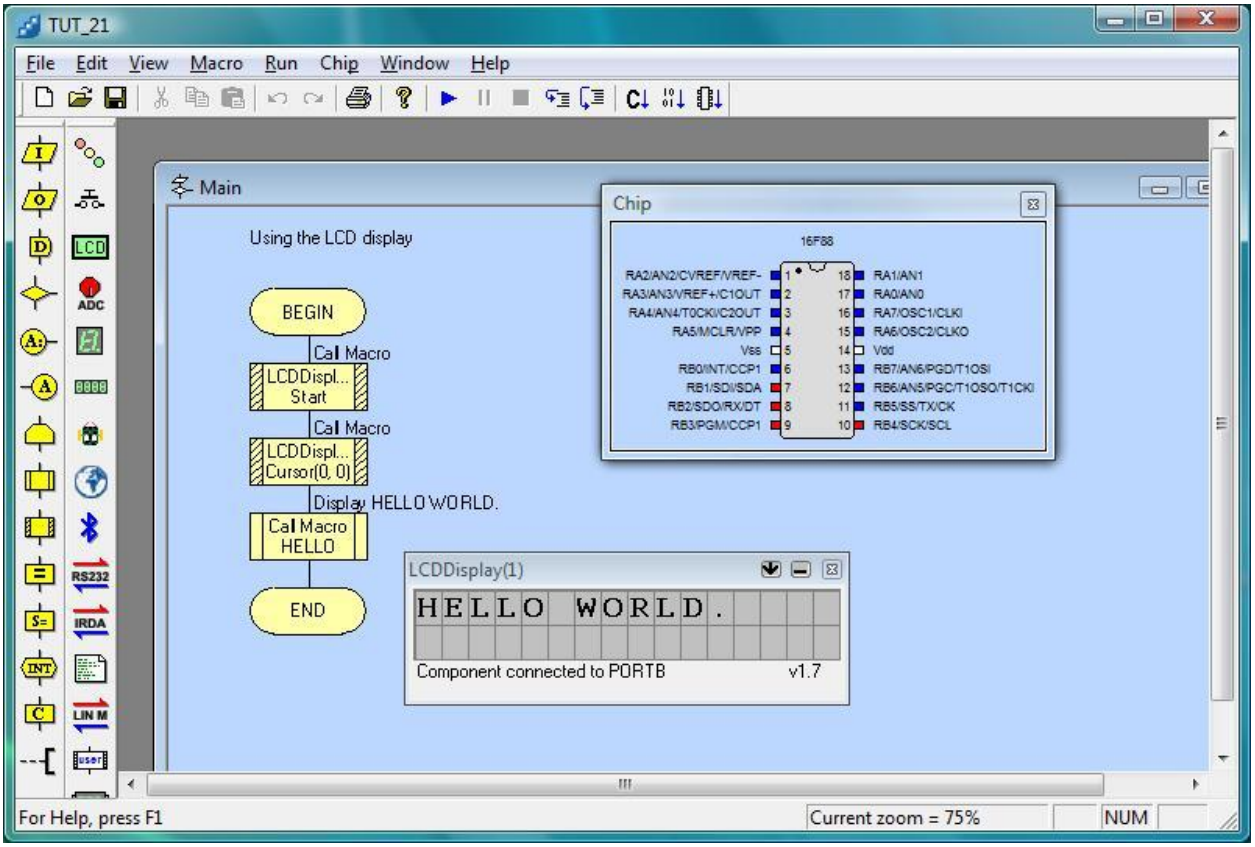


Рис. 2.31. Вывод фразы на дисплей

Если вы откроете свойства дисплея, то можете убедиться, что это не единственный дисплей, доступный для ваших экспериментов. Это семейство дисплеев. И обратите внимание на подключение дисплея.

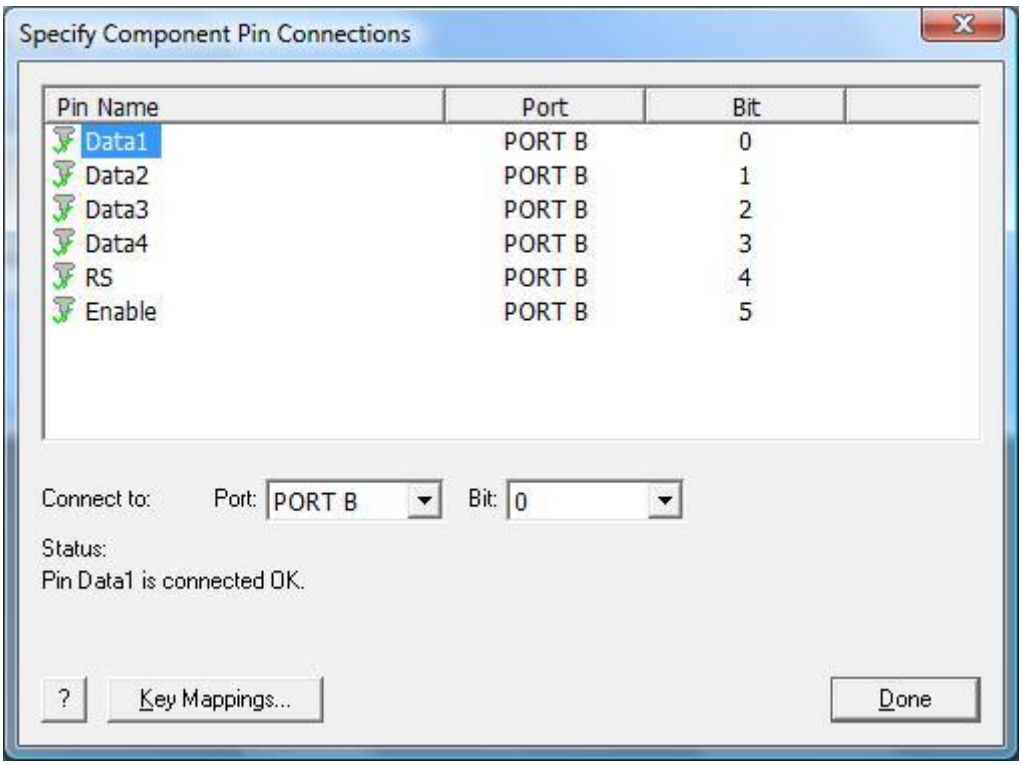


Рис. 2.32. Подключение дисплея к порту контроллера

В программе несколько раз используется макрос, связанный с этим элементом – **Component Macro**. Если вы откроете любой из них, то увидите, сколько встроенных в эту подпрограмму функций есть в вашем распоряжении (для этого нужно щелчком выделить название компонента).

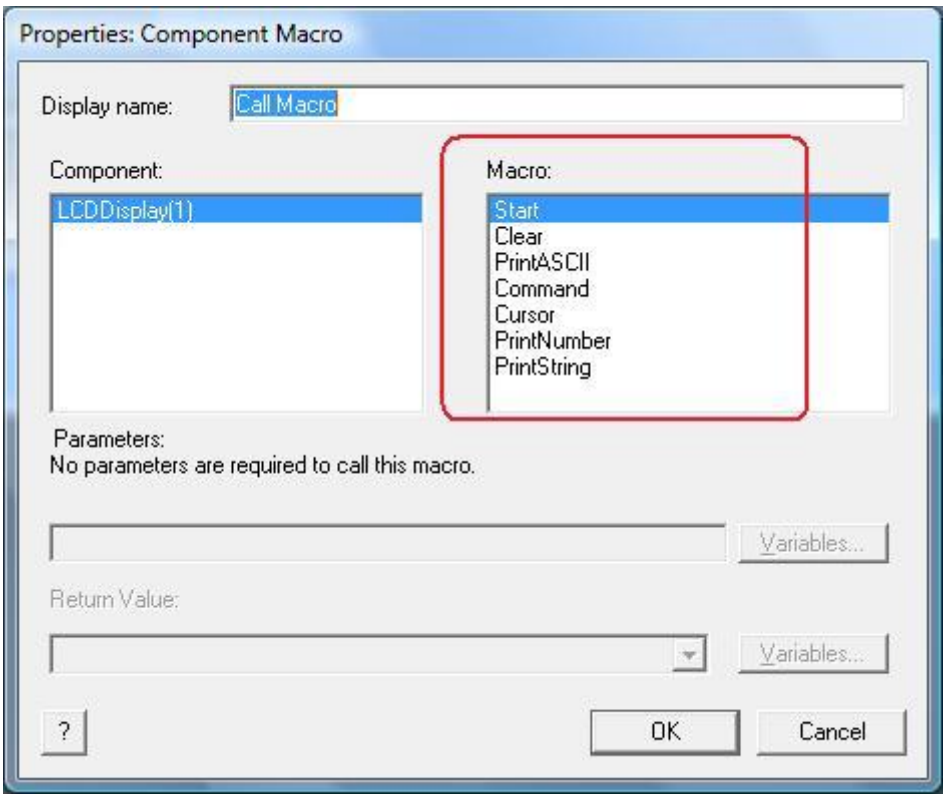


Рис. 2.33. Доступные функции компонента LCDDisplay

Если ваш микроконтроллер имеет встроенный аналого-цифровой преобразователь, АЦП, то вы можете использовать его, совместно с дисплеем, для создания целого ряда автоматических станций, которые могут следить за температурой или влажностью почвы.

Рис. 2.34. АЦП на инструментальной панели

Дополнительный компонент ACD позволяет проверить работу встроенного АЦП, при этом ручка, изображенная на иконке, вращается, если ее «подцепить» мышкой.

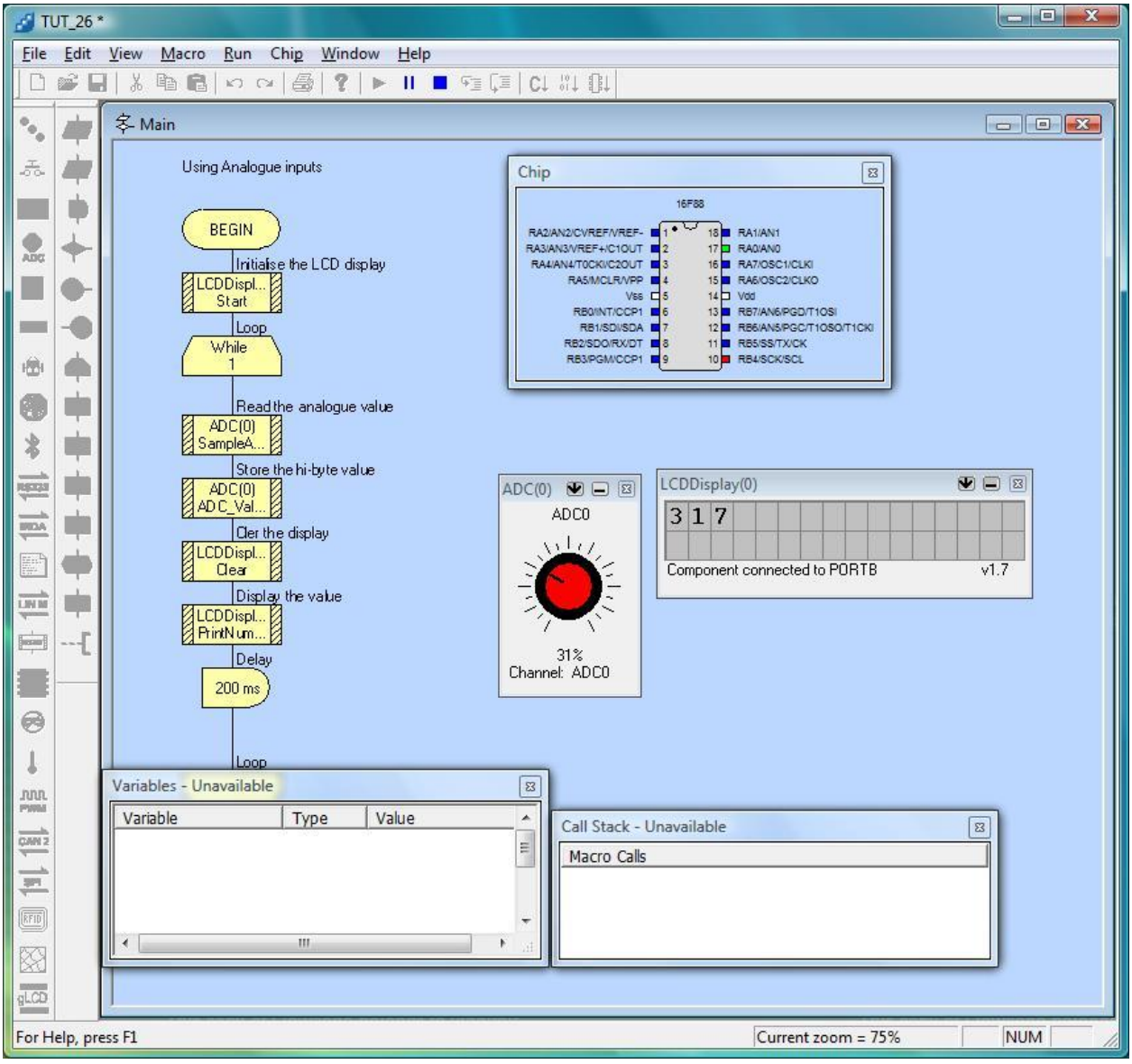


Рис. 2.35. TUT_26 – пример использования АЦП

При этом вы можете использовать программный компонент String Manipulation для создания строк, которые можно вывести на дисплей.

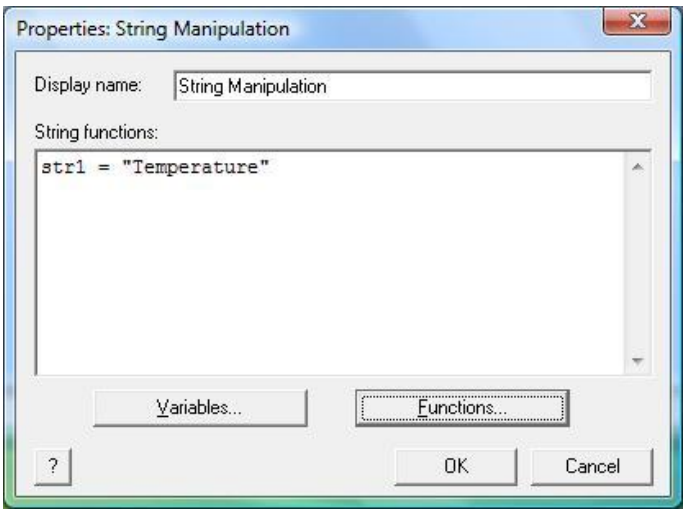


Рис. 2.36. Использование программного компонента String Manipulation

Этот программный компонент имеет ряд встроенных функций для работы со строковыми переменными, которые отображаются, если нажать кнопку ***Functions***. Добавив к программе показанной выше еще несколько программных компонентов, вы можете получить, например, следующее:

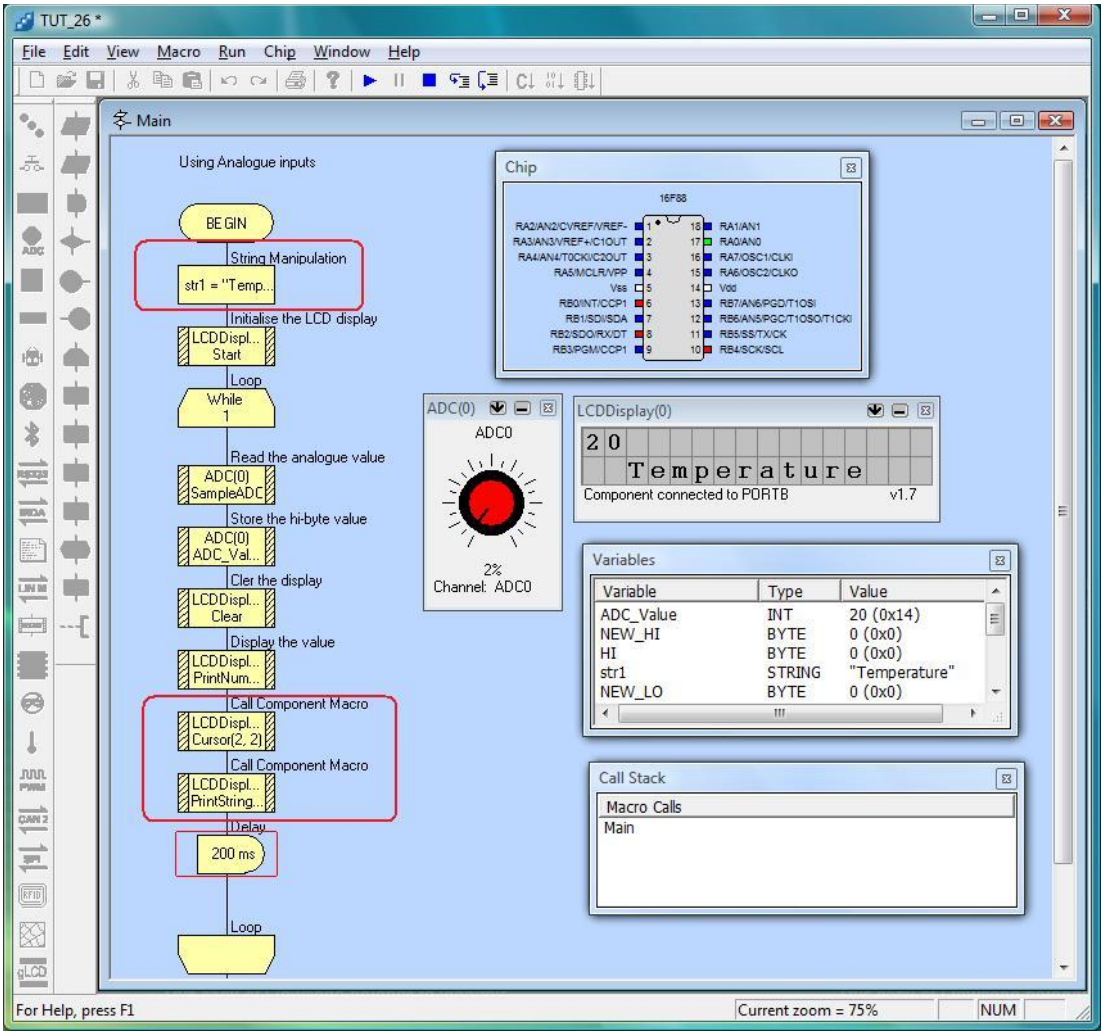


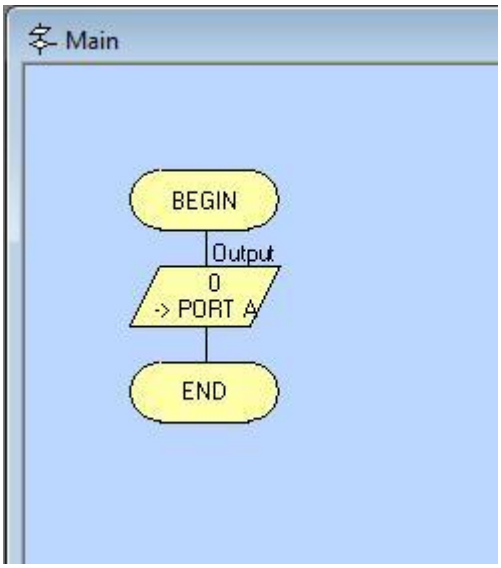
Рис. 2.37. Модификация программы TUT_26

Переход к программированию на языке Си

Программа, если вы ее купите, возможно, полностью удовлетворит все ваши нужды в работе с микроконтроллерами. Но, используя демо-версию, вы очень быстро столкнетесь с ограничениями. Да и ощущение, что не все в ваших руках, не все возможности вы используете, это можно отнести к таким программным компонентам FlowCode, как вставки на Си и ассемблере, должно подвигнуть вас на изучение языка Си и ассемблера. И программа FlowCode хороший помощник в этом.

Первые шаги

Сейчас нам нужен только один программный элемент – **Output**. «Подцепив» его мышкой, перенесем между элементами *BEGIN-END*. И ничего с ним делать не будем. Таким образом, мы все выводы порта А включаем «на выход» и в низкое состояние, то есть в «0».



Оттранслируем программу на язык Си: для этого в программе FlowCode в основном меню выбираем раздел **Chip** и пункт **Compile to C**.

Мы можем посмотреть полученный результат: **Chip-> View C**.

Рис. 3.1. Простейшая программа в FlowCode

Программа, которую мы открыли во встроенном редакторе, уже достаточно длинная (для первого шага). Но нас интересует только основная ее часть:

```
void main()
{
//Initialisation
cmcon = 0x07;
//Output: 0 -> PORT A
trisa = 0x00;
porta = 0;
}
```

main – это начало программы на языке Си. Сама программа находится в фигурных скобках {...}. Каждый оператор завершается точкой с запятой. *void* перед *main* означает, что функция не возвращает значения. Я привел только основную часть программы, удалив все директивы компилятору и часть, относящуюся к прерываниям, их отложим «в долгий ящик».

Мы получили код на языке Си для программы, которая устанавливает выводы порта в «0». Модифицируем эту программу так, чтобы: устанавливать не весь порт, а один вывод порта, и не в «0», а в «1».

Для этого двойным щелчком по программному компоненту **Output** откроем его диалоговое окно:

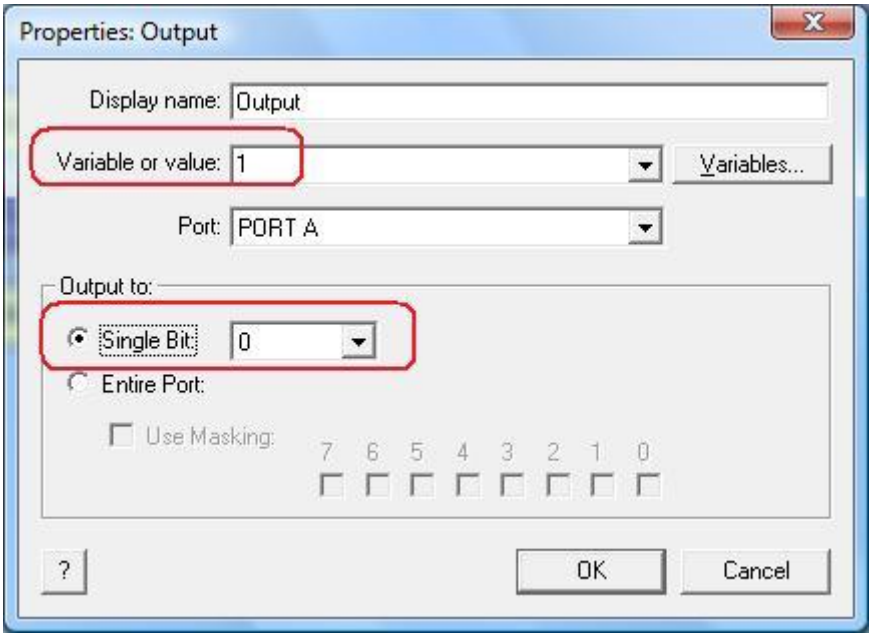


Рис. 3.2. Диалоговое окно программного компонента **Output**

На рисунке отмечены внесенные изменения: вместо 0 в окошке *Variable or value* (переменная или значение) вписана 1; вместо *Entire Port* (весь порт) выбрана опция *Single Bit* (единственный бит) и оставлен вывод 0 порта А (хотя можно было изменить и порт, чуть выше). Нажмем кнопку ***ОК*** и сохраним проект в новой папке с новым именем, чтобы не путать с прежним.

Повторим операцию трансляции на язык Си: **Chip->Compile to C...**

Так же выделим только нужную часть кода:

```
void main()
{
//Initialisation
cmcon = 0x07;

//Output: 1 -> A0
trisa = trisa & 0xfe;
if (1)
porta = (porta & 0xfe) | 0x01;
else
porta = porta & 0xfe;
}
```

Вот так выглядит установка одного вывода порта А в «1». Если повторить все установки для одного вывода, но с установкой его в «0» (а не всего порта, как в самом начале), то текст программы будет выглядеть следующим образом:

```
void main()
{
//Initialisation
cmcon = 0x07;

//Output: 0 -> A0
trisa = trisa & 0xfe;
if (0)
porta = (porta & 0xfe) | 0x01;
else
porta = porta & 0xfe;
}
```

В итоге мы располагаем тремя фрагментами текста на языке Си, которые делают базовые для микроконтроллера операции – устанавливают выбранный нами вывод порта в «0» и в «1» или устанавливают весь порт в «0».

Проверим еще один вариант: установим в «1» не нулевой вывод порта А, а первый.

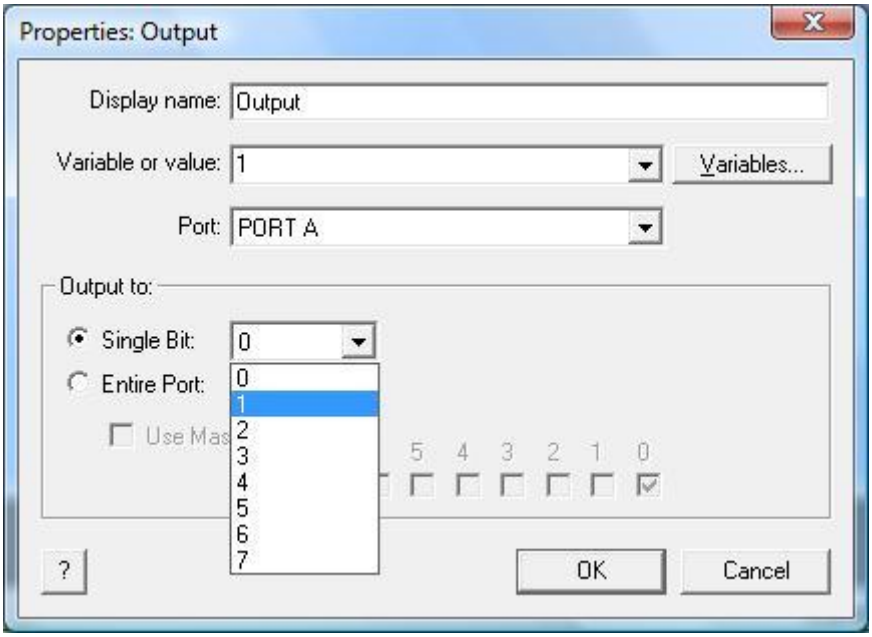
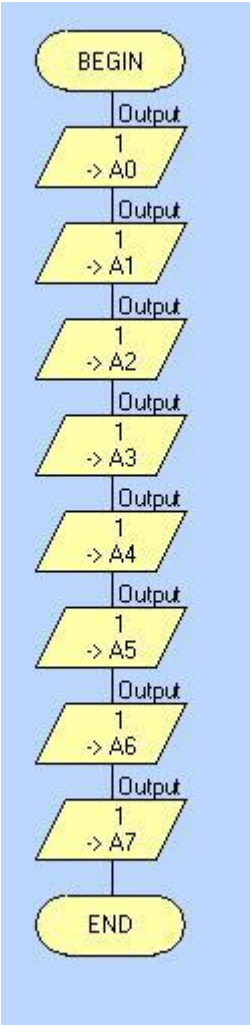


Рис. 3.3. Установка в «1» первого вывода порта А

Посмотрим, как изменится текст программы на языке Си (лишнее опять отбросим):

```
void main()
{
//Initialisation
cmcon = 0x07;

//Output: 1 -> A1
trisa = trisa & 0xfd;
if (1)
porta = (porta & 0xfd) | 0x02;
else
porta = porta & 0xfd;
}
```

На некоторое время оставим «заготовки» на языке Си и вернемся к программе FlowCode. Создадим такую программу для микроконтроллера: последовательно все выводы порта А устанавливаются в «1». В FlowCode программа выглядит так:



Заметьте, что выводы в программных элементах Output меняют свой номер: A0, A1 и т.д.

Эту программу можно проверить в отладчике программы FlowCode, используя линейку светодиодов.

Если теперь нажать кнопку Run основной инструментальной панели, то программа запустится на выполнение. Но ничего интересного мы не увидим – все светодиоды загорятся, и все. Микроконтроллер работает очень быстро, не успеешь глазом моргнуть. И чтобы посмотреть, что происходит в программе, используем кнопку пошагового управления отладкой.

Нажимая эту кнопку, мы увидим, как последовательно зажигаются светодиоды.

Рис. 3.4. Программа последовательной установки всех выводов порта

Но нас сейчас интересует не проверка работы программы (или микроконтроллера), а работа с кодом на языке Си. У нас есть нужные фрагменты программы, постараемся разобраться в них с тем, чтобы воспроизвести последовательное включение выводов порта A на языке Си. Мы можем использовать самый первый фрагмент текста, напомню его:

```
void main()
{
//Initialisation
cmcon = 0x07;
//Output: 0 -> PORT A
trisa = 0x00;
porta = 0;
}
```

В этом фрагменте мы использовали весь порта A для вывода, что, похоже, отображено оператором `trisa = 0x00;` поскольку в следующих фрагментах этот оператор выглядит иначе: с регистром, в котором записывается назначение выводов, проделывается логическая операция `trisa = trisa & 0xfe;`

Если заглянуть в описание микроконтроллера (datasheet), то выяснится, что запись всех нулей в регистр TRISA означает, что все выводы порта A предназначаются на выход. То есть, тот вывод, что для выхода, должен здесь прописываться, как «0», тот, что для входа, как «1». В нашей программе мы можем записать в регистр TRISA все нули, как в первом случае.

Следующий оператор записывает 0 в порт, что переводит все выходы в низкое состояние. Но, если мы изменим оператор, написав `porta = 1;`, то можно предположить, что нулевой бит (вывод RA0) порта A будет «поднят». Какие числа нужно вписывать в порт, чтобы последовательно

установить все выводы в высокое («1») состояние? Восемь выводов порта соответствуют восьми битам регистра порта. А для работы с битами есть хороший инструмент – шпаргалка (калькулятор в любой из ОС).

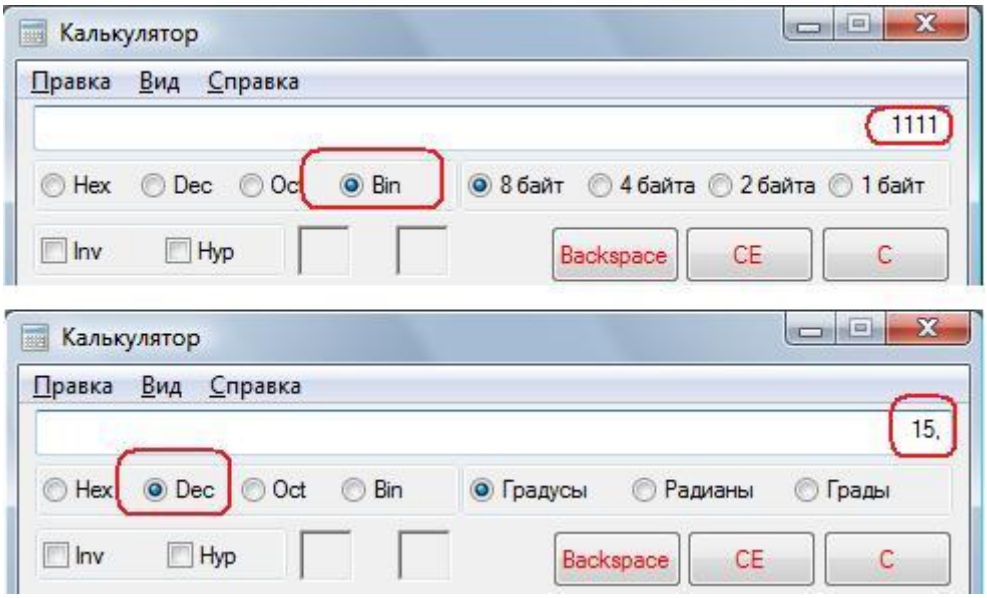


Рис. 3.5. Калькулятор, как шпаргалка для работы с битами

Выяснив, какие числа вписывать в порт А, мы можем записать программу на языке Си:

```
void main()
{
//Initialisation
cmcon = 0x07;
trisa = 0x00;
//Output
porta = 0;
porta = 1;
porta = 3;
porta = 7;
porta = 15;
porta = 31;
porta = 63;
porta = 127;
porta = 255;
}
```

Бумага, как известно, стерпит все. Что ни напишешь, все хорошо. Но понравится ли наша запись кода на языке Си самому языку Си? Вот, что интересно.

Мы можем проверить и это. Изготовители микроконтроллеров предлагают разработчикам бесплатно версии среды разработки. Так производитель PIC-контроллеров предлагает среду разработки MPLAB IDE: http://www.microchip.com/

Эта среда разработки контроллеров прекрасно работает с языком ассемблер, но для использования языка Си нужно добавить компилятор, производимый другим разработчиком. Есть облегченная, но бесплатная версия, HI-TECH (PICC Lite). Основное ограничение – не все контроллеры поддерживаются, и использовать можно только половину памяти. Чтобы скачать компилятор, нужно зарегистрироваться на сайте:

http://www.htsoft.com/products/compilers/PICClite.php

Кроме этого компилятора есть еще SDCC, бесплатный и полный, но его поддерживаются не все версии MPLAB, хотя можно попробовать предыдущие версии на предмет поддержки компилятора SDCC: http://sdcc.sourceforge.net/

И еще, последние версии MPLAB устанавливают компилятор при установке программы.

Вернемся к нашей программе, написанной на языке Си. Запустим установленную на компьютере среду разработки программ MPLAB для микроконтроллеров серии PIC.

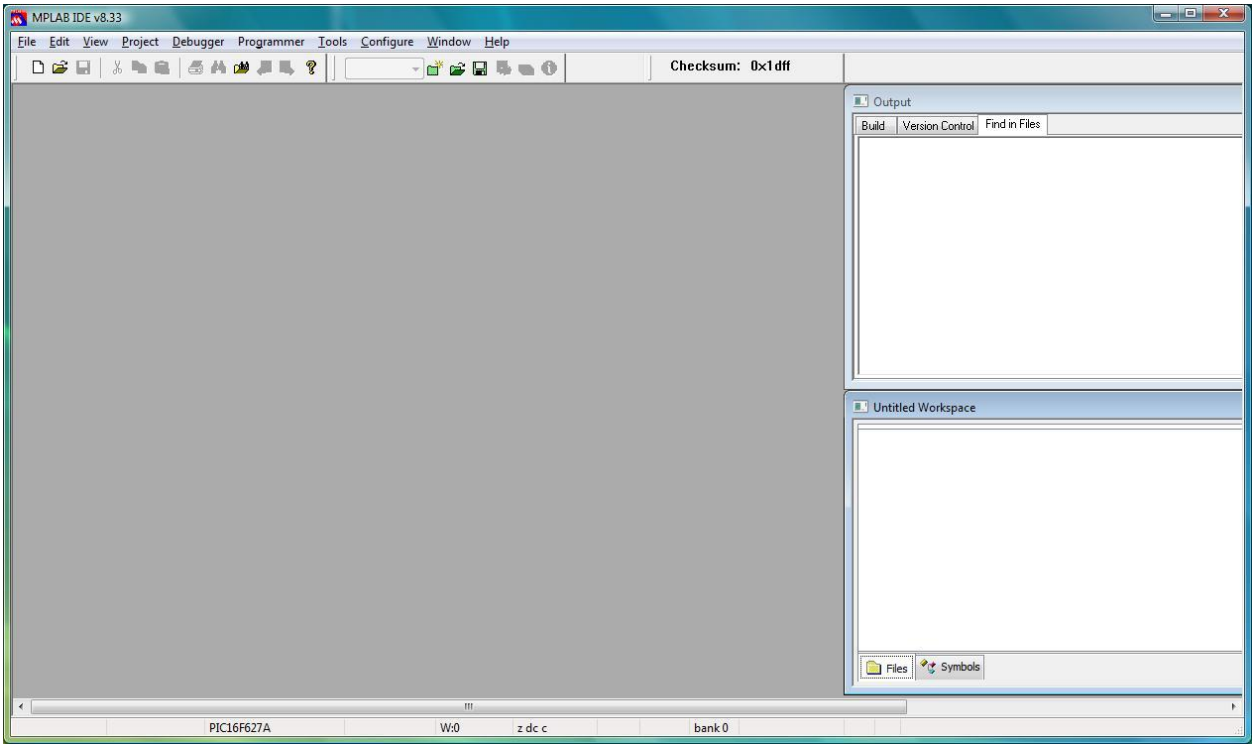


Рис. 3.6. Первый запуск программы MPLAB

Самое разумное при первом знакомстве с программой воспользоваться услугами мастера создания нового проекта: **Project->Project Wizard**.

Рис. 3.7. Мастер создания нового проекта

Кнопка ***Далее>*** открывает первое диалоговое окно:

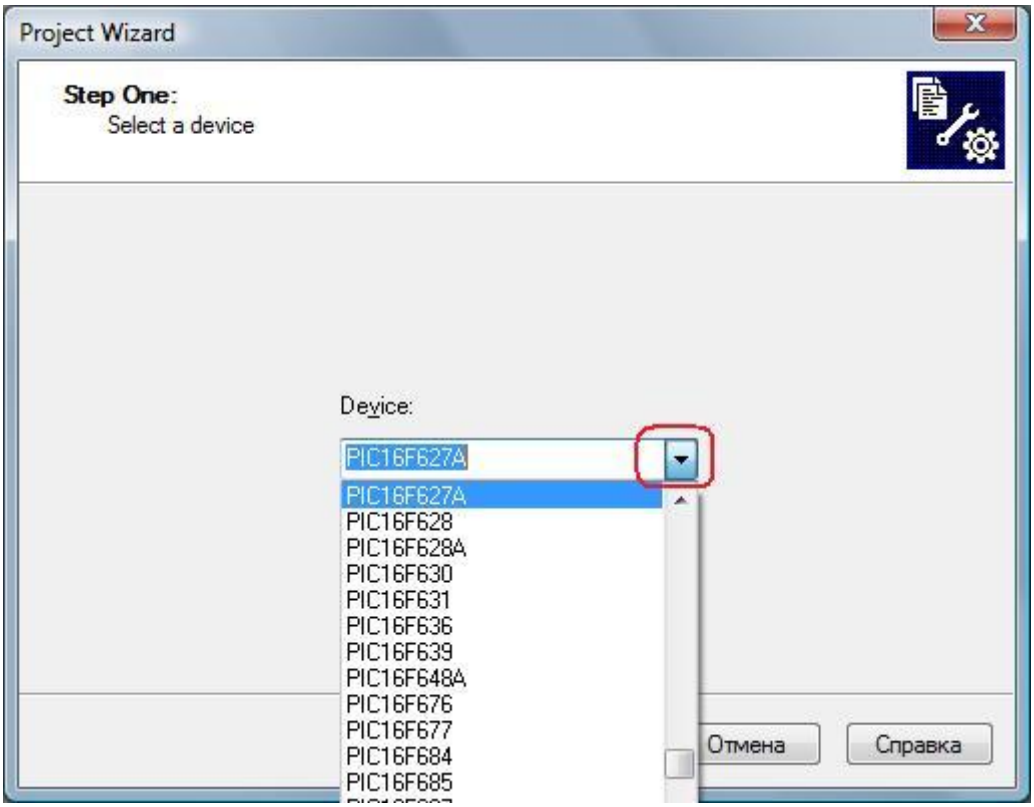


Рис. 3.8. Диалоговое окно выбора модели микроконтроллера

Бесплатная версия компилятора может не работать с PIC16F628A. На всякий случай, пока это не проверено, остановим свой выбор на PIC16F627A. Для выбора служит отмеченная на рисунке кнопка. В следующем диалоге определяется компилятор. Возможно, установленный на этапе установки MPLAB компилятор PICC Lite будет обнаружен сразу, но, если этого не произойдет, можно указать его, используя клавишу ***Browse***. Если установить опцию *Show all installed toolsuites*, то список доступных компиляторов пополнится.

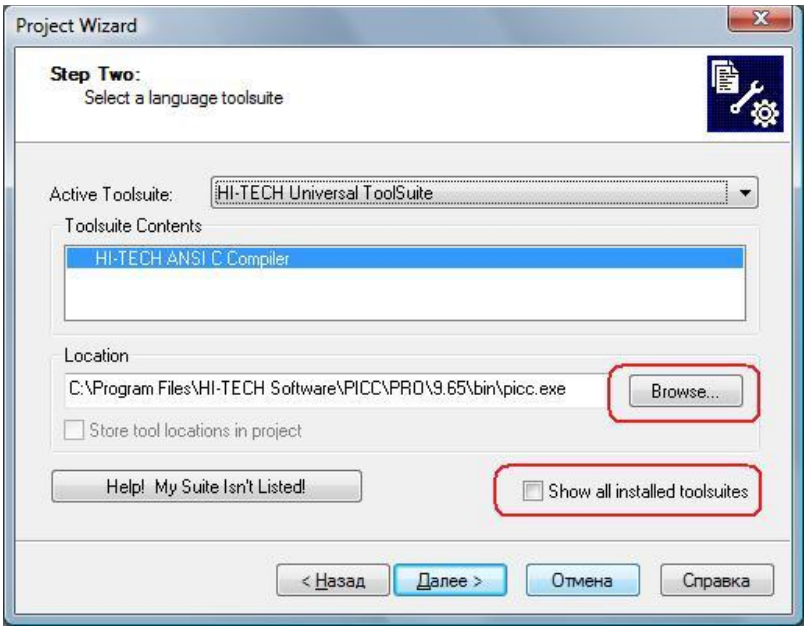


Рис. 3.9. Выбор компилятора для работы с проектом

А в следующем диалоге назовем проект *test* и укажем путь к папке, которую предварительно создали, для этого проекта.

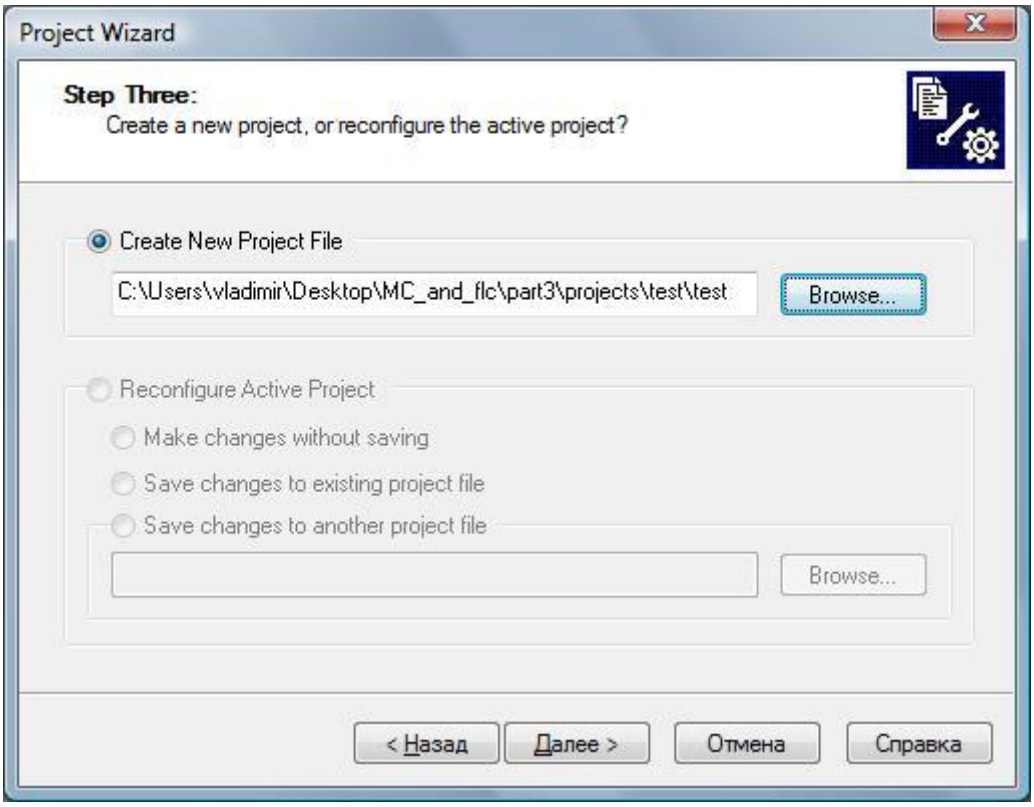


Рис. 3.10. Путь к проекту и имя проекта

По опыту работы с разными программами я знаю, что лучше всего создавать папку, которую можно назвать *work*, в корневой директории основного диска. Но сейчас меня это не очень волнует, и, пока не возникнут трудности, я создаю новый проект на рабочем столе.

На следующем шаге мастер предлагает включить в проект ранее созданные файлы, но их нет, и этот шаг можно пропустить, нажав кнопку ***Далее>***.

Закончив подготовку и нажав на кнопку ***Готово***, вы можете увидеть следующее:

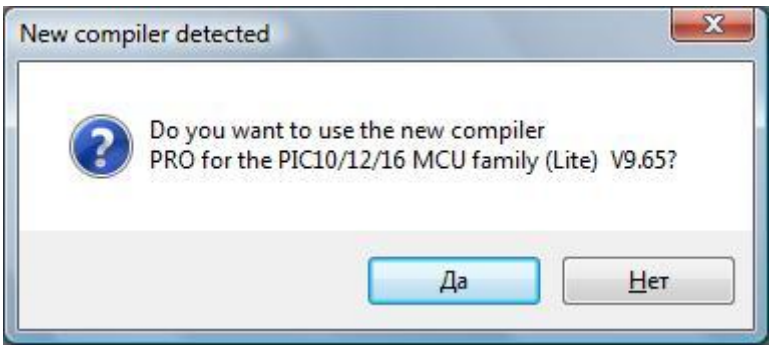


Рис. 3.11. Сообщение о компиляторе

Нажмите кнопку ***Да***.

Прежде пустые окна основного окна программы начинают заполняться.

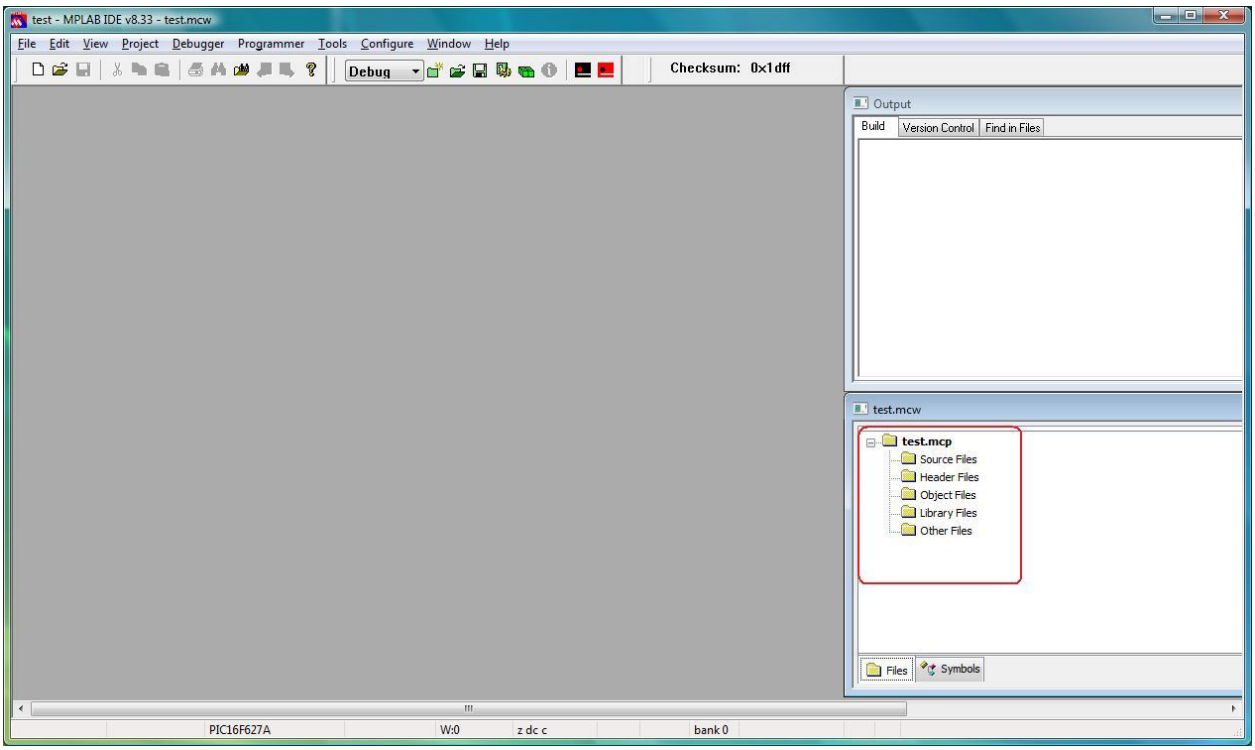


Рис. 3.12. Изменения в основном окне программы после создания проекта

Создадим файл (**Project->Add New File to Project)**, укажем имя нового файла *test.c*, а файл сохраним в той же папке, где лежит проект. Скопируем нашу программу в открытое окно встроенного текстового редактора. Сохраним файл, сохраним проект. И попробуем компилировать файл. В окне Output программы MPLAB на закладке Build появляются сообщения об ошибках.

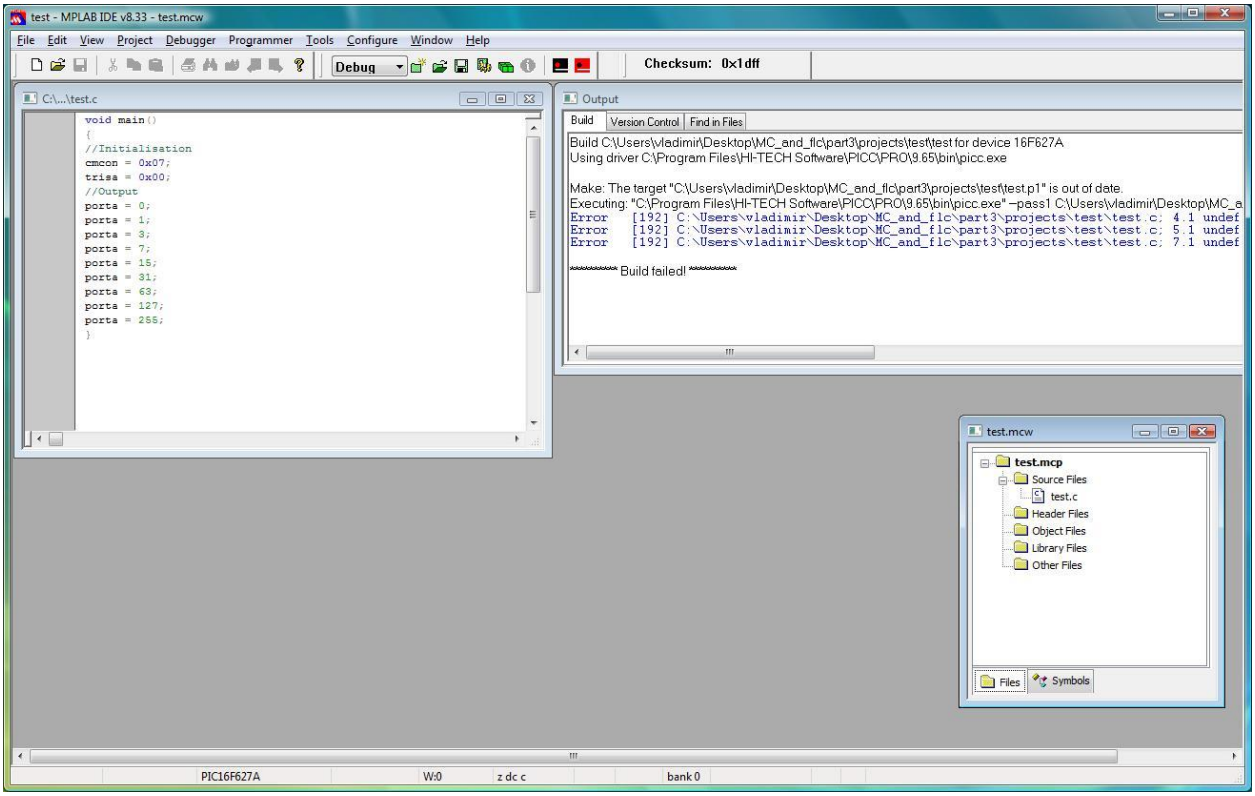


Рис. 3.13. Первая неудачная попытка компиляции

Возможная причина неудачи заключена в следующей фразе:

```
test.c; 4.1 undefined identifier "cmcon"
```

Компилятор сообщает, что идентификатор `cmcon` не определен. В предыдущих версиях компилятора и программы MPLAB (сейчас версия MPLAB 8.33) нужно было включить файл заголовка соответствующего микроконтроллера и изменить названия регистров – заменить маленькие буквы заглавными. Включение файла заголовка дает сообщение о том, что этот файл уже включен, вместо этого предлагается включить другой файл. Для этого достаточно скопировать в программу предложенную строку. После замены букв, перезагрузки программы и выбора ранее созданного проекта компиляция проходит успешно.

В MPLAB есть свой отладчик. Можно проверить, что происходит с портом А. Выберем: **View->Spicial Function Registers**. Этим открывается окно наблюдения за регистрами. Выбираем отладчик:

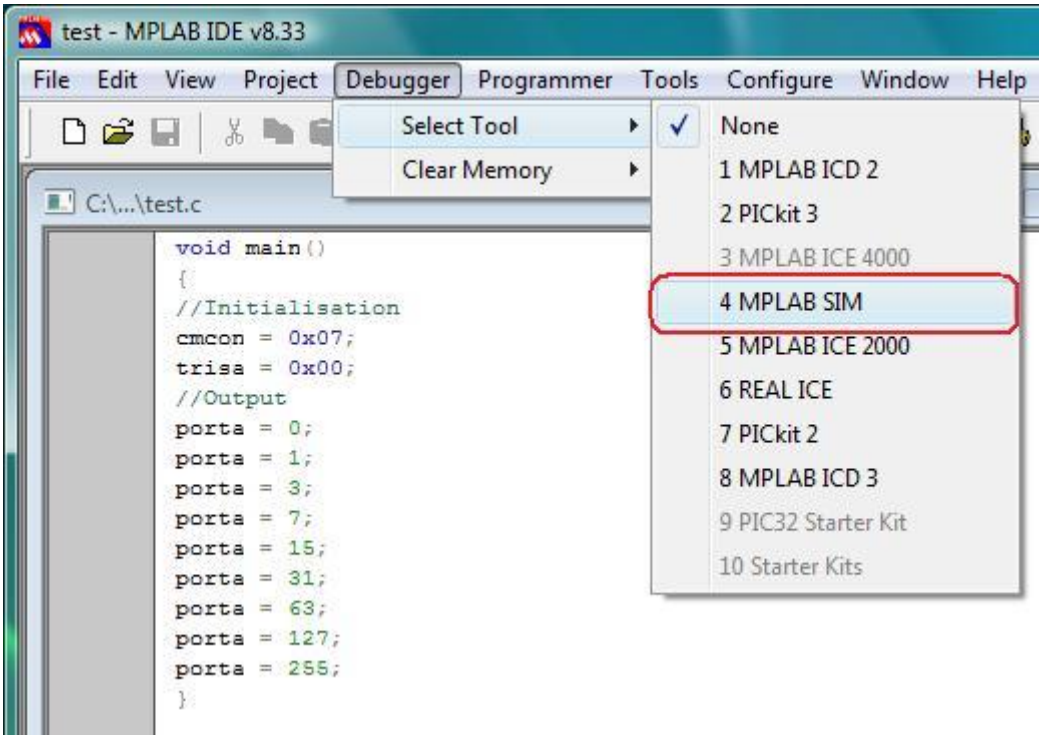


Рис. 3.14. Выбор отладчика в программе MPLAB

Теперь можно запустить отладку: **Debugger->Animate**.

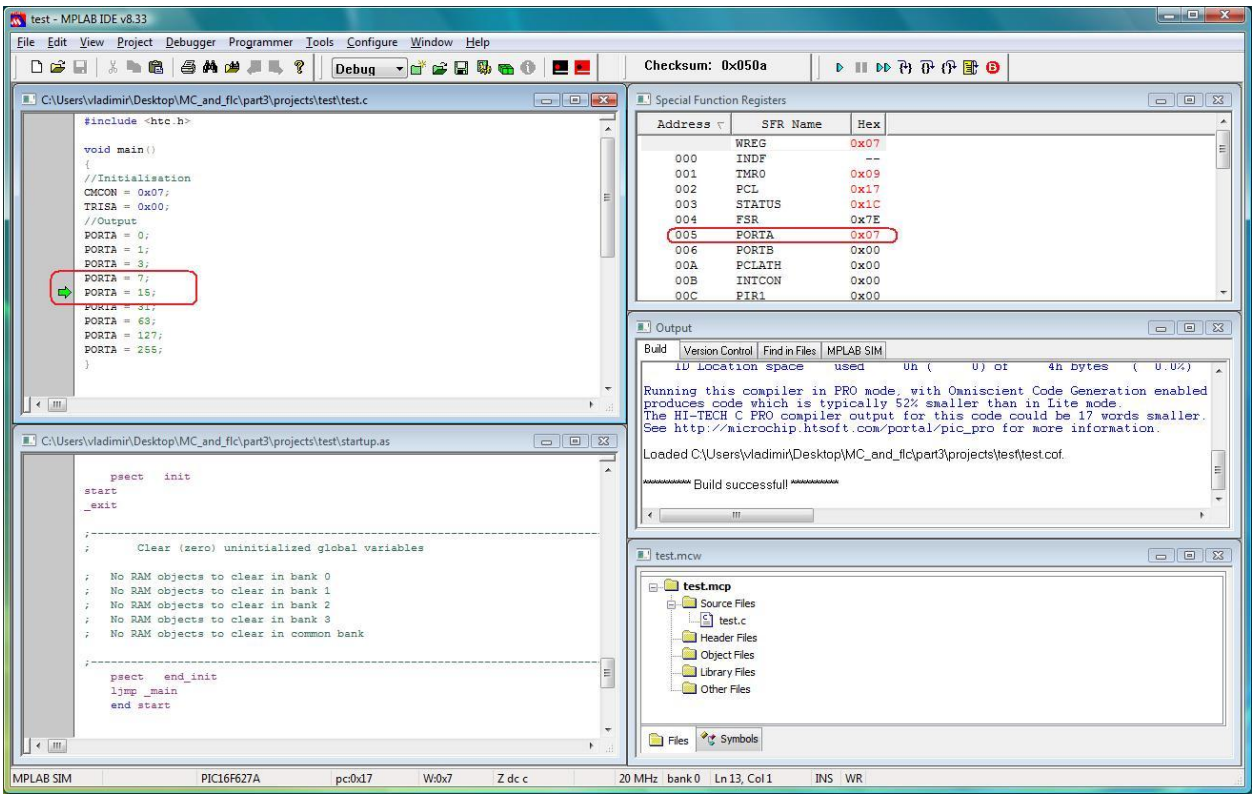


Рис. 3.15. Наблюдение за портом в отладчике программы MPLAB

Состояние порта A меняется согласно тому, что написано в программе.

Мы можем, как и в программе FlowCode, получить загрузочный hex-файл, загрузить его в микросхему, если есть программатор, но, как сказано выше, ничего интересного мы пока на макетной плате не увидим – все светодиоды, связанные с портом A, загорятся разом. Это можно наблюдать в отладчике программы FlowCode. Чтобы создать некоторый «наблюдаемый» эффект для макетирования, нам предстоит еще поработать над программой.

Подведем первые итоги.

Мы сделали несколько «заготовок» на языке Си в программе FlowCode. Использовали одну из них, чтобы получить программу на языке Си, которую можно использовать в среде разработки MPLAB. Мы проверили, что полученная нами программа вполне «устраивает» программу MPLAB, потребовались лишь незначительные поправки. Мы использовали тот фрагмент, который был удобнее для создания программы – запись числа в порт, но могли бы использовать и другие «заготовки».

При этом мы еще даже не приступили к изучению языка Си. Мы руководствовались, скорее, догадками и соображениями.

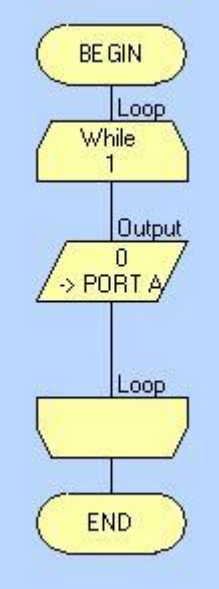
Первые шаги (продолжение)

Вернемся в FlowCode и подумаем над улучшением вида нашей программы. Сейчас программа очень маленькая. Она работает так, как нам хотелось бы, но, когда мы начнем ее «расширять», она разрастется так, что станет «неудобочитаема». Это, как правило, происходит очень быстро. Да и записывать (даже в FlowCode) множество однотипных операций, согласитесь, утомительное занятие. На этот счет программисты давно придумали разные операции, которые помогают сократить число записей.

Заметим, что мы повторяем несколько раз одну и ту же операцию – записываем в порт А некоторое число. Число меняется, но операция нет. Для повторения одной и той же операции (или нескольких операций) давно есть два механизма: цикл и подпрограмма.

Цикл – это повторение одной и той же операции множество раз, вплоть до бесконечного повторения одного и того же.

Есть разные виды циклов, но мы остановимся на двух видах: счетный цикл и условный цикл. В первом случае операция в цикле повторяется заданное число раз, то есть, «цикл считает» количество проходов, а, достигнув нужного, прекращает свою работу. Обращаясь к рисунку 3.4, можно сказать, что программе требуется восемь проходов в цикле. При каждом из них в порт А отправляется число. Если не задумываться над тем, как менять это число, то программа выглядит следующим образом:



Программный элемент **Loop** (цикл), в программе FlowCode находится на инструментальной панели программных компонентов. Как и другие программные компоненты, он «перетаскивается» мышкой в рабочее поле и помещается на линию, соединяющую «программные скобки» BEGIN-END.

В данном случае показан условный цикл *While*. Но в его свойствах (двойной щелчок по элементу открывает диалоговое окно свойств) можно заменить условный цикл счетным.

Рис. 3.16. Цикл в программе FlowCode

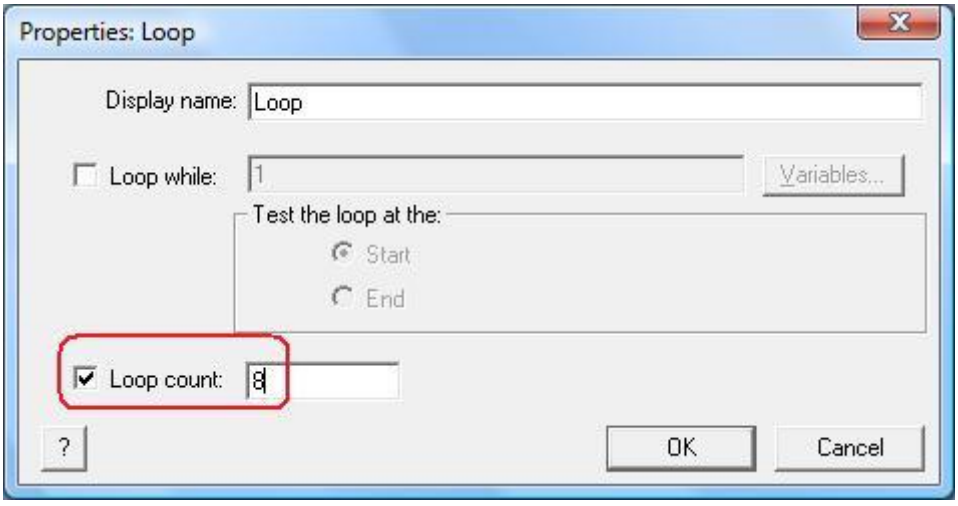


Рис. 3.17. Замена условного цикла счетным

В окошке *Loop count:* количества проходов можно вписать требуемое число. Если вписать число восемь, то цикл будет выполняться восемь раз. И опять, как и прежде, посмотрим, как эта программа выглядит на языке Си.

```
//Variable declarations (объявление переменных)
char FCLV_LOOP1;

void main()
{
      //Initialisation
      cmcon = 0x07;

      //Loop: Loop 8 times
      for (FCLV_LOOP1=0; FCLV_LOOP1<8; FCLV_LOOP1++)
      {
      //Output: 1 -> PORT A
      trisa = 0x00;
      porta = 1;
      }
}
```

Как и прежде, из текста удалено все лишнее. Есть и знакомое место, его мы видели раньше, и оно выделено. Выпишем новые строки кода, поясняя их значение.

```
//Variable declarations (объявление переменных)
char FCLV_LOOP1;
```

Для выполнения цикла восемь раз нужно где-то вести счет. Этой цели служит переменная `FCLV_LOOP1`, которая может иметь любое, в рамках допускаемых компилятором, имя. Кроме того, любая переменная должна иметь заданный тип данных. В данном случае это тип `char`, символьная переменная. Хотя она не используется для хранения символов, но восемь бит этого типа данных вполне устраивают нас.

И дальше новое в коде:

```
void main()
{
      //Loop: Loop 8 times
      for (FCLV_LOOP1=0; FCLV_LOOP1<8; FCLV_LOOP1++)
      {
      что-то делается в этом месте, но не важно, что
      }
}
```

Кстати, все, что следует за двойной косой чертой, это комментарии: записи для программиста, которые поясняют, что это такое или как это понимать. Компилятор игнорирует эти записи. А счетный цикл начинается со слова `for`. Далее в скобках следует запись, что переменная от значения 0 до значения меньшего 8 будет увеличиваться на единицу. Любую переменную можно увеличивать на единицу записью:

```
FCLV_LOOP1 = FCLV_LOOP1 + 1;
```

Но в языке Си часто употребляют запись `FCLV_LOOP1++`, что сокращает время написания оператора, а смысл остается прежним – увеличить значение переменной на единицу. Попутно отметим, что знак «=» в записи выше, отнюдь не знак равенства, а оператор присваивания в языке Си. Для языка Паскаль он выглядит иначе «:=».

Я назвал цикл счетным, но он и в этом случае остается условным, поскольку есть условие:

```
FCLV_LOOP1<8;
```

Все-таки будем называть такой цикл счетным, чтобы легче было различать разные виды циклов. Итак, покинем программу FlowCode, вернемся к программе MPLAB и используем новый программный элемент языка Си для проверки того, как к нему относится компилятор языка Си программы MPLAB. Чтобы не вносить путаницу, я вновь создаю папку с именем test3. И начинаю работу в MPLAB с мастера создания проектов, создавая новый проект под именем test3. Как и в прошлый раз, я копирую текст с записью цикла и вставляю его в новый файл встроенного редактора текста программы MPLAB. Как и в прошлый раз, я меняю маленькие буквы в названиях регистров большими, чтобы не было ошибок на этот счет.

Полученная программа компилируется без ошибок, можно запустить отладку, хотя отлаживать, вернее, смотреть не на что. И можно было бы вернуться к FlowCode, чтобы решить, как нам менять число, которое мы будем записывать в порт, но я хочу поступить иначе.

Мы знаем, как можно менять переменную. Знаем, как записать объявление переменной. Поэтому исправим текст сразу на языке Си:

```
char leds; //объявление новой переменной
```

Внутрь цикла добавим запись:

```
leds = 1 + leds*2;
```

А в порт будем записывать не число, а переменную:

```
PORTA = leds;
```

Программа выглядит следующим образом:

```
#include <htc.h>

//Variable declarations (объявление переменных)
char FCLV_LOOP1;
char leds;

void main()
{
      //Initialisation
      CMCON = 0x07;

      //Loop: Loop 8 times
      for (FCLV_LOOP1=0; FCLV_LOOP1<8; FCLV_LOOP1++)
      {
      //Output: leds -> PORT A
      TRISA = 0x00;
      leds = 1 + leds*2;
      PORTA = leds;
      }
}
```

По совету компилятора, как и в прошлый раз, добавлено включение файла заголовка:

```
#include <htc.h>
```

После компиляции программы и запуска отладчика (**Debugger->Animate**):

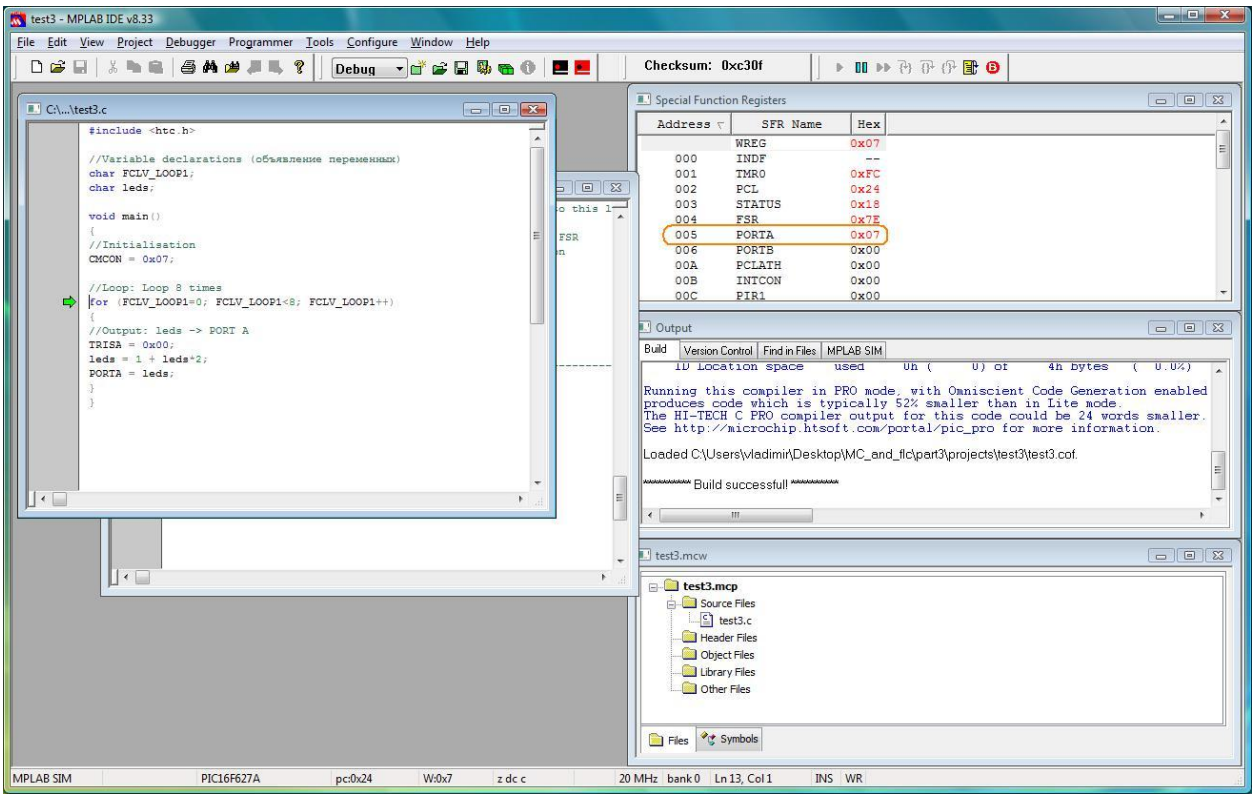


Рис. 3.18. Запуск в отладчике MPLAB модифицированной программы

Если теперь получить hex-файлы двух программ: последней с циклом и первой без него, – то, загрузив эти файлы в две микросхемы контроллеров, можно убедиться, что обе программы работают одинаково (и, кстати, до сих пор ничего интересно увидеть в их работе нельзя).

Как в разговорном языке, так и в языке программирования, можно описать одно и то же различным образом. И, если в начале работы мы переходили от программы FlowCode к программе MPLAB, то сейчас сделаем обратный переход: используем код программы на языке Си для преобразования программы в графический язык.

Как и программа на языке Си позволяет нам вводить переменные, так и программа на графическом языке позволяет сделать это.

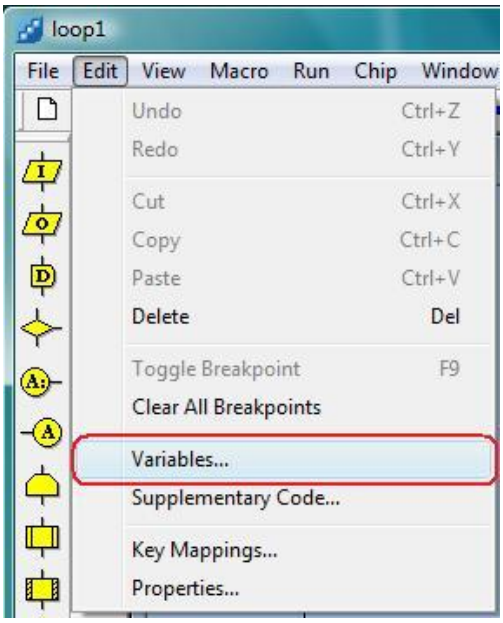


Рис. 3.19. Вызов менеджера обслуживания переменных

Если мы не создали ни одной переменной, то в диалоговом окне увидим следующее:

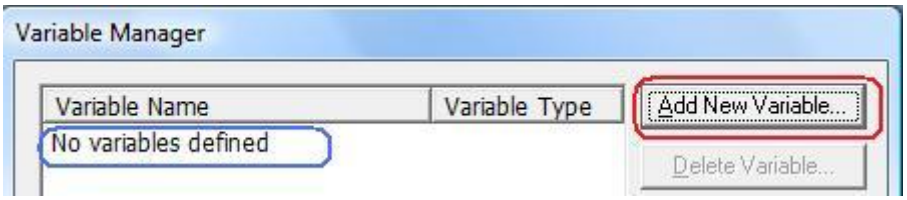


Рис. 3.20. Диалоговое окно менеджера переменных

Воспользуемся кнопкой ***Add New Variable***, чтобы создать новую переменную. После нажатия кнопки открывается новый диалог.

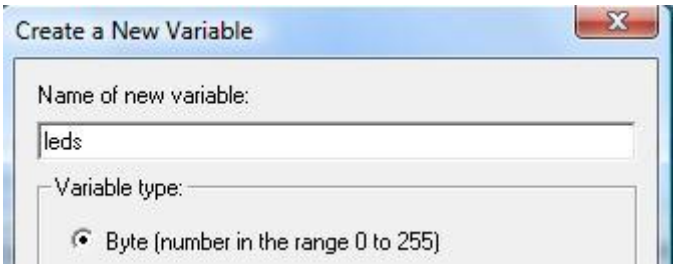


Рис. 3.21. Диалоговое окно создания переменной

Впишем имя переменной, как на языке Си, *leds*, ее тип оставим *Byte*.

Теперь мы можем в порта A писать не число, а переменную.

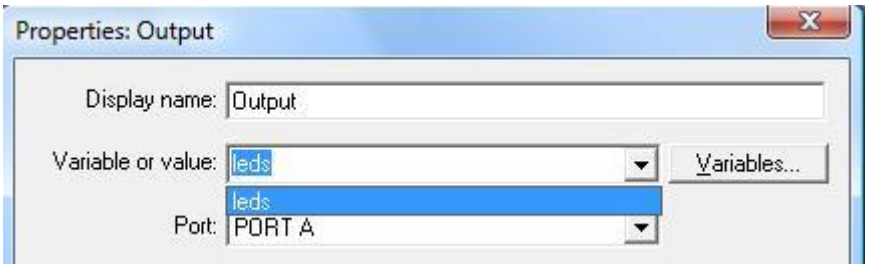
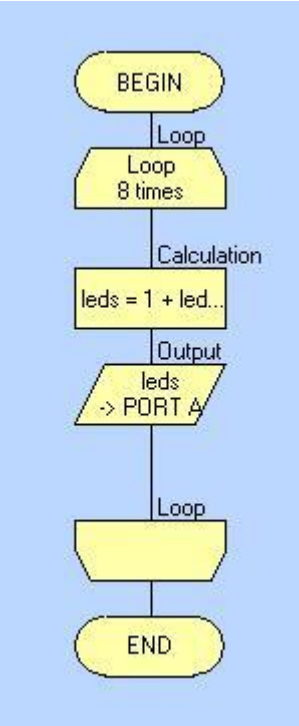


Рис. 3.22. Исправление в диалоге компонента **Output**

Осталось, используя программный компонент **Calculation** программы FlowCode, записать необходимые преобразования в переменную *leds*:

Рис. 3.23. Управление переменной внутри цикла

Программа в FlowCode принимает следующий вид:



Обратите внимание на схожесть записей программы. Если на языке Си для выделения программы используются фигурные скобки, то в FlowCode программа выделяется скобками BEGIN-END.

Если на языке Си цикл начинается с его задания, начинаясь служебным словом `for`, то в FlowCode используется элемент **Loop**. В Си используются фигурные скобки для выделения «тела» цикла, в FlowCode используются графические элементы.

Как в программе на языке Си, так и в программе FlowCode мы меняем переменную *leds* внутри цикла.

Как в программе на языке Си, так и в программе FlowCode мы отправляем переменную в порт A.

Очень похоже, не правда ли?

Рис. 3.24. Запись программы в FlowCode

Может возникнуть вопрос, откуда взялась запись `leds = 1 + leds*2` и в той, и в другой программе? Что означает эта запись, понятно – мы меняем переменную, присваивая ей новое значение. Но почему так, а не иначе? Здесь готового ответа нет. И решение, мне кажется, может быть не одно. Я использовал тот факт, что ряд чисел 1, 3, 7, 15 и т.д., можно представить такой формулой с нулевым начальным значением. Кстати, многие компиляторы при задании переменной дают ей нулевое значение по умолчанию, но неплохо было бы записать это явным образом (что я не сделал!). Подобная небрежность в программировании может приводить к трудно обнаруживаемым ошибкам.

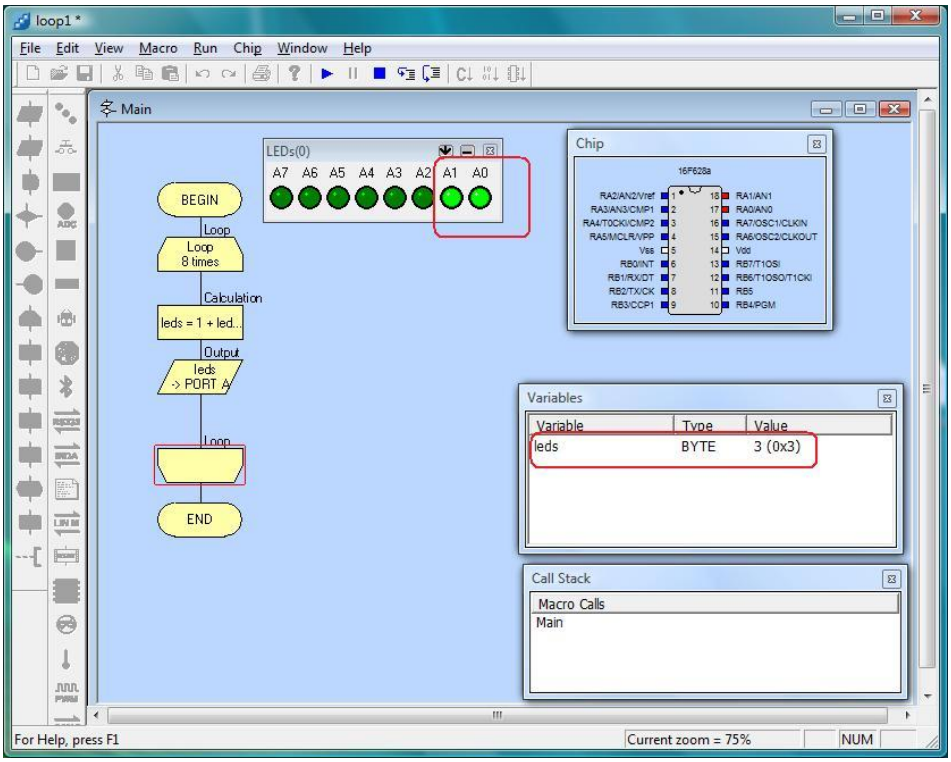


Рис. 3.25. Работы программы в пошаговом режиме в FlowCode

И все-таки хотелось бы увидеть результат работы программы не в отладчике, а на макетной плате. Чтобы это получилось достаточно добавить в программу (на рисунке выше) еще одни программный элемент – **Delay** (пауза). Его можно добавить ниже элемента **Calculation**, в котором меняется число для записи в регистр порта А. По умолчанию длительность паузы одна миллисекунда. Достаточно изменить это значение на 300 мс, скажем, чтобы, запустив «прогон» программы в отладчике (кнопка на инструментальной панели или **Run->Go/Continue** в основном меню) FlowCode увидеть то, что будет показано и на макетной плате.

Однако цель – получить код программы на языке Си – еще не достигнута. Конечно, можно и в этот раз скомпилировать программу в FlowCode и посмотреть код программы на Си. Но ничего интересного, в конечном счете, мы не увидим, поскольку вот такой код:

```
//Delay: 300 ms
delay_ms(255);
delay_ms(45);
```

для любого другого, отличного от компилятора FlowCode, компилятора будет непонятен. Компилятор FlowCode имеет добавляемый файл, где функция `delay_ms()` полностью определена и описана. А компилятор MPLAB просто покажет ошибку. Важно еще, что функция дает интервал времени, выраженный в долях секунды, но определяемый в зависимости от тактовой частоты микроконтроллера. А тактовая частота может в широких пределах меняться. Чтобы получить нужный интервал времени, придется использовать какие-то другие механизмы в программе на языке Си. В данном конкретном случае нам не нужна строго определенная по времени пауза, что позволяет использовать в коде программы пустой счетный цикл, с которым мы знакомы.

```
#include <htc.h>

//Variable declarations (объявление переменных)
char FCLV_LOOP1;
char leds;
int dl;

void main()
{
      CMCON = 0x07;
      for (FCLV_LOOP1=0; FCLV_LOOP1<8; FCLV_LOOP1++)
      {
      TRISA = 0x00;
      leds = 1 + leds*2;
      PORTA = leds;
      for (dl=0; dl<=10000; dl++){}
      }
}
```

Для добавленного цикла используется еще одна переменная *dl* типа *int* (целое). В остальном все нам знакомо.

Длительность паузы, «изготовленной» подобным образом, можно проверить на макетной плате, подбирая подходящее значение. Можно использовать отладчик, чтобы «оценить» длительность паузы. Но самое время заглянуть в папку примеров, которые установлены вместе с компилятором HI-TECH для программы MPLAB. Среди примеров есть и пример использования функции паузы, которая поддерживается файлом *htc.h*.

Все, что необходимо добавить в программу, оказывается парой строк, а программа приобретает вид:

```
#include <htc.h>
#define _XTAL_FREQ 4000000

//Variable declarations (объявление переменных)
char FCLV_LOOP1;
char leds;

void main()
{
      //Initialisation
      CMCON = 0x07;

      //Loop: Loop 8 times
      for (FCLV_LOOP1=0; FCLV_LOOP1<8; FCLV_LOOP1++)
      {
      //Output: leds -> PORT A
      TRISA = 0x00;
      leds = 1 + leds*2;
      PORTA = leds;
      __delay_ms(100);
      }
}
```

Длительность в 100 мс получается, как определено инструкцией:

```
#define _XTAL_FREQ 4000000
```

при частоте 4 МГц. Если использовать в МК PIC16F628A внутренний тактовый генератор, то частота, как раз, получится равной 4 МГц.

Эту задержку, если нужно ее увеличить до 300 мс, можно повторить три раза с помощью того же счетного цикла:

```
char i;
for (i=0; i<3; i++) __delay_ms(100);
```

С паузами в FlowCode есть одна небольшая неприятность – можно использовать только целые значения секунд или миллисекунд и нельзя получить паузы длительностью в несколько микросекунд. В программе MPLAB эта проблема решается гораздо проще.

Итак.

С помощью программы FlowCode мы получили текст программы на языке Си, который можно компилировать в среде разработки MPLAB, предназначенной для работы с языками Си и ассемблером. Результат полной компиляции, hex-файл, можно загрузить в микросхему, создав почти полезное устройство, если всю программу поместить в бесконечный цикл while. Микроконтроллер будет, например, включать гирлянды или оформление витрины. При этом мы использовали только три программных компонента программы FlowCode.

Шире шаг – ветвление, подпрограмма, ввод, прерывание, программатор

Очень часто есть необходимость в ветвлении программы. То есть, если произошло какое-то событие или изменение, скажем, переменной, то соответственно этому программа должна выполнить разные действия. Это условное ветвление программы. Например, если переменная, которую мы назовем *SOME*, принимает значение равное единице, то выполняется одно действие, иначе другое.

Программный компонент **Decision** в FlowCode, если его добавить в программу, а затем получить код на языке Си, даст следующую запись:

```
if (FCV_SOME == 1)
      {
            // Некое действие

      } else {
            // Другое действие
      }
```

Это буквальная запись: если (в скобках условие), то… иначе другое. С подобной записью мы сталкивались, когда переводили в код Си вывод одного бита порта А.

Условия могут при необходимости следовать одно за другим, выстраиваясь в цепочку. В языке Си, чтобы не загромождать текст программы подобными конструкциями, предусмотрен оператор, который называется «программным переключателем».

В программе FlowCode необходимость в использовании условного ветвления возникает чаще всего тогда, когда используется ввод (программный компонент **Input**). К микроконтроллеру может быть подключена клавиатура, и программа ждет нажатия клавиши, и когда клавиша нажата, программа меняет свой ход. Или к микроконтроллеру подключен датчик. Пока на входе одно значение, получаемое от датчика, программа, скажем, подключает обогреватель, а при изменении состояния датчика, когда меняется значение на входе микроконтроллера, программа отключает обогреватель.

Ветвления программы возникают и при использовании циклов, используются при программировании и безусловные переходы (оператор *GOTO*). Но, пожалуй, условный переход используется наиболее часто. Особенно это характерно для микроконтроллеров, основное назначение которых обеспечить отслеживание событий во внешнем мире и выбрать реакцию на эти события, в зависимости от порядка или сочетания событий, отражающихся на входах микроконтроллера.

Кроме явных ветвлений программы есть ряд ветвлений, которые, собственно, ветвлениями программы и не считаются, хотя программа, дойдя до этого элемента, не спешит выполнить следующий.

В программе FlowCode есть программный компонент **Macro**.

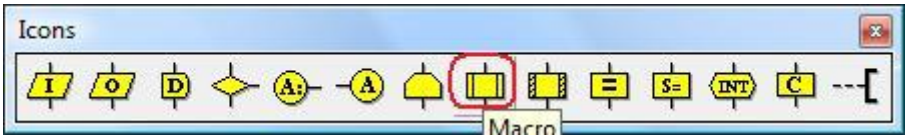
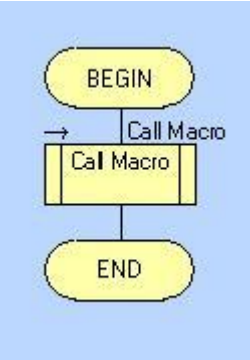


Рис. 3.26. Подпрограмма в FlowCode

Мы можем добавить этот компонент в программу.



Добавление компонента, как это видно из рисунка, не более чем вызов подпрограммы (*Call Macro*). Саму подпрограмму следует написать.

Рис. 3.27. Создание программы

Открыв диалоговое окно этого компонента двойным щелчком левой клавиши мышки по нему, дадим ему имя. Но перед этим следует, используя кнопку ***Create New Macro***, создать подпрограмму.

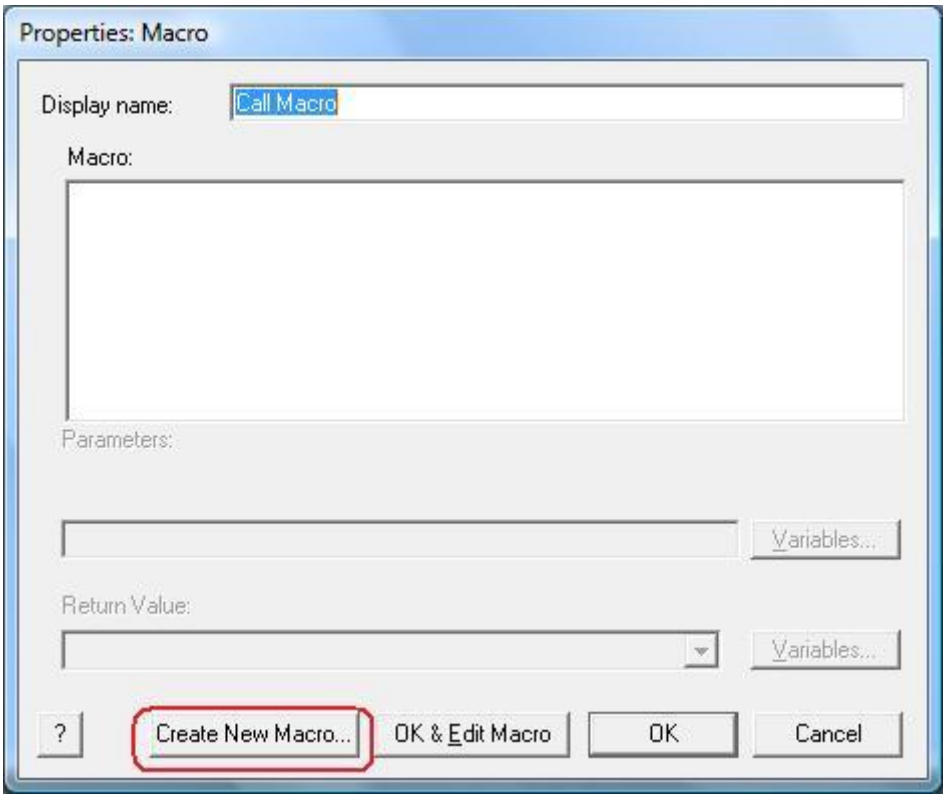


Рис. 3.28. Диалоговое окно **Macro**

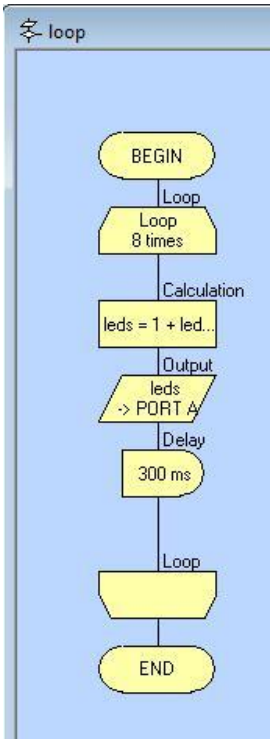
В новом диалоговом окне можно вписать имя подпрограммы.

Рис. 3.29. Задание имени подпрограмме

Нажимая кнопки ***OK***, мы возвращаемся в окно основной программы. Мы еще не написали подпрограмму, но можем компилировать результат на язык Си (**Chip->Compile to C**). Открыв полученный код на языке Си во встроенном текстовом редакторе, можно посмотреть, что из текста, касающегося подпрограммы, было добавлено.

```
//Macro function declarations (объявление)
void FCM_loop();

//Macro implementations (реализация)
void FCM_loop()
{
}

void main()
{
    //Initialisation
    cmcon = 0x07;

    //Call Macro
    //Call Macro: loop (вызов подпрограммы)
    FCM_loop();
}
```

Хотя нам это и не нужно, мы можем ранее написанную программу вставить в подпрограмму.



Нам это не нужно, поскольку программа уже есть. Но, с другой стороны, чтобы не писать новую программу для этого примера, отчего бы не использовать уже готовый вариант?

Чтобы создать подпрограмму можно воспользоваться кнопкой ***OK & Edit Macro*** в диалоге (рис. 3ю28). В результате появится второе рабочее поле с именем *loop*.

Все элементы программы, благо их немного, можно быстро разместить так же, как это было сделано в основной программе ранее. Но, можно было в предыдущей программе использовать функцию экспорта подпрограммы, а сейчас импортировать подпрограмму.

Для экспорта подпрограммы нужно было добавить компонент Macro, дать ему имя, перенести всю программу в подпрограмму. После этого активизируется команда экспорта.

Рис. 3.30. Создание подпрограммы

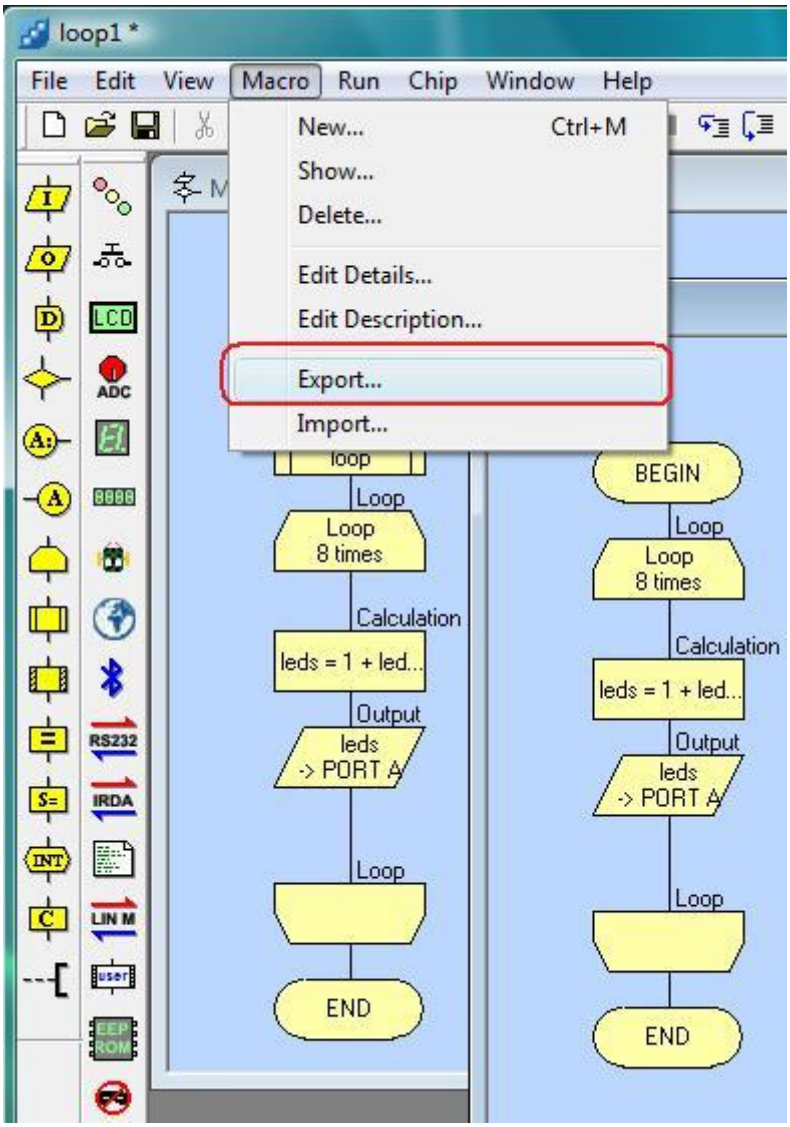


Рис. 3.31. Подготовка программы к экспорту подпрограммы

Экспортированная подпрограмма сохраняется с расширением, которое FlowCode понимает как подпрограмму, и именем, которое мы дали подпрограмме.

Экспортированную подпрограмму можно импортировать в новой программе (если сделать это до того, как мы дали имя подпрограмме).

Но вернемся к подпрограмме на языке Си. Написав подпрограмму, можно получить код на языке Си в FlowCode. И посмотреть, что изменилось с прошлого раза?

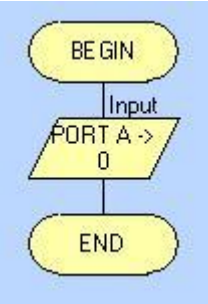
```
//Macro implementations (реализация)
void FCM_loop()
{
    //Loop
    //Loop: Loop 8 times
    for (FCLV_LOOP1=0; FCLV_LOOP1<8; FCLV_LOOP1++)
    {
          //Calculation
          //Calculation:
          //  leds = 1 + leds * 2
          FCV_LEDS = 1 + FCV_LEDS * 2;
```

```
            //Output
            //Output: leds -> PORT A
            trisa = 0x00;
            porta = FCV_LEDS;


            //Delay
            //Delay: 300 ms
            delay_ms(255);
            delay_ms(45);
    }
```

В первую очередь изменилось содержание реализации подпрограммы. У нас есть опыт использования полученного в программе FlowCode кода на языке Си, поэтому, поменяв паузу, изменив написание регистров, мы можем использовать подпрограмму в среде разработки MPLAB.

Начиная рассказ о программных компонентах FlowCode, мы добавили единственный элемент – **Output**. Повторим это для **Input** – программного компонента, обеспечивающего ввод информации в контроллер. В самом простом случае эта информация определяет состояние одного или нескольких выводов: высокий уровень или низкий. Например, к этому выводу подключена кнопка. Если вывод «подтянут» резистором к плюсу источника питания, то кнопка «вторым концом», видимо, должна быть соединена с общим проводом. Тогда при нажатой кнопке на входе будет низкий уровень. Изменение уровня на входе можно зафиксировать как событие, на которое микроконтроллер должен как-то отреагировать.

Но вначале проделаем все операции, то есть, создадим новый файл, «подцепим» мышкой программный компонент **Input** на инструментальной панели программных компонентов и перенесем его на линию, соединяющую программные скобки BEGIN-END.



Пока мы не меняем свойств компонента, не возникает никаких вопросов. Но, если открыть диалоговое окно, чтобы убедиться в использовании всех выводов порта для ввода, то при попытке закрыть окно, нажав на кнопку ***OK***, мы увидим сообщение…

Рис. 3.32. Добавление в программу компонента **Input**

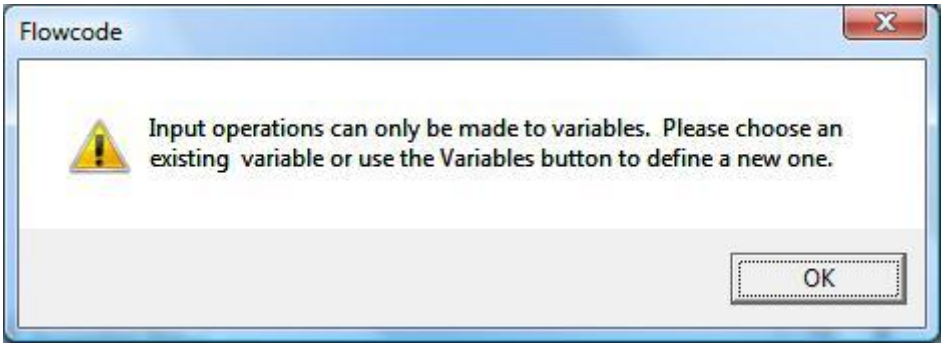


Рис. 3.33. Сообщение по нажатию на кнопку ***OK*** диалога

В этом сообщении сказано, что операция ввода может быть осуществлена только для переменной. Предлагается выбрать либо уже существующую переменную, либо использовать

кнопку ***Variables*** для создания новой переменной. Воспользуемся последним предложением для создания переменной *key*, которая будет «привязана» к порту для отображения его состояния. Закрыв диалоговое окно, мы можем оттранслировать программу на Си (**Chip->Compile to C**). Убрав, как обычно, все «лишнее», получим код на языке Си:

```
//Variable declarations
char FCV_KEY;

void main()
{
    //Initialisation
    cmcon = 0x07;

    //Input: PORT A -> key
    trisa = trisa | 0xff;
    FCV_KEY = porta;
}
```

Запись: `trisa = trisa | 0xff;` означает логическую операцию ИЛИ между значением регистра `trisa` и шестнадцатеричным числом `ff`, в результате которой в регистр будут записаны единицы, что, в свою очередь, означает – порт предназначен для ввода. А мы получили на языке Си фрагмент программы, позволяющий отслеживать состояние порта, при изменении которого, мы можем что-то сделать, исходя из наших намерений и нужд.

И еще одно, если использовать не порт A, а порт B, то… для отслеживания состояния порта можно использовать прерывание. Иногда вполне можно обходиться без прерываний, но не всегда. Посмотрим, как выглядит прерывание, записанное на языке Си.

Создадим новый файл, в программу добавим программный элемент **Interrupt**.

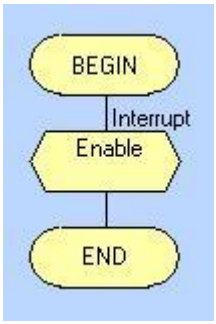


Рис. 3.34. Программа с компонентом **Interrupt**

Пока мы не задали свойств прерывания, программа показывает только то, что разрешено «какое-то» прерывание. Откроем диалоговое окно свойств прерывания. Из возможных для микроконтроллера PIC16F628A прерываний выберем одно, отмеченное на рисунке.

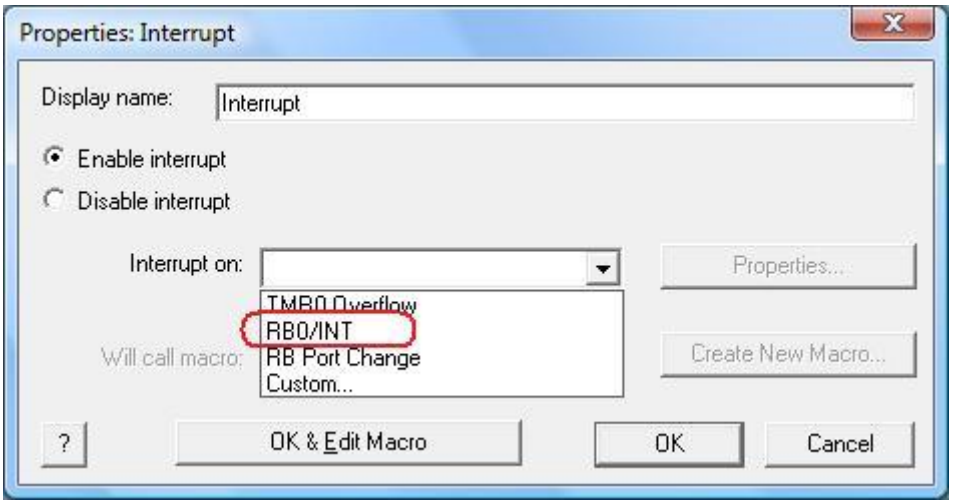


Рис. 3.35. Диалоговое окно программного компонента прерывание

Выбор этого вида прерывания означает, что прерывание разрешено при изменении состояния одного вывода порта B, нулевого бита регистра порта. Но, если мы разрешили прерывание, мы должны создать подпрограмму обслуживания прерывания, что можно сделать, используя кнопку *Create New Macro* (после выбора вида прерывания она активизируется). Подпрограмму назовем key_intr, впрочем, вы вольны выбрать другое название.

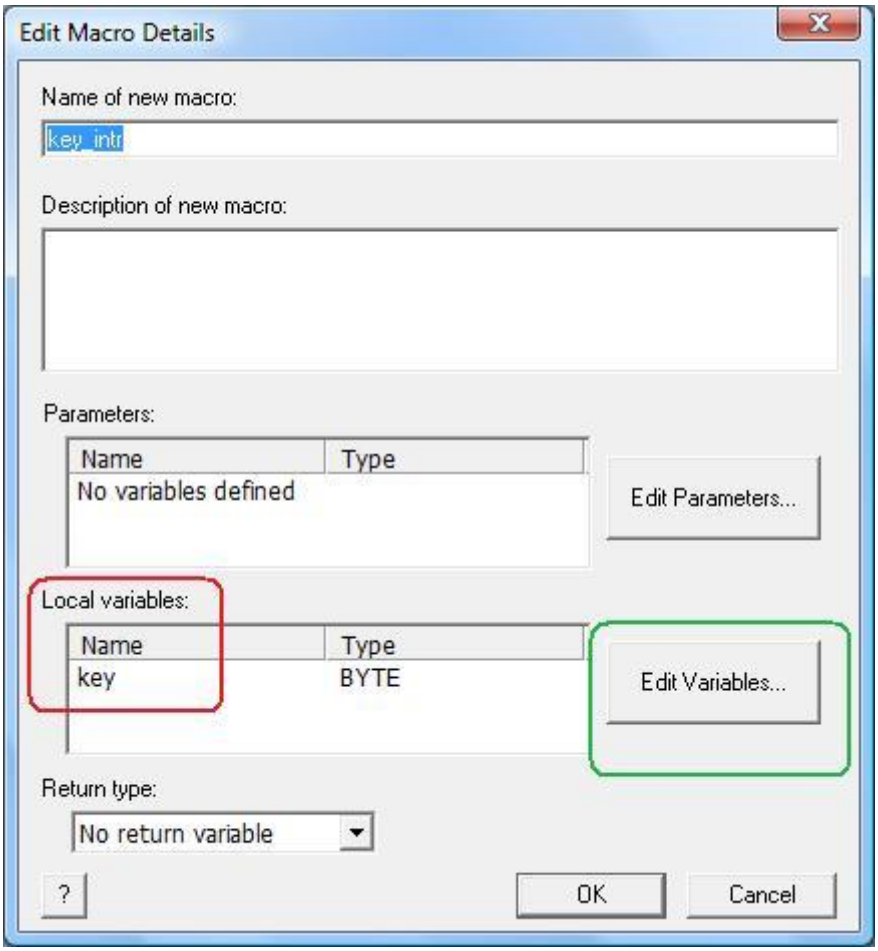


Рис. 3.36. Диалоговое окно создания подпрограммы обслуживания прерывания

Для подпрограммы я создал (с помощью кнопки ***Edit Variables***) локальную, то есть, существующую только в пределах подпрограммы, переменную *key*. И создал только для того, чтобы показать разницу между локальной и глобальной, действительной для всей программы, переменными. В самой подпрограмме обслуживания прерывания я добавлю только один программный компонент **Calculation**.

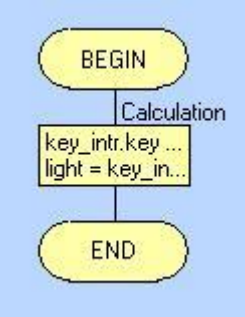


Рис. 3.37. Вид подпрограммы обслуживания прерывания

А сам элемент **Calculation** заполнен:

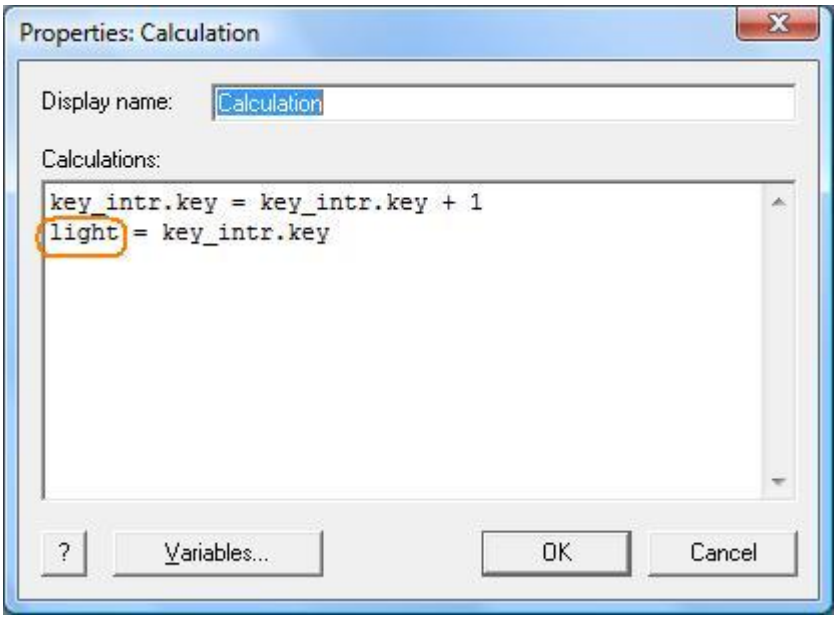
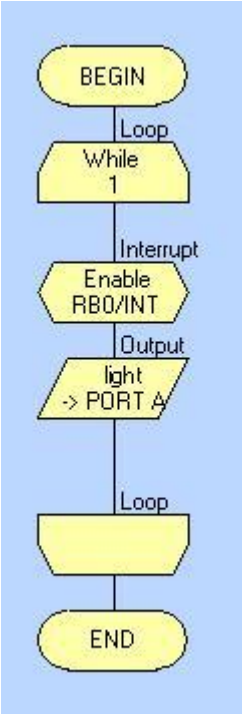


Рис. 3.38. Записи в **Calculation**

Вторую переменную, *light*, нужно создать в программе, как обычно, чтобы она стала доступна в любом месте программы, в частности, и в подпрограмме. Кстати, если сейчас заглянуть в менеджер переменных (**Edit->Variables**), то мы увидим только переменную *light*.

Добавим в программу бесконечный цикл, мы умеем это, добавим вывод в порт A, и это мы умеем, чтобы программа выглядела так:



Для отладки программы, вернее, чтобы увидеть, что делает программа, если она что-то будет делать, используем линейку светодиодов с панели дополнительных компонентов и выключатели. Можно оставить выключатели, изменив только характер включения, как есть, но можно убрать все лишние, оставив только один.

Посмотреть, что происходит в программе, удобнее в режиме пошаговой отладки.

Проходя цикл While несколько раз, вы не увидите изменений в состоянии порта A – ни один из светодиодов не загорается. Но, если вы включите **Switches**, то попадете в подпрограмму обработки прерывания, где изменится значение переменной *light* через локальную переменную *key*.

Рис. 3.39. Программа с прерыванием

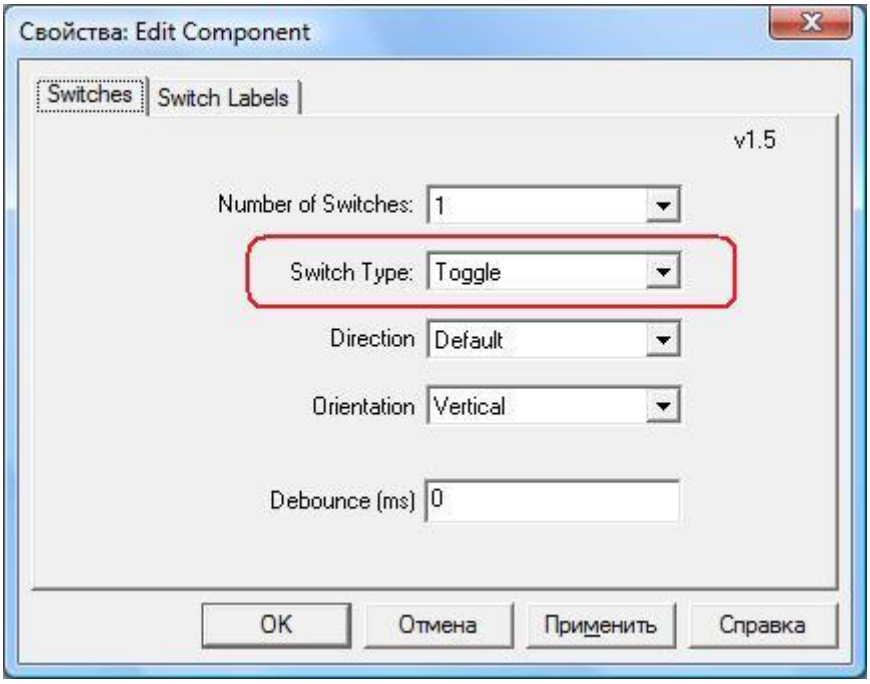


Рис. 3.40. Диалоговое окно свойств дополнительного компонента **Switches**

Разница между локальной и глобальной переменными отражается в том, что сколько бы раз вы ни попали в подпрограмму, значение глобальной переменной *light* остается прежним. Каждый раз, когда возникает прерывание, локальная переменная *key* «начинает новую жизнь», принимая значение по умолчанию равное нулю. В компоненте подпрограммы **Calculation** она увеличивает это значение на единицу и передает его глобальной переменной *light*. Все. В следующий раз, при следующем прерывании, все начинается сначала.

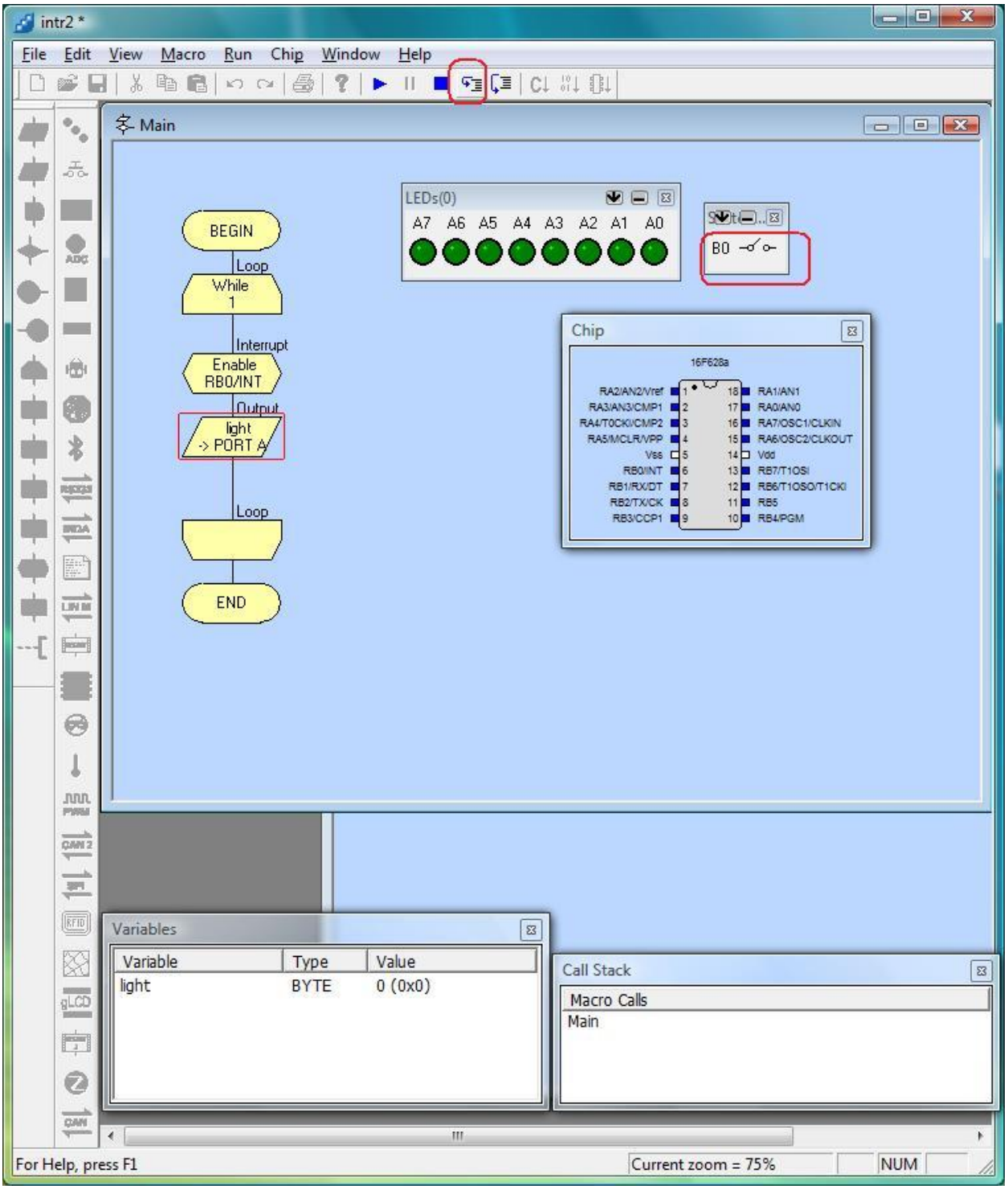


Рис. 3.41. Пошаговое прохождение программы до изменения состояния выключателя

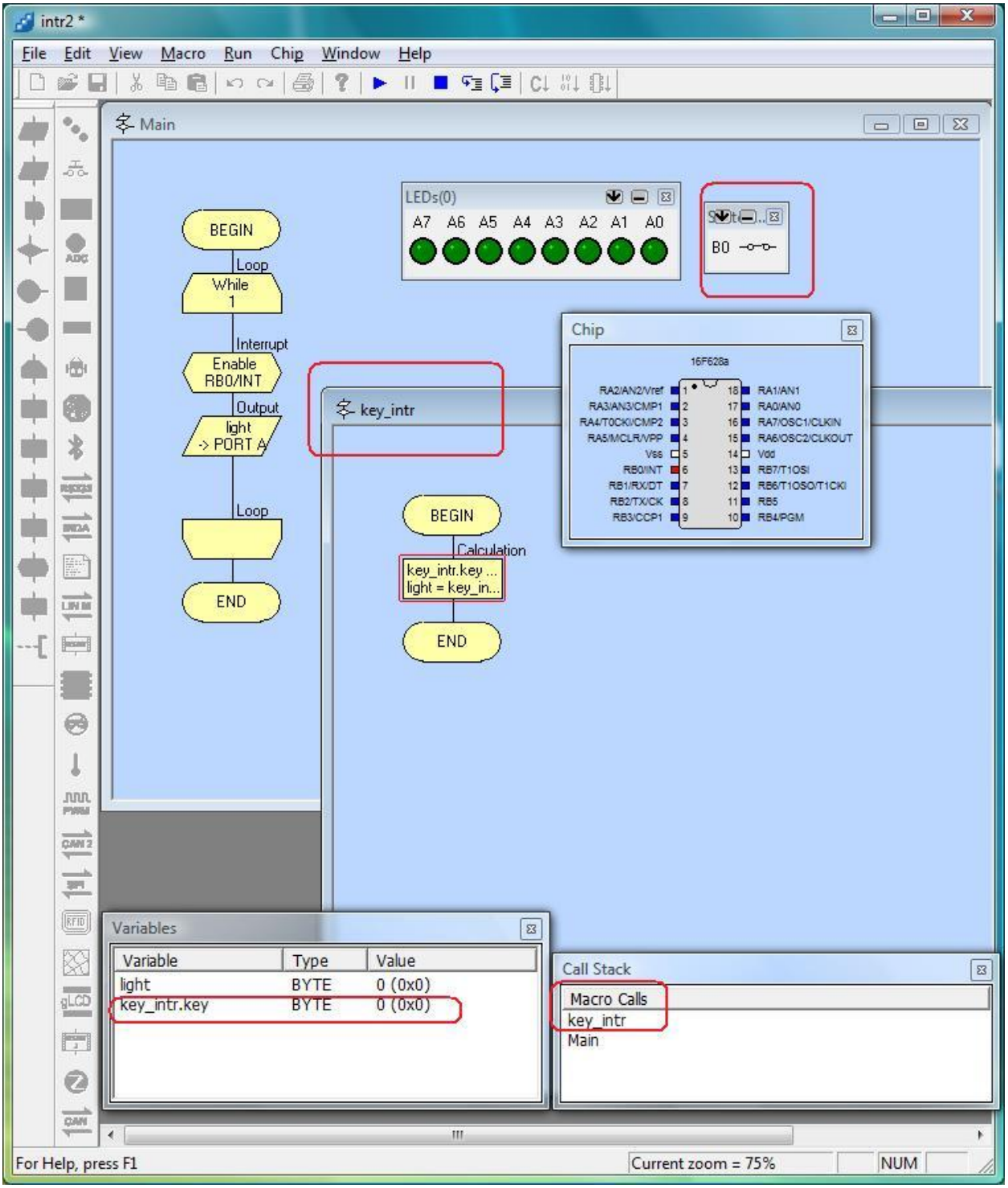


Рис. 3.42. Прерывание при изменении состояния выключателя

И, оттранслировав полученную программу, мы получим код на языке Си:

```
//Macro function declarations
void FCM_key_intr();

//Variable declarations
char FCV_LIGHT;

//Macro implementations
void FCM_key_intr()
{
 //Local variable definitions
 char FCL_KEY;

 //Calculation:
 //  key_intr.key = key_intr.key + 1
 //  light = key_intr.key
 FCL_KEY = FCL_KEY + 1;
 FCV_LIGHT = FCL_KEY;
}

void main()
{

 //Initialisation
 cmcon = 0x07;

 //Interrupt initialisation code
 option_reg = 0xC0;

 //Loop
 //Loop: While 1
 while (1)
 {
       //Interrupt: Enable RB0/INT
       option_reg.INTEDG=1;
       intcon.GIE=1;
       intcon.INTE=1;

       //Output: light -> PORT A
       trisa = 0x00;
       porta = FCV_LIGHT;
 }
}
```

Проверить, как отнесется к этому коду на языке Си компилятор программы MPLAB (или AVRStudio), я предоставлю вам, а сейчас хочу упомянуть о программаторе. С программой MPLAB работает программатор PICkit 2 (появился и PICkit 3), подключаемый к порту USB компьютера. Есть подозрение, что он будет работать и с программой FlowCode. Стоит программатор не слишком дорого, вдобавок его можно собрать и самостоятельно – в Интернете можно найти и схему, и прошивку. А выбор программатора осуществляется в диалоговом окне конфигурации (**Chip->Configure**).

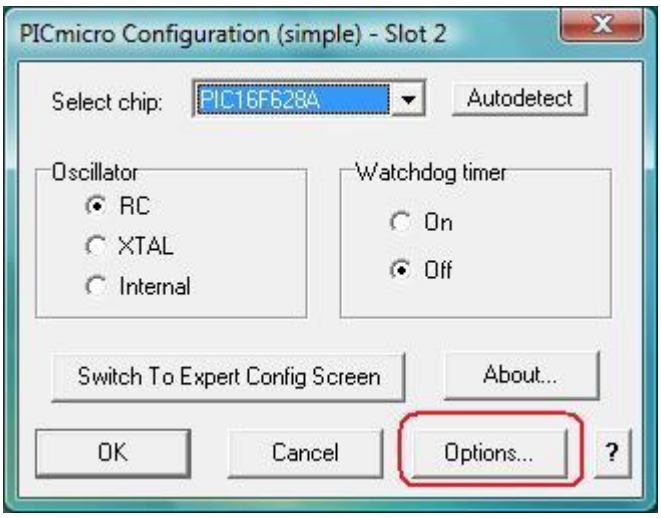


Рис. 3.43. Диалоговое окно конфигурации микроконтроллера

Кнопка ***Options*** открывает следующий диалог:

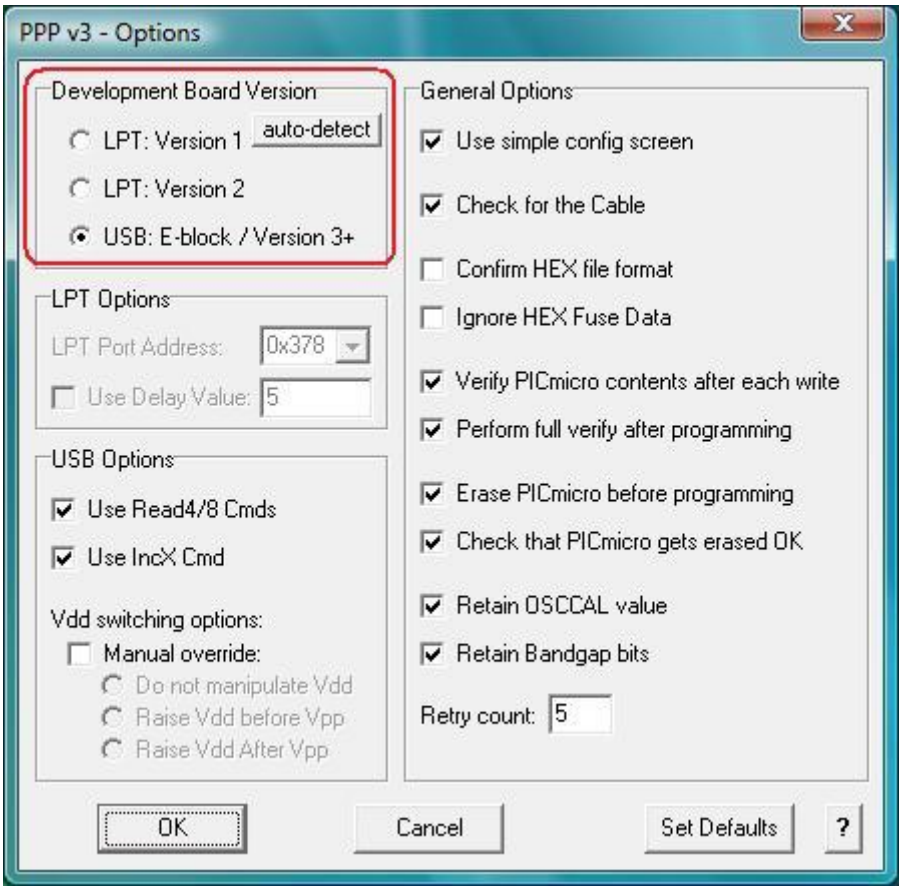


Рис. 3.44. Диалоговое окно настроек программатора

Кроме подключения программатора к порту USB компьютера, есть возможность использовать программатор, работающий с портом LPT. Для тех, кого интересует программатор, работающий с FlowCode, я советую заглянуть на страницу одного из сайтов:

http://microsin.ru/content/view/820/1/

А рассказ будет продолжен о…

Встроенные модули USART и PWM

Микроконтроллер PIC16F628A,как, впрочем, и другие, имеет встроенный модуль для поддержки сетевой работы – USART.

Используем компонент **RS232** панели дополнительных компонентов для программы, поддерживающей работу с USART.

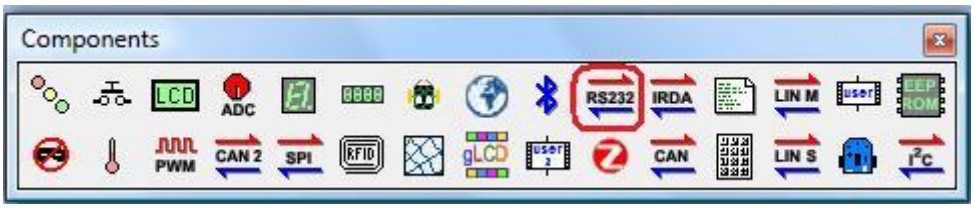


Рис. 3.45. Компонент RS232 для работы с USART

После того, как RS232 добавлен (достаточно «щелкнуть» по нему мышкой), можно добавить в программу программный компонент **Component Macro**. К этому моменту все выглядит как-то так:

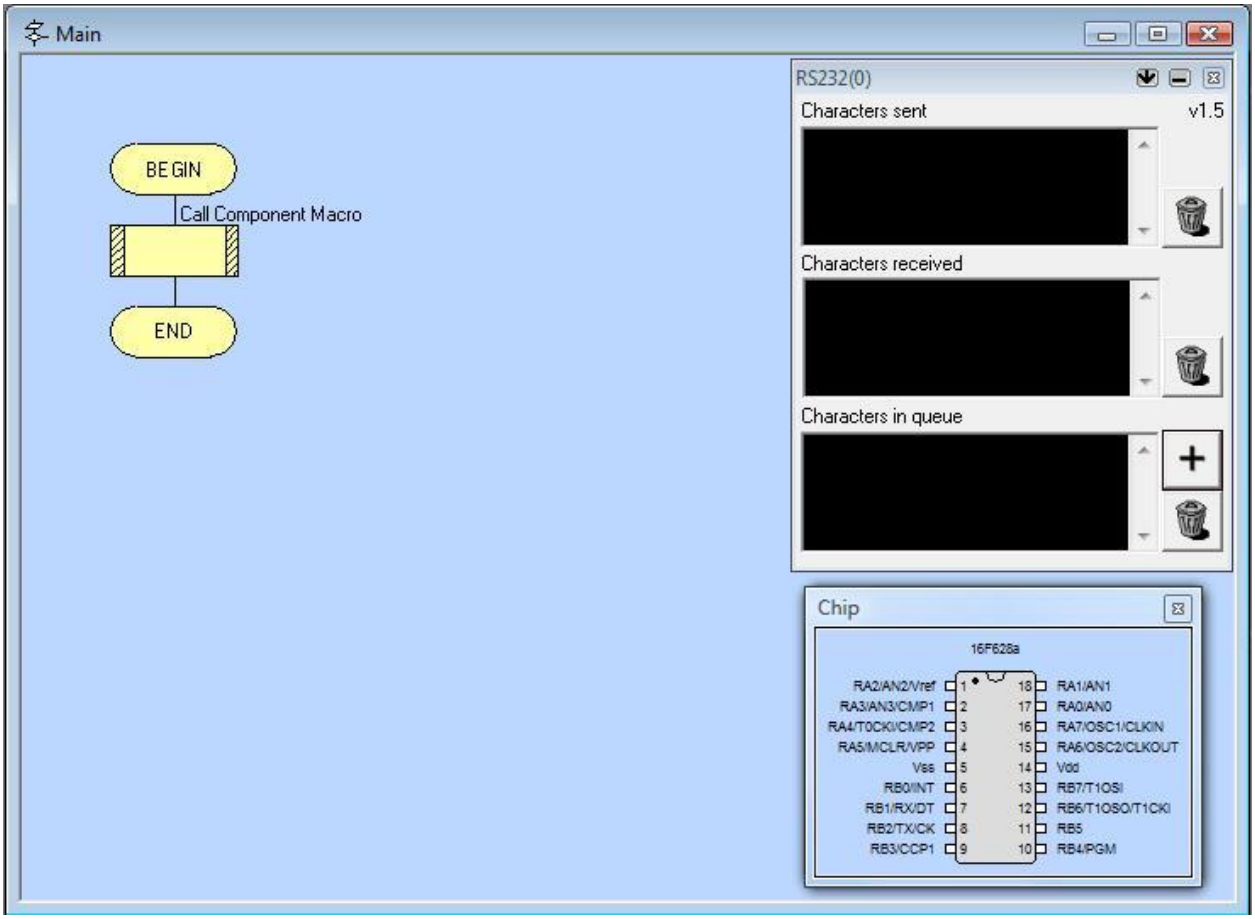


Рис. 3.46. Начало работы с программой, использующей USART

Перед тем, как продолжить создание программы, есть резон решить, а что мы хотим от программы? Пусть будет так: мы получаем по сети литеру N, что устанавливает (в высокое состояние) вывод 0 порта A; мы получаем по сети литеру F, что сбрасывает (устанавливает в состояние низкого уровня) вывод 0 порта A.

Двойной щелчок по **Component Macro** открывает диалоговое окно свойств этого компонента. Выбрав (выделив) получение символа, мы с помощью кнопки ***Variables*** создадим переменную *inp* и зададим параметр 0.

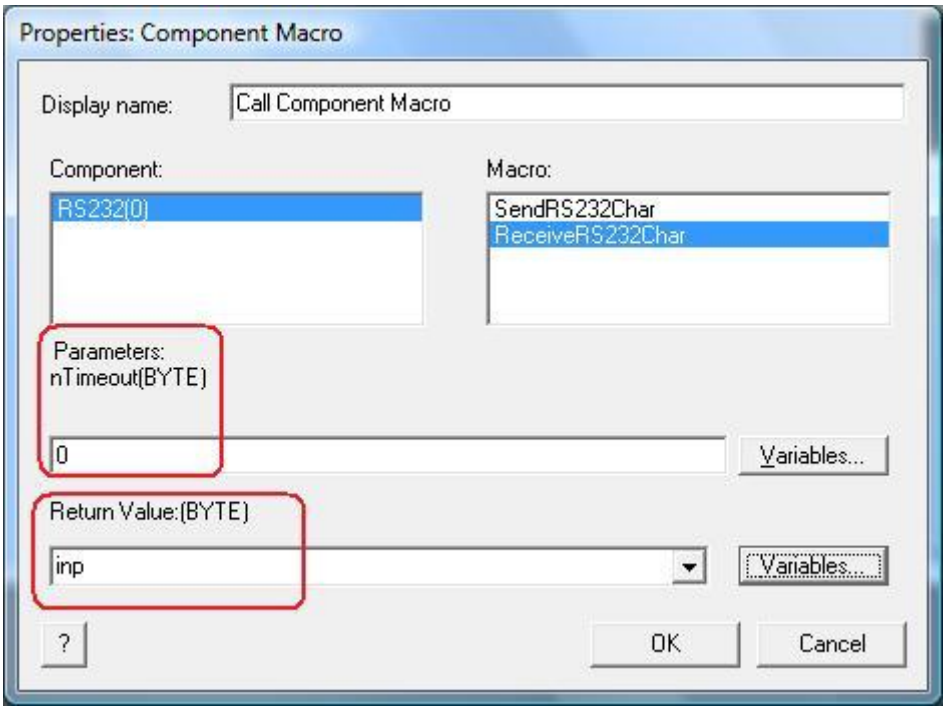
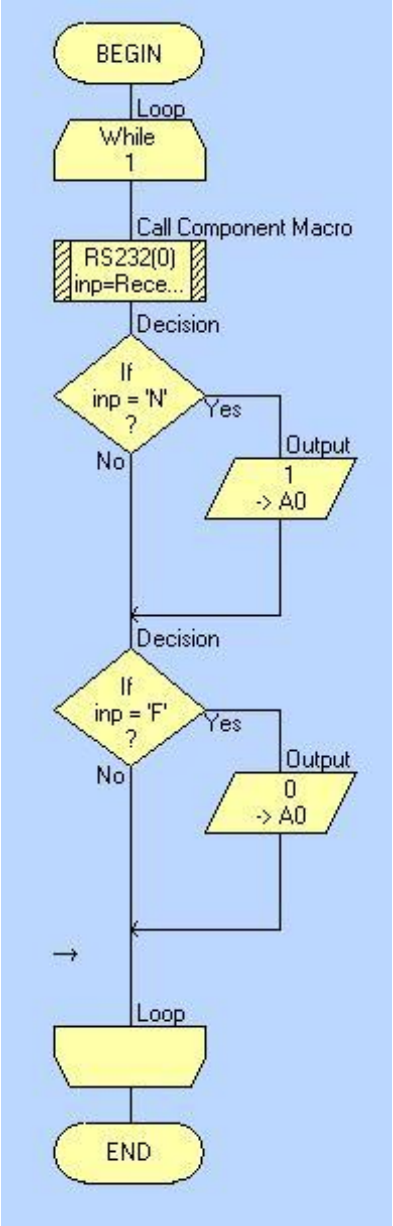


Рис. 3.47. Диалоговое окно **Component Macro** для **RS232**

Задав свойства, «погрузим» всю программу в бесконечный цикл и используем условное ветвление для выполнения ответных действий на приход управляющих символов.

Напомню, что с приходом символа «N» микроконтроллер должен установить высокий уровень на выводе 0 порта A. К этому выводу можно подключить, например, реле, которое будет своими контактами включать свет; можно подключить, например, светодиод оптопары для управления триаком, и в этом случае обойтись без реле. С приходом же символа «F» микроконтроллер должен установить низкий уровень на выводе, выключая нагрузку.

Об оптопаре я упомянул по той причине, что для отладки удобно использовать линейку светодиодов, а начинающим, привыкшим понимать все буквально, бывает трудно отрешиться от мысли, что они что-то недопонимают: написано, что микроконтроллер должен включить свет, а весь свет от светодиода. Кстати, с появлением новых типов светодиодов и это не исключено.



Подпрограмма **Component Macro**, принимающая данные по протоколу RS232, возвращает переменную *inp*.

Первое ветвление программы происходит в зависимости от того, какое значение принимает эта переменная. Если она равна символу «N», то в регистр порта A записывается 1, иначе программа проходит дальше.

Второе ветвление происходит тогда, когда переменная *inp* принимает значение «F». В этом случае в регистр порта A записывается 0.

Цикл *While* в данном случае не имеет определенного условия выхода, то есть, становится бесконечно выполняемым.

Рис. 3.48. Программа отклика на приход символов по сети

Чтобы проверить работу программы, добавив с панели дополнительных компонентов линейку светодиодов, следует использовать кнопку «***+***», отмеченную на рисунке ниже, после запуска программы (**Run->Go/Continue** основного меню, или функциональная клавиша F5 клавиатуры, или кнопка ***Run*** инструментальной панели).

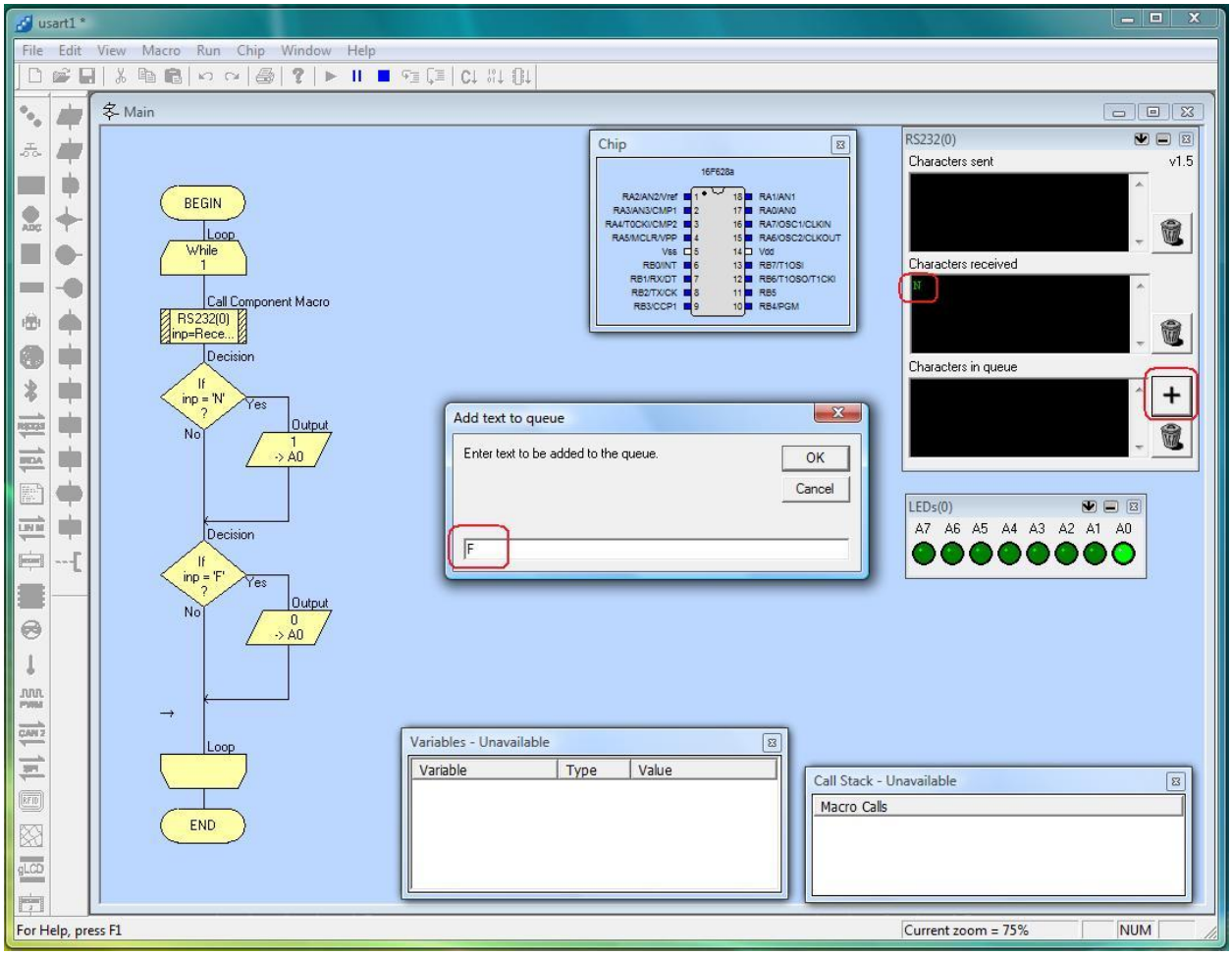


Рис. 3.49. Отладка программы

Нажав кнопку «***+***» панели *Characters in queue*, вы попадаете в диалог ввода символа. После ввода с клавиатуры нужного символа, что подтверждается кнопкой ***OK*** диалога ввода, этот символ появится в окне панели *Characters Received*, а светодиод A0 загорится. Если же повторить эту операцию, введя символ «F», то светодиод погаснет.

Создадим еще одну программу, которая не получает символы по USART, а отправляет их.

Программа простая: при каждом нажатии на кнопку микроконтроллер попеременно отправляет в сеть символы «N» и «F».

Мы знаем, что для обслуживания кнопки, соединенной с выводом порта, назначенным на вход, следует использовать переменную. Выберем переменную *inp*, которая будет обслуживать нулевой вывод порта B. Чтобы организовать переключение, используем переменную *flag*, которая будет менять свое значение с 0 на 1 при каждом нажатии на кнопку. И для отправки символа используем переменную *out*. Эти переменные можно сразу ввести в программу, но можно создавать их по мере продвижения в написании программы.

Итак. Создав новую программу, сразу добавим в нее цикл (программный компонент **Loop**). Но перед циклом, используя программный компонент **Calculation**, зададим переменной *flag* нулевое значение. Внутри бесконечного цикла будем (с помощью программного компонента **Input**)

опрашивать вход B0 и разветвим программу: ничего не будем делать, если состояние кнопки не изменилось, и будем делать «все остальное», если кнопка была нажата.

Что «остальное»? В первую очередь будем менять значение переменной *flag*: используем такую конструкцию из двух логических операций `flag = ( NOT flag ) AND 1`, которую запишем с помощью программного компонента **Calculation**. Если бы в программе FlowCode был булев тип данных, было бы достаточно операции инверсии (отрицания `NOT`), но нам не важно, с помощью скольких операций мы выполним задуманное. Тем более что можно было бы использовать и запись `flag = NOT flag`, только в этом случае переменная приобретала бы два значения: 0 и 255.

И нам остается только отправлять символ «N», если переменная *flag* равна 1, и символ «F», когда она становится равна нулю.

Как и в предыдущей программе, мы используем программный элемент **Component Macro** для RS232. Но на этот раз в его свойствах выберем отправку символа и используем переменную *out*.

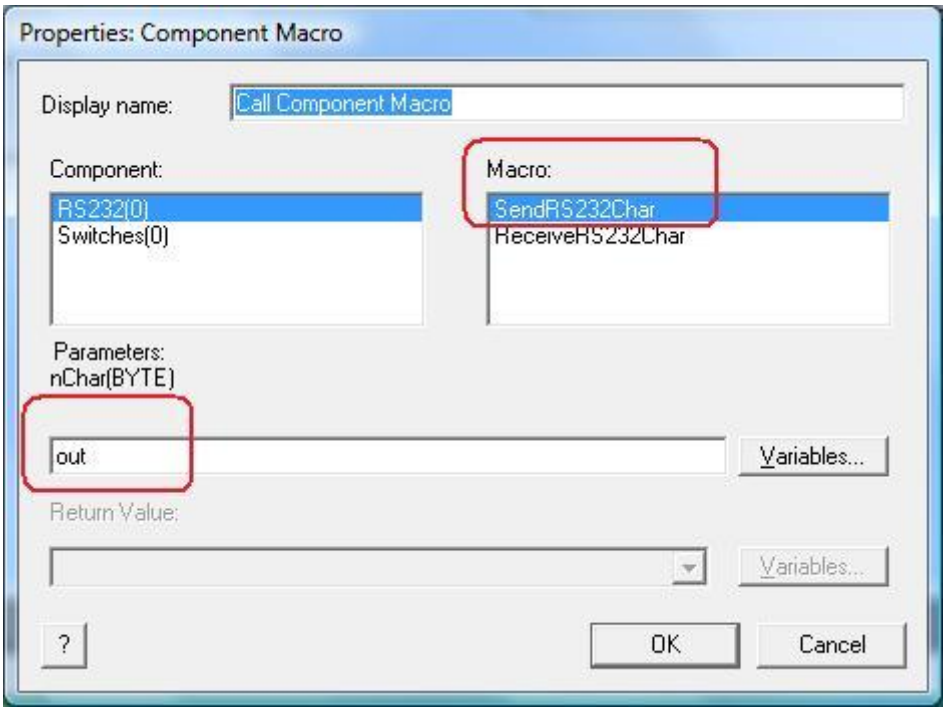


Рис. 3.50. Настройка компонента RS232 в диалоговом окне его свойств

Если запустить отладку программы в том виде, который она сейчас приняла, то… попробуйте, сами увидите, что произойдет. А, чтобы избежать этого, добавим паузу после каждой отправки символа.

Программа приобретает такой вид:

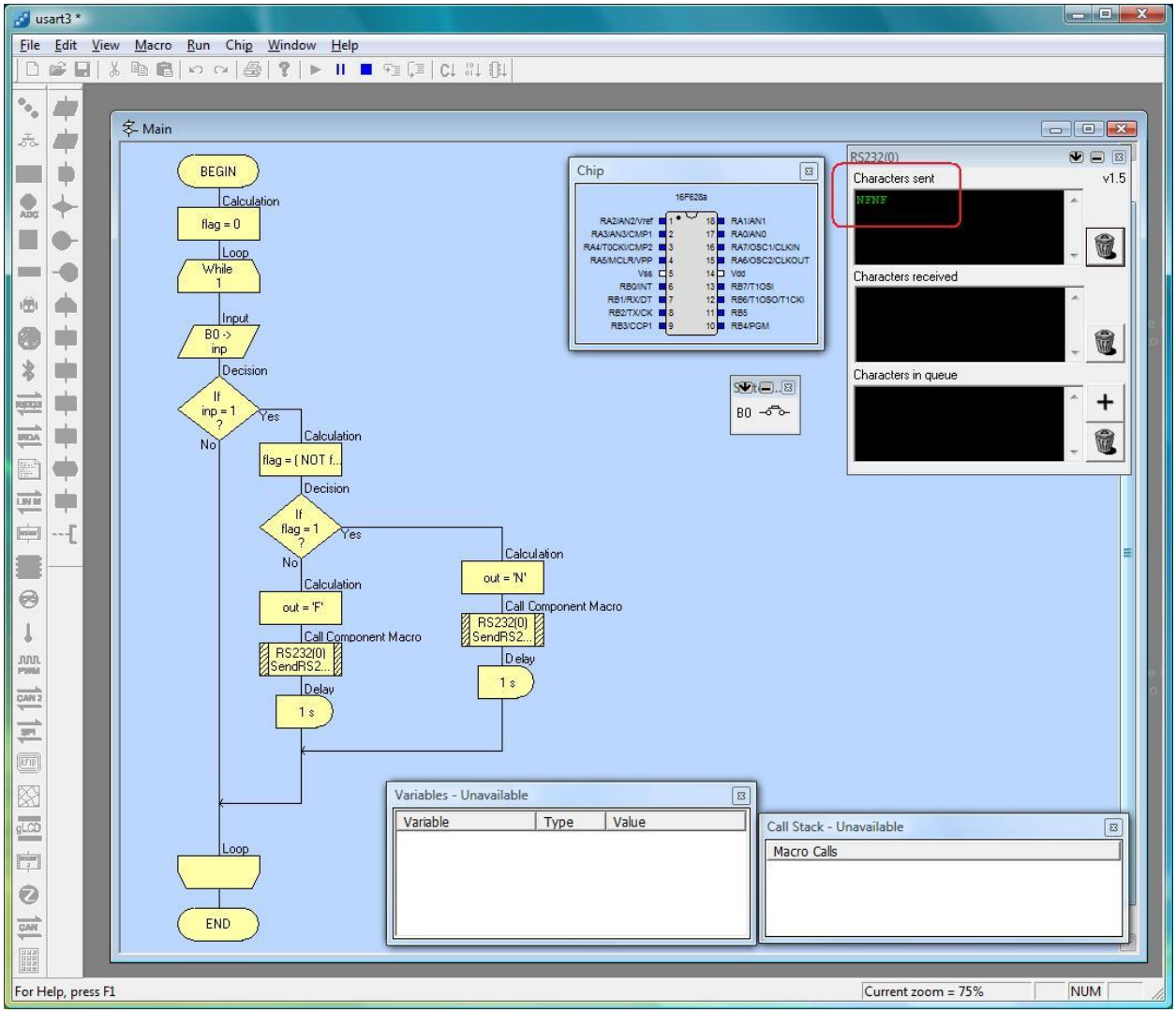


Рис. 3.51. Вторая программа, использующая USART

На рисунке отмечено, какие символы отправляются в сеть при каждом нажатии на кнопку B0.

Я полагаю, что, прочитав предыдущие главы, вы уже вполне можете справиться с написанием программы на языке Си и без использования программы FlowCode. Хотя согласитесь, что использовать ее очень удобно.

Оставим пока эти две простенькие программы, о которых хотелось бы добавить еще несколько слов, но позже. А сейчас несколько слов о модуле PWM, который есть в PIC16F628A. Модуль может использоваться для разных целей. Но я хочу привести пример программы, где меняется скважность импульсов на выходе 3 порта B, с тем, чтобы показать, как просто это можно сделать в программе FlowCode.

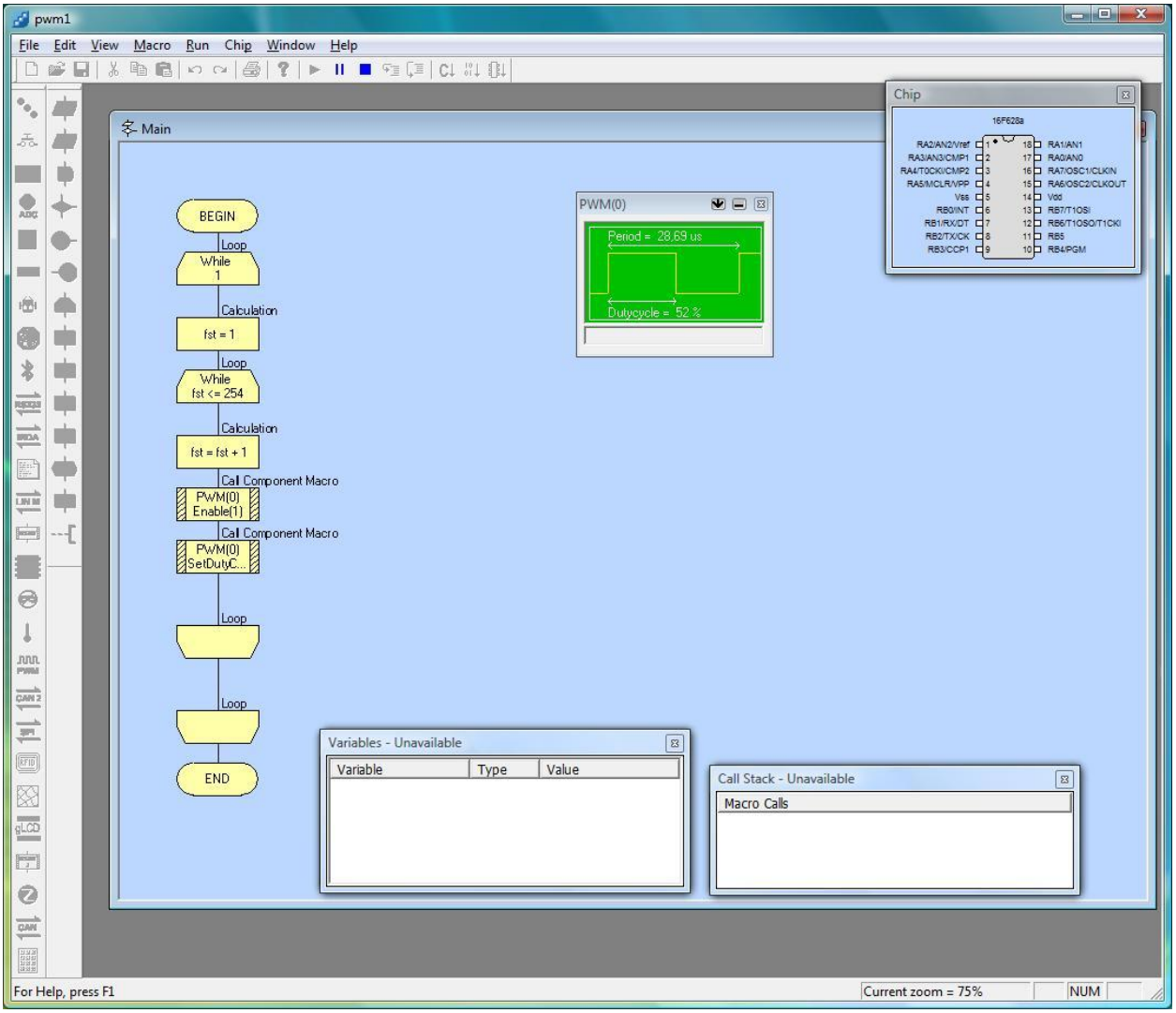
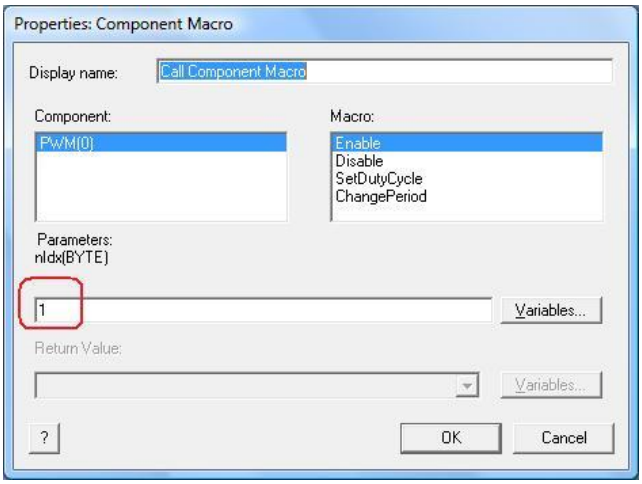


Рис. 3.52. Программа работы с ШИМ (широтно-импульсной модуляцией)

Вся программа, кроме двух циклов: бесконечного внешнего и условного внутреннего, – сводится к добавлению двух программных элементов **Component Macro** и компонента **Calculation**, где переменная *fst* увеличивается на единицу при каждом проходе внутреннего цикла. Первый **Component Macro** разрешает использование ШИМ.



Отмеченная на рисунке единица относится к номеру канала, поскольку есть модели, имеющие несколько каналов ШИМ.

Рис. 3.53. Диалоговое окно дополнительного компонента PWM

В следующем диалоговом окне (второго **Component Macro**) задается скважность импульсов.

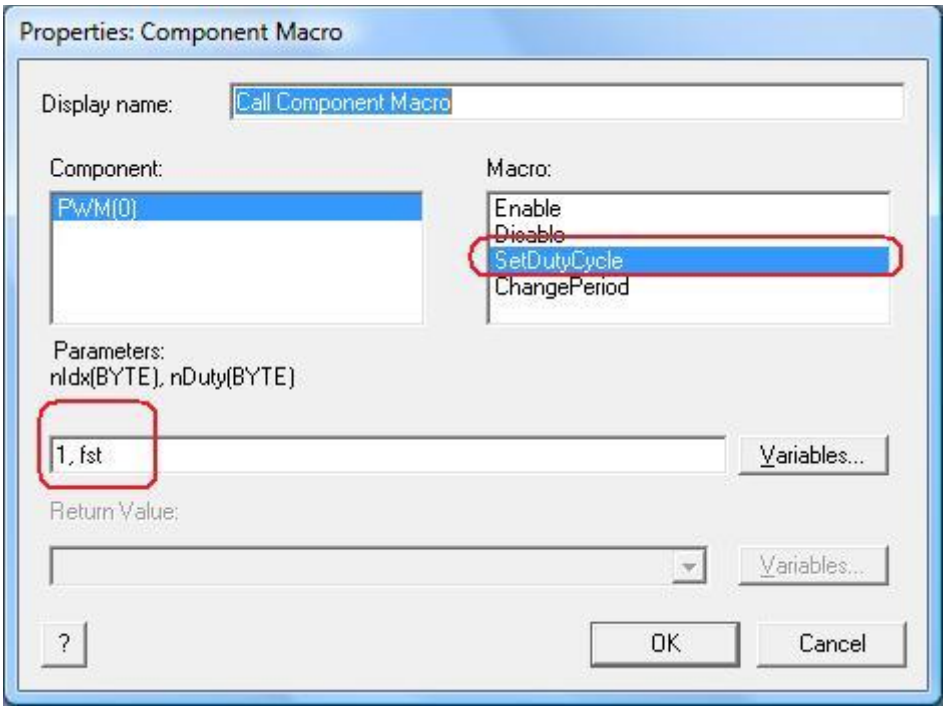


Рис. 3.54. Задание скважности импульсов через переменную *fst*

После запуска программы в окне компонента появится сигнал, плавно меняющий скважность импульса.

Таким образом, полная версия программы FlowCode позволяет быстро и эффективно разрабатывать устройства, базирующиеся на микроконтроллерах. Помимо этого можно проекты, разработанные для PIC-контроллера, импортировать в версию, предназначенную для работы с AVR-контроллерами, что еще больше расширяет возможности применения программы.

Однако это несколько уводит нас от основной темы рассказа – как использовать демо-версию FlowCode, чтобы научиться программированию на языке Си (или ассемблере). Вернемся к теме рассказа и рассмотрим несколько конкретных примеров.

Некоторые примеры программирования в среде FlowCode

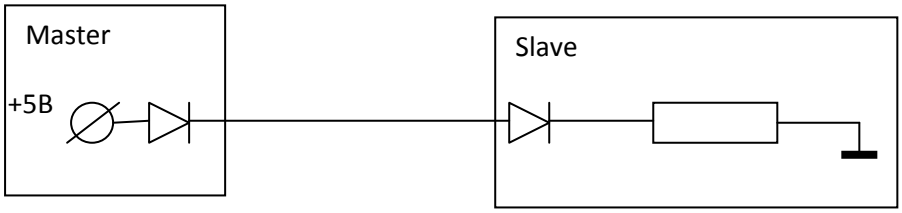
Первый пример

Первый пример прост, но полученное устройство вполне может оказаться полезным, когда покупка готового прибора по каким-то причинам не целесообразна.

Сегодня часто используют кабель из нескольких витых пар, который заканчивается разъемом RJ45. Разъем «обжимается» быстро, но всегда полезно проверить результат. Конечно, для устройства «прозвонки» проводов не обязательно использовать микроконтроллер. Но посмотрите, как просто сделать устройство, используя микроконтроллер.

Итак, сформулируем задачу: необходимо «прозвонить» кабель, в котором восемь проводов. Для индикации используем светодиоды, выбрав модель с падением напряжения 1.5-2 В на диоде. Конструктивно устройство будет состоять из двух блоков: основной блок (Master) и вспомогательный (Slave). Оба блока снабжены гнездами для разъема RJ45 и линейкой светодиодов. При включении прибора светодиоды последовательно зажигаются на обоих блоках при исправных проводах и соединении. Если провода перепутаны, то это должно отобразиться в последовательности зажигания светодиодов или в отсутствии свечения.

Немного о схеме. На первый взгляд схема крайне проста:



Но есть один «подводный камень»: для самого простого решения по «прозвонке» восьми проводов не хватает девятого, общего провода. В этом случае микроконтроллер окажет неоценимую услугу, поскольку можно модифицировать включение светодиодов следующим образом:

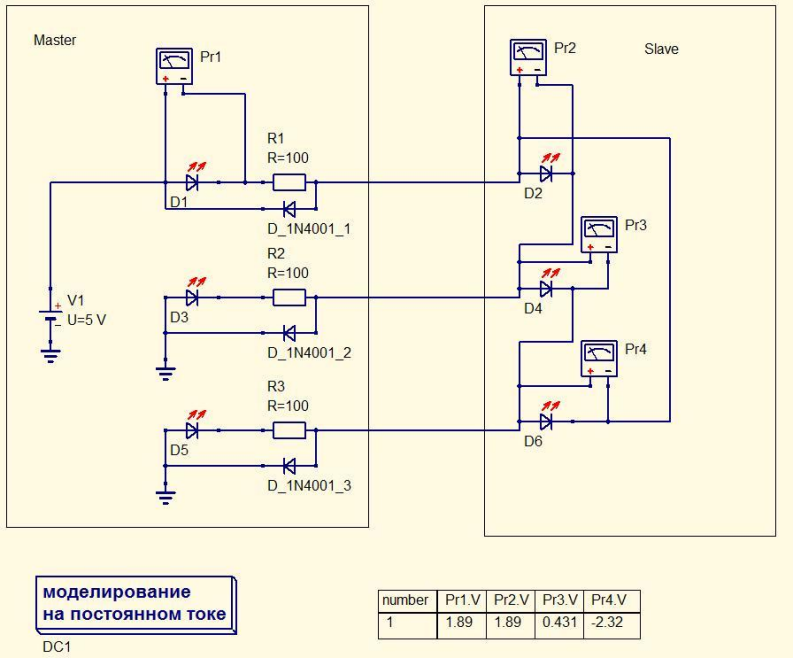
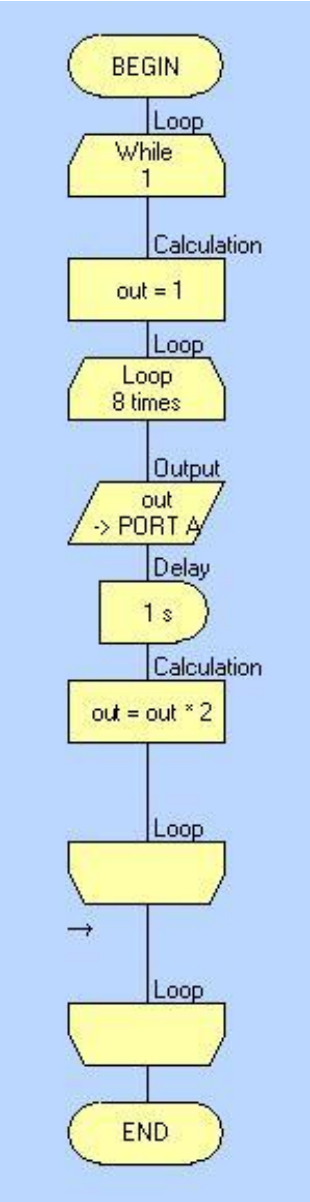


Рис. 4.1. Модификация схемы для трех проводов

Если исключить из схемы выше (для трех проводов) приборы Pr1-4, то смысл схемы ясен, а роль источника питания V1 будет играть микроконтроллер. И программа, обслуживающая работу микроконтроллера, тоже проста: на один из выходов микроконтроллера подается высокий уровень, а на остальные низкий. Вот вид программы в FlowCode:



В бесконечном цикле (первый компонент **Loop**) мы восемь раз (второй компонент **Loop**) отправляем в порт А число, которое начинается с 1 (первый компонент **Calculation**), а после паузы в 1 секунду умножается на 2 (второй компонент **Calculation**).

Для перевода программы на язык Си у нас все есть: вывод числа, цикл. Даже с паузой при использовании компилятора HI-TECH мы разобрались. Единственное отличие от алгоритма, записанного в FlowCode – двухступенчатое получение нужной длительности паузы.

Напишем так:

```
#include <htc.h>
#define _XTAL_FREQ 4000000
void main()
{
char out;
char i;
char k;
CMCON = 0x07;
      while (1) {
            out = 1;
            for (i=0;i<8;i++) {
                  TRISA = 0x00;
                  PORTA = out;
                  for(k=0;k<10;k++) {
                  __delay_ms(100);
                  }
                  out = out*2;
            }
      }
}
```

Рис. 4.2. Программа обслуживания «прозвонки»

Теперь эту программу можно использовать в среде разработки MPLAB: проверить, оттранслировать.

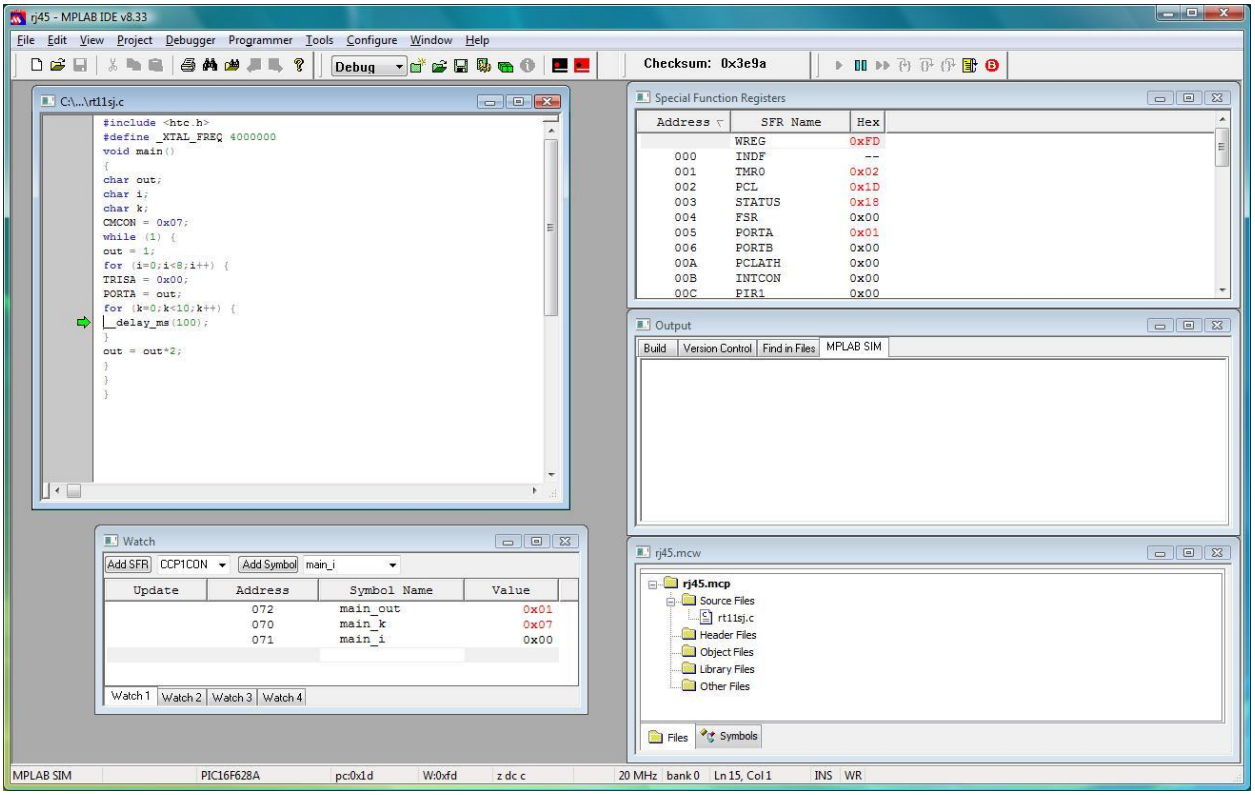


Рис. 4.3. Поверка программы в среде MPLAB

Или можно использовать для проверки работы схемы другую программу.

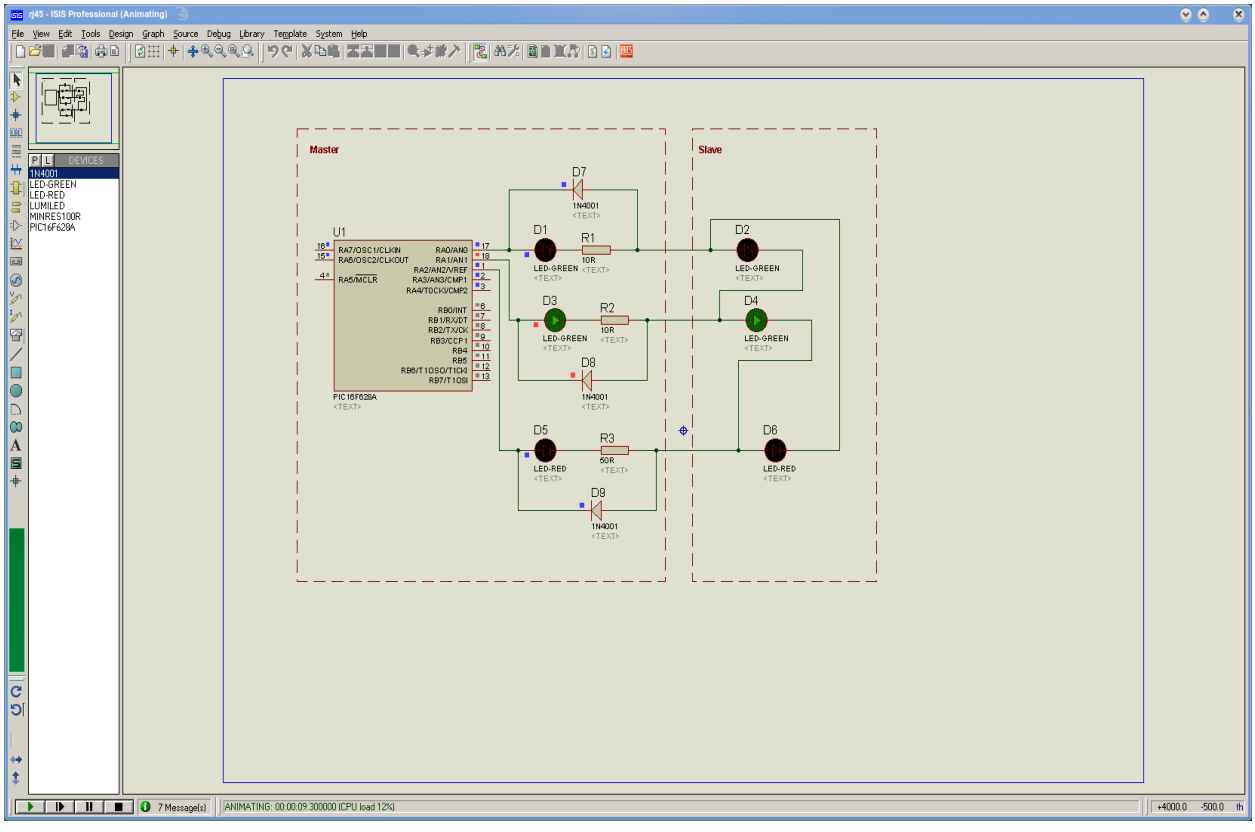


Рис. 4.4. Проверка схемы в Proteus

Или проверить работу схемы на макетной плате, что даст самый убедительный результат. Нет, не в проверке работы программы, а всего устройства в целом. Например, даст ответ на вопрос, удобно ли работать с устройством, если один или несколько проводов в обрыве, если один или несколько проводов перепутаны и т.п.

Многие любители, особенно «бывалые», пренебрежительно относятся к простым программам и инструментам, облегчающим создание устройств на базе микроконтроллеров. Между тем, сложность конструкции плохо соотносится с ее полезностью, и имеет смысл только тогда, когда без этой сложности не обойтись.

Не могу сказать, нужно ли кому-нибудь устройство, описанное выше. Но мне хотелось показать, что, зная несколько простых фрагментов на языке Си, полученных с помощью демо-версии программы FlowCode, можно создавать вполне реальные, жизнеспособные устройства. И это главное.

Второй пример

Этот пример построим на основе первого: добавим к устройству одну кнопку, нажатием которой ускорим обход всех проводов. Если прежде каждый светодиод был включен в течение секунды, то при нажатой кнопке он будет включаться только на одну миллисекунду. Что при этом изменится в работе устройства? Внешне это выразится в том, что визуально все светодиоды будут гореть одновременно, но слабее, чем при отжатой кнопке.

Вспомним, как выглядит фрагмент кода на языке Си для ввода:

```
char FCV_IN;
void main()
{
        //Input: B0 -> in
        trisb = trisb | 0x01;
        FCV_IN = (portb & 0x01);
}
```

Для ввода (программный компонент Input) понадобится переменная, в данном случае `char FCV_IN`. Для ввода используется единственный вывод RB0 порта B, что отражено в строке `trisb = trisb | 0x01`. Переменной, обслуживающей ввод, присваивается значение, прочитанное в RB0.

И вспомним, как выглядит условное ветвление:

```
 if (FCV_IN == 1)
       {
             // Некое действие

       } else {
             // Другое действие
       }
```

Вся программа в среде разработки FlowCode имеет вид:

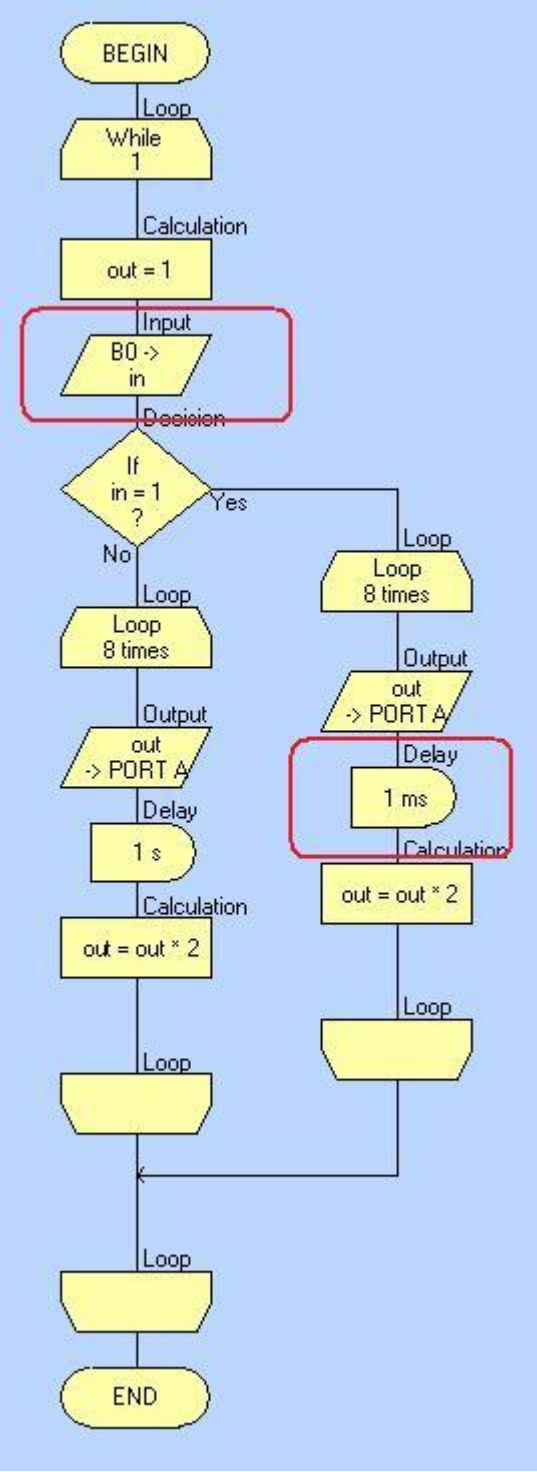


Рис. 4.5. Модификация предыдущей программы

Кроме добавления компонента **Input**, свойства которого соответствующим образом изменены, и компонента ветвления **Decision**, все остальное сводится к копированию предыдущей программы во вторую ветку и изменению длительности паузы.

Думаю, вам интереснее самостоятельно написать эту программу на языке Си и отладить ее в программе MPLAB.

Но несколько слов о смысле, который можно придать новой программе, вернее, модификации старой. Когда светодиоды последовательно зажигались и горели секунду, легко заметить обрыв одного из проводов. Но, если разъем обжат плохо: соединение есть, а переходное сопротивление достаточно большое, – то при использовании первого режима работы устройства это не будет бросаться в глаза. А когда все светодиоды горят, они горят с одинаковой яркостью, если все в порядке. Если же один (или несколько) проводов имеют плохой контакт с выводом разъема, соответствующий светодиод (при выбранной конструкции два светодиода) будет светиться слабее остальных. Что сразу бросится в глаза.

Конечно, справедливость этой идеи следует проверить на макетной плате, но мне хотелось показать совсем другое – то, как легко, практически без переделок схемы, можно расширить функциональность устройства.

И что для этого можно использовать простые фрагменты языка Си, полученные в FlowCode.

Пример третий

Рассказывая о встроенных в микроконтроллер модулях, я предложил рассмотреть простое устройство, принимающее по сети команду и выполняющее простое действие – включение реле, например. И позже я собирался добавить несколько слов. Вот это позже и настало.

Полнофункциональная версия FlowCode позволяет быстро добавить работу с такими модулями, как USART или PWM. Не так сложно, посмотрев пример, который устанавливается с компилятором HI-TECH, понять, как следует на языке Си написать эту часть работы контроллера. Программа и в этом случае получается «простенькой».

Однако, вспоминая как «подвисали» компьютерные концентраторы в нашей сети, когда достаточно было выключить его и включить вновь, я думаю, что применение устройства, созданного из «простенькой» программы для выполнения этого «обряда» не на месте, а нажатием одной кнопки в региональном офисе, было бы весьма полезно. Или, скажем, свет в подъезде, который горит всю ночь. Программа, о которой идет речь, выполняло простое действие – нажатием кнопки в сеть отправлялась команда, которая включала, положим, свет. Любой электрик нарисует вам эту схему без применения контроллеров. Но реализуйте эту схему не для одной лампочки, а для лампочек всего подъезда, да еще так, чтобы можно было выключить свет на своем этаже. Здесь преимущества «простенькой» программы очевидны.

А говорю я об этом по той причине, что хочу привести пример еще одной «простенькой», но полезной программы для микроконтроллера.

Очень часто устройство на базе микроконтроллера имеет клавиатуру. С целью экономии выводов кнопки можно включить «матрицей». При четырех кнопках, как на рисунке ниже, мы не получаем существенного выигрыша, но, используя восемь выводов, можно получить клавиатуру с шестнадцатью клавишами. И здесь выигрыш очевиден.

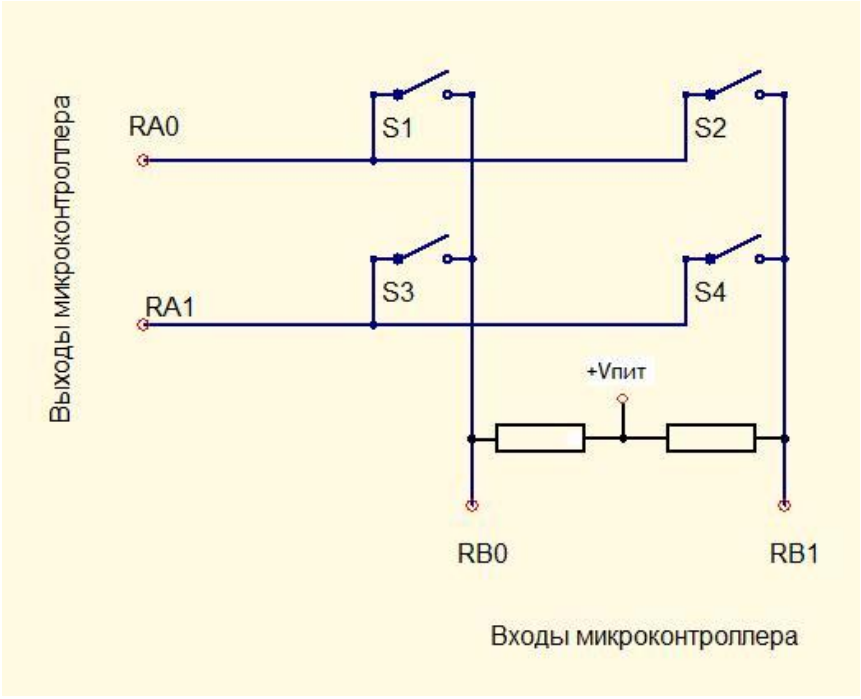
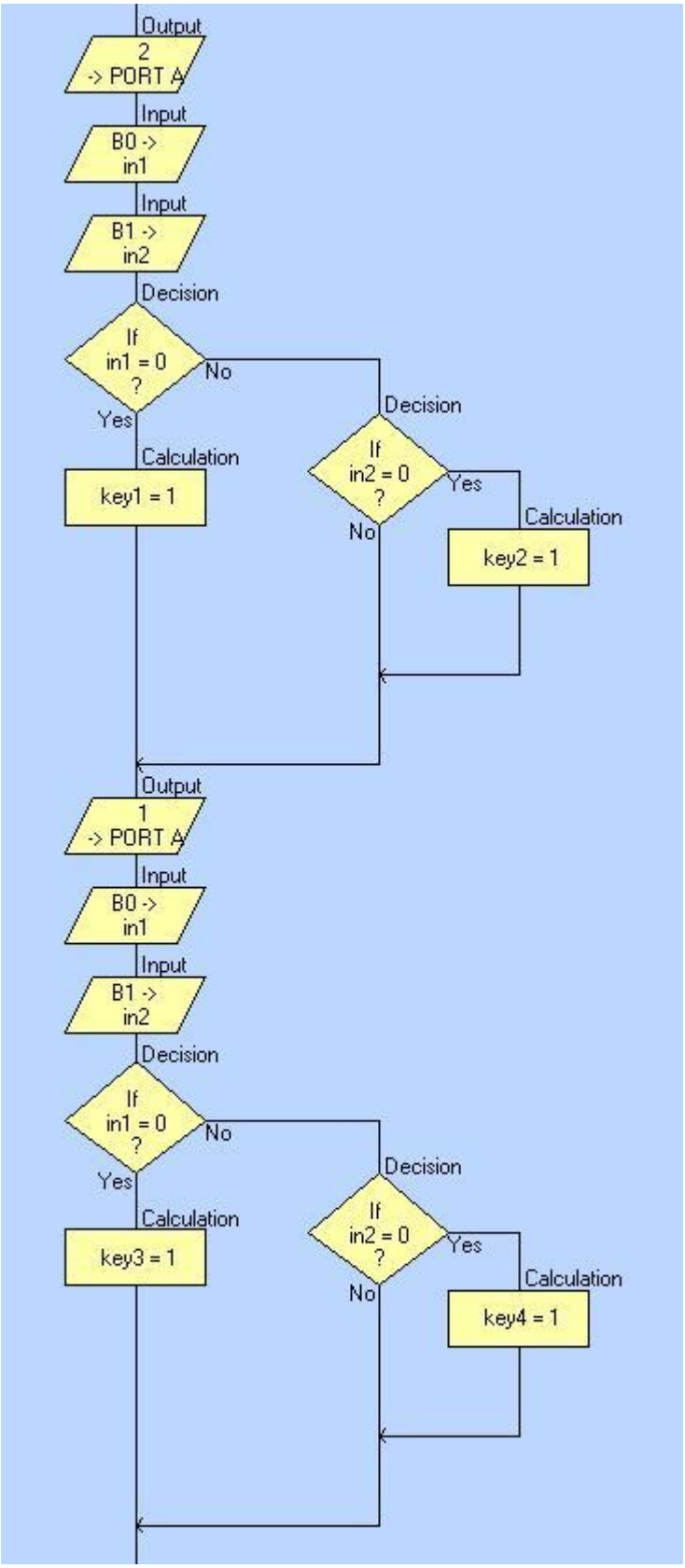


Рис. 4.6. «Матричное» включение четырех клавиш

Попеременно подавая низкий уровень на выходы микроконтроллера RA0 и RA, определяя состояние входов RB0 и RB1, можно определить состояние четырех кнопок S1-S4. Для реализации программы достаточно уже известных фрагментов на языке Си. «Блок-схема» программы может выглядеть так:



«За кадром» остался бесконечный цикл, в котором работает опрос клавиатуры. Программа может быть построена и иначе. Например, с использованием подпрограмм.

Результат работы не заметен – переменные программы key1 - key4 приобретают некоторое значение, которое невозможно проверить на макетной плате.

Более того, программу трудно проверить в FlowCode из-за особенностей свойств компонента выключатели.

Но если заменить присваивание значений переменным на изменение состояния выходов, то работу программы можно проверить на макетной плате.

Реальная программа, конечно, выполняет какие-то действия по нажатию кнопок, что позволяет проверить ее работу.

Тем не менее, будет полезно записать программу на языке Си и постараться проверить ее работу в MPLAB или Proteus.

В полной версии FlowCode работает дополнительный компонент KeyPad. Для работы с клавиатурой можно использовать этот компонент.

Рис. 4.7. «Блок-схема» программы обслуживания клавиатуры

Как и другие компоненты, **KeyPad** добавляется в программу щелчком по кнопке инструментальной панели дополнительных компонентов. После этого достаточно добавить программный компонент **Component Macro**, в диалоговом окне свойств которого выбрать, например, в качестве возвращаемого значения номер клавиши.

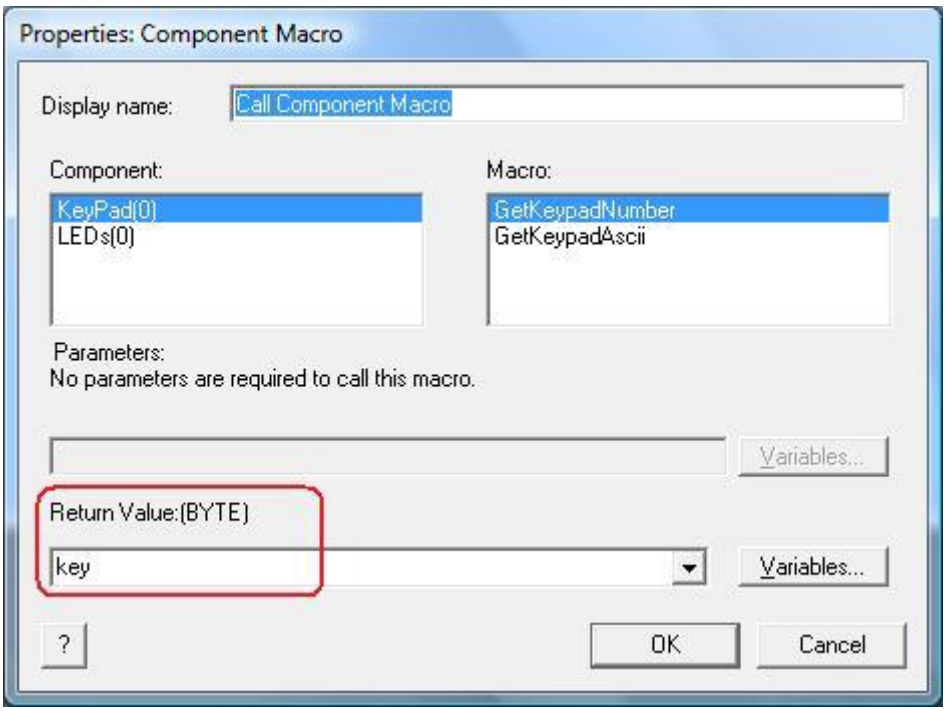


Рис. 4.8. Диалоговое окно программного компонента для **KeyPad**

Этот номер будет присваиваться созданной нами переменной *key*. Дальнейшие действия зависят от наших намерений. Простейшая программа проверки клавиатуры может иметь следующий вид:

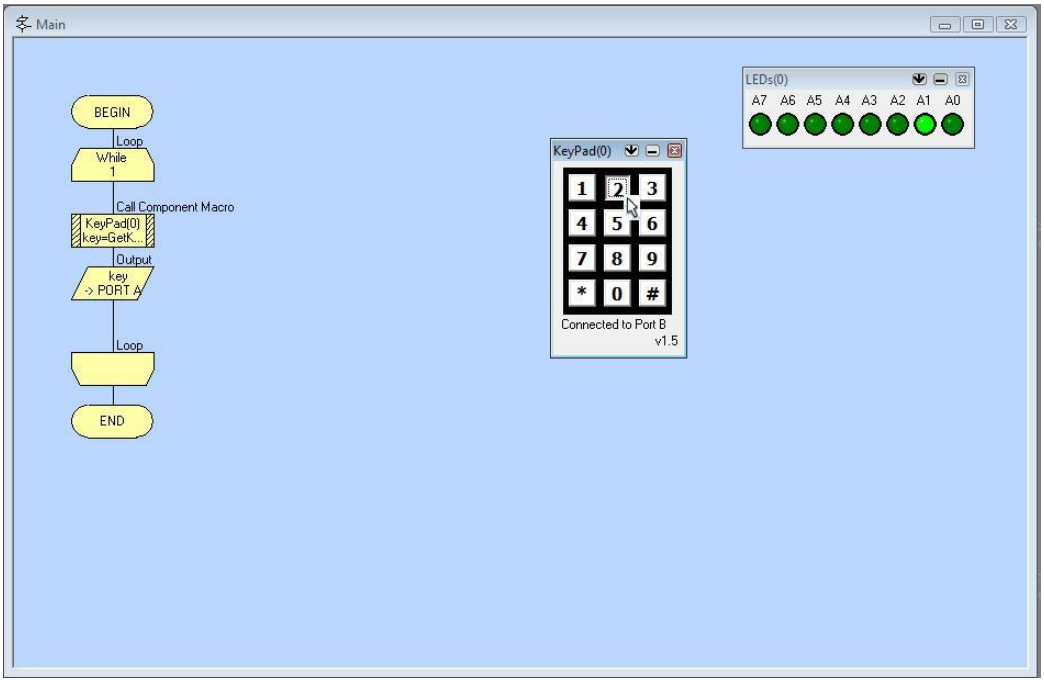


Рис. 4.9. Работа клавиатуры в программе FlowCode

Пример четвертый

Много лет назад мой знакомый спросил, его тоже кто-то попросил об этом, как удобнее сосчитать количество посетителей. По тем временам микроконтроллеры были достаточно дороги, да и требовали много из того, что не у всякого любителя найдется. Но сегодня я точно порекомендовал бы использовать микроконтроллер для создания подобного счетчика.

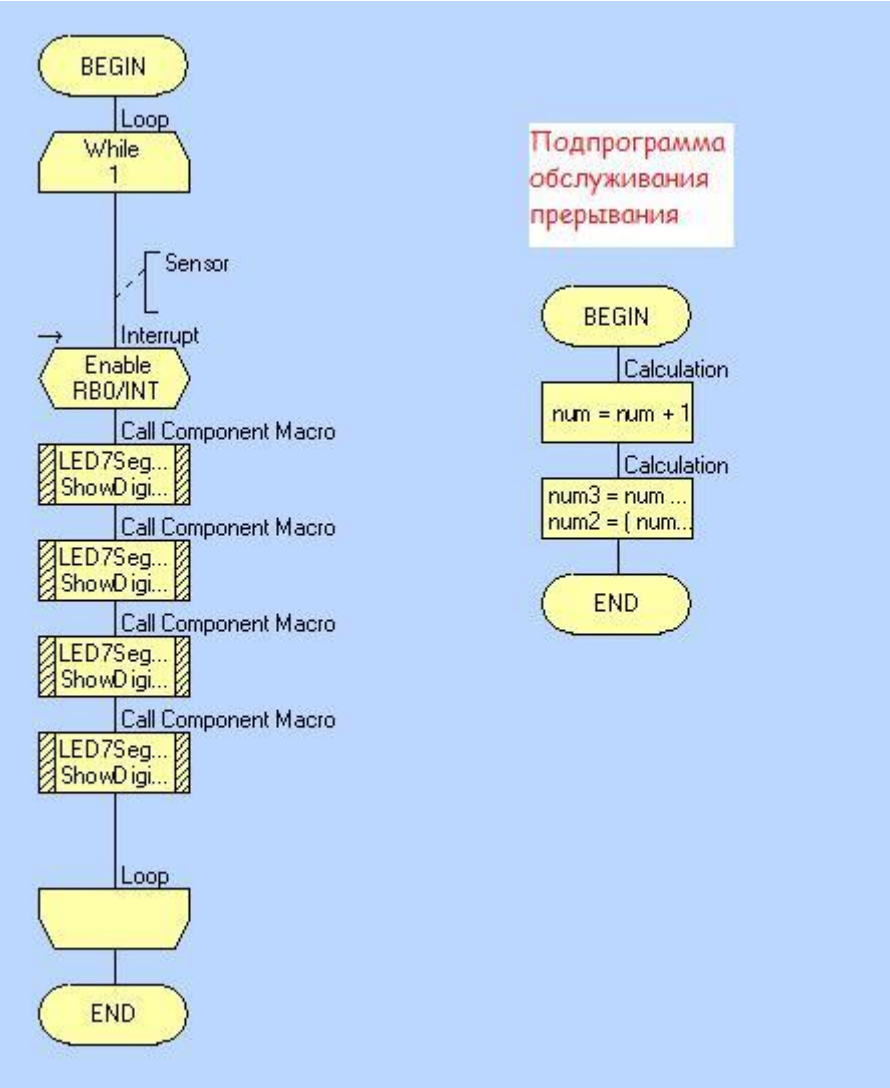


Рис. 4.10. Программа счета срабатывания датчика на входе RB0

Программа FlowCode предлагает четырехразрядный семисегментный индикатор в качестве одного из дополнительных компонентов. Открыв свойства вызова подпрограммы обслуживания этого индикатора, можно увидеть:

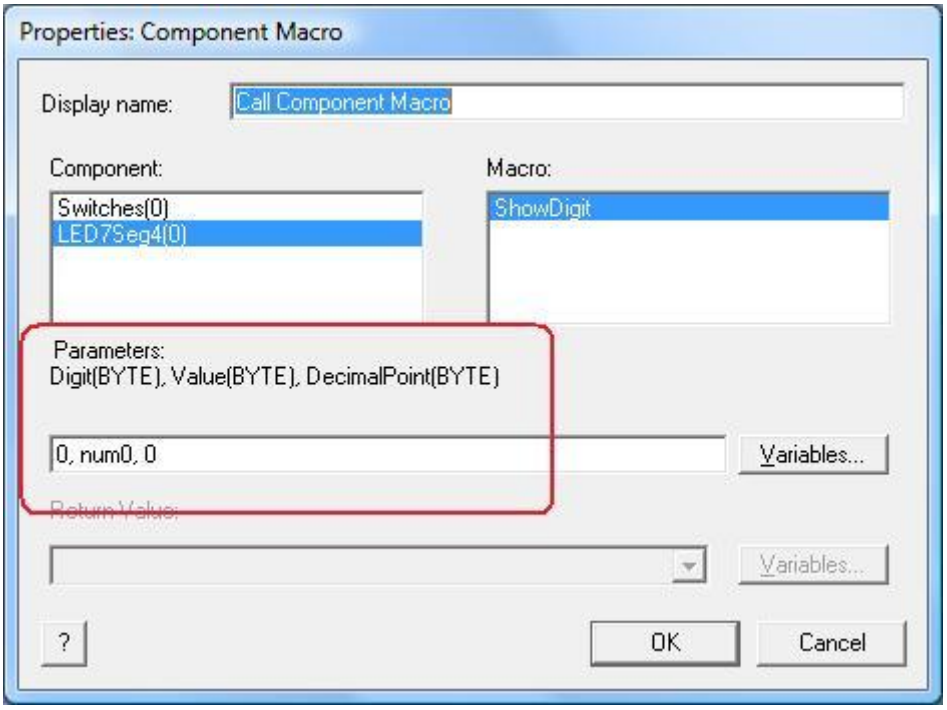


Рис. 4.11. Диалоговое окно свойств подпрограммы индикатора

Первый параметр – номер разряда, второй – значение, третий – положение точки.

При использовании простейшего герконового датчика, установленного на двери, вся тяжесть работы ляжет на создание подходящего корпуса устройства. Но, индикаторы бывают двух типов: с общим анодом, что представлено в программе FlowCode, и с общим катодом. Программа FlowCode, похоже, не позволит применить такой индикатор. В этом случае знания, полученные при изучении языка Си с помощью программы FlowCode, позволят вам быстро преобразовать готовый код к нужному виду – заменить высокий уровень на общих анодах на низкий, а низкий уровень сегментов высоким.

Фрагмент, разрешающий прерывание, выглядит так:

```
 //Interrupt initialisation code
 option_reg = 0xC0;

//Interrupt: Enable RB0/INT
       option_reg.INTEDG=1;
       intcon.GIE=1;
       intcon.INTE=1;
```

Прерывание:

```
void interrupt(void)
{
 if (intcon & (1 << INTF))
 {
       FCM_display();
       clear_bit(intcon, INTF);
 }
}
```

А функция `FCM_display()` в обслуживании прерывания:

```
void FCM_display()
{
    //Calculation:  num = num + 1
    FCV_NUM = FCV_NUM + 1;

    //Calculation:
    //  num3 = num / 1000
    //  num2 = (num - num3 * 1000 ) / 100
    //  num1 = (num - num3 * 1000 - num2 * 100 ) / 10
    //  num0 = (num - num3 * 1000 - num2 * 100 - num1 * 10 )
    FCV_NUM3 = FCV_NUM / 1000;
    FCV_NUM2 = (FCV_NUM - FCV_NUM3 * 1000 ) / 100;
    FCV_NUM1 = (FCV_NUM - FCV_NUM3 * 1000 - FCV_NUM2 * 100 ) / 10;
    FCV_NUM0 = (FCV_NUM - FCV_NUM3 * 1000 - FCV_NUM2 * 100 - FCV_NUM1 * 10);
}
```

И заметьте, если вы поспешили и сделали печатную плату, то вам не понадобится вносить изменения в монтаж, достаточно изменить программу. А если вы еще раз поспешили и сделали вторую печатную плату, то немного переделав программу, вы можете использовать устройство, например, для велосипеда. Конечно, датчик, закрепленный на колесе, потребуется, скорее всего, другого типа, но посчитывая количество оборотов колеса, зная длину его окружности, вы можете отображать на индикаторе пройденный путь. Или, еще раз изменив программу, можете отображать скорость движения. И ничто не помешает вам добавить одну кнопку для переключения этих режимов работы. Само устройство остается прежним. В этом проявляется одно из самых важных свойств микроконтроллера – меняя программу, устройство можно превратить в другое, не менее полезное.

В этом примере можно отметить еще две важные детали – полная версия программы FlowCode позволяет работать быстрее, чем другие среды разработки; а знание языка программирования Си помогает быстрее и легче справляться с возникающими осложнениями.

В этом смысле трудно переоценить возможности, предоставляемые программой FlowCode – увидеть необходимые действия, которые образуют достаточно понятную конструкцию, а затем уже обратиться к коду на языке программирования. Думаю, любой специалист согласится, что рисунок схемы, использующий «прямоугольник» микросхемы с заданными свойствами, гораздо понятнее, чем схема, где все микросхемы нарисованы с полным внутренним содержанием.

Пятый пример

Или пример, который должен показать, как легко можно создать устройство, используя FlowCode, но как важно, не останавливаясь на достигнутом, потратить некоторое время и приложить усилия к освоению, скажем, языка Си.

Сегодня в каждом доме есть хотя бы один пульт управления, использующий инфракрасное излучение. Разработчики телевизоров, музыкальных центров, домашних кинотеатров и т.п. прилагают немалые усилия к тому, чтобы их ИК-пульт обладал уникальностью. Иначе, переключая телевизионные каналы, вы можете заставить кондиционер «нагнать» такого холода, что «заморозите» картинку на экране.

Однако в любительской практике, где управление с помощью инфракрасного излучения достаточно привлекательно, нет нужды излишне усложнять код, излучаемый пультом. Создадим такой пульт на базе МК, максимально упростив задачу: она должна быть легко обозримой, поскольку усложнить ее всегда можно простым повторением уже полученных фрагментов программы. Но вначале посмотрим на один из вариантов организации управляющего ИК кода.

У начинающих порой вызывает недоумение наличие несущей частоты, которая может меняться от нескольких килогерц до нескольких сотен килогерц. Нужно ли это усложнение?

Это усложнение в любом случае очень полезно: без несущей частоты код управления будет подвержен воздействию конкурирующих тепловых излучений, воздействующих на приемник. Кроме того, для повышения яркости свечения излучателя приходится увеличивать ток через него, а последний ограничен для обычных светодиодов несколькими десятками миллиампер. Тогда как через излучающий светодиод можно пропустить импульс тока в несколько сотен миллиампер, но при условии, что этот импульс будет коротким, а скважность достаточно большая.

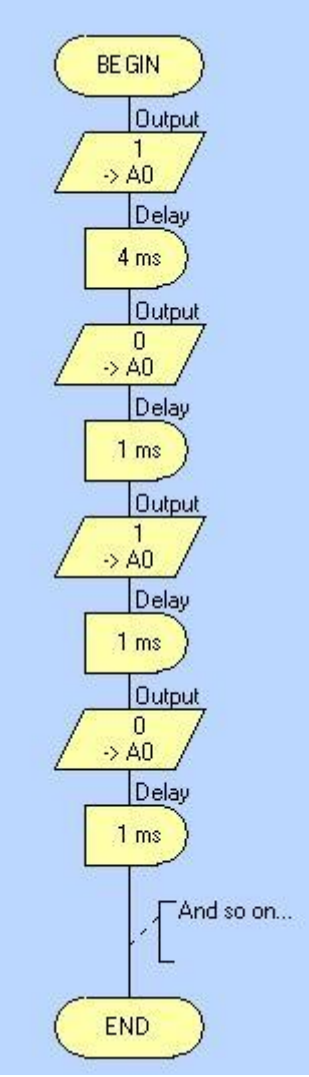
Но оставим пока вопрос о несущей и вернемся к организации управляющего кода. Выберем базовый период *T* для нашего кода. Длительность обязательного заголовка примем равной *4T*. Паузу, разделяющую информационные биты, примем равной по длительности *T*. Для единицы примем длительность импульса равную 2T, а для нуля, как и для паузы, длительность пусть будет равна *T*. Какой быть длительности, выраженной в единицах времени, для базового периода, мы определим позже, а пока примем ее равной одной миллисекунде. Примем, что мы используем самый простой вариант – после заголовка будем передавать один байт, как он есть. То есть, код для команды номер 1, выражаемой числом 1, будет выглядеть (в терминах базового периода) так:

Импульсный код для байта 00000001

Рис. 4.12. Структура импульсов кода

Пауза между командами должна быть не менее 25 мс.

Таким образом, программа должна воспроизводить примерно такую последовательность команд:



Я отнюдь не фанат «изящного кода», но даже мне такое лобовое решение не по нраву.

Но мы уже знакомы со счетным циклом, который можно выполнить восемь раз (по количеству отправляемых бит).

При этом каждый бит может принимать только два вида, либо единицы, либо нуля.

Выше мы знакомились с использованием клавиатуры, которая по нажатию кнопки может задавать значение (номер кнопки) переменной. Назовем ее *key*. Эту переменную можно использовать в программе для генерации соответствующего кода.

Чтобы определять значения каждого бита, можно использовать маску и программную конструкцию вида *key AND mask*, где *mask* – это маска, выделяющая (нашими усилиями) значение нужного бита.

Ниже приведена программа для пошаговой проверки и отладки генерации кода. Значение кнопки, якобы нажатой, вводится вручную. При необходимости эту часть можно заменить программой опроса клавиатуры.

Рис. 4.13. Программа, генерирующая управляющий код

В подпрограмме *cod_out* ниже используются два фрагмента **Calculation**. Но их можно объединить в один.

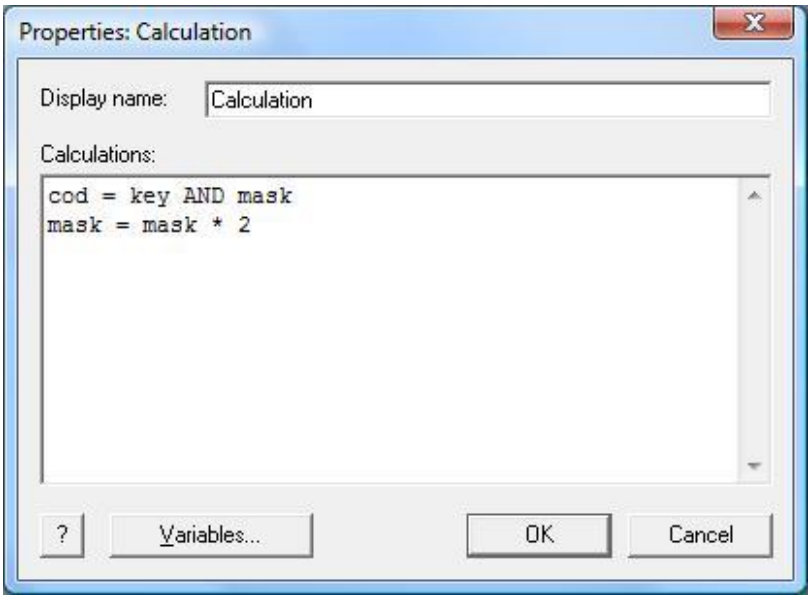


Рис. 4.14. Манипуляции с битами номера нажатой на пульте кнопки

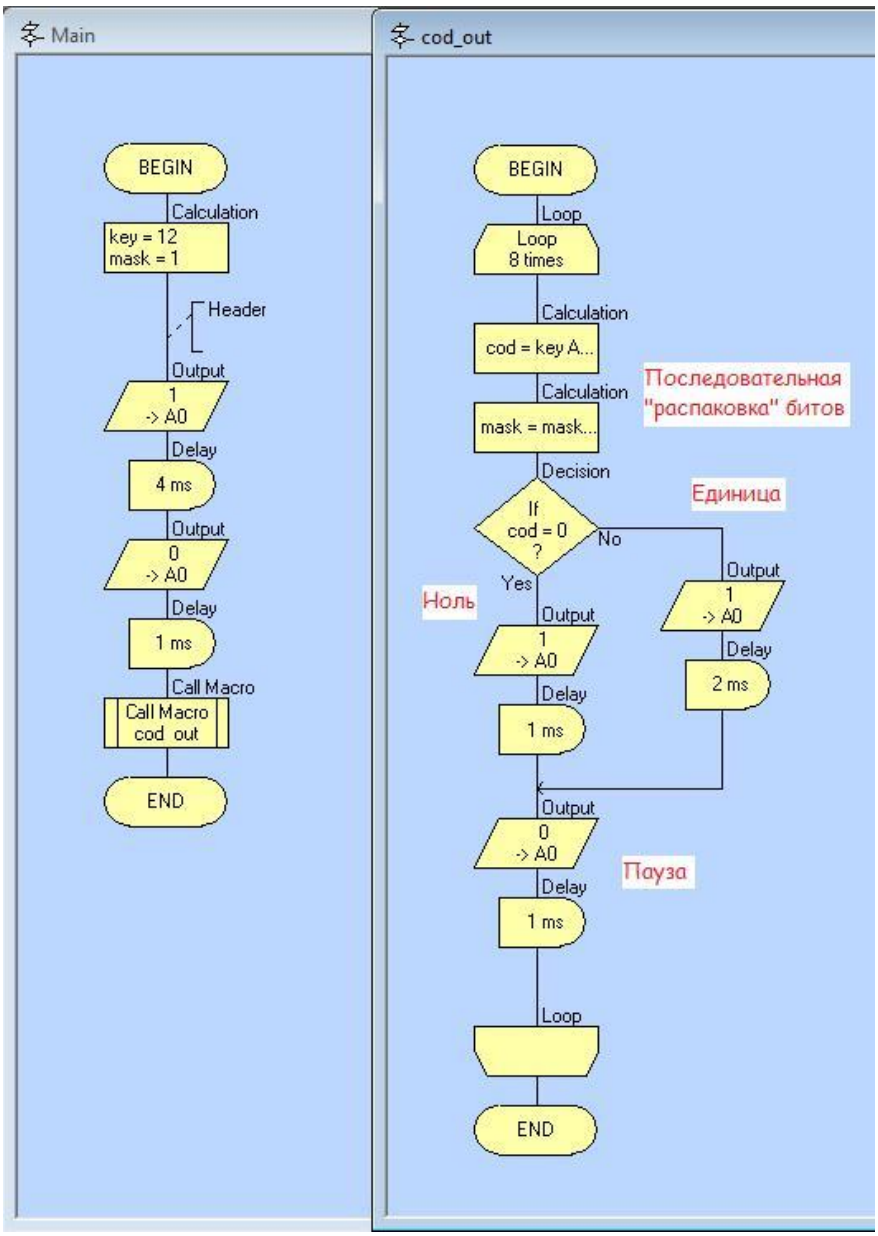


Рис. 4.15. Программа и подпрограмма подготовки номера кнопки к генерации кода

После заголовка, который остался без изменений, первым «в путь» отправится младший бит, а последним старший.

И вновь – мы используем только несколько уже знакомых фрагментов кода на языке Си. Для пущей надежности генерируемый код можно повторить несколько раз, не забывая вставлять паузу в 25 мс между повторениями команды. И еще одно: для своих экспериментов с ИК управлением можно использовать готовый пульт. Но, как мне кажется, разумнее создать свое устройство. В качестве излучающего светодиода можно использовать красный индикаторный диод АЛ307А. Если его «дальнобойности» не хватит, тогда поискать более мощный ИК светодиод. АЛ307А допускает импульсный ток до 100 мА при длительности импульса менее 10 мс и среднем токе порядка 20 мА. Для обеспечения нужного среднего тока мы примем меры, однако прежде все те фрагменты программы, где вывод RA0 принимает значение равное единице, следует заменить модулирующими импульсами.

Как видно из вышеизложенного, создание программы в FlowCode занимает немного времени и не требует больших усилий. Не намного больше усилий потребуется для создания кода программы на языке Си с использованием фрагментов, полученных в FlowCode. Но мне хотелось подчеркнуть, я об этом упоминал, что освоение языка Си будет полезно и в том случае, если вы намерены использовать полную версию программы.

Рассмотрим вопрос о замене состояний RA0 равных единице модулирующим кодом, или какими должны быть модулирующие импульсы? Если в приемнике использовать, что мне кажется очень разумным, приемный модуль типа TSOP с частотой фильтра 36-38 кГц, то период модулирующих импульсов окажется близким к 27 мкс. Если мы хотим использовать ток в 100 мА, то длительность включения 5 мкс должна обеспечить средний (за период) ток порядка 20 мА. Итак, импульс модуляции в программе может выглядеть таким:

Эта конструкция должна заменить и «единицу», и «ноль», и «заголовок» в предыдущей программе.

И все было бы хорошо, если бы ни одно «НО»!.. В программе FlowCode нет пауз меньше одной миллисекунды.

Вот здесь и самое время использовать знание языка Си и возможности, например, компилятора PICC Lite. Повторяя модулирующий импульс столько раз, сколько требует каждая из перечисленных «заготовок», мы сформируем полный ИК управляющий код.

И последнее – если использовать это устройство с автономным питанием, то следует подумать о режиме «сна» для МК. Это будет экономить энергию батареек.

Рис. 4.16. Программное формирование импульса модуляции

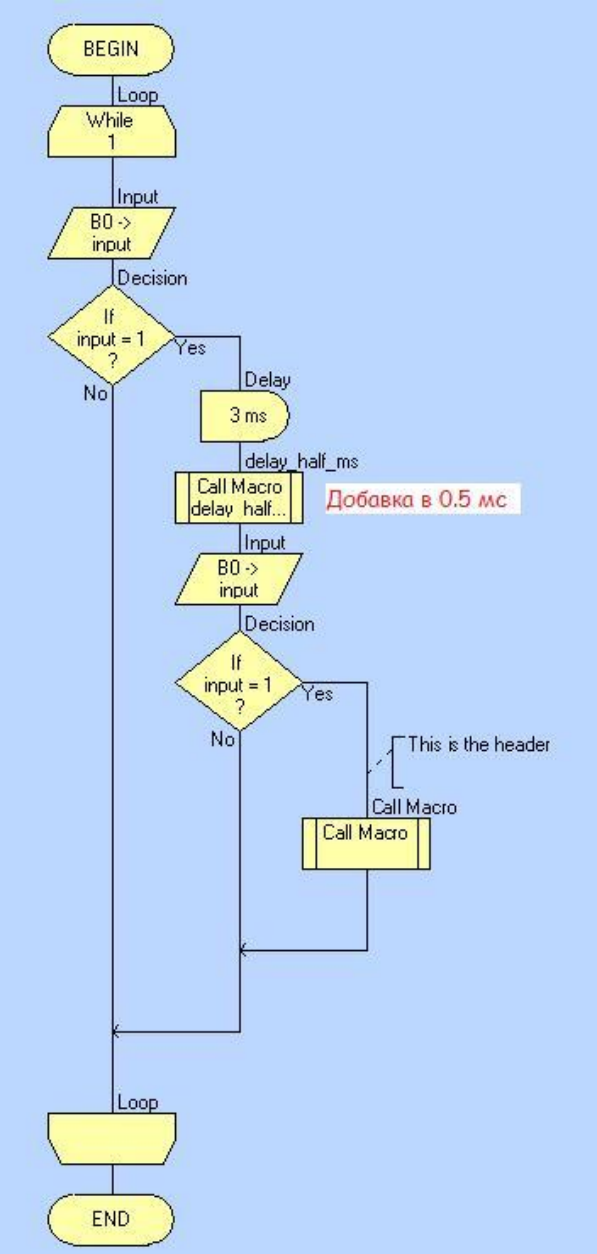
Если заменить ИК излучатель радио-модулем в разрешенной для, например, управления моделями частотной области с дальностью действия в несколько метров, то можно использовать пульт для передачи команд в соседнюю комнату, где может располагаться приемник спутникового телевидения. А в качестве приемного устройства использовать аналогичный пульту модуль, который по полученным командам будет воспроизводить ИК коды пульта управления приемником спутникового телевидения. Для реальной работы вам потребуется не так много команд, а на создание программ для МК вы потратите несколько часов. Да и те уйдут, скорее всего, на прочитывание кодов пульта управления приемником спутникового ТВ.

А в следующем примере мне хотелось бы пояснить, почему я предпочел бы пульт ИК управления, описанный выше, использовать собственного «изготовления».

Шестой пример

Сегодня, когда в магазине можно купить готовый пульта управления, когда, как правило, есть старые, уже не нужные пульты, когда относительно недорого можно купить пульт, способный запоминать нужные ИК коды, сегодня изготовление пульта управления может показаться не нужным. Собственно, так оно и есть.

Однако прочитать коды, отправляемые с пульта управления, гораздо легче, если вы сами создали удобные для прочитывания коды. Как прочитать управляющие коды пульта из предыдущего примера, покажем в этом примере. Подразумевается, что в качестве фотоприемника используется микросхема типа TSOP, на выходе которой мы получим код, изображенный на рисунке 4.12. Только порядок битов будет обратным (если мы не изменим этого в программе пульта).



В данном случае программа ожидает появления сигнала на входе RB0.

Если приходит импульс от фотоприемника, то, учитывая, что задуманный нами заголовок имеет длительность 4 мс, программа делает паузу на 3.5 мс и вновь проверяет состояние входа. При приходе заголовка команды входной импульс не изменит своего состояния, что можно фиксировать, как начало команды.

И в этой программе «довесок» в 0.5 мс потребуется «изготовить» на языке Си.

После подтверждения прихода заголовка программа вызывает подпрограмму обработки команды. Сканирование команды можно организовать по тому же алгоритму, что был использован для чтения заголовка. Сдвиг на 0.5 мс существенно облегчит работу по сканированию кода.

Рис. 4.17. Один из возможных вариантов программы прочитывания кода

Но можно обойтись в программе приемника без вставки паузы в 0.5 мс. Вернее, полностью использовать другой алгоритм считывания кода.

Программа будет начинаться тоже с ожидания изменения состояния входа RB0 (кстати, можно использовать для этой цели и прерывание по изменению состояния RB0).

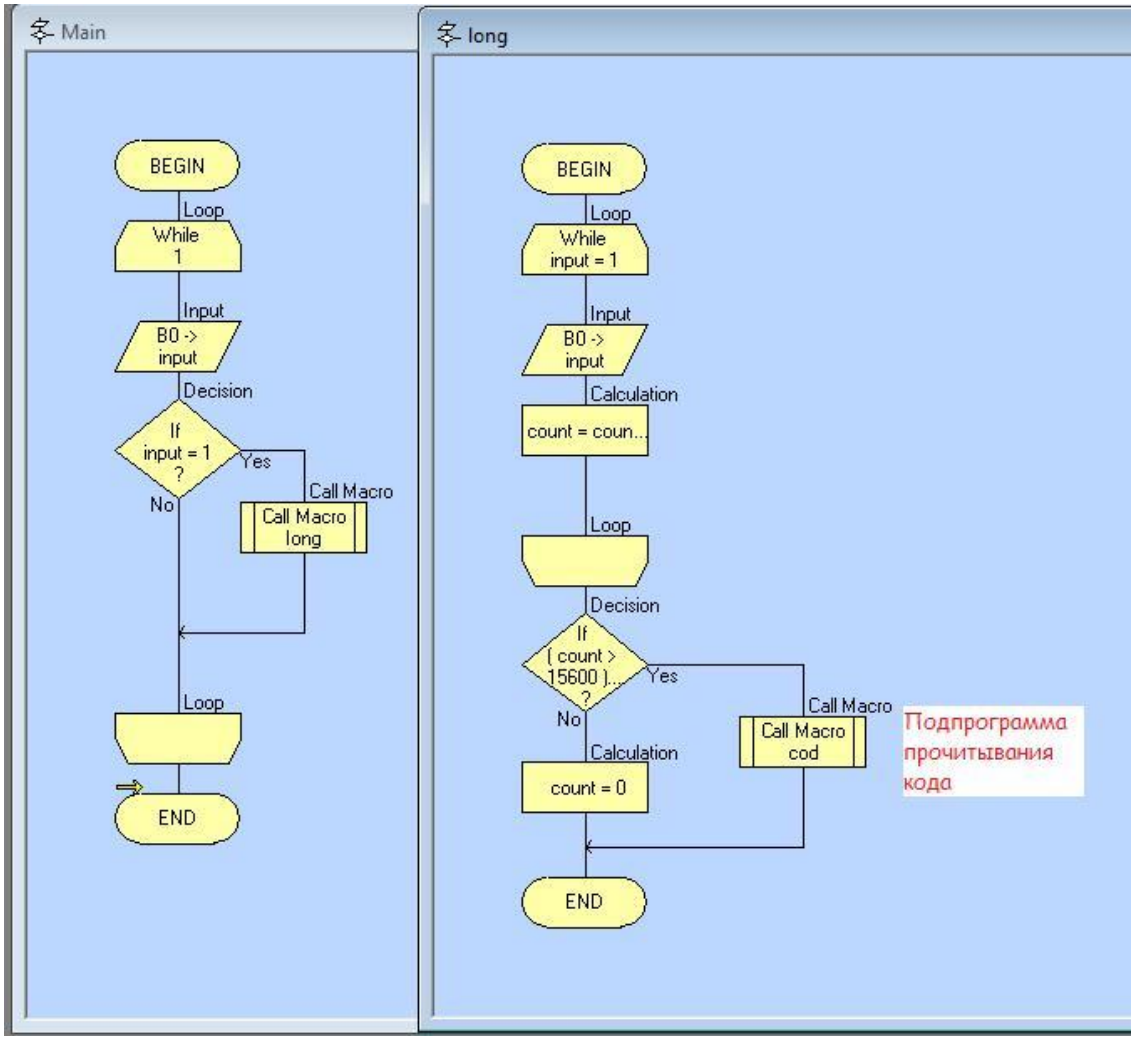


Рис. 4.18. Другой вариант программы прочитывания кода ИК команды

В этом варианте с началом прихода положительного импульса запускается счетчик. Грубая оценка полученного числа (без учета того, что цикл While input = 1 не будет выполняться за один машинный цикл) дает значение 16000. Заголовок длится 4 мс, а один такт МК занимает 0.25 мкс. Можно скорректировать это значение, если учитывать «неоднородность» приходящих модулированных импульсов, например, используя условие ветвления в подпрограмме:

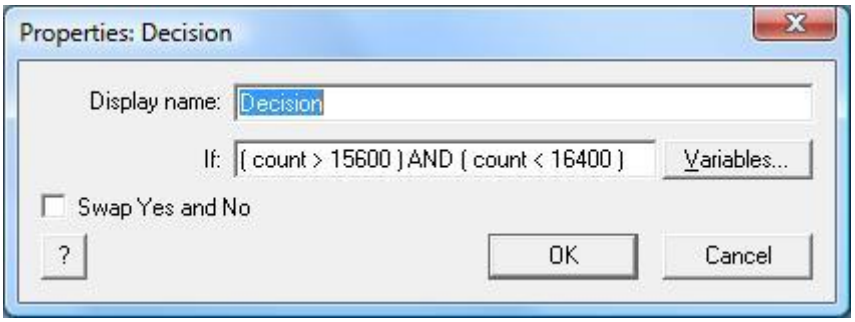


Рис. 4.19. Задание возможного разброса значений импульса

Аналогично можно использовать подсчет длительности импульсов и в подпрограмме «расшифровки» управляющего кода.

Кроме того, можно построить алгоритм прочитывания кода, используя встроенный таймер. Словом, здесь и таится самое интересное в работе с микроконтроллером – придумать и реализовать свой алгоритм работы устройства.

Но то, что мне хотелось показать в этом примере, надеюсь, мне удалось показать: если вы точно знаете формат принимаемого кода, у вас появляется много вариантов того, как создать программу для приемника.

Приложение

Немного о четвертой версии FlowCode

Четвертая версия, которая появилась в настоящее время, даже при первом запуске явно показывает изменения, произошедшие с программой.

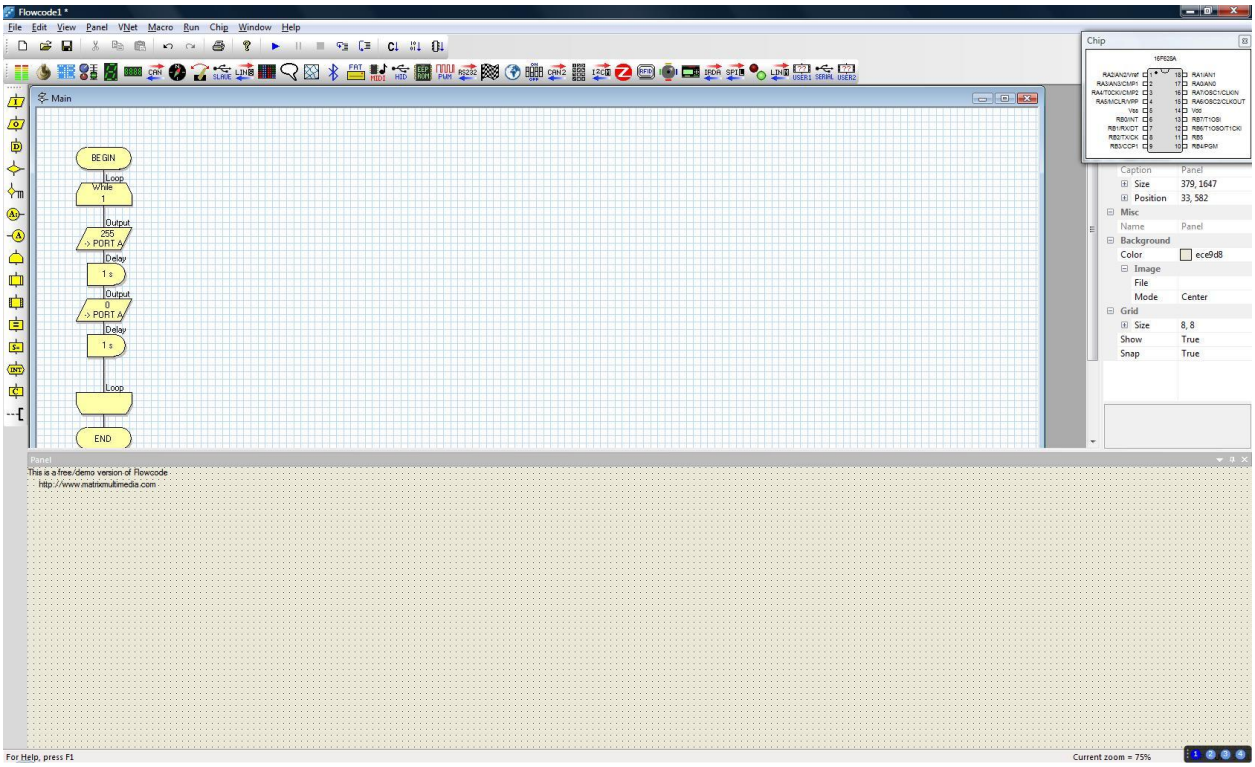


Рис. 5.1. Новое окно FlowCode

Кроме рабочего поля программы появилось новое поле – Панель для размещения дополнительных компонентов, список которых тоже существенно пополнился.

Для профессионалов существенным покажутся изменения в компиляторе. Например, появление возможности работать с числами с плавающей точкой.

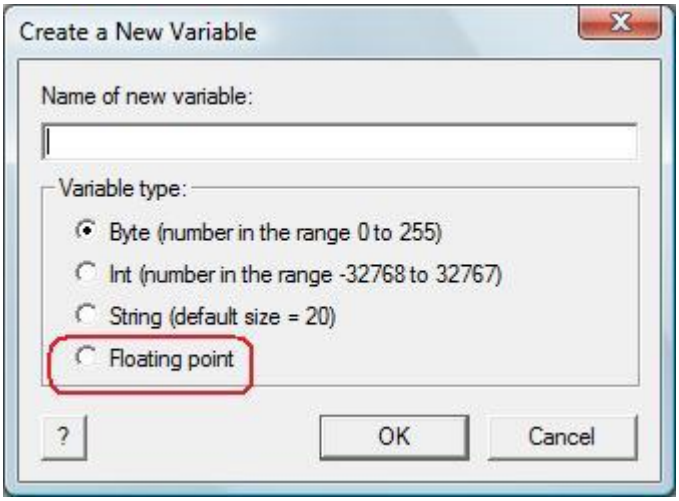


Рис. 5.2. Новое диалоговое окно переменной

Для всех удобным будет появление новой единицы в программном компоненте **Delay**.

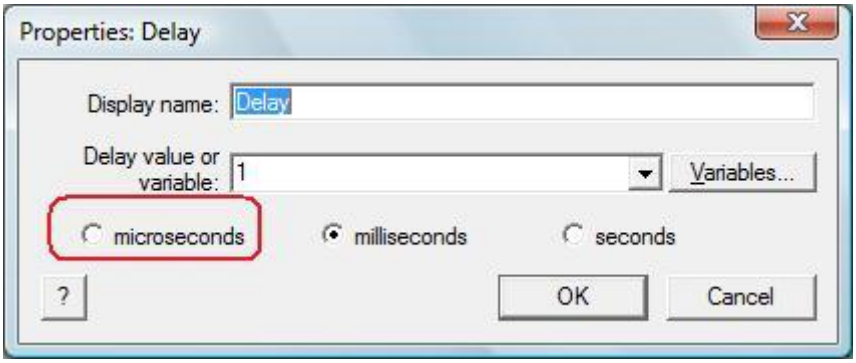


Рис. 5.3. Диалоговое окно компонента **Delay**

Не менее удобно появление нового программного компонента **Switch**.

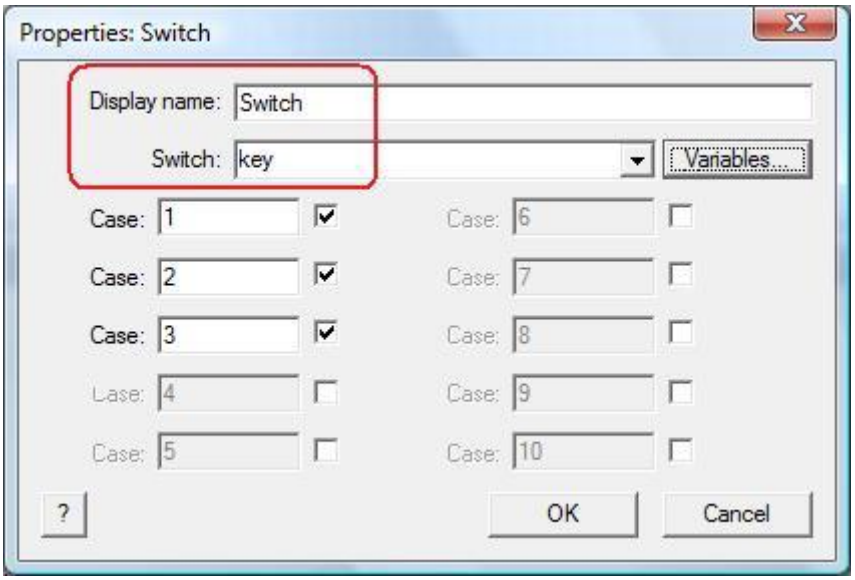


Рис. 5.4. Диалоговое окно компонента **Switch**

Использование этого компонента делает программу нагляднее.

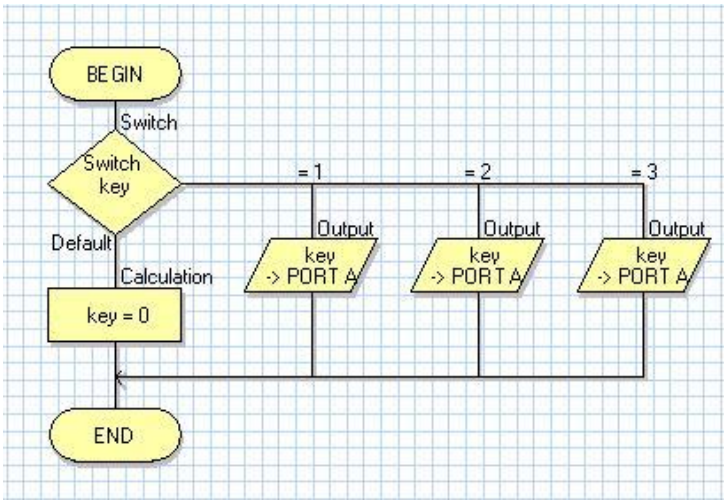


Рис. 5.5. Использование компонента **Switch** в программе

Изменения коснулись и дополнительных компонентов. Так появилось окно свойств, где двойной щелчок левой клавишей мышки в поле *Connections* открывает диалоговое окно подключения компонента.

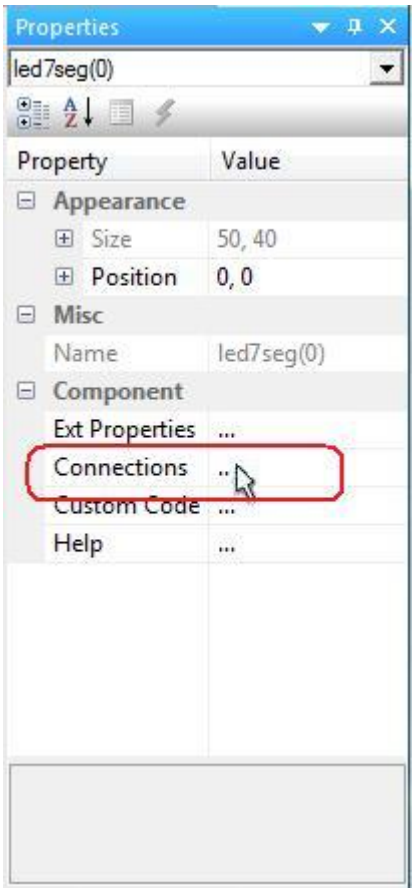


Рис. 5.6. Окно свойств компонентов

Есть изменения и в подключении компонентов. Если диалоговое окно *Connection* мало отличается от привычного ранее, то второй элемент в окне свойств *Ext Properties* открывает новый диалог.

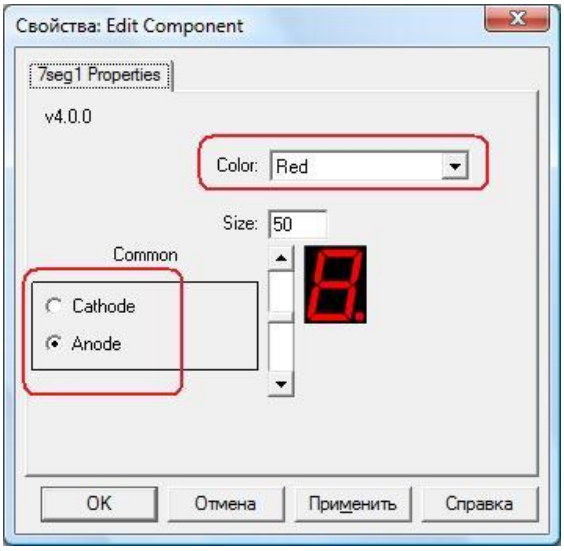


Рис. 5.7. Диалоговое окно дополнительных свойств семисегментного индикатора

Теперь возможно изменить цвет индикатора, его размер, а для многих важно то, что можно изменить общий вывод: общий катод или общий анод.

Несколько изменилось и поведение ряда дополнительных компонентов, так четырехзначный индикатор теперь будет работать при его подключении к порту в том случае, когда вы используете **Component Macro**. Что вполне разумно.

Изменился вид отдельных дополнительных компонентов, вернее, появился выбор этого вида.

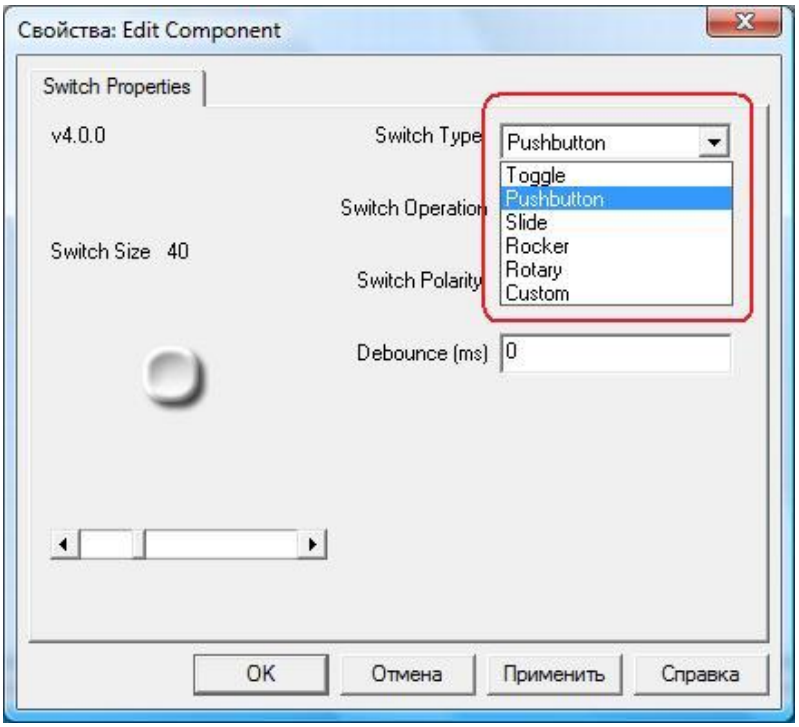


Рис. 5.8. Свойства компонента выключатель

И еще одно полезное нововведение:

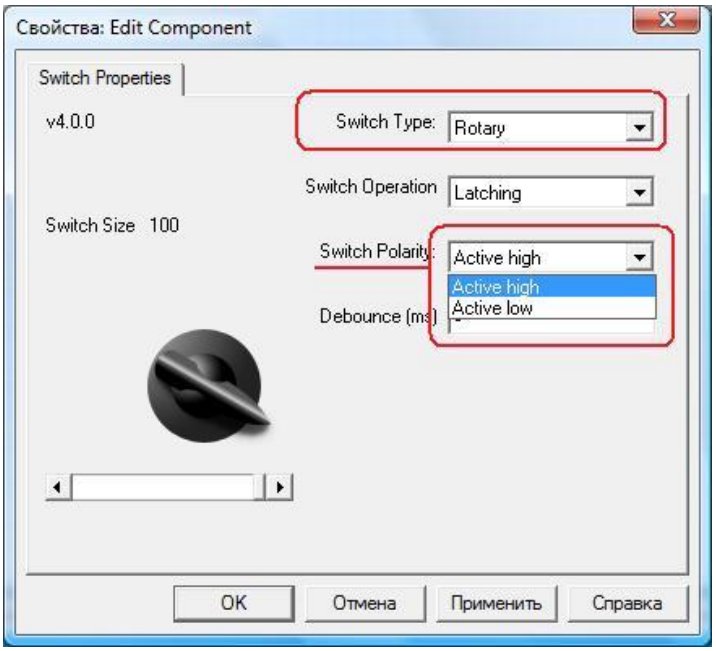


Рис. 5.9. Выбор полярности подключения выключателя

Панель для установки внешних компонентов может по вашему желанию изменить вид.

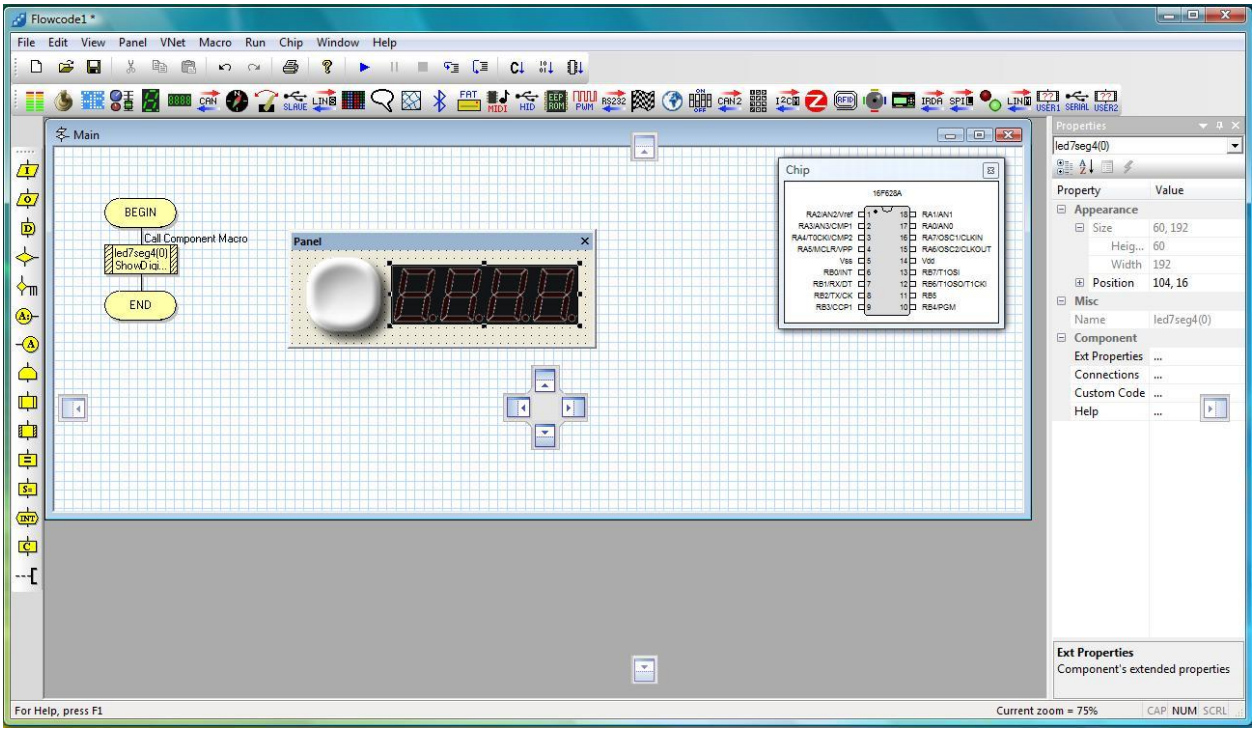


Рис. 5.10. Измененный вид панели (Floating – плавающая)

Для выбора вида панели достаточно щелкнуть правой клавишей мышки по ее титульной панели.

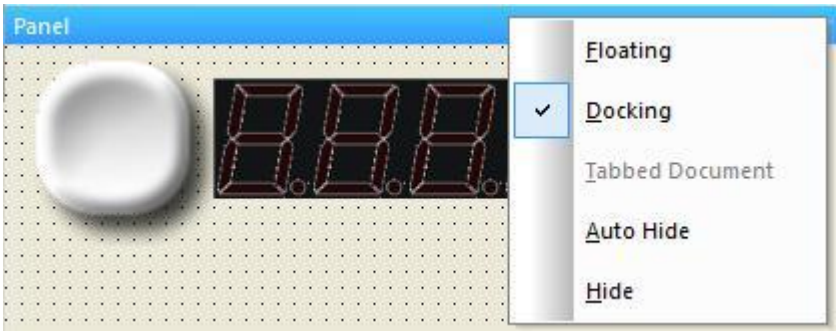


Рис. 5.11. Меню выбора вида панели

Словом, изменений много, а, став обладателем полной версии, вы найдете много для себя полезного. Вот, например, что говорит **Help** об изменениях в новой версии.

Развитие компонентов:

- Новые компоненты, включающие Stepper, Servo, Speech…
- Улучшена функциональность компонентов, включая ADC, RS232, I2C, PWM…
- Симуляция компонентов стала более гибкой и изменяемой
- Компонентные макросы теперь изменяемы
- Компоненты классифицированы

Программные возможности:

- Виртуальные сети FlowCode
- Внутрисхемная отладка (ICD)
- Окно наблюдения за переменными может отображать значения в реальном времени с использованием ICD
- Окно наблюдения за переменными может отображать значения регистров устройства в реальном времени с использованием ICD
- Поддерживается арифметика с плавающей точкой
- Пользовательские макросы только для чтения
- Паузы в микросекундах
- Сторожевой таймер теперь поддерживается паузами и компонентами
- Функции преобразования чисел с плавающей точкой и строковых данных
- Функции преобразования шестнадцатеричных чисел и ASCII
- Встроенный редактор кода с подсветкой синтаксиса

Улучшения графического интерфейса:

- Новый улучшенный графический интерфейс
- Панель симуляции
- В подсказках под иконками полный текст
- Окна с закладками
- Новый вид
- Темы фонового рисунка
- Улучшенные иконки

Улучшения иконок:

- Новая иконка программного переключения
- Новые иконки компонентов и графики симуляции
- Точкам соединения могут назначаться выразительные этикетки

Совместимость с моделями

- Внешний целевой микроконтроллер поддерживается прямо из FlowCode
- Внешние прерывания поддерживаются как стандартные для целевых устройств
- Улучшена поддержка программных средств других производителей

Изменений, действительно, достаточно много. Все ли они к лучшему, покажет время.

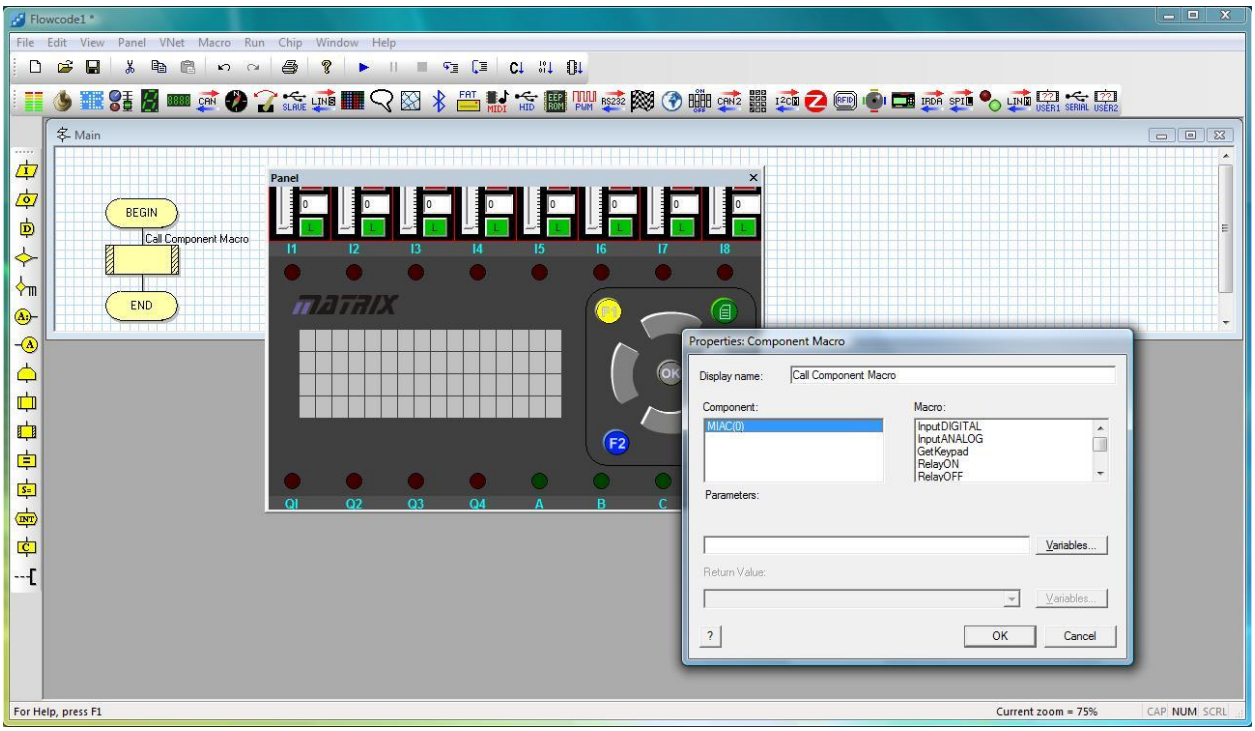


Рис. 5.12. Четвертая версия FlowCode

Немного о программе Proteus

Вернее о ее составляющей ISIS. Для работы с микроконтроллерами достаточно программы FlowCode, если вы точно знаете, что хотели получить от контроллера и получили в результате. Сама программа FlowCode предоставляет более чем исчерпывающие средства отладки. Однако, скажем, для проверки работы контроллера с внешним генератором, или при проверке сетевой связи двух микроконтроллеров можно воспользоваться программой Proteus, если она у вас есть.

Поводом использовать все имеющиеся средства может стать и появление неких чудесных явлений, причина которых, впрочем, кроется либо в неисправности микроконтроллера, либо, что бывает чаще, в собственных ошибках. Вот пример такой ошибки, которая, как я полагаю, заставит и вас работать по принципу «доверяй, но проверяй».

Все достаточно банально. Для решения задачи нужно бы «засветить» четырехразрядный семисегментный индикатор. Программа пишется и проверяется в FlowCode. Все, похоже, работает. Но при проверке «в железе» не горят два сегмента: E и F. Индикатор подключен к порту A контроллера PIC16F628A. С сегментом E понятно, на выходе транзистор с «открытым» стоком, то есть, требуется его подтянуть к плюсу питания. А со вторым сегментом...

На помощь приходит программа Proteus.

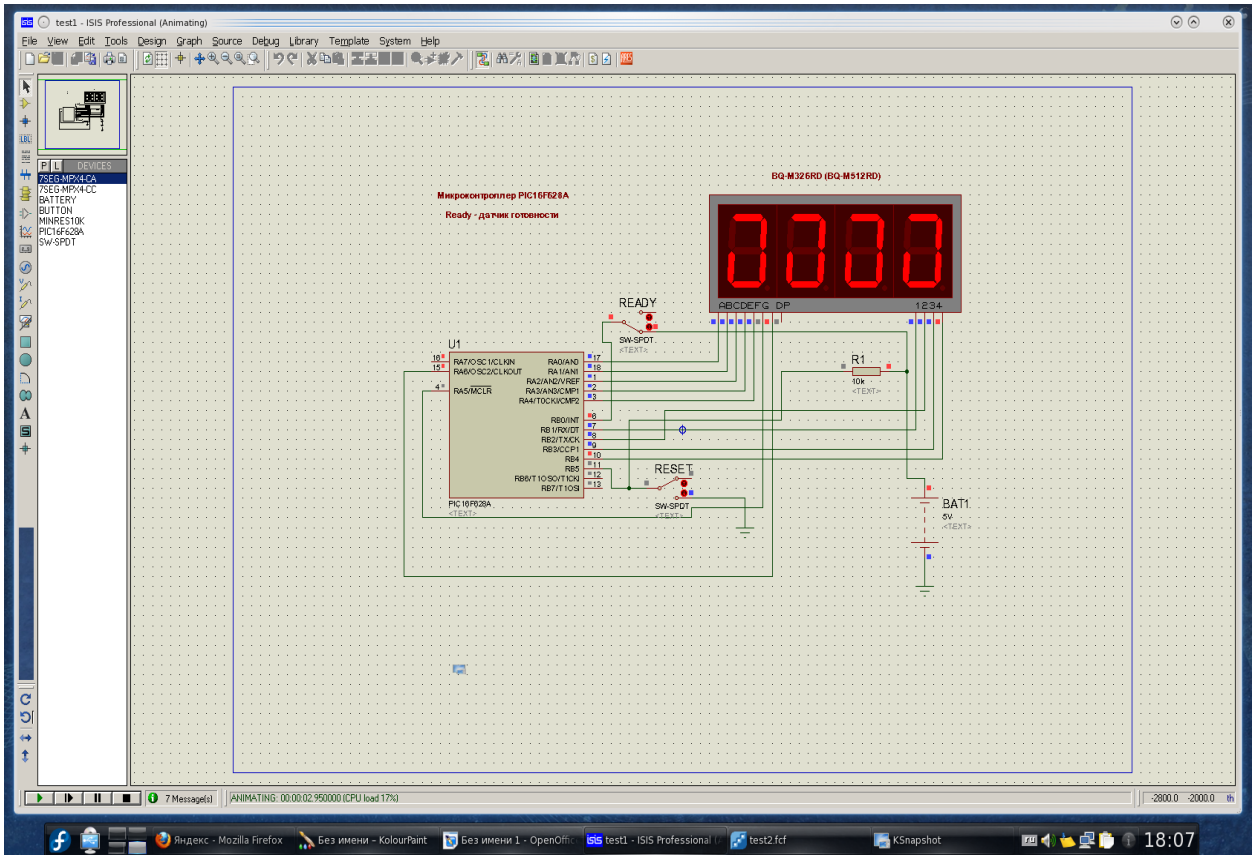
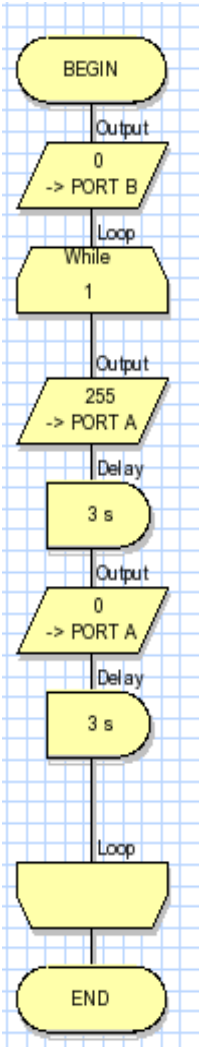


Рис. 5.13. Моделирование в программе Proteus

И в Proteus сегмент F не горит. Такое впечатление, что один из выводов порта A не желает, в отличие от моих планов, работать на выход. Можно, конечно, проверить программу на макетной плате, но не хочется покупать индикатор только для единственной проверки. Задачу можно,

впрочем, упростить, вернее, упростить программу, оставив только зажигание и гашение всех сегментов для индикатора с общими катодами, которые подключены к порту B.



Программа подключает катоды всех разрядов к общему проводу (устанавливает низкий уровень на выходах порта B).

Затем в бесконечном цикле все выводы порта A устанавливаются в высокое состояние, а после паузы в 3 секунды все выводы порта устанавливаются в низкое состояние.

Программа совершенно проста и ясна, нет никаких сомнений относительно ошибок, скажем, в использовании режима динамической индикации.

Но прежде, чем начинать проверки, я еще раз читаю описание микроконтроллера PIC16F628, отрыв первое, подвернувшееся под руку, благо оно на русском языке. Получается, что если слово конфигурации задано верно (а компараторы отключены), то весь порт A должен работать как цифровой вход/выход.

Получить hex-файл в такой простой программе дело минутное.

Рис. 5.14. Проверочная программа

Полученный hex-файл загружается в программу Proteus, запускается и…

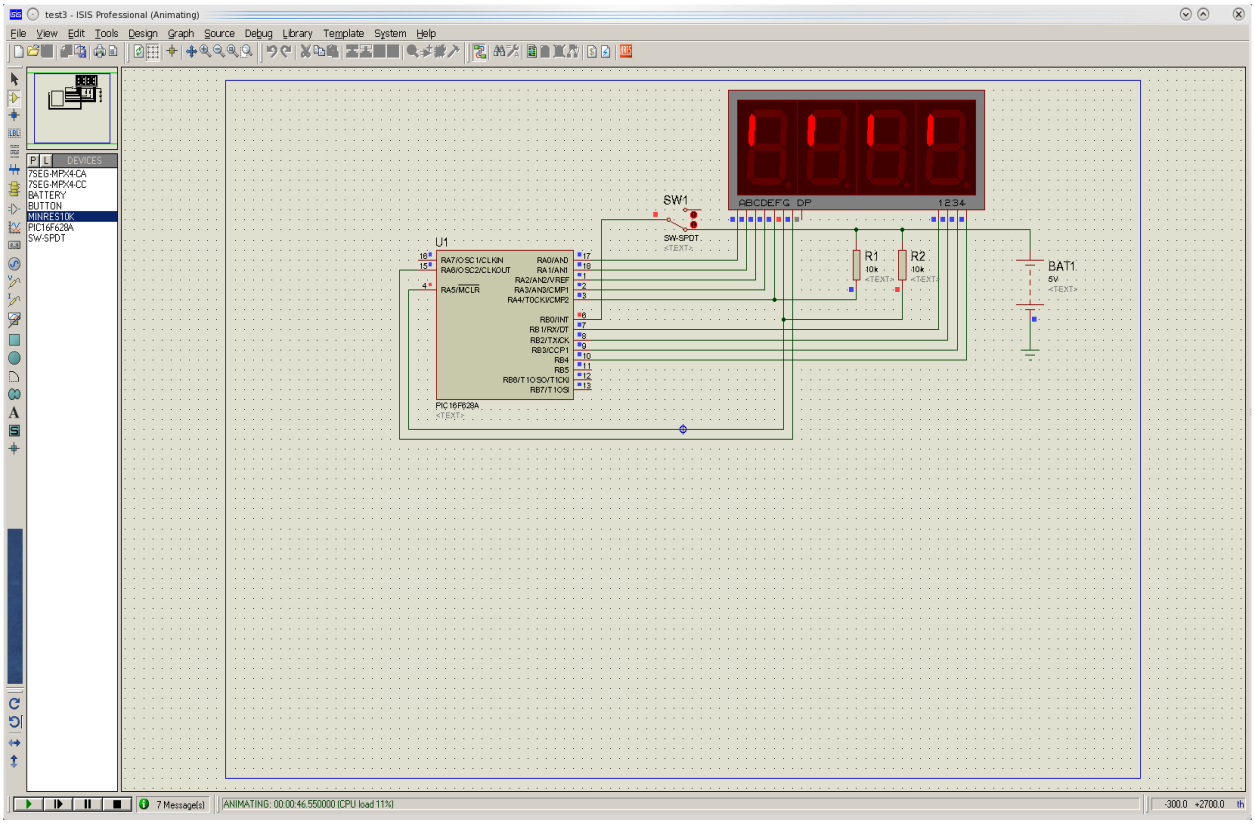


Рис. 5.15. Проверка работы предыдущей программы в Proteus

И... теперь сегмент зажигается, но не гаснет. Повторяем все тщательно еще раз. Заводим новую папку для программы ISIS (Proteus). Запускаем программу. Устанавливаем микроконтроллер. Есть разные способы достичь этого. Я привык (и когда успел?!) использовать кнопку инструментальной панели и менеджер библиотеки.

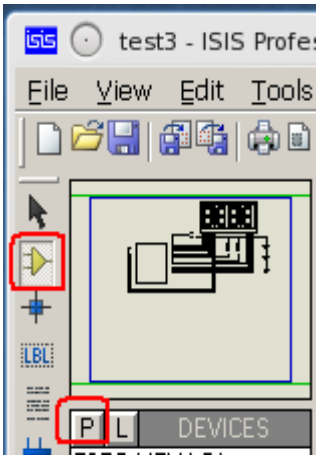


Рис. 5.16. Доступ к компонентам в Proteus

Proteus я использую в Linux, там есть небольшие проблемы с выбором из списка. Но есть поиск по названию, который я использую:

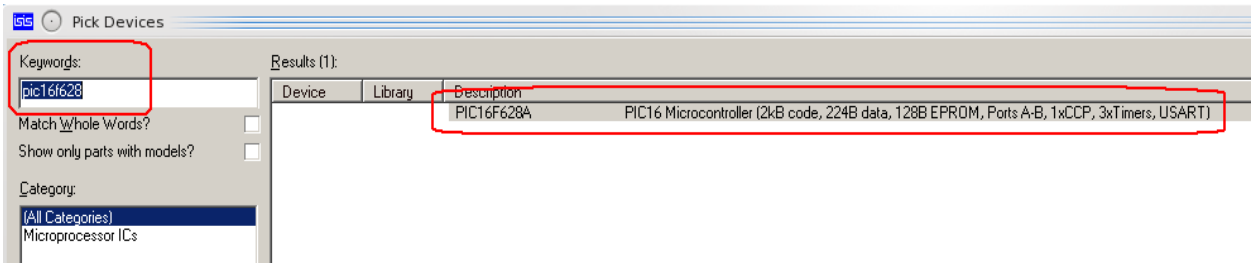


Рис. 5.17. Поиск в менеджере компонентов Proteus

Установив (в менеджере компонентов щелчок по кнопке ОК, затем в нужном месте рабочего поля щелчок левой клавиши мышки) контроллер на схему, в разделе Optoelectronics менеджера компонентов я нахожу две модели индикаторов: 7SEG-MPX4-CA (общий анод) и 7SEG-MPX4-CC (общий катод). Последний и выбираю, вводя его в строку поиска. Теперь осталось соединить все выводы согласно программе и рисунку индикатора.

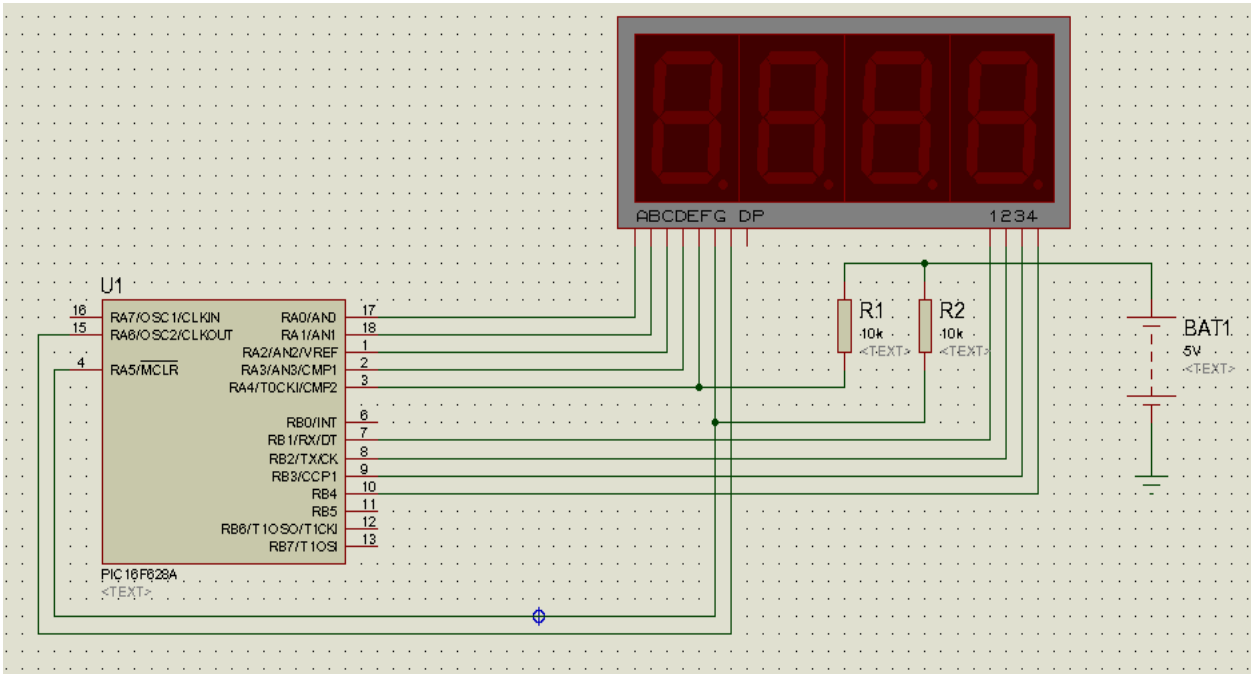


Рис. 5.18. Схема проверки

Двойной щелчок по микроконтроллеру открывает диалоговое окно его свойств.

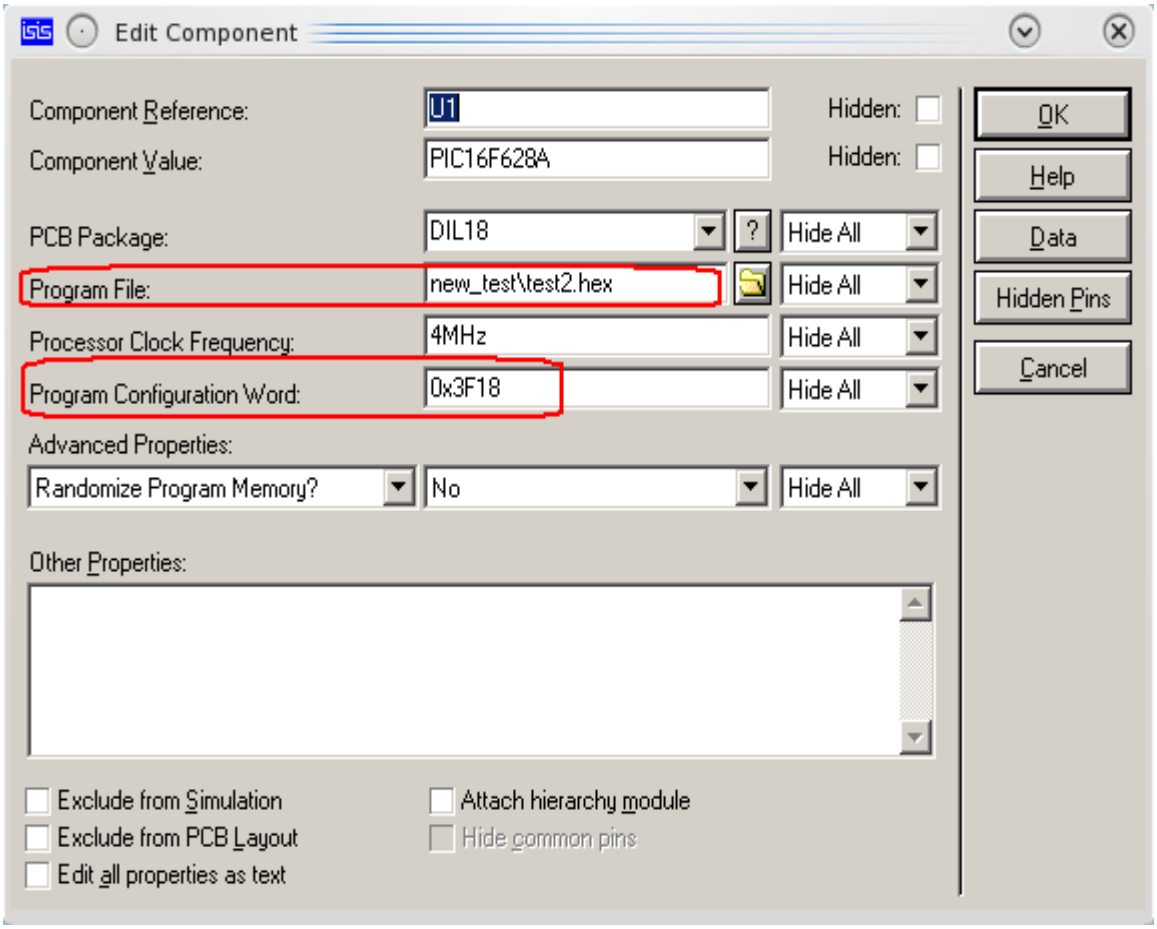


Рис. 5.19. Диалоговое окно свойств микроконтроллера

В окне диалога следует указать путь к hex-файлу, который лучше перенести в папку с проектом Proteus, указать слово конфигурации. Проделав все это, я запускаю программу, и... получаю тот же результат, что изображен выше. Похоже, что бит, связанный с сегментом F не желает работать. Но программа Proteus имеет свои возможности отладки, которые можно запустить, например, из основного меню (пункт Debug).

Конечно, сам запуск программы уже операция по отладке схемы. Можно использовать графики. Эта возможность, присущая всем программам, в которых разрабатываются электрические схемы, эквивалентна применению осциллографа при работе с макетной платой. Здесь вполне справедлив подход: лучше один раз увидеть, чем сто раз услышать.

Программа Proteus, к слову, имеет виртуальный осциллограф. Но в данном случае удобнее было бы использовать график, с которым можно работать, детально разбирая все аспекты полученной осциллограммы. Однако при работе с микроконтроллерами, все-таки основой является программа, которую в первую очередь и следует проверить.

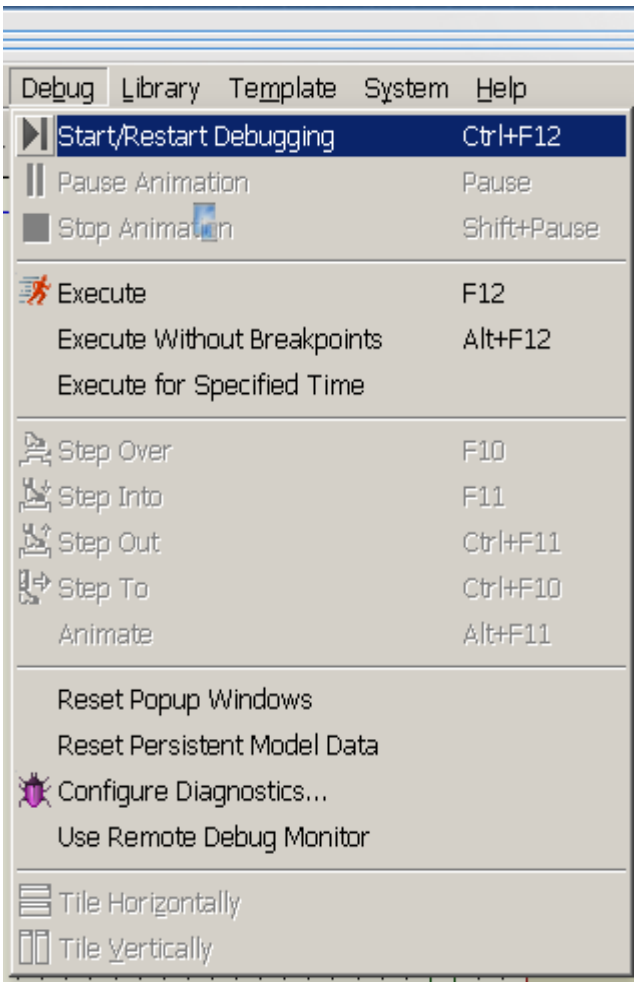


Рис. 5.20. Выпадающее меню отладчика в Proteus

Нажав ***Start/Restart Debugging***, после повторного входа в этот раздел основного меню можно обнаружить перемены, которые произошли с меню.

Такие изменения весьма часто встречаются в программах. А меню такого рода называют контекстным меню. Это относится и к тем разделам меню, которые выглядят бледно. Они активизируются только тогда, когда программа переходит к выполнению действий, затрагивающих эти команды. Иначе эти команды не используются. Поэтому они не активизируются.

Очень полезно, осваивая программу, внимательно присматриваться к тому, что происходит и с программой, и с ее компонентами: элементами схемы, например, с выпадающими меню, с всплывающими меню и подсказками.

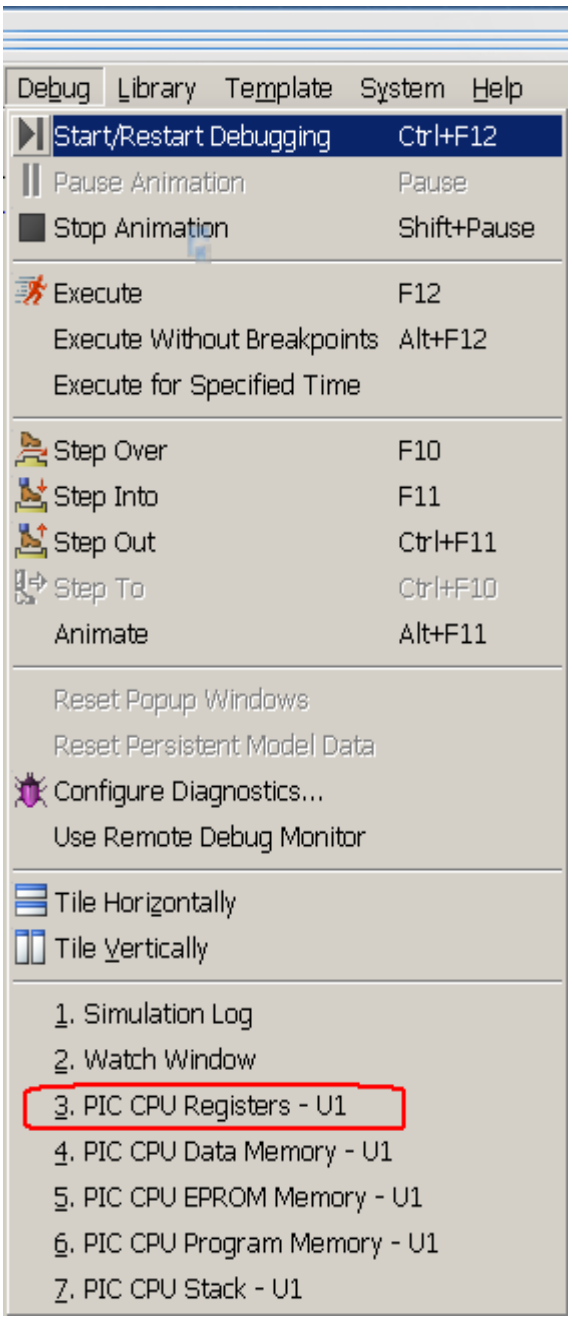


Рис. 5.21. Выпадающее меню отладчика после запуска отладки

Нажимая пункт, отмеченный на рисунке, получаем возможность наблюдать за регистрами. Это можно сделать в пошаговом режиме (например, клавиша F11 на клавиатуре), можно использовать команды ***Step Over*** (клавиша **F10**) или ***Step Out***, (клавиши **Ctrl+F11**), можно, наконец, запустить программу клавишей «►» на панели строки состояния.

Рис. 5.22. Клавиши управления симуляцией

Но главное, что мне хотелось увидеть, это:

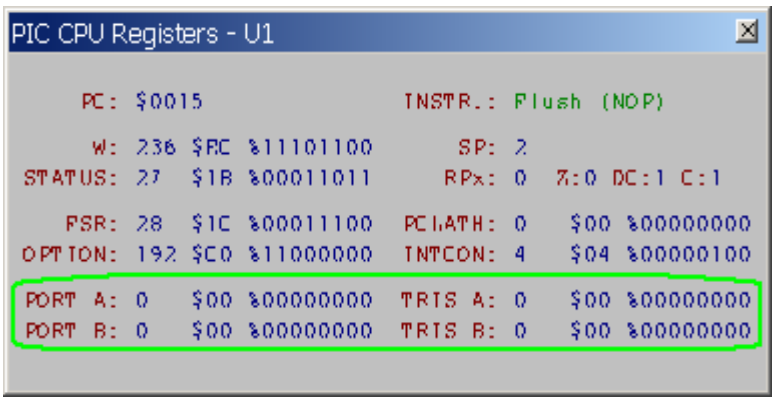


Рис. 5.23. Окно наблюдения за регистрами

Как я полагаю, порты A и B работают на выход, в портах все нули, значит... все сегменты должны погаснуть. Кстати, чтобы не возвращаться к этому: в примерах к программе Proteus есть возможность лучше посмотреть, как используются отладка программы микроконтроллера.

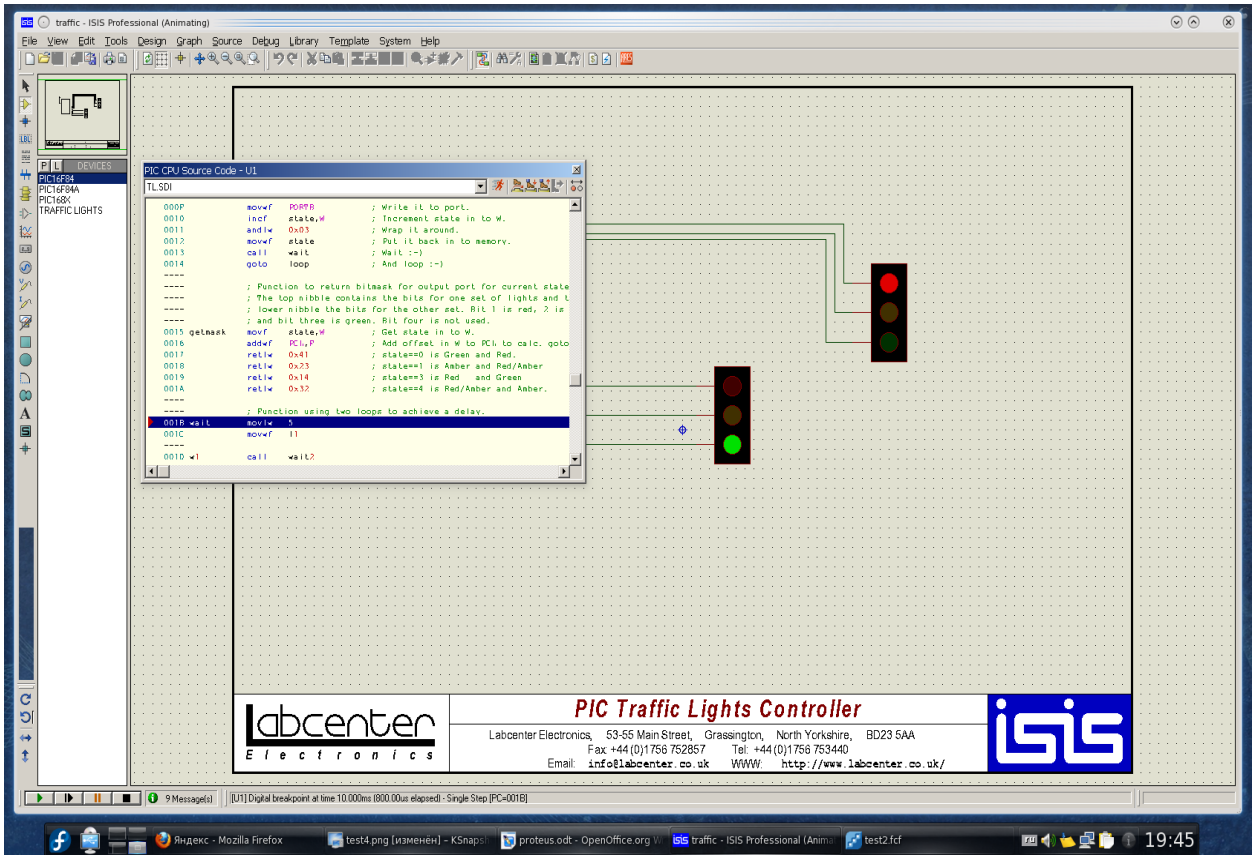


Рис. 5.24. Отладка программы МК из примеров Proteus

Когда знаешь решение задачи, когда знаешь ответ на вопрос, то удивляешься, отчего же ты не увидел то, что лежит на поверхности. Но это, когда знаешь.

Проверка на макетной плате показала, что подтянутый резистором к плюсу источника питания вывод RA5 остается в состоянии с высоким уровнем после команды – установить низкий уровень на всех выводах порта A.

Теперь самое время вспомнить о том, с чего начинался разговор о программе FlowCode. С того, что она хороший помощник в изучении языка Си. Повторить программу, изображенную на рисунке 5.14, совсем несложно.

```
#include <htc.h>
#define _XTAL_FREQ 4000000
void main()
{
    char i;

    while (1) {
          CMCON = 0x07;
          TRISA = 0x00;
          PORTA = 0xFF;
          for (i=0;i<30;i++) {
                __delay_ms(100);
          }
          TRISA = 0x00;
          PORTA = 0x00;
          for (i=0;i<30;i++) {
                __delay_ms(100);
          }
    }
}
```

И, создав проект в программе MPLAB, добавив основной файл проекта на языке Си, запустить отладку тестовой программы в этой среде работы с микроконтроллерами.

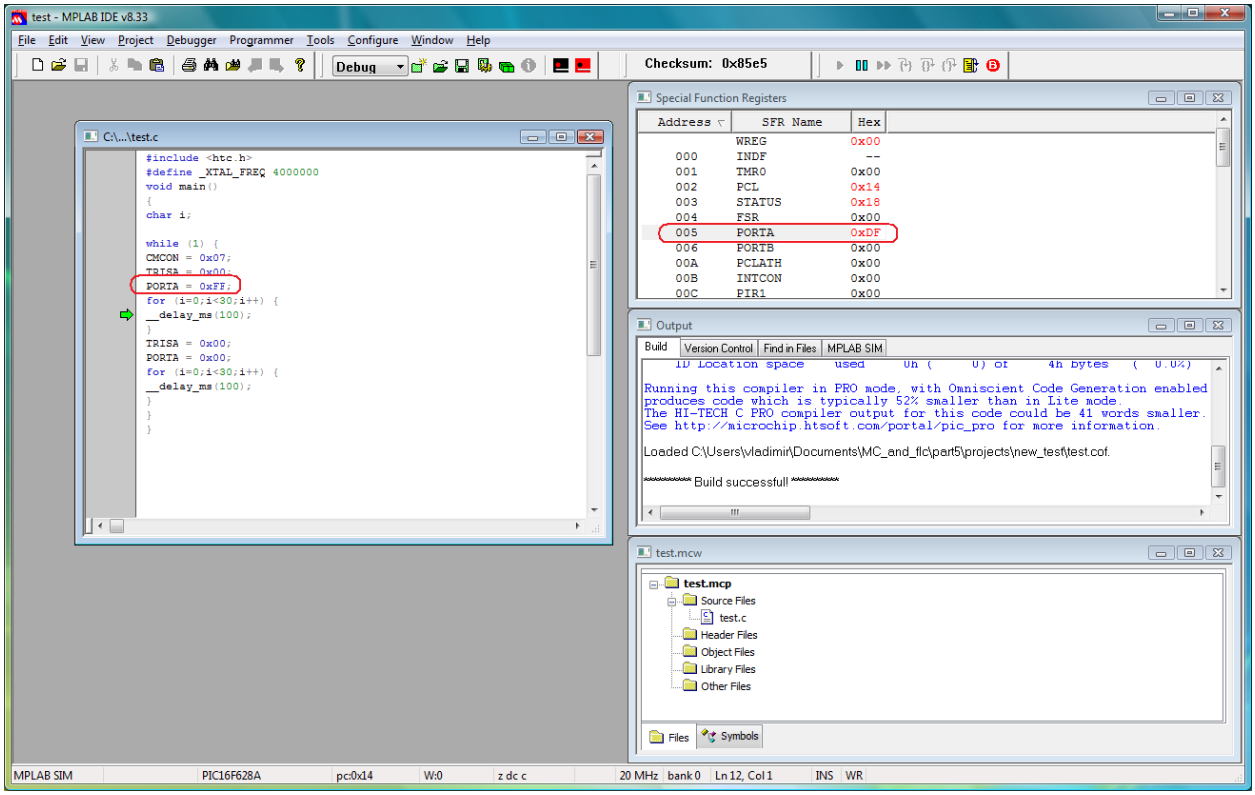


Рис. 5.25. Запуск отладки программы в среде MPLAB

В этот раз видно, что после команды установить все выводы порта А в высокое состояние, в высокое состояние устанавливаются все выводы, кроме RA5 (в окне наблюдения за регистрами видно, что значение не FF, а DF).

И, наконец, решение задачи, ответ на вопрос, что происходит с выводом RA5, обнаруживается в оригинальной версии описания микроконтроллера, где написано:

RA5 is a Schmitt Trigger input only and has no output driver (RA5 – это только вход триггера Шмитта и не имеет выходного драйвера)

и далее о слове конфигурации, касательно этого вывода

RA5/MCLR pin function is digital Input (функция вывода RA5/MCLR – это цифровой вход).

В заключение хочу сказать, что эта банальная ошибка позволила мне чуть-чуть рассказать о программе Proteus и еще раз напомнить, что как бы ни удобно было работать с программой FlowCode, очень полезно освоить работу над программой хотя бы на языке Си.

Пока все получается, пока нет вопросов – дело вкуса. Вы можете изучить ассемблер и пользоваться только им. Вы можете изучить Си и пользоваться только им. Если для вас важно быстро создать программу, вы обязательно обратите внимание на FlowCode.

Но, когда что-то идет не так, когда появляются вопросы, никакие знания не будут лишним обременением. И в этот момент изучать что-то будет гораздо труднее.

Не слушайте тех, кто говорит, что пользоваться FlowCode плохо. Они заблуждаются. Но и повторять их заблуждения, пользуясь только FlowCode, не совсем разумно. Не так ли?

Читая учебник

Аналогично тому, как удобно изучать электротехнику и электронику с программой и книжкой, не менее удобно изучать язык, повторяя программу за компьютером.

Есть прекрасная книга по использованию языка Си в работе с микроконтроллерами, автор Шпак Ю.А., «Программирование на языке С для AVR и PIC микроконтроллеров», издательство «МК-Пресс». Безусловно, самым удобным сочетанием было бы использование тех (или той) программ, о которых пишет автор. Но ничто не мешает к его рекомендациям добавить свои предпочтения. Итак.

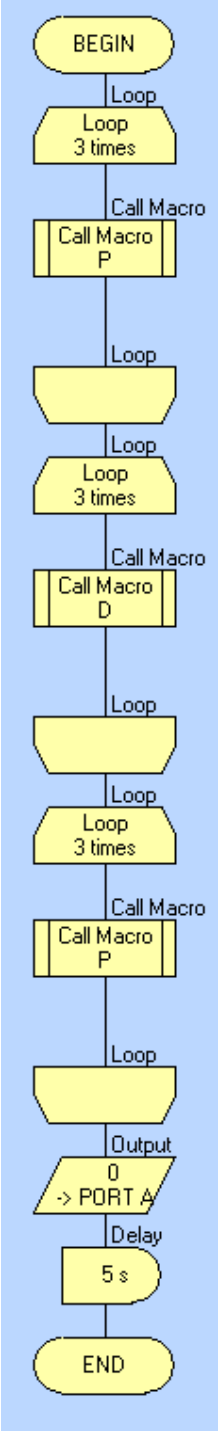
Одной из первых встреченных программ в книге будет генератор сигнала SOS.

```
#include <18F458.h>
#use delay(clock=20000000)
#fuses HS,NOWDT

void Pause(int ms)
{
      output_D(0xFF);  //Все светодиоды отключены
      delay_ms(ms);    //Задержка
}

void P(void)
{
      output_D(0);     //Включаем все светодиоды
      delay_ms(5);     //Короткая задержка
      Pause(5);        //Пауза с погасшими светодиодами
}

void D(void)
{
      output_D(0);      //Включаем все светодиоды
      delay_ms(20); //Длинная задержка
      Pause(5);         //Пауза с погасшими светодиодами
}

int main (void)
{
      set_tris_D(0xFF); // Настройка порта D для вывода
      while(1)    //Бесконечный цикл
      {
        P(); P(); P(); //. . .
        D(); D(); D(); //- - -
        P(); P(); P(); //. . .
        Pause(100);
      }
   }
```

Читать код программы, повторяя его на языке FlowCode, будем с конца: трижды повторить функцию *P*, трижды повторить функцию *D*, трижды повторить функцию *P* и следом функция *Pause*.

Функциям языка Си будут соответствовать макросы FlowCode, имена для которых оставим оригинальные. Длительности сигналов увеличим, чтобы можно было увидеть работу программы в отладочном режиме, и вместо функции *Pause* используем программный компонент **Delay**. И пока не будем программу помещать в бесконечный цикл.

Вид основной программы получится таким:



Первым используем счетный цикл, внутри которого повторим трижды вызов макроса **P**, который еще предстоит написать.

Затем повторим это для макроса **D**.

И опять используем макрос P.

Завершит программу выключение всех светодиодов и пауза в 5 секунд.

Длительность свечения светодиодов для точки выберем 1 секунда, для тире 3 секунды.

При таких длительностях запуск отладки в программе FlowCode позволит удобно наблюдать за результатом.

В качестве порта вывода (подключения светодиодов) в данный момент можно использовать порт A, задаваемый по умолчанию. При необходимости порт, как и длительности, можно заменить другим.

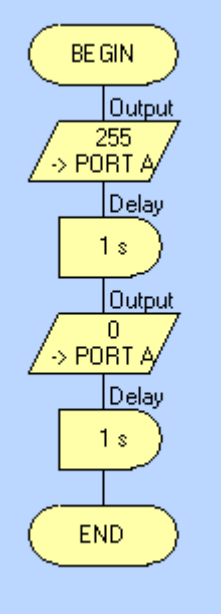
Когда программа готова, можно записать два необходимых для ее работы макроса.

Для этого двойным щелчком по программному компоненту открываем диалог, в котором выбираем кнопку ***OK&Edit Macro*** или в основном меню выбираем пункт **Macro** и команду **Show**, где выбираем по имени нужный нам макрос. Подразумевается, что сами макросы мы создавали сразу после установки программного компонента **Macro**.

И еще одно замечание: автор использует подключение светодиодов между выводом и плюсом питания. Удобнее (для отладки в FlowCode) использовать подключение светодиодов между выводом и общим проводом.

Рис. 5.26. Вид основной программы SOS в программе FlowCode

Вместе с внесенными нами изменениями подпрограмма (макрос P) для точки Морзе приобретает вид:



Все выводы порта A мы устанавливаем в высокое состояние из-за измененного нами способа подключения светодиодов.

Поскольку мы отказались от функции *Pause*, эту функцию мы повторим, устанавливая все выводы порта A в низкое состояние после выбранной нами паузы в 1 секунду, и делаем такую же паузу с погашенными светодиодами.

Рис. 5.27. Вид макроса для точки Морзе

Вид макроса D будет отличаться только длительностью первой паузы, что вытекает из вида функции, предложенной автором.

Теперь можно запустить отладку программы:

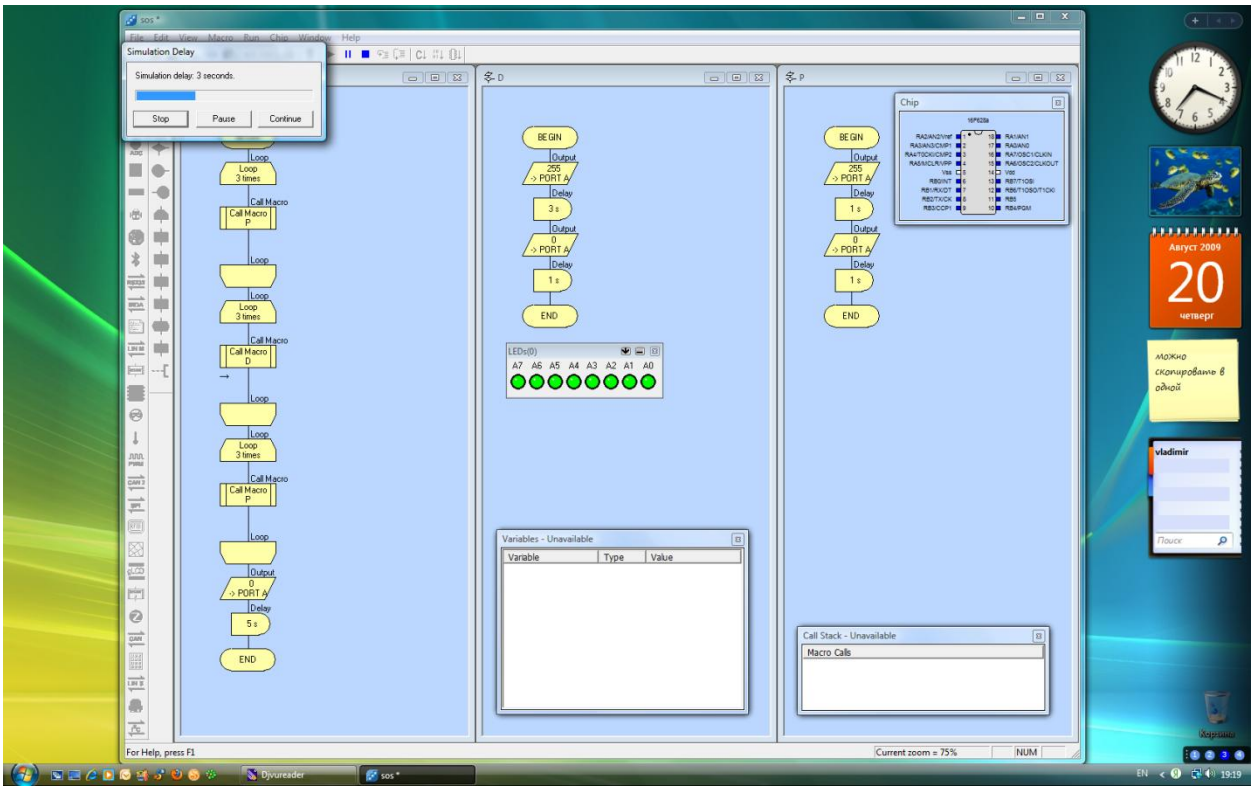


Рис. 5.28. Отладка программы SOS

Поскольку программа FlowCode имеет одинаковый вид и для PIC, и для AVR контроллеров, можно использовать программы из учебника, предназначенные для любого типа микроконтроллеров.

Например, вот программа «Бегающие глаза»:

«К выводам порта B подключены переключатели, а к выводам порта D - светодиодные индикаторы. Эффект "бегающих глаз" может быть создан путем поочередного включения двух светодиодов при выключенных остальных или, наоборот, поочередного выключения двух светодиодов при включенных остальных. Один из этих двух типов выбирается с помощью переключателя, подсоединенного к выводу 7 порта B. Остальные входы этого порта определяют скорость "бега" (коэффициент от 0 до 127)».

Листинг программы есть на CD-диске, приложении к книге:

```
#include <avr/io.h>
#include <avr/delay.h>

unsigned long DelayCount;
unsigned long Velocity = 0;
unsigned char EyeType = 0;

void ShowEyes(int i)
{
  if(EyeType) PORTD = ~i; else PORTD = i;
  _delay_loop_2(DelayCount);
}

int main (void)
{
  DDRB = 0x00;
  DDRD = 0xFF;
  while(1)
  {
    Velocity = PINB;
    if(Velocity > 127)
    {
      Velocity -= 127;
      EyeType = 1;
    }
    else EyeType = 0;
    DelayCount = 500 + (Velocity * 50);
    for(int i = 1; i <= 8; i = i*2) ShowEyes(i * 16 + i);
    for(int i = 8; i > 1; i -= i/2) ShowEyes(i * 16 + i);
  }
  }
```

Полезнее «перевести» листинг программы на графический язык, но прежде, не вижу в этом ничего плохого, можно посмотреть, например, что имеет автор ввиду, когда говорит о двух вариантах.

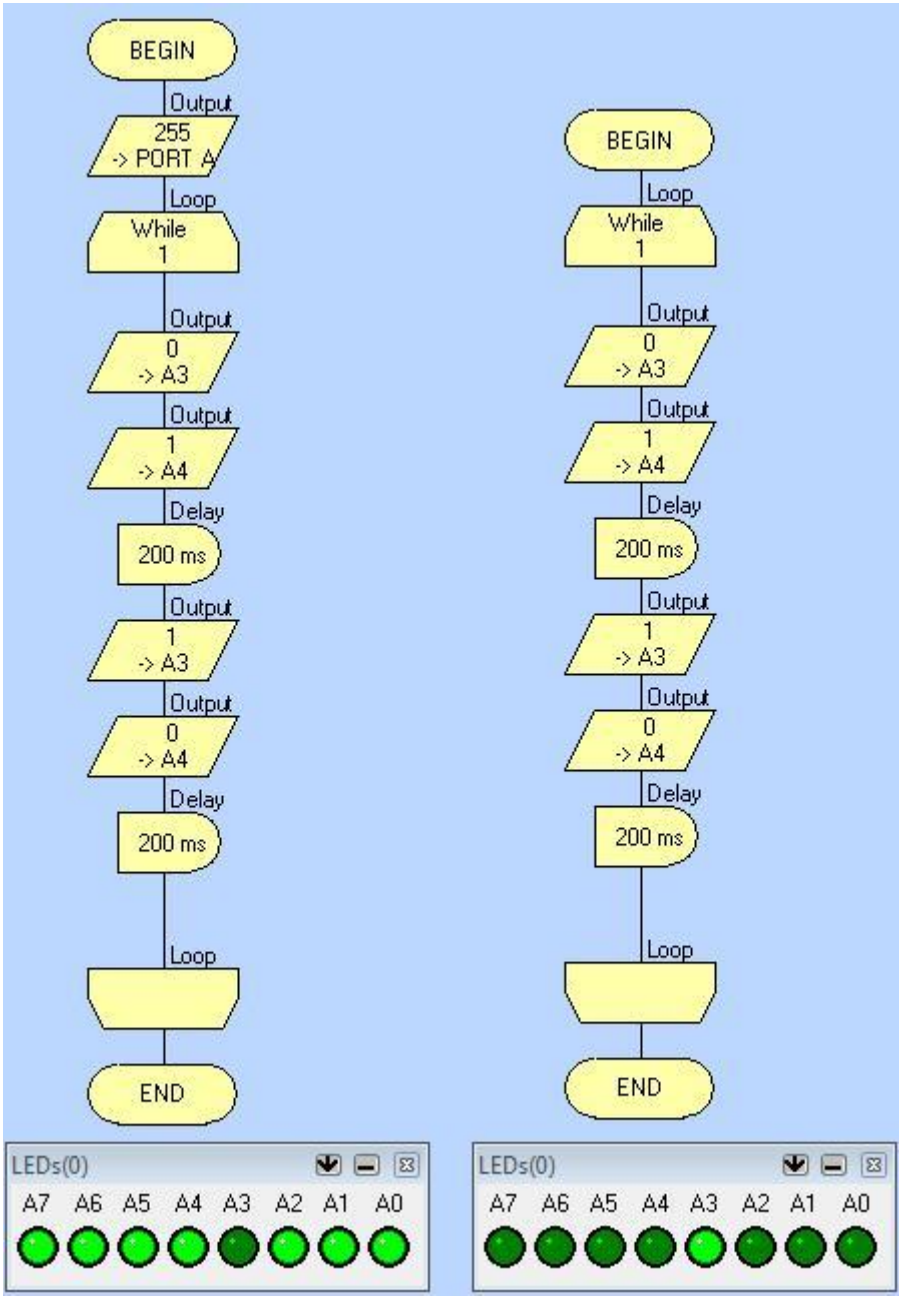


Рис. 2.29. Два варианта программы

Затем, скажем, создать программу с переключением между вариантами, оформляя варианты как две подпрограммы. На этом этапе можно убедиться, что программа, выполняющая одинаковые действия, может быть написана разными способами.

Вот один из вариантов.

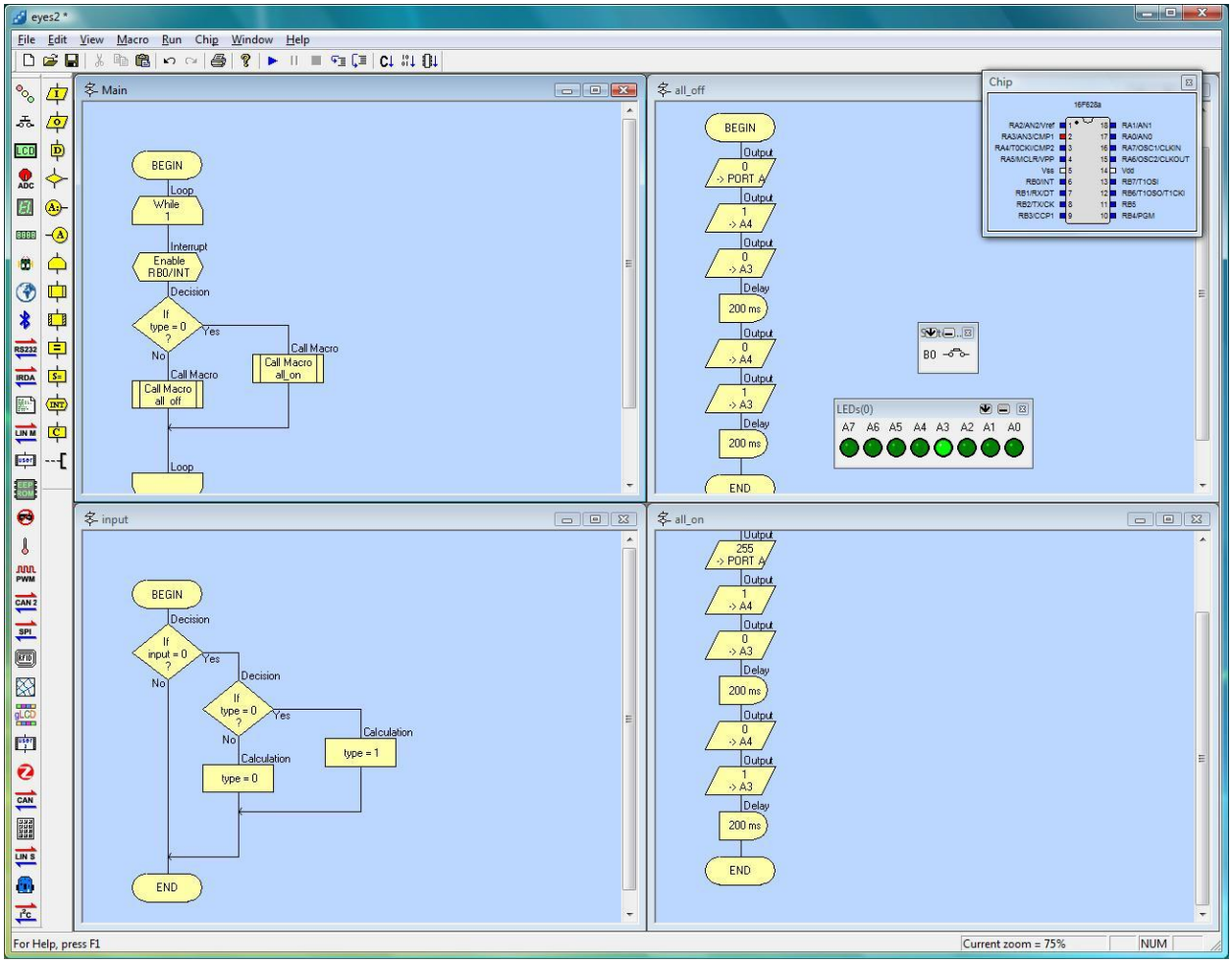


Рис. 2.30. Один из вариантов программы

Здесь используется выключатель, соединенный с выводом 0 порта B. Это позволяет применить прерывание при изменении состояния вывода RB0.

Несложно заметить, что часть программы с «мигающими» светодиодами остается одинакова в обеих подпрограммах, а меняется только «фон». Следовательно, можно убрать из подпрограмм эту часть, перенеся ее в основную программу.

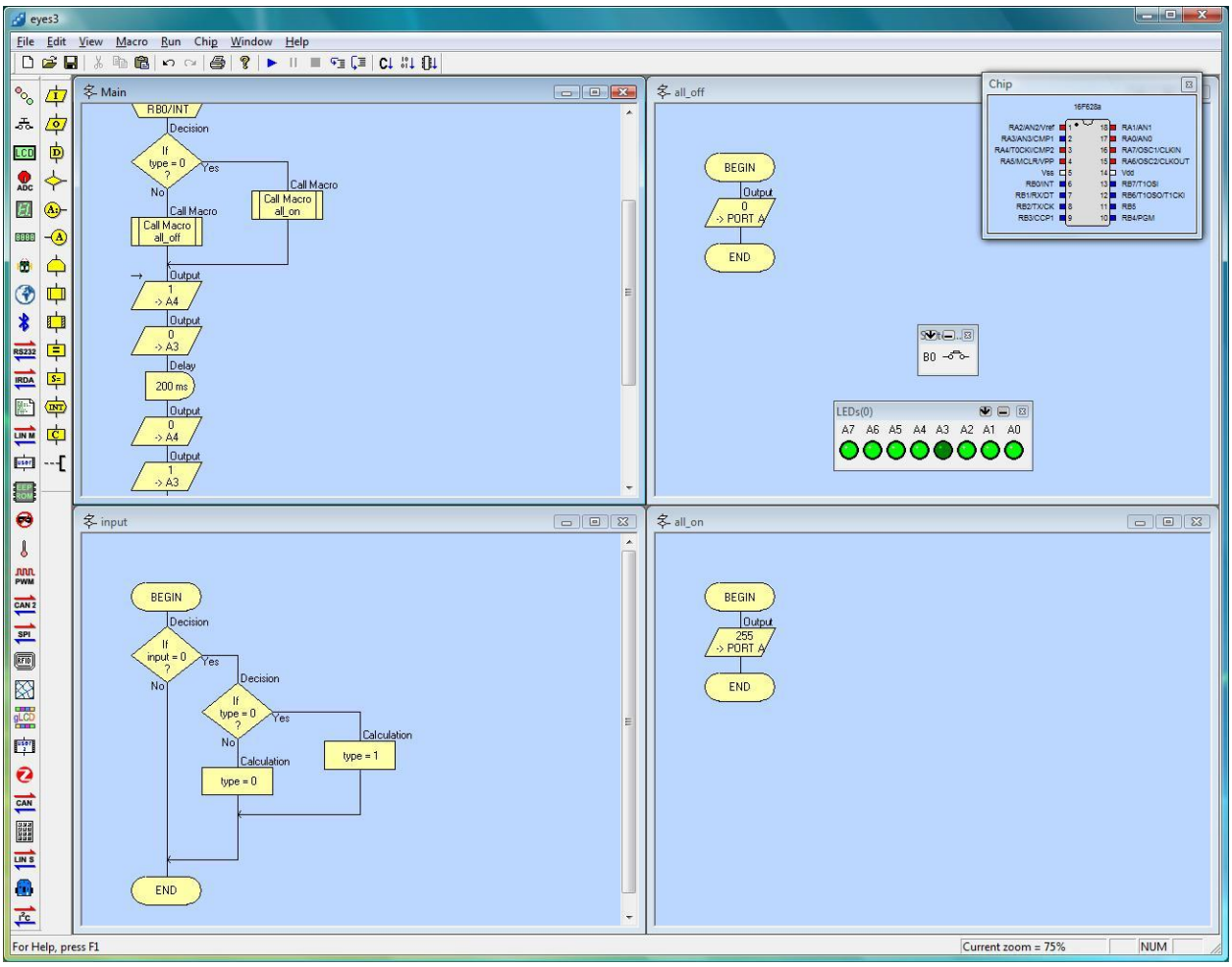


Рис. 2.31. Второй вариант программы

Варьируя разные написания программ, можно проверить, как эти вариации сказываются, например, на общем количестве кода, то есть, насколько рационально используется памяти микроконтроллера.

Затем можно перейти к той части, где меняется скорость «мигания», заменив значение в компоненте **Delay** переменной.

В конечном итоге можно постараться создать программу, которая в точности будет соответствовать авторской в книге Шпака Ю.А. При этом можно просматривать полученный результат в виде кода на языке Си и сравнивать его с листингом.

Заключение

Обязательно ли использовать учебник? Нет. Но, если вы используете учебник, то полезно будет повторить рекомендованные им программы в FlowCode. Чем привлекательна книга Шпака Ю.А., так это тем, что помимо языка Си вы узнаете много о самих контроллерах, да и язык Си в применении к контроллерам имеет особенности, которые лучше узнать от человека в этом хорошо разбирающегося.

Я не настаиваю на полной и незыблемой плодотворности того варианта изучения работы с микроконтроллером, который описан здесь, но считаю его вполне жизнеспособным. А для тех, кто пользуется операционной системой Linux, напомню, что есть полнофункциональная программа, похожая на FlowCode, с графическим языком программирования, которая называется KTechlab.

Для любителей, как мне кажется, полезно использовать все средства, включая книгу Шпака, чтобы научиться создавать свои программы для микроконтроллеров. Но еще полезнее самостоятельно придумывать устройства, где возможности микроконтроллера проявляются в полной мере, а сами устройства ближе к вашим интересам, чем приводимые учебные примеры.

Наверное, если вы еще не начинали освоение микроконтроллеров, не следует сразу браться за сложные устройства, представляющие для вас несомненный интерес, сделайте несколько простых устройств, изучайте микроконтроллер и программирование, а параллельно думайте о том устройстве, всегда есть над чем поразмышлять, которое вы сделаете спустя немного времени, когда освоитесь с созданием устройств на базе микроконтроллеров.