

ПРОГРАММИСТ



Введение в Scheme. Часть 2



Построение сети масштаба
микрорайона. Часть 1



MP3 изнутри



Делаем динамические тени на
OpenGL. Часть 1

И многое другое...

Редакция:
Utkin, JTG, Алексей Шульга, Ксения Павлова,
Антон Бердников, Егор Горохов, Сергей Бадло

Дизайн и верстка:
Егор Горохов, Сергей Бадло

Авторский состав:
Utkin, WildHunter, Алексей Шишкин,
Виталий Иванов, Вадим Буренков,
Александр Терлецкий, Вячеслав Мовила,
Александр Демьяненко, Александр Архипов,
Сергей Матрунчик, Сергей Бадло

Контакты:
Авторские статьи направляйте на
maindatacentr@gmail.com
Вопросы и предложения для редакции
reddatacentr@gmail.com

Информационная поддержка:
Международная Академия Информатизации
(МАИН) РК www.academy.kz
Журнал «Радиолобитель»
www.radioliga.com
Клуб ПРОграммистов
www.programmersforum.ru

Примечание:
Издание некоммерческое. Все материалы,
товарные знаки, торговые марки и логотипы,
упомянутые в журнале, принадлежат их
владельцам. Статьи, поступающие в редакцию,
рецензируются. Мнение авторов не всегда
совпадает с мнением редакции. Перепечатка
материалов журнала и использование их в
любой форме, в том числе в электронных СМИ,
возможно только с разрешения редакции.
Тираж неограничен. Формат А4, 83 стр.

Учредитель:
Клуб ПРОграммистов
www.programmersforum.ru

Обложка номера:
Дизайн Егора Горохова

ТЕМА НОМЕРА

Наши разработки с.0x02

НЕВЕРОЯТНО, НО ФАКТ

Любопытные факты с.0x03

НОВОСТИ ПРОГРАММИРОВАНИЯ

Введение в Scheme. Часть 2 с.0x08

ОТДЕЛ ТЕСТИРОВАНИЯ

Маленькие помощники программиста с.0x13

Разработчики - интерфейс - пользователи. Часть 1 с.0x16

WI-FI СЕТИ

Беспроводная сеть масштаба микрорайона. Часть 1 с.0x1F

АЛГОРИТМЫ

Основы неврологии с.0x25

2D ГРАФИКА

Делаем динамические тени на OpenGL. Часть 1 с.0x2F

ЛАБОРАТОРИЯ

MP3 изну-три с.0x36

Энкодер датчика PDF на ПЛИС. Часть 2 с.0x3A

Библиотека файловой системы AT45DB161 с.0x43

ИГРОВАЯ ПЛОЩАДКА

Искусство изменения GTA с.0x48

АРХИВ

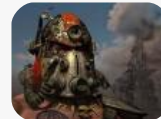
Новая рубрика с.0x4D

Как создать собственную DLL на Дельфи с.0x4E

ЮМОР

Фразеологизмы, хохмы, загадки с.0x51

От редактора. Приветствую старых и новых читателей журнала «ПРОграммист» от клуба программистов www.programmersforum.ru. Жаркий июльский месяц подготовил ряд неожиданных и жутко полезных материалов.



В этом выпуске вас ожидает... продолжение материала по *Scheme* и практика работы с нейронными сетями от **Utkin**-а. Вадим Буренков научит создавать динамические тени на примере движка динамического освещения в OpenGL.

Если вы собираетесь крупномасштабно и серьезно заниматься Wi-Fi и предоставлением Интернета, то безусловно стоит обратить пристальное внимание на опыт построения беспроводной сети масштаба микрорайона. С данным материалом в новой рубрике нашего журнала «Wi-Fi сети» дебютирует Александр. Официальная страничка проекта <http://airnet.sytes.net>.

Маленькие помощники программиста, кто они? Каковы предпосылки их появления? На эти вопросы постарается ответить Алексей Шишкин в рубрике «Отдел тестирования». Философия взаимодействия пользователя и разработчика заключена в интерфейсе. К обсуждению данной темы предлагает присоединиться Александр Демьяненко.

Рубрика «Лаборатория» по-прежнему радует эксклюзивными проектами... Многие начинающие и опытные разработчики рано или поздно сталкиваются с необходимостью использования внешней памяти в своих микроконтроллерных устройствах. Нет ничего проще, ведь Вячеслав Мовила разработал библиотеку файловой системы последовательного доступа, значительно упрощающую данную задачу. MP3 у всех на слуху. А что у него внутри и как с ним работать в своих программах? В этом вам поможет Александр Терлецкий. Ну и конечно-же, заключительная часть статьи по работе с энкодером датчика PDF на ПЛИС и тестовая утилита визуализации положения ротора на СИ от Сергея Бадло.

Вы заядлый игроман? Вам-бы хотелось добавить себе ресурсы и артефакты в любимой игрушке? Рубрика «Игровая площадка» познакомит с искусством изменения GTA от Виталия Иванова. На примере данной статьи вы с легкостью осуществите свою мечту...

Рубрики журнала (плавающие)

- Новости программирования (новые языки, концепции, среды)
- Отдел тестирования
- Общие вопросы (вопросы правового использования, личные мнения и т.д.)
- Алгоритмы
- Юмор (специфические хохмы программистов)
- Реализация (описание различных тонкостей программирования)
- Разработка (создание программных проектов от этапа ТЗ до работающей программы)
- Переводные материалы
- Рубрика про железки / Лаборатория

Общие требования к материалам

У нас нет категоричных требований к оформлению, но в связи с особенностями верстки (используется свободное ПО «**SCRIBUS**») и облегчения труда редакторов, есть некоторый желательный минимум:

- статья должна иметь выраженную структуру с разделами и содержать - название статьи, сведения об авторах, экскурс или введение, сведения об используемых средствах разработки, теоретическую и/или практическую часть, заключение (выводы, чего добились) и ресурсы к статье
- текст статьи без табуляции в формате MS Word, [VK WordPad](#) или обычным текстовым файлом, шрифт Arial 10
- все рисунки, таблицы должны иметь упоминание в тексте и иметь подпись
- рисунки к статье должны прилагаться **в виде отдельных файлов** в формате PNG, TIF
- разделы статьи отделять двумя <ENTER>
- по присланным материалам автор получает рецензию и корректирует статью согласно замечаниям
- шаблон для написания статьи можно взять [тут](#)
- бесплатный редактор VK WordPad можно взять [тут](#)

До новых встреч на страницах нашего журнала!
С уважением, Редакция

Здравствуйтесь, уважаемые читатели. Пора летних отпусков таки внесла свои коррективы в работу Редакции, что немедля сказалось на графике выхода журнала. Приносим свои извинения за задержку. Часть новостей потеряла свою актуальность и была в срочном порядке заменена на новые. Были добавлены некоторые технические моменты, что, надеемся, никак не скажется на вашем интересе к рубрике...



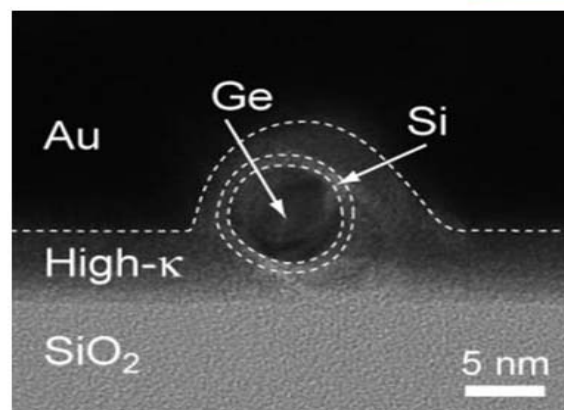
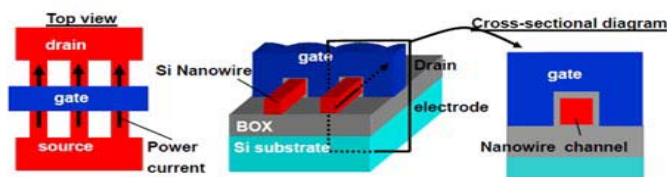
Сергей Бадло

by **raxp** <http://raxp.radioliga.com>

Хорошая новость для всех, кто еще не успел подать доклад на главную конференцию программистов России и Восточной Европы «Разработка ПО 2010». Срок подачи докладов продлен до 9 августа. Крупнейшая в Центральной и Восточной Европе конференция «Разработка ПО 2010» пройдет в Москве с 11 по 15 октября 2010 года. Одно из центральных событий конференции – семинар легендарного Бьярна Страуструпа, создателя языка программирования C++, который состоится 13 октября 2010 года. Г-н Страуструп, который в этом году впервые посетит Россию, проведет мастер-класс «Виртуозное программирование».

Зарегистрироваться на семинар можно на сайте конференции <http://cee-secr.org/participants/registration>. Вы тоже можете выступить с докладом на конференции «Разработка ПО 2010», и получить возможность принять в ней участие бесплатно. На конференцию принимаются доклады по следующим тематическим направлениям: разработка ПО на практике; технологии и технологические решения; человеческий фактор в разработке программного обеспечения; управление проектами; процессы разработки ПО. Доклады отбираются на конкурсной основе. С правилами подачи докладов можно ознакомиться [здесь](http://cee-secr.org/submission) <http://cee-secr.org/submission>.

16-нанометровый транзистор из нанопровода удалось разработать специалистам компании Toshiba. Общеизвестно, что при уменьшении размеров планарных транзисторов ток утечки между истоком и стоком в закрытом состоянии становится недопустимо большим. В транзисторе из кремниевого нанопровода ток утечки уменьшен практически на порядок, поскольку тонкий канал управляется окружающим его



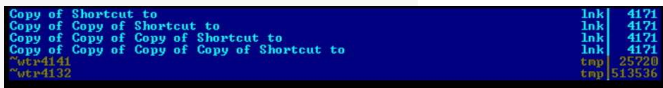
затвором. К сожалению, паразитное сопротивление, особенно в районе боковых стенок затвора, снижает ток в открытом состоянии. По словам Toshiba, эту проблему удалось преодолеть за счет оптимизации процесса формирования затвора и существенного уменьшения толщины боковых стенок, с 30 нм до 10 нм.

Дешевых интернет-провайдеров отключат от Сети на Украине. Нацкомиссия по вопросам регулирования связи обязала участников рынка интернет-доступа отключить от Сети всех провайдеров, не имеющих регистрации и соответствующей лицензии. Хотя законных оснований для этого нет, но крупные операторы не пойдут на конфликт с регулятором и поднимут цены на свои услуги как минимум на 20%.

Microsoft предупреждает о случаях эксплуатации новой бреши в [Windows 2000/XP/Vista/7](#), связанную с инфицированными USB-накопителями. Уязвимость проявляется в том, что ОС Windows обычно работает с файлами-ярлыками, которые традиционно размещаются на Рабочем столе или в меню Пуск системы и как правило эти ярлыки указывают на конкретные локальные файлы или программы.

Методика заражения

Вы должны принять во внимание, что этот вирус заражает систему необычным путем (без файла autorun.inf), через уязвимость в lnk-файлах. Так вы открываете зараженную флешку с помощью проводника или другого менеджера способного отображать иконки (например TotalCommander) и запускаете вирус, компьютер заражен. Ниже на скриншоте вы можете увидеть содержимое флешки FAR (он не заражает систему):



Следующий скриншот демонстрирует содержимое lnk файлов:



Процесс заражения развивается по следующему пути:

1. Два файла ([mrxnet.sys](#) и [mrxcls.sys](#), один из них действует как драйвер-фильтр файловой системы, а второй включает в себя вредоносный код) располагаются в директории `%SystemRoot%\System32\drivers`. Ниже

представлен скриншот программы GMER
отображающей вирусные драйверы. Данные
драйверы подписаны (имеют цифровые подписи)
Realtek Semiconductor Corp.

2. Два файла (**oem6c.pnf** и **oem7a.pnf**, содержимое которых зашифровано) располагаются в папке **%SystemRoot%\inf**. Вирус запускается сразу после заражения, так что перезагрузка не требуется. Драйвер-фильтр скрывает файлы **~wtr4132.tmp** и **~wtr4141.tmp** и соответствующие **lnk** файлы. Поэтому пользователи не заметят новых файлов на USB-устройстве.

3. Также руткит добавляет потоки в системные процессы, и в тоже время прячет модули которые запускаются в потоках. Антируткит GMER фиксирует следующие аномалии:

4. Руткит устанавливает ловушки в системных процессах

Как защититься

Данные советы не исправят уязвимость, но они помогут снизить риск успешность эксплуатации уязвимости до времени выхода заплатки:

- Отключение отображения иконок ярлыков
(Пуск-Выполнить, в открывшемся окошке

написать Regedit, нажать ОК. Пройти в данную ветку реестра

`HKEY_CLASSES_ROOT\lnkfile\shell\IconHandler`

Щелкните File (Файл) в меню и выберите Export (Экспорт). В окне экспорта введите `LNK_Icon_Backup.reg` и нажмите Save (Сохранить). Выберите параметр Default в правой стороне окна Редактора реестра. Нажмите Enter чтобы отредактировать значение. Удалите значение, т.е. поле станет пустым, нажмите Enter. Перезапустите процесс `explorer.exe` или перезагрузите компьютер)

- Отключение службы WebClient

(Пуск-Выполнить, впишите `Services.msc` и нажмите ОК. Правой кнопкой щелкните по службе `WebClient` и выберите свойства. Измените тип запуска на Отключено. Нажмите ОК и закройте приложение)

Когда вы отключите службу WebClient, перестанут посылаться запросы Web Distributed Authoring and Versioning (WebDAV), и любая служба зависящая от WebClient также не будет запускаться. Пока в корпорации не говорят, когда будет выпущено исправление. Пока же пользователи могут защищаться путем отключения отображения ярлыков на съемных носителях или за счет отключения сервиса WebClient. Оба метода выполняются через реестр Windows.

В ОС Windows 8 будут включены:



соединения с мониторами и телевизорами, поддержка USB 3.0 и Bluetooth 3.0. Еще одной возможностью, которую наверняка оценят обладатели Windows-ноутбуков, станет поддержка динамической яркости, позволяющая в режиме реального времени менять освещение дисплея ноутбука в зависимости от объема света, падающего на него.

Олимпийские игры для роботов стартовали в Китае. Роботы из разных стран соревнуются между собой в футболе, прыжках в высоту, баскетболе, боксе и даже в танцах и игре на барабанах. В Технологическом университете



города Харбин в Китае проходят первые в истории Олимпийские игры среди человекоподобных роботов. Размеры спортсменов варьируются от 20 см до полуметра. Большинство роботов собраны вручную, из простых деталей

Новая технология 3D-видео от Microsoft, как замена несуразно дорогим 3D-панелям с 3D-очкам. Для создания эффекта объемного изображения предложено несколько неожиданное решение - подсветка ЖК-дисплея. При этом новая система прицельно направляет картинку в каждый глаз по отдельности.



Система, разработанная инженерами подразделения прикладных научных исследований компании Microsoft (Applied Sciences Group), включает в себя два основных компонента. Первый - массив клинообразных линз. Толщина каждой линзы составляет 11 мм в верхней части и 6 мм в основании. Это позволяет особым образом фокусировать свет и выдавать картинку, используя свойства изометрии. Размещаются новые линзы поверх LED-монитора. Набор скоростных светодиодов позволяет выдавать через верхнюю поверхность линзы направленный пучок света. В результате экран

позволяет смотреть 3D-видео, и даже двум зрителям сразу, при этом они не должны думать о том, чтобы попасть в зону доступности картинки.

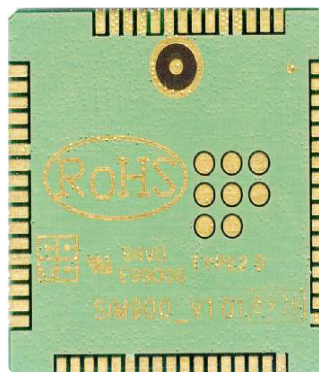
Второй - специализированная трехмерная камера, которая видит, где именно располагается зритель (или зрители), отслеживает направление взгляда пользователя и последовательно направляет в его глаза разные части стереоскопического изображения.

Каждый российский участковый обзаведется личной интернет-страничкой, где будут размещены его персональные и контактные данные. За счет «интернетизации» участковых МВД хочет повысить качество обратной связи между населением и милицией. Однако за счет каких средств будут созданы более 54 тыс.



сайтов, пока не понятно. Такие сайты будут содержать интерактивные разделы, где граждане смогут задать участковому вопрос. Появится и раздел «Добровольный помощник», где можно будет сообщить о правонарушении, в том числе и анонимно. По задумке МВД, создание персональных сайтов для участковых должно простимулировать социальную активность граждан.

Новый малогабаритный GSM/GPRS модуль SIM900 имеет следующие характеристики: четыре диапазона GSM 850/900/1800/1900 МГц, класс передачи данных GPRS multi-slot class 10/8, класс мощности 4 (2 Вт в диапазонах 850/900 МГц), класс мощности 1 (1 Вт в диапазонах



1800/1900MHz), размеры 24x24x3 мм, масса 7 г, управление AT командами (GSM 07.07 и фирменные AT команды SIMCOM), встроенный стек TCP/IP, напряжение питания 3.4...4.5 В. Температурный диапазон -30+80 °C. Официальный анонс компании SIMCOM здесь [http://www.mtsystem.ru/documents/sim900%20announcement\(%20dec2009\).pdf](http://www.mtsystem.ru/documents/sim900%20announcement(%20dec2009).pdf)

Прототип гибкого LCD-дисплея, работающего и как устройство ввода представила на выставке SID 2010 компания Toshiba. Ее 8.4-дюймовая LCD-панель имеет толщину всего 0.1 мм и разрешение SVGA. Управление



деформацией осуществляется изящно и просто. Открыв на дисплее приложение Google Earth, пользователь может в реальном времени изгибать экран и тем самым масштабировать изображение карты. Стоит пользователю выгнуть дисплей - и изображение приближается, вогнуть - отдаляется. Именно это интересное свойство и демонстрируется на фото с помощью карты Google Earth.

Прецизионные MEMS гироскопы для высокоточных устройств навигации, позиционирования (ГЛОНАСС/GPS системы) и стабилизации

положения начала производить Silicon Sensing Systems под торговой маркой PinPoint. Миниатюрный корпус гироскопа размером 6x5x1.2 мм содержит чувствительный элемент VSG5, основанный на балансирующем вибрирующем кремниевом кольце 5-го поколения, который обеспечивает наилучшую по сравнению с аналогами этого класса точность

CRM100



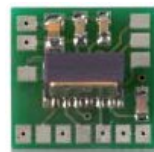
CRM200



измерения и минимальные отклонения показаний. Микросхема PinPoint может использоваться в условиях с агрессивными средами. Потребляя ток всего до 4 мА при напряжении 3 В гироскоп может успешно применяться в портативных



устройствах с питанием от батарей. Это позволяет использовать его, например в персональных и автомобильных навигаторах.

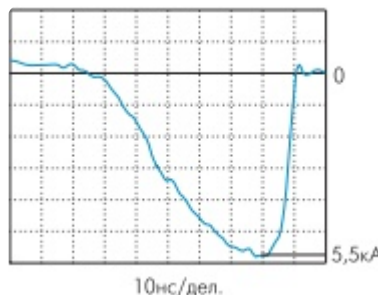


Полупроводниковые SOS генераторы ЭМИ были разработаны в Уральском отделении института электрофизики РАН (Екатеринбург), проникающая способность излучения которых намного выше, чем у ВМГ. Принцип действия SOS генераторов основан на эффекте наносекундной коммутации сверхплотных токов в полупроводниковых приборах.

SOS эффект представляет собой качественно новый вариант коммутации тока - развитие процесса стремительного падения тока происходит не в низколегированной базе полупроводниковой структуры, как в других приборах, а в ее узких высоколегированных областях. База и p-n переход остаются при этом заполненными плотной избыточной плазмой, концентрация которой приблизительно на два порядка превышает исходный уровень легирования. Эти два обстоятельства и приводят к сочетанию высокой плотности коммутируемого тока с наносекундной длительностью его отключения.

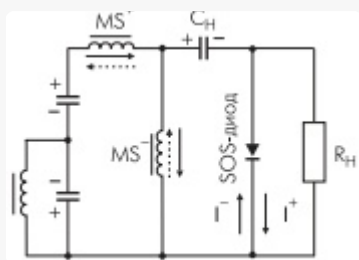


Другое важное свойство SOS-эффекта - в том, что стадия срыва тока характеризуется автоматическим равномерным распределением



напряжения по последовательно соединенным полупроводниковым структурам. Это позволяет создавать

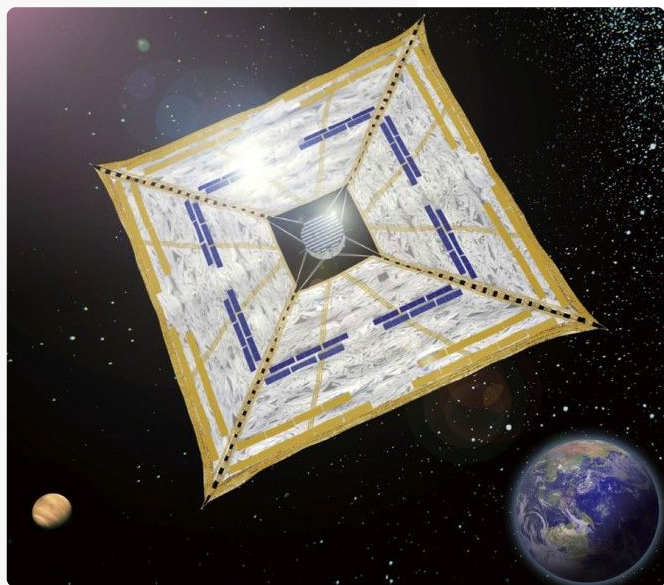
прерыватели тока с напряжением мегавольтного уровня путем простого последовательного соединения SOS структур. SOS эффект



обнаружен в 1991 году в обычных высоковольтных выпрямительных полупроводниковых диодах подбором

сочетания плотности тока и времени накачки. В дальнейшем была разработана специальная полупроводниковая структура сверхжестким режимом восстановления, на основе которой электродами. Значение коммутируемого тока – 5.5кА, время его срыва (падения с 0.9 до 0.1 амплитуды) – 4.5 нс. Скорость коммутации – 1200 кА/мкс, что приблизительно на три порядка превышает токовый градиент в обычных быстродействующих тиристорах. Самый мощный из разработанных на сегодня SOS-диодов при площади структуры 4 см² имеет рабочее напряжение 200 кВ и коммутирует ток 32 кА, что соответствует коммутируемой мощности 6 ГВт...

Солнечный парус развернули в космосе японцы. Экспериментальный аппарат IKAROS стартовал 21 мая с космодрома Танэгасима в составе миссии PLANET-C по направлению к Венере. Миссия будет считаться успешной, если, используя импульс солнечного света, аппарат сможет направиться к Венере за следующие полгода. Разворачивать парус аппарат начал еще



3 июня, увеличив скорость своего вращения до 25 оборотов в минуту. При содействии центробежной силы четыре груза массой по 0.5 кг отошли от центра IKAROS, увлекая за собой сложенную мембрану, после чего парус медленно расправлялся. Раскрытие паруса завершено 10 июня. В дальнейшем держать парус в натянутом состоянии станут все те же центробежные силы, IKAROS будет непрерывно вращаться вокруг своей оси на скорости 1-2 оборота в минуту. Такая технология значительно проще, чем использование тяжелых штанг или рей. Парус космического аппарата изготовлен из алюминированного полиимида толщиной 7.5 мкм и покрыт тонким слоем солнечных батарей толщиной в 25 мкм. Когда фотоны света ударяются в парус, они поглощаются или отражаются, сообщая ему дополнительный импульс силы, которая разгоняет космический аппарат. Фотоны являются очень маленькими частицами и их импульс весьма мал, но, учитывая их огромное количество, можно надеяться, что в течение долгого времени космический аппарат накопит достаточную для полета скорость. 15 июня парусник сфотографировал сам себя, выпустив в свободное плавание отделяемую камеру (возвращение ее не предусмотрено). Этот цилиндрик длиной и диаметром по 6 сантиметров был выброшен в сторону пружинной. Камера передавала сигнал на сам космический аппарат, а уже тот с помощью своего более мощного передатчика транслировал картинку на Землю.

НКРС Украины одобрила новый проект Регламента аматорской радиосвязи Украины, который определяет порядок пользования радиочастотным ресурсом для радиолюбителей. Так, чтобы получить разрешение, радиолюбитель должен обратиться в региональный филиал УГЦР лично или рекомендательным письмом с просьбой о разрешении на эксплуатацию АРС. К заявлению прилагаются справка о квалификационном экзамене, акт техосмотра радиостанции, копия паспорта заявителя. Не позднее 15 дней от даты регистрации заявки УГЦР направляет заявителю счет на оплату работ по оформлению разрешения. Разрешение выдается не позднее, чем через 3 рабочих дня после того, как заявитель предъявит в центр документы, подтверждающие эту оплату.

Здравствуйте, дорогие читатели. Как и обещал, сегодня мы продолжим рассмотрение наиболее распространенных команд языка *Scheme* и механизмов, облегчающих управление процессом выполнения, а также познакомимся ближе со средой разработки PLT Scheme.



Продолжение. Начало цикла смотрите в четвертом выпуске журнала...

by **Utkin** www.programmersforum.ru

Нужно помнить, что данные предикаты нежелательно использовать на неточных числах – ошибки округления могут приводить к обратным результатам. Кстати, определить является ли число точным или не точным можно определить с помощью предикатов:

```
(exact? z)      Тест на точность
(inexact? z)    Тест на неточность
```

Для одного и того же числа один из этих предикатов всегда результат **#t**, тогда как второй **#f**. Для сравнения чисел можно использовать следующие предикаты, думаю назначение их понятно и так:

```
(= x1 x2 x3 ...)
(< x1 x2 x3 ...)
(> x1 x2 x3 ...)
(<= x1 x2 x3 ...)
(>= x1 x2 x3 ...)
(= 1 1 1)      #t
(= 1 2 1)      #f
```

Аналогично, предикатам типов числа, не стоит применять сравнения на неточных числах, поскольку ошибки округления могут дать противоположный результат для почти равных чисел. Еще предикаты для работы с числами:

```
(zero? z)      Проверка на нуль
(positive? x)   Проверка является ли число положительным
(negative? x)   Проверка является ли число отрицательным
(odd? n)        Проверка является ли число не четным
(even? n)       Проверка является ли число четным
```

Поиск минимального и максимального чисел:

```
(max x1 x2 ...)
(min x1 x2 ...)

// Примеры:
(max 3 4)      Результат 4
```

```
(max 3.9 4 9 48 99 5)    Результат 99.0 При этом число
                          будет преобразовано в неточное
(min 54 565 9L0)         Результат 9.0
(min 4 4)                 Результат 4
```

Нужно соблюдать осторожность при сравнении почти равных неточных чисел, поскольку правильный результат работы данных предикатов не гарантирован. Теперь напомним стандартные арифметические операции:

```
(+ z1 ...)          Сложение
(* z1 ...)          Умножение
(- z1 z2)           Вычитание
(- z)               Унарный минус
(- z1 z2 ...)       Вычитание (с произвольным числом аргументов)
(/ z1 z2)           Деление (деление на нуль не допускается)
(/ z)               Деление 1 / z, при этом образуется дробь
(/ z1 z2 ...)       Деление z1 на произвольное число аргументов
```

Теперь примеры (поскольку некоторые результаты не очевидны):

```
(+ 3 4)            Результат 7
(+ 3)              Результат 3
(+)               Результат 0
(* 4)             Результат 4
(*)              Результат 1
(- 3 4)          Результат -1
(- 3 4 5)        Результат -6
(- 3)            Результат -3
(/ 3 4 5)        Результат 3/20 (3 разделить на 4*5)
(/ 3)            Результат 1/3
```

Кстати, дроби являются точными числами, поэтому их удобно использовать в предикатах для работы с числами (гарантируются однозначные результаты сравнений, отношений и т.д.). Дополнительные операции над числами:

```
(abs x)            Абсолютное значение числа
```

Дополнительные операции деления:

```
(quotient n1 n2)
(remainder n1 n2)
(modulo n1 n2)
```

При этом `n2` не должен быть равен нулю (независимо от точности числа). Если `n1 / n2` есть целое:

<code>(quotient n1 n2)</code>	Результат <code>n1/n2</code>
<code>(remainder n1 n2)</code>	Результат <code>0</code>
<code>(modulo n1 n2)</code>	Результат <code>0</code>

Если в результате `n1 / n2` образуется не целое число:

<code>(quotient n1 n2)</code>	Результат <code>nq</code>
<code>(remainder n1 n2)</code>	Результат <code>nr</code>
<code>(modulo n1 n2)</code>	Результат <code>nm</code>

где: число `nq` - `n1/n2`, округленное к нулю, $0 < |nr| < |n2|$, $0 < |nm| < |n2|$, `nr` и `nm` отличаются от `n1` множителем `n2`, `nr` получит знак `n1`, `nm` получит знак `n2`.

Особенностью этих функций (по сути, вариации на тему нахождение остатка от деления) является факт, того, что они возвращают точные числа, в случае если их входящие параметры также являются точными:

<code>(modulo 13 4)</code>	Результат <code>1</code>
<code>(remainder 13 4)</code>	Результат <code>1</code>
<code>(modulo -13 4)</code>	Результат <code>3</code>
<code>(remainder -13 4)</code>	Результат <code>-1</code>
<code>(modulo 13 -4)</code>	Результат <code>-3</code>
<code>(remainder 13 -4)</code>	Результат <code>1</code>
<code>(modulo -13 -4)</code>	Результат <code>-1</code>
<code>(remainder -13 -4)</code>	Результат <code>-1</code>
<code>(remainder -13 -4.0)</code>	Результат <code>-1.0</code>

Обратите внимание, результат не точное число.

Следующие функции возвращают наибольший общий делитель или наименьший общий множитель их параметров. Результат является всегда неотрицательным:

<code>(gcd 32 -36)</code>	Результат НОД (наибольший общий делитель) <code>4</code>
<code>(gcd)</code>	Результат <code>0</code>
<code>(lcm 32 -36)</code>	Результат НОМ (наименьший общий множитель) <code>288</code>
<code>(lcm 32.0 -36)</code>	Аналогично Результат <code>288.0</code> , но теперь результат будет неточным
<code>(lcm)</code>	Результат <code>1</code>

Следующие ниже функции возвращают числитель,

и знаменатель дроби (знаменатель всегда положительное число):

<code>(numerator q)</code>	
<code>(denominator q)</code>	
 // Примеры:	
<code>(numerator (/ 6 4))</code>	Результат <code>3</code>
<code>(denominator (/ 6 4))</code>	Результат <code>2</code>
<code>(denominator (exact->inexact (/ 6 4)))</code>	Результат <code>2.0</code> , для получения неточного числа используется явное преобразование. Естественно удобно и для дробей:
<code>(numerator 6/4)</code>	Результат <code>3</code> (рассматривается дробь <code>3/2</code> , и только потом берется знаменатель)

Следующая группа функций:

<code>(floor x)</code>	всегда возвращает целые числа.
<code>(ceiling x)</code>	<code>Floor</code> возвращает наибольшее целое число не большее чем <code>x</code> .
<code>(truncate x)</code>	<code>Ceiling</code> возвращает наименьшее целое число не меньше чем <code>x</code> .
<code>(round x)</code>	<code>Truncate</code> возвращает целое число, которое наиболее ближе к <code>x</code> , и абсолютная величина которого не больше абсолютной величины <code>x</code> .
	<code>Round</code> возвращает целое число путем округления <code>x</code> (половина икса округляется в большую сторону):

<code>(floor -4.3)</code>	Результат <code>-5.0</code> (не точные)
<code>(ceiling -4.3)</code>	Результат <code>-4.0</code>
<code>(truncate -4.3)</code>	Результат <code>-4.0</code>
<code>(round -4.3)</code>	Результат <code>-4.0</code>
<code>(floor 3.5)</code>	Результат <code>3.0</code>
<code>(ceiling 3.5)</code>	Результат <code>4.0</code>
<code>(truncate 3.5)</code>	Результат <code>3.0</code>
<code>(round 3.5)</code>	Результат <code>4.0</code> - не точное число
<code>(round 7/2)</code>	Результат <code>4</code> - точное число
<code>(round 7)</code>	Результат <code>7</code>
<code>(rationalize x y)</code>	возвращает самое простое рациональное число, отличающееся от <code>x</code> не больше, чем <code>y</code>

Рациональное число `x` более простое чем другое рациональное число `y` если $x = p1/q1$ и $y = p2/q2$ и $|p1| < |p2|$ и $|q1| < |q2|$. Таким образом, `3/5` более простое число чем `4/7`. Хотя не все рациональные числа сопоставимы в этом упорядочении (например, `2/7` и `3/5`), любой числовой интервал содержит рациональное число, которое более просто, чем любое рациональное число в том же интервале. Известно также, что `0`

= 0/1 - самые простые рациональные числа из всех:

```
(rationalize
  (inexact->exact .3) 1/10)
(rationalize .3 1/10)
```

Результат 1/3, точное число

Результат 0.3333333333333333

Трансцендентальные функции:

```
(exp z)
(log z)
(sin z)
(cos z)
(tan z)
(asin z)
(acos z)
(atan z)
(atan y x)
```

Здесь все просто, стоит отметить **log** - имеется ввиду натуральный (а не десятичный), **atan** с двумя параметрами использует знаки **x, y** для определения квадранта, к которому принадлежит результат.

(sqrt z) - вычисляет квадратный корень, включая мнимую часть числа:

```
(sqrt -1)
(expt z1 z2)
(expt 2.1 3.3)
(expt 2.1 -3.3)
```

Результат 0+1i (всегда мечтал в Паскале получить корень из -1)

возведение в степень, довольно-таки мощная функция, единственное ограничение - в степень запрещено возводить ноль

Результат 11.569741950241465

Результат 0.08643235124004903

Следующие функции предназначены для поддержки комплексных чисел:

```
(make-rectangular x1 x2)
(make-polar x3 x4)
(real-part z)
(imag-part z)
(magnitude z)
(angle z)
```

Здесь **x1-x4** вещественные числа, **z** комплексное.

```
(make-rectangular x1 x2)
(make-polar x3 x4)
(real-part z)
(imag-part z)
(magnitude z)
(angle z)
```

Результат комплексное число

Результат комплексное число

Результат реальную часть числа (x1)

Результат мнимую часть числа (x2)

Результат |x3|

Результат xangle

Точные и неточные числа можно преобразовывать следующим образом:

```
(exact->inexact z)
(inexact->exact z)
```

Первая функция переводит точное число в неточное, вторая выполняет обратные действия.

Естественно, такое преобразование возможно не всегда (такая ситуация считается ошибочной).

Теперь рассмотрим ввод и вывод чисел. Числа можно получать из строки следующим образом:

```
(string->number string)
(string->number string radix)
```

Здесь **radix** есть основание системы счисления (точное целое число 2, 8, 10 или 16). Для первого варианта имеется ввиду основание системы счисления равным 10. Если число **z** является неточным и может быть выраженным с использованием точки в качестве разделителя целой и дробной частями, то число будет представлено с минимально возможным числом разрядов:

```
(string->number "100")
(string->number "100" 16)
(string->number "1e2")
(string->number "15##")
```

Результат 100

Результат 256

Результат 100.0

Результат 1500.0

В случае, если строку в число преобразовать не удастся, результат функции **#f**.

Пары и списки

Парой (иногда называется точечной парой) называется структура данных, представляющая собой запись, состоящую из двух полей, первое из которых называется **car**, а вторая **cdr** (названия сложились исторически). Пара создается функцией **cons**. Доступ к полям пары осуществляется одноименными функциями **car** и **cdr**. Присваивание значений полям пары осуществляется с помощью процедур **set-car!** и **set-cdr!**. Основное назначение пар это образование и представление списков. Список может быть определен рекурсивно, как пустой список (список, не содержащий элементов) или как пара поле **cdr** которого является списком. Если **X** является списком, то:

- в **X** содержится пустой список
- если список находится в **X**, то любая пара, поле **cdr** которой содержит список, находится также в **X**.

Объекты в полях **car** последовательных пар списка считаются элементами списка. Например,

двухэлементный список это пара, `car` которой первый элемент списка и чей `cdr` пара, `car` которой второй элемент списка и чей `cdr` есть пустой список. Длина списка – это число элементов (которые можно выразить как число пар):

Список состоит из пар					
Список	Пара1	Car	1-й элемент списка		
		Cdr	Пара2	2-й элемент списка	
				Car	элемента
				Cdr	списка
					Пустой список

Список `(a b c d e)` можно выразить так `(a . (b . (c . (d . (e . ()))`). То есть в списке пары расположены не последовательно друг за другом, а являются вложенными объектами. Пустой список – специальный объект своего собственного типа (это не пара); у него нет никаких элементов, и его длина равна нулю. При этом все списки имеют конечную длину (точное число) и заканчиваются пустым списком.

Для представления пар используется точечная нотация, поля разделяются точкой. Например, пара `(4 . 5)` имеет поле `car` 4 и поле `cdr` 5, при этом внутреннее представление пары будет иным. Списки выражаются проще `(x1 x2 ... xn)`, где `x` – это элемент списка. Пустой список выражается, как `()`. Как уже было отмечено ранее, вся программа на языке *Scheme* является списком (как и любые ее логически законченные фрагменты), поэтому программы, и выражения можно обрабатывать также как и списки.

Существуют структуры, похожие на списки, которые не удовлетворяют данному выше определению, они называются неподходящие списки. Неподходящий список списком не является, вот его пример:

<code>(a b c . d)</code>	что эквивалентно <code>(a . (b . (c . d)))</code> , то есть неподходящие списки есть пары
<code>(define x (list 'a 'b 'c))</code>	
<code>(define y x)</code>	
<code>y</code>	Результат <code>(a b c)</code> , <code>list</code> создает список
<code>(list? y)</code>	Результат <code>#t</code> , предикат проверки списка
<code>(set-cdr! x 4)</code>	Результат не определен (побочный эффект <code>cdr</code>

Предикат `(pair? obj)` проверяет является ли объект парой:

<code>(pair? '(a . b))</code>	Результат <code>#t</code>
<code>(pair? '(a b c))</code>	Результат <code>#t</code>
<code>(pair? '())</code>	Результат <code>#f</code>
<code>(pair? '#(a b))</code>	Результат <code>#f</code>

Следующая функция возвращает пару:

`(cons obj1 obj2)` `car` пары будут представлять собой `obj1`, `cdr` соответственно `obj2`.

Квотирование в данном случае используется для того, чтобы поля пары

<code>(cons 'a '())</code>	Результат <code>(a)</code>
<code>(cons '(a) '(b c d))</code>	Результат <code>(c d)</code>
<code>(cons "a" '(b c))</code>	Результат <code>("a" b c)</code> , обратите внимание на квотирование и кавычки
<code>(cons 'a 3)</code>	Результат <code>(a . 3)</code>
<code>(cons '(a b) 'c)</code>	Результат <code>((a b) . c)</code>

не вычислялись в момент внесения в пару. Это позволяет вносить в пару как простые данные, вроде чисел, так и функции и выражения.

`(car pair)` – возвращает поле `car` пары (так и хочется сказать первое поле, но внутреннее представление пар может быть иным, поля пары могут находиться даже не в смежных областях памяти). Попытка получения `car` пустой пары вызовет ошибку:

<code>(car '(a b c))</code>	Результат <code>a</code>
<code>(car '((a) b c d))</code>	Результат <code>(a)</code>
<code>(car '(1 . 2))</code>	Результат <code>1</code>
<code>(car '())</code>	Результат ошибка вычислений
<code>(cdr pair)</code>	возвращает поле <code>cdr</code> пары
	Получение <code>cdr</code> пустого списка вызовет ошибку
<code>(cdr '((a) b c d))</code>	Результат <code>(b c d)</code>
<code>(cdr '(1 . 2))</code>	Результат <code>2</code>
<code>(cdr '())</code>	Результат ошибка вычислений
<code>(set-car! pair obj)</code>	и
<code>(set-cdr! pair obj)</code>	помещает соответственно в <code>car</code> и <code>cdr</code> пары <code>pairs</code> объект <code>obj</code>

<code>(a b c . d)</code>	что эквивалентно <code>(a . (b . (c . d)))</code> , то есть неподходящие списки есть пары	<code>x</code>	<code>x</code> равен 4)
<code>(define x (list 'a 'b 'c))</code>		<code>(eqv? x y)</code>	Результат <code>(a . 4)</code>
<code>(define y x)</code>		<code>y</code>	Результат <code>#t</code>
<code>y</code>	Результат <code>(a b c)</code> , <code>list</code> создает список	<code>(list? y)</code>	Результат <code>(a . 4)</code>
<code>(list? y)</code>	Результат <code>#t</code> , предикат проверки списка	<code>(set-cdr! x x)</code>	Результат <code>#f</code>
<code>(set-cdr! x 4)</code>	Результат не определен (побочный эффект <code>cdr</code>	<code>(list? x)</code>	Результат не определен
			Результат <code>#f</code>

Результат их работы не определен, функции имеют побочные эффекты. Очень часто во время манипуляций со списками возникает много `car` и `cdr`. Такие выражения допускается сокращать:

```
(car (cdr (cdr x)))      эквивалентно caddr
```

То есть справедливо следующее утверждение:

```
(define caddr (lambda (x) (car (cdr (cdr x)))))
```

Правило формирования функции следующее: все `car` представляются как `a`, а все `cdr` как `d` в наименовании функции, которая начинается с `c` и заканчивается `r`. При этом допускается формирование функций не более чем из четырех `car` и `cdr`.

`(null? obj)` – возвращает `#t`, если `obj` есть пустой список, `(list? obj)` – возвращает `#t`, если `obj` есть список:

```
(list? '(a b c))      Результат #t
(list? '())           Результат #f
(list? '(a . b))      Результат #f
(let ((x (list 'a)))
  (set-cdr! x x)
  (list? x))          Результат #f
```

`(list obj ...)` – создает список:

```
(list 'a (+ 3 4) 'c)  Результат (a 7 c)
(list)                Результат ()
```

`(length list)` – передает число элементов в списке:

```
(length '(a b c))      Результат 3
(length '(a (b) (c d e))) Результат 3
(length '())            Результат 0
```

`(append list ...)` – объединение списков, при этом происходит перераспределение их внутренней структуры (таким образом, чтобы результат удовлетворял определению списка):

```
(append '(x) '(y))      Результат (x y)
(append '(a) '(b c d))  Результат (a b c d)
(append '(a (b)) '((c))) Результат (a (b) (c))
(append '(a b) '(c . d)) Результат (a b c . d)
(append '() 'a)         Результат a, список был изменен
```

`(reverse list)` – возвращает список, перераспределенный в обратном порядке:

```
(reverse '(a b c))      Результат (c b a)
(reverse '(a (b c) d (e (f)))) Результат ((e (f)) d (b c) a)
```

`(list-tail list k)` – возвращает список на основе списка `list`, но без `k` первых элементов. Входящий список должен содержать не менее `k` элементов. Работу `list-tail` можно выразить функцией-эквивалентом:

```
(define list-tail
  (lambda (x k)
    (if (zero? k)
        x
        (list-tail (cdr x) (- k 1)))))
```

`(list-ref list k)` – возвращает `k`-й элемент списка. В списке должно быть не менее `k` элементов.

Нумерация элементов списка осуществляется от нуля. Плюсом пар и неподходящих списков является возможность создания древовидных структур данных с неограниченным числом узлов (пока хватит памяти компьютера).

Символы

Это специальные объекты, особенность которых заключается в том, что символы считаются эквивалентными (для предиката `eqv?`), если получены одинаковым путем. Символы в частности могут представлять идентификаторы программы, и их уникальность дает возможность использовать их во внутреннем представлении программы. Правила образования символов полностью соответствуют правилам образования идентификаторов. Не следует путать данные символы (`Symbols`) с символами строки (`Characters`). Это символы программы и не предназначены для представления строк (хотя и могут быть конвертированы в строки). В частности символы можно получить, квоотируя идентификаторы.

Следующий предикат подтверждает, все то, что сказано выше. `(symbol? obj)` – определяет является ли объект символом:

```
(symbol? 'foo)          Результат #t
```

<code>(symbol? (car '(a b)))</code>	Результат <code>#t</code>
<code>(symbol? "bar")</code>	Результат <code>#f</code> , строки представляются строковыми символами!
<code>(symbol? 'nil)</code>	Результат <code>#t</code>
<code>(symbol? '())</code>	Результат <code>#f</code>
<code>(symbol? #f)</code>	Результат <code>#f</code>
<code>(symbol? 5)</code>	Результат <code>#f</code>
<code>(symbol? '5)</code>	Результат <code>#f</code> , это не идентификатор и не выражение

Для преобразования символов в строки используется `(symbol->string symbol)`. Примеры:

<code>(symbol->string 'flying-fish)</code>	Результат <code>"flying-fish"</code>
<code>(symbol->string 'Martin)</code>	Результат <code>"martin"</code>
<code>(symbol->string</code>	
<code>(string->symbol "Malvina"))</code>	Результат <code>"Malvina"</code>

Здесь: `(string->symbol "Malvina")` выполняет обратную функцию `(string->symbol "Привет")`. Символы, в частности, предоставляют программисту возможность работать с объектами программы через операции над их идентификаторами.

Строковые символы

Строковой символ – минимальная единица представления текстовой информации. Строковые символы можно образовывать так `#\<character>` или `#\<character name>`. Вот примеры образования строковых символов:

<code>#\a;</code>	буква в нижнем регистре
<code>#\A;</code>	буква в верхнем регистре
<code>#\ (;</code>	открывающая скобка
<code>#\ ;</code>	пробел
<code>#\space;</code>	пробел (образование строкового символа по его имени)
<code>#\newline;</code>	символ перехода на новую строку

Поскольку *Scheme* во многом система, не зависящая от конкретной платформы, то получать константы строковых символов по кодам в какой-либо кодировке не допускается. Поэтому нельзя получить строковой символ `#\13`, в тоже время *PLT Scheme* позволяет получать символы национальных алфавитов и задать строковый символ `#\й` вполне допустимо. Для определения

факта того, что объект является строковым символом, используется следующая функция: `(char? obj)` – возвращает `#t`, если объект является строковым символом:

`(char? 'try)` Результат `#f`, поскольку язык проводит четкую границу между символами и строковыми символами

Все строковые символы определяются на основе их порядка следования в алфавите. Поэтому между строковыми символами существуют отношения, связанными с их расположением относительно друг друга. Для оценки строковых символов используется ряд функций:

`(char=? char1 char2)`
`(char<? char1 char2)`
`(char>? char1 char2)`
`(char<=? char1 char2)`
`(char>=? char1 char2)`

Сначала следуют цифры, потом прописные буквы, потом строчные (национальные символы «больше» латиницы). Кстати, *PLT Scheme* версии 3.50 допускает более двух символов для сравнения (на манер числовых предикатов):

`(char-ci=? char1 char2)`
`(char-ci<? char1 char2)`
`(char-ci>? char1 char2)`
`(char-ci<=? char1 char2)`
`(char-ci>=? char1 char2)`

Тоже самое, но без учета регистра:

`(char-ci=? #\A #\a)` Результат `#t`

Аналогично, функции с приставкой `-ci` в *PLT Scheme* могут содержать больше чем два аргумента:

`(char-alphabetic? char)` - возвращает `#t`, если строковой символ буква.
`(char-numeric? char)` - возвращает `#t`, если строковой символ цифра.
`(char-whitespace? char)` - возвращает `#t`, если строковой символ пробел.
`(char-upper-case? letter)` - возвращает `#t`, если строковой символ буква в верхнем регистре.
`(char-lower-case? letter)` - возвращает `#t`, если строковой символ буква в нижнем регистре.

Несмотря на то, что задавать константы строковых символов их кодами нельзя, получать коды из строковых символов разрешено: `(char->integer char)` – вернет точное целое число, которому сопоставлен данный строковой символ. Обратная ей функция `(integer->char n)`. Изменить регистр символов можно с помощью функций: `(char-upcase char)` – изменяет регистр строкового символа на верхний, `(char-downcase char)` – изменяет регистр строкового символа на нижний.

Строки

Строки это последовательности символов. В *Scheme* строки выделяются кавычками. В строки допускается включать Escape-последовательности через использование обратного слеша (`\`). Например, если известны коды символов, то строку можно задать следующим образом: `"\u65\u65"` (чего нельзя делать для строковых символов). Длина строки есть число содержащихся в ней символов (точное целое неотрицательное число). Индексация символов начинается от нуля. Аналогично операций с символами все функции имеющие приставку `-ci` есть операции со строками без учета регистра. Начнем с определения строки:

```
(string? obj) – возвращает #t, в случае, если объект является строкой
```

Существует два способа напрямую создать строку с помощью функций:

```
(make-string k)
(make-string k char)
```

Здесь, `k` – есть длина создаваемой строки, `char` – символ, которым должна быть наполнена строка (в первой функции она будет наполнена символами вида `"\u0000\u0000\u0000"`). Также строки можно создавать путем конкатенации (объединении) СИМВОЛОВ:

```
(string char ...)
```

Длина строки:

```
(string-length string)
```

Можно также получить доступ к каждому элементу строки (строковому символу):

```
(string-ref string k)
(string-ref "Привет" 1)  Результат #\р, так как нумерация в строке идет от нуля
```

Изменить конкретный символ в строке можно с помощью: `(string-set! string k char)`. При этом следует помнить, что `string-set!` возвращает неопределенное значение. Строки можно сравнивать (на порядок расположения в них строковых символов):

```
(string=? string1 string2)
(string-ci=? string1 string2)
(string<=? string1 string2)
(string>=? string1 string2)
(string-ci<=? string1 string2)
(string-ci>=? string1 string2)
(string-ci<=? string1 string2)
(string-ci>=? string1 string2)
```

PLT Scheme разрешает использовать более двух аргументов при сравнении строк:

```
(string=? "12" "15" "gsjfds")  Результат #f
```

При этом более короткие строки считаются меньше длинных:

```
(string-ci<? "fff" "ffff")  Результат #t
```

Выделение подстроки:

```
(substring string start end)
```

При этом входящие параметры должны удовлетворять условию:

```
0 < start < end < (string-length string)
```

Объединение (конкатенация) строк:

```
(string-append string ...)
(string-append "Привет " "Мир!")  Результат "Привет Мир!"
```

Также важны функции преобразования строк:

```
(string->list string)  преобразование строки в список строковых символов
(list->string list)    преобразование списка в строку (список)
```

```
(string->list string)  должен быть списком строковых символов)
(list->string list)     и
                        являются противоположными по отношению
(string-copy string)    друг к другу
                        возвращает копию строки
```

Векторы

Векторы в Scheme есть неоднородные структуры данных индексированные целыми числами. Одним из плюсов векторов является организация, построенная таким образом, что скорость доступа к ним выше, чем к спискам. Длина вектора есть число элементов, которые он содержит (целое точное неотрицательное число, также как и индекс любого из доступных элементов вектора). Индексация элементов вектора начинается от нуля. В общем, векторы сильно похожи на массивы императивных языков программирования, но имеют одну приятную особенность – вектор может содержать данные различных типов.

Для записи векторов используется следующая нотация `#(obj ...)`: `#(0 (2 2 2 2) "Anna")` – вектор, состоящий из трех элементов – числа ноль, списка `(2 2 2 2)` и строковой константы `"Anna"`.

`(vector? obj)` – определяет, является ли объект вектором (`#t`, если объект вектор).

`(make-vector k)` – создает вектор из `k` элементов, сами элементы не определены.

`(make-vector k fill)` – создает вектор из `k` элементов, элементы которого являются `fill`.

`(vector obj ...)` – создает вектор из указанных объектов:

```
(vector 'a 'b 'c)  Результат #(a b c), сами элементы при этом
                   не вычисляются (из-за одинарной кавычки)
```

`(vector-length vector)` – возвращает длину вектора (точное целое число).

`(vector-ref vector k)` – возвращает `k`-й элемент вектора (`k` как и все индексы должен быть точным и целым):

```
(vector-ref '#(1 1 2 3 5 8 13 21) 5)  Результат 8
                                       (нумерация от нуля)
```

`(vector-set! vector k obj)` – присваивает `k`-му

элементу вектора значение `obj`. Результат не определен, функция обладает побочными эффектами.

`(vector->list vector)` – создает список из вектора.

`(list->vector list)` – создает вектор из списка:

```
(vector->list '(dah dah didah))  Результат (dah dah didah)
(list->vector '(dididit dah))    Результат #(dididit dah)
```

`(vector-fill! vector fill)` – заполняет указанный вектор объектами `fill`. Результат функции не определен, функция имеет побочные эффекты.

Особенности управления

Здесь мы опишем некоторые (не все, например, отложенные вычисления уже рассматривались) механизмы, облегчающие управление процессом выполнения.

`(procedure? obj)` – возвращает `#t`, в случае если перед нами процедура. Надо отметить, что в Scheme очень часто идет слияние понятий функция и процедура:

```
(procedure? car)          Результат #t
(procedure? 'car)         Результат #f
(procedure? (lambda (x) (* x x)))  Результат #t
(procedure? '(lambda (x) (* x x)))  Результат #f
```

`(apply proc arg1 ... args)` – вызывает указанную функцию к аргументам заданным списком:

```
(apply + (list 3 4))      Результат 7, для функции
                           + (плюс) задается
                           параметр (только один),
                           заданный в виде списка
```

`(map proc list1 list2 ...)` – применение функции `proc` к каждому из списков. Списки должны иметь одинаковый размер. Одна из мощнейших функций языка. Работает это следующим образом:

```
(map car '((a b) (d e) (g h)))  Результат (a d g)
(map cdr '((a b) (d e) (g h)))  Результат ((b) (e) (h))
(map cadr '((a b) (d e) (g h)))  Результат (b e h)
```

Функция всегда работает со списками и списки же и возвращает. Эти примеры не полностью раскрывают работу функции, вот еще примеры:


```
(map + '(1 2 3) '(4 5 6))          Результат (5 7 9)
(map + '(1 2 3) '(4 5 6) '(7 8 9))  Результат (12 15 18)
(define my-list '(1 2 3 4 5))
(define (square val) (* val val))
(map square my-list)                Результат (1 4 9 16 25)
```

(for-each proc list1 list2 ...) - аналогична map, но используется в основном для функций с побочными эффектами, результат которых не определен:

```
(let ((v (make-vector 5)))
  (for-each (lambda (i)
              (vector-set! v i (* i i)))
            '(0 1 2 3 4))
  v)                                Результат #(0 1 4 9 16)
```

Создаем вектор *v* из пяти элементов. Затем применяем (lambda (i) к вектору *v*. Заносим данные из списка '(0 1 2 3 4) (который сразу не вычисляется из-за одинарной кавычки) следующим образом: в вектор *v* записывается элемент списка, умноженный сам на себя (запись осуществляется посредством vector-set!, при этом в качестве индексов для доступа к вектору используется все тот же список '(0 1 2 3 4)). После выхода из блока let вектор *v* будет разрушен, а его содержимое будет передан как результат работы блока let.

PLT Scheme

Теперь немного подробнее о самой IDE. *PLT Scheme* состоит из нескольких компонентов:

- MrEd - расширение MzScheme для графического программирования
- DrScheme - среда проектирования
- Mzc - компилятор в байт-код, не зависящий от платформы (позволяет создавать переносимые приложения)
- MzScheme3m - экспериментальная версия MzScheme, особенность которой более точное управление распределением памятью (в сравнении MzScheme).

MzScheme, основной компилятор-интерпретатор и

система поддержки исполнения программ. Кроме того, *PLT Scheme*, содержит в себе не только описанный выше стандарт **R5RS**, но также дополнительные возможности, а именно:

- система поддержки пространства имен и управления трансляцией
- поддержка механизма исключений
- приоритетные потоки
- классы и система объектов
- регулярные выражения
- расширенная поддержка макросов
- поддержка хэшей (как встроенного типа данных)
- поддержка юникода.

И более того, *PLT Scheme* содержит также группу диалектов *Scheme*, каждый из которых имеет свои задачи (например, есть диалект специально адаптированный для изучения функционального программирования студентами высших учебных заведений). Так как система построена в минималистичном и строгом стиле рекомендуется начинать изучение с *DrScheme* (он не так суров, и не сильно отпугивает пользователей привыкших к обилию кнопок, картинок и иконостасу на Рабочем столе). Его-то мы и рассмотрим подробнее...

Как уже упоминалось ранее, *DrScheme* имеет два окна, одно для ввода текста программы, второе есть командный интерпретатор (см. рисунок 1):

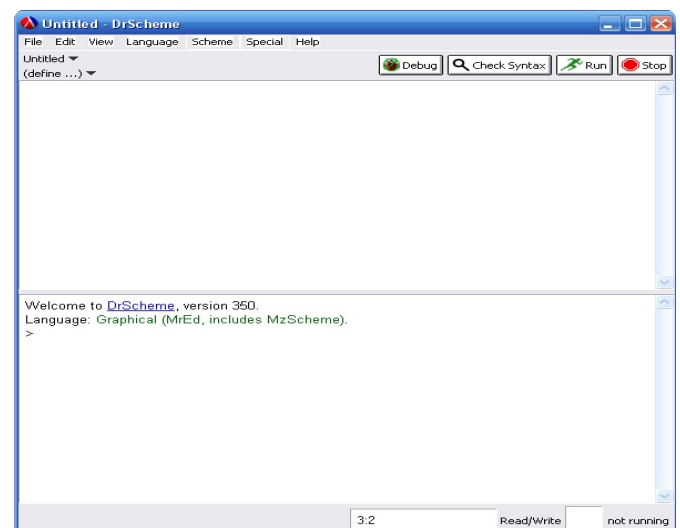


Рис. 1. Среда drScheme

Поскольку *PLT Scheme* поддерживает несколько языков программирования, то прежде чем приступить к работе, нужно выбрать* соответствующий язык программирования *Language|Choose Language* (см. рисунок 2):

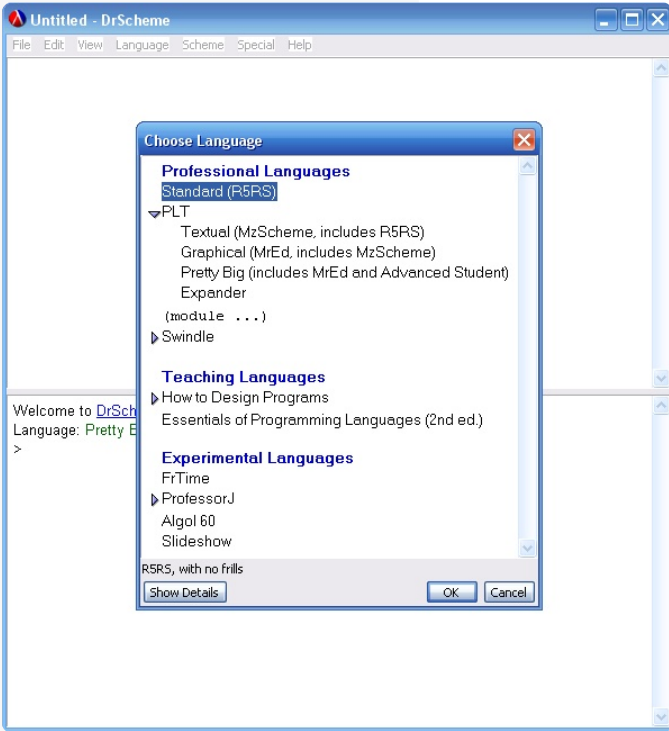


Рис. 2. Выбор языка

Стандарт языка называется *Standart (R5RS)*, для освоения материала статьи его вполне достаточно. Остальные диалекты отличаются различными свойствами, в том числе и поддержкой графического интерфейса пользователя. *PLT Scheme* запоминает последний выбранный диалект и при следующем запуске сразу же готов работать в выбранном контексте.

Изюминкой *PLT Scheme* является возможность компиляции текстов программ. Исторически и по ряду объективных причин большинство реализаций *Scheme* есть интерпретаторы. Однако готовые к употреблению программы без стороннего окружения также необходимы (что, кстати, требуют и решаемые задачи, круг которых гораздо шире в *PLT Scheme*, чем в голом стандарте *R5RS*). Чтобы получить из текста программы на

языке *Scheme* вполне функциональный **exe**-файл требуется, чтобы программа была введена в окно ввода программы (обычно оно первое) (см. рисунок 3). А также чтобы она была уже сохранена во внешнем файле.

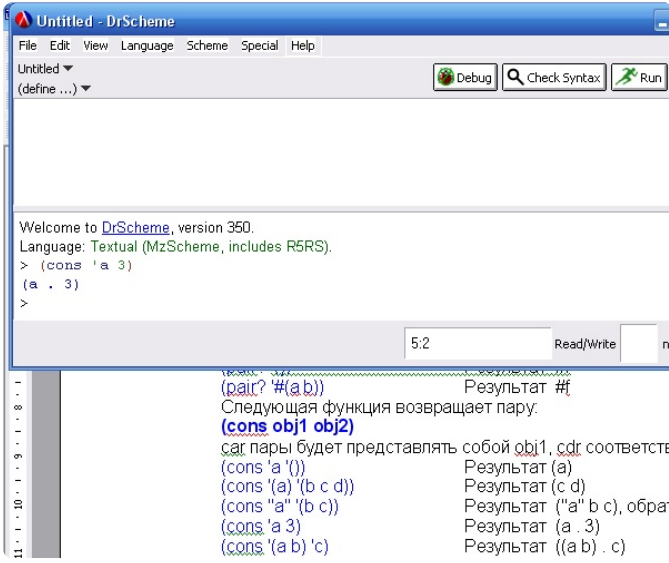


Рис. 3. Работа с примером

Затем выбираем пункт меню *Scheme|Create Executable...* Далее необходимо выбрать тип исполнения файла (см. рисунок 4):

- Launcher
- Stand-alone
- Distribution

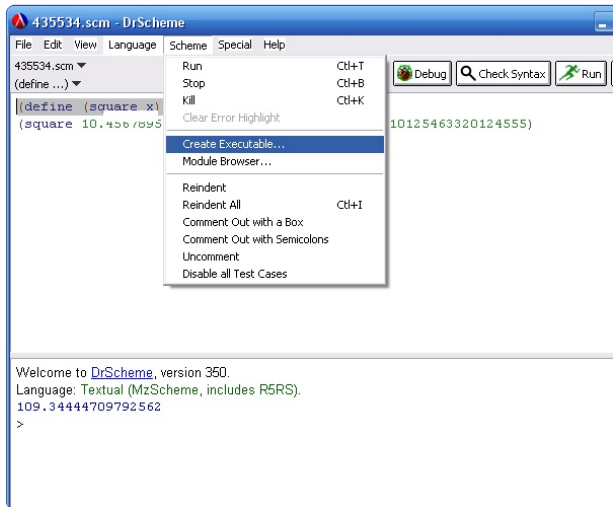


Рис. 4. Создание исполняемого файла

Для простых учебных примеров *Stand-alone* вполне достаточно (однако, если необходимо использовать программу на другом компьютере, то

*** Комментарий автора.**
...наименование пунктов меню может немного отличаться в зависимости от версии среды разработки

потребуется перетащить некоторые библиотеки).

Заключение

Введение дает только элементарные азы и рассчитано на плавный переход от императивного стиля к функциональному. Данное введение не дает представления о функциональном программировании – это просто справочник наиболее распространенных команд языка Scheme, также знакомство с конкретной средой разработки *PLT Scheme*. Более полное и подробное руководство по *PLT Scheme* можно найти на сайте проекта (на английском языке), статья предназначена быть отправной точкой для дальнейшего самостоятельного обучения.

Дополнительно можно отметить возможность встраивания *Scheme* и в другие языки программирования (как встроить *Scheme* в Дельфи, можно узнать здесь: <http://www.orlovsergei.com/Progs/Scheme/SchemeToDelphi.htm>). Плюсы такого союза очевидны, например, это удобно для работы с длинной арифметикой.

Ресурсы

- Лисп как альтернатива Java <http://alexey.tamb.ru/scheme/lisp-scheme-java.html>
- Сайт «схемщиков» <http://schemers.org>
- PLT Scheme <http://www.plt-scheme.org>
- Дополнительные библиотеки для PLT Scheme <http://planet.plt-scheme.org>
- Неформальное введение в Scheme http://ru.wikibooks.org/wiki/Введение_в_язык_Scheme_для_школьников

Ежедневно мы сталкиваемся с рутинной работой, которая отнимает львиную долю нашего времени. В этой статье я попробую «приучить» читателя к созданию маленьких помощников, оптимизирующих работу или сокращающих время рутинных операций...



Алексей Шишкин

by **Alex Cones** <http://flsoft.ru>

В фантастических фильмах мы часто видим, что человека окружают маленькие роботы, которые помогают ему, выполняют его рутинную работу. Роботы пылесосы убирают пыль и мусор, маленькие роботы кофеварки подадут Вам свежий кофе, а маленький робосекретарь напомнит Вам о важной встрече. В жизни все не так просто.

Но, не смотря на такую жестокую реальность, программисты главным образом живут в мире виртуальном. Поэтому ничто не мешает им улучшать свою жизнь, создавая «роботов»-помощников. «Но какие-же помощники могут быть у программиста?» - скажете Вы. Я постараюсь ответить на Ваш вопрос, опираясь на собственный опыт.

История появления...

Итак, первая вещь, которая была создана мной для облегчения собственной жизни – это «Заполнялкин» (см. рисунок 1). Эта программа предназначалась для того, чтобы оптимизировать написание больших блоков кода, отличающихся только ссылками. Так, например, введя шаблон:

```
Label@.Caption := IntToStr(@);
```

Можно было получить практически неограниченное количество следующих строк:

```
Label1 := IntToStr(1);
Label2 := IntToStr(2);
Label3 := IntToStr(3);
Label4 := IntToStr(4);
```



```
Label5 := IntToStr(5);
```

Кстати говоря, данные строки были получены с помощью вышеописанной программы. Итак, вопрос создания многократной записи большого количества похожего кода уже не стоял, и я занялся другими проблемами.

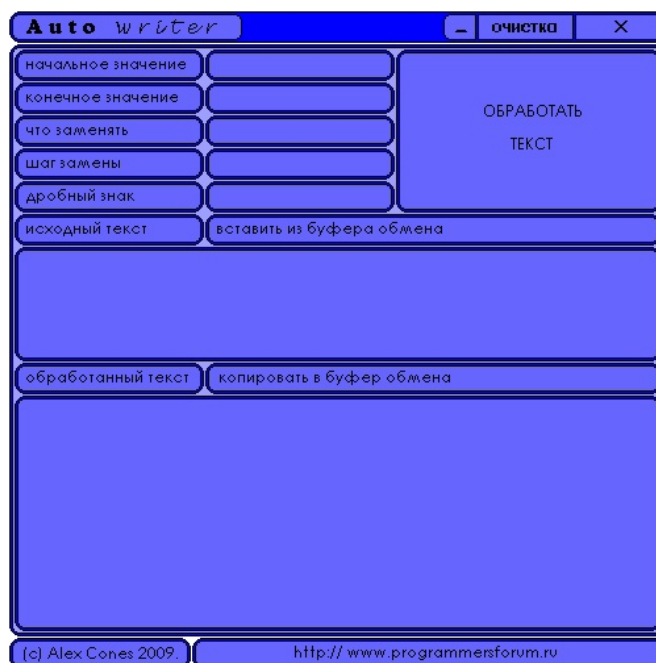


Рис. 1. Утилита «Заполнялкин»

Второй программой стал **Resource Builder** (см. рисунок 2). Да, возможно некоторые станут упрекать меня за то, что такое название уже существует, но я ведь не собираюсь продавать это

творение, поэтому не обеспокоен нарушением авторских прав на название программы. Моя версия* создателя ресурсов к программам отличалась тем, что в ней можно было добавить любой файл в ресурсы к программе.

буфере обмена (см. рисунок 3). А однажды мне потребовалось залить на файлообменный сервис достаточно большой файл. С моей полу-диалогной скоростью эта задача имеет решение только посредством FTP доступа. К счастью сервис

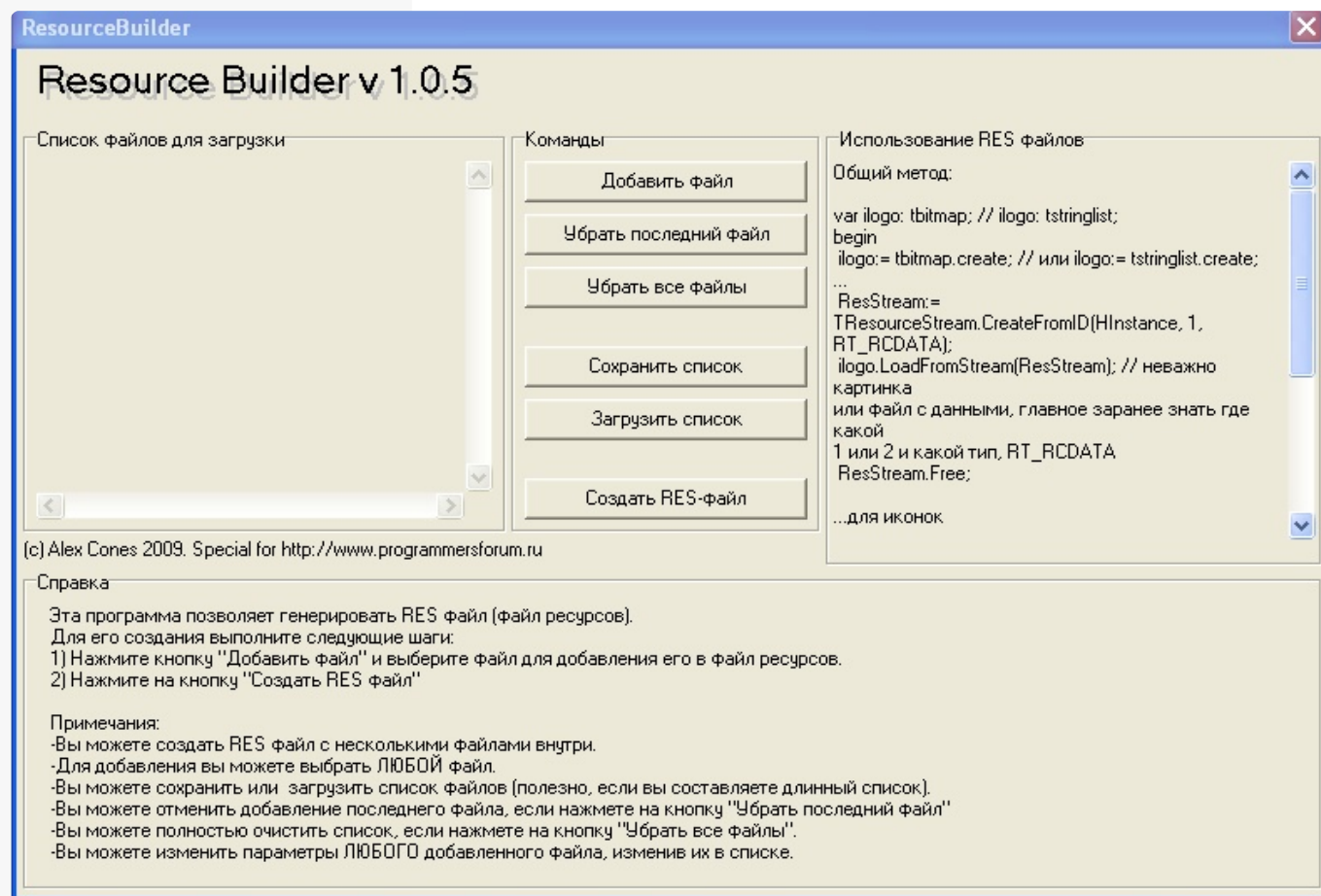


Рис. 2. Утилита Resource Builder

Итак, вопрос удобства создания программ уже не стоял, я приступил к улучшению окружающей меня обстановки, созданию **G.A.P.** Вдохновили меня действия **Educated Fool**: он создал excel-ский макрос, который упаковывал проект в архив, создавал к нему превью и отправлял на FTP сервер. По аналогии моя программа делает снимок экрана (или части его — по выбору пользователя), дает возможность создать превью к снимку, отметить на нем что-то и отправить на сервис хранения картинок, оставив ссылку на картинку в

предоставляет такую услугу. Радости моей не было предела и на первую же ночь я поставил на

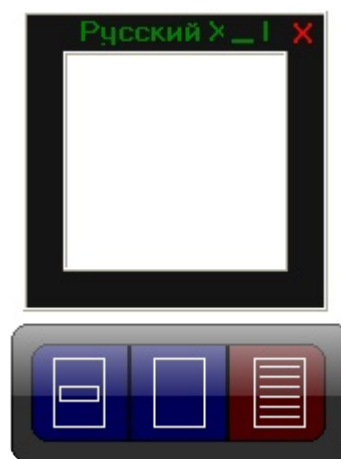


Рис. 3. Утилита G.A.P

загрузку злополучный файл. Проснувшись утром и просмотрев логики, я ужаснулся — сервер отключает меня каждые 15 минут бездействия. Даже если в этот момент загружается файл. Выход был лишь один — отправлять команду просмотра каталогов каждые 10

минут (благо для этого была выделена отдельная кнопка). Но не кликать-же по ней каждые десять минут, пока файл не загрузится? Хотя-я... Собственно, почему нет? За 15 минут был создан

* Комментарий автора.

На этом месте я хочу предупредить читателя о том, что данная статья задумывалась вовсе не как реклама этих программ, а как пособие начинающему «импруверу» (от англ. improve — улучшать). Не бойтесь экспериментировать, и запомните одну вещь — ЛЮБАЯ работа может быть оптимизирована. Даже если кажется, что это не так.

Click Shot (программа, которая будет кликать за меня в нужную точку экрана через заданный промежуток времени):

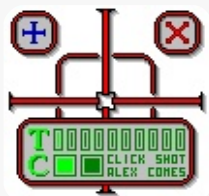


Рис. 4. Утилита Click Shot

Думаю, лишним будет говорить то, что файл был успешно загружен.

Вчера один из моих товарищей вставил в мой ноутбук свою флешку. Несмотря на то, что на ней были только документы, Windows спросила разрешения запускать с неё программы. Снизойдя до отказа, я включил отображения скрытых и системных файлов и обнаружил autorun и ехе-шник. Открыв авторан, я понял, почему антивирус продолжал молчать:

```
[AutoRun  
;lsbvrksjvbliurbsv  
;srvlbsrvksrjksr  
open = klbhk.exe  
;kjbsjvbkvksjvn
```

Одна закрывающаяся скобка... И план вторжения армий провалился... Но что-то я отвлекся. Удаление файлов прокатывать не захотело по причине атрибута «системный» у обоих файлов. Форматировать флешку мне не позволили, и я накатал программу, изменяющую атрибуты каталогов и файлов по выбору пользователя. Так появился на свет **A.ch** (см. рисунок 5):

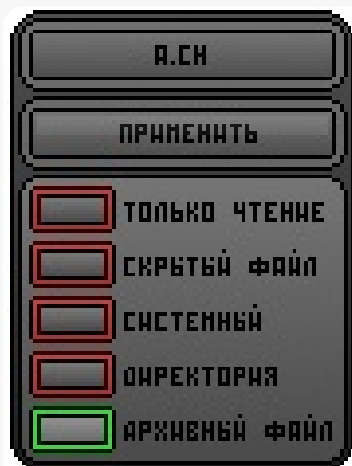


Рис. 5. Утилита A.ch

Заключение

В завершение статьи хочу отметить, что каждая решенная проблема приносит удовольствие, но лично для меня большее удовольствие приносит решение проблемы. Дерзайте, и да придут к вам маленькие помощники программиста!

Ссылки

- Обсуждение утилиты Заполнялкин 1.0
<http://www.programmersforum.ru/showpost.php?p=367784&postcount=26>
- Обсуждение утилиты Resource Builder
<http://www.programmersforum.ru/showthread.php?t=69505>
- Обсуждение утилиты G.A.P
<http://www.programmersforum.ru/showthread.php?t=69505>
- Обсуждение утилиты Click Shot
<http://www.programmersforum.ru/showthread.php?t=92768>
- Обсуждение утилиты A.ch - Attribute Changer
<http://www.programmersforum.ru/showthread.php?t=104574>

В статье написано то, что автор смог прочесть, понять и пересказать своими словами и немного того, до чего додумался сам; все то, что касается процесса взаимодействия двух человек: разработчика и пользователя. Кратко это выражается одним словом – интерфейс...



Александр Демьяненко

by Grenles GRENLES@yandex.ru

Философия

Надо же! Почти вступление! Не скажу, что я большой профессионал, особенно по теме данной статьи, но и не скажу, что я совсем уж любитель. Я учусь писать, читать, видеть, думать, и снова начинаю этот процесс сначала*. Очень сложно учить чему-то профессионалов, особенно в той области, которую они знают лучше тебя, гораздо проще – стать их учеником. Однако, как ни странно, порой именно любители пишут книги для профессионалов и учат их ремеслу и только лишь потому, что не всегда профессионалы умеют выразить доходчиво свои знания для других. Поэтому не все так плохо, но и не так просто, как кажется на первый взгляд. В мире всегда полно тех, кто задает вопрос: «А с чего мне лучше начать?». Мало того, есть еще такие профессионалы, которые проснувшись утром, понимают, что они ничего не знают из того, что должны знать. Именно поэтому каждый день они начинают с поиска и изучения новых горизонтов знания. Каждый раз, узнав больше, они понимают, что все равно ничего не знают и снова начинают бесконечный процесс приобретения знаний. Вот для этих людей и не только для них я и начал разговор на заданную тему.

Тема, которую хочу затронуть, выражается одним словом – интерфейс. Тема с одной стороны кажется простой и легкой, а с другой – это весьма сложная и обширная область знаний, которую трудно уместить в рамки одной статьи. Анализируя современный рынок программного

обеспечения, множество интернет сайтов, различных печатно-книжных изделий я пришел к выводу, что знания о правилах создания интерфейса должны быть у всех, так или иначе связанных с созданием продуктов, предназначенных для пользователей, потребителей. На сегодняшний день эта область знания вполне достойна того, чтобы идти отдельным курсом в высших учебных заведениях.

От того, насколько удачно продуман и реализован интерфейс продукта, зависит 75% успеха разработчика и 100% эффективности использования потребителем конечного продукта. Случается, что и начинающие разработчики, и гранды компьютерной индустрии, как студенты на экзаменах, «засыпаются» на плохо сделанном интерфейсе для своих великих проектов. К сожалению, не всегда с первого раза удается найти удобный способ взаимодействия пользователя и продукта. То, что ясно разработчику, не всегда очевидно пользователю и наоборот. Почти как мужчина и женщина всегда говорят на разных языках, так разработчик и пользователь живут и мыслят на основе разных категорий.

Сядя за статью и желая раскрыть тему полнее и правильнее, я решил обратиться в Интернет, чтобы узнать: «А как на мой вопрос отвечали другие?». Не скажу, что эта фраза мне попалась первой, но зацепился я за нее сразу: «Для ReGet дизайн придумывала студия Артемия Лебедева. В результате пользоваться ReGet, в отличие от FlashGet очень удобно. Можете в этом сами убедиться» [1]. При этом, я не думаю, что функционально обе программы сильно разнятся, выполняемые ими задачи практически одни и те же, но разница в том, что одну программу мы используем, а другую просто имеем ввиду, зная что она есть.

* Комментарий автора.

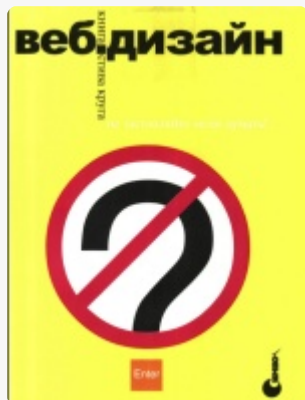
Кто-то сказал: «хочешь что-то понять, - объясни это что-то другому, - он может и не понять сказанного тобою, но уж ты сам точно это поймешь».

Самый веский аргумент, окончательно подтолкнувший меня на создании серии статей по заданной теме, я нашел в источнике [1]: «в России слишком много программистов-самоучек, которые все этапы создания программы от идеи программы до ее реализации выполняют сами». К сожалению, не у всех есть возможность нанять профессиональных дизайнеров или найти специалиста в этой области, поэтому приходится «изобретать велосипед» самому.

Начнем. С чего? Элементарно! What song? Книги!

Я нашел очень много информации по теме и около нее: статьи, примеры, книги, журналы. Среди множества источников выделил три книги. Не буду утверждать, что они являют собой ту самую истину, которая незыблема и нет других источников информации, но прочесть бы их я посоветовал:

1. Стив Круг. Веб-дизайн или не заставляйте меня думать



ее стоит прочесть хотя бы для того, чтобы понять основные принципы «юзабилити» и того, как не стоит делать интерфейс. Кто-то скажет, - но она же только для тех, кто создает веб-сайты,

Отвечу, - а кто сказал, что создание интерфейса программы сильно отличается от создания интерфейса веб-сайтов? Я бы сказал иначе, создание веб-сайтов выросло из принципов создания программ, именно поэтому

внешний вид может быть разным, но подходы, принципы, правила одни и те же. Но, как ни странно, думать надо в любом случае и при создании сайта, и при создании программы. Хорошо думать, когда есть знания. Замечательно думать, когда знания перешли в умения. Надеюсь, уважаемый читатель, ты меня понял.

2. Джеф Раскин. Интерфейс: новые направления в проектировании компьютерных систем

Джеф Раскин
Интерфейс: новые направления в проектировании

- Предисловие
- Введение
- Глава 1. Принципы проектирования
- Глава 2. Принципы проектирования
- Глава 3. Принципы проектирования
- Глава 4. Принципы проектирования
- Глава 5. Принципы проектирования
- Глава 6. Принципы проектирования
- Глава 7. Принципы проектирования
- Глава 8. Принципы проектирования
- Глава 9. Принципы проектирования
- Глава 10. Принципы проектирования
- Глава 11. Принципы проектирования
- Глава 12. Принципы проектирования
- Глава 13. Принципы проектирования
- Глава 14. Принципы проектирования
- Глава 15. Принципы проектирования
- Глава 16. Принципы проектирования
- Глава 17. Принципы проектирования
- Глава 18. Принципы проектирования
- Глава 19. Принципы проектирования
- Глава 20. Принципы проектирования
- Глава 21. Принципы проектирования
- Глава 22. Принципы проектирования
- Глава 23. Принципы проектирования
- Глава 24. Принципы проектирования
- Глава 25. Принципы проектирования
- Глава 26. Принципы проектирования
- Глава 27. Принципы проектирования
- Глава 28. Принципы проектирования
- Глава 29. Принципы проектирования
- Глава 30. Принципы проектирования
- Глава 31. Принципы проектирования
- Глава 32. Принципы проектирования
- Глава 33. Принципы проектирования
- Глава 34. Принципы проектирования
- Глава 35. Принципы проектирования
- Глава 36. Принципы проектирования
- Глава 37. Принципы проектирования
- Глава 38. Принципы проектирования
- Глава 39. Принципы проектирования
- Глава 40. Принципы проектирования
- Глава 41. Принципы проектирования
- Глава 42. Принципы проектирования
- Глава 43. Принципы проектирования
- Глава 44. Принципы проектирования
- Глава 45. Принципы проектирования
- Глава 46. Принципы проектирования
- Глава 47. Принципы проектирования
- Глава 48. Принципы проектирования
- Глава 49. Принципы проектирования
- Глава 50. Принципы проектирования
- Глава 51. Принципы проектирования
- Глава 52. Принципы проектирования
- Глава 53. Принципы проектирования
- Глава 54. Принципы проектирования
- Глава 55. Принципы проектирования
- Глава 56. Принципы проектирования
- Глава 57. Принципы проектирования
- Глава 58. Принципы проектирования
- Глава 59. Принципы проектирования
- Глава 60. Принципы проектирования
- Глава 61. Принципы проектирования
- Глава 62. Принципы проектирования
- Глава 63. Принципы проектирования
- Глава 64. Принципы проектирования
- Глава 65. Принципы проектирования
- Глава 66. Принципы проектирования
- Глава 67. Принципы проектирования
- Глава 68. Принципы проектирования
- Глава 69. Принципы проектирования
- Глава 70. Принципы проектирования
- Глава 71. Принципы проектирования
- Глава 72. Принципы проектирования
- Глава 73. Принципы проектирования
- Глава 74. Принципы проектирования
- Глава 75. Принципы проектирования
- Глава 76. Принципы проектирования
- Глава 77. Принципы проектирования
- Глава 78. Принципы проектирования
- Глава 79. Принципы проектирования
- Глава 80. Принципы проектирования
- Глава 81. Принципы проектирования
- Глава 82. Принципы проектирования
- Глава 83. Принципы проектирования
- Глава 84. Принципы проектирования
- Глава 85. Принципы проектирования
- Глава 86. Принципы проектирования
- Глава 87. Принципы проектирования
- Глава 88. Принципы проектирования
- Глава 89. Принципы проектирования
- Глава 90. Принципы проектирования
- Глава 91. Принципы проектирования
- Глава 92. Принципы проектирования
- Глава 93. Принципы проектирования
- Глава 94. Принципы проектирования
- Глава 95. Принципы проектирования
- Глава 96. Принципы проектирования
- Глава 97. Принципы проектирования
- Глава 98. Принципы проектирования
- Глава 99. Принципы проектирования
- Глава 100. Принципы проектирования

считаю весьма полезными для разработчиков, желающих стать профессионалами. Я бы не сказал, что по году создания и издания книга нова. В найденной

мною версии она датируется 1996 годом, но только что это меняет? В книге весьма интересно разложены по полочкам элементы интерфейса их плюсы и минусы, и порой высказывается совершенно иной взгляд на привычные, казалось бы, вещи: отказ от GUI, отказ от использования мыши, отказ от разбиения задач по приложениям. В определенном смысле программы интернет-браузеры реализуют эту идею, когда в одном окне выполняется все – звук, графика, редактирование, просмотр видео и прочие действия.

3. Иоханнес Иттен. Искусство цвета



книга тоже не новая, но привожу я ее из тех соображений, что дизайнеру, просто глупо ничего не знать о свете и цвете и не уметь применять эти знания на практике. Да и просто

это повод открыть интернет и поискать любую литературу на тему рисования, цвета, способов изображать предметы. Скажу вам – очень интересная тема. Исследуйте на досуге, а я как-нибудь позднее чего-нибудь вам об этом расскажу. Искусство художника оказывается полезным для дизайнера – это основа его творчества.

4. В.В. Головач. Дизайн пользовательского интерфейса

Скажу сразу, она попала мне на глаза гораздо позже других книг и понравилась намного больше

тем, что в ней нет строгой теории, а сразу идет разбор конкретных ситуаций – ошибок и решений. Как ни странно, на фоне прочтения первых трех книг, эта – четвертая книга как будто подвела итоги решений, описанных теоретически в

«умных» книгах. Мало того, изложение материала таково, что я бросил чтение других книг и продолжил чтение только этой книги. При этом, на основании изложенных примеров, я реализовал несколько удобных «фишек» для своих программ на будущее, и они мне понравились. Советую прочесть эту книгу, просто потому, что это полезно и интересно. Есть и более новая версия этой книги – «Дизайн пользовательского интерфейса 2. Искусство мыть слона».

Начав читать, и, почти дочитав эти книги, я понял, что дизайн это бесконечный процесс слияния знаний, фантазии и ресурсов, которыми обладает дизайнер, в котором многое зависит от личности и ее таланта. Задумавшись, решил, что точнее назвать этого человека дизайнера – создатель: во-первых – это по-русски, а во-вторых – более отражает процесс дизайна (создавать нечто удобное, новое и оригинальное).

Что наша жизнь? И...

Классический ответ на заданный вопрос – «игра». В каком-то смысле, если задуматься, так и есть – вся наша жизнь есть одна сплошная игра. И в любой игре есть маски, костюмы, актеры, зрители. Я нашел этой игре новое обличье:

. зрители – это пользователи, потребители созданного вами продукта – программы, фильма, музыки, книги и тому подобного

. маски и костюмы – это тот самый интерфейс, в который мы одеваем наш продукт и то, что видит в первую очередь зритель. Это только потом он начинает разбираться в сюжете (логике), в

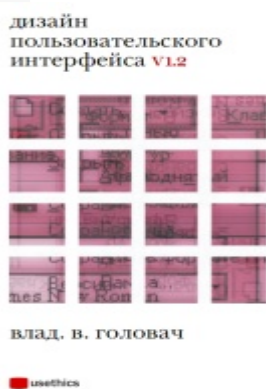
таланте актера (профессионализме разработчика), имя которого завтра будет трудно вспомнить, если он оказался плох и т.д.

. актеры – это разработчики, по сути – мы с вами, те, кто хоть однажды пытался что-то проектировать, создавать, строить и прочее. От того, насколько мы талантливые актеры будет зависеть успех всего спектакля, аншлаг, признание публики, слава, деньги и прочее.

Разрабатывая программу, мы играем роль самого Создателя. Может быть, именно поэтому большинству так нравится играть, многим – нравится программировать, а некоторым из числа избранных – создавать и творить. Все это теоретически хорошо, но возникает вопрос, а какой ты создатель? И для кого ты создаешь? Мучаешься, не спишь ночами, читаешь гору разной литературы, бьешься над парой строк кода и все ради чего? Чтобы однажды узнать – твой труд никому не нужен, потому что ты не смог найти удачный способ взаимодействия пользователя с твоим творением. Не смог донести до него свою мысль и идею – ответ на главный вопрос: «...зачем ему нужно то, что ты создал?». Тебе-бы встретиться с ним, поговорить по душам за рюмкой чаю, но уже поздно – время упущено и пользователь ушел к другому, так и не поняв, что же ты хотел для него сделать. К сожалению, в жизни часто так бывает, что разработчик и пользователь встречаются друг с другом тет-а-тет весьма и весьма не часто. Особенно последняя фраза касается начинающих разработчиков – опытные-то знают, что пользователя надо завлечь пряничком, блестящей оберткой, усадить на мягкий стульчик, подать вкусный чай... Часто начинающие разработчики не то чтобы забывают о пользователе, они порой плохо представляют, а кто это вообще такой.

Пройдемся по науке

Собственно, что я хотел сказать? Для начала надо немного определиться и ответить на вопрос главный вопрос статьи: «Что такое интерфейс*?». Ответ обобщенный, подходит для разных сторон и



сфер человеческой деятельности, но в своей сути он отражает смысл этого явления. Далее, подумав немного, решил, что мир придуман не вчера и все процессы должны быть так или иначе давно регламентированы и описаны. Я начал искать стандарты, правила, ГОСТ-ы. Приводить здесь подробное описание этих документов абсолютно бессмысленно. Каждый сам может найти их в сети Интернет. Будет гораздо лучше, если выскажу свои соображения, которые сделал, переработав найденную информацию...

Большинство ГОСТ-ов, найденных мною, направлены на разработку технической документации, связанной с процессом разработки программного обеспечения. Создается впечатление, что изначально в 70-х начале 80-х годов прошлого столетия упор делался на техническую сторону этого вопроса – разработку и детализацию алгоритмов, средств сопряжения, подбор технической базы, создание различных инструкций по эксплуатации и прочее. Старый ГОСТ именно это и регламентировал. Впрочем, если подумать, так оно и должно было быть, так как программирование по сути родилось из математики и на первом месте было решение задач, а не внешнее оформление. Лишь в ГОСТ, относящимся к недавним советским временам, а именно, ГОСТ 19.201-78 и ГОСТ 24.207-80 можно неявно увидеть фразы, частично указывающие на средства взаимодействия с пользователем: «требования к программе или программному изделию», «требования к маркировке и упаковке». Правда, прочтя эти фразы, явно и не скажешь, что тут имеется ввиду «интерфейс». Тем не менее, в источнике [2] нашел следующее: «Техническое задание, как правило, разрабатывается на основе ГОСТа 19.201-78 «ЕСПД. Техническое задание. Требования к содержанию и оформлению». Таким образом, получается, что разработчик должен сам знать и подразумевать, что в задание должны закладываться вопросы, связанные с

интерфейсом. Явно это в ГОСТ не звучит. Но это было «тогда». По ГОСТ советских времен подразумевалось, что интерфейс у нас возникает сам собой, - это следует из текстов документов. Получалось, как в известной фразе: «В СССР секса нет, а дети есть». Всем известна фраза о том, что в мире все подвержено изменениям. Тоже самое можно утверждать и про ГОСТ-ы. Не прошло и 20-ти лет, изменились требования времени, реалии, изменилось само время. Согласно документам, датированным 2000-м годом и выше, разработчики ГОСТ уже узнали про интерфейс и даже явно написали об этом в «Пример шаблона технического задания (ТЗ) на сайт» [3], где есть два раздела: «Требования к графическому дизайну сайта», и «Требования к дизайну сайта». Это уже радует. Конечно же, кажется совершенно очевидным, что сайт без дизайна не сайт. Однако, судя по всему, на осознание этого факта разработчикам ГОСТ потребовалось время. Думаю, что оно им еще будет нужно, чтобы создать отдельный ГОСТ, касающийся только дизайна, как такового.

В итоге, проглядев различные ГОСТ, документы, шаблоны и примеры, сделал основной вывод. Сейчас, как и раньше, основной упор идет на правильное, точное, детальное оформление документации по всем этапам разработки продукта – замысел, поиск информации для начала разработки, процесс разработки, тестирование, внедрение, эксплуатация. С одной стороны – это правильно, сложные продукты делает много людей и для удобства их работы и взаимодействия нужны документы, описывающие детально и правильно различные этапы процесса создания конечного продукта. А с другой стороны – это жуткая бюрократическая формальность, требующая от каждого участника процесса описывать каждое свое действие и решение. Именно поэтому создание документации так не любят разработчики. Порой время, требуемое на ее создание соизмеримо со временем создания самого программного продукта. Но, опять же, мы вернулись к тому, от чего пришли – в настоящее время по ГОСТ дизайн и интерфейс упирается в

**** Комментарий автора.**

Интерфейс (от англ. interface — поверхность раздела, перегородка) – совокупность средств и методов взаимодействия между элементами системы [4].

«бумагу».

Если внимательно подумать, то это правильно, заказчику «нечто» не покажешь, а нарисованное и написанное на бумаге вполне возможно. Поэтому, насколько мне представляется, знание ГОСТ и этапов от создания технического задания до сопровождения готового продукта необходимо не только крупным фирмам, но и одиночным разработчикам, так как это позволяет делать все правильно и приучает к определенному порядку. Конечно же, для написания программы, из разряда «калькулятор» вряд-ли нужно техническое задание, но знание этого необходимо. Прошлись по науке? Хотя ГОСТ и прочие нормативные документы необходимо знать, но, по сути, это скучно, сухо и сложно. Пожалуй, хватит – идем дальше.

Что вы делаете после того, как решили написать собственную программу?

Предполагаю, что 95% респондентов ответят – обдумываю идею программы, способы реализации, типы данных, выбор среды программирования и прочие детали и ... будут почти правы. А почему – почти? А потому «почти», что мало-кто сразу из большинства разработчиков начинает думать над тем: а как будет выглядеть ваше будущее творение перед лицом пользователя. Удобно-ли пользователю будет работать с программой***. Согласен, что на первом этапе весьма сложно представить цельный интерфейс программы, так как и программы, собственно говоря, еще и нет, но задуматься над этим стоит уже с самого начала.

А как вы пишете программу?

Думаю, что большинство ответят примерно так: «Сажусь за компьютер, открываю среду программирования и начинаю писать код». Если

это визуальная среда программирования, то обычно 100% действий начинается с того, что на пустую форму перетаскивается мышкой какой-либо элемент из палитры компонентов и... вот тут начинается самое интересное, вы ступаете на дорожку «войны». Я не ошибся, именно дорожку войны, даже точнее будет сказать – тропу. Почему? Все очень просто – весь процесс разработки есть борьба с самим собой, с кодом, ошибками и прочими неприятностями. В любой программе во время ее написания никогда с первого раза не бывает правильно размещенного элемента в нужном месте формы, соседние элементы иногда начинают мешать друг другу или просто не помещаются на форме так, как этого хочется. В итоге вы все время воюете – с алгоритмом, со средой разработки с внешним видом. Причем чаще всего страдает от всей этой «битвы» интерфейс. Да и кто серьезно задумывается**** над тем, насколько удобно пользователю будет работать с программой, когда на первое место выходят задачи логики функционирования программного продукта?

Вывод, напрашивающийся из всей этой ситуации таков – в условиях нехватки времени (а в обычных «рабочих» условиях так и есть) одному человеку практически невозможно создать удобную во всех отношениях программу. Все усилия в этом случае направлены на то, чтобы найти и реализовать логическое решение, внедрить и запустить в работу программу. Понимание того, как все должно быть, как удобнее работать, приходит уже после того, когда программа запущена в работу и эксплуатируется. И, часто получается так,

что реализация «удобства интерфейса» уходит на второй план, так как возникают более важные задачи, - исправление допущенных ошибок в логике, модернизация и прочее. Именно поэтому большинство серьезных фирм очень много времени тратят на подготовку процесса, создание

*** Комментарий автора.

Скажу, что два года назад, когда я стал активно заниматься программированием, а не просто сопровождением программ и написанием мелких макросов, я больше корпел над алгоритмами и достижением результата, - программа должна делать то, что хочет от нее пользователь. Интерфейс возникал или по ходу написания, или так, как было удобно мне, а не пользователю. После запуска продукта в эксплуатацию как минимум неделя была посвящена только тому, что я отвечал на вопросы что, где, почему и зачем. Теперь же я стараюсь думать не только над «внутренностями», но и над «внешним видом».

«стартовой базы», и лишь потом начинают программировать. В экстремальных условиях интерфейс страдает всегда.

Когда вы задумываетесь о том, кто и как будет работать с вашей программой и какие ситуации могут при этом возникнуть?

А вот на эти вопросы я не могу дать однозначного для всех ответа. У каждого найдется свой ответ, но, в конечном итоге, правильный ответ на эти вопросы всегда звучат из уст пользователя. Почему из уст пользователя? Отвечу цитатой рецензента этой статьи, литературного редактора этого журнала, **Utkin**: «...даже в идеальной программе пользователи найдут что покритиковать. Просто потому что пользователи всегда придирчивы. Одним подавай одно, другим требуется прямо противоположное».

Поспорить с этим сложно. Еще классик И.А. Крылов написал басню про Слона, который рисовал картину в угоду всем и в итоге никому его шедевр не понравился. В данном случае – тот же самый процесс.

Мало того, этот самый «пользователь» всегда найдет ту самую ошибку, которую никто из разработчиков даже в страшном сне не мог увидеть. При этом, сложно сказать, какая ошибка хуже – алгоритмическая или визуальная, обе одинаково неприятны. Разница лишь в том, что логическую ошибку среднестатистический пользователь не всегда можно найти, особенно, если плохо знает логику функционирования

программы, – такую ошибку не всегда видно. С точки зрения пользователя, логические ошибки могут быть вообще никогда не обнаружены. По известной статистике большинство пользователей используют лишь 5-10% возможностей программы, про остальные 90-95% возможностей, где и может оказаться логическая ошибка, они могут не знать. Визуальная ошибка всегда хуже – она просто заметнее. Классический пример из недавнего прошлого – это ошибка вывода изображения видеокарты. Чаще всего такие ошибки

выявлялись в играх, как более требовательным к ресурсам компьютера, а не «офисных» приложениях, которым хватало

«гарантированного минимума». Они проявлялись в том, что могли «ломаться» контуры изображаемых предметов, возникали непредвиденные визуальные эффекты, искажения картинки в целом, в худшем варианте или просто ничего не было видно, или компьютер шел на перезагрузку. При этом, источник ошибки мог быть с любым месте – как в сбое видеокарты, так в используемом программном

обеспечении (драйверах видеокарты, драйверах DirectX или OpenGL, несовместимости с операционной системой и прочее).

Для крупных корпораций уровня Microsoft или Adobe, когда есть рынок, бренд, сформированный круг пользователей, которые все равно не уйдут, такие ошибки менее ощутимы. Да, они неприятны, никто не говорит, что ошибки – это хорошо, но все-

****** Комментарий автора.**

Пример из моей практики. В связи с производственной необходимостью потребовалось решить проблему автоматизации учета. Проблема была в том, что «низы» уже не могли вести расчеты «вручную» в связи с их сложностью, объемом и цикличностью, а «верхи» не знали быстрого решения проблемы, то есть подходящего программного решения по цене и решаемым задачам не было. В итоге, по стечению различных обстоятельств, наш системный администратор взялся за разработку программы «с нуля». То, что для него это было тяжело, это мягко сказано, это было очень тяжело. Ему приходилось решать проблемы абсолютно разного рода – выполнять свою непосредственную работу, параллельно изучать новые методы и технологии программирования, изучать предметную область задачи, решать дела домашние, между этим делать еще что-то. Мало того, периодически он «воевал» с непосредственным начальством по поводу совмещения прямых рабочих обязанностей и работы по написанию программы (программу он писал добровольно и, в том числе, на своем рабочем месте, а какому начальству понравится такое?). Смысл всей этой предыстории таков, что ему было не до интерфейса и дизайна программы. Для него было важным решить технические проблемы стоявшей перед ним задачи. В итоге задачу он решил для своего уровня и того времени достойно. Однако, в процессе эксплуатации выяснилось, что где-то он не додумал логику, где-то проиграл в удобстве интерфейса. При этом каждый день он решал какие-либо проблемы, связанные с внедренной программой – дописывал, исправлял, модернизировал, консультировал пользователей, писал документацию. Добавление новых решений и отчетов в программу, привело к тому, что с каждым днем она становилась все функциональнее и сложнее. В итоге, несмотря на то, что он видел и знал огрехи интерфейса, переделать или исправить что-то было уже практически невозможно, так как это означало переписывание программного кода практически сначала.

таки, менее ощутимы. Для таких корпораций они всего-лишь повод создания обновлений и выпуска новых версий программ. При этом, вовсе на факт, что старые версии работали хуже или не удовлетворяли потребностям пользователя. На мой взгляд, рассматривая, как пример, программу Adobe Photoshop, для большинства пользователей (не дизайнеров) вообще достаточно ее варианта в версии 9.0. Новые возможности программы, выпуск линейки продуктов от Adobe в виде нескольких DVD дисков, еще раз повторю, обычным пользователям, вообще не понятны и не нужны, - они просто не пользуются ими в полной мере. Хотя, как бренд и признак «статуса», большинство просто устанавливают эти продукты на свой компьютер, не зная и 5% всех возможностей. Грубо говоря, они микроскопом забивают гвозди, изменяя размеры фотографий или сохраняя их в другой формат.

Возвращаясь к теме разговора, еще раз скажу, что ошибки для крупных корпораций менее болезненны, чем для разработчиков-одиночек. И вот почему. Образно говоря, обслуживание ошибок у них поставлено на конвейер – службы поддержки, бесплатная горячая телефонная линия, консультации он-лайн и через различные сервисы в сети. В данном случае под «ошибками» я подразумеваю не только непосредственно ошибки в программе, найденные пользователями, но и «ошибки в мозгах пользователей», связанные с обычной человеческой ленью и нежеланием разобраться и думать. С одной стороны – такая ситуация везде и всюду и считается обычной, а с другой – это повод, способ, средство знать еще сильнее «привязывать» пользователей к себе. Если бы большинство из нас внимательно читали бы документацию и перед тем, как задать вопрос, подумали над его решением, то половина вопросов исчезла, а службам поддержки нечего было бы делать.

Если вы «один из подающих надежды», тот самый герой, призванный удивить мир, то у вас просто нет права на ошибку. Вы должны выставить точно в цель, чтобы громко и красиво заявить о себе. В

противном случае велика вероятность того, что пользователи вас не поймут и забудут навсегда ваш адрес и ваш продукт и, развернувшись, уйдут к другим. Исключение в этой ситуации, составляет всеми горячо любимый Билл Гейтс, который, несмотря на то, что первые версии его операционной системы Windows содержали много ошибок и пользователями по всему миру изначально вообще не воспринимались, смог продать свою систему этому миру. Возможно, одна из причин его успеха в том, что первыми пользователями его системы были студенты, ставившие над ней эксперименты. А так как чаще всего то, что изучается в университете, дальше используется в повседневной работе, то Windows и получила такое распространение. Впрочем, умалять заслуг Билла Гейтса, как удачного менеджера и торговца не стоит. Думаю что, еще одна хитрость, принесшая успех Windows состоит в том, что долгое время вообще никто даже и не знал, что надо покупать лицензии на эту систему. Она эксплуатировалась везде и всюду просто так, благодаря умным ребятам, называющих себя хакерами. В свое время ходила шутка о том, что самая лицензионная часть операционной системы – драйвер от мыши.

Другая исключительная ситуация, может быть только одна – ваша программа настолько уникальна и нужна пользователю, что он вынужден терпеть и ждать, и прощать вам ваши ошибки. Правда, такие исключения бывают редко и чаще всего встречаются в сфере специфических видов деятельности человека. Например, программа расчета поведения группы микроорганизмов под влиянием факторов внешней среды, или программа учета параметров работы газотурбинного двигателя. Рядовому пользователю такие программы просто не нужны, подозреваю, что он даже не догадывается о существовании таких программ.

Если кратко подводить итог***** всему сказанному в этих абзацах, что я бы сказал так. Если вы, как разработчик задумались над удобством использования вашего продукта не на стадии

разработки, а гораздо позднее, то можно с уверенностью утверждать, что чем дальше вы от точки старта, тем больше затрат и усилий придется приложить, чтобы внести одно «элементарное» исправление. Это касается как общей логики функционирования, так и интерфейса в целом. Копеечная ошибка на старте всегда выливается миллионные затраты на ее исправление на финише. Необходимо как можно раньше начинать глядеть на ваше творение со стороны пользователя и делать это чаще и не просто глядеть (любоваться), а пытаться использовать все то, что вы создали, именно как пользователь. Забыть все, что известно о «внутреннем содержимом» продукта и стать неопытным пользователем, задающим извечные детские вопросы: «Как?» и «Почему?». Поверьте мне, это правило очень и очень часто помогает увидеть такие интересные вещи.

Элементы и проблемы... все мы родом из детства

Насколько я помню, начиная со школьных уроков информатики и продолжая на специализированных курсах университета, везде учат практически одинаково. Ученику рассказывают про алгоритмы, основы написания программ, грамматике языков программирования. Задавая типовые задачи, пытаются выработать логическое мышление, умение применять операторы и структуры изученного языка. Это все полезно и нужно, так как на самом деле заставляет думать и приводит к приобретению навыков логического мышления.

Я могу ошибаться, утверждая следующее, но на мой взгляд, практически нигде нет ни слова о том,

как сделать так, чтобы с программой, в общем смысле – продуктом, было удобно работать. Учат только языку, логике и умению применить знания на практике. А о том, чтобы умения, воплощенные в продукте, было еще и удобно использовать на практике, если и говорится, то уж очень мало, настолько мало, что об этом не остается даже воспоминания.

Говорить, что нас ничему не учили, тоже нельзя. Если чуть-чуть подумать, то можно немного провести аналогии, хотя это будет слишком размыто и не очень конкретно. Уроки рисования в школе, когда каждый как умел, так и рисовал, теоретически можно взять в качестве элементарной базы графического дизайна. Тоже можно сказать о черчении и геометрии, из них можно почерпнуть понятия о проекциях, симметрии, способах изображения объектов, пропорциях. Можно еще упомянуть историю, касаясь истории искусств, обычаев, культуры – оттуда можно взять некоторые элементы для декора, дизайна, оформления. Но, мне кажется, это все-таки сложно и «притянута за уши». Мало того, требует определенных усилий и особого мышления, чтобы все это собрать воедино. Пожалуй, из школьной программы – это и все, что можно взять в багаж, если не учитывать книги и специальные курсы, где непосредственно учат дизайну и оформительскому искусству, я их не рассматриваю, так как они идут факультативно и не входят в основную программу обучения, то есть массово не изучаются.

Если идти обучаться далее в университет (не могу со стопроцентной вероятностью утверждать, что в настоящее время ситуация не изменилась), то до недавнего времени в большинстве высших технических учебных заведений не было специально выделенной дисциплины, так или иначе связанной с интерфейсом. Студентам преподавались эргономика, трехмерная графика, черчение, инженерное моделирование, но увязать их с интерфейсом, опять же, можно лишь косвенно. Выделенной специализированной дисциплины, технических ВУЗах до недавнего

******* Замечание автора.**

Часто в процессе создания возникает иллюзия «понятности», – это состояние, в котором разработчик настолько проникается идеей продукта и его тонкостями, что ему начинает казаться, что и другим также все очевидно и понятно в функционале создаваемого продукта, как и ему самому. На деле же все выходит наоборот, – даже знающего и опытного пользователя всегда требуется обучать и отвечать на такие вопросы, которые с точки зрения разработчика находятся в ряду «элементарных».

времени не было. Я не беру во внимание обучение специальности дизайнер, так как считаю, что её изучает отдельно взятая группа людей и, распространяемые знания, не идут в широкие массы в отличие, например, от математики, физики, химии, изучаемых всеми в обязательном порядке на первых курсах университета.

Таким образом, молодой инженер, выпускник высшего учебного заведения с техническим уклоном, может оказаться хорошим специалистом, весьма подкованным логически и технически, но имеющим слабые знания в части дизайна, интерфейса и способов взаимодействия с пользователями. В результате, придя на рабочее место, молодому специалисту приходится восполнять этот пробел в знаниях самому и, порой, «изобретать велосипед» там, где он давно уже придуман. Иначе говоря, технические ВУЗы выпускают хороших специалистов, обладающих знаниями по своей специальности, но в разной степени хорошо умеющих эти знания красиво и удобно представить в конечном продукте. Именно по этой причине, как одной из основных, имеют такую популярность различные дизайнерские фирмы. Конечно же тому есть и другие причины – не все умеют созидаяще мыслить, красиво рисовать, придумывать, но научить делать это можно всех. С одной стороны удобно поручить создание интерфейса и дизайна знающим людям, а с другой стороны подобное происходит потому, что большинство просто не знает и не умеет это делать правильно. Итак, завершаю философствовать и перехожу к описанию отдельных элементов касающихся интерфейса и не только его.

Продолжение следует...

Ресурсы

- Техническое задание на разработку интернет-сайта http://www.rugost.com/index.php?option=com_content&task=view&id=182&Itemid=85
- Википедия. Интерфейс <http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81>
- Джеф Раскин. Интерфейс: новые направления в проектировании компьютерных систем <http://www.kodges.ru/9701-interfejjs-novye-napravlenija-v-proektirovanii.html>
- С.Круг. Не заставляйте меня думать http://www.bookshunt.ru/b10909_veb_dizajn_knig_a_stiva_kruga_ili_quotne_zastavlyajte_menya_dumatquot/download
- Об оформлении программной документации <http://www.raai.org/about/persons/karpov/pages/ofdoc/ofdoc.html>
- ГОСТ 19.102-77.Стадии разработки <http://www.nist.ru/hr/doc/gost/19102-77.htm>
- Студия Артемия Лебедева. Создание интерфейса навигатора «Штурман» <http://www.artlebedev.ru/everything/shturmann/process>
- Этот мерзкий, неудобный, противостественный оконный интерфейс <http://epikoiros.narod.ru/public/antiwind.htm>
- Обзор эргономических проблем и недостатков пользовательского интерфейса ПО бухгалтерского учёта на примере 1С:Предприятие 7.5 <http://www.usability.ru/Articles/1.htm>
- Ссылка на отдельное сообщение форума <http://forums.drom.ru/1067823597-post12.html>
- Размышления об интерфейсе <http://tclstudy.narod.ru/articles/mytk.html>
- Культура разработки программного обеспечения http://www.orientiryug.ru/kult_po.htm

Общаясь с друзьями и знакомыми, работающими в области ИТ, не раз приходилось слышать заявление, что Wi-Fi подходит только для сетей масштаба квартиры или небольшого офиса. В качестве аргументов приводились малая дальность, зависимость от погодных условий, ненадежность оборудования. Примерно такое же мнение бытует о том, что для провайдерских сетей не подходит платформа Windows. Аргументы: высокая ресурсоемкость, ненадежность и вообще неспособность работать в качестве системы биллинга.



Александр

by **WildHunter** <http://airnet.sytes.net>



А так ли это? Этот вопрос нас всерьез заинтересовал и подтолкнул к попытке создания беспроводной сети как раз на базе Wi-Fi и

Windows. Оговорюсь, что некоторый опыт создания беспроводных сетей у нас уже был, правда, немного другого назначения - корпоративных.

Начали мы естественно с постановки самим себе техзадания:

1. Сетевая технология Wi-Fi IEEE 802.11b/g, возможно в будущем 802.11a и 802.11n <http://ru.wikipedia.org/wiki/802.11>
2. Серверы на базе Windows Server. В качестве основной ОС был выбран Windows Server 2003, как достаточно надежная и неприхотливая система, при этом обеспечивающая нужный функционал http://ru.wikipedia.org/wiki/Windows_Server_2003
3. Максимальная простота развертывания, эксплуатации и администрирования сети.
4. Минимальные затраты, поскольку проект некоммерческий и финансирования со стороны не будет, придется обходиться собственными силами и средствами.
5. Надежность, производительность и пропускная способность, достаточные для обслуживания городского микрорайона.

Выбор оборудования

Основательно изучив рынок предложений недорогого Wi-Fi оборудования, мы остановили

свой выбор на нескольких моделях точек доступа (ТД) от D-Link: DWL-2100AP, DAP-1150 и DAP-1160 <http://www.d-link.ru>. Основными причинами выбора именно этих точек стали их приемлемая цена, накопленный в России и на Украине большой опыт их эксплуатации, а также наличие большого количества альтернативных прошивок, как от различных производителей оборудования, так и *open-source*.

Для тестирования было закуплено несколько экземпляров указанных выше точек доступа, а также различных антенн к ним. Началось тестирование в различных режимах, на разных расстояниях, с различными антеннами и прошивками. От «родных» прошивок мы отказались практически сразу, поскольку они рассчитаны как раз на применение в домашних или условиях небольшого офиса, не позволяют менять многие важные параметры (ACK timeout, мощность передатчика, чувствительность приемника и т.д.), а также практически не обеспечивают возможности мониторинга беспроводных соединений.

Точки **DWL-2100AP** оказались хороши для построения мостов на большие расстояния (до 50 км), но капризными в эксплуатации и несовместимыми со многими другими моделями Wi-Fi устройств. Также у них обнаружилась неприятная проблема: довольно часто при большой нагрузке «слетали» прошивки, а их восстановление оказалось довольно непростой процедурой. В итоге DWL-2100AP была признана нами годной к применению только для организации мостов. Хотя вполне возможно, что нам просто достались неудачные экземпляры.

DAP-1150 – очень простое и надежное устройство,

тем не менее отвечающее всем основным критериям хотспота. А с немного доработанной прошивкой от Conceptronic <http://www.conceptronic.net> эта точка стала практически полностью соответствовать нашим требованиям.

Основные характеристики DAP-1150 с прошивкой Conceptronic:

- *Стандарты:* 802.11b/g, 802.3/802.3u 10Base-T/100Base-TX Ethernet, ANSI/IEEE 802.3 NWay auto-negotiation.
- *Интерфейсы:* 802.11b/g беспроводная LAN, 1 порт 10/100Base-TX Ethernet LAN
- *Диапазон частот:* 2.4 - 2.4835 ГГц
- *Количество каналов:* 13 (ETSI)
- *Схемы модуляции:*
802.11b: DQPSK, DBPSK, DSSS, CCK
802.11g: BPSK, QPSK, 16QAM, 64QAM, OFDM
- *Режимы работы:* Station - Ad Hoc, Station - Infrastructure, AP, AP Bridge - Point to Point, AP Bridge - Point to MultiPoint, AP Bridge - WDS, Universal Repeater.
- *Скорость передачи данных:*
802.11g: 6, 9, 12, 18, 24, 36, 48, 54 Мбит/с
802.11b: 1, 2, 5.5, 11 Мбит/с
- *Чувствительность приемника:* до -100dBm
- *Выходная мощность передатчика:* 20dBm
- *Безопасность:* WEP, WPA/WPA2, фильтрация MAC-адресов, SSID broadcast disable.



Рис. 1. Точка доступа DAP-1150

- *Дополнительные возможности:* IAPP, встроенный Radius-сервер.

DAP-1160 с прошивкой AProuter <http://aprouter.com.br> превратилась в довольно серьезное устройство с широким функционалом, включающим в себя массу полезных возможностей, от контроля качества WDS-соединений <http://ru.wikipedia.org/wiki/WDS> до многофункционального шейпера [http://ru.wikipedia.org/wiki/Шейпинг\(информатика\)](http://ru.wikipedia.org/wiki/Шейпинг(информатика)).

Основные характеристики DAP-1160 с прошивкой AProuter:

- *Стандарты:* 802.11b/g, 802.3/802.3u 10Base-T/100Base-TX Ethernet, ANSI/IEEE 802.3 NWay auto-negotiation.
- *Интерфейсы:* 802.11b/g беспроводная LAN, 2 порта 10/100Base-TX Ethernet LAN
- *Диапазон частот:* 2.4 - 2.4835 ГГц
- *Количество каналов:* 13 (ETSI)
- *Схемы модуляции:*
802.11b: DQPSK, DBPSK, DSSS, CCK
802.11g: BPSK, QPSK, 16QAM, 64QAM, OFDM
- *Режимы работы роутера:* WISP Client, Bridge, Gateway, Router (WAN Ethernet), Router (WAN Wireless).
- *Режимы работы Wireless:* Wireless Client, AP, WDS+AP, WDS-Bridge.
- *Скорость передачи данных:*



Рис. 2. Точка доступа DAP-1160

802.11g: 6, 9, 12, 18, 24, 36, 48, 54 Мбит/с

802.11b: 1, 2, 5.5, 11 Мбит/с

- Чувствительность приемника: до -100dBm
- Выходная мощность передатчика: 20dBm
- Безопасность: WEP, WPA/WPA2, фильтрация MAC-адресов, SSID broadcast disable.
- Дополнительные возможности: IP Aliases, IAPP, Block Relay, Firewall, Traffic Control (шейпер), DDNS, Watchdog.

Антенны ANT24-0700C производства D-Link были выбраны по наилучшему показателю цена/качество в своем классе и из-за небольших размеров:



Рис. 3. Антенна ANT24-0700C

Основные характеристики ANT24-0700C:

- Диапазон частот: 2,4-2,5ГГц
- Сопротивление: 50 Ом

- VSWR: 1.92 (макс.)
- Максимальное усиление: 7dBi
- Допустимая мощность: 1 Вт
- Диаграмма направленности в вертикальной плоскости (Вектор E): 24 градуса.
- Диаграмма направленности в горизонтальной плоскости (Вектор H): 360 градусов.
- Разъем: Reverse SMA «мама» (встроенный в антенну), переходник с RP-SMA на RP-TNC (внешний).
- Материал корпуса: ABS, ABS+PC
- Рабочая температура: От -20 до 65 градусов.

Построение сети

После завершения тестирования мы приступили к созданию собственно сети. В качестве базовой была выбрана классическая топология «звезда», центральной точкой которой стала DAP-1160 в режиме Bridge WDS, а «лучами» DAP-1150 в режиме WDS + AP. Использование технологии WDS позволяет получить на каждой точке доступа

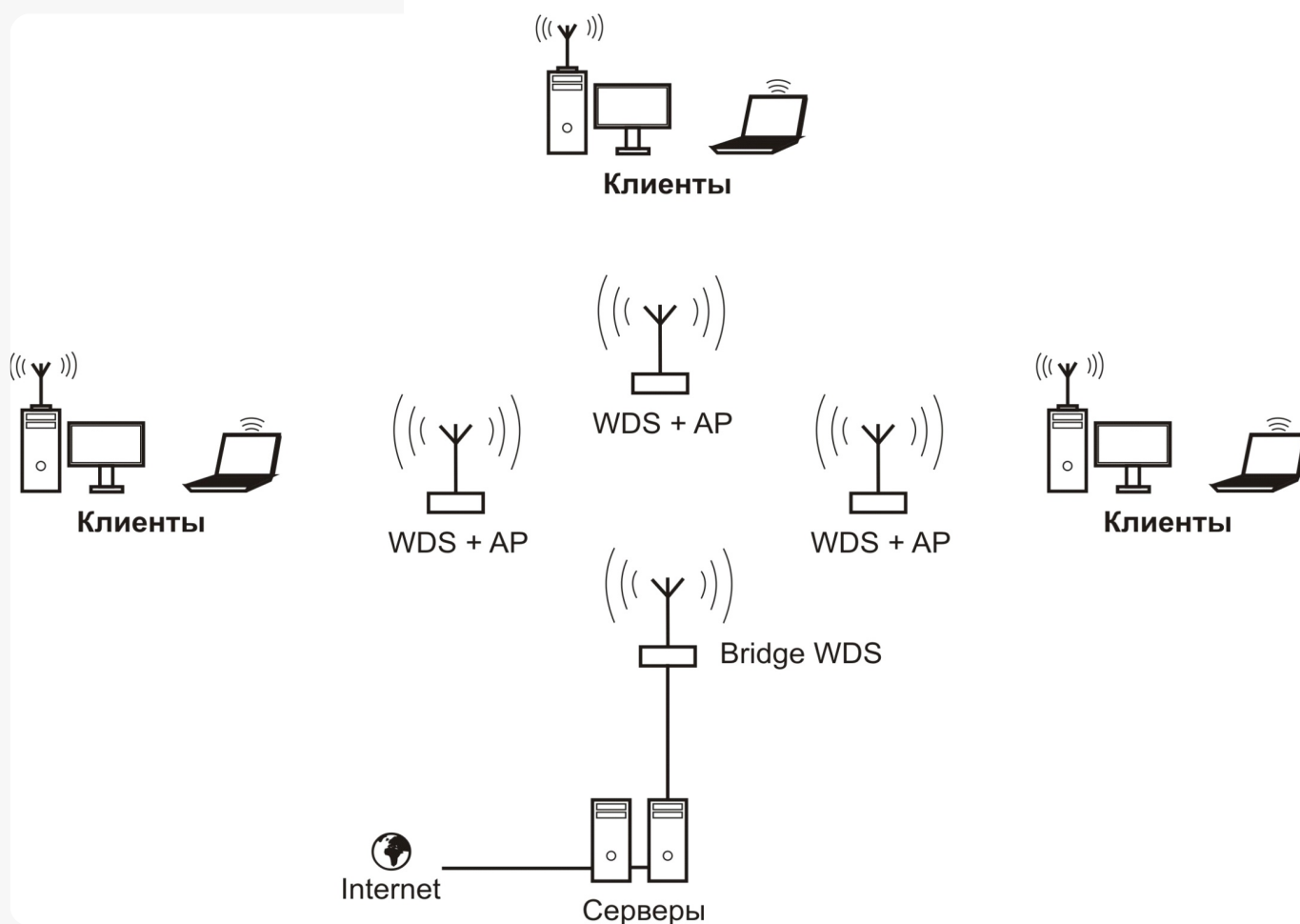


Рис. 4. Общая схема сегмента сети

скорость передачи данных на уровне 1.2-1.4 Мбайт/с, максимальная скорость у клиентов составляет порядка 550-600 кБайт/с. Реальная скорость передачи данных зависит от типа клиентского оборудования и качества беспроводного соединения клиент-ТД, а также от количества клиентов и общей загрузки сети.

Покрывание каждой ТД составляет (в радиусе, значения указаны при наличии прямой видимости или незначительных препятствий, например деревьев):

- Мобильные устройства (смартфоны, КПК, PSP и т.п.) - до 200м
- Ноутбуки, нетбуки, USB-устройства со встроенными антеннами - до 300м
- Ноутбуки, USB-устройства с внешними (или внутренними дипольными) антеннами - до 1500м
- ТД в режиме клиента с направленными антеннами - до 10км

Установка точек доступа

Естественно, что мы старались устанавливать ТД как можно выше, но не всегда это удается. Кроме того, при установке ТД на самую высокую мачту в округе, есть риск получить прямое попадание в нее молнии, от которого не спасет никакая грозозащита. Поэтому второй фактор, который мы старались учитывать - наличие поблизости громоотводов или более высоких мачт.

Все наши хотспоты стоят на мачтах высотой 4-6 метров, которые расположены на крышах зданий высотой в 5-6 этажей. Рядом есть небольшой массив из 9-этажных зданий, но нам не удалось договориться с местным ОСМД*.

Питание на точки подается по витой паре, обычно

* Комментарий автора.

Объединение совладельцев многоквартирного дома (ОСМД). Юридическое лицо, созданное владельцами для содействия использованию их собственного имущества и управления, содержания и использования неделимого и общего имущества.

из квартир наших клиентов (чтобы избежать бюрократии с ЖЕКа и ОСМД), по этому же кабелю клиенты входят в сеть. Взаимовыгодное сотрудничество: мы получаем место для установки и запитки точки, а клиенты получают VIP-доступ к сети :)

Качество

На следующем скриншоте показаны характеристики соединения с ТД на расстоянии ~1000 метров при прямой видимости (см. рисунок 5):

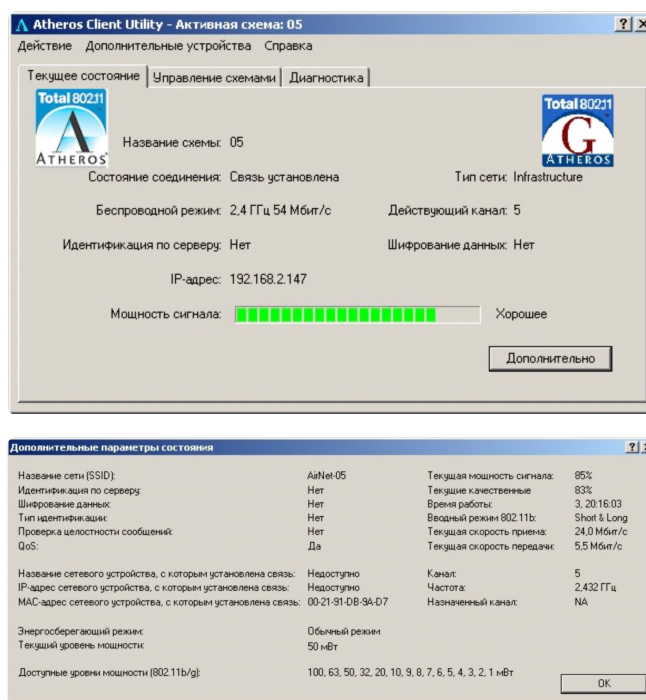


Рис. 5. Качественные характеристики соединения в Atheros Client Utility

ТД: DAP-1150, мощность 20dBm, антенна штыревая 7dBi.

Клиент: ноутбук Asus X51RL, WiFi-плата Atheros AR2425, антенна встроенная дипольная 4dBi.

Линк стабильный, текущая скорость линка «гуляет» в пределах 18-36 Мбит/с, это нормально. Скорость загрузки файла с сервера сети держится в пределах 400-500 Кбайт/с. Может показаться, что при средней скорости линка в 24 мегабита это немного, но нужно учитывать особенности технологии Wi-Fi, в которой мегабиты совсем другие, чем в кабельных сетях. Кроме того, режим

WDS также вносит свои коррективы, и не в большую сторону. Если вы ожидаете от Wi-Fi волоконно-оптических скоростей, то вам действительно лучше посмотреть в сторону оптики. А Wi-Fi это простая, недорогая, удобная и достаточно качественная связь для всех.

Как это ни странно, но на качество связи почти не влияют погодные условия. Если линк стабилен в хорошую погоду, то он таким и останется в дождь, снег или туман. Возможно, немного снизится скорость, но связь будет. Наш опыт практической эксплуатации показывает следующие результаты: в дождь (даже сильный) или снег уровень сигнала между ТД снижается в среднем всего на 1-4 dBm, туман никакого заметного влияния вообще не оказывает. Замечу, что все точки стоят на примерно одинаковом расстоянии от центральной, которое составляет 800-1200 метров. Для Wi-Fi при прямой видимости это расстояние несерьезное. Так что мнение о влиянии погоды на Wi-Fi сильно преувеличено. Влияние погодных условий становится заметным

только на больших расстояниях, от 5 км и более. А вот что действительно отрицательно влияет**: преграды, помехи в эфире, избыточная мощность радиоустройств. Данные факторы обсуждались и не раз, например на <http://www.lan23.ru>, поэтому подробно мы их здесь рассматривать не будем.

Каждая ТД нормально выдерживает до 20 одновременно работающих клиентов, если нет «тяжелого» трафика, в первую очередь P2P. Если есть, то даже один клиент в состоянии через минуту сделать для всех остальных вход в сеть недоступным, а минуты через 3-5 отправить точку на перезагрузку. ТД почти не критичны к

скорости или общему объему трафика, но очень критичны к количеству подключений. Это не значит, что использовать «тяжелый» трафик невозможно. Вполне возможно, если предпринять определенные меры для защиты сети от перегрузок. К таким методам защиты, например, относятся ограничение количества подключений на хост, работа по VPN, а также использование адаптивных шейперов и QoS. Но об этом поговорим в другой раз, так как это более относится к серверным технологиям, а не к Wi-Fi.

Несколько слов о клиентских устройствах

Сейчас на рынке представлена масса

самых разнообразных устройств 802.11b/g/n. Из них довольно трудно, но возможно выбрать оптимальный*** вариант. Некоторые устройства, например USB-брелки со встроенной антенной, рассчитаны на работу только внутри помещения и обладают очень ограниченным радиусом действия.

** Комментарий автора.

В последнее время технология Wi-Fi в наших краях сделала явный рывок, но непонятно, вперед, назад или «налево». В эфире полный бардак, огромное количество различных устройств, большинство из которых настроены как попало или вообще не настроены. Все это снижает качество связи, а если так и будет продолжаться дальше – связь вообще станет невозможной... Вот такая картина открывается с некоторых наших хотспотов:

SSID	BSSID	Channel	Type	Encrypt	Signal
AirNet-06	00:21:91:b:e0:6f	13 (B+G)	AP	no	56
AirNet-02	00:21:91:0b:07:73	13 (B+G)	AP	no	49
AirStream-01	00:1f:1f:7b:c8:1c	5 (B+G)	AP	no	30
AirTies_Air4240	00:1c:a8:75:8e:48	11 (B+G)	AP	no	26
	00:02:2d:c2:6a:42	3 (B)	AP	no	24
govd	00:1e:58:e9:84:91	9 (B+G)	AP	yes	16
AIR	00:50:7f:8f:e8:48	13 (B+G)	AP	yes	16
Broadcom	00:22:b0:be:af:f3	11 (B+G)	AP	yes	13

TP-LINK	00:23:cd:d0:16:be	6 (B+G)	AP	no	21
Home	00:26:5a:db:78:81	11 (B+G)	AP	yes	21
edi	00:1f:1f:93:92:ca	7 (B+G)	AP	yes	9
AirNet-03	00:21:91:12:79:14	13 (B+G)	AP	no	6
TP-LINK	00:1d:0f:a7:8f:99	6 (B+G)	AP	no	4
Teplonet	00:1e:58:2b:d0:89	1 (B+G)	AP	no	4
JoikuSpot_0025CF43C72F	96:8e:23:06:38:e5	10 (B+G)	Ad hoc	no	1
Domik	00:1e:58:e5:5c:91	11 (B+G)	AP	yes	1
AirNet-02	00:21:91:0b:07:73	13 (B+G)	AP	no	1

SSID	BSSID	Channel	Type	Encrypt	Signal
AirNet-04	00:21:91:14:7c:03	13 (B+G)	AP	no	100
AirNet-02	00:21:91:0b:07:73	13 (B+G)	AP	no	53
AirNet-06	00:21:91:b:e0:6f	13 (B+G)	AP	no	44
	00:02:2d:c2:6a:42	3 (B)	AP	no	43
AirStream-01	00:1f:1f:7b:c8:1c	5 (B+G)	AP	no	40
govd	00:1e:58:e9:84:91	9 (B+G)	AP	WEP	21
GM170	00:15:6d:ad:54:df	9 (B+G)	AP	no	18
	00:02:2d:c2:6a:e2	7 (B)	AP	no	18
Andrik	00:0e:2e:4b:bc:9d	11 (B+G)	AP	WEP	16
DLink	00:26:5a:db:80:ab	11 (B+G)	AP	no	15
DLink	00:24:01:8a:df:2f	11 (B+G)	AP	WPA-PSK	12
AIR	00:50:7f:8f:e8:48	13 (B+G)	AP	WPA2-PSK	9

Даже одна стена или дерево под окном, для них уже неодолимая преграда.

Другой класс – это устройства с внешними антеннами. Даже со штатными антеннами они показывают неплохую дальность при прямой видимости, а при установке хорошей антенны вполне могут работать на расстояниях, измеряемых километрами. Поэтому при выборе Wi-Fi устройства стоит обратить внимание не только на дизайн, размеры и цену, но в первую очередь – на технические характеристики.

При покупке ноутбука, КПК или любого другого устройства, оснащенного встроенным Wi-Fi, также обязательно интересуйтесь характеристиками Wi-Fi платы, типом и характеристиками встроенной антенны. Зачастую, встроенные платы подходят только для связи внутри комнаты или на расстояниях до 100 метров на открытой местности. Хотя есть и большое количество устройств, оснащенных качественным Wi-Fi и вполне способных поддерживать связь на приличных расстояниях.

Основные характеристики Wi-Fi устройств

1. Стандарты, поддерживаемые устройством. От этого зависит универсальность и максимальная скорость передачи данных. В настоящее время в Европе поддерживаются следующие стандарты:

- 802.11b (максимальная скорость 11 Мбит/с)

*** Комментарий редакции.

Многие задаются вопросом: «Как увеличить дальность связи?». Перво-наперво, при выборе WLAN-аппарата, обратите внимание на то, чтобы выходная мощность была как можно ближе к разрешенной 20 dB. Следующим решающим фактором является чувствительность. У лучших современных аппаратов она находится на уровне -97 dB. Чем выше чувствительность к слабым сигналам, тем выше дальность. Но это палка о двух концах, так как мы не учитываем при этом помеховую обстановку вокруг.

Как влияют эти величины на дальность связи? К примеру, аппарат с мощностью в 20 dB сможет обеспечить в два раза большую дальность приема по сравнению с 14 dB, т.е. разница в 6 dB дает двойной выигрыш. Если к этому прибавить, что аппарат с чувствительностью -97 dB, позволяет получить выигрыш в 4 раза по сравнению с аппаратом, у которого чувствительность равна -76 dB, то общий выигрыш будет 8-ми кратным.

- 802.11g (максимальная скорость 54 Мбит/с)

- 802.11n (максимальная скорость 150/300 Мбит/с)

2. Мощность передатчика и чувствительность приемника. Не стоит чрезмерно увлекаться мощностью, слишком мощный сигнал в городских условиях скорее вреден, так как вызывает многократные отражения и наложения, а также создает массу помех. Да и для здоровья он не очень полезен. На наш взгляд оптимальные значения составляют до 100 мВт (20 dBm) мощности и -95 dBm чувствительности.

3. Возможность подключения внешней антенны и характеристики штатной. «Главная часть любого радиоустройства – антенна», это вам скажет любой специалист по любому виду радиосвязи. К Wi-Fi это тоже относится в полной мере. Для нормальной работы (приличной дальности и скорости) любого Wi-Fi-устройства необходима антенна с коэффициентом усиления не менее 4dBi.

4. Функционал устройства. К этому относится все, что устройство умеет делать и насколько удобно его использовать. По этому критерию выбор огромный, от самых простых до сложнейших устройств, которые «умнее» вашего компьютера. Главное – достаточно четко представлять, чего вы хотите и сколько вы согласны за это заплатить.

Заключение

Это основные характеристики, но существует еще масса различных тонкостей, специфических моментов и технических особенностей. Рассказать здесь обо всех нереально, поэтому выбор оборудования – совсем непростое дело, в котором не помешает консультация специалиста. Если вы, конечно, сами не являетесь таковым :)

Продолжение следует...

Ресурсы

- Сетевая аутентификация на практике <http://www.citforum.ru/nets/articles/authentication>
- Крупнейший и самый активный сайт рунета по Wi-Fi <http://www.lan23.ru/index.html>
- Антенны, много антен... хороших и разных <http://radiowolna.narod.ru>

В последнее время появилось много публикаций на тему нейронных сетей, однако большинство из них научного характера, предназначенные, как правило, для специалистов, пресыщены формулами и абстракциями, не дают четкого представления о предмете. В этой статье мы попытаемся понять основные принципы работы и устройство нейронной сети. Статья, дающая вводные представления о нейронных сетях...



by **Utkin** www.programmersforum.ru

*«Настоящий программист должен: пройти все уровни Тетриса, пользоваться только своим бубном и написать нейронную сеть»
/ Махабхарата, эпос народов Индии*

Нейронные сети – это математические модели биологических нейронных сетей, выраженные как программным, так и аппаратным способами. Поэтому, сначала рассмотрим единицы, из которых состоит биологическая нейронная сеть – нейроны. Нейроны – это специальные биологические клетки, объединенные в нервную систему организма. Это нужно для осуществления нервной деятельности, а именно принятие информации об окружающем мире (и внутреннем состоянии организма), выработка решений и управление исполнительными органами. Нейроны бывают нескольких видов, но мы рассмотрим только один, который для простоты восприятия классифицируем как классический вид нейронов, т.е. тех, что в основном используются в компьютерных моделях. Итак, нейрон представляет собой клетку, имеющую несколько отростков, один из которых является выводом нейрона, а остальные являются его входами. Вывод нейрона называется аксоном, входы – дендритами, а точка соединения аксонов и дендритов называется синапс. Нейроны по своей сути схожи с микропроцессорами (или ядрами процессора) и фактически занимаются обработкой информации, поступающей на их дендриты и выдающие результат на аксон. Практически нейроны работают с электрическими импульсами (только на основе ионов – прямая передача электронов в жидких системах не очень удачная вещь, приводящая обычно к разрушению жидкости либо емкости, в которой данная жидкость находится). Ничего не напоминает? Иными словами, нервная система подавляющего большинства организмов – есть огромная электрическая схема. Один нейрон может



соединяться с несколькими тысячами других, что дает высокий параллелизм (именно параллелизм, а не модную многопоточность). Некоторые принципы работы нейронов не установлены (потому что нейроны-то у всех есть, а вот электронный микроскоп не у каждого), однако известно, что существует класс нейронов, использующий две группы входов – возбуждающие и тормозящий. Сигналы на возбуждающих входах заставляют нейрон генерировать сигнал на аксоне, тормозящие соответственно подавляют выходной сигнал. Дендриты имеют порог срабатывания, то есть требуют, чтобы на их вход поступал сигнал определенного уровня, иначе он будет не засчитан, И который может изменяться под воздействием ряда факторов – например попадания определенных веществ, таких как гормоны. Время срабатывания нейронов относительно небольшое: 2-5 мс, Однако более 6 миллиардов нейронных сетей доказали, что с их помощью можно решать довольно-таки сложные задачи, такие как: разум, научно-технический прогресс и создание цивилизаций (наравне с такими задачами как разрушение природы и конкурентных видов, а также активная и очень эффективная внутривидовая борьба). Более того, миллионы лет эволюции убедительно доказывают, что такого быстрого действия вполне достаточно для решения задач реального

времени. Компьютерные же модели пока, что ушли не так далеко и используются в научных целях (в основном для того, чтобы понять каким же образом 6-ти миллиардам нейронных сетей вообще удалось выработать такую концепцию как нейронные сети).

Анамнез

Несмотря на громкие заявления, реальное использование нейронных сетей на практике ничтожно в сравнении с традиционными алгоритмами. Основная причина – неточное формулирование задач, результаты которых также не очевидны – такие как прогнозирование погоды, биржевые сводки, в общем, все, что сводится к гаданию на кофейной гуще и где нельзя однозначно поймать за руку. Наиболее серьезным является применение нейронных сетей для распознавания образов на базе персептронов (сети, где нейроны сгруппированы в слои), что не удивительно, теоретические предпосылки данных концепций (и персептронов и распознавания образов и распознавания образов на персептронах) были разработаны 60-70-х годах прошлого столетия, примерно в то же время, когда был создан автомат для автоматического распознавания индексов на почтовых конвертах. В последнее время мощностей обычных компьютеров вполне достаточно для создания полноценных нейронных сетей (чем мы собственно и будем заниматься), что позволяет все чаще применять их на практике (например, интересной темой является использование нейронных сетей для сжатия информации). Следует сразу же предупредить: использование нейронных сетей в задачах, алгоритмы которых легко перенести на языки программирования, в подавляющем большинстве случаев не эффективно (обычно по быстродействию).

Настольная инструкция по приготовлению нейронных сетей

Прежде чем писать программу, имитирующую биологические нейроны, нужно выработать модель. Я предлагаю следующую модель нейрона...

Каждый нейрон имеет 32 входа, из них 16

положительных (считаю, использование термина возбуждающие выходы, здесь использовать не стоит) и 16 отрицательных. Выход соответственно один. Сразу договоримся о терминах – входы будем называть дендритами. Хотя в некоторых литературных источниках используется обозначение синапса, это на самом деле не совсем верно (почему написано выше). Выход также будем называть аксоном. Входы получают сигналы по принципу: есть сигнал / нет сигнала (0/1 или ложь/истина). Сам нейрон будет работать по принципу сумматора – он складывает все сигналы на каждой из групп входов. Соответственно, если число сигналов на положительных входах больше, чем на отрицательных, то нейрон устанавливает сигнал на выходе (аксоне). В обратном случае сигнал с выхода снимается (даже если сумма сигналов на положительных входах равна сумме сигналов на отрицательных).

Для данных нейронов порог срабатывания дендритов будет всегда и для всех одинаковым и равным уровню выходного сигнала. То есть наш нейрон будет работать с логическими величинами, и фактически будет являться предикатом (функцией возвращающей результат логического типа). Здесь по законам жанра научно-популярных статей следует погрузиться в обилие формул и не только математических. Но, как правило, для большинства читателей таких статей они абсолютно бесполезны, поэтому мы приводить их здесь не будем (кому интересно, найдет в прилагаемых источниках).

Почему 32 входа? Это компромисс между производительностью сети и быстродействием компьютера. Зависимость здесь следующая – чем больше выходов имеет нейроны, тем больше вычислительная мощность сети и тем больше вычислительных ресурсов требуется на реализацию.

Сам по себе нейрон устройство узкоспециализированное и его использование не в сети (даже если она и состоит из одного нейрона) является весьма проблематичным занятием, и поэтому далее мы рассмотрим модель нейронной сети.

Итак, помимо самих нейронов, сеть может

содержать таблицу входных данных и таблицу выходных данных. Таблица входных данных характеризует информацию, поступающую в нейронную сеть – в биологии ее прототипом являются рецепторы. То есть датчики, с которых берется информация о задаче. Нейроны подключаются входами к таблице входных данных (заодно и к выходам нейронов) и формируют результаты, часть из которых помещаются в таблицу выходных данных. Данные, помещенные в таблицу выходных данных, символизируют решение поставленной задачи. Существует множество вариантов нейронной сети, но наиболее распространены персептроны – нейронные сети, где нейроны объединены в группы (слои). Обычно нейроны одного слоя могут соединяться с выходами нейронов другого, конкретного слоя, но бывают и исключения. В нашем варианте мы будем использовать хаотичное соединение нейронов (здесь имеется ввиду, что если порядок соединения нейронов и существует, то на данный момент он неизвестен), каждый нейрон имеет право быть соединенным с любым объектом нейронной сети, включая и таблицу входных данных, и таблицу выходных данных. Потому что это ближе к реальной биологической модели и строгих доказательств того, что нейроны объединены в группы или слои не обнаружено. Зато обнаружены нейроны – цель которых только передача импульса от входа к выходу, то есть это удлинители, которые соединяют между собой нейроны (те, что не могут быть соединены между собой напрямую ввиду их расположения).

Серьезным недостатком решения задач на нейронных сетях является отсутствие четких условий решения при постановке задачи перед нейронной сетью. Иными словами нельзя точно сказать, сколько нейронов требуется для решения данной задачи или достаточно ли данного числа нейронов для получения положительных результатов. Возможно, при определении конфигурации нейронной сети будет выбрано недостаточное число нейронов и решение задачи никогда не наступит. Существует ряд работ направленных на решение этой проблемы [1-5], однако до успешных результатов пока далеко (опять-таки, несмотря на заверения академиков и обилие формул). Сейчас выбор параметров в основном

определяется на основании предыдущих опытов, либо экспериментальным путем. Я же выбрал хаотичную модель по двум причинам. Во-первых, персептроны не способны решать некоторые задачи независимо от числа нейронов в них (это было известно еще в Советском Союзе), а во-вторых, персептроны есть ограниченное подмножество моделей с хаотичным образованием нейронов и при изменении связей можно добиться получения персептрона практически любого типа. А возможность замыкания отдельных входов нейрона на его же выход или на константные сигналы позволяет имитировать дискретные уровни порога срабатывания.

Важной способностью нейронной сети – является возможность ее обучения за счет изменения порогов срабатывания и/или переключения связей между нейронами. Поскольку в нашем случае нейроны имеют одинаковые пороги срабатывания на всех входах и выходах, то обучение нашей модели будет происходить через изменение связей между нейронами. Кстати, на изменениях порога срабатывания есть алгоритмы автоматического обучения нейронных сетей, но опять же все это дело весьма и весьма условно. Потому что обучение возможно также только для некоторых видов задач и потому что такое обучение вырождается в генетический алгоритм, а это уже другая история, хоть и смежная (естественно исследователи не ищут легких путей). Обучение нейронных сетей такого типа производится случайным изменением связей между нейронами (как правило). Происходит это следующим образом. Пусть имеется некоторая задача, заключающаяся в решении некоторой функции $f(x)$. То есть на каждое состояние таблицы входных данных должно быть только одно состояние таблицы выходных данных. Возьмем избитый пример – распознавание образов. Здесь на каждую картинку имеется соответствующая цепочка сигналов. Имеющейся сети дается тестовое задание – серия образов, затем сеть прореживается и получившаяся серия результатов сравнивается с эталонными данными. Если сеть отвечает требованиям задачи, то ее уже можно использовать (чего с первого раза на практике не бывает). Если же нет, то текущая конфигурация запоминается и

на ее основе формируется новая сеть путем случайного изменения связи случайного дендрита случайного нейрона. Процесс распознавания повторяется. Теперь уже сравниваются оба результата тестирования: первоначальный и новый, полученный в результате мутации (здесь много биологических терминов). Например, сравнивать можно по проценту правильных результатов в серии распознавания образов. Теперь за основу берется та сеть, в которой процент совпадений больше и процесс повторяется до тех пор, пока результат тестирования не даст полного совпадения с эталоном. Или заранее определенное количество раз, иначе есть вероятность бесконечного процесса обучения. Здесь же нужно сразу определиться какой вариант сети лучше, в случае если оба варианта дают одинаковый процент узнаваний (новый вариант сети предпочтительней).

Подводные камни и течения

А что вообще может решать нейронная сеть? Теоретически даже может решить теорему Ферма. Для этого требуется не так уж много – нейронная сеть с числом нейронов примерно 10^{12} – 10^{15} степени, нейроны должны иметь возможность соединяться с несколькими тысячами других (порядка 20000). Если еще не понятно, то это мозг человека – лучшая иллюстрация работы нейронных сетей. На самом деле нейронов требуется еще меньше, потому что значительная их часть требуется на обслуживание и управление полуавтоматическими системами, таких как легкие, мышцы, желудок, саморегуляция и т.д. А также на передачу данных для других групп нейронов в те же самые органы. Плюс обработка огромного количества датчиков. Такую нейронную систему можно считать эталонной, но не идеальной. На ее обучение требуются годы.

На самом деле нейрон очень мощная логическая единица. С помощью нее можно эмулировать, например такие элементы как логическое «И», логическое «ИЛИ» и логическое «НЕ», то есть практически все современные логические операции легко могут быть выражены через нейронные сети. Одной из причин сложности конструирования эффективных сетей является

тот факт, что логика на нейронах на порядок мощней обычной логики и всех ее смежных дисциплин, потому-то и выразить ее в рамках стандартной довольно-таки проблемно без привлечения интегралов и прочих трехэтажных формул (включая и логических). Далее с помощью нейронов можно эмулировать и работу сразу сложных блоков, таких как триггеры, счетчики, шифраторы, дешифраторы и т.д. И, наконец, нейронная сеть позволяет эмулировать аналоговые элементы (при наличии творческой жилки у программиста) и сложные схемы (на манер программ Qucs, Electronics Workbench, Microcap и т.д.).

Далее, нейронные сети способные решать задачи даже, если никогда до этого не сталкивались с такими условиями ранее, хаотичные соединения позволяют формировать различные как положительные, так и отрицательные обратные связи, что может порождать решения на грани интуиции. Однако в таких условиях возникает новая проблема – подобно человеку, нейросети способны ошибаться.

Еще одна частая ошибка (наблюдается даже в серьезных трудах) – это временные интервалы функционирования нейрона относительно других в нейронной сети. Представим работу какого-либо нейрона. Итак, он прочитал данные и сформировал новое состояние аксона. В результате какой-либо другой нейрон будет читать уже новое состояние аксона, и работа всей системы в целом будет нарушена (потому как нейрон может обратиться к любому другому нейрону и не факт, что тот уже поменял свое состояние на новое). Проблема усугубляется тем, что новое значение на первом нейроне не всегда влияет на состояние последующих, а только для некоторых дендритов или некоторых состояний системы. Далее существует вероятность, что данный нейрон также изменит свое состояние и т.д. Таким образом, результаты работы могут быть полностью искажены. Отчасти благодаря такой проблеме персептроны и получили такое распространение. В них слои, нейроны и их взаимосвязи организованы в иерархии таким образом, что все нейроны всегда получают новые сигналы, то есть сначала первый слой берет данные из рецепторов, второй слой берет данные из первого слоя и т.д. Но нас это ни как не

останавливает, решение этой проблемы снижает быстродействие, но зато позволяет имитировать нейронные сети любой конфигурации. Смысл заключается в кэширование результатов работы нейрона. Иными словами нейрон изменяет (или не изменяет) состояние аксона не сразу, а только после того, как абсолютно все нейроны в сети выполняют свою работу. Только после этого происходит изменение состояния всех нейронов. Это гарантирует, что все нейроны получают достоверные сигналы, и, следовательно, выработают достоверные результаты.

Далее следует правильно трактовать результаты работы. Для простых задач, решение которых однозначно описывается $f(x)$, то есть один параметр и только одно результирующее значение для каждого значения входящего параметра, это самое результирующее значение перестает играть особую роль. Всегда можно написать транслятор результатов, с использованием стандартных средств программирования, в случае если их число не велико. Объясню на примере все того же распознавания образов. Допустим, перед сетью стоит задача распознавания образов букв А, В и С. Результатом должно быть 2 бита (см. таблицу 1 и 2):

Таблица 1. Вариации распознавания. Вариант-1

Входной образ	Результат	
Буква А	0	1
Буква В	1	0
Буква С	1	1
Любое другое изображение	0	0

Таблица 2. Вариации распознавания. Вариант-2

Входной образ	Результат	
Буква А	0	1
Буква В	1	1
Буква С	1	0
Любое другое изображение	0	0

И нет особого смысла мучить сеть мутациями (особенно если она состоит из тысяч нейронов), гораздо быстрее и проще написать транслятор, который брал бы данные из таблицы выходных значений и выдавал требуемый результат (см. рисунок). Как уже было отмечено ранее, не стоит ожидать сразу ошеломляющей эффективности при использовании нейронных сетей, по разным

причинам. Одна из них (необъективная) связана с восприятием самого человека. Как известно, приборы неизбежно вносят свои погрешности в результаты измерений, аналогично и человек при оценке работы нейросети, в большинстве случаев вносит в результаты свои погрешности.

Продолжим с тем-же распознаванием образов. Требуя от сетей четкого и однозначного определения результатов, очень часто человек сам не в состоянии адекватно оценить предлагаемое изображение по ряду причин (это связано не только со зрением). В тоже время человек способен распознать образ даже очень плохого качества, основываясь:

- а) на предыдущем опыте – что, как правило, недоступно для нейронных сетей. Иными словами, человек не только обучен узнавать образ, но он может и просто помнить его, что значительно упрощает идентификацию объекта. Нейросетям для запоминания же требуется несколько большее количество нейронов, чем у них есть.
- б) на информации, которая недоступна нейронным сетям по условиям задачи. Самый простой пример – это разбор рукописного текста. Чтобы понять все слова совершенно необязательно понимать все буквы – этот эффект давно известен и в частности им пользуются американцы в повседневной речи. Они проглатывают окончания (иногда середину) слов, просто не договаривая их. Также и человек, распознав большинство членов предложения, в состоянии восстановить слово исходя из контекста, а не за счет определения символов слова.

Теперь представим, что у нас уже имеется нейронная сеть (в смысле компьютерная модель). Она способна решать задачи и вроде все отлично, но настоящего исследователя такая

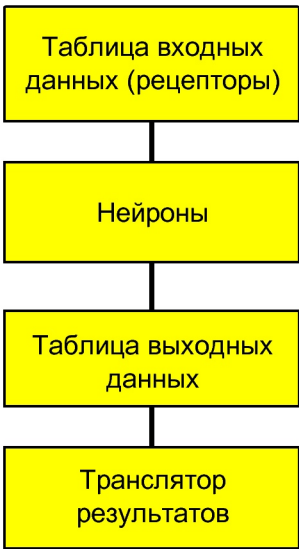


Рис.1. Алгоритм

продвижения данных

позиция нисколько не устраивает – как это работает? Может для решения задачи требуется меньше нейронов? Что будет если отключить вон тот нейрончик? Ведь чем меньше в сети нейронов, тем меньше ресурсов требуется для ее выполнения, а скорость выполнения – один из серьезных недостатков, сдерживающий развитие науки о нейронных сетях.

Заблокировать нейрон можно, если его заставить читать информацию из самого себя (и только из самого себя). Тогда на его аксоне будет пожизненный ноль (напомню, что в хаотичной модели нейронной сети любой нейрон имеет право адресоваться к любому источнику сигнала, включая и самого себя).

Далее анализ, проведенный на базе имитации основных логических элементов, показывает, что нейронные сети более предрасположены к обучению, если помимо изменяемых данных они содержат в себе некоторые константные сигналы (в большинстве случаев достаточно постоянного 1 и постоянного 0). Получить их можно искусственно, например, жестко задать неизменяемое значение в таблице входных сигналов. Ну и ноль всегда можно получить, заблокировав нейрон.

Как это ни странно, но к нейронным сетям можно применять и некоторые стандартные средства отладки. Один из них – контрольные точки. Можно получать данные с одного нейрона и писать их в массив для дальнейшего анализа. Какого? Самый примитивный пример – если нейрон не меняет своего значения на протяжении всей работы нейросети, независимо от входящих параметров, то следует задуматься над этим фактом. Может проще все дендриты, которые подключены к нему, переключить на константные значения в таблице входных значений? Фактически такой нейрон выполняет транспортную функцию – он передает константу (или формирует ее) для тех нейронов, которые подключены к нему. Это актуально для биологических нейронов (не может нейрон из мозга напрямую подключиться к нейрону из копчика), но бессмысленно для нашей сети – каждый нейрон имеет возможность напрямую подключаться к источникам сигнала. Более сложный анализ предусматривает сбор

информации и ее сравнение с группой нейронов. Цель – поиск дубликатов, то есть устранение все тех же нейронов, выполняющих транспортную функцию.

Можно также поискать нейроны, выполняющие бесполезную работу. Это нейроны, к которым никто не обращается, то есть ни другие нейроны, ни таблица выходных данных. Соответственно и результаты их работы не нужны. Аналогично можно поступить и со значениями из таблицы входных значений: если к данному элементу никто не обращается, то возможно данный элемент просто не нужен, либо сеть работает неправильно и требуется расширенное тестирование. Задача всех оптимизаций такова, чтобы получить нейронную сеть без элементов, работа которых не влияет на результат работы нейронной сети: кто не работает, тот не должен есть ресурсы компьютера.

Препараты

Здесь мы рассмотрим структуры данных, с помощью которых можно создать нейросеть. Запись будет производиться на языке Дельфи. Однако я постараюсь дать развернутое обоснование выбранных полей, так чтобы нейронную сеть легко можно было организовать и с помощью других языков программирования.

Прежде всего, нам нужна модель нейрона, который будет являться центральным элементом нейронной сети. Итак, предлагаю следующую модель:

```
type
  TNeron = class (TObject)
  protected
    Akson: Boolean;           // Выход нейрона
    Akson2: Boolean;          // Кэш нейрона
    Dendrits: Array [0..31] of Integer; // Входы
                                // нейрона (как адрес другого нейрона)

  private
  public
    // Подготовка к работе нейрона
    constructor Create;
    destructor Destroy; override;
    procedure Init();         // Инициализация нейрона
                                // Организация доступа

    // Установка связи дендрита
    procedure SetDendrit(Num, Value: Integer);
```

```

// Запись данных из кэша
procedure Update();

// Запись значения в кэш
procedure SetAkson2(Value: Boolean);
// Чтение аксона
function GetAkson(): Boolean;
// Чтение значения дендрита (связи)
function GetDendrit(Num: Integer): Integer;

end;
```

Akson – значение логического типа, это выход, откуда будут читаться результаты работы нейрона.

Akson2 – это кэш аксона, предназначен для синхронизации работы нейронной сети.

Dendrites – это массив ссылок на аксоны и другие элементы нейронной сети. Всего их 32 и на данном этапе нет различий между положительными и отрицательными входами (это делается программно).

Ссылка представляет собой идентификатор объекта, в качестве которого может выступать:

- а) нейрон;
- б) элемент таблицы входных данных;
- в) элемент таблицы выходных данных.

Вообще-то, в нейронной сети каждый тип элементов организован в свои собственные динамические массивы и имеет свой собственный индекс для доступа. Общий же (или абсолютный) идентификатор вычисляется для удобства использования и адресации в тех методах нейронной сети, где это непосредственно требуется. Собственно ничего сложного нет – нейрон, по сути, хранилище данных, не реализована даже функция работы нейрона. Просто потому, что его работа без нейронной сети невозможна. Все методы направлены в основном на чтение/запись полей класса.

Вообще реализацию можно было сделать и без класса (даже проще), но потом это отразится на удобстве дальнейшей модификации нейронной сети. Теперь рассмотрим модель нейронной сети:

*** Комментарий автора.**

Обратите внимание, идентификатор является общим для всех элементов (в том смысле, что, используя данный идентификатор, дендрит, может обратиться к любому объекту, включая и свой собственный аксон).

type

```

TNNNet=class

protected

    // Поля
    Data: Array of TNeuron; // Нейроны
    Count: Integer; // Количество нейронов
    TableIn: Array of Boolean; // Рецепторы
    CountIn: Integer; // Количество рецепторов
    TableOut: Array of Boolean; // Таблица результатов
    TableOut2: Array of Integer; // Ссылки на
                                // существующие нейроны,
                                // откуда читать сигналы

    CountOut: Integer; // Число сигналов в таблице рез-в

private

    // Выполнение указанного нейрона
    function RunNeuron(Neuron: Integer): Boolean;

    // Перенос сигналов из кэша нейронов на их выходы
    procedure Updating();

    // Берем все значения для таблицы выходных данных
    procedure OutPutting();

public

    procedure InitNeurons; // Инициализация набора нейрона
    procedure InitTableIn(); // Инициализация рецепторов
    // Инициализация таблицы выходных сигналов
    procedure InitTableOut();

    // Конструктор нейросети
    constructor Create; overload;
    constructor Create(InTable, Neurons, OutTable:
                                Integer); overload;

    // Деструктор нейросети
    destructor Destroy; override;

    // Работа с рецепторами
    // Устанавливаем число рецепторов
    procedure SetCountIn(Value: Integer);
    // Устанавливает все рецепторы в False
    procedure ClearTableIn();

    // Возвращает число рецепторов
    function GetCountIn(): Integer;

    // Устанавливает сигнал для указанного рецептора
    procedure SetElemIn(Num: Integer; Value: Boolean);

    // Запись сигналов сразу для всех рецепторов
    procedure SetTableIn(Value: Array of Boolean);

    // Работа с таблицей выходных данных
    // Устанавливает число элементов таблицы вых-х данных
```

```

Procedure SetCountOut(Value: Integer);
// Возвращает число элементов таблицы вых-х данных
Function GetCountOut(): Integer;
// Чтение выходного сигнала
Function GetValueOut(Num: Integer): Boolean;
// Читаем линк для данной ячейки таблицы вых-о сиг-а
Function GetLinkOut(Num: Integer): Integer;
// Установка линка на нейрон
Procedure SetLinkOut(Num, Value: Integer);
// Очистка таблицы выходных сигналов
Procedure ClearOut();

// Работа с нейронами
// Установка линка
Procedure SetDendrit(Neron, Num, Value: Integer);
// Установка линка (абсолютная адресация в рамках
// нейросети)
Procedure SetDendrit2(Neron, Num, Value: Integer);
// Чтение линка
function GetDendrit(Neron, Num: Integer): Integer;
// Выполнение одного шага нейросети
Procedure Run();
// Выполним указанное число циклов нейросети
Procedure Run2(Tik: Integer);
// Минимальный запуск
Procedure Run3();
// Задаем количество нейронов
Procedure SetNeronCount(Value: Integer);
// Мутация нейросети (произвольного входа
// произвольного нейрона)
Procedure Mutation();
// Формирование сети с заданным количеством нейронов
// и данных в таблице
Procedure Generator(InTable, Nerons, OutTable:
                    Integer);
end;
```

Как уже было отмечено ранее, все типы элементов нейросети сгруппированы в рамках своего типа в динамические массивы. Здесь только следует обратить внимание на **TableOut2** – эта структура относится к таблице выходных данных и представляет собой ссылки на нейроны, откуда следует читать информацию в элементы таблицы. Все **Count**'ы в данном случае являются счетчиками числа элементов в массиве. В принципе, число элементов динамического

массива можно узнать и в Дельфи, не пользуясь дополнительной переменной. Но наш пример учебный и предназначен для понимания принципов работы нейронной сети. Оптимизацию можно проводить уже после ознакомления с данным примером.

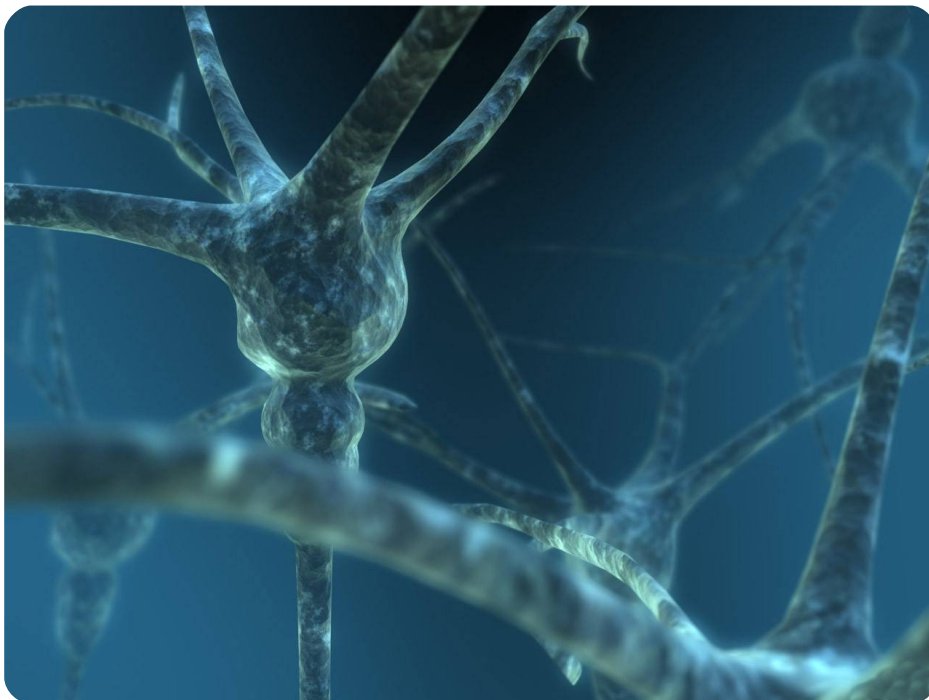
Итак, из полей класса нейронной сети видно, что он также не представляет собой ничего сложного. Теперь немного уделим внимание методам, это поможет понять логику их работы. Сначала рассмотрим методы в секции **Private**. Это методы для внутреннего пользования, то есть, они поддерживают работу класса, и не должны вызываться извне – это может нарушить нормальную работу нейронной сети.

Сразу возникает вопрос – почему **RunNeron** есть функция? Она возвращает **True** (истина), если нейрон может выполнить свою работу (и тогда он ее выполнит). Сделано так специально с расчетом на будущую модификацию сети. Например, в данном классе не реализовано сохранение и чтение нейронной сети во внешний файл. Представьте себе ситуацию, что Вы загрузили поврежденный файл или же во время работы изменили нейронную сеть. Тогда возможно нарушение ее внутренней структуре, что может привести к тому, что дендрит будет ссылаться на нейрон (или на элемент одной из таблицы), которого не существует в данной нейронной сети.

Updating должен выполняться (и выполняется) сразу после выполнения всех нейронов в сети – это процесс обновления сигналов на выходах нейронов. Она переносит значение из **Akson2** в **Akson** в каждом нейроне нейронной сети. Функция **RunNeron** помещает результат работы нейрона именно в **Akson2**, что дает возможность другим нейронам получать старые данные и сохраняет целостность и корректность модели.

OutPutting – отвечает за сбор информации в таблицу выходных значений. После обработки всех нейронов осуществляется обновление их выходов (посредством **Updating**). Затем **OutPutting** читает информацию из объекта нейронной сети, руководствуясь информацией из **TableOut2**, и переносит результат в соответствующий элемент таблицы выходных данных.

Большинство остальных методов класса ориентированы на получение данных об объектах нейронной сети, а также на внесение в нее изменений, в том числе и внесение информации о задаче (информация в таблице входных данных). Пожалуй, интерес представляет только Run – выполнение нейронной сети. На самом деле это группа операций. А именно – выполнение всех нейронов, обновление состояний выходов нейронов и получение результатов в таблицу выходных данных.



Заключение

В данной статье описана простая модель нейронной сети, при доработке которой можно добиться весьма неплохих результатов. Здесь-бы хотелось отразить те моменты, с помощью которых данный пример можно развить до вполне конкурентоспособного и, возможно, даже коммерческого варианта, а именно:

1. Сохранение и чтение нейронной сети в файл. В случае если Вы только отрабатываете свои навыки, то это может быть простенький текстовый формат по типу CSV (где все поля в строке разделяются символами табуляции). Простота устройства сети позволяет легко реализовать процедуру сохранения информации с использованием буфера, например, через список строк (такой как TStringList). Процесс чтения, как правило, немного сложнее из-за необходимости контроля целостности внутренней структуры сети.

** Комментарий автора.

Напоследок – не совсем удачное название класса выбрано специально, чтобы избежать возможного конфликта имен. Дело в том, что существует ряд классов и компонентов, имеющих в названии Net. Да и самописные инструменты, связанные с работой через сеть (не нейронную) обычно называются аналогично, а длинное имя элементарно лень писать. Все остальные вопросы можно уточнить в проекте, в котором реализована данная модель нейронной сети.

2. Введение отладочных механизмов. Несмотря на то, что речь о них в статье шла, в примере они не реализованы. Здесь необходимо уделить внимание оптимизации алгоритма по скорости. Сложности возникать не должно, что и зачем в статье описано. Главное помнить, что работа сети есть большое количество итеративных (циклических) процессов, а на это требуется время, что может потребовать принудительного выделения ресурсов, например, с помощью Application.ProcessMessages, либо аналогичных средств.

3. Введение дополнительных удобств использования. В примере данные возвращаются побитно, но можно группировать информацию и возвращать массивы, либо упаковывать в байты и анализировать их далее.

4. Оформление проекта в качестве динамической библиотеки, что позволит подключать ее и для других языков программирования, и вообще будет способствовать распространению.

5. Написание дополнительных инструментов. В частности, можно написать генератор нейронных сетей, позволяющий создавать нейронные сети с некоторыми заданными характеристиками. Тогда проект может уже подгружать и работать с готовыми нейронными сетями.

6. Написать распределенную версию нейронной сети. Это позволит запускать ее на нескольких компьютерах и производить внутренний обмен посредством локальной и глобальной сети. Что в свою очередь дает решать более глобальные и ресурсоемкие задачи.

7. Оформление подробной документации. Один из важнейших вопросов, к которому большинство программистов, обычно относятся халатно.

Источники. Что почитать

- Раздел википедии <http://ru.wikipedia.org/wiki/Нейросети>
- Введение в искусственные нейронные сети <http://www.osp.ru/os/1997/04/179189>
- Нейронные сети <http://www.statsoft.ru/home/textbook/modules/stneunet.html>
- Введение в теорию нейронных сетей <http://www.orc.ru/~stasson/neurox.html>
- Материал по нейронным сетям http://www.artkis.ru/neural_network.php
- Нейронные сети - математический аппарат <http://www.basegroup.ru/library/analysis/neural/math>
- Лекции по теории и приложениям искусственных нейронных сетей http://alife.narod.ru/lectures/neural/Neu_ch12.htm
- Основные понятия Нейронных Сетей <http://oasis.peterlink.ru/~dap/nneng/nnlinks>
- Нейронные сети: прогнозирование как задача распознавания образов <http://www.masters.donntu.edu.ua/2003/fvti/paukov/library/neurow.htm>

Здравствуйте. В этой статье я хочу рассмотреть создание движка динамического освещения с помощью графической библиотеки OpenGL. Писаться движок будет на Delphi, но это не мешает переписать его на любой другой язык, так как главное, рассматриваемое в статье, это алгоритмы...



Вадим Буренков

vadim_burenkov@mail.ru

Чтобы не тратить время на инициализацию OpenGL и избежать других проблем (например, с настройкой таймеров и рендера в текстуру) я буду использовать движок ZenGL **[1]**. Впрочем, от него нам многого не понадобится. Итак, приступим...

Инициализация OpenGL в ZenGL

Первым делом качаем ZenGL и создаем в нем простейшее приложение (вы можете найти его в папке LightEngine ресурсов статьи, там же вы найдете ZenGL):

```
program LightEngine;
uses
  zgl_main,
  zgl_screen,
  zgl_window,
  zgl_timers,
  zgl_textures,
  zgl_textures_jpg,
  zgl_sprite_2d,
  zgl_mouse,
  zgl_keyboard,
  zgl_utils;

var
  BackTex:zglPTex;      // текстура фона
  bTiles:zglTTiles2D;   // параметры тайлинга

procedure Init;
var n,j:integer;
begin
  // отключаем очищение буфера
  zgl_disable(COLOR_BUFFER_CLEAR );

  // Тут можно выполнять загрузку основных ресурсов

  // загрузка текстуры и настройка тайлинга
  BackTex:=tex_LoadFromFile( 'Back.jpg',0,TEX_DEFAULT_2D);
```

```
// параметры тайлов фона
bTiles.Count.X:=7;
bTiles.Count.Y:=5;
bTiles.Size.W:=128;
bTiles.Size.H:=128;
SetLength(bTiles.Tiles,7,5);
for n:=0 to 4 do
  for j:=0 to 6 do bTiles.Tiles[j,n]:=1;
end;

procedure Draw;
begin
  // Тут "рисует" что угодно :)
  tiles2d_Draw(BackTex,0,0,bTiles);      // отрисовка фона
end;

procedure Update;
begin
  // Тут выполняется обработка данных
  if key_Press( K_ESCAPE ) then zgl_Exit;
  // обновление клавиш
  key_ClearState;
  Mouse_ClearState;
end;

procedure Timer;
begin
  // Будем в заголовке показывать количество кадров в секунду
  wnd_SetCaption( 'LightEngine [ FPS: ' + u_IntToStr( zgl_Get(
SYS_FPS ) ) + ' ]' );
end;

procedure Quit;
begin
  // Тут выполняется очищение данных
end;

Begin
  // Создаем таймер с интервалом 1000мс.
  timer_Add( @Timer, 1000 );
  // Создаем таймер с интервалом 10мс.
```

```

timer_Add( @Update, 10 );

// Регистрируем процедуру, что выполнится сразу после
// инициализации ZenGL
zgl_Reg( SYS_LOAD, @Init );

// Регистрируем процедуру, где будет происходить рендер
zgl_Reg( SYS_DRAW, @Draw );

// Регистрируем процедуру, которая выполнится после завершения
// работы ZenGL
zgl_Reg( SYS_EXIT, @Quit );

// Устанавливаем заголовок окна
// Разрешаем курсор мыши
wnd_ShowCursor( TRUE );

// Указываем первоначальные настройки
scr_SetOptions( 800, 600, REFRESH_MAXIMUM, FALSE, FALSE );

// Инициализируем ZenGL
zgl_Init;

End.

```

При инициализации мы указываем процедуры в которых будут производиться различные действия (инициализация/обработка/очистение) а также параметры окна.

В инициализации загружается текстура и настраивается тайлинг (количество и размер настроен так, чтобы текстура закрывала весь экран). В обработке обновляются состояния мыши и клавиатуры, а также стоит проверка на нажатие **ESC**. Процедура очищения пока пуста, так как ресурсы движка очищаются самостоятельно.

Другие непонятные процедуры можно посмотреть в справке, которая находится в папке **doc** движка. Чтобы при компиляции не возникло проблем необходимо указать расположение модулей движка в **Project->Options->Directories/Conditionals->SearchPath**, а именно папки **zengl/src** и **zengl/src/PasZLib** (см. рис.1). Можно скомпилировать проект, увидеть вы должны следующее (см. рис.2).

Немного теории

Теперь перейдем к теории вопроса. В движке мы должны реализовать два типа – источники света и объекты, которые отбрасывают тени (см. рис.3):

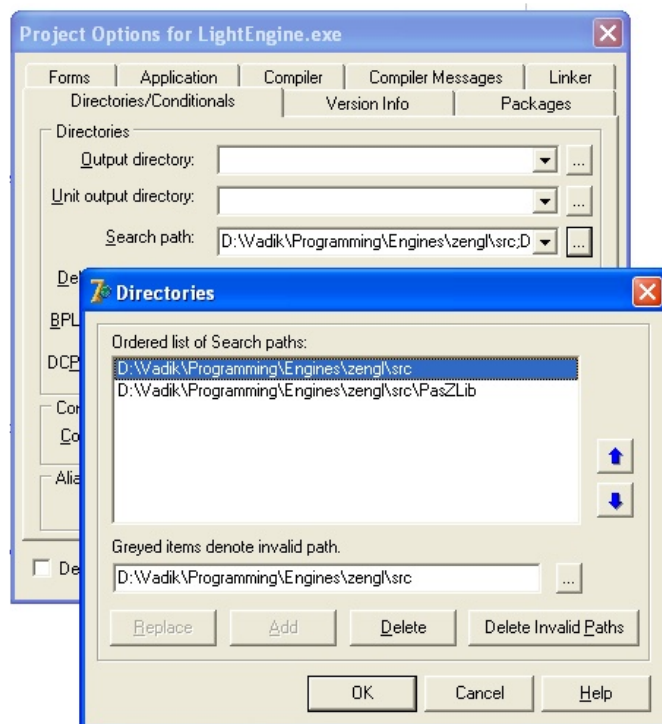


Рис. 1. Пути



Рис. 2. Тайлинг

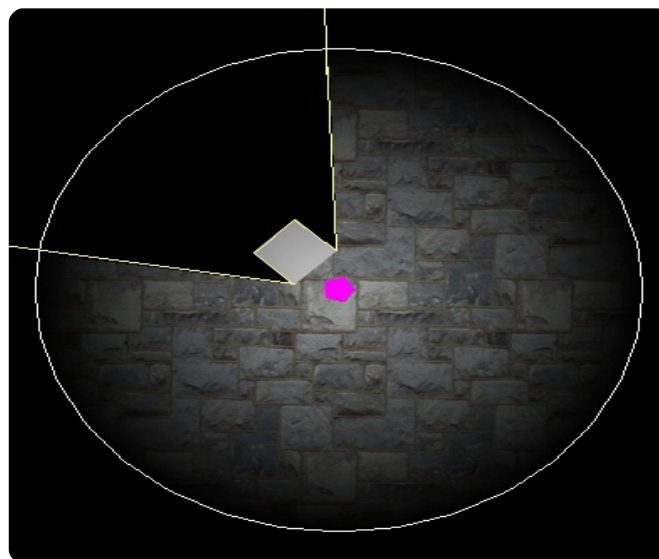


Рис. 3. Тень от объекта

Источник света обладает параметрами:

- положение
- радиус
- цвет
- интенсивность

Все объекты являются невыпуклыми многоугольниками. Они имеют:

- локальные координаты вершин
- мировые координаты вершин
- количество вершин
- положение
- угол поворота

Локальные координаты нужны, так как через положение и угол поворота объекта его можно разворачивать.

Для хранения данных об освещенности нам понадобятся два буфера размером в экран. Первый – альфа буфер. В него выводится круглый источник света (см. рис.4):

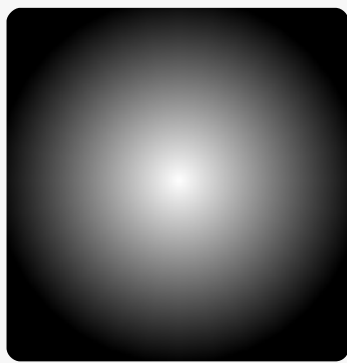


Рис. 4. Источник света

После этого альфа буфер рисуется во второй буфер – буфер аккумуляции. При этом используется аддитивный режим блендинга, то есть цвета смешиваются. В буфере мы получаем такую картинку (см. рис.5):



Рис. 5. Смешивание источников света

В нем светлые участки – там где свет, а темные – там где тьма. А теперь мы выводим буфер аккумуляции на экран с блендингом MULT. Получается так, что чем светлее цвет, тем он прозрачнее (см. рис.6):



Рис. 6. Рисование с блендингом MULT

Как же делаются тени от объектов? При выводе источника света в альфа буфер на него рисуется форма тени черным цветом. Получается что от круга света «отрезают» кусок (см. рис.7):

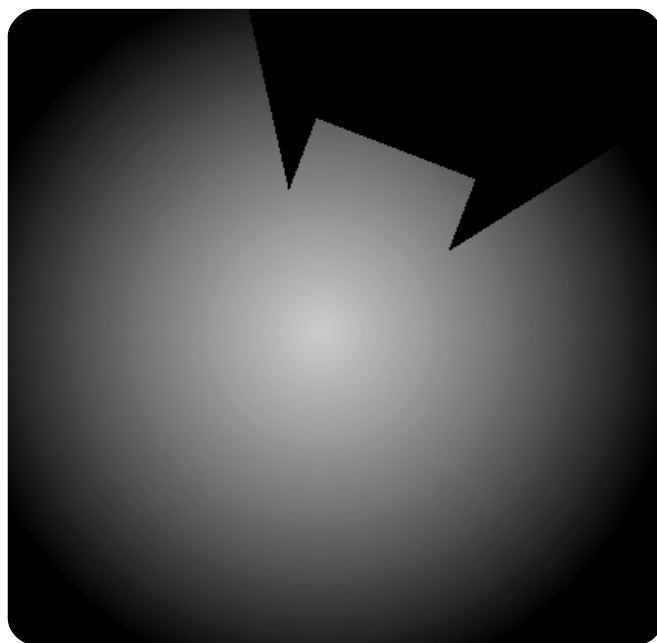


Рис. 7. Форма тени на свете

С помощью такого алгоритма получаются тени любой сложности, причем их количество, как и источников света с объектами неограниченно (см. рис.8).

Да будет свет!

Под следующий код сделаем модуль, который и будет отвечать за тени. Назовем его ZGLShadows.

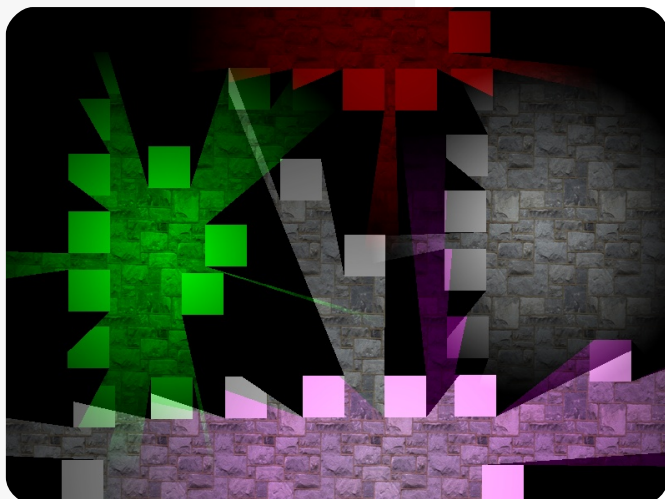


Рис. 8. Сцена с большим количеством теней

Напишем тип света:

```
type
  PLightSource=^TLightSource;
  TLightSource=record
    position:leVect;      // положение
    radius:single;        // радиус
    color:TColorRGB;      // цвет
    intensity:single;     // интенсивность света
    prev,next:PLightSource;
  end;
```

Все данные будут храниться в “prev-next” (двухсвязных) списках (см. рис.9):

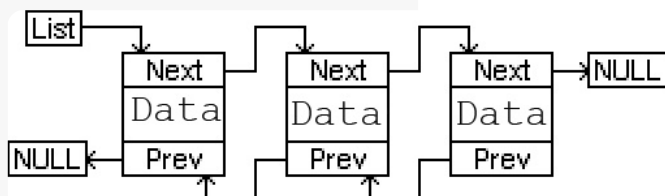


Рис. 9. Списки

Каждый элемент является звеном цепи и имеет указатели на предыдущее и следующее звено. Нам же нужно иметь первый элемент и длину цепи для управления списком:

```
Var
  // Источники света
  le_Lights:PLightSource;    // список
  le_NumLights:integer;      // количество
```

Более подробно о такой системе хранения данных можно почитать в Интернете. Как мы видим, у нас появились новые типы данных:

```
type
  TColorRGB=record
    r,g,b:single;
  end;
```

Данный тип нужен для хранения цвета в формате rgb, так как его использует OpenGL. ZenGL использует integer для хранения цветов (например \$FFFFFF соответствует 1,1,1 в RGB), поэтому нам может понадобится процедура для перевода цвета в RGB:

```
function IntToRGB(Color:Integer): TColorRGB;
begin
  Result.r := ((Color and $FF0000) shr 16) / 255;
  Result.g := ((Color and $FF00) shr 8) / 255;
  Result.b := (Color and $FF) / 255;
end;
```

Все координаты будут храниться в типе:

```
type
  leVect=record
    x,y:single;
  end;
```

Подробнее о нем будет написано позже, пока нам понадобится только формирование вектора по x и y:

```
function le_v(x,y:single):leVect;
begin
  result.x:=x;
  result.y:=y;
end;
```

Перейдем к процедурам управления источниками света:

```
Function le_CreateLightSource(p:leVect;
                               radius,intensity:single;
                               color:TColorRGB):PLightSource;

var t: PLightSource;

begin
  new(t);
  t.Next:= nil;
  t.Prev:= nil;

  t.position:=p;
```

```

t.radius:=radius;
t.intensity:=intensity;
t.color:=color;

t.Next:= le_Lights;
if le_Lights <> nil then le_Lights.Prev:= t;
le_Lights:= t;
Result:= le_Lights;
inc(le_NumLights);
end;

```

Функция создает источник света в памяти и возвращает указатель на него. Большую часть кода занимает работа со списками.

В следующей процедуре происходит рисование круга света как на рисунке 4. Он рисуется через `GL_TRIANGLE_FAN`. Первая точка в центре имеет цвет и интенсивность света, далее идут точки по радиусу окружности с нулевым цветом, благодаря чему мы имеем плавный переход цвета:

```

procedure le_DrawLightSource(t:PLightSource);
var angle:single;
begin
angle:=0;

glBegin(GL_TRIANGLE_FAN);
glColor4f(t.color.r, t.color.g, t.color.b, t.intensity);
glVertex2f(t.position.x,t.position.y);
glColor4f(0, 0, 0, 0 );
while angle<=Pi*2 do begin
glVertex2f( t.radius*cos(angle) + t.position.x,
            t.radius*sin(angle) + t.position.y);
angle:=angle+((PI*2)/le_numSubdivisions);
end;
glVertex2f(t.position.x+t.radius, t.position.y);
glEnd();
end;

```

Количество треугольников, из которого рисуется круг, задается константой:

```

const
le_numSubdivisions = 32;

```

Следующая процедура рассчитывает и рисует тень для объектов, но о ней я напишу позже:

```

procedure le_RenderShadowGeometry(t:PLightSource);

```

Далее напишем процедуру, которая освобождает память, занятую источником света:

```

procedure le_FreeLightSource(t:PLightSource);
var DelT: PLightSource;
begin
DelT:= t;
if t.Prev <> nil then t.Prev.Next := t.Next
else le_Lights:= t.Next;
if t.Next <> nil then t.Next.Prev := t.Prev;

Dispose(DelT);
dec(le_NumLights);
t:=nil
end;

```

Следующая процедура вспомогательная, она обрабатывает каждый источник света передаваемой в нее процедурой:

```

procedure le_EachLightSource(p:le_proc);
var t, tNext: PLightSource;
begin
t:= le_Lights;
while t <> nil do
begin
tNext:= t.Next;
p(t);
t:= tNext;
end;
end;

```

`le_proc` – тип процедуры:

```

type
le_proc=procedure(d:Pointer);

```

Например, данная строчка нарисует все источники света:

```

le_EachLightSource(@le_DrawLightSource);

```

Хочу заметить, что при попытке передать в `le_EachLightSource` процедуру очищения возникнет ошибка, связанная с памятью, поэтому для очищения всех источников света напишем отдельную процедуру:

```

procedure le_FreeAllLightSources;
var t, tNext: PLightSource;
begin
    t:= le_Lights;
    while t <> nil do begin
        tNext:= t.Next;
        le_FreeLightSource(t);
        t:= tNext;
    end;
end;

```

Заставим это работать

С типом света мы управились, теперь надо заставить его работать. Объявим следующие переменные:

```

var // Движок
le_AlphaBuffer: zglPRenderTarget; // буфер для света
le_AccBuffer: zglPRenderTarget; // буфер для сложения
// изображений света
le_DarkColor: integer; // цвет тени

```

В этой процедуре инициализируются буферы для рендеринга. К проекту нужно также подключить модули:

- `zgl_textures` – создание текстуры-буфера
- `zgl_render_target` – управление рендерингом
- `zgl_primitives_2d` – очищение рендер target. Хотя можно было бы просто рисовать QUAD на OpenGL.
- `zgl_fx` – процедуры управления блендингом
- `zgl_sprite_2d` – рисование текстур

Напишем процедуру инициализации буферов:

```

procedure le_InitLightEngine(DarkColor:Integer);
begin
    le_DarkColor:=DarkColor;
    // инициализация буферов
    le_AlphaBuffer:=rtarget_Add( RT_TYPE_FB0, tex_CreateZero(
        800,600, 0, TEX_DEFAULT_2D ), RT_FULL_SCREEN );
    le_AccBuffer :=rtarget_Add( RT_TYPE_FB0, tex_CreateZero(
        800,600, 0, TEX_DEFAULT_2D ), RT_FULL_SCREEN );
end;

```

`le_DarkColor` – переменная, которая отвечает за освещенность. К ее смыслу и принципу работы я еще вернусь. Обобщающая процедура полной обработки света:

```

procedure le_RenderLight(t:PLightSource);
begin
    rtarget_Set( le_AlphaBuffer ); // начинаем рендер в буфер
    pr2d_Rect(0,0,800,600, 0,255,PR2D_FILL); // очищаем черным
    // цветом
    le_DrawLightSource(t); // отрисовка источника света
    rtarget_Set( nil );

    rtarget_Set( le_AccBuffer ); // отрисовка полученного
    // изобра-я в буфер аккумуляции
    fx_SetBlendMode(FX_BLEND_ADD); // при отрисовке используем
    // блендинг для сложения
    // интенсивностей источ-в света
    ssprite2d_Draw( le_AlphaBuffer.Surface, 0,0,800,600,0);
    fx_SetBlendMode(FX_BLEND_NORMAL);
    rtarget_Set( nil );
end;

```

И завершающая процедура – вывод буфера с использованием `MULT` блендинга:

```

procedure le_FinishRender;
begin
    // выводим полученные тени и свет на экран с использованием
    // блендинга mult (чем светлее изображение тем прозрачней)
    fx_SetBlendMode(FX_BLEND_MULT);
    ssprite2d_Draw( le_AccBuffer.Surface, 0,0,800,600,0);
    fx_SetBlendMode(FX_BLEND_NORMAL);

    // очищаем буфер для следующего кадра (цветом le_DarkColor!)
    rtarget_Set( le_AccBuffer );
    pr2d_Rect(0,0,800,600, le_DarkColor,255,PR2D_FILL);
    rtarget_Set( nil );
end;

```

Хочу заметить, что очищение `le_AccBuffer` проводится цветом `le_DarkColor`. Следовательно, чем светлее этот цвет тем прозрачнее тени (см. рис.10). На изображении видно, что справа фон просвечивается даже там, где света нет, так как `le_DarkColor` не черный, а серый (\$1f1f1f).

Все, система света написана. Правда, пока без

теней от объектов. Теперь проверим ее в действии. Добавим переменную под управляемый свет:

```
var
UserLight:PLightSource; // указатель на управляемый свет
```

Создадим его, и еще 4 света в **Init**:

```
le_InitLightEngine(0); // инициализируем световой движок

// загружаем источники света
le_CreateLightSource
(le_v(520,550),400,0.3,IntToRGB($FF00FF));
le_CreateLightSource
(le_v(470,50),250,0.5,IntToRGB($FF0000));
le_CreateLightSource
(le_v(200,300),270,1,IntToRGB($00FF00));
le_CreateLightSource
(le_v(640,270),300,0.8,IntToRGB($FFFFFF));
UserLight:=le_CreateLightSource(le_v(0 ,
0),100,0.8,IntToRGB($FFFFFF));
```

Теперь в **Draw** напишем рисование теней:

```
// обработка всех источников света
le_EachLightSource(@le_RenderLight);

// отрисовка теней
le_FinishRender;
```

В **Update** привяжем **UserLight** к мышке, сделаем изменение размера при нажатии на кнопки мыши и цвета при нажатии на колесико:

```
if mouse_Down( M_BLEFT ) then
    UserLight.radius:=UserLight.radius+3;
if mouse_Down( M_BRIGHT ) then
    if UserLight.radius>0 then
        UserLight.radius:=UserLight.radius-3;

if mouse_Click( M_BMIDDLE ) then begin
    UserLight.color.r:=random(100)/100;
    UserLight.color.g:=random(100)/100;
    UserLight.color.b:=random(100)/100;
end;
```

И наконец, в **Quit** очищаем источники света:

```
le_EachLightSource(@le_FreeLightSource);
```

Запускаем, любимся результатом (см. рис.11).

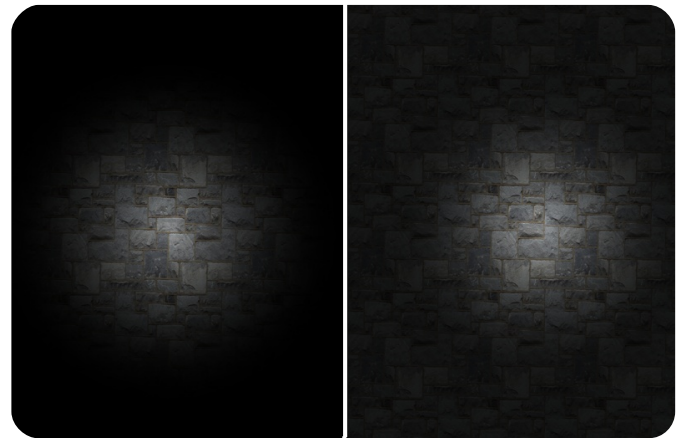


Рис. 10. Различный DarkColor



Рис. 11. Результат

Заключение

В следующей части статьи я рассмотрю создание теней от объектов, а также оптимизирую код, чтобы добиться большей производительности. Весь исходный код проекта приложен к журналу «ПРОграммист. Пятый выпуск».

Продолжение следует...

Ресурсы

- Страница разработчика ZenGL <http://andru-kun.inf.ua/zengl.html>
- Михалкович С.С. Основы программирования. Второй семестр 08-09, 2 часть

В этой статье я расскажу, как устроен MP3 файл, и покажу, как можно работать с ним в ваших программах. Мы попробуем извлечь информацию о файле, такую как длину трека, его битрейт и частоту дискретизации. Воспроизведение звука мы рассматривать не будем, это отдельная, и я думаю намного более сложная тема...



Александр Терлецкий

by [mutabor altair.79@mail.ru](mailto:mutabor.altair.79@mail.ru)

Немного про формат

MP3 расшифровывается как MPEG 1 Layer 3, т.е. MPEG версии 1, третья редакция, или как-то в этом роде. Нам важно понять, что бывает еще MPEG 2, а Layer не обязательно может быть третьим. Что такое MPEG, почему Layer 3, чем отличается MPEG1 от MPEG2, и прочие подобные вопросы я рассматривать не буду, т.к. это само по себе тянет на отдельную статью. MP3 это сжатый формат, сжатие достигается за счет убирания из исходного звука частот заведомо не слышимых человеком, ну и еще за счет алгоритма сжатия какого-нибудь (это я тоже не буду здесь рассматривать). Именно поэтому сжимать архиваторами MP3 файлы не получается (вернее получается, но результат не впечатляет), они уже сжатые. Внутри файл состоит из фреймов. Заголовка у MP3 файла нет, зато у каждого фрейма есть свой заголовок, с ним то мы и будем в основном работать. Фрейм можно рассматривать, как некий дискретный кусок звукового потока.

Теги

Помимо фреймов, в файле могут быть один или несколько ID3 тегов. ID3 – это теги специально разработанные для формата MP3, т.к. он сам не содержит никакой описательной части. Теги бывают разных версий, чаще всего это или ID3v1.x или ID3v2.x. Теги первой версии находятся в последних 128 байтах файла, начинаются с символов TAG (такой тег может занимать и более чем 128 байт, но это редко, это усовершенствованная первая версия ID3). Теги второй версии могут находиться в любой части файла, но чаще они располагаются в начале файла, и начинаются с символов ID3. Теги второй версии намного более расширенные, чем теги первой версии, в них нет ограничений на длину

полей с описанием трека, количество доступных полей намного больше, можно использовать Юникод, тег может содержать в себе изображение. Длина тега второй версии не фиксированная, и определяется по заголовку тега (в отличие от MP3 файла, у ID3v2 тега есть заголовок). Наличие тегов в файле не является обязательным, и их может не быть совсем, а могут быть оба, и тег первой версии в конце файла, и второй в начале, это делается в целях совместимости с большим количеством плееров. Часто возникает проблема с русскими символами в тегах в плеерах мобильных телефонах, я думаю это связано с Java машиной в телефоне, дело в том, что она поддерживает строки в формате UTF-8, а русские теги часто имеют кодировку Win-1251. Чтобы избежать «козябриков» на этих устройствах, нужно сохранять теги в Юникоде.

Теперь немного о том, как читать теги программно. Я не буду подробно освещать эту тему здесь, скажу лишь, что существуют библиотеки, компоненты для их чтения, также в сети доступна спецификация на эти теги, так что можно и самому написать их обработку, если есть желание. Звуковые движки, например тот же BASS, тоже умеют читать теги. Они кстати умеют и все остальное, о чем я буду писать ниже, и если ваша задача – получить информацию о файле, и вам не интересно как он устроен, можете в принципе дальше не читать, так как через интерфейс движка это сделать намного легче. Если вам звук нужно еще и воспроизводить, то этот способ даже лучше, зачем ковыряться в спецификациях, если есть удобный инструмент. Но бывают случаи, когда звук воспроизводить не надо, а нужна только информация о файле, и тогда лучше подключить легкий модуль, чем таскать движок за программой.

Битрейт

В этой статье я затрону только те

характеристики, которые мы будем читать из файла, остальную общую информацию об MP3 можно без проблем найти в Интернете, ее достаточно, в отличие от более специализированной информации о формате.

Как вы все, наверное, знаете, MP3 файл может иметь различный битрейт, это кол-во бит выделенных на кодирование звука в единицу времени. Понятно, что чем он выше, тем качество звука лучше, и размер файла соответственно тоже больше. Значение битрейта в MP3 может находиться в пределах от 8 до 320 кбит/с. Полный список смотрите в Рис 2. Битрейт может быть постоянным (constant) и переменным (variable), это обозначается аббревиатурами CBR и VBR соответственно. Переменный битрейт позволяет снизить размер файла, не снижая качества. Это достигается за счет того, что на участках, где это не требуется, например тишина в начале трека, используется меньшее количество бит для кодирования. На уровне структуры файла это выражается в том, что один фрейм может иметь битрейт, например 128, а следующий может иметь уже 192, и т.д. В каждом отдельном фрейме битрейт имеет значение кратное степени двойки, соответствующее спецификации (см. рисунок 2). В целом по файлу, битрейт, в случае если он постоянный, не будет отличаться от значения битрейта первого фрейма, таким образом, нам достаточно взять информацию из первого фрейма, и мы имеем информацию о файле. В случае если битрейт переменный, то все усложняется, нам недостаточно одного фрейма, чтобы делать выводы об общем битрейте файла.

Частота дискретизации

Вторая характеристика, которую мы рассмотрим, это частота дискретизации или Sample Rate. Это частота, с которой при кодировании звука снимаются замеры с источника звука. Например, если частота у нас 44100 Гц, то это значит, что столько раз в секунду снимаются и сохраняются значения оригинального, аналогового звука. По

сути, оцифровка. Можно конечно и уже оцифрованный звук переписать с другим сэмпл-рейтом, но качество можно только понизить, повысить уже вряд ли удастся. Как и в случае с битрейтом, от частоты дискретизации напрямую зависит качество звука. Частота дискретизации в MP3 не может варьироваться, и всегда постоянна для всего файла.

Длина

У любой музыкальной композиции или любой другой звуковой записи есть такая характеристика, как длина. Упомяну я ее отдельно для того, чтобы обрадовать вас, что так как у MP3 файла общего заголовка нет, то и готовые сведения о длине нам взять в принципе неоткуда (ID3 тег не в счет, в нем может быть длина трека, а может и не быть, а может и самого тега не быть). Поэтому длину нам придется высчитывать самостоятельно. На этом с характеристиками закончим и перейдем к разбору фреймов.

Фреймы

Один фрейм состоит из двух блоков (последовательностей бит) – заголовок и блока данных (см. рисунок 1). Заголовок представляет собой последовательность из 32 бит (4 байта), в которых описываются все необходимые параметры звука, а также параметры самого фрейма (например, его длина). В блоке данных находятся непосредственно звуковые данные. Рассмотрим заголовок подробнее (см. рисунок 2). Вначале идут 12 бит сигнатуры (Sync), по этим битам нужно искать фреймы, все они установлены в единицу. Затем идут еще три бита: версия, слой и защита от ошибок. Я сделал вывод, что если принять по умолчанию что мы работаем с MP3 файлом, а не с какой-нибудь другой версией MPEG, то можно брать первые два байта как сигнатуру, в этом случае они могут иметь всего две комбинации: FF FA или FF FB. Мне показалось так удобнее искать фреймы, и именно так

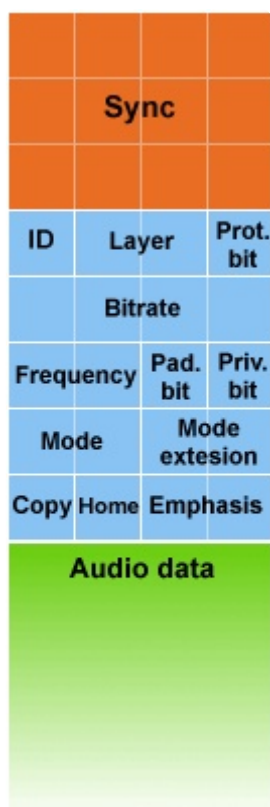
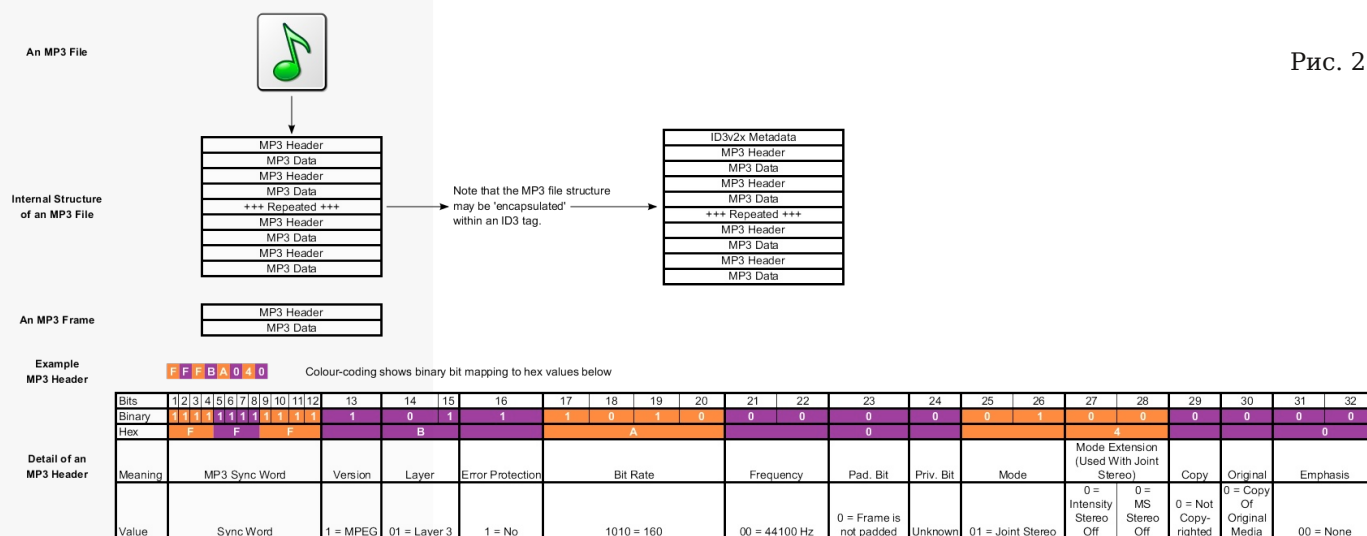


Рис. 1. Структура



я реализовал это в коде. Через третий байт мы пока перескочим, и я вкратце расскажу о четвертом. В нем есть такая интересная штука как режим стерео. Дело в том, что MP3 имеет еще один способ уменьшить вес файла, не ухудшив качество. Это режим Joint Stereo (объединенное стерео). Что же это такое? Пройдемся по порядку. Режим моно, это, как вы знаете, когда всего один звуковой канал. Стерео - это независимые левый и правый каналы. Для хранения стерео данных, необходимо ровно в два раза больше места, чем для моно. Joint Stereo же позволяет хранить два независимых канала, при этом, занимая меньше места, это достигается за счет "умной" паковки во время сжатия. Если выбран этот режим, и в данном сэмпле звука левый и правый каналы не отличаются, то кодер сохраняет только один из них, когда же каналы отличаются, то они сохраняются оба. Это в своем роде стерео по требованию, оно используется там, где это реально нужно, а где не нужно, место экономится. В четвертом байте хранится еще ряд параметров, не буду на них останавливаться, кому интересно, смотрите на рисунке, там все подписано, обычно они интереса не представляют.

Самый значимый для нас это 3-й байт заголовка. В нем содержится информация о битрейте, частоте дискретизации, установлен или нет Pad бит (определяет наличие добавочного байта в фрейме), все это вместе взятое позволяет нам высчитать размер этого фрейма. Размер фрейма высчитывается по формуле 1:

$$144 * \text{BitRate} / \text{SampleRate} + \text{Pad}; \quad (1)$$

Если Pad бит равен единице то и в формулу подставляем единицу, если нулю - подставляем нуль. Битрейт и частоту подставляем в их полном виде, без округлений, битрейт в битах, а частоту в герцах. В файлах к этой статье вы найдете пример извлечения нужной информации из заголовка фрейма и реализацию этой формулы на языке Delphi. Чтобы что-то извлечь из заголовка, его нужно сначала найти, это тоже там есть. На самом деле достаточно найти первый фрейм, позицию каждого следующего мы уже будем знать, прибавляя к позиции текущего фрейма его длину.

Теперь поговорим о том, как найти переменный битрейт и длину трека. С постоянным битрейтом все ясно, он одинаковый для всего файла, и его можно взять из первого фрейма. С переменным не так. Во-первых, нигде не написано что он переменный, и чтобы это определить, нужно прочесть больше чем один фрейм. Я пошел самым простым путем, и читаю два первых фрейма. Если битрейт у них одинаковый я считаю что битрейт постоянный, если разный то переменный. Это скорее всего не точно, ведь переменным он может стать и после второго фрейма. Однозначных рекомендаций по этому вопросу я не встречал, возможно, есть соглашение, что если битрейт переменный, обязательно кодировать первые два фрейма с различным битрейтом, я бы так и сделал на месте разработчиков, но это только мои предположения. Если у вас есть более точная информация на этот счет, оставляйте комментарии к статье на сайте журнала, интересно будет почитать. Понятно, что в случае

переменного битрейта необходимо как-то высчитать его среднее значение по всем фреймам. Оставляю это вам, у меня в коде в этом месте стоит заглушка, т.е. просто в случае переменного битрейта, в качестве его значения присваивается ноль.

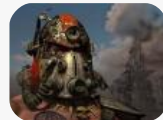
Заключение

Теперь про длину трека. Ее можно высчитать, поделив размер файла (за вычетом тегов и прочего мусора, нужны только фреймы, не совсем ясно только, включать заголовки или нет) на битрейт, если он постоянный. Таким образом, мы получим длину трека в секундах. Если битрейт переменный, этот способ не подходит (хотя можно на усредненный битрейт поделить), тогда мы можем задействовать сэмпл-рейт, он у нас постоянный для всего файла, и представляет собой кол-во сэмплов в секунду. Если предположить что сэмпл и фрейм это одно и то же, то можно узнать количество секунд в треке, посчитав все фреймы. Я не пробовал ни тот, ни другой способ, не буду портить вам удовольствие и позволю самим попробовать высчитать длину трека. Если что интересное получится, пишите в комментариях. Опять же, можно переменный битрейт считать от обратного, найти по сэмпл-рейту длину, и потом уже одним действием найти средний битрейт.

Все исходные коды, упомянутые в статье, приложены непосредственно к журналу «ПРОграммист. Пятый выпуск».

*На этом закончим, спасибо за внимание,
читайте наш журнал.*

Итак, сегодня мы с вами продолжим наш материал по промышленной автоматизации, рассмотрим реализацию алгоритма энкодера на ПЛИС, особенности прошивки модулей Faswell UNIOxx-5 и создадим тестовую утилиту визуализации состояния шифратора приращений.



Продолжение. Начало цикла смотрите в четвертом выпуске журнала...

Сергей Бадло

by **raxp** <http://raxp.radioliga.com>

Согласно требований из прошлого материала [1], приведем реализацию алгоритма энкодера на ПЛИС в среде Xilinx Foundation 3.1i [2] (см. рисунки 2...8).

Каждая микросхема FPGA обрабатывает сигналы от двух датчиков. По каждому четвертому сигналу чтения READ и нулевому адресу (битовая последовательность A0...A2) передаются: младшие 8- бит 11-ти разрядной последовательности, содержащей информацию о

текущем количестве импульсов поступивших с канала А (угловом положении). По первому адресу (A0...A2) передаются (с младшего по старший бит D00...D07): старшие 3- бита 11-ти разрядной информации о текущем количестве импульсов поступивших с канала А, признак отказа канала А (в течении длительности стробирующего импульса (одного оборота)), признак отказа канала В (в течении длительности стробирующего импульса (одного оборота)), признак направ-



Рис. 1. Тестируемый пациент

ления вращения (определяется с началом стробирующего импульса по приходу сигналов с

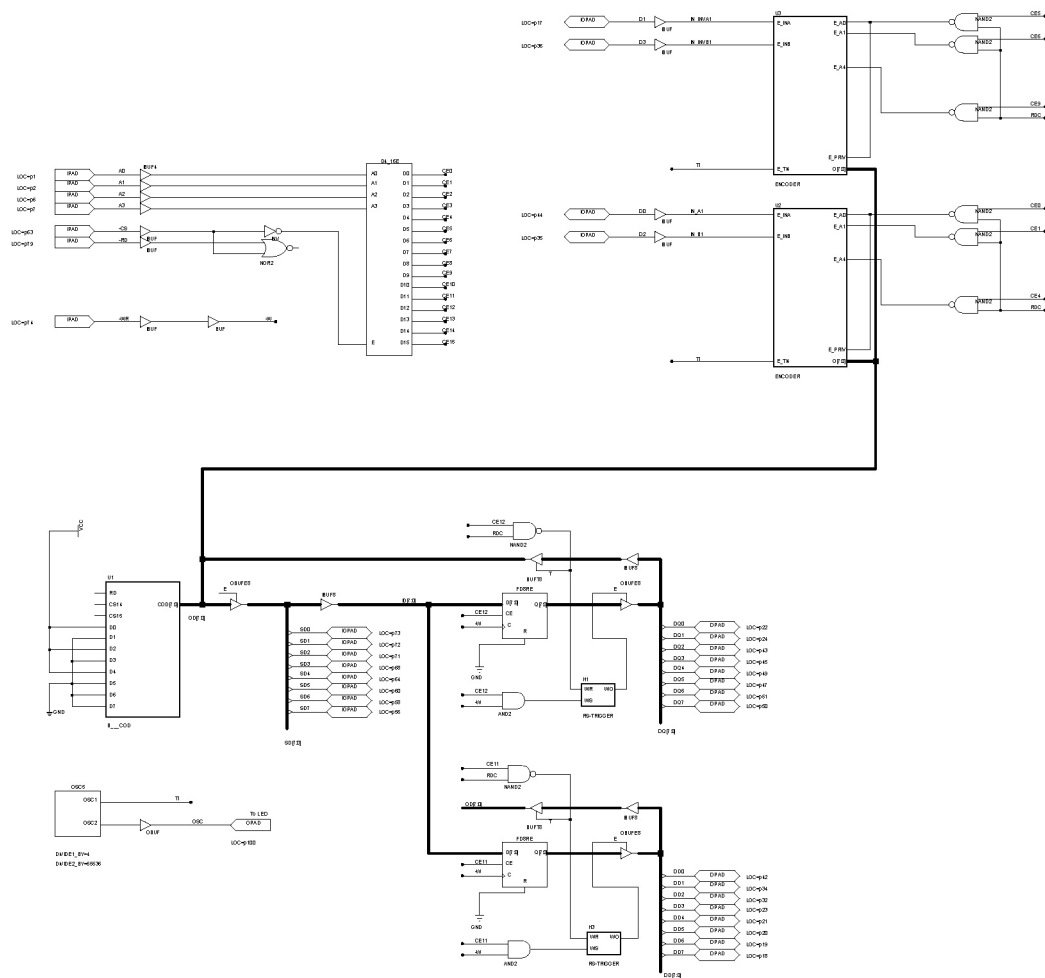


Рис. 2. Общая схема энкодера (дешифратор адреса, формирователь кода схемы, генератор ТИ, 2 канала энкодера, 16 каналов дискретного ввода-вывода)

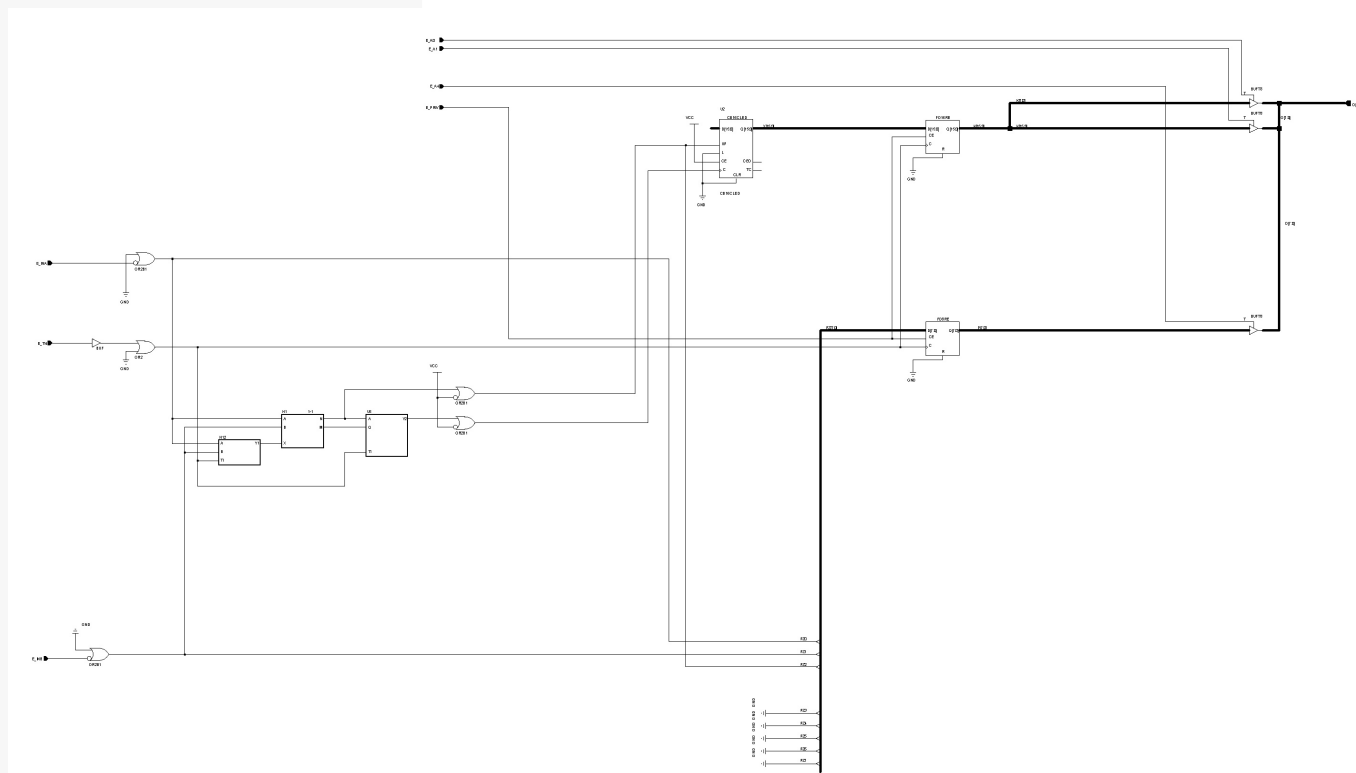


Рис. 3. Схема одного канала энкодера (антидребезг, задатчик идентификатора схемы, реверсивный счетчик импульсов, узел формирования выходного пакета данных)

канала А и В (в течении одного оборота)), признак отказа канала строба О. По второму и третьему адресу (A0...A2) передаются (с младшего по старший бит DO0...DO7) 16- бит информации о количестве оборотов. По четвертому и пятому адресу (A0...A2) передаются (с младшего по старший бит DO0...DO7) 16- бит информации о количестве ТИ за длительность строба (относительной* скорости).

Каждая из схем энкодера (алгоритм) осуществляет:

- измерение относительного углового положения датчика (по количеству импульсов поступивших с канала А);
- индикацию в соответствующем бите признака наличия импульсов (исправность А, В, Q);
- подсчет числа оборотов (количества импульсов строба), максимальное ограничено величиной 215, далее осуществляется сброс, и счет начинается заново;
- измерение относительной скорости*, т.е. подсчета количества ТИ, поступивших за длительность строба (погрешность ограничена длительностью ТИ и нестабильностью тактового генератора).

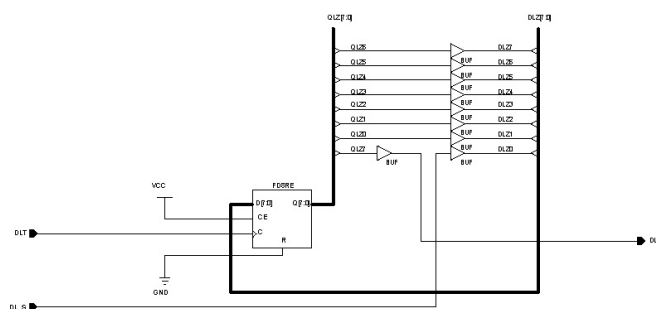


Рис. 4. Модуль задержки на 8 тактов встроенного генератора

Информация о текущей скорости определяется сразу же по истечении импульса строба, т.е. после первого оборота и далее по каждому последующему стробу.

* Комментарий автора.

Номинальная (фактическая) скорость определяется программным методом путем вычисления:

$$V = 4 * 2048 * N_{ти} * T_{ти}; \quad (1)$$

где: 4 – коэффициент, учитывающий соотношение длительности строба и длительности канальных (информационных импульсов); 2048 – максимальное количество импульсов по любому из каналов; V – фактическая скорость, об/сек; N_{ти} – количество ТИ за строб; T_{ти} – период ТИ, поступающих с внешнего генератора (равен 20 нс), сек.

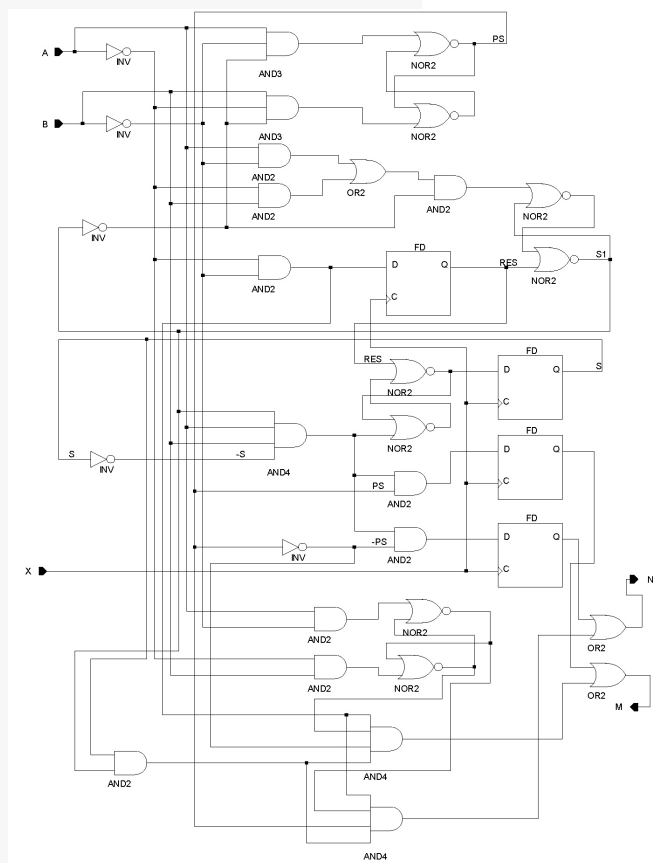


Рис. 5. Схема определения направления вращения ротора датчика PDF

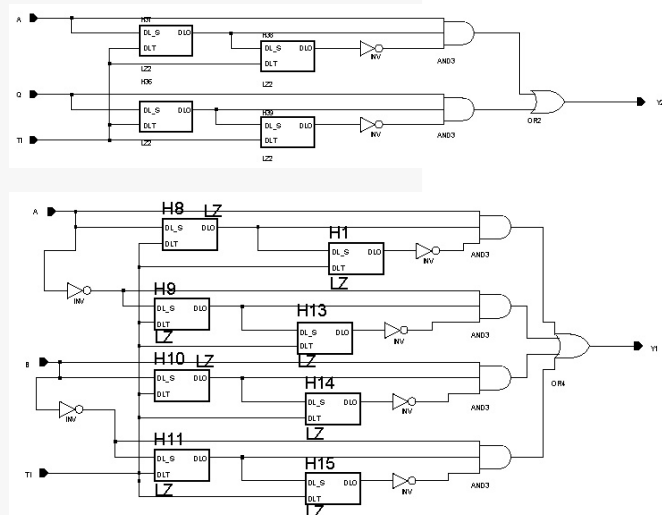


Рис. 6. Схемы нарезки сигналов А/В

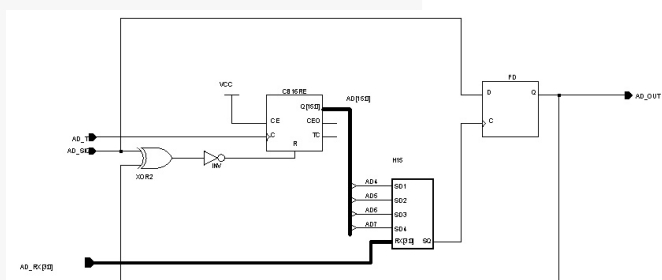


Рис. 7. Схема регулирования (СР) ширины ЗИ

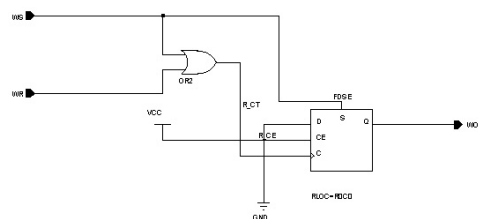


Рис. 9. Реализация RS-триггера

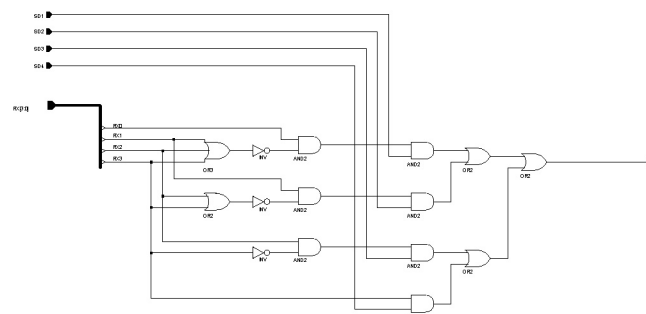


Рис. 8. Внутренняя схема узла адресации СР

Запустим наш проект на компиляцию. После проверки схемы, среда создаст битовый файл прошивки (файлы `bitstream` фирмы XILINX), который нужно прошить в соответствующую матрицу FPGA. Как это сделать?

Программирование платы UNIOxx-5

Обычно, для загрузки конфигурационной информации в ПЛИС используется JTAG интерфейс [3]. При этом последовательность действий следующая. Откройте приложение «JTAG Programmer» пакета Xilinx Foundation Series 3.1i. В меню «Edit» выберите пункт «Add Device», после чего появиться вспомогательное окно, в котором необходимо отыскать наш файл прошивки и открыть его. В меню «Operations» выберите пункт «Program». В появившемся окне нажмите кнопку «OK». После удачной загрузки, должна появиться фраза «Programming completed successfully». На этом программирование завершено (см. рисунок 10):

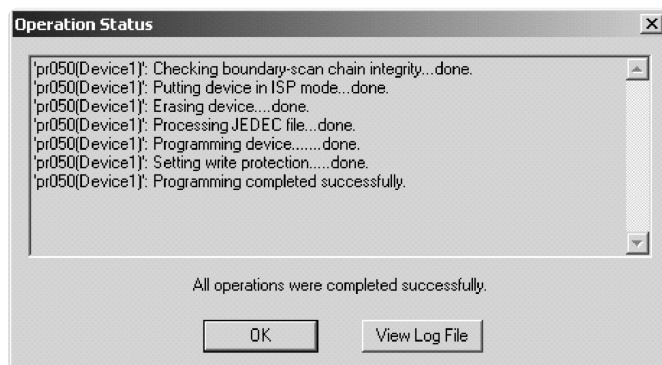


Рис. 10. Загрузка прошивки через JTAG

Однако этот способ не лишен недостатков: требуется наличие JTAG программатора практически все время «под рукой».

Второй способ, получивший название ISP (In System Programming), более удобен, так как не требует извлечения микросхемы конфигурационного ПЗУ (или самой ПЛИС) из платы или наличия дополнительного разъема на плате под ПЛИС. Программирование каждой матрицы FPGA1...FPGA4 модуля UNIOxx-5 осуществляется с помощью программ: <isp.exe> (загрузка схемы с записью в EEPROM), <isl.exe> (загрузка схемы без записи в EEPROM). Программы позволяют осуществить загрузку файлов схем *.bit матриц FPGA. Рассмотрим на примере...

Для записи схемы и загрузки в модуль необходимо запустить программу <isp.exe> с указанием Базового Адреса (Hex) модуля и кодов схем матриц FPGA, например (см. рисунок 11):

запись схем b21,b21,b21,b21 в модуль UNIOxx:

```
isp.exe 110 b21 b21 b21 b21
|   |   |   |   |
|   |   |   |   +----- Код схемы матрицы FPGA4
|   |   |   +----- Код схемы матрицы FPGA3
|   |   +----- Код схемы матрицы FPGA2
|   +----- Код схемы матрицы FPGA1
+----- Базовый Адрес 110h
```

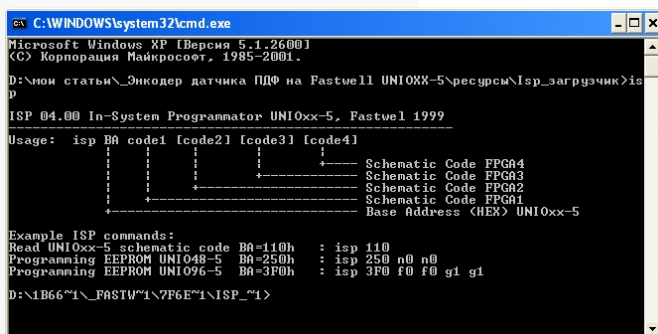


Рис. 11. Окно командной строки.
Запуск утилиты ISP

Питание и установка

Модуль с прошивкой энкодера устанавливается в слот (ISA) промышленного компьютера и запитан от напряжения $\pm 5V$. При этом требуется обеспечить стабильность питающего напряжения

с пульсациями не более 200 мВ. Выходной сигнал – однополярное напряжение (цифровая последовательность) с током нагрузки до 10 мА. Максимальная длина шлейфа, соединяющая выход модуля энкодера (платы UNIOxx-5) к входному разъему до 1.5 м. Проверка прошитой платы UNIOxx-5 производится на рабочем месте следующим образом. Земли шасси, платы UNIOxx-5, осциллографа, генератора импульсов и генератора сигналов высокочастотного должны быть объединены (заземлены) в одной точке проводником минимальной длины сечением не менее 2 мм² (см. рисунок 12):

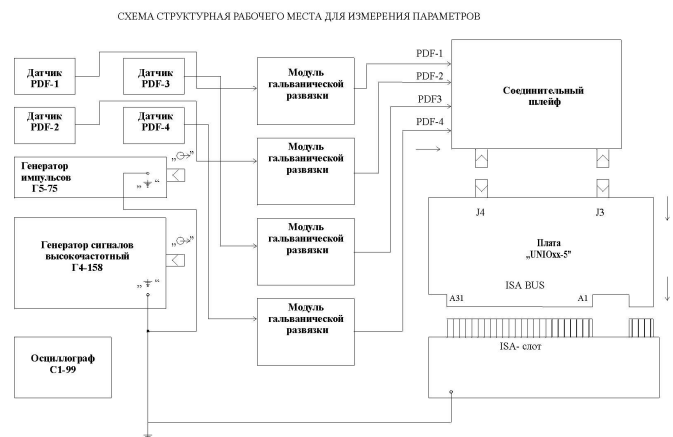


Рис. 12. Схема проверки платы UNIOxx-5 с прошивкой энкодера

Плата UNIOxx-5 при выключенном питании шасси вставляется в ISA слот. С помощью соединительного шлейфа присоединяют выход датчиков с соответствующими входами модулей гальванической развязки или модулей связи, сигнал с которых подан на вход платы UNIOxx-5. Далее производится включение промышленного шасси и загрузка обслуживающей прошивки. Осциллографом контролируют наличие напряжения питания (+5 В) на плате модуля UNIOxx-5. С помощью осциллографа производится контроль ТИ с тактового генератора платы. Включив генератор импульсов с заранее выставленными выходными уровнями сигналов, подают импульсы по входу строба для каждой из матрицы модуля (поочередно), контролируя одновременно с помощью программы обслуживания (или осциллографом) наличие признака строба в последовательности (бит DO7 при втором адресе A0...A2).

Кроме того, тестирование прошивки энкодера желательно провести в условиях, приближенных

к «реальным». Для этого необходимо, подключить шифратор приращений через наш модуль связи, используя несколько десятков метров кабеля UTP 5-й категории обмотанного вокруг работающего частотного привода под нагрузкой. Для визуализации положения ротора и подсчета количества импульсов с датчика приращений, реализуем на языке С простейшую утилиту приема данных с платы UNIO по шине ISA...

Реализация тестовой утилиты приема данных с энкодера и визуализация

Прежде всего, на основе схемы (алгоритма) энкодера составим таблицу распределения его выходных сигналов (см. таблицу). Для упрощения написания программы можно воспользоваться библиотечными функциями языка С. В качестве примера рассмотрим фрагмент программы, осуществляющей тестовое считывание данных с контроллера и вывод графического отображения положения ротора датчика PDF. В начале программы необходимо определить адрес

считывания, согласно таблице (см. листинг 1):

пример считывания с портов

ЛИСТИНГ-1

```
...
int f_dba(int dba,int d[16])
{
    int i,PA;
    for (i=0;i<16;i++)
    {
        PA=inp(BA+dba+i);
        d[i]=PA;
    }
}
```

// dba- адрес FPGA, d- данные

Адрес считывания по ISA = BA+BAx+Ax, где BA – базовый адрес платы UNIOxx (выставлен 0x110), BAx – базовый адрес матрицы FPGA:

- для FPGA-1 (BAx=0xA000)
- для FPGA-2 (BAx=0xA400)
- для FPGA-3 (BAx=0xA800)
- для FPGA-4 (BAx=0xAC00)

Таблица. Расшифровка выходных сигналов энкодера

Адрес считывания	Распределение выходных сигналов по битам								Наименование
0x0	1D7	1D6	1D5	1D4	1D3	1D2	1D1	1D0	1D- угловое положение (младшие разряды) 1-го канала
0x1	1D15	1D14	1D13	1D12	1D11	1D10	1D9	1D8	1D- угловое положение (старшие разряды) 1-го канала
0x2	-	-	-	-	-	-	-	-	не задействованы*
0x3	-	-	-	-	-	-	-	-	не задействованы*
0x4	-	-	-	-	-	1HB	1OB	1OA	признаки (флаги): 1OA, 1OB, 1HB
0x5	2D7	2D6	2D5	2D4	2D3	2D2	2D1	2D0	2D- угловое положение (младшие разряды) 2-го канала
0x6	2D15	2D14	2D13	2D12	2D11	2D10	2D9	2D8	2D- угловое положение (старшие разряды) 2-го канала
0x7	-	-	-	-	-	-	-	-	не задействованы*
0x8	-	-	-	-	-	-	-	-	не задействованы*
0x9	-	-	-	-	-	2HB	2OB	2OA	признаки (флаги): 2OA, 2OB, 2HB
0x11	S7	S6	S5	S4	S3	S2	S1	S0	8- дискретных каналов (S) ввода-вывода
0x12	K7	K6	K5	K4	K3	K2	K1	K0	8- дискретных каналов (K) ввода-вывода

Рассмотрим подробнее...

Запустим среду Turbo C++ IDE ver.3.0 от Borland и создадим пустой проект. Далее необходимо провести инициализацию графического режима через InitGraph() и заикнуть опрос порта INP(). Причем предусмотреть выход из цикла по нажатию определенной комбинации клавиш во время опроса, к примеру, использованием функций Kbhit() и Getch(). В основном цикле необходимо осуществить считывание двухбайтного кода идентификатора схемы, положения ротора, признаков отказа и направления вращения. После чего, произвести отображение выделенных параметров в графическом виде. Оптимальным с точки зрения наблюдения – является имитация положения ротора датчика приращений с выводом дополнительной служебной информации. Реализация подобного подхода представлена в листинге 2:

тестовая программа считывания данных и ЛИСТИНГ-2

графического отображения состояний двух датчиков PDF

...

```
#include "dos.h"
#include "string.h"
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
#include <stdlib.h>

#define BA 0xA110
#define pi 3.14159

int f_dba(int dba,int d[16]);
void main(long int argc, char *argv[])
{
    // переменные порта
    int dba;int d[16];
    // переменные данных
    int FL,ugc,ugc2,FL1,FL2;
    // служебные переменные
    int s,pd,i,nomer,d1,d2;
    long int k,KT,KT2;
    float ugo1,ugol2,ug,n,ug2,ug1,KTT,vm,ug_1,ug_2;
    double v,ng1,ng2,nn;
```

```
int gdriver = DETECT, gmode, errorcode;
char msg[80];
int x,x2,y,y2,ns,nc,ugg,ugg2,xr,xr2,yr,yr2,wr,wr2,hr,hr2;
char nbuf[25];
char kbuf[25];
char ubuf[25];
char ibuf[25];
char sbuf[25];
char pbuf[25];

clrscr();
if(argc>1) // проверка параметров командной строки
{
    printf("argc=%d\n",argc);
    for(i=0;i<argc;i++)
        printf("%d = %s\n",i,argv[i]);
    if(strcmp(argv[1],"?") == 0)
    { // выдаем описание
        printf("Program driver encoder\n");
    }
    else if (strcmp(argv[1],"indtest") == 0){}
}
//
clrscr();
k = 1;
puts("Рассчитывать параметры по FPGA-?");
puts("Количество импульсов с датчика PDF-?");
scanf("%d %d", &s, &pd); // считываем номер FPGA
if (s == 1) {dba = 0x0;} // задаем базовый адрес
if (s == 2) {dba = 0x400;}
if (s == 3) {dba = 0x800;}
if (s == 4) {dba = 0xC00;}

/*Инициализация графического режима*/
initgraph(&gdriver, &gmode, "");
errorcode = graphresult();
if (errorcode != grOk)
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
}

/*Параметры рисунка*/
xr=180; // положение 1-го графика
```

```

yr=250;

xr2=470;                // положение 2-го графика

yr2=250;

wr=100;                 // размеры 1-го графика

hr=100;

wr2=wr;                 // размеры 2-го графика

hr2=hr;

//

setfillstyle(1,BLUE);

fillellipse(xr, yr, wr, hr);

fillellipse(xr2, yr2, wr2, hr2);    // 2-канал

setcolor(GREEN);

outtextxy(0,10, "Count:");          // количество импульсов

outtextxy(0,30, "Angle, deg:");     // угол поворота ротора

outtextxy(0,41, "Speed, r/s:");     // скорость

outtextxy(0,52, "HB:");             // направление вращения

outtextxy(0,63, "delta:");

// градусы

outtextxy(xr+wr+15, yr-5, "0");

outtextxy(xr2+wr2+15, yr2-5, "0");

outtextxy(xr-wr-40, yr-5, "180");

outtextxy(xr2-wr2-40, yr2-5, "180");

outtextxy(xr-5, yr-hr-25, "90");

outtextxy(xr2-5, yr2-hr2-25, "90");

outtextxy(xr-10, yr+hr+15, "270");

outtextxy(xr2-10, yr2+hr2+15, "270");

do {

    f_dba(dba,d);

    printf("Опос:%d", k);

    if (k>1)

    {

        setcolor(BLACK);

        outtextxy(90,10, kbuf);

    }

    setcolor(RED);

    itoa(k, kbuf, 10); // конвертируем в текст (max =10 символов)

    outtextxy(90,10, kbuf);

    /*FPGA_1*/

    printf("Идентификатор схемы= %c", d[14]); // см. схему

    алгоритма на ПЛИС

    //

    n = ((d[3]<=8)|d[2]);

```

```

if (n < 0) {nn = 65535 + n - 32768;}

if (n > 0) {nn = n;}

printf("Оборот: %7.1f", n);

if (k>1)

{

    setcolor(BLACK);

    outtextxy(5,90,nbuf);

}

nn=165001;

setcolor(RED);

itoa(nn,nbuf,25);

outtextxy(5,90,nbuf);

// формируем из двух байт значение положения (см. схему)

KT      = ((d[1]<=8)|d[0]);

KT2     = ((d[6]<=8)|d[5]);

ugol1   = KT;

ugol2   = KT2;

// пересчитываем в угловые коорд.положение ротора-1 и ротора-2

ug       = ugol*360/pd;

ug2      = ugol2*360/pd;

ugc      = ug/360;

ugc2     = ug2/360;

ug       = 360*(ugc - (ug/360));

ug2      = 360*(ugc2 - (ug2/360));

ugg      = 90-ug;

ugg2     = 90-ug2;

if (k>1)

{

    setcolor(BLACK);

    outtextxy(90,32, ubuf);

    outtextxy(130,32, ibuf);

}

setcolor(RED);

itoa(ug,ubuf,10);

itoa(ugol,ibuf,10);

outtextxy(90,32,ubuf);

outtextxy(130,32,ibuf);

//

FL=d[4]&0x04; // определяем направление вращения

if (k>1)

{

    if (FL == 0x04) {setcolor(BLACK);outtextxy(90,53,"left");}

```

```

    else {setcolor(BLACK);outtextxy(90,53,"righth");}
}
if (FL==0x04)
{
    setcolor(BLACK);outtextxy(90,53,"righth");
    setcolor(RED);outtextxy(90,53,"left");
}
else
{
    setcolor(BLACK);outtextxy(90,53,"left");
    setcolor(RED);outtextxy(90,53,"righth");
}

// значение разницы
if (k>1)
{
    setcolor(BLACK);
    outtextxy(90,63,pbuf);
}
setcolor(RED);
if (k == 1) {d1 = ug;d2 = ug2;}
itoa((ug-ug2)*pd/360,pbuf,10);
outtextxy(90,63, pbuf);

// скорость вращения
if (k==1) {ug_1=ug;ng1=ugc;}
ug_2=ug;ng2=ugc;
vm = ((ng2-ng1)*360-ug_1+ug_2)/(360*(125e-3));
printf("Vm, об/с= %8.6f\n",vm);
if (k>1)
{
    setcolor(BLACK);
    outtextxy(90,43,sbuf);
}
setcolor(RED);
itoa(vm,sbuf,10);
outtextxy(90,43,sbuf);
ug1=ug2; ng1=ng2;

// маркер (вектор на графике)
if (k>1)
{
    setcolor(BLUE); // 1-канал
    line(x,y,xr,yr);
    line(x2,y2,xr2,yr2);

```

```

}
setcolor(RED);
line(xr-wr-10,yr,xr+wr+10,yr); // ось-x
line(xr2-wr2-10,yr2,xr2+wr2+10,yr2); // ось-x 2-канал
line(xr,yr+hr+10,xr,yr-hr-10); // ось-y
line(xr2,yr2+hr2+10,xr2,yr2-hr2-10); // ось-y 2-канал

x=xr+(wr*sin(ugg*pi/180)); x2=xr2+(wr2*sin(ugg2*pi/180));
y=yr+(hr*cos(ugg*pi/180)); y2=yr2+(hr2*cos(ugg2*pi/180));
setcolor(RED); // 1-канал
line(xr,yr,x,y);
line(xr2,yr2,x2,y2);
// конец маркер

k+ = 1;
delay(100);
if (!kbhit()) s=getch(); // перехватываем нажатую клавишу для
                        // остановки программы

if (s==0x9) break;
} while (s!=0x13);

closegraph();
textmode(LASTMODE);
return;
}

// опрос порта по базовому адресу + смещение
int f_dba(int dba,int d[16])
{
    int i,PA;
    for (i=0; i<16; i++)
    {
        PA = inp(BA + dba + i);
        d[i] = PA;
    }
}

```

Запустим проект на компиляцию и выполнение командой <Ctrl>+<F9>. При этом, появится экран с запросом параметров считывания (см. рисунок 13). Задав номер FPGA = 3 и максимальное количество импульсов для подключенного датчика = 2048 (для нашего датчика RV-58N), нажмем <ENTER> и получим экран с визуализацией состояний двух энкодеров (см. рисунок 14-15). На рисунке 14 представлен экран состояния датчиков приращений при

удаленном просмотре через сеть TCP/IP, но это уже тема отдельной статьи...

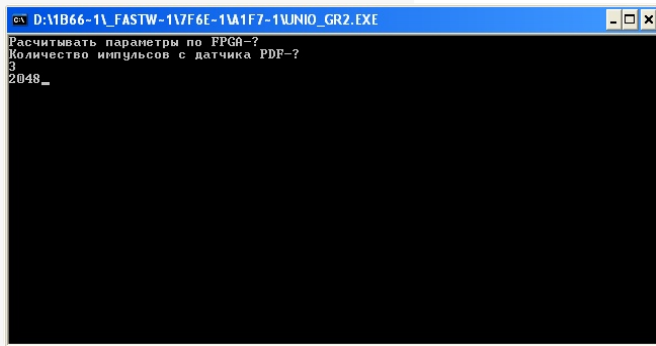


Рис. 13. Экран задания параметров считывания



Рис. 14. Удаленный просмотр датчиков приращений

Заключение

Вот в принципе и все. Наслаждаемся аппаратным декодированием состояния шифратора приращений и визуальным просмотром...

Рассматриваемые в данной статье прошивка энкодера для ПЛИС в пакете XILINX 3.1, исходники и компиляция тестовой утилиты визуализации состояния датчика приращений и загрузчик ISP полностью приведены в виде ресурсов в теме «Журнал клуба программистов. Пятый выпуск» или непосредственно в архиве с журналом [4].

Ресурсы

- С. Бадло. Энкодер датчика PDF на ПЛИС. – ПРОграммист, Клуб ПРОграммистов, 2010, №4, с.40
<http://raxp.radioliga.com/cnt/s.php?p=pro4.pdf>
- Сайт производителя Xilinx
<http://www.xilinx.com>
- С. Бадло. JTAG.Xilinx программатор. – Радиолучитель, Минск, 2008, №7, с.38
<http://raxp.radioliga.com/cnt/s.php?p=jtag.pdf>
- Модули и проекты, использованные в статье
<http://raxp.radioliga.com/cnt/s.php?p=pro5.pdf>



Рис. 15. Экран визуализации состояний двух датчиков PDF

При разработке микроконтроллерных устройств с внешней памятью типа AT45DBxx приходится сталкиваться с ситуацией, когда блоки данных (файлы) записываются редко, обычно при обновлении ПО устройства, а иногда вообще только один раз - при изготовлении устройства. В таких случаях нет необходимости в использовании файловой системы дискового типа, таких как FAT...



Вячеслав Мовила

<http://movilavn.narod.ru>

Проще и эффективнее использовать файловую систему последовательного доступа, наподобие ленточной. При этом файлы вместо названия имеют только индексы - последовательный номер файла на устройстве. Данные файла при записи записываются в конец свободного адресного пространства устройства, в этом случае при закрытии файла ему назначается уникальный номер (индекс). Доступ к файлу при чтении происходит по его индексу. На устройстве может находиться не более 62 записанных файлов. Рассматриваемая библиотека организует такую файловую систему для микросхемы AT45DB161, порт CodeVision. Листинги библиотеки подробно прокомментированы [1...3]. Так, что при желании не потребуются больших усилий для портирования на другой компилятор и для внесения изменений с целью применения для другого типа микросхем серии AT45DBxx.

Краткое описание AT45DB161

AT45DB161 (см. рис.1) - является Flash памятью с

последовательным интерфейсом, и идеально подходит для широкого спектра цифровых голосовых приложений, приложений визуализации, и приложений хранения программного кода и данных. 17 301 504 бит памяти данной ИС организованы в 4096 страниц по 528 байт каждая. Кроме памяти общего назначения ИС, также, имеет два SRAM буфера данных по 528 байт. Буферы обеспечивают возможность приема данных в режиме перепрограммирования страницы основной памяти, или считывание, или запись непрерывных потоков данных. Режим эмуляции EEPROM (с побитным или побайтным изменением) прост в применении, благодаря встроенной, трехступенчатой системе команд **Read - Modify - Write**. В отличие от стандартных типов Flash памяти, обращение к которым, происходит произвольным образом в режиме многочисленных адресных строк и при помощи параллельного интерфейса, память типа DataFlash использует последовательный интерфейс для обращения к своим данным в режиме последовательного доступа. ИС поддерживает SPI - режимы типа 0 и 3. Простой последовательный интерфейс облегчает разводку платы, увеличивает отказоустойчивость системы, минимизирует коммутационные шумы,



Мовила Вячеслав Викторович

родился 28 июля 1957 г.р., инженер.

Интересы: фантастика, читаю много и с увлечением, имею неплохую библиотеку - собирал много лет. Шахматы, к

сожалению сейчас не хватает времени. Но самое страстное увлечение и работа - это программирование, основной язык C++, а также, естественно Assembler, Pascal. WEB-программирование, создание динамических, интерактивных сайтов с использованием JavaScript, Java, PHP, MySQL. Базы данных, работа с CASE-средствами разработки информационным систем. Схемотехника микропроцессорных и микроконтроллерных систем, программирование микроконтроллеров и DSP.

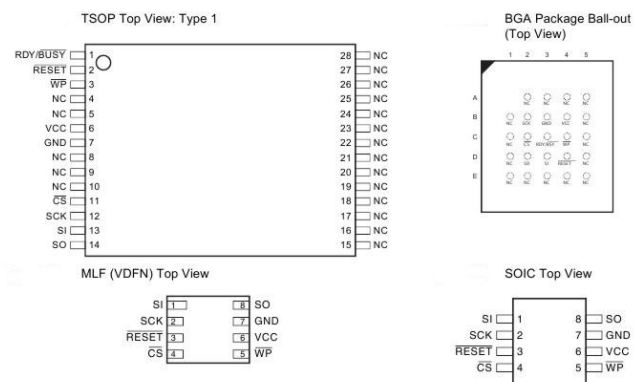


Рис. 1. Распиновка ИМС

а также, уменьшает размер корпуса и число необходимых активных выводов. ИС оптимизирована для использования в широком

круге коммерческих и индустриальных приложений, для которых существенную роль играют высокая плотность размещения, малое число выводов, низкое напряжение питания, и низкое энергопотребление. ИС функционирует с тактовыми частотами, вплоть до 20 МГц при типовом потребляемом токе в режиме активного чтения 4 мА.

Предпосылки выбора

Для обеспечения удобства внутрисистемного перепрограммирования, ИС АТ45DB161 не требует высоких входных напряжений в режиме программирования. ИС питается от однополярного источника с напряжением от 2.5 В до 3.6 В, или от 2.7 В до 3.6 В, как в режиме программирования, так и в режиме чтения. Выборка ИС АТ45DB161 производится по входу **CS** (активный низкий), а доступ к ИС обеспечивается посредством 3-х проводного последовательного интерфейса, состоящего из сигнала последовательного входа **SI**, последовательного выхода **SO** и последовательного тактового сигнала **SCK** (см. таблицу):

Таблица. Расшифровка сигналов ИМС

Наименование вывода	Назначение
CS (активный низкий)	Вход выборки
SCK	Последовательный тактовый сигнал
SI	Последовательный вход
SO	Последовательный выход
WP (активный низкий)	Вход аппаратной защиты записи страницы памяти
RESET (активный низкий)	Инициализация
RDY/BUSY (активный низкий)	Готовность/ занятость

Все циклы программирования имеют встроенный контроль временных характеристик, а для проведения программирования предварительный цикл стирания не требуется. При поставке ИС от Atmel, старшая значащая страница массива памяти может не быть чистой. Другими словами, содержимое последней страницы может быть заполнено содержимым, отличным от FFH.

Отличительные особенности

- Однополярное напряжение питания от 2.5 В до 3.6 В, или от 2.7 В до 3.6 В
- Совместимость с последовательным периферийным интерфейсом типа SPI
- Максимальная тактовая частота 20 МГц

- Постраничный режим программирования:
- Одиночный цикл перепрограммирования (стирание плюс программирование)
- 4096 страниц основной памяти (528 байт на страницу)
- Поддержка страничного и блочного режимов стирания
- Два 528-ми байтных буфера данных SRAM, обеспечивающих прием данных в режиме перепрограммирования энергонезависимой памяти
- Поддержка режима непрерывного считывания полного массива данных
- Идеально для приложений теневого дублирования кода
- Низкое энергопотребление:
4 мА - типичный ток в режиме активного чтения
2 мкА - типичный потребляемый ток в режиме ожидания
- Аппаратная функция защиты данных
- Входы сигналов SI, SCK, CS (активный низкий), RESET (активный низкий) и WP (активный низкий) устойчивы к логическим уровням 5 В
- Коммерческий и промышленный диапазоны температур

Схема подключения

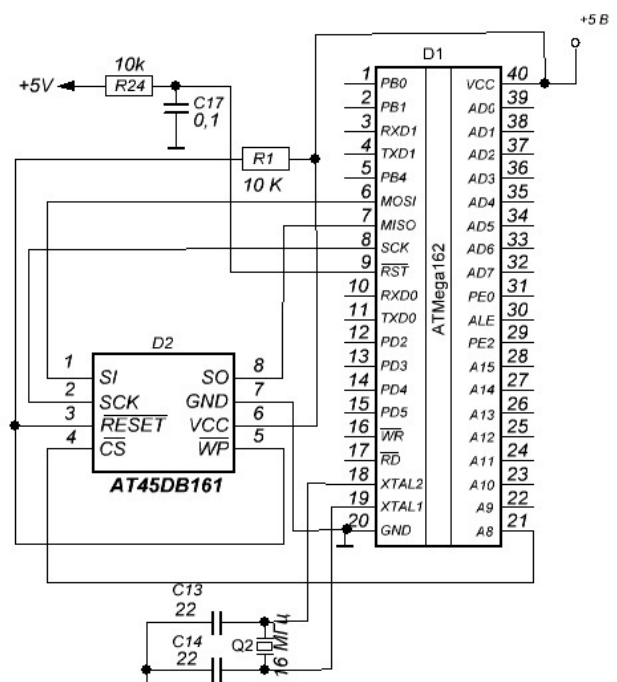


Рис. 2. Схема электрическая принципиальная
тестового модуля считывания данных

Несмотря на то, что напряжение питания микросхемы AT45DB161 указано от 2.7 до 3.6 В, микросхема нормально работает при напряжении 5 В. В реальном проекте – используется питание 3 В. Поэтому подключил AT45DB161 на свою макетницу с микроконтроллером ATmega162 с 5-ти вольтовым питанием. Все тесты прошли исключительно хорошо: от самых простых до более сложных – воспроизведение звука с использованием ШИМ-модуляции. Микросхема AT45DB161 использовалась в корпусе SOIC, в котором отсутствует отдельный вывод BYSE, что видно на схеме принципиальной (см. рисунок 2) и было учтено при написании библиотеки.

Структура файловой системы

Первая страница (528 байт) отводится для хранения системной информации о файловой структуре устройства.

Первые 11 байт - это строка `*AT45DB161#`. Строка является идентификационной записью, которая показывает, что устройство отформатировано и готово к работе. В 12-ом байте содержится число, показывающее количество файлов на устройстве.

Далее идут 8-ми байтовые структуры, выполняющие функцию описателей файлов:

- в первых 2-х байтах адрес 1-ой страницы файла
- следующие 2 байта содержат количество страниц в файле
- последние 4 байта структуры - размер файла в байтах

Функции библиотеки

1. Инициализация файловой системы AT45DB161.
`unsigned char InitialFlash();`

Возвращает значения:

- 0 - инициализировано.
- 1 - устройство отсутствует в системе.
- 2 - устройство не форматировано.

* Комментарий автора.

Внимание! При 6 В быстро выгорает, по крайней мере так было у меня.

2. Форматирование файловой системы AT45DB161

`void FormatFlash();`

3. Закрывать файл

`void CloseFile();`

4. Открыть файл

`unsigned char OpenFile(char mode, unsigned char nfile);`

Параметр `mode`:

`r` - Открыть файл в режиме чтения.

`w` - Открыть файл в режиме записи.

Параметр `nfile`:

В режиме чтения указывает индекс (номер) файла. В режиме записи не имеет значения, следует указать 0.

Возвращает значения:

0 - ошибка при открытии файла.

`N(1...62)` - индекс успешно открытого файла..

После успешного открытия файла будут доступны глобальные переменные статуса файла:

```
// Переменные описывающие состояние рабочего файла
/*
0 - Файл закрыт
1 - Файл открыт в режиме записи
2 - Файл открыт в режиме чтения
*/
unsigned char at45_file_mode = 0;
// Счетчик страниц при записи
unsigned int at45_cpwrite = 0;
// Номер текущей страницы
unsigned int at45_npage = 0;
// Индекс открытого файла
unsigned char at45_currfile = 0;
// Размер открытого файла/счетчик байт откр. файла при записи
unsigned long at45_cbfile = 0;
// Счетчик байт открытого файла при чтении
unsigned long at45_countb = 0;
// К-во страниц в файле
unsigned int at45_kpage = 0;
// Счетчик байт записанных/читанных в/из буфер
unsigned int at45_buffer_counter = 0;
```


5. Считать из файла count байт в буфер dest
`unsigned int ReadFile(char * dest, unsigned int count);`

`dest` - Указатель на буфер чтения.

`count` - Количество байт для считывания.

Возвращает значения:

`N` - количество считанных байт.

6. Записать в файл count байт из буфера source.
`unsigned int WriteFile(char * source, unsigned int count);`

`source` - Указатель на буфер записи.

`count` - Количество байт для записи.

7. Читать файл побайтно

`unsigned char ReadByte(unsigned char * flag);`

`flag` - Указатель на переменную типа `unsigned char`. При чтении первого байта должен из вызывающей функции установлен в 0. При успешном чтении байта внутри функции `ReadByte` устанавливается в 1. При ошибке чтения или достижении конца файла внутри функции `ReadByte` устанавливается в 2.

Несмотря на то, что функция выглядит несколько неуклюже, она очень удобна при побайтном считывании по прерыванию.

Применение библиотеки

Для того, чтобы использовать библиотеку в проект ПО следует включить файлы: `<at45db161.c>` и `<at45db161.h>`. В файле `<at45db161.h>` следует скорректировать 2 настройки:

Первая:

```
// DataFlash chip select
// Здесь следует определить пин порта мк. для CS AT45DB161!!!
#define DF_CHIP_SELECT PORTC.0
```

В строке 7 указать бит порта вывода который Вы будете использовать в качестве сигнала CS микросхемы AT45DB161:

Вторая:

Переопределить настройки интерфейса SPI в строке 38:

```
// Здесь следует определить настройки SPI!!!
// Включить SPI.
// SPI initialization
// SPI Type: Master
// SPI Clock Rate: 4000.000 kHz
// SPI Clock Phase: Cycle Start
// SPI Clock Polarity: High
// SPI Data Order: MSB First
#define SPI_ON SPCR=0x5c
```

Простой пример применения библиотеки:

```
unsigned char temp;

temp = InitialFlash();
if(temp) {
if(temp == 1) {
// И зацкливаемся - ус-во не обнаружено в системе
while(1);
}

// ИНАЧЕ УСТРОЙСТВО ОБНАРУЖЕНО НО НЕ ФОРМАТИРОВАНО!

// ФОРМАТИРУЕМ FLASH
FormatFlash();
}

// Записали первый файл:
OpenFile('w', 0);
memset(Buff, 0, 128);
for(ic = 0; ic<256; ic++) Buff[ic] = ic;
WriteFile(Buff, 256);
WriteFile(Buff, 256);
WriteFile(Buff, 256);
WriteFile(Buff, 256);
WriteFile(Buff, 256);
CloseFile();

// Записали второй файл:
OpenFile('w', 0);
memset(Buff, 2, 128);
//for(ic = 0; ic<256; ic++) Buff[ic] = ic;
WriteFile(Buff, 128);
WriteFile(Buff, 55);
CloseFile();
```

```
// Записали третий файл:

OpenFile('w', 0);

memset(Buff, 3, 128);

//for(ic = 0;ic<256;ic++) Buff[ic] = ic;

WriteFile(Buff, 128);

WriteFile(Buff, 23);

CloseFile();


// Считали третий файл:


If (OpenFile('r', 3)) {
do {
memset(Buff, 0, 528);
ic = ReadFile(Buff, 64);
} while(ic);
CloseFile();
}

//Считали второй файл:
if(OpenFile('r', 2)) {
do {
memset(Buff, 0, 528);
ic = ReadFile(Buff, 64);
} while(ic);
CloseFile();
}

// Считали первый файл:


if(OpenFile('r', 1)) {
memset(Buff, 0, 528);
do {
ic = ReadFile(Buff, 64);
} while(ic);
CloseFile();
}
```

Заключение

Как видите, использование библиотеки последовательного доступа значительно упрощает разработку целого ряда устройств с голосовым сопровождением на основе Flash-памяти.

** Важное замечание.

При считывании последнего блока данных из файла в буфер заносится действительное оставшееся количество байтов, а их количество указывается в возвращаемом функцией значении.

*** Комментарий автора.

Программа, которая умеет открывать файлы мультимедиа типа WAV, идентифицировать их, на соответствии требованиям зачатки (частота дискретизации, количество каналов и т.д.) и пересылать их в микроконтроллерную систему.

Рассматриваемые в данной статье исходники библиотеки файловой системы последовательного доступа полностью приведены в виде ресурсов в теме «Журнал клуба программистов. Пятый выпуск» или непосредственно в архиве с журналом.

Ресурсы

- Ссылка на скачивание библиотеки LIB_AT45
http://movilavn.narod.ru/LIB_AT45.rar
- Проект для микроконтроллера ATmega162 - загрузка файлов в AT45DB161
<http://movilavn.narod.ru/AT45LOAD.rar>
- Проект для ПК (Builder C++ 6). LoadAT45DB***
<http://movilavn.narod.ru/LoadAT45DB.rar>
- Модули и проекты, использованные в статье
<http://programmersclub.ru/pro/pro5.pdf>

Здравствуйтесь, любители гейминга. В данной статье, я хочу показать, как делаются плагины для всем известной GTA. Начнем мы самого простого - это программирование плагинов на Delphi для Grand theft Auto ViceCity. А поняв принцип их работы, никакого труда не составит написать плагин и для других серий GTA...



Виталий Иванов

by **VintProg** vintprog@gmail.com

Пишем простой плагин для GTA - VC*

Итак, Для работы нам понадобится следующее:

- 1) IDE среда Delphi [1]
- 2) знание языка
- 3) утилита ArtMoney** [2]

И немного теории, что же такое плагины. Плагины - это те же динамические подключаемые библиотеки DLL. Однако часто бывает, что им изменяют расширение.

Возможно, вы спросите: «...как же это работает все?», А работает оно следующим образом... При запуске <gta-sa.exe> загружаются комплектные библиотеки от разработчиков. В одной из этих DLL, в частности - `vorbisFile.dll` имеется функция загрузки библиотек *.asi. И пожелавшие остаться неизвестными, программисты написали <cleo.asi> и набили ее весьма и весьма полезными функциями, такими как: новые опкоды, загрузка плагинов *.cleo и т.п. Когда загрузилась библиотека <cleo.asi>, ее код выполняет нужные функции в памяти игры.

* Комментарий автора.

Вы наверняка встречали Cleo на GTA-SA, и видели, что там существует такая библиотека `cleo.asi`. Так вот она и загружает из папки Cleo - скрипты и сами плагины *.cleo.

Именно благодаря этому и появляются новые возможности в игре. А что касается GTA Vice City, то в ней тот же процесс, только библиотеки *.asi загружаются из библиотеки <Mss32.dll>. Отсюда понятно, что для того чтобы писать плагины - необходимо хорошо уметь работать с памятью

** Комментарий редакции.

ArtMoney - программа, предназначенная для редактирования параметров в компьютерных играх, для получения бесконечных денег, жизней, патронов и т.п. Она умеет сканировать память или файлы игры для поиска каких-то определенных значений (деньги, ресурсы). Официальный сайт www.artmoney.ru



игры и знать что за значения находятся в игровой памяти в определенном адресе.

Приступим... Для начала запустим Borland Delphi, после чего кликаем на «File -> New -> other...» и перед нами откроется вот такое окно (см. рисунок 2):

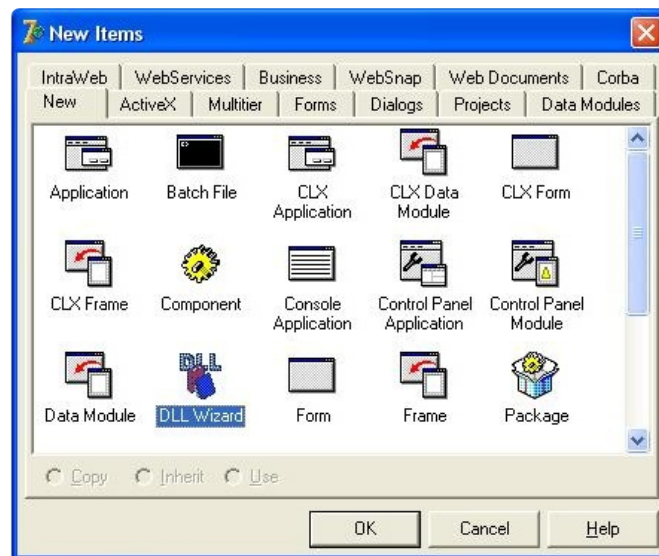


Рис. 2. Выбор DLL Wizard

Далее выделяем DLL Wizard и жмем OK. Сразу возьмем и сохраним наш проект «File->Save Project As» и под именем ShowMessage, чтобы

получилось как показано на рисунке 3:

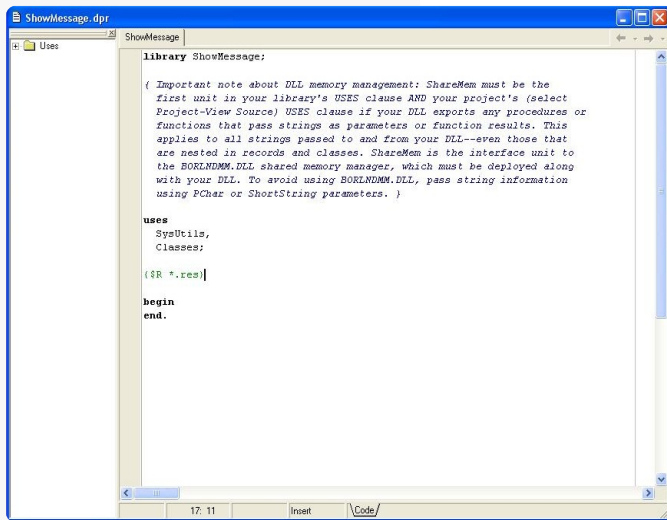


Рис. 3. Заготовка плагина

Также можно удалить директиву `{ $R *.res }`, потому-что для данного плагина мы не будем использовать ресурсы. Из дополнительных модулей оставим лишь – Windows, а остальные удалим. Теперь напишем следующий код:

```
library ShowMessage;

uses

    Windows;

var HWND : THandle;
begin
    // Получить хендл окна GTA: Vice City
    HWND := FindWindow(nil, 'GTA: Vice City');

    if HWND <> 0 then // Проверка если окно GTA: Vice City
        // существует, то тогда выполнить MessageBox
        MessageBoxA(HWND, 'Плагин загружен', 'Сообщение', 0)
end.
```

И скомпилируем его. Прокомментирую работу данного кода... Когда библиотека загружается, между `Begin` `end` начинает выполняться код, Тут сразу появляется окошко с сообщением. Чтобы это заработало, переименуйте расширение *.DLL на *.ASI или воспользуйтесь директивой `{ $E .ASI }`. После чего, скопируйте библиотеку в каталог GTA и запустите игру GTA-VC.exe, Далее мы увидим окошко, когда загрузится <mss32.dll>. Поздравляю, вы написали свой первый плагин для GTA***!

Пишем простой плагин-трейнер для GTA - VC

Настала пора сделать что-то полезное. Итак, сперва подумаем, что нам еще нужно? А нужно нам сделать плагин-трейнер. Для чего он предназначен? К примеру, мы можем сделать так, чтобы при нажатии определенной кнопки в игре – появлялось окно, в котором можно прибавить деньги игроку. Для этого, запускаем Gta-VC, сворачиваем ее и запускаем ArtMoney. В этой утилите выбираем «Искать -> Объект -> Процесс» и в выпадающем списке, где написано «Выбери процесс», выберем нашу GTA Vice City (см. рисунок 4). Теперь зайдём в GTA-VC и соберем небольшое количество денежных средств (к примеру, как я набрал \$18). Судя по представлению «денег» в игре, видно, что они целого числа и можно набрать их в игре большое количество. Отсюда, мы уже знаем тип представления данных «денег».

Это 4-байтные целые числа, которые нам и нужно отыскать (см. рисунок 5).

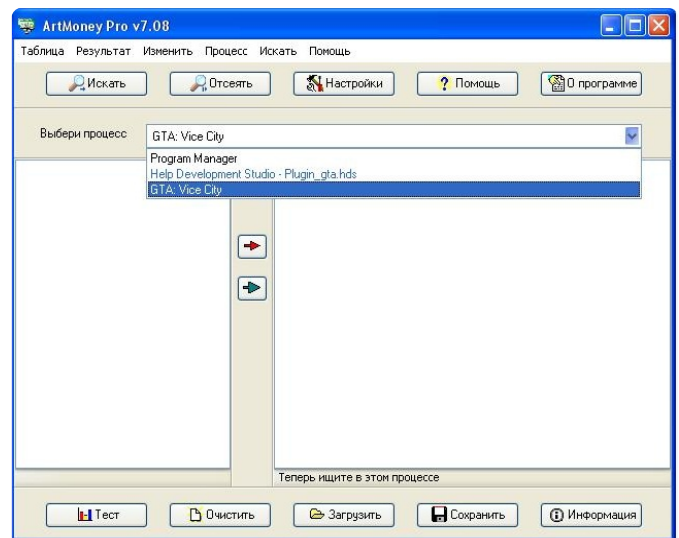


Рис. 4. Выбор GTA-VC в ArtMoney

Обратите внимание! Вам необходимо повторить те же действия, как показано на рисунке, только у вас будет свое число. По завершению поиска у вас появится длинный список адресов (см. рисунок 6). Но не переживайте по этому поводу. Ведь адрес «денег» мы будем искать более легким методом, На то она и ArtMoney. Следующий шаг будет таков: снова заходим в

*** Комментарий автора.

Если что то не получается то пример расположен в каталоге "Examples\Plugin1".

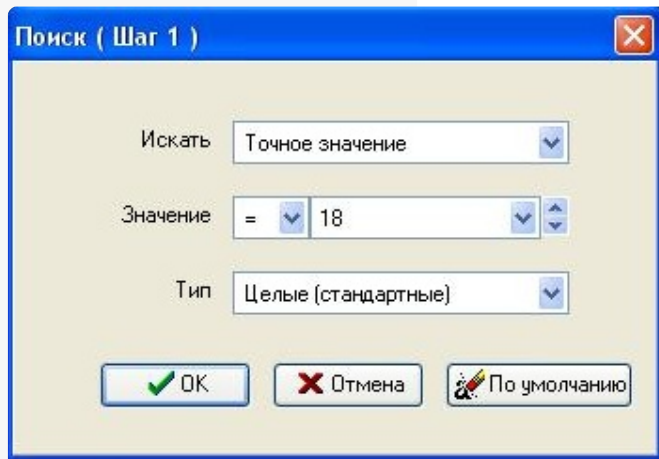


Рис. 5. Поиск значений по заданному типу данных

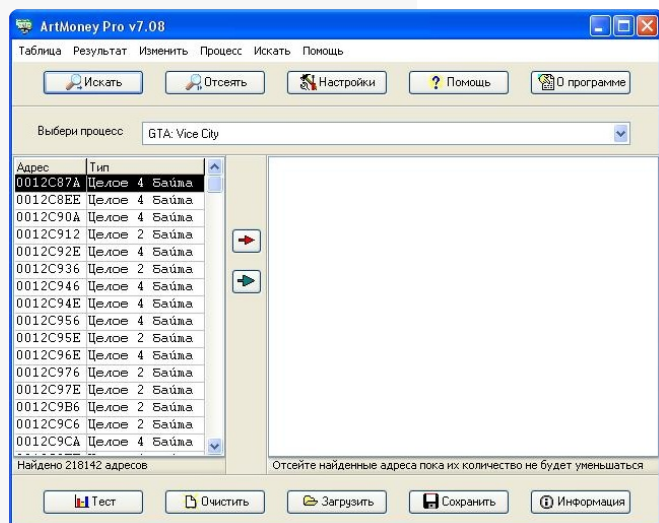


Рис. 6. Выборка адресов в ArtMoney

GTA и либо тратим, либо добавляем деньги к игроку Томму, запоминаем измененное количество «денег» и сворачиваем GTA. Вписываем новое значение (см. рисунок 7):



Рис. 7. Повторный поиск значений по заданному типу данных

Теперь нажимаем «отсеять». Как видите, длина списка адресов значительно уменьшится. И так продолжаем до тех пор, пока не останется один адрес. Если же все равно остается несколько адресов, то меняем в них значения и проверяем изменилось ли количество «денег» в GTA (см. рисунок 8). Если все нормально, то нормально. Однако, еще осталось сделать одну проверку на указатель. ArtMoney не закрываем, так все и оставляем. Вырубаем GTA-VC и заново запускаем GTA-VC. В ArtMoney, в выпадающем списке, где написано «Выбери процесс», заново выбираем GTA и повторим операции по изменению значений по адресу местонахождения «денег». Если все работает нормально, то записываем

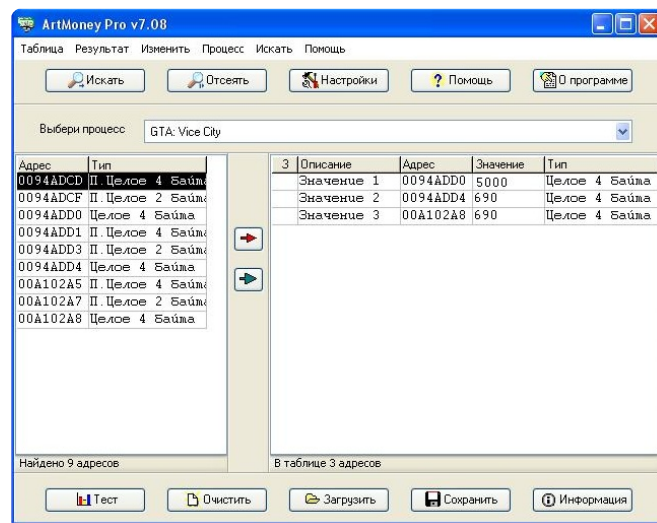


Рис. 8. Повторная выборка адресов в ArtMoney

этот адрес в текстовой блокнот и вырубаем GTA-VC и выключаем ArtMoney. Теперь он нам не понадобится.

Итак, первый шаг сделан. Мы нашли адрес «денег» и остается лишь написать плагин.

Распишем особенность работы плагина, то есть то – как он будет работать: так, при нажатии кнопки <M> добавляется 1000 долларов, а значит нам нужен обработчик нажатия кнопки. Воспользуемся таймером. Теперь, точно так же как и в первом примере, создадим новый проект DLL и назовем его «MoneyAdds». И напишем следующий код:

**** Комментарий автора.

Хочу напомнить, что не на всех версиях GTA-VC могут быть одинаковые адреса. Так что имейте это в виду, при написании плагинов.

```

library MoneyAdds;

{ GTA-VC 1.1 Плагин для добавления денег }

uses Windows;

type // определяем свой тип (указатель целых чисел)
  P_Integer = ^Integer;

var
  GTA_VC_Handle : THandle;
  CurrentMoney : Integer;
  keyUp : boolean;

const // тут твой найденный адрес «денег»
  Address_Money = $0094ADD0;

{$E .asi}

//---- Эта процедура будет вызываться таймером ---
procedure Timer_begin;
begin

  // Нажатие и отпуск "M"
  if not GetKeyState($4D) < 0 then
    keyUp := true;

  if (GetKeyState($4D) < 0) and (keyUp = true) then
    begin
      // Читаем деньги из GTA-VC и присваиваем в CurrentMoney
      CurrentMoney := P_Integer(Address_Money)^;

      // Записываем 1000 + текущие деньги
      P_Integer(Address_Money)^ := CurrentMoney + 1000;

      keyUp := false;
    end
  end;

  //-----

begin
  GTA_VC_Handle := FindWindow(nil, 'GTA: Vice City');
  if GTA_VC_Handle <> 0 then
    begin
      SetTimer(GTA_VC_Handle, 0, 25, @Timer_begin);
    end;
  end.

```

Исходный код этого плагина находится в «Examples\Trainer1» [3]. Вот мы и реализовали плагин-трейнер добавления «денег» по нажатию клавиши «М», Только скажу – не выгодно на каждый плагин делать один таймер, Поэтому имейте ввиду, что таймер нагружает процессор. Для решений данной проблемы можно воспользоваться функциями DirectX для обработки нажатий клавиатуры.

Пишем свой собственный менеджер-загрузчик плагинов

Вы наверняка заметили одну вещь: когда создаешь новый плагин, его приходится бросать рядом с GTA-VC.exe, А представьте себе, если таких плагинов будет больше десятка? Это не наш метод, поэтому мы напишем свой загрузчик плагинов из отдельно созданного каталога под наши плагины, скажем <bin>. И пускай там будет их хоть 1000!

Итак, запускаем среду Delphi и по аналогии создадим проект DLL-ки. Внутри напишем следующий код:

```

library LoaderVc;

{ Даная библиотека нужна для загрузки плагинов в GTA-VC }

{$E .ASI}

uses SysUtils, Windows;

var
  SearchRec : TSearchRec;
  filename : pAnsiChar;

const
  dir_bin = 'Bin\*.bin';
  dir_dll = 'Bin\*.dll';

//--- Процедура отыскивает все плагины из папки Bin и
// подгружает их ---
procedure Load_libs(FileName : string);
begin

  if FindFirst(FileName, faAnyFile, SearchRec) = 0 then
    repeat

```

```
begin  
    filename := pAnsiChar('Bin\' + SearchRec.name);  
    LoadLibrary(filename);  
end;  
  
until FindNext(SearchRec) <> 0;  
  
end;  
//-----  
  
begin  
    Load_libs(dir_dll);  
    Load_libs(dir_bin);  
end.
```

Исходный код этого менеджера-загрузчика находится в «Examples\Loader_VC» [3].

Заключение

Вот теперь готов загрузчик плагинов bin и dll, Теперь достаточно бросить его в корневую папку GTA, создать там каталог <BIN> со всеми нашими плагинами, запустить игру и полюбоваться результатом наших трудов.

Рассматриваемые в данной статье исходники плагина добавления «денег», плагина-трейнера и менеджера-загрузчика полностью приведены в виде ресурсов в теме «Журнал клуба программистов. Пятый выпуск» или непосредственно в архиве с журналом.

Продолжение смотрите в следующем выпуске нашего журнала...

Ресурсы

- Бесплатный TurboDelphi-Lite (over BDS-2006)
<http://www.andyaska.com/?act=download&mode=get&id=34>
- Скачать ArtMoney
http://www.artmoney.ru/r_download_se.htm
- Учебник. Искусство изменения GTA
http://programmersup.3dn.ru/load/skachat_vse_na_gta/uchebniki_po_gta/10
- Модули и проекты, использованные в статье
<http://programmersclub.ru/pro/pro5.pdf>

Приветствую, читателей журнала «ПРОграммист», эта статья стартует новую рубрику журнала «Архив», где вы сможете прочитать, интересные статьи, мысли и заметки, ранее опубликованные на форуме программистов <http://programmersforum.ru>. Тема сегодняшнего номера – DLL. Как создать собственную Dll на Delphi...



Александр Архипов

by Alar www.programmersforum.ru

Когда мы только начинали изучать программирование в *Pascal*, мы познакомились с функциями и процедурами из внешних модулей – файлом с кодами для выполнения стандартных операций. Например, `uses crt` (модуль CRT) – содержит операции для работы с экраном. Вероятно, первую, которую вы использовали, была процедура очистки экрана – `ClrScr`.

Позже было написано множество модулей для использования в Pascal, далее Delphi. На сайте <http://delphibasics.ru> вы можете ознакомиться с самими популярными или необходимыми модулями: `System`, `SysUtils`, `StrUtils`, `DateUtils`, `FileCtrl`, `ConvUtils`, `StdConv`, `Math`, `Classes`, `Dialogs`, `Types`, `Variants`.

В СИ – тоже есть аналогичные возможности по подключению стороннего или ранее написанного кода. Например, использование заголовочных файлов – имеет по умолчанию расширение `<.h>`.

Но использование одного модуля в другом компиляторе, особенно если компилятор другого языка программирования – становится проблемой. Раньше – эта проблема решалась переписыванием кода с одного языка на другой.

Потому для разработчиков использование DLL было прорывом. Вы можете создать один раз DLL и использовать ее для следующих разработок. Также плюсом является то, что DLL можно динамически подгрузить во время работы программы и после подключения использовать функции DLL несколькими приложениями.

С примером эффективного использования DLL вы можете ознакомиться в игре Fortress <http://pkonkurs.ru>, где несколько независимых разработчиков, пишут искусственный интеллект или же алгоритм действий по определенным

правилам. И на платформе тестирования определяется лучшая разработка. Аналогично можно проводить состязания по классическим играм: шашки, шахматы.

Пример создания функции для DLL, которую смогут использовать любые игровые платформы для реализации стратегий бота:

```
function GetTurn(aPlayerNumber:integer; aGame:PGame;
  aAvailProjects: PAvailProjects;
  AGI:PAAdditionalGameInfo):integer; stdcall; export;
var
  i, myi, ch:integer;
begin
  if aAvailProjects=nil then exit;

  myi:= aPlayerNumber;
  i:= aPlayerNumber;
  if i=1 then i:=0
    else i:=1;

  // Функция хода бота. если номер хода больше нуля, то выбрать
  // крайний элемент массива aAvailProjects (доступные проекты)
  if AGI.TurnCount > 0 then Result:=aAvailProjects^[0];

  // Выбрать проект 27 - спецатака, если щит противника равен
  // нулю и достаточно ресурсов на использование проекта
  if (aGame[myi].Elements>3) and
    (aGame[myi].Metal>2) and
    (aGame[i].Shield=0) then Result:=27;

end;
```

SkyM@n, еще в 2007 году опубликовал эту статью на форуме, но она так и не получила широкой огласки, потому исправляемся и публикуем ее в журнале и клубе <http://programmersclub.ru>.

Когда-то, как только начал изучать Дельфи, возник вопрос о создании динамически подключаемых библиотек, не ActiveX, а Native DLL (Dynamic Link Library). Например, на Visual Basic это сделать сложно, теоретически (да и практически, юзая ассемблерные вставки) – можно, но на Дельфи все проще. Эта статья, думаю, пригодится не одному начинающему дельфисту-испытателю, так что если на то позволяет время и желание – советую испытать все на себе...



Сергей Матрунчик

by SkyM@n www.programmersforum.ru

В связи с бурным развитием технологий программирования, все больше людей сталкиваются с проблемой наращивания возможностей своих программ. Данная статья посвящена именно этому вопросу, а именно - программирование DLL в Borland Delphi.

Кроме того, так как мы затронем вопросы по использованию библиотек DLL, то попутно коснемся импортирования функций из чужих DLL (в том числе и системных, т.е. WinAPI).

Области применения DLL

Итак, зачем же нужны библиотеки DLL и где они используются? Перечислим лишь некоторые из областей их применения:

- отдельные библиотеки, содержащие полезные для программистов дополнительные функции (например, функции для работы со строками, или же - сложные библиотеки для преобразования изображений)
- хранилища ресурсов (в DLL можно хранить не только программы и функции, но и всевозможные ресурсы - иконки, рисунки, строковые массивы, меню, и т.д.)
- библиотеки поддержки (в качестве примера можно привести библиотеки таких известных пакетов, как: DirectX, ICQAPI (API для ICQ), OpenGL и т.д.)
- части программы (в DLL можно хранить окна программы (формы), и т.п.)
- плагины* (Plugins)
- разделяемый ресурс (DLL может быть использована сразу несколькими программами или процессами, так называемый **sharing** - разделяемый ресурс)

Краткое описание функций и приемов для работы с DLL

Каковы же приемы и функции необходимо использовать, чтобы работать с DLL? Разберем два метода импортирования функций из библиотеки:

1 способ. Привязка DLL к программе

Это наиболее простой и легкий метод для использования функций, импортируемых из DLL. Однако, и на это следует обратить внимание, этот способ имеет очень весомый недостаток: если библиотека, которую использует программа, не будет найдена, то программа просто не запустится, выдавая ошибку и сообщая о том, что ресурс DLL не найден. А поиск библиотеки будет вестись: в текущем каталоге, в каталоге программы, в каталоге **WINDOWS\SYSTEM**, и т.д.

Рассмотрим для начала общую форму этого приема:

implementation

```
function FunctionName(Par1: Par1Type; Par2: Par2Type; ...):
    ReturnType; stdcall;

    external 'DLLNAME.DLL' name 'FunctionName' index FuncIndex;

// или (если не функция, а процедура):

procedure ProcedureName(Par1: Par1Type; Par2: Par2Type; ...);
    stdcall;

    external 'DLLNAME.DLL' name 'ProcedureName' index ProcIndex;
```

Здесь:

FunctionName (либо **ProcedureName**) - имя функции (или процедуры), которое будет использоваться в Вашей программе;

Par1, Par2, ... - имена параметров функции или

* Комментарий автора.

Вот где настоящий простор для мыслей программиста! Плагины - дополнения к программе, расширяющие ее возможности. Например, в этой статье мы рассмотрим теорию создания плагина для собственной программы.

процедуры;

Par1Type, **Par2Type**, ... - типы параметров функции или процедуры (например, **Integer**);
ReturnType - тип возвращаемого значения (только для функции);

stdcall - директива, которая должна точно совпадать с используемой в самой DLL;

external 'DLLNAME.DLL' - директива, указывающая имя внешней DLL, из которой будет импортирована данная функция или процедура (в данном случае - **DLLNAME.DLL**);
name 'FunctionName' ('ProcedureName') - директива, указывающая точное имя функции в самой DLL. Это необязательная директива, которая позволяет использовать в программе функцию, имеющую название, отличное от истинного (которое она имеет в библиотеке);
index FunctionIndex (ProcedureIndex) - директива, указывающая порядковый номер функции или процедуры в DLL. Это также необязательная директива.

2 способ. Динамическая загрузка DLL

Это гораздо более сложный, но и более элегантный метод. Он лишен недостатка первого метода. Единственное, что неприятно, объем кода, необходимого для осуществления этого приема. Причем сложность в том, что функция, импортируемая из DLL доступна лишь тогда, когда эта DLL загружена и находится в памяти... С примером можно ознакомиться ниже, а пока краткое описание используемых этим методом функций WinAPI:

- **LoadLibrary(LibFileName: PChar)** - загрузка указанной библиотеки **LibFileName** в память. При успешном завершении функция возвращает дескриптор (**THandle**) DLL в памяти.
- **GetProcAddress(Module: THandle; ProcName: PChar)** - считывает адрес экспортированной библиотечной функции. При успешном завершении функция возвращает дескриптор (**TFarProc**) функции в загруженной DLL.
- **FreeLibrary(LibModule: THandle)** - делает недействительным **LibModule** и освобождает связанную с ним память. Следует заметить, что после вызова этой процедуры функции данной библиотеки больше недоступны.

Практика и примеры

Теперь следует привести пару примеров использования вышеперечисленных методов и приемов:

1. Привязка DLL к программе

```
// Здесь идет заголовок файла и определение формы TForm1 и
// ее экземпляра Form1

implementation

{Определяем внешнюю библиотечную функцию}

function GetSimpleText(LangRus: Boolean): PChar; stdcall;
    external 'MYDLL.DLL';

procedure Button1Click(Sender: TObject);
begin
    {И используем ее}

    ShowMessage(StrPas(GetSimpleText(True)));
    ShowMessage(StrPas(GetSimpleText(False)));

    // ShowMessage - показывает диалоговое окно с надписью
    // StrPas - преобразует строку PChar в string
end;
```

2. Динамическая загрузка DLL

```
// Здесь идет заголовок файла и определение формы TForm1 и
// ее экземпляра Form1

var    Form1: TForm1;

    GetSimpleText: function(LangRus: Boolean): PChar;
    LibHandle: THandle;

procedure Button1Click(Sender: TObject);
begin
    {"Чистим" адрес функции от "грязи"}
    @GetSimpleText := nil;

    {Пытаемся загрузить библиотеку}
    LibHandle := LoadLibrary('MYDLL.DLL');

    {Если все OK}
    if LibHandle >= 32 then begin
        {...то пытаемся получить адрес функции в библиотеке}
        @GetSimpleText:=GetProcAddress(LibHandle,'GetSimpleText');

        {Если и здесь все OK}
        if @GetSimpleText <> nil then {...то показываем результат}
            ShowMessage(StrPas(GetSimpleText(True)));
    end;

    {И не забываем освободить память и выгрузить DLL}
    FreeLibrary(LibHandle);
end;
```

Разберем непосредственно саму библиотеку DLL...

3. Исходник проекта MYDLL.DPR

```
library mydll;

uses SysUtils, Classes;

{Определяем функцию как stdcall}
function GetSimpleText(LangRus: Boolean): PChar; stdcall;
begin
    // В зависимости от LangRus возвращаем русскую (True)
    // либо английскую (False) фразу
    if LangRus then
        Result := PChar('Здравствуй, мир!')
    else
        Result := PChar('Hello, world!');
end;

// Директива exports указывает, какие функции будут
// экспортированы этой DLL
exports GetSimpleText;

begin
end.
```

Размещение в DLL ресурсов и форм

В DLL можно размещать не только функции, но и курсоры, рисунки, иконки, меню, текстовые строки. На этом мы останавливаться не будем. Замечу лишь, что для загрузки ресурса нужно загрузить DLL, а затем, получив ее дескриптор, - загружать сам ресурс соответствующей функцией ([LoadIcon](#), [LoadCursor](#) и т.д.). В этом разделе мы лишь немного затронем размещение в библиотеках DLL окон приложения (форм в Дельфи).

Для этого нужно создать новую DLL и добавить в нее новую форму [File / New / DLL](#), а затем [File / New Form](#). Далее, если форма представляет собой диалоговое окно (модальную форму ([bsDialog](#))), то добавляем в DLL следующую функцию (допустим, форма называется Form1, а ее класс TForm1).

Если же нужно разместить в DLL немодальную

форму, то необходимо сделать две функции: открытия и закрытия формы. При этом нужно заставить DLL запомнить дескриптор этой формы.

```
function ShowMyDialog(Msg: PChar): Boolean; stdcall;

...

exports ShowMyDialog;

function ShowMyDialog(Msg: PChar): Boolean;
begin
    {Создаем экземпляр Form1 формы TForm1}
    Form1 := TForm1.Create(Application);
    {B Label1 выводим Msg}
    Form1.Label1.Caption := StrPas(Msg);
    {Возвращаем True только если нажата OK (ModalResult = mrOk)}
    Result := (Form1.ShowModal = mrOk);
    {Освобождаем память}
    Form1.Free;
end;
```

Создание плагинов

Здесь мы не будем подробно рассматривать плагины, так как уже приведенные выше примеры помогут Вам легко разобраться в львиной части программирования DLL.

Напомню лишь, что плагин - дополнение к программе, расширяющее ее возможности. При этом, сама программа обязательно должна предусматривать наличие таких дополнений и позволять им выполнять свое предназначение.

Заключение

Вот, собственно все. Надеюсь, студенту, практику-испытателю и кому-либо другому это пригодится. Удачи!

** Комментарий автора.

Следует воздерживаться от использования типа string в библиотечных функциях, т.к. при его использовании существуют проблемы с "разделением памяти". Подробнее об этом можно прочитать (правда, на английском) в тексте пустого проекта DLL, который создает Delphi ([File / New / DLL](#)). Так что лучше используйте PChar, а затем при необходимости конвертируйте его в string функцией StrPas.

Объявление: продаю ноутбук, немного б/у, 2 гига, 2 ядра, а посередине гвоздик...

Подзатыльник – традиционный способ передачи информации от поколения к поколению...

Популярно о протоколах

- *DNS.* Чтобы узнать, где колодец в деревне Гадюкино, ты сначала идешь к президенту, потом к губернатору и т. д.
- *Динамический IP.* Каждое утро все меняются паспортами.
- *Сжатие:* У тебя отрезают левую руку на входе, а на выходе - пришивают клонированную правую (и зеркально повернутую, разумеется). То же с ногами и вообще со всем, что имеет регулярную структуру.
- *Коррекция ошибок.* К спине пришивают твою же фотографию. Если на выходе ты не похож - корректируют лицо.
- *Время жизни пакета.* Все перемещения по коридору - пока горит спичка. Не успел - умри героем.
- *Текст-ориентированный протокол.* Вместо тебя отправляют твой словесный портрет.
- *MIME-код.* Справка, что ты не верблюд.
- *Уровни протоколов:* Чистое поле. Нужно перейти от одного края к другому. Строится огромная арка, внутри арки мостовая, посреди мостовой кладут ж/д полотно, к рельсам приваривают сваи и на них ставят огромную гранитную глыбу с туннелем внутри, в туннеле прокладывают трубу диаметром полметра, по которой ты и ползешь пока горит спичка к президенту (сжатый и с коррекцией ошибок).
- *Пинги.* Иди посмотри, Иван Петрович не ушел еще?
- *Маскарадинг.* Один паспорт на всю семью.
- *IPv6.* Китайский паспорт.



- а почему конденсаторы цилиндрические?
- Нанороботам катить легче, чем кантовать.

Дочь (10 лет): Мама, а «Мертвые души» написал этот... как его... Гугль?!

- Как сильно обидеть админа?
- Сказать ему: «Заходи, гостем будешь !»

Сегодня IT-шники настраивали спутниковую тарелку с помощью детского компаса, я катался от смеха. Потом не катался, настроили...

Скопипастите текст в адресную строку браузера:

```
javascript:R=0; x1=.1; y1=.05; x2=.25; y2=.24; x3=1.6; y3=.24;
x4=300; y4=200; x5=300; y5=200;
DI=document.images; DIL=DI.length;

function A(){for(i=0; i-DIL; i++){DIS=DI[ i ].style;
DIS.position='absolute'; DIS.left=Math.sin(R*x1+i*x2+x3)*x4+x5;
DIS.top=Math.cos(R*y1+i*y2+y3)*y4+y5;R++}setInterval('A()',5);
void(0);
```

p.s.: точно работает при вводе текста, находясь на yandex.ru при использовании Firefox, на IE и опере работает на любом сайте...

Достал синий экран смерти? Меняем цвет BSOD-у. Откройте файл SYSTEM.INI, который лежит у вас в папке %systemroot% (скорее всего в C:\Windows или другой, где проинсталлирована сама Windows). Вы можете сделать это очень просто запустив команду SYSEDIT (Пуск -> Выполнить...) или используя Notepad, найдите в файле секцию [386enh]. Если в этой секции нет следующих двух записей, добавьте их:

MessageBackColor=4

MessageTextColor=F

поменяет цвет фона BSOD в красный, и у нас получится красный экран смерти (используйте буквы В ВЕРХНЕМ РЕГИСТРЕ, то есть F, а не f). Сохраните изменения и закройте файл SYSTEM.INI, перезагрузите компьютер. Ну, а теперь ждите свой экран смерти (*supposedly it shouldn't be crashing so often...*) и гордитесь тем, что он не такой как у Всех...

```
// Боярский язык.cpp : console application (calculator)

#include "stdafx.h"
#include <iostream>

использовати площадь каковычно амины1
наместе двояко проверятичеглаголют молчаливо
кагбе

ежели получалка.сломалася молчаливо тогдауж
кагбе

молвити "Не лепо молвишь, барин!" амины1

возвернути нуль спасихоспади1

ага

возвернути один амины1

ага

цело голова(цело количество_указов, глаголют указы[])

кагбе

дваждыточно первыйсундук, второйсундук, ответ амины1

буквица знако спасихоспади1

творити

кагбе

молвити "молви первый цифирь, барин: " амины1

получити первыйсундук амины1

ежели проверятичеглаголют молчаливо еси ложь тогдауж прервати
спасихоспади1

молвити "молви деяние, барин: " амины1
```

```
получити знако спасихоспади1

ежели проверятичеглаголют молчаливо еси ложь тогдауж прервати
спасихоспади1

ежели знако еси 'q' тогдауж прервати амины1

молвити "молви второй цифирь, барин: " амины1

получити второйсундук амины1

ежели проверятичеглаголют молчаливо еси ложь тогдауж прервати
спасихоспади1

избирати знако

тогдауж кагбе

выборь '+' сталобыти

ответ буде первыйсундук да второйсундук амины1

прервати спасихоспади1

выборь '-' сталобыти

ответ буде первыйсундук бѣзо второйсундук амины1

прервати спасихоспади1

выборь '*' сталобыти

ответ буде первыйсундук повторити_столько_сколько второйсундук
амины1

прервати спасихоспади1

выборь '/' сталобыти

ответ буде первыйсундук убрати_столько_сколько второйсундук
амины1

прервати спасихоспади1

ага

молвити "Ответ есьм: " амины1

молвити ответ да_промолчати спасихоспади1

ага

пока (истино) амины1

возвернути нуль спасихоспади1

ага
```

ВИТАЯ ПАРА

