



[ПРОграммист]

Программирование и алгоритмизация.
Для молодых и активных людей.

Введение в Scheme

Управлению
шаговым двигателем
через LPT порт

Защита от спама в

phpBB
CAPTCHA

Практика ремонта
компьютерной
техники

Энкодер датчика
PDF на ПЛИС

И многое другое...

Издается с марта 2010. Выходит ежемесячно
№4, июнь 2010 г.

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

Дизайн и верстка:
Егор Горохов, Сергей Бадло

Авторский состав:
Utkin, Владимир Дегтярь, Arigato,
Алексей Шульга, Дмитрий Дмитренко,
Руслан Аблязов, 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 «Радиолюбитель»
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, 59 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 1 с.0x07

АЛГОРИТМЫ. ГРАФИКА В DELPHI

Как работать с графикой на канве в среде Дельфи. Урок 7-8 с.0x13

ДЕЛИМСЯ ОПЫТОМ

Кое-что о ремонте с.0x16

ЛАБОРАТОРИЯ

LPT порт. Компьютер в роли контроллера. Часть 2 с.0x1F
Энкодер датчика PDF на ПЛИС. Часть 1 с.0x27
Защита от спама в форумах phpBB2. Captcha с.0x2E
Основы анимации в WEB на JavaScript с.0x31

ИГРОВАЯ ПЛОЩАДКА

FORTRESS2. Создание лучшего бота - приз 5000 руб с.0x33

ЮМОР

Фразеологизмы, хохмы, загадки с.0x37

От редактора. Здравствуйте, дорогие наши читатели. Добро пожаловать в июньский выпуск журнала «ПРОграммист» от клуба программистов www.programmersforum.ru. Вот уже неслышно и размеренно наступило лето и нам исполнилось четыре месяца. Возраст можно сказать младенческий, чего совсем нельзя сказать о материалах сегодняшнего номера.



В этом выпуске вас ожидает... заключительная часть серии уроков по графике и практический материал по управлению шаговым двигателем через LPT порт от Владимира Дегтяря. Как всегда, на острие новых технологий, наш старина **Utkin**. Сегодня не без его помощи вы познакомитесь с одним из функциональных языков программирования – Scheme. Новую рубрику «Делимся опытом» открывает Дмитрий Дмитренко со своей статьей по практике ремонта компьютерной техники. Основы и примеры, изложенные в материале пригодятся многим, кто с техникой на Вы. Рубрика «Лаборатория» набрала немалый вес. Разработку сайта нашего журнала и внедрение новых «вкусностей» продолжают Егор Горохов и Алексей Шульга. Наш добрый друг и товарищ по клавиатуре **Arigato** расскажет начинающим о том, как защититься от спама в форумах phpBB2. И конечно-же, не оставит равнодушными разработчиков промышленной автоматики Сергей Бадло с новым материалом по аппаратной реализации энкодера датчика приращений на ПЛИС...

Рубрики журнала (плавающие)

- Новости программирования (новые языки, концепции, среды)
- Отдел тестирования
- Общие вопросы (вопросы правового использования, личные мнения и т.д.)
- Алгоритмы (без привязки к языку и платформе)
- Юмор (специфические хохмы программеров)
- 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 (код, интернет-ресурсы, литература)
- текст статьи 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, BMP или TIF
- разделы статьи отделять двумя <ENTER>
- не используйте табуляцию и лишние пробелы без необходимости
- 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](#)

Замечание редактора обязательно должно вызывать реакцию автора. Если автор не согласен с комментарием или исправлением редактора, он должен разъяснить вопрос в виде комментария к статье на ящик-копилку. Если редактор согласен с тем, что ошибся, вопрос снимается. Если редактор не согласен, решает группа.

С уважением, Редакция

Все уже давно привыкли к тому, что сфера высоких технологий стала неотъемлемой частью нашей жизни. И сегодня в рубрике вам тоже не придется скучать. Вы узнаете: последние новости современной радиоэлектроники, малоизвестные факты и загадки истории. Итак, приступим...



Сергей Бадло

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

Детектор лжи с электрошокером будет служить отличным подарком любителям розыгрыша. Достаточно просто положить руку на детектор лжи и закрепить ее манжетой, которая входит в комплект. Задайте несколько вопросов испытуемому и в случае нечестного ответа электрошокер не сильно, но ощутимо даст знать, что вас пытались обмануть. Этот гаджет показывает

уровень лжи на специальной шкале и имеет отличную память, поэтому отъявленного лжеца будет бить током все сильнее с каждым неправильным ответом.



Со священником по Интернету можно пообщаться в Актобе (Казахстан). Для этого достаточно сесть за компьютер и зайти на сайт местного храма святого Владимира. Настоятель храма отец Дмитрий до недавнего времени не знал даже, как подойти к компьютеру. Теперь же священник в свободное от молитв время активно повышает компьютерную грамотность. «Я и не думал, что буду интересоваться высокими технологиями. Всегда считал, что лучше книгу прочитать. Но нужно идти в ногу со временем. Поэтому и пришла идея организовать электронную проповедь. Она предназначена не только для верующих, но и для тех, кто далек от веры. Самое интересное, что ничего подобного на данный момент в Казахстане нет», — рассказывает отец Дмитрий. В церкви признаются, что иногда поступают неожиданные

вопросы, ответить на которые непросто. Но здесь не избегают диалога и стараются наставлять виртуальных прихожан на путь истинный. В планах священников открыть на сайте форум и сделать раздел, где будут помещать фото детей из местных приютов. Так здесь хотят устроить судьбу сирот, усыновить которых, возможно, изъявят желание верующие люди.

Самые старые батарейки? Любопытную загадку задали ученым небольшие глиняные сосуды, отысканные археологами после 2-й

мировой при раскопках старого античного городка Селевкия (на местности Ирака). Вазы любопытны тем, что в них отлично сохранились встроенные медные цилиндры со стальными сердечниками. Для пайки цилиндров употреблялся сплав свинца и олова в той же пропорции, что применяется и в современной электротехнике. Из предположения, что это остатки электролитических частей батареи, были изготовлены экспериментальные модели и залиты электролитом (медным

купоросом). На клеммах появилось напряжение около 6 вольт. Для чего шумеры употребляли эти источники тока (возраст селевкийских ваз насчитывает 2000 лет) ученым узнать так и не удалось. Однако разумно предположить, что раз существовали

источники электроэнергии, а ведь при соединении их в батареи можно получить сколь угодно высокое напряжение, то должны были существовать устройства, использующие электрический ток.

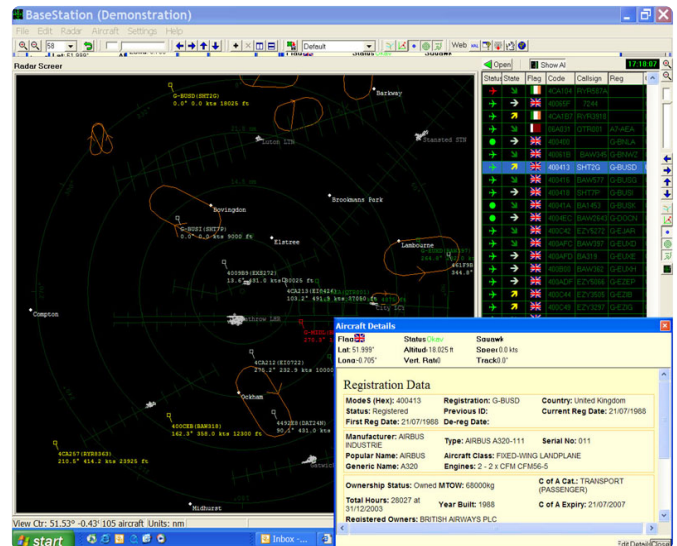
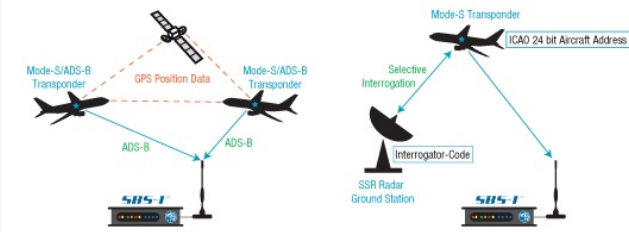
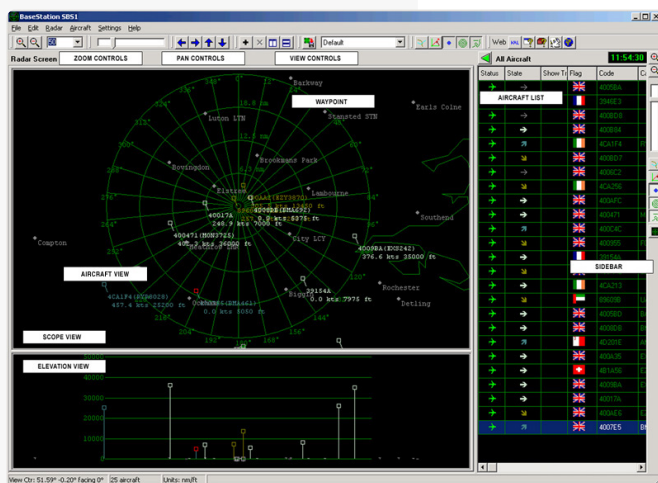




Виртуальный радар SBS-1
представляет собой
недорогой и легкий
приемник, работающий в
режиме S/ADS-B*,
декодирующий сигналы
ответчика на частоте 1090

Automatic Dependent Surveillance Broadcast (ADS-B)
Aircraft broadcast position, altitude, speed etc

Secondary Surveillance Radar (SSR) Mode-S Surveillance
Altitude and identity data available


[illegible]

МГц, поступающие от воздушного судна. Система SBS-1 воспроизводит эту информацию на экране виртуального радиолокатора и дает возможность в реальном времени отслеживать местонахождение воздушных объектов.

* Комментарий редакции

(S) - ответчик RBS "режим S" с передачей опознавательного индекса воздушного судна и данных о барометрической высоте. По сути, расширение вторичного локатора, однако работать будет только там, где поблизости ВРЛ с режимом S. А иначе бортовой ответчик не ответит в режиме S, если его не запросят с земли. Что касается режима 1090ES, то его иностранцы над Россией, как правило, выключают, т.к. в России официально он не используется. Воздушную обстановку в реальном времени можно посмотреть тут <http://radarspotters.eu>

Газовые паяльники Weller работают на бутане – очищенном газе для заправки газовых зажигалок. Рекомендуется применять очищенный газ производства фирм Ronson, Braun, Weller. Заправка газом осуществляется через стандартный клапан и занимает 7-10 секунд времени. К примеру, Rygoron WP 60K – недорогой портативный газовый паяльник с ручным кремниевым поджогом, малым временем нагрева и прозрачным окошком для контроля наличия газа.



Длительность работы от одной заправки ~60 минут. Рабочая температура в режиме паяльника ~500°С; в режиме фена горячего воздуха ~650°С. В комплект поставки входит паяльник, защитный колпачок, сопло Ø 4,7 мм для работы в режиме фена горячего воздуха и жала для паяльника: игольчатое 0,5 мм и скошенное под углом 45° типа «горячий нож» шириной 5 мм.



Танталовые конденсаторы



большой емкости были представлены компанией AVX. Конденсаторы PulseCap имеют два типоразмера: 14.5x7.5x2 мм и 7.3x6.1x2 мм.

Диапазон возможных номиналов от 1000 мкФ до 3300 мкФ. Компоненты рассчитаны на напряжение от 4 В до 1 В.

Немногие знают, что «бесплатное электричество» есть практически в каждом многоквартирном доме**. Речь идет об использовании «0» провода и заземления, скажем шахты лифта. Дело в том, что всегда существует ток в нулевом проводнике, из-за разности мощностей нагрузок, подключенных к фазам. При этом возникает дисбаланс фаз, которым можно воспользоваться. Естественно, эта разность будет колебаться в районе от 0-7В, в зависимости от времени суток (от ваших соседей). Но и ток приличный, до 1.5А. Чего вполне достаточно, при наличии защитной схемы, для питания аварийного освещения и иных домашних целей.

** Комментарий редакции

Более подробно о варианте применения смотрите на <http://rnxp.radioliga.com/cnt/s.php?p=ib5.djvu>

Кварцевый генератор NBXSBA025 компании ON Semiconductor, предназначен для применения в 2.5/3.3В LVPECL приложениях тактовой генерации. В устройстве применяется высокочастотный кварцевый резонатор, работающий на основной моде, и умножитель частоты с фазовой автоподстройкой, обеспечивающие рабочую частоту 425 МГц, сверхмалый джиттер и низкий уровень фазовых шумов. NBXSBA025 поставляется в корпусе SMD (CLCC), габаритные размеры 5x7 мм.

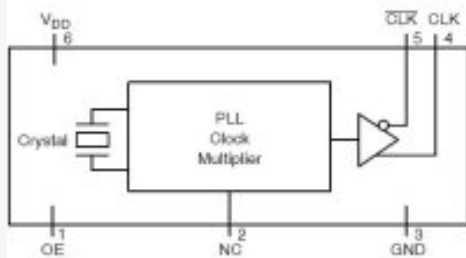


Figure 1. Simplified Logic Diagram

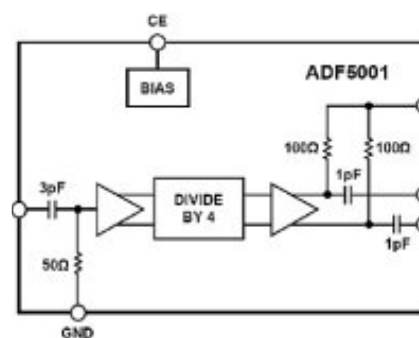


Мощный

нейроинтерфейс на основе функционального магнитно-резонансного сканера (fMRI), разрабатывают специалисты корпорации Intel, а также ученые университетов

Питтсбурга и Карнеги-Меллона. В первых тестах система показала высокую точность распознавания мыслей – людям предлагали загадать два слова, а затем система определяла эти слова с точностью порядка 90%. На данный момент у системы есть один серьезный недостаток – ее можно использовать только рядом с многотонным аппаратом fMRI, который в ближайшем будущем вряд ли станет мобильным.

Предварительные делители частоты



до 18ГГц от Analog Devices серии ADF500x (ADF5000 – делитель на 2, ADF5001 – делитель на 4 и ADF5002 – делитель на 8), предназначены для применения в различных приложениях, включая системы связи «точка-точка», терминалы VSAT и микроволновые системы связи.

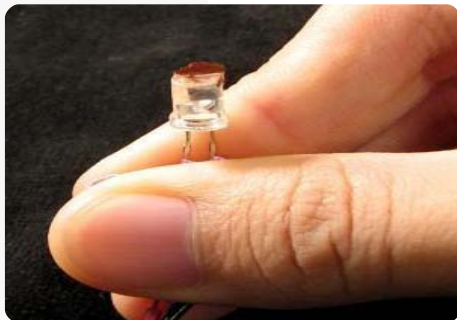
В активном режиме ток потребления составляет 30 мА, в режиме ожидания 7 мА. Спектральная плотность фазового шума составляет 150 дБн/Гц. Компоненты поставляются в корпусах LFCSP, габаритные размеры 3x3 мм. Цена от \$6.97 в партии 10000 штук.

Немногие знают, что монитор с технологией PC-over-IP SyncMaster 930ND можно подключать к ПК, используя IP-сеть. Таким образом, появляется возможность работать на удаленном ПК, выводя изображение на нужный монитор. Монитор оснащен четырьмя портами USB, разъемами HD Audio и DVI-out для подключения еще одного монитора. Совместим с ОС Windows Vista и XP.



Мобильники с детекторами токсинов

предложены учеными Калифорнийского университета (Сан-Диего). Эта технология позволит телефонам обнаруживать и отслеживать все опасные вещества и поля, включая радиацию, загрязняющие вещества и даже рак. Детектор состоит из нескольких модулей:



ключевым элементом является кремниевый сенсор, обнаруживающий присутствие токсинов в воздухе и изменяющий свой цвет в зависимости от конкретного вещества. Датчик, созданный из хлопьев пористого кремния, меняет цвет при взаимодействии с конкретными химическими веществами. Манипулируя формой пор, исследователи могут заставлять отдельные пятна на поверхности кремния реагировать на специфические особенности химического вещества. Сенсор размещается у камеры с макролинзой, эта камера контролирует динамику сенсора и выводит результаты на дисплей телефона.

Важная особенность этой технологии заключается в том, что количество датчиков, содержащихся в одном из массивов, определяется разрешением камеры мобильного телефона, что позволит различать миллион различных пятен. Таким образом, не требуется разрабатывать миллионы отдельных датчиков, достаточно только одного. Это существенно упрощает процесс производства. Исследовательская группа успешно завершила первый этап разработки датчиков и начала работы по созданию прототипа, который будет встраиваться в мобильный телефон.

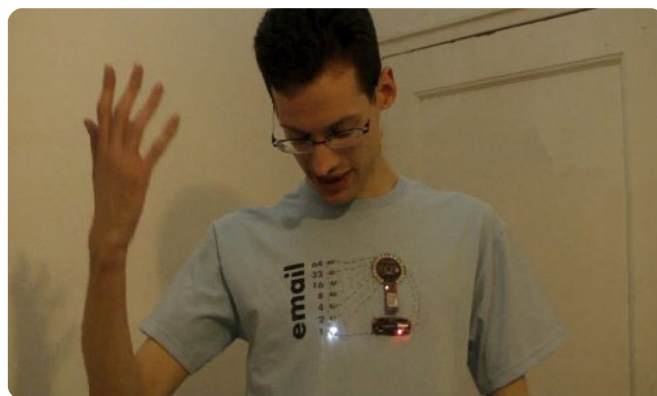
Систему аутентификации по

электрокардиограмме представила компания IDesia. Созданная система опирается на тот факт, что образцы электрокардиограмм уникальны. По словам разработчиков, точность их системы составляет около 99%. К преимуществам электро-кардиограммной аутентификации относятся низкая стоимость и

компактные габариты. Малая себестоимость объясняется простотой устройства – для измерений сигналов сердца достаточно всего двух электродов (один для левого пальца и один для правого). В отличие от других биометрических систем, изобретение IDesia не требует каких-либо специализированных сенсоров или камеры. Небольшие габариты изделия позволяют встроить его, например, в смарт-карту.

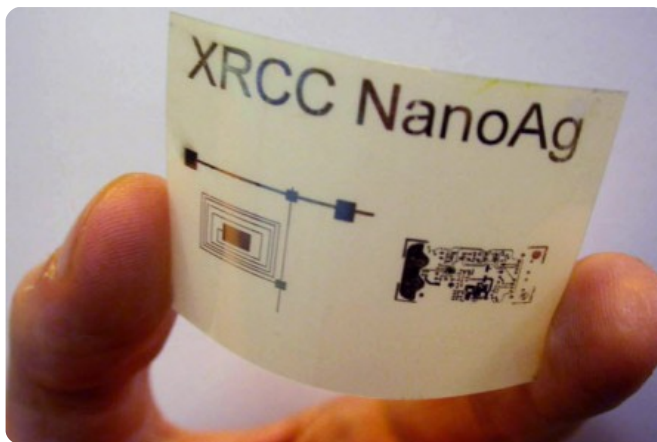
Очередное забавное изобретение – футболка,

которая сообщает хозяину, что на его почтовый ящик пришло письмо, и даже показывает, сколько именно писем пришло (1, 2, 4, 8, 16, 32 или 64). Эта футболка для тех, кому мало всевозможных гаджетов, чтобы быть на связи. Футболка содержит в себе пару светодиодов,

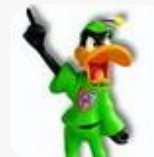


токопроводящую нить и модуль блютуз, чтобы связываться с телефоном на базе Android, который в свою очередь имеет доступ к электронному ящику.

Новый вид чернил, которые могут проводить электричество, разработали в компании Xerox. Таким образом, при нанесении их на пластик, пленку или ткани можно создавать рабочие электрические цепи, поддающиеся сгибанию.



Приветствую читателей нашего журнала. Сегодня мы с вами рассмотрим краткое описание функционального языка программирования *Scheme*. Данный материал рассчитан на программистов, имеющих навыки программирования в императивном стиле...



by **Utkin** www.programmersforum.ru

Бутерброд всегда падает маслом вниз. Если нет - значит, его не с той стороны намазали...

/ неизвестный автор

В 1970 году Гай Стил (Guy L. Steele) и Джеральд Сассман (Gerald Jay Sussman) из Массачусетского технологического института начали работу на новым языком программирования. Тогда Джеральд Сассман уже имел ученую степень, а его помощник Гай Стил являлся студентом данного института. В 1975 году работы были закончены и получили свое дальнейшее воплощение в принципах работы СБИС (в 1978 году) и CAD-системах. Идеи, положенные в эти работы, позже использовались для конструирования элементов космических аппаратов и проведения экспериментов в космосе (с использованием элементов искусственного интеллекта). В 1984 году большое количество версий в разных институтах заставило разработчиков выработать единый стандарт (при участии Массачусетского технологического института и университета Индианы). Такие серьезные сферы дали хороший старт языку, однако его распространение сдерживалось низкой скоростью интерпретации. В 90-х годах прошлого века были разработаны новые технологии компиляции, что позволило повысить эффективность Scheme путем компиляции в байт-код и машинные коды (хотя экспериментальные компиляторы со Scheme существовали и ранее). На данный момент производительность Scheme в целом выше, чем у Java (учитывая, что языки разных стилей, то такое сравнение условно).

Введение

Scheme («скиим») – это функциональный язык программирования, диалект Лиспа. Особенности:

разумный минимализм, интерактивность, однородность синтаксиса (листовые списки), высокий уровень абстрагирования, возможность писать программы оторванные от конкретной платформы, «строгий» язык (требует придерживаться определенного стиля), позволяет использовать разные парадигмы программирования и много чего еще особенного.

В дальнейшем будем рассматривать язык на примере **PLT Scheme**. Данная среда выбрана не случайно (имеются десятки других). Она бесплатна, имеет широкое распространение, большой набор библиотек и позволяет компилировать программы в exe-файлы. Взять можно отсюда: <http://www.plt-scheme.org>. IDE в PLT Scheme называется DrScheme, имеет минималистический вид, однако содержит практически все, что необходимо для работы программиста. Редактор имеет два окна – первое для записи программы, второе информационное и окно интерактивного режима – команды языка, введенные в него, исполняются немедленно (см. рисунок):

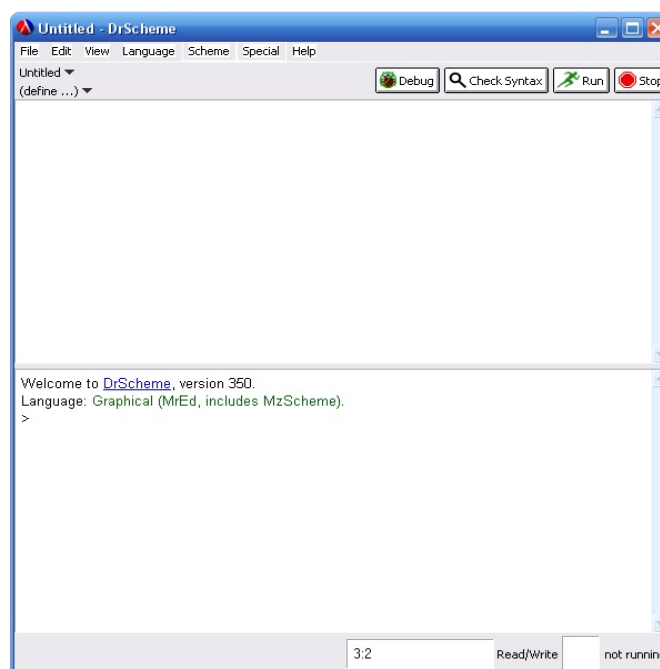


Рис. Среда drScheme

Также в комплекте имеется MrEd – система для создания переносимого графического интерфейса пользователя. Далее будет следовать краткое описание языка, настолько, насколько это возможно в рамках статьи. Все примеры, приведенные ниже можно сразу опробовать в окне интерактивного режима.

Синтаксические элементы и что еще обязательно нужно знать

Прежде чем, начать описание синтаксических элементов следует обратить внимание на некоторые вещи. Все вычисления, в процедурах, аргументах всегда осуществляются слева направо. Язык имеет поддержку типов, однако, типизация скрытая (динамический контроль типов). Объекты (включая процедуры – сама программа также может являться объектом данных) имеют неразрушаемую природу, однако переполнение памяти не возникает за счет того, что объекты могут быть перезаписаны, в случае, если больше не будут участвовать в вычислительном процессе. На данный момент все реализации Scheme используют хвостовую рекурсию, это позволяет выделять на выполнение рекурсионных вызовов ограниченный размер памяти. Такие вызовы позволяют не использовать специализированные операции цикла (могут быть получены за счет автоматического преобразования хвостовой рекурсии в итерации). Все функции являются самостоятельными объектами (могут создаваться динамически, сохранены в структурах данных, возвращены как результаты функций и т.д.). Примером таких языков являются Common Lisp и ML. Параметры процедур всегда передаются по значению (в противовес Haskell, который позволяет откладывать вычисление параметра, до тех пор, пока он не потребуется явно).

Модель арифметических вычислений создана таким образом, чтобы максимально не зависеть от конкретных аппаратных платформ. В Scheme каждое целое число является рациональным, каждое рациональное число является реальным, а каждое реальное к тому же является еще и

комплексным. Поэтому разница между различными типами чисел, возведенная в ранг абсолютизма в императивных языках, здесь не проявляется.

Программа является списком, элементы списка отделяются друг от друга пробелом, сам список заключен в скобки. Операции представляется списком, в котором символ операции (который, в сущности, является именем функции) всегда занимает первое место в списке (префиксная форма или польская нотация):

(+ 2 2)	Результатом будет 4.
(+ 2 (* 2 2))	Результатом будет 6.

Особенностью такой записи является тот факт, что операция может быть применена к неограниченному количеству входящих параметров, можно написать и (+ 2 2 2 2 2 2 2 2 2 2) (сравните 2 + 2). Это приводит к тому, что программы, написанные на Scheme, могут являться данными для других программ написанных на Scheme. Для функциональных языков программирования является традицией писать интерпретаторы/компиляторы с них на этих же языках программирования.

PLT Scheme не делает разницы между идентификаторами, набранными в разных регистрах. В тоже время правила образования идентификаторов может различаться для разных версий и реализаций языка, однако есть общее требование – идентификатор не должен начинаться с цифры. Примеры идентификаторов:

```
lambda
q
list->vector
soup
+
V17a
<=?
a34kTMNs
the-word-recursion-has-many-meanings
```

Для идентификаторов разрешено использовать символы ! % \$ & * +-./: <=>? ^ _ ~. Можно использовать, например, такой идентификатор ---->, однако по соглашению фрагмент -> делит

идентификатор на две части и говорит о преобразовании типа, например `list->vector` для функции может трактоваться как преобразование списка в вектор (элементы которого являются элементами списка). Помимо пробелов и переходов на новую строку допускается также использовать символы табуляции, все эти элементы обычно используются для улучшения восприятия текста программы и в качестве разделителей лексем языка.

Одним из важных понятий языка является понятие представления объектов. Если в программе, это всего лишь поток символов, то внутренне объект может представляться совершенно иным способом. Далее, представление объектов не обязательно уникально, так число 28 может быть представлено как #e28.000 или #x1c. Список (1 2) можно представить как (01 02) или (1 (2.))) Здесь () является пустым списком. Однако, список (+ 1 2) не является эквивалентом числа 3, это список, состоящий из трех элементов (результат вычисления которого является числом 3). С одной стороны, это может путать на начальном этапе знакомства с языком, но с другой это источник его мощи, данные могут являться программой, программа может являться данными.

Переменные перед использованием должны быть объявлены: (define x 28) В данном случае мы определяем переменную x со значением 28. В дальнейшем допускается неоднократное переопределение (например, (define x 38)).

Точка с запятой (;) служит для обозначения комментария, который продолжается до конца строки и является неисполняемым оператором языка. Поэтому комментарии посреди лексемы языка не допускаются. Приведем небольшой пример:

```
;; процедура FACT вычисляет факториал
;; из неотрицательного целого числа.
(define fact
  (lambda (n)
    (if (= n 0)
        1 ;Base case: return 1
        (* n (fact (- n 1))))))
```

[illegible]

Это говорит о том, что строгий диапазон для чисел не установлен и ограничен только объемом памяти и Вашим временем (кстати, считает за вполне приемлемое время), на операции над ними (справедливо для целых чисел).

Существует соглашение, по которому предикаты (элементы, возвращающие значения Истина/Ложь) должны именоваться с вопросительным знаком на конце.

Ну и еще что нужно знать об основных синтаксических элементах: *, + - могут использоваться в идентификаторах (например, предыдущем примере мы могли бы обозначить идентификатор функции как fact+ или fact+1).

Круглые скобки **()** используются для обозначения списков (напомню, программа представляет собой один сплошной список, параметры функции - список и т.д.).

Одинарная кавычка (') используется для абсолютного значения, которое вычислять не требуется (допускается второе обозначение данной функции - `quote`). Например, имеется элемент списка (x) и имеется функция с аналогичным именем, чтобы в списке x не вычислялся, используется данная функция. Чтобы было понятней, объявим такую функцию: `(define (prostoX x) (quote x))`

Вызовем ее:

(prostoX 100)	Резултат будет x, а не 100
(quote a)	Резултат a
(quote #(a b c))	Резултат #(a b c)
(quote (+ 1 2))	Резултат (+ 1 2), а не 3
'(quote a)	Резултат (quote a)

Можно заставить Scheme вычислять частично, для этого используется запятая:


```
`(list ,(+ 1 2) 4)    Результат (list 3 4)
```

то есть выражение перед запятой `(+ 1 2)` было вычислено. Более сложный пример:

```
(let ((name 'a)) `(list ,name ',name))
Результат (list a (quote a))
```

Числовые, символьные, строковые и булевые константы «квотировать» не допускается. Символ двойной кавычки (`"`) используется для ограничения строковых констант. Символ обратного слеша (`\`) используется в строковых константах. `[] { } |` Левые и правые квадратные скобки, изогнутые скобки и вертикальный штрих зарезервированы для возможных будущих расширений языка. Символ шарп (`#`) используется для множества целей в зависимости от символа, который немедленно следует за этим:

```
#t #f          - обозначение булевых констант;
# \           - вводит символьную константу;
#(            - вводит векторную константу;
#e #i #b #o #d #x - используются для обозначения чисел.
```

Объявление процедур:

```
(lambda <formals> <body>)
<Formals> - список формальных параметров
<Body>    - последовательность из одного или более выражений.
```

Лямбда-выражение считается процедурой, не имеющей имени. Результат(ы) последнего выражения в теле будет возвращен как результат(ы) вызова процедуры:

```
(lambda (x) (+ x x))    Есть описание процедуры без имени,
                        имеющий один входящий параметр и
                        возвращающий удвоенное значение
                        переменной x.
((lambda (x) (+ x x)) 4) Результат 8 (последнее выражение,
                        передано в процедуру)
```

Определим такую функцию:

```
(define reverse-subtract
  (lambda (x y) (- y x)))
```

Ее вызов с параметрами `(reverse-subtract 7 10)` Результат 3 (для последнего выражения происходит обмен параметров в списке). Таким образом, сначала определяется безымянная процедура с двумя входящими параметрами `x` и `y`.

Ее задача вычитать из `y` `x`. Далее, ей просто присваивается имя `reverse-subtract`, которую мы и вызвали с параметрами 7 и 10. `x` получает значение 7, `y` соответственно 10, затем мы вычитаем из `y` `x` тело `(- y x)`, то есть `(- 10 7)`. Результатом будет 3.

Определим следующую функцию:

```
(define add4
  (let ((x 4))
    (lambda (y) (+ x y))))
```

Ее вызов с параметрами `(add4 6)` Результат 10. Обратите внимание – внутри функции определяется локальная переменная `x`, имеющая значение 4. Данная переменная существует только в рамках `add4`.

Lambda удобно использовать для образования хвостовой рекурсии – один из важнейших элементов функционального программирования. Для императивного программирования хвостовую рекурсию можно выразить через цикл. Также lambda в связке с `define` можно рассматривать как аналог механизма интерфейсов ООП.

Допускается и короткая форма функций, без использования процедур `lambda`: `(define (имя параметры) (тело_функции))`. Определим функцию возведения в квадрат: `(define (square x) (* x x))`. Вызовем ее `(square 10)`, в результате получим 100. Вызовем ее еще раз `(Square 10.5)`, в результате получим 110.25. Код может следовать сразу друг за другом без всяких приведений типов и никаких фокусов с параметризацией здесь не требуется совершенно. Хотелось бы на этом заострить внимание: то, что в последнее время привносится как новая веха ООП – параметризация, существует в функциональном программировании чуть ли не с момента его возникновения. Однако в функциональном программировании оно представлено на совершенно ином уровне.

Еще один момент, касающийся идентификаторов – в PLT Scheme все идентификаторы могут содержать дополнительные символы. Это значит,

что вместо `square` можно смело определить функцию КВАДРАТ, и вообще можно переписать практически все ключевые элементы языка, используя национальный алфавит.

Условные выражения представляются в следующих формах:

```
(if <тест> <следствие> <альтернативный_вариант>)
```

Здесь все просто и аналогично другим языкам программирования:

```
(if (> 3 2) 'да 'нет)  Результат да
                        (обратите внимание на одинарные кавычки)

(if (> 2 3) 'да 'нет)  Результат нет

(if (> 3 2)
    (- 3 2)
    (+ 3 2))          Результат 1
                        (3 больше 2? Если да, то выполним
                        (- 3 2), то есть получим в итоге 1)
```

Можно использовать также `cond`:

```
(cond ((> 3 2) 'больше)
      ((< 3 2) 'меньше))  Результат больше

(cond ((> 3 3) 'больше)
      ((< 3 3) 'меньше)
      (else 'равно))     Результат равно
```

Общая форма: `(cond выражение1 выражение2 ... выражениеN)`, где выражение состоит из `((условие) (действие))`. Ветвь `else` будет выполнена в случае, если ни одно из предыдущих условий не было выполнено. `Cond` аналогична множественному использованию `if` (изначально конструкции `if` не было, и все условия обсчитывали через `cond`). В качестве задания: попробуйте выразить `if` через использование `cond`.

Давайте рассмотрим такой фрагмент программы:

```
(define (PAIR (a b))
  (lambda (msg)
    (case msg
      ; конструкция case для каждого значения msg
      ; выполняет соответствующее действие
      ((first) a)
      ; если сообщение — символ first, вернуть a
      ((second) b))))
```

В зависимости от `msg` будет возвращено соответствующее значение, то есть фактически полностью соответствует `case` из императивных языков (таких как C++ или Дельфи). `Case`

аналогично `cond` поддерживает `else` – ветвь которая будет обязательно выполнена в случае, если ни одна из предыдущих выполнена не была.

Можно составлять сложные условия за счет грамотного использования `and` и `or`:

```
(and (= 2 2) (> 2 1))  Результат #t (истина). 2=2 и 2>1? Истина

(and (= 2 2) (< 2 1))  Результат #f (ложь). 2=2 и 2<1? Ложь
(and 1 2 'c '(f g))   Результат (f g) (список не вычисляется
                        из-за одинарных кавычек)

(and)                  Результат #t
(and (< 2 2) (= 2 2)) Результат #f
```

Выражения оцениваются слева направо до первого ложного утверждения. Если все утверждения верны, то возвращается последний входящий параметр функции (у нас это пример со списком `(f g)`). Если `and` не имеет входящих утверждений, будет возвращено `#t` (истина):

```
(or (= 2 2) (> 2 1))  Результат #t
(or (= 2 2) (< 2 1))  Результат #t
(or #f #f #f)         Результат #f
```

Выражения оцениваются слева направо до первого истинного утверждения (оставшиеся не рассматриваются). Если все выражения ложны, будет возвращено последнее из них. Теперь немного о присваивании переменных. Вообще-то, в чистом функциональном программировании переменных вообще существовать не должно, но подавляющее большинство функциональных языков программирования признают пользу существования переменных и вводят различные механизмы для их использования. Присваивание в PLT Scheme осуществляется таким образом:

```
(define x 2)
(set! x 4)
(+ x 1)  Результат 5.
```

Ничто нам не мешает переопределить переменную `x` через `define`, но все же необходимо помнить, что `define` не эквивалентно `set!` (хотя в нашем случае и приводит к одному результату). Разница заключается в том, что `define` связывает идентификатор переменной с ее новым значением (предыдущее значение переменной будет

уничтожено в результате сборки мусора), а `set!` присваивает значение уже существующей. Вообще, следует учитывать, что имя переменной и ее значения связаны не напрямую между собой. В частности, это проявляется в том, что переменная может существовать по имени, без значения (проявляется, например, при операции `quote`). При этом нет точных рамок между именем переменной и именем функции. И через `define` имя переменной можно переопределить как имя функции.

Теперь рассмотрим более подробно конструкцию `let`. Собственно у нее есть два родственника `let*` и `letrec` с тем же синтаксисом (`let` ((определение_переменной1) (определение_переменнойN)) (тело)). Разница в них только в уровнях видимости связанных ими переменных. Это позволяет строить блочные структуры на манер принятый в структурном программировании. Короткая форма такова: (`let` (определение переменной) выражение), где определение переменной выглядит все в том же списочном стиле (переменная значение):

```
(let ((x 2)) (* x 2))  Результат 4.
```

Создаем переменную `x=2`, умножаем на 2 и передаем значение. Сам `x` при выходе из данного выражения будет разрушен. `Let` позволяет создавать сколько угодно переменных, действие которых будет распространяться исключительно на тело определения (при этом в рамках одного определения не допускается использовать несколько переменных с одинаковыми идентификаторами):

```
(let ((x 2) (y 3))
  (* x y))  Результат 6

(let ((x 2) (y 3))
  (let ((x 7)
        (z (+ x y)))
    (* z x)))  Результат 35.
```

В первом блоке определим `x=2`, `y=3`, переходим к исполнению тела `let`. Создадим второй `x` со значением 7, затем определяем еще одну переменную второго блока `z`, которая будет равна `x+y` (число 5) первого блока (поскольку `x` второго

блока имеет силу только в теле второго блока, но не на этапе определения переменных этого же блока). Приступаем к исполнению тела второго блока – умножаем 5 (в данном случае, переменная `z`) на 7 (переменная `x` второго блока). Результат 35.

Как видно из примера, допускается неограниченная глубина вложенности `let`, внутри каждого определения допускается свободное повторение идентификаторов по отношению к другим `let`. В Scheme `let` часто используется как аналог программных скобок (`{-}`, `begin-end`) императивных языков программирования. Как правило, `let` используется чаще родственных ей `let*` и `letrec`.

Рассмотрим пример с `let*`:

```
(let ((x 2) (y 3))
  (let* ((x 7)
         (z (+ x y)))
    (* z x)))  Результат 70.
```

В первом блоке определим `x=2` и `y=3`. Далее конструкция `let*` определяет переменную `x=7`, действие которой распространяется на все определения переменных справа и на тело `let*`, таким образом, `z=7` (новый `x`)+3 (`y` из первого блока)=10. Приступаем к выполнению тела `let*` - `z` (число 10) умножаем на `x` блока `let*` (число 7). Результат 70.

Что касается `letrec`, ее особенность в том, что переменная получает свое значение не мгновенно, а в момент, когда она будет необходима (соответственно в теле переменная может получать разные значения в зависимости от того выражения, каким она должна инициализироваться). Это удобно в случае многочисленных рекурсионных вызовов:

```
(letrec ((even?
  (lambda (n)
    (if (zero? n)
        #t
        (odd? (- n 1))))))
  (odd?
  (lambda (n)
    (if (zero? n)
        #f
        (even? (- n 1)))))) (even? 88))
```


Данный пример определяет четность числа (`#t`, если четное). Как видно из примера, `even?` может вычисляться в зависимости от входящего числа через рекурсию. Еще один момент – если Вы заметили, то `even?` это должна быть переменная или функция, имеющая булевый тип, а 88 есть число (в рамках Scheme – не важно какое, считайте, что все числа есть комплексные величины). Здесь видна работа `letrec`, для тела вычисляется `even?` определенная не как переменная, а как функция (через `lambda`). `Letrec` часто используют для получения значения переменной через рекурсионные вызовы. Эта конструкция имеет один подводный камень – поскольку все параметры передаются по значению, должны существовать однозначные условия для получения результата. Никаких двусмысленностей в `letrec` быть не должно (могут возникнуть, в частности, при комбинациях родственников `letrec`, `let` и `let*`).

Одной из важных особенностей функциональных языков программирования является концепция функций – иными словами процедуры в обычном понимании в функциональном программировании быть не должно. Выражения и функции получают параметры и передают результат. Однако на практике часто возникают ситуации, когда результат либо не требуется, либо не определен, либо просто не имеет смысла. Самым простым примером является вывод информации на экран. Какой результат может в данном случае получить функция вывода сообщения на экран? В тоже время отсутствие результата (либо возникновение ситуации, когда результат работы функции определить невозможно) может привести программу к краху – следующая функция не сможет получить входящий параметр (либо не сможет определить что это за параметр). Явление, при котором функция проявляет себя каким-либо образом помимо передачи результата, называется побочными эффектами (явлениями). В чистом функциональном программировании побочных явлений быть не должно в принципе (например, побочным эффектом является присваивание переменной какого-либо значения, создание

объектов без их непосредственной передачи (так себя ведет `define`), ввод и вывод данных и т.д.). Специально для решения подобных проблем существует конструкция `(begin (выражение1) ... (выражениеN))`, все вычисления в которой осуществляются последовательно и строго слева направо. Особенностью данной конструкции является результат – он всегда будет возвращен для последнего выражения. Пример:

```
(define x 0)
(begin (set! x 5)
      (+ x 1))          Результат 6 (5+1)

(begin (display "4 plus 1 equals ")
      (display (+ 4 1)))  Результат 5 (4+1)
```

проигнорировав результаты вывода на экран посредством `display` (при этом сама строка будет напечатана). `Begin` также удобно использовать при создании законченных блоков, содержащих несколько выражений (например в `If` или `cond`) как замена `let`, с той лишь разницей, что для данного блока новых переменных не создается (соответственно и не разрушаются).

Рассмотрим итерации. Хотя хвостовая рекурсия может быть преобразована в цикл (при правильном ее составлении), циклы в Scheme существуют также и в общей форме. Один из них:

```
(do ((<переменная1> <инициализатор_переменной1>
      <шаг_приращения1>)
     ...)
    (<тест> <выражение> ...)
    <операторы_языка> ...))
```

Сразу стоит обратить внимание, на то, что приращивать можно не одну переменную и условие для выхода из цикла тоже может быть не одно:

```
(do ((vec (make-vector 5))
     (i 0 (+ i 1)))
    ((= i 5) vec)
    (vector-set! vec i i))  Результат вектор #(0 1 2 3 4),
                             здесь создается вектор и
                             переменная i )

(let ((x '(1 3 5 7 9)))
  (do ((x x (cdr x))
      (sum 0 (+ sum (car x))))
      ((null? x) sum)))  Результат 25,
```

проводиться подсчет списка. Здесь создается две переменные `x` (обратите внимание, переменная цикла инициализируется переменной, объявленной через `let`) и `sum`. Также если рассмотреть конструкцию `let` более внимательно, то можно обнаружить, что с помощью нее также можно формировать итеративные процессы. Для этого определяемую переменную можно представить как функцию (через ее инициализатор) и вызывать ее для каждого упоминания в теле `let` (через условные операторы `cond` или `if`).

Несмотря на то, что все параметры в функции передаются по значению, есть возможность отложить вычисление параметров до момента их действительного использования. Плюсов от этого как минимум два: первое – это увеличение скорости вычислений (при грамотном использовании) и второе – решение задач, которые нельзя вычислить обычным способом. Увеличение скорости возможно, в случае, если в функцию передается несколько параметров и часть из них напрямую влияет на логику работы функции. Допустим, первый параметр функции однозначно говорит о том, что результат ее работы известен (например, частный случай), и в дальнейшем ничего вычислять в рамках данной функции не требуется. Если передавать все параметры по значению, то в момент входа в функцию, должны быть известны все параметры, значит, все выражения должны быть вычислены. Даже если как в нашем абстрактном примере эти параметры для получения результата не понадобятся. Можно конечно и посчитать, но не следует забывать, что в функциональном программировании рекурсия обычное дело, а это либо чистая рекурсия, либо итеративный процесс, и то и другое медленно. Поэтому прирост скорости может достигать больших величин (а до недавнего времени в функциональном программировании это был очень актуальный вопрос). Что касается дополнительных возможностей, продолжим рассмотрение все того же примера. Что если выполнение одного или нескольких параметров невозможно в конкретный момент

вычислительного процесса? Допустим, в выражении происходит деление на ноль. Решить такую задачу обычным способом невозможно. Однако, есть математические выкладки, говорящие о том, что функция имеет решение (потому что ее результат в данном случае зависит от другого параметра, например общий коэффициент, который в данный момент равен нулю, тогда результат функции также будет равен нулю независимо от остальных параметров). Если отложить на некоторое время вычисление остальных параметров и работать с тем, что влияет на логику вычислений, то функция способна вернуть корректный результат, даже если вычисление остальных параметров может привести к фатальной ошибке.

В Scheme отложить вычисление можно через (`delay` (выражение)). Выражение вычислено не будет до конструкции `force` (синтаксис аналогичен). `Force` требует немедленного вычисления выражения (обычно используется в связке с `delay`):

```
(force (delay (+ 1 2)))      Результат 3

(let ((p (delay (+ 1 2))))
  (list (force p) (force p)))  Результат список (3 3)
```

Как видите, между `delay` и `force` может находиться произвольное число выражений, лишь бы `p` по-прежнему оставалось актуальным (то есть в нашем случае `force` должно наступить в рамках `let`, иначе после `p` перестанет существовать). На самом деле пользоваться `delay` и `force` легко, главное чтобы каждое обращение к не вычисленному параметру сопровождалось `force`.

Подобно многим языкам программирования Scheme поддерживает макроопределения (макросы), позволяющие образовывать новые типы выражений. Можно переопределить большинство стандартных конструкций, с помощью уже известной нам (`define` новое_имя наименование_конструкции). При этом конструкция под старым именем тоже сохраняется. Возьмем наш факториал:

```
(define fact
  (lambda (n)
    (if (= n 0)
        1 ;Base case: return 1
        (* n (fact (- n 1))))))
(define mmm fact)
(mmm 1000)
```

Теперь `fact` можно вызывать как под старым именем, так и под именем `mmm`. Scheme обладает мощными средствами для образования новых возможностей языка на основе макроопределения ключевых слов (не просто синтаксическая замена), но в связи со сложностью и большим объемом материала мы далее макросы рассматривать не будем. Если есть желание, то описание работы с макросами можно найти в большой и подробной справке по PLT Scheme (справочная система называется Help Desk).

И традиционная строчка кода:

```
(write "Hello, world!")
```

Типы данных и работа с ними

Предикаты, значения #t, #f.

Предикат – это функция, возвращающая булево значение (Истина/Ложь или `#t` или `#f` в терминах Scheme). Основное их назначение это использование в функциях оценки условий (таких как `cond`). Собственно оценка булевых `#t` и `#f` построена по следующему принципу: все, что не является `#f`, есть `#t` (то есть остальные типы данных, функции, определения и т.д. – все это является `#t`). `#f` может быть получена либо как константа, либо как результат оценки выражения (через предикаты). Особенностью булевого типа в Scheme является тот факт, что на них не распространяется кавычирование, то есть результатом `'#f` будет все тот же `#f`.

Теперь рассмотрим некоторые предикаты:

```
(not obj) – логическое отрицание, все, что есть #t будет
            возвращено как #f (обратное утверждение также
            справедливо)
(not 100)  Результат #f, потому что все не булевы объекты
            являются #t
```

Для того, чтобы определить, что же действительно является булевым значением, а что иными объектами, используется следующий предикат:

```
(boolean? #f)      Результат #t
(boolean? 0)       Результат #f
(boolean? '())      Результат #f

(eqv? obj1 obj2) - оценка объектов, возвращает #t, если объекты
                  эквиваленты между собой (аналогично себя
                  ведет и операция (= obj1 obj2), но только
                  для чисел).

(eqv? 1 2)         Результат #f,
                  поскольку число 1 не является числом 2

(eqv? 2 2)         Результат #t
(eqv? 'a 'a)       Результат #t
(eqv? 'a 'b)       Результат #f
(eqv? '() '())     Результат #t
(eqv? 1000000000 1000000000)  Результат #t
(eqv? (cons 1 2) (cons 1 2))  Результат #f
(eqv? (lambda () 1)
      (lambda () 2))          Результат #f
(eqv? #f 'nil)                Результат #f
(let ((p (lambda (x) x)))
  (eqv? p p))                  Результат #t
```

`eqv?` процедура возвращается `#t` если:

- `obj1` и `obj2` - оба `#t` или оба `#f`;
- `obj1` и `obj2` - оба символы и являются эквивалентными;
- `obj1` и `obj2` - оба числа, равные в цифровой форме;
- `obj1` и `obj2` - оба символы и являются тем же самым символом;
- `obj1` и `obj2` - пустой список;
- `obj1` и `obj2` - пары, векторы, или строки, которые обозначают те же самые местоположения в памяти (хотя определение и похоже, нужно помнить, что в Scheme нет указателей и любое их упоминание не допускается);
- `obj1` и `obj2` - процедуры, тэги местоположения которых равны (не путайте тэги с адресами ячеек памяти).

`eqv?` возвращает `#f`, если:

- `obj1` и `obj2` - имеют различные типы, один из `obj1` и `obj2` `#t`, но другой `#f`;
- `obj1` и `obj2` - неэквивалентные символы;
- один из `obj1` и `obj2` - точное число, но другой неточное число;
- `obj1` и `obj2` - числа, для которых = процедура возвращается `#f`.
- один из `obj1` и `obj2` - пустой список, но другой содержит элементы.
- `obj1` и `obj2` - пары, векторы, или строки, которые обозначают отличные местоположения в памяти.
- `obj1` и `obj2` - процедуры, которые вели бы себя по-другому (возвратили различное значение или имеют различные побочные эффекты).

Случаи, когда `eqv?` возвращает `#t`, а когда `#f` приведены не случайно, есть ситуации, когда

данная функция не может дать точного ответа.

Пример:

```
(eqv? "" "")
(eqv? '() '())

(eqv? (lambda (x) x)
      (lambda (x) x))

(eqv? (lambda (x) x)
      (lambda (y) y))
```

Вернее ответ будет получен, но его результат зависит от реализации (PLT Scheme версии 350 дает ответ `#f`). Ну и не следует забывать о рекурсионных вызовах, в совокупности с определениями локальных переменных они могут быть неэквивалентными (даже если `obj1` есть `obj2`). Аналогично работают `eq?` и `equal?`, однако, в ряде случаев они вернут противоположные значения. Также отметим, что `equal?` лучше использовать для сложных составных типов данных, таких как строки, списки и вектора.

Числа

До недавнего времени числовые вычисления диалектов Лиспа были неэффективными, однако при компиляции уже сейчас дают неплохие результаты (конечно тягаться с тем же C Scheme не сможет). Прежде всего, нужно всегда помнить, что числа в Scheme обычно не имеют представления в типах чисел конкретных аппаратных платформ (с плавающей точкой, байт, слово и т.д.).

Итак, в Scheme имеются следующие основные числовые типы:

- . комплексные;
- . реальные;
- . вещественные;
- . целые.

Например, число 3 является целым. Но в любой момент времени оно также является вещественным, реальным и комплексным. Это справедливо для всех представлений числа 3. Чтобы определить тип числа используются предикаты `number?`, `complex?`, `real?`, `rational?`, и

`integer?`

```
(real? 3)
```

Результат `#t`

Использование остальных предикатов по определению типа числа полностью аналогично. При этом не существует никакого однозначного машинного представления числа 3 (в современных реализациях Scheme существует как минимум два варианта представления целого числа). Например, Scheme одинаково вычисляет `10!` и `10000!`. Понятно, что представление `10000!` ни в какой регистр процессора не поместится, поэтому для такого результата используется альтернативное представление.

Числовые данные обрабатываются на абстрактном уровне (без использования прямых вычислений с помощью арифметических команд процессора) и хотя Scheme и позволяет использовать такой подход, при котором операции над числами будут выполняться в регистрах процессора, такие действия далеко не очевидны, и нельзя однозначно определить текущее машинное представление числа.

В тоже время нужно различать те числа, которые могут быть представлены точно, а какие нет. Например, индексы структур данных имеют однозначное представление, в тоже время результаты некоторых вычислений, иррациональные числа могут быть только приближены к рациональным, что приводит к потерям точности. Поэтому для вычислений (точных и приближенных) могут использоваться ортогональные подходы (взаимодополняющие друг друга) к представлению типов чисел.

В Scheme числа бывают двух видов точные и не точные. Число является точным, если оно выражено точной константой или получено в результате ряда операций (которые также называются точными) над точными числами. Число является не точным, если оно выражено с помощью не точной константы или получено в результате не точной операции или в результате

операций над неточными числами. Таким образом, неточность инфекционное свойство числа (иными словами неточное число получить проще, чем точное). Если реализация выражения с использованием двух неточных чисел приведет к точному результату, считается, что данные вычисления математически эквивалентны (то есть вычисления подвержены ошибкам округления). Однако, в подавляющем большинстве случаев, Scheme использует вычисления, таким образом, чтобы обеспечить максимальную точность результата.

Рациональные операции, такие как сложение, при точных параметрах всегда возвращают точный результат. В случае если операция неточная (то есть ее результат может быть возвращен как неточное число), различные реализации языка могут вести себя по-разному: либо выдадут сообщение о потере точности, либо просто приведут свой результат к неточному числу (без каких-либо сообщений). За исключением явного преобразования неточного числа в точное, операции над неточными числами, как правило, порождают неточный результат. Однако, в ряде операций неточность входящих параметров необязательно приводят к неточным результатам. Например, умножение неточного числа на точный ноль приведет к точному результату.

Конкретные реализации языка могут вводить свои дополнительные типы, которые должны придерживаться ряду требований (иначе такие реализации не будут подходить под стандарт). Это, например, четкое определение вида числа для нового типа – число должно быть либо точным, либо не точным. Соответственно результатом операций должны быть такие числа, которые должны быть либо точными, либо неточными (причем каким будет число, зависит от правил описанных выше). И также существует ограничение для целых чисел – целые числа должны быть точными при их использовании в качестве индексов структур данных, таких как, например, вектора и списки. Кроме того, любая целочисленная константа всегда будет точным

числом. Следующие операции возвращают точные числа, если их входящими параметрами также являются точные числа:

+	-	*
quotient	remainder	modulo
max	min	abs
numerator	denominator	gcd
lcm	floor	ceiling
truncate	round	rationalize
expt		

Реализация операций над не точными числами должны обеспечивать точность не ниже, чем описано в стандарте IEEE (в частности для неточных чисел ограниченных 32-х и 64-х битным диапазоном). Числовые константы могут быть написаны в двоичном, восьмеричном, десятичном, или шестнадцатеричном при помощи префикса основания системы счисления. Префиксы основания следующие: `#b` – двоичное число, `#o` – восьмеричное, `#d` – десятичное, `#x` – шестнадцатеричное. Число без основания является числом, представленным в десятичной форме.

Также существуют префиксы для определения точности: `#e` – число точное, `#i` – не точное число (можно использовать хоть до префикса основания системы счисления, хоть после или вообще без него). Если префикс точности не указан, то число считается неточным в случае, когда содержит точку, экспоненты или префикс `#` вместо цифры. Если число точное и может иметь переменную точность, то ее можно задать с помощью символов `s`, `f`, `d`, и `l`, которые определяют короткую, одинарную, двойную и длинную точности соответственно (при этом внутреннее представление числа может отличаться от заданного). Также можно использовать символ `e`, который выбирает точность, которая должна использоваться для данного типа по умолчанию (некоторые версии языка могут позволять программисту настраивать точности, задаваемые по умолчанию):

3.14159265358979F0	Из-за префикса <code>f</code> число будет округлено (эквивалентно) 3.141593
0.6L0	0.6
0.6L045	6e+044

Вот еще примеры чисел и их тип:

```
(complex? 3+4i)  #t
(complex? 3)     #t
(real? 3)        #t
(real? -2.5+0.0i) #t (некоторые комплексные числа являются
                      реальными)
(real? #e1e10)   #t
(rational? 6/10) #t 6/10 – дробь, способ записи чисел
(integer? 3+0i)  #t
(integer? 3.0)   #t эту константу можно считать целым числом
(integer? 8/4)   #t 8/4 есть форма записи чисел – дроби

(напомню еще раз, что операция деления записывается так:
(/ x y), подробности далее...)
```

Заключение

В следующем номере нашего журнала мы продолжим рассмотрение типов данных в среде Scheme, узнаем об особенностях управления и структуре IDE.

Продолжение следует...

Ресурсы

. Биография Джеральда Сассмана

<http://www.csail.mit.edu/user/1512>

. Все про Scheme <http://alexey.tamb.ru/scheme>

Список сокращенных обозначений и терминов, использованных в статье:

СБИС – сверхбольшие интегральные схемы.

CAD – Computer Aided Design – компьютерная поддержка конструирования, предполагает объемное и плоское геометрическое моделирование, инженерный анализ на расчётных моделях высокого уровня, оценку проектных решений, получение чертежей.

Java – язык программирования.

Хвостовая рекурсия – вид рекурсии, когда вызов функцией самой себя осуществляется в конце ее работы (используется для оптимизации (в частности уменьшает объем требуемой для рекурсионных вызовов памяти за счет автоматического преобразования рекурсии к итерационному виду)).

Польская нотация – способ записи математических и логических выражений, при котором оператор располагается слева от операндов.

Лексема – последовательность допустимых символов языка программирования, имеющая смысл для транслятора. Транслятор рассматривает программу как последовательность лексем.

Сборка мусора – одна из форм автоматического управления памятью. Специальный код, называемый сборщиком мусора (garbage collector), периодически освобождает память, удаляя объекты, которые уже не будут востребованы приложением – то есть производит сборку мусора.

Лямбда-исчисление – формальная система, разработанная американским математиком Алонзо Черчем, для формализации и анализа понятия вычислимости. λ -исчисление может рассматриваться как семейство прототипных языков программирования. Их основная особенность состоит в том, что они являются языками высших порядков. Тем самым обеспечивается систематический подход к исследованию операторов, аргументами которых могут быть другие операторы, а значением также может быть оператор. Языки в этом семействе являются функциональными, поскольку они основаны на представлении о функции или операторе, включая функциональную аппликацию и функциональную абстракцию.

Цикл – разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций. Также циклом может называться любая многократно исполняемая последовательность инструкций, организованная любым способом (например, с помощью условного перехода).

Императивный стиль программирования – это парадигма программирования, которая, в отличие от декларативного программирования, описывает процесс вычисления в виде инструкций, изменяющих состояние программы. Императивная программа очень похожа на приказы, выражаемые повелительным наклонением в естественных языках. Императивные языки программирования противопоставляются функциональным и логическим языкам программирования.

Указатели – переменная, диапазон значений которой состоит из адресов ячеек памяти и специального значения – нулевого адреса. Значение нулевого адреса не является реальным адресом и используется только для обозначения того, что указатель в данный момент не может использоваться для обращения ни к какой ячейке памяти.

Инфекционность – свойство объекта распространять свои свойства (включая и инфекционность) на другие объекты.

Рациональное число (лат. ratio – отношение, деление, дробь) – число, представляемое обыкновенной дробью m/n , где числитель m – целое число, а знаменатель n – натуральное число. Такую дробь следует понимать как результат деления m на n , даже если нацело разделить не удастся. В реальной жизни рациональные числа используются для счёта частей некоторых целых, но делимых объектов, например, тортов или других продуктов, разрезаемых на несколько частей, или для грубой оценки пространственных отношений протяжённых объектов.

Иррациональное число – это вещественное число, которое не является рациональным, то есть которое не может быть представленным в виде дроби m/n , где m – целое число, n – натуральное число.

Натуральное число – числа, возникающие естественным образом при счёте (как в смысле перечисления, так и в смысле исчисления).

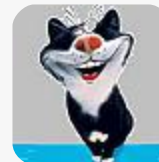
IEEE (англ. Institute of Electrical and Electronics Engineers) (I triple E – «Ай трипл и») – международная некоммерческая ассоциация специалистов в области техники, мировой лидер в области разработки стандартов по радиоэлектронике и электротехнике.

Haskell (русс. Хаскель, Хаскелл) – стандартизованный чистый функциональный язык программирования общего назначения. Является одним из самых распространённых языков программирования с поддержкой отложенных вычислений.

Реальное, или действительное число (англ. real number, нем. reelle Zahl, лат. numerus realis) – математическая абстракция, возникшая из потребности измерения геометрических и физических непрерывных величин окружающего мира, а также проведения таких операций как извлечение корня, вычисление логарифмов, решение алгебраических уравнений [2].

Хэш или ассоциативный массив – абстрактный тип данных (интерфейс к хранилищу данных), позволяющий хранить пары (ключ, значение) и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу.

Вот мы и подошли к заключительной серии наших уроков по графике в среде DELPHI. Сегодня рассмотрим практическое применение модуля <LoadObjectToBufferMod>, из нашего прошлого материала, на примере работы с движущимися графическими объектами...



Продолжение. Начало цикла смотрите в предыдущих выпусках журнала...

Владимир Дегтярь

by DeKot degvv@mail.ru

Проект с использованием дополнительного универсального модуля. Урок 7

Создайте и сохраните аналогично предыдущим урокам новый проект <Lesson 4>. В разделе uses введем модуль и в разделе var переменные для фона и звездолета 'ship1'. Сам модуль должен находиться в папке проекта (см. листинг 1):

ЛИСТИНГ 1

```
Interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, LoadObjectToBufferMod, ExtCtrls;

var
  Form1: TForm1;
  xf, yf: integer;      // координаты вывода общего буфера на
                        // форму
  dyf: integer;         // приращение изменения координаты yf
                        // по вертикали
  xS1, yS1: integer;    // координаты звездолета 'ship1'
  dxS1, dyS1: integer;  // приращение координат 'ship1' по гориз.
                        // и вертик.
  xR, yR: integer;      // координаты звездолетов
                        // 'ship4'...'ship7'
  dyR: integer;         // приращение координат 'ship2 - 5'
  N_kadrS1: byte;       // номер изображения спрайта 'ship1'

implementation
```

В процедуре OnCreate() формы проведем инициализацию буферов и введем начальные данные для переменных (см. листинг 2):

ЛИСТИНГ 2

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // инициализация и загрузка битовых образов буфера фона и
  // общего буфера
  InitFon(1,2,'data/star1.bmp');
  LoadFon(0,0,'data/star1.bmp');
  LoadFon(0,540,'data/star2.bmp');
  InitBuff;
  // начальные значения переменных
  xS1:= 250; yS1:= 1006;
```

```
dxS1:= 5; dyS1:= 2;
xf:= 0; yf:= -540; dyf:= 2
end;
```

Процедура InitFon() вызывается из модуля '1' - количество используемых файлов фона по ширине, '2' - по высоте, 'data/star1.bmp' - по этому файлу определяем размеры основного буфера фона BufFon. Далее в процедурах loadFon() загружаем все файлы фона в BufFon. InitBuff() - инициализация буфера Buffer и загрузка в него фона. Движение графических объектов организовано в обработчике таймера (см. листинг 3):

ЛИСТИНГ 3

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // вывод движения 'ship1'
  N_kadrS1:= InitSprite('data/ship1.bmp',2,1,1,N_kadrS1);
  LoadBuff(xS1-dxS1,yS1+dyS1,xS1,yS1,0);
  Form1.Canvas.Draw(xf,yf,Buffer);

  yf:=yf+dyf; // приращения координат для движ. фона
  if yf >= 0 then
  begin
    // если половина фона "прошла" через окно формы
    yf:= -540; yS1:= 1006; // возврат к начальным координатам
    FreeBuff; // "перезагрузка" фона
  end;

  // приращение для 'ship1', обратное приращению фона
  yS1:= yS1 - dyS1
end;
```

Функция InitSprite() возвращает номер следующего кадра (следующий рисунок спрайта 'ship1'), который при следующем такте таймера опять передается в эту же функцию. В процедуре LoadBuff() предыдущее изображение спрайта перекрывается областью TRect фона и затем выводится новое изображение спрайта.

Когда весь фон проходит через окно формы в процедуре FreeBuff() фон снова перерисовывается в Buffer и возвращаются начальные координаты вывода Buffer на форму. Движение звездолета по горизонтали организовано в обработчике нажатия

клавиш (см. листинг 4):

```
procedure TForm1.FormKeyDown(Sender: TObject;
var Key: Word; : TShiftState);
begin
  case Key of
    VK_Left: begin // клавиша "стрелка" - влево
      {если звездолет у левого края формы xS1 не изменяется}
      if xS1 <= 0 then EXIT;
      dxS1:= -5; // движение звездолета влево
      xS1:= xS1 + dxS1;
    end;
    VK_Right: begin // клавиша "стрелка" - вправо
      {если звездолет у правого края формы xS1 не изменяется}
      if xS1 >= 630 then EXIT;
      dxS1:= 5; // движение звездолета вправо
      xS1:= xS1 + dxS1;
    end;
  end;
end;
end;
```

ЛИСТИНГ 4

Запустите данный проект. Получим приложение аналогичное проекту <Lesson 3>. Сравнив коды этих проектов, заметно преимущество применения модуля. Сам проект приведен в папке <Lesson 4_1> (см. ресурсы к статье). Мы же продолжим создание проекта <Lesson 4> далее, введя еще один движущийся графический объект (звездолет из файлов 'ship4' ... 'ship7'). Эти файлы имеют разный размер и разное количество рисунков).

В разделе var введем переменные для нового графического объекта:

```
var
  // номер изображения спрайта 'ship4' ... 'ship7'
  N_kadrR: byte;
  // номер файла спрайтов
  ns: byte = 5;
  // кол-во изображений в строке файла спрайтов
  Nspr: byte = 2;
```

Добавим начальные значения новых этих переменных:

```
// начальные значения переменных-
xs1 := 250; ys1 := 1006;
dxs1:= 4; dys1:= 2;
xf := 0; yf := -540; dyf:= 2;
xr := 50; yr:= 450; dyr:= 3;
Randomize;
```

В обработчике таймера для нового объекта повторно вызовем методы вывода изображений

спрайтов (см. листинг 5):

```
// вывод движения 'ship4-ship7'
N_kadrR:= InitSprite('data/ship' + inttostr(ns) + '.bmp',
Nspr, 1, 1, N, N_kadrR);
LoadBuf(xR, yR - dyR, xR, yR, 0);
Form1.canvas.draw(xf, yf, Buffer);

yR:= yR + dyR; // приращение для 'ship4-ship7'
if yR >= yS1 + 100 then
begin // проход 'ship4-ship7' через окно формы
  xR:= random(600);
  yR:= yS1 - 600;
  ns:= random(4) + 4;
  if ns = 7 then Nspr:= 4
  else Nspr:= 2
end
end
```

ЛИСТИНГ 5

Здесь, в функции InitSprite() переменная Nspr означает количество изображений в файле спрайтов и определяется по значению ns – номера файла нового объекта, который в свою очередь определяется функцией random() после каждого прохождения новым графическим объектом окна формы. Причем, движение 'ship1' аналогично предыдущим примерам в процедуре FormKeyDown(). Полностью проект приведен в папке <Lesson 4> (см. ресурсы к статье).

Развитие проекта Lesson 4. Урок 8

Продолжим наш проект <Lesson 4>, превратив его в небольшой «шутер» или попросту в «стрелялку». Добавим также возможность стрельбы ракетами для 'ship1' по встречным звездолетам и эффект взрыва при попадании. Соответствующие графические файлы ракеты (bullet) и взрыва (explo) находятся в папке <data>. Сделаем это новым проектом с использованием проекта <Lesson 4>.

Итак, приступим... Создайте новую папку и присвойте ей имя <Lesson 5>. Скопируйте в нее все файлы проекта <Lesson 4> и затем откройте проект в Delphi. Переименуем проект - File => Save Project as ... и в диалоговом окне "Введите имя файла" - введите Lesson5 и сохраните проект под новым именем. Теперь снова откройте проект <Lesson 5> и добавьте переменные для новых

объектов:

```
xb, yb: integer;           // координаты ракеты
dyb: integer;              // приращение координат ракеты
n_kadrb: byte;             // номер изображения ракеты
n_kadre: byte;             // номер изображения взрыва
vistrel, flag_s: boolean;
```

Флаги `vistrel` и `flag_s` типа `Boolean` необходимы для управления выводом изображения при выстреле ракетой. Случайный выбор звездолетов 'ship4' ... 'ship7' выделим в отдельную процедуру, так как выбор в программе осуществляется несколько раз после прохождения звездолетом нижней границы окна формы, после попадания ракеты, а значит и уничтожения звездолета (см. листинг 6):

```
procedure randomship;      ЛИСТИНГ 6
begin
  xr:= random(600);
  yr:= ys1 - 600;
  ns:= random(4) + 4;
  if ns = 7 then nspr:= 4
  else nspr:= 2
end;
```

В процедуре `FormKeyDown()` добавим обработку клавиши "пробел" – пуск ракеты (см. листинг 7):

```
vk_space: // пробел – выстрел      ЛИСТИНГ 7
if not vistrel then begin
  xb:= xs1 + 22;
  yb:= ys1 - 30;
  dyb:= 20;
  vistrel:= true
end;
```

В обработчике таймера выводим движение ракеты при произведенном пуске – флаг (см. листинг 8):

```
if yr >= ys1 + 100 then      ЛИСТИНГ 8
  randomShip; // проход ship4-7 через окно формы
  if (vistrel) and (not flag_s) then begin
    // если произведен выстрел, то
    // вывод изображения ракеты
    n_kadrb:= initSprite('data/bullet.bmp', 1, 1, 1, n_kadrb);
    loadBuff(xb, yb + dyb, yb, 0);
    yb:= yb - dyb;
    // ракета вышла за пределы окна формы
    if yb <= ys1 - 510 then begin
      vistrel:= false;
      freeBuff // ПЕРЕЗАГРУЗКА фона
    end
  end;
```

и также вводим "эффект взрыва" при попадании

ракеты в цель (см. листинг 9):

```
if (yb <= yr) and           ЛИСТИНГ 9
  (xb >= xr) and
  (xb <= xr+100) and
  (vistrel) then begin
  flag_s:= true;
  dyb:= 0;
  n_kadre:= initSprite('data/explo.bmp', 8, 1, 1, n_kadre);
  loadBuff(xb-54, yb + 2, xb-54, 0);
  form1.canvas.draw(0, yf, buffer);

  if n_kadre = 0 then begin // конец эффекта взрыва
    vistrel:= false;
    flag_s:= false;
    freeBuff;
    randomShip
  end
end;
```

И собственно то, чего добивались. Скомпилируем и запустим наш проект (см. рисунок):



Рис. Окно проекта игры

Заключение

На этом мы закончим рассмотрение простейших приемов работы с движущимися графическими объектами. Предложу еще самостоятельно разобрать метод вывода на канву текстовой графики. Добавьте в код последнего приложения счетчики количества пусков ракет и попаданий в цель (например: `count1`, `count2`) и используя свойства `Font(Size,Style,Color)` и метод `TextOut(X,Y,IntToStr(count..))` введите статистику в приложении. Будем считать это домашним заданием...

Ресурсы к статье в архиве с **4-м** номером журнала.

Почти несерьезная статья о серьезной работе: аппаратном ремонте компьютера. Коротко рассказано об основных неисправностях компьютера, и методах их устранения. Приведены основные направления поиска неисправностей, в основном статья предназначена для начинающих ремонтников, а также тех, кто хочет отремонтировать свой компьютер, но не знает, с чего начать...



Дмитрий Дмитренко

ddn.research@gmail.com

Скажу сразу: полностью «спалить» компьютер можно только в печке, если что-то в нем выходит со строя, то обычно какая-то отдельная часть, блок, который вполне возможно восстановить собственными силами, в крайнем случае – купить. Так что, если паяльник Вы держите в руках без заметного волнения, то «лечение» Вашего доброго помощника, советчика, а для некоторых, и «сожителя», Вам очень даже по плечу! Дело за малым: определить, какая именно часть «тела» Вашего «друга» отказала.

Краткий экскурс...

Я не занимаюсь специализированным ремонтом компьютеров, сфера моей деятельности немного не та [1, если кому интересно], но у меня есть много знакомых, друзей и родственников, у которых есть компьютеры, имеющие дурную привычку ломаться. Поэтому я решил описать здесь большинство поломок, имеющих наглость встречаться лично мне. Конечно, можно было бы привести кучу скопированного текста с разных сайтов и умных книжек, но я ограничусь своим собственным мнением на эту тему.

Лично я разделяю проявления неисправностей компьютеров на следующие группы:

1. Компьютер не включается (не запускается) – при этом не подаются никакие признаки жизни, максимум, что можно от него «добиться», это включения и последующего отключения. Как ни странно, но для меня такого рода поломки – одни из самых легких, хотя, частенько бывают самые материалозатратные...
2. Компьютер включается, но останавливается на

стадии BIOS, то есть зависает при появлении информации о системе, или до начала загрузки операционной системы на экране появляются различные сообщения с ключевыми словами «failure» или «failed», или, наконец, раздаются различные кодированные сигналы внутренним динамиком. Такие неисправности также удачно «лечатся», если знать особенности Вашей материнской платы (я имею виду расшифровки этих самых звуковых сигналов).

3. Не запускается операционная система. Это, конечно, тяжелый случай, но исправимый. Если операционка не грузится, а в общем компьютер проявляет все радости жизни, то, вероятнее всего, виноваты Вы, когда беспардонно лазя по Интернету или всовывая в порты флешки и диски непроверенные антивирусом, поймали вирус, также, как и, когда удаляли какую-нибудь архисложную программу, удалили вместе с ней очень важный системный файл... Да много еще чего может быть! Я в таких случаях восстанавливаю систему, в крайнем случае – переустанавливаю. Но эту неисправность паяльником не вылечить, поэтому здесь рассматривать не будем.

4. Сбои во время работы. Самая сложная неисправность, причин может быть много, но, зная принципы возникновения этих причин, а также физические свойства полупроводников, проводников и диэлектриков, можно с легкостью одолеть и их.

Что ж, начнем...

С радостью прибежав с работы, наспех разувшись и раздевшись, не поевши и не поздоровавшись с родными, с мыслями о вчерашней «недобитой» игре, недописанному сообщению друзьям, недоперенабранной давеча статье, подходите к

компьютеру, включаете его, нажимаете заветные кнопки, и... Нет повести печальнее на свете, чем время, проведенное не в нете! Что делать, спросите Вы? Конечно, можно и повеситься, но не сейчас!

Для начала давайте проверим, а поступает ли вообще напряжение на наш компьютер. При наличии тестера это сделать проще простого, вплоть до того, что замерять напряжение на штекере питания компа. Если напруга есть - идем дальше*. Снимаем крышку корпуса и смотрим умоляющими глазами на это разобранное безобразие. Скорее всего, сейчас Вам понадобится пылесос. Не побрезгуйте, сделайте милость Вашему помощнику - почистите его. Кстати, очень рекомендую как можно чаще чистить внутренности компьютеру, развитие передовых технологий в наше время привело к тому, что даже пыль может влиять на стабильность работы любого электронного прибора...

Ну вот, и почистили его, и поумоляли, и с бубном потанцевали, а он молчит, гад! Сейчас сразу определимся: Ваш комп вообще не включается, так как если он включается, даже на короткое время, а потом отключается, то это уже может быть другая причина.

Значит так, придется опять брать в руки тестер, если Вы его уже брали до этого... Будем измерять выходные напряжения питания.

Выходные - в смысле, не потому, что отдыхают, а потому что на выходе.

Находим разъем питания на материнской плате, его можно издали заметить, он немаленький, и практически все идущие от него провода далеко не худенькие. Сейчас появилось множество стандартов разводки разъемов блока питания, значения напряжений на нем можно посмотреть в инструкции к материнке, поэтому я здесь конкретно останавливаться на этом не буду. Вот, например, на рисунке 1 приведены чертежи разводки некоторых особенно часто встречающихся в последнее время разъемов. Ну как, измерили? Здесь возможны три с половиной варианта: все напряжения отсутствуют, одно-два (но не все) из напряжений не соответствует номинальному значению (номинальное значение должно отклоняться более чем на 5%), или

1	10	11	20
2	19	12	21
3	18	13	22
4	17	14	23
5	16	15	24
6	15	16	23
7	14	17	22
8	13	18	21
9	12	19	20

Контакт	Цель	Цвет провода	Назначение
1	3.3V	Оранжевый	Напряжение +3,3В
2	3.3V	Оранжевый	Напряжение +3,3В
3	COM	Черный	Общий провод, «земля»
4	5V	Красный	Напряжение +5В
5	COM	Черный	Общий провод, «земля»
6	5V	Красный	Напряжение +5В
7	COM	Черный	Общий провод, «земля»
8	PWR_OK	Серый	Выход сигнала запуска
9	5VSB	Фиолетовый	Дежурный режим +5В
10	12V	Желтый	Напряжение +12В
11	3.3V	Оранжевый	Напряжение +3,3В
12	COM	Черный	Общий провод, «земля»
13	-12V	Синий	Напряжение -12В
14	/PS_ON	Зеленый	Управление включением
15	COM	Черный	Общий провод, «земля»
16	COM	Черный	Общий провод, «земля»
17	COM	Черный	Общий провод, «земля»
18	-5V	Белый	Напряжение -5В
19	5V	Красный	Напряжение +5В
20	5V	Красный	Напряжение +5В

1	12	13	24
2	23	14	25
3	22	15	26
4	21	16	27
5	20	17	28
6	19	18	29
7	18	19	30
8	17	20	31
9	16	21	32

Контакт	Цель	Цвет провода	Назначение
1	3.3V	Оранжевый	Напряжение +3,3В
2	3.3V	Оранжевый	Напряжение +3,3В
3	COM	Черный	Общий провод, «земля»
4	5V	Красный	Напряжение +5В
5	COM	Черный	Общий провод, «земля»
6	5V	Красный	Напряжение +5В
7	COM	Черный	Общий провод, «земля»
8	PWR_OK	Серый	Выход сигнала запуска
9	5VSB	Фиолетовый	Дежурный режим +5В
10	12V	Желтый	Напряжение +12В
11	12V	Желтый	Напряжение +12В
12	3.3V	Оранжевый	Напряжение +3,3В
13	3.3V	Оранжевый	Напряжение +3,3В
14	-12V	Синий	Напряжение -12В
15	COM	Черный	Общий провод, «земля»
16	/PS_ON	Зеленый	Управление включением
17	COM	Черный	Общий провод, «земля»
18	COM	Черный	Общий провод, «земля»
19	COM	Черный	Общий провод, «земля»
20	-5V	Белый	Напряжение -5В
21	5V	Красный	Напряжение +5В
22	5V	Красный	Напряжение +5В
23	5V	Красный	Напряжение +5В
24	COM	Черный	Общий провод, «земля»

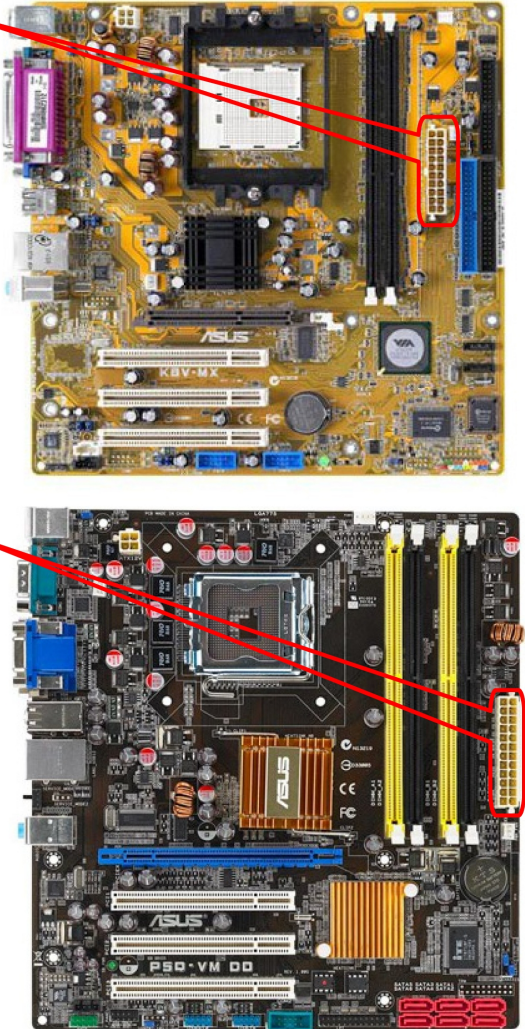


Рисунок 1. Разъем источника питания

вообще отсутствует, все напряжения в норме. Сразу скажу, что последний вариант не рассматриваю, потому что это – практически из области фантастики, очень редкий случай! По поводу первых двух вариантов немного Вас огорчу – придется разбирать блок питания. Естественно, возможны и повышенные значения напряжений, но это довольно редкий и опасный случай, который встречается чаще всего у некачественных блоков питания в основном из-за выхода из строя цепей обратной связи (оптопара и ее обвязка) или микросхемы управления. В таком случае вся надежда на «здоровье» материнки и периферии: смогут ли они выдержать повышенное напряжение, или нет, есть ли у них защита от повышенного напряжения питания, или нет...

Стоп! Забыл... Если отсутствуют все напряжения, кроме 5 и (или) 3,3 Вольта, например, цепи 5VSB, то еще проверьте сигнал включения источника питания PS_ON (чаще всего он зеленого цвета), поступающий с процессора. Если его нет – проверьте цепь, в крайнем случае – откиньте этот

*** Важное замечание.**

Следует обратить ваше внимание на опасность статического электричества для компьютерных комплектующих. И хоть времена КМОП без защитных диодов на входе отошли, но лучше перестраховаться. Прежде чем прикасаться к чему-либо внутри компьютера, избавьтесь от заряда статического электричества, прикоснувшись к неокрашенной металлической поверхности, например к металлической части на задней панели.

В процессе работы периодически дотрагивайтесь до неокрашенных металлических поверхностей на корпусе компьютера для снятия статического напряжения, которое может повредить внутренние компоненты. Отсоедините компьютер и все подключенные к нему устройства от электросети. Также отключите от компьютера все телефонные и телекоммуникационные линии. Это уменьшает риск несчастного случая или удара электрическим током.

Кроме того, придерживайтесь следующих правил техники безопасности... При отключении кабеля от сети беритесь за вилку или за специальную петлю на вилке. Не тяните за кабель. Некоторые кабели имеют фиксаторы на разъемах. Чтобы отсоединить такие кабели, нужно предварительно нажать на эти фиксаторы. Разъединяя разъемы, держите их прямо, чтобы не погнуть контакты. Аналогично, перед подключением кабеля убедитесь в правильной ориентации и соответствии частей разъемов. При работе с компонентами и платами соблюдайте осторожность. Не касайтесь компонентов и контактов плат. Держите плату за края или за металлические кронштейны. Держите компоненты, например микропроцессор, за края, не дотрагиваясь до контактов.

провод от материнки, и подайте на него «общий» питания, или «массу». Если блок питания (не компьютер!!!) запустился, об этом судят по запуску его вентилятора, то, вероятно, у Вас тяжелый случай – что-то с процессором или материнкой... Если же этот номер оказался «дохлым», с чем Вас пока и поздравляю, то переходим к следующему этапу – разборке источника питания.

Проверьте наличие коротких замыканий в самой схеме компа. Отключите разъем, подайте «массу» на вывод включения источника питания PS_ON, и измерьте напряжения (или именно то напряжение, которое отсутствовало). Если все чудесным образом пришло в норму, или оказалось несколько завышено (такое бывает при работе блока питания на «холостом ходу»), то Вам следует искать причину в самом компьютере, а блок питания пока не трогать. Но об этом позже. А сейчас...

Разобрали? Опять пыль? Пылесос в студию! Первым делом проверьте предохранитель. Целый? Тогда нам опять не повезло...

Практически все источники питания собраны по идентичным схемам, впрочем, как и все остальные импульсные блоки питания. Поначалу кажущийся таким сложным, он состоит из высоковольтной и низковольтной частей, и трансформатора. Не будем вникать в технические и технологические подробности, начнем ремонтировать. Только помните – ремонт производят при отключенном от сети устройстве!

Я не буду останавливаться на нюансах ремонта блоков питания, так же, как и на подробных принципах их работы, это тема отдельного и очень длинного разговора, я дам лишь краткие сведения, без описания физических данных блоков, коих сейчас выпущено так много, что всех их и не рассмотришь и не опишешь.

Сначала производим внешний осмотр. Если не выявлено явных повреждений в виде «горелых»,

оплавленных, треснувших от кропотливой работы деталей, переходим к следующей стадии ремонта, если выявлены, то определяем вышедшую из строя запчасть, и... переходим к следующей стадии ремонта.

Проверяем низковольтную часть на наличие пробитых выпрямительных диодов. Это – самая частая неисправность. В качестве таких диодов используют мощные высокочастотные диоды Шоттки**. На рисунке 2 изображен стандартный разобранный блок питания, все основные элементы подписаны, диоды тоже. Измерять обычным тестером, как обычные диоды, они проводят ток в одном направлении, в другом – не проводят.

Затем проверяем высоковольтную часть: входные диоды, задающий транзистор. Я не буду рассказывать, как их проверять, скажу только, что лучше их проверять выпаиванными из платы. Затем на очереди – конденсаторы фильтра питания.

Также можно проверить микросхему, и оптопару, если она присутствует, и их обвязку. Для некоторых, конечно, это высший пилотаж, значения напряжений на выводах микросхемы можно посмотреть только в инструкции к блоку питания, или в datasheet на эту микросхему. В общем – Google рулит! Также проверяем трансформатор на наличие обрывов и, если есть чем измерять, короткозамкнутых витков. Впрочем, если «вылетел» трансформатор, в большинстве случаев придется покупать новый блок питания, хотя, если у Вас есть друзья на «разборке», или Вы специалист по перематыванию импульсных

трансформаторов с кучей свободного времени, то Вам невероятно повезло... И, напоследок, проверка емкостей фильтра на выходах. Они бывают как обрывной, потерявшие емкость, так и короткозамкнутые. Конденсаторы, которые не соответствуют стандартным, «строгим», размерам (например, раздутые до неприличия, или с подтеками какой-то желтой жидкости), должны быть проверены тщательнейшим образом!

Ну вот, в принципе, и все, что я хотел сказать про блок питания...

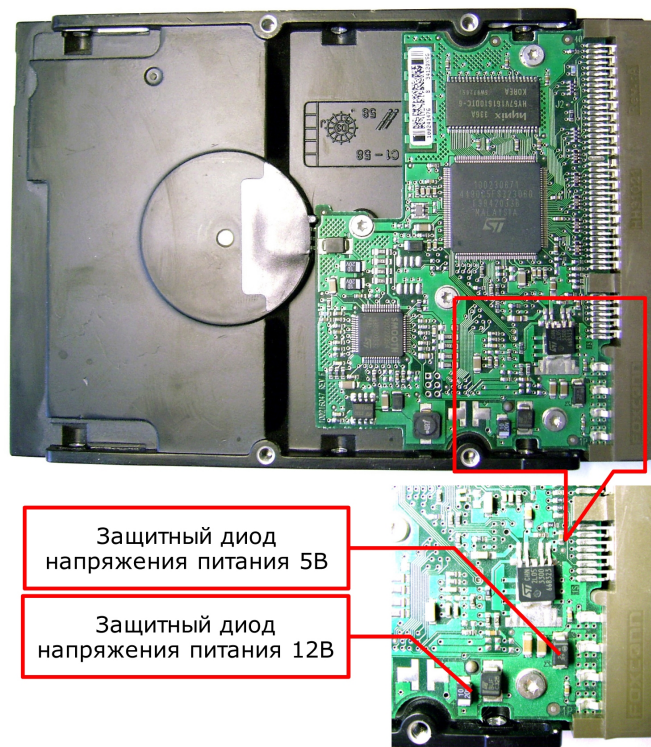


Рисунок 2. Блок питания ПК

Теперь поговорим о случае, когда присутствует замыкание по шине питания в самом компьютере, а блок питания вполне исправен.

Сразу оговоримся, что если КЗ «расположено» на материнской плате, то у Вас ничего не получится! Но попытаемся...

Рецепт прост: отключаем сеть, отключаем по одному устройству и включаем компью-

тер. Очередность приблизительно такая: винчестер (очень часто встречается, во многих винчестерах установлен защитный стабилитрон на входе питания, который при, даже маленьком, превышении значения напряжения питания сразу «коротит» шину питания на землю), приводы CD-DVD-флоппи (причина та же, что и у «винтов»), периферия, расположенная на самой материнке (видеокарта, TV-тюнер и прочее).

Если после отключения какого-либо устройства компьютер заработал или хотя бы включился вентилятор процессора, то виновник видимо

определен. Кстати, совсем забыл о USB, COM и прочих портах. Отключите их в самую первую очередь. Никаких флешек, принтеров, джойстиков!

Сейчас немного отвлекусь от темы и расскажу, что можно сделать, если все-таки вышел из строя защитный стабилитрон. На рисунке 3 изображен жесткий диск производства Samsung (не новый, правда). Измеряем сопротивление стабилитрона, если он «коротит», то есть проводит ток во всех направлениях, притом его сопротивление приближается к нулю, то это он, 100%! Что можно сделать? Да выпаять его, и все тут! Работать будет, но... мало ли, от чего произошел перепад напряжения, из-за чего этот стабилитрон закоротило... Лучше поставить другой, необязательно такой же, просто смотрим, какое напряжение вышло закорочено, и, в зависимости от этого ставим новый. Если закорочена пятивольтовая шина – стабилитрон на 6.3В, можно и 5,6В, если двенадцативольтовая – 12.6 или 13.2В. Если после замены стабилитрона и движений бубном устройство не заработало, то имеет смысл его покрасить в синий цвет, и выбросить.

всего придется покупать новую, а старую похоронить с почестями и салютом. Все, по первому пункту я уже все, что знал – рассказал. Переходим ко второму...

Начнем с теории

Как известно, прежде операционной системы в компьютере запускается встроенная в чип материнской платы программа BIOS (Base Input/Output System, основная система ввода-вывода). Назначение этого небольшого (256 Кб) программного кода – свести к «общему знаменателю» аппаратные различия компьютерного оборудования. Параметры настройки BIOS хранятся в энергозависимой CMOS RAM, которая питается от батарейки на материнской плате. Отсюда вывод: если у вас часто «слетают» установки компьютера или «вредничают» часы – скорее всего, пора менять батарейку. После включения питания напряжение подается на центральный процессор и другие микросхемы материнской платы. «Проснувшись», CPU запускает из микросхемы программу BIOS – и начинается процедура POST (Power On Self Test,

инициализация при первом включении). Ее задача – просканировать и настроить все «железо». Прежде всего формируется логическая архитектура компьютера. Подается питание на все чипсеты, в их регистрах устанавливаются нужные значения. Затем определяется объем ОЗУ (этот

процесс можно наблюдать на экране), включается клавиатура, распознаются LPT- и COM-порты. На следующем этапе определяются блочные устройства – жесткие диски IDE и SCSI, флп-

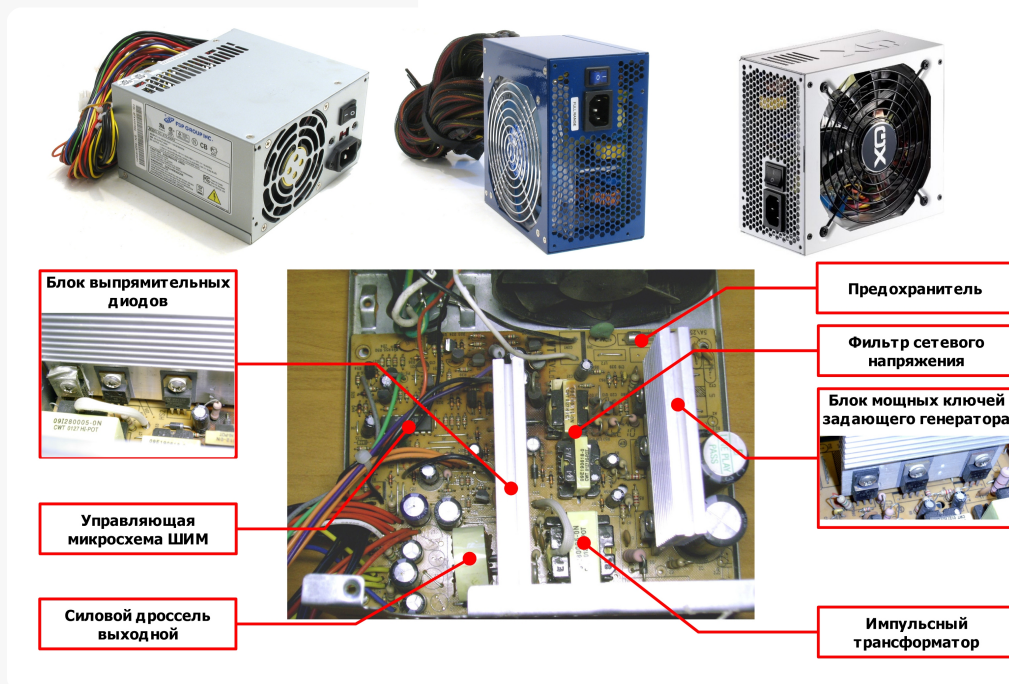


Рисунок 3. Ремонтируем жесткий диск

Если виновник торжества – видеокарта, или прочая периферия, то тут делать нечего, скорее

дисководы. Для устройств SCSI процедура несколько усложняется наличием собственной BIOS, которая берет на себя работу с соответствующим оборудованием и имеет собственную программу настройки. На заключительной стадии происходит отображение итоговой информации.

После окончания работы POST, BIOS ищет загрузочную запись. Эта запись, в зависимости от настройки, находится на первом или втором жестком диске, флоп-диске, ZIP или CDROM. После того как загрузочная запись найдена, она загружается в память и управление передается ей [2].

Если во время работы POST будут обнаружены ошибки, устройство попытается пользователя об этом предупредить. При этом вполне вероятна полная остановка работы компьютера. Задача пользователя – расшифровать сигналы ошибок, передаваемые POST, перевести их на человеческий язык. Для этого существуют специальные POST-карты. Диагностика материнской платы с помощью индикатора POST кодов, превращается в простое и увлекательное дело. Действительно, что может быть проще: вставить POST карту в слот (ISA, часто встречающаяся в «немолодых» компьютерах, или PCI) запустил плату и смотри на результат. В зависимости от модели POST-card коды ошибок выводятся в десятичном или шестнадцатеричном виде на цифровом индикаторе или простыми светодиодами. На некоторых современных материнских платах уже встроены функции заменяющие POST-карту и коды отображаются на находящемся на материнской плате индикаторе.

Но имейте в виду, что POST-карта не может являться спасением от всех бед, так как она отображает только результат выполнения тестовой

процедуры которая находится внутри микросхемы BIOS и при каждом включении и перезагрузке компьютера запускает функции самотестирования основных компонентов и подсистем ПК (таких как процессор, память, чипсет, видеокарту, клавиатуру, жесткие и гибкие диски и т.д.). Если в ходе выполнения обнаружена ошибка, система выдает звуковой сигнал и выводит сообщение на экран (конечно только в случае, если ошибка не была обнаружена раньше, чем например, видеоадаптер). Соответственно, POST карта не покажет полезной информации при неисправности самой микросхемы BIOS, ее обвязке, системах запуска и питания материнской платы. В других, не особенно тяжелых, случаях с материнской платой диагностическая POST карта

может существенно упростить поиск неисправности компонента платы или других деталей компьютера. Таблицы POST кодов можно легко скачать с сайтов производителей микросхем BIOS. Также можно их посмотреть в источниках [2, 3]. Единс-

твенная проблема – наличие собственно этой POST-карты. Если вы не занимаетесь специализированным ремонтом компьютеров, то она Вам нужна, как собаке пятая нога, тем более в денежном выражении она «нормально стоит». Поэтому остановимся на другой индикации ошибок POST – звуковой. Если во время самотестирования BIOS не может вывести информацию на монитор, используется звуковой сигнал или голос, воспроизводимый при помощи встроенного динамика.

Для различных материнских плат применяется различная звуковая «кодировка». Остановимся только на самых распространенных [4], и тех, что мне встречались чаще всего (см. табл.1-3**).

**** Замечание автора.**

Звуковые коды в данном случае представлены в количестве звуковых сигналов. Например, 1-2-2 означает 1 звуковой сигнал, пауза, 2 звуковых сигнала, снова пауза, и опять 2 звуковых сигнала.



Таблица 1. Коды ошибок Award Bios

Последовательность звуковых сигналов	Описание ошибки
1 короткий	Успешный POST
2 коротких	Обнаружены незначительные ошибки. На экране монитора появляется предложение войти в программу CMOS Setup Utility и исправить ситуацию. Проверьте надежность крепления шлейфов в разъемах жесткого диска и материнской платы.
3 длинных	Ошибка контроллера клавиатуры
1 короткий, 1 длинный	Ошибка оперативной памяти (RAM)
1 длинный, 2 коротких	Ошибка видеокарты
1 длинный, 3 коротких	видеокарты
1 длинный, 9 коротких	Ошибка при чтении из ПЗУ
Повторяющийся короткий	Проблемы с блоком питания
Повторяющийся длинный	Проблемы с ОЗУ
низкая частота	Проблемы с CPU
Непрерывный	Проблемы с блоком питания

Таблица 2. Коды ошибок AMI Bios

звуковых сигналов	Описание ошибки
1 короткий	Ошибок не обнаружено, ПК исправен
1 длинный, 1 короткий	Проблемы с блоком питания
2 коротких	Ошибка четности RAM
3 коротких	Ошибка в первых 64 КБ RAM
4 коротких	Неисправность системного таймера
5 коротких	Проблемы с процессором
6 коротких	Ошибка инициализации контроллера клавиатуры
7 коротких	Проблемы с материнской платой
8 коротких	Ошибка памяти видеокарты
9 коротких	Контрольная сумма BIOS неверна
10 коротких	Ошибка записи в CMOS
11 коротких	Ошибка кэша, расположенного на системной плате
1 длинный, 2 коротких	Ошибка видеокарты
1 длинный, 3 коротких	Ошибка видеокарты
1 длинный, 8 коротких	Не подключен монитор

неисправностей – четвертому. Сбой во время работы – это и испорченные нервы, и, нередко, разбитый монитор! Поэтому с этим нужно что-то делать.

Определить причину такого сбоя непрофессионалу можно, в основном, только методом «научного тыка». К примеру, у меня был случай, когда во время работы компьютер просто «виснул» без всяких причин. Причина выяснялась довольно долго, пока, во

Все, пожалуй, хватит на сегодня...

Как пользоваться данными таблицами? Ну вот, например, при включении Вы услышали один повторяющийся длинный «пик» динамика, и Вы знаете точно, что у вас Award BIOS. Разбираем компьютер и смотрим ОЗУ, это такая планка, «память» иногда называется (как оказалось в нашем случае – не вечная). Можно попытаться ее вынуть из разъема и почистить контакты, например, обычным ластиком, и снова поставить на место, иногда помогает. А теперь переходим к самому «страшному» пункту моей классификации

время работы компьютера, я случайно не задел рукой шлейф жесткого диска. Оказалось, окислился контакт, заменил шлейф и «зависания» прошли. Еще когда-то принесли ноутбук, у которого при включении было видно, что загрузка идет, а экран как был темным, так и оставался, только подсветка немного светила. Такое, как оказалось позже, часто встречается при перегреве чипсета видеокарты. В том случае проблема решилась просто: я снял радиатор с этого чипсета, включил компьютер и немного подождал, пока не сработала защита от перегрева, такая защита есть практически у всех компьютеров и ноутбуков.

И, о чудо, при последующем включении все дешево – видеоплата. Ведь до этого в сервисном чудесным образом заработало! Конечно, центре предлагали поменять не только ненадолго, но причина была установлена быстро и видеокарту, но и ОЗУ. Непонятно только – зачем?

Таблица 3. Коды ошибок Phoenix Bios

звуковых сигналов	Описание ошибки
1-1-3	Ошибка записи/чтения данных в/из CMOS-памяти.
1-1-4	BIOS.
1-2-1	Ошибка инициализации материнской платы.
1-2-2 или 1-2-3	Ошибка инициализации контроллера DMA.
1-3-1	Ошибка инициализации схемы регенерации оперативной памяти.
1-3-3 или 1-3-4	Ошибка инициализации первых 64 Кбайт оперативной памяти.
1-4-1	Ошибка инициализации материнской платы.
1-4-2	Ошибка инициализации оперативной памяти.
1-4-3	Ошибка инициализации системного таймера.
1-4-4	Ошибка записи/чтения в/из одного из портов ввода/вывода.
2-1-1	Обнаружена ошибка при чтении/записи 0-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-1-2	Обнаружена ошибка при чтении/записи 1-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-1-3	Обнаружена ошибка при чтении/записи 2-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-1-4	Обнаружена ошибка при чтении/записи 3-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-2-1	Обнаружена ошибка при чтении/записи 4-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-2-2	Обнаружена ошибка при чтении/записи 5-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-2-3	Обнаружена ошибка при чтении/записи 6-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-2-4	Обнаружена ошибка при чтении/записи 7-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-3-1	Обнаружена ошибка при чтении/записи 8-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-3-2	Обнаружена ошибка при чтении/записи 9-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-3-3	Обнаружена ошибка при чтении/записи 10-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-3-4	Обнаружена ошибка при чтении/записи 11-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-4-1	Обнаружена ошибка при чтении/записи 12-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-4-2	Обнаружена ошибка при чтении/записи 13-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-4-3	Обнаружена ошибка при чтении/записи 14-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
2-4-4	Обнаружена ошибка при чтении/записи 15-го бита (в шестнадцатеричном представлении) первых 64 Кбайт ОЗУ
3-1-1	Ошибка инициализации второго канала DMA.
3-1-2 или 3-1-4	Ошибка инициализации первого канала DMA.
3-2-4	Ошибка инициализации контроллера клавиатуры.
3-3-4	Ошибка инициализации видеопамати.
3-4-1	Возникли серьезные проблемы при попытке обращения к монитору.
3-4-2	Не удастся инициализировать BIOS видеоплаты.

Так что, экономия – на лице... заказчика. Вообще, практически все спецэффекты, ибтаситьтаскотерые могут наблюдаться на экране монитора, будь то квадратики-ромбики или цветовой «галлюцинации» происходят по вине видеокарты, а уж затем среди виноватых ищем оперативку или даже процессор.

Заключение

Кстати, перегревания не такая уж редкость в наше пыльное время. И чтобы этого не произошло, и Вам действительно ценно здоровье Вашего компьютера, производите хотя бы раз в полгода-год плановые «медосмотры» с присутствием острого глаза и пылесоса. Подправьте все разъемы, уложите провода, затяните гайки и болты. Не поленитесь также, я уже об этом говорил, очистить от пыли все внутренности, и будет Вам счастье в виде незабываемых минут с развлечениями или работой без нервов и «зависаний»!

Очень полезны в плане ремонта и обслуживания сайты **[5-8, 10]**, есть что почитать, а если нужно

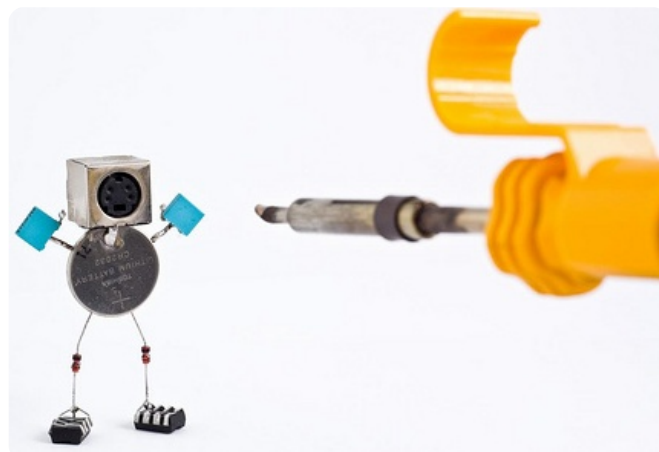
*** Справочная информация.

Диод Шоттки – полупроводниковый диод с малым падением напряжения при прямом включении. Назван в честь немецкого физика Вальтера Шоттки. Диоды Шоттки используют переход металл-полупроводник в качестве барьера Шоттки (вместо p-n перехода, как у обычных диодов). Допустимое обратное напряжение промышленно выпускаемых диодов Шоттки ограничено 250 В (MBR40250 и аналоги), на практике большинство диодов Шоттки применяется в низковольтных цепях при обратном напряжении порядка единиц и нескольких десятков вольт.

Оптопара (оптрон) – электронный прибор, состоящий из излучателя света (обычно — светодиод, в ранних изделиях — миниатюрная лампа накаливания) и фотоприемника (биполярных и полевых фототранзисторов, фотодиодов, фототиристор, фоторезисторов), связанных оптическим каналом и как правило объединенных в общем корпусе. Принцип работы оптрона заключается в преобразовании электрического сигнала в свет, его передаче по оптическому каналу и последующем преобразовании обратно в электрический сигнал. В оптроне входная и выходная цепи гальванически развязаны между собой...

Нижняя рабочая частота оптрона не ограничена – оптроны могут работать в цепях постоянного тока. Верхняя рабочая частота оптронов, оптимизированных под высокочастотную передачу цифровых сигналов, достигает сотен МГц / Википедия

не нашли, то можно и лично вопрос задать. В данной статье я, в основном, описал методику поиска простейших, наиболее часто встречающихся неисправностей, которые можно выявить с помощью отвертки и обычного мультиметра и устранить с помощью обычных рук и обычного паяльника. Для более сложных поломок я и рекомендую эти сайты, иногда полезнее воспользоваться чьим-то опытом, чем наработать свой. Это может слишком дорого стоить...



А вообще – опыт приходит с годами, главное – не бояться и соблюдать меры предосторожности, особенно при работе с металлическими предметами (отвертками, пинцетами) под напряжением. И, я уверен, у Вас все получится.

Источники и ссылки

- Сайт автора статьи <http://ddn.at.ua>
- Программирование под Windows <http://www.ru-coding.com>
- Ремонт материнских плат <http://materinki.narod.ru>
- Звуковые коды BIOS <http://www.umopit.ru/CompLab/BIOSbeeps.htm>
- Форум, посвященный ремонту ноутбуков <http://notebook1.ru/forma1>
- Конференция по ремонту аппаратуры <http://monitor.net.ru>
- Конференция IXBT <http://forum.ixbt.com>
- Ремонт и настройка компьютеров <http://comp-remont.info>

Здравствуйте, уважаемые читатели. Сегодня мы с вами рассмотрим варианты схемотехники подключения LPT порта для ввода и вывода данных, а также некоторые нюансы по формированию импульсного сигнала для управления такими нагрузками как шаговый двигатель (ШД).



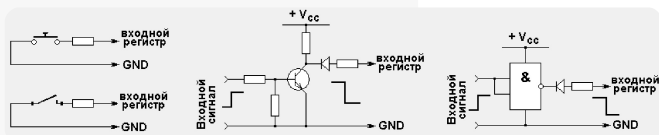
Продолжение. Начало цикла смотрите в третьем выпуске журнала...

Владимир Дегтярь

by DeKot degvv@mail.ru

Подключаем внешние устройства

Уровни напряжения на выводах регистров могут принимать значения + 5В (логическая «1») или 0В (логический «0»), тогда как для управления внешними устройствами бывает, необходимы совсем другие уровни напряжения. В таких случаях используются схемы согласования входов/выходов компьютера со схемами включения внешних устройств. Входы регистров приема информации имеют уровень + 5В. Для подачи входного сигнала на такие входы достаточно подключить через токоограничивающий резистор (100...470 Ом) общий провод, что соответствует подаче «0» на вход (см. рисунок 1):



Vcc - напряжение питания внешних устройств

GND - общий провод, попросту «земля»

Рис. 1. Согласование входных цепей

При подключении внешних устройств к выходам порта LPT следует также применять согласующие устройства с целью предотвращения выхода из строя порта. В простейшем случае возможно использование транзисторных ключей. Однако более надежно применение транзисторных или симисторных оптопар, обеспечивающих полную гальваническую развязку цепей (см. рисунок 2, 3). Имеющейся комбинации входов и выходов: 5 входов «status» и 12 выходов «data» и «control» или же 13 входов «data» и «status» и 4 выхода «control» может оказаться недостаточно для подключения всех необходимых устройств. В таком случае расширить количество подключаемых устройств

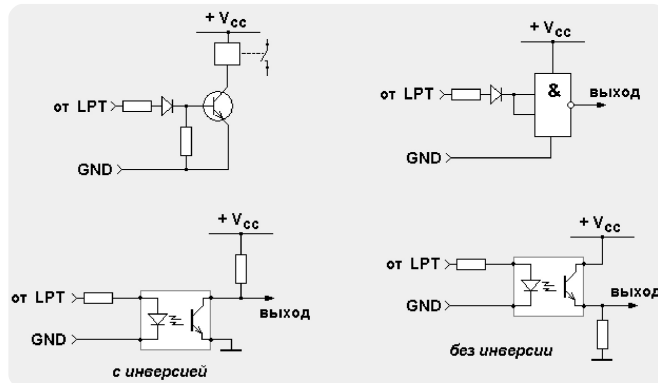


Рис. 2. Согласование и развязка выходных цепей

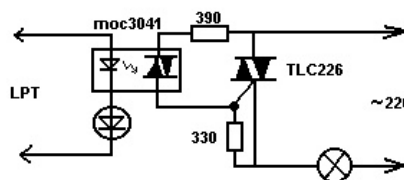


Рис. 3. Согласование и развязка выходных цепей.

Управление мощной нагрузкой

можно применением последовательно-параллельного метода включения внешних устройств. Так 5-битный входной регистр «status» может

принимать 32 различных состояния. Подавая на этот вход поочередно коды состояний подключенных устройств, получим возможность одновременного подключения до 31 устройства. В качестве управляющих адресов используется 4-х битный регистр «control» (см. рисунок 4):

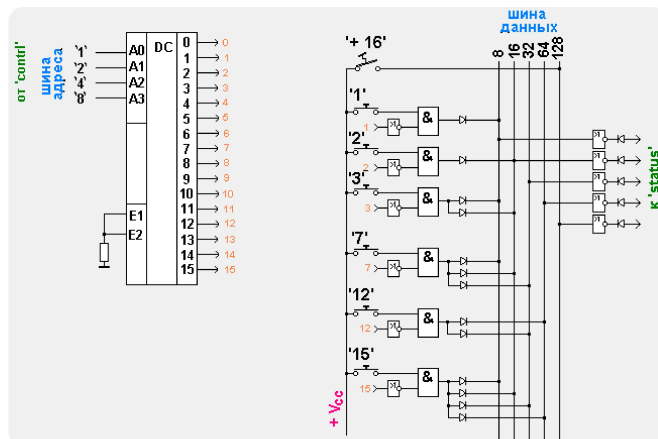


Рис. 4. Вариант расширения. Ввод

Программно это организуется в таймере:

```
var
  // управляющая последовательность
  // см. таблицу 1 из предыдущей статьи [1]
  mas_cntrl : array [0 .. 15] of byte = ( 11, 10, 9, 8, 15, 14,
                                           13, 12, 3, 2, 1, 0, 7, 6, 5, 4);
  // массив кодов подключенных устройств
  mas_cod_in : array [0 .. 15] of byte;

procedure TForm1.Timer1Timer(Sender: TObject);
var i : byte;
begin
  for i := 0 to 15 do begin
    // передаем через «cntrl» адреса (0 .. 15) считывания
    Out32 ($37A, mas_cntrl[ i ] );
    // запоминаем в массиве коды включенных устройств,
    // если устройство не включено - код = 0
    mas_cod_in [ i ] := Inp32 ($379);
    { обработка полученных кодов }
  end;
end;
```

За каждый цикл работы таймера происходит прием кодов включенных устройств и обработка этих кодов. Аналогичным методом можно расширить количество устройств, подключенных к выходам. В качестве запоминающих устройств используются синхронные триггера (см. рисунок 5):

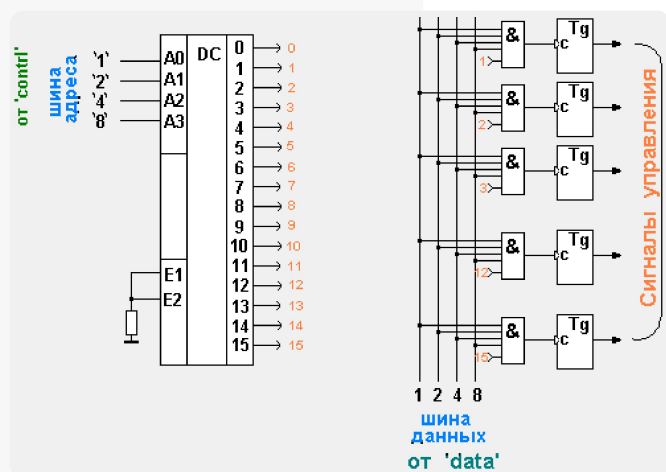


Рис. 5. Вариант расширения количества подключаемых устройств. Вывод

Управление записью выходных кодов организуется по такой же схеме через регистр «control», а результат обработки программой сигналов включения / выключения внешних устройств предварительно преобразуется в массив кодов и затем последовательно параллельным кодом передается на внешние триггеры управления:

```
var
  // массив кодов включения / выключения устройств
  mas_uprav : array [0 .. 15] of byte;

procedure TForm1.Timer2Timer(Sender: TObject);
var i : byte;
begin
  for i := 0 to 15 do begin
    Out32 ($37A, mas_cntrl[ i ] );
    Out32 ($378, mas_uprav[ i ] );
  end;
end;
```

Еще один, довольно «экзотический» способ расширения количества входов связан с использованием порта клавиатуры, вернее имитации нажатия кнопок клавиатуры. Если разобрать современную клавиатуру (речь идет о простых китайских клавиатурах самой низкой стоимости), то мы увидим, что она состоит из двух пленок, на которых нанесены контактные площадки под клавишами и соединяющих дорожек. Все это выводится на плату с управляющим контроллером с помощью 26 контактов (количество контактов на разных моделях может отличаться, но это несущественно). Принцип работы клавиатуры заключается в том, что, нажимая какую либо клавишу, мы тем самым замыкаем определенную пару из этих 26 контактов. Для каждой клавиши комбинация такой пары контактов своя. Далее контроллер клавиатуры обрабатывает этот сигнал и передает в компьютер код нажатой клавиши. Поэтому есть возможность подсоединить к этим 26 контактам свою комбинацию входных сигналов и обрабатывать их уже программно по кодам нажатых клавиш. Особенностью здесь является то, что не должно быть одновременно подаваемых нескольких сигналов таким способом.

Практические приемы программирования в среде IDE Delphi

Для управления состоянием регистров порта LPT мною разработан модуль управления <CtrlBitLPT.pas>. В основу модуля положена все та же библиотека <InpOut32.dll>. Модуль включает в себя уже знакомые нам функции

Inp32(PortAdr) и Out32(PortAdr, data). Кроме этого в модуль входят функции преобразования десятичного числа в двоичное Dec_Bin(N_dec): TMas (функция возвращает массив битов, соответствующий десятичному числу), функция обратного преобразования двоичного числа в десятичное Bin_Dec(mas_bit): byte и функция установки указанного бита регистра порта LPT в заданное состояние (0 или 1) ChangBit(PortAdr, _bit, data): byte. Сам модуль <CtrlBitLPT.pas> приведен в ресурсах к статье [1]. Для применения модуля в папку с проектом скопируйте <InpOut32.dll>, <CtrlBitLPT.pas> и в модуле Unit проекта в разделе uses подключите CtrlBitLPT. Далее очень просто использовать вызовы функций из модуля:

- чтение десятичного числа из регистра порта LPT (просто используем функцию из <InpOut32.dll>) `N_dec := Inp32(PortAdr)`
- чтение значения бита из регистра порта LPT (считываем сначала состояние регистра (PortAdr) в виде десятичного числа, далее преобразовываем его в массив битов (двоичное число) и берем i-й элемент массива)
`Bit_Val (i) := Dec_Bin(Inp32(PortAdr)) [i]`
- запись в регистр порта десятичного значения (снова используем функцию из <InpOut32.dll>) `Out32()`
- управление отдельными битами регистров порта LPT (`ChangBit()`)

Алгоритм работы функции ChangBit() здесь следующий (выполняется в модуле <CtrlBitLPT> при вызове функции ChangBit()):

- считываем значение регистра Inp32 (PortAdr) преобразовываем в двоичное число Dec_Bin (N_dec) в виде массива битов (mas_bit)
- устанавливаем необходимый бит в нужное значение (0 или 1) `Dec_Bin(Inp32(PortAdr)) [_bit] := data` или же `mas_bit [_bit] := data`
- обратно преобразовываем измененный массив битов в десятичное число `N_dec_new := Bin_Dec(mas_bit)`
- запись нового значения в регистр порта `Out32()`

Импульсный сигнал на выходах LPT-порта

Для получения последовательности импульсов на выходах LPT-порта можно применить два метода:

1. Применение таймера

Когда в каждом его цикле можно последовательно производить переключение выходных импульсов. Однако, здесь следует учитывать, что при использовании таймера из палитры стандартных компонентов (вкладка System) или системного таймера Windows, минимальный интервал времени этих таймеров находится в пределах 55 - 64 мсек (т.е. практически максимальная частота импульсов до 150 Гц). Кроме того, значение минимального интервала зависит от особенностей «железа» ПК и нагрузки процессора.

Более стабильные результаты даст применение мультимедийного таймера MMTimer* (модуль MMSystem). Однако и здесь есть ограничение – минимальный интервал такого таймера 1 мсек, но с большей стабильностью и точностью. Таким образом, максимальная частота при применении мультимедийного таймера MMTimer равна 500 Гц (при скважности импульсов 2, наименьшая длительность импульсов = 1 мсек, период следования импульсов = 2 мсек, частота = 500 Гц). Рассмотрим применение мультимедийного таймера MMTimer. Подключаем в раздел Uses модули <CtrlBitLpt> (управление портом LPT) и <MMSystem>:

```
var mmTimer : integer;      // идентификатор таймера
    T       : integer;      // интервал таймера
    on_off  : Boolean;       // флаг вкл-выкл импульсов

// процедура работы таймера
procedure mmTimerProc(TimerID,Msg : UInt ; dwUser, dw1, dw2 :
DWORD ); pascal; // код процедуры формирования импульсов на
begin           // контакте 2 LPT (регистр «data» = $378)
    if on_off = true then Out32($378, 1) // включение импульса
    else Out32($378, 0);                 // выключение импульса
    on_off := not on_off
end;

// управление таймером -- включение
mmTimer:=TimerSetEvent(T, 0, @mmTimerProc, 0, TIME_PERIODIC);
// выключение
TimerKillEvent(mmTimer);
```

Пример проекта с использованием мультимедийного таймера <Имп.ММTimer> приведен в ресурсах к статье. Внешний вид тестовой утилиты <Имп_ММTimer> представлен на рисунке 6:



Рис. 6. Утилита формирования импульсов с помощью мультимедийного таймера

2. Использование циклической процедуры формирования импульсов

Этот метод позволяет получить последовательность импульсов с частотой до 60 кГц. При этом цикл управления выходом импульсов должен быть максимально малым. Смысл такого метода в том, что вся основная часть программы (формирование импульсов, обработка управляющих сигналов, расчеты, графика и т. п.) организована в едином цикле. Но длительность такого цикла зависит от системных свойств компьютера и от загруженности ОС. Поэтому при таком методе следует разработать основные процедуры программы, определить время выполнения одного цикла программы и затем уже определяться с параметрами формирования импульсов. Принцип получения импульсного сигнала в цикле показан на следующей диаграмме и блок-схеме процедуры (см. рисунок 7):

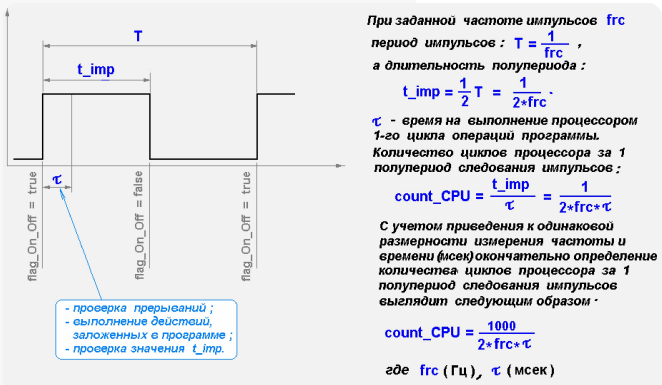
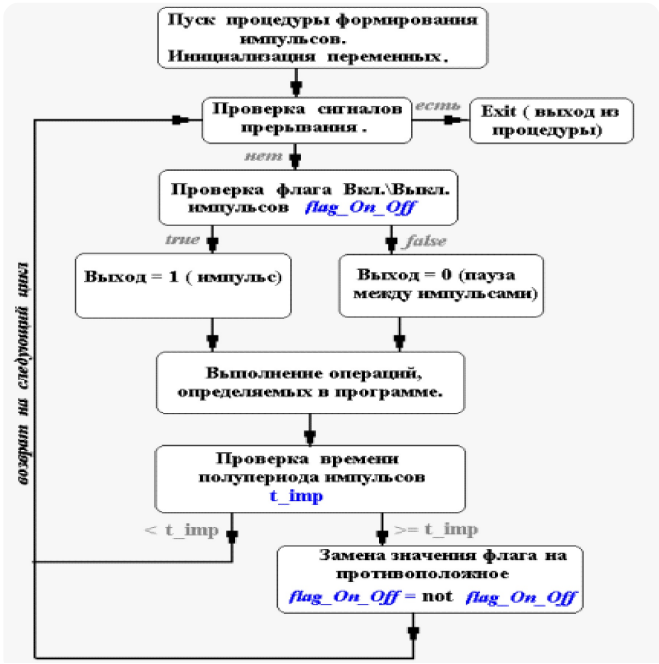


Рис. 7. Диаграмма и блок-схема получения импульсного сигнала

Цикличность выполнения процедуры формирования импульсов определяется величиной σ (временем выполнения процессором команд и операторов программы). Величина σ во многом зависит от аппаратных свойств компьютера (быстродействие, величина свободной памяти) и загруженности ОС системными процессами, а также от величины кода самой программы. При этом значение σ не является постоянной величиной и меняется процессе выполнения программы. Поэтому при высоких требованиях к параметрам импульсов применение ПК в качестве управляющего контроллера бывает невозможно. Задачу следует решать аппаратными методами. Но в большинстве случаев нестабильность частоты в пределах 3-5 % является вполне приемлемой. Рассмотрим управление шаговым двигателем...

Управление шаговым двигателем

Что такое шаговый двигатель и зачем он нужен? Шаговый двигатель - это электромеханическое устройство, которое преобразует электрические импульсы в дискретные механические перемещения. Наверное, каждый видел, как выглядит шаговый двигатель внешне: он практически ничем не отличается от двигателей других типов. Чаще всего это: круглый корпус,



вал, несколько выводов (см. рисунок 8):

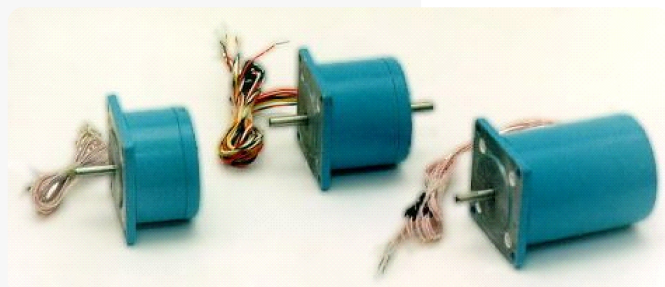


Рис. 8. ШД типа ДШИ-200

Однако шаговые двигатели обладают некоторыми уникальными свойствами, что делает порой их исключительно удобными для применения или даже незаменимыми. Чем же хорош шаговый двигатель? Приведем основные положительные моменты*:

- . угол поворота ротора определяется числом импульсов, которые поданы на двигатель
- . двигатель обеспечивает полный момент в режиме остановки (если обмотки запитаны)
- . прецизионное позиционирование и повторяемость (хорошие шаговые двигатели имеют точность 3-5% от величины шага. Эта ошибка не накапливается от шага к шагу)
- . возможность быстрого старта-остановки-реверса
- . высокая надежность, связанная с отсутствием щеток, срок службы шагового двигателя фактически определяется сроком службы подшипников
- . однозначная зависимость положения от входных импульсов обеспечивает позиционирование без обратной связи
- . возможность получения очень низких скоростей вращения для нагрузки, присоединенной непосредственно к валу двигателя без промежуточного редуктора
- . может быть перекрыт довольно большой диапазон скоростей, скорость пропорциональна частоте входных импульсов

Одним из главных преимуществ шаговых двигателей является возможность осуществлять точное позиционирование и регулировку скорости без датчика обратной связи. Это очень важно, так

как такие датчики могут стоять намного больше самого двигателя. Однако это подходит только для систем, которые работают при малом ускорении и с относительно постоянной нагрузкой. В то же время системы с обратной связью способны работать с большими ускорениями и даже при переменном характере нагрузки. Если нагрузка шагового двигателя превысит его момент, то информация о положении ротора теряется и система требует базирования с помощью, например, концевого выключателя или другого датчика. Системы с обратной связью не имеют подобного недостатка.

При проектировании конкретных систем приходится делать выбор между сервомотором и шаговым двигателем. Когда требуется прецизионное позиционирование и точное управление скоростью, а требуемый момент и скорость не выходят за допустимые пределы, то шаговый двигатель является наиболее экономичным решением. Как и для обычных двигателей, для повышения момента может быть использован понижающий редуктор. Однако для шаговых двигателей редуктор не всегда подходит. В отличие от коллекторных двигателей, у которых момент растет с увеличением скорости, шаговый двигатель имеет больший момент на низких скоростях. К тому же, шаговые двигатели имеют гораздо меньшую максимальную скорость по сравнению с коллекторными двигателями, что ограничивает максимальное передаточное число и, соответственно, увеличение момента с помощью редуктора. Готовые шаговые двигатели с редукторами хотя и существуют, однако являются экзотикой. Еще одним фактом, ограничивающим применение редуктора, является присущий ему люфт. Возможность получения низкой частоты вращения часто является причиной того, что

*** Комментарий редакции.**

- . шаговым двигателем присуще явление резонанса
- . возможна потеря контроля положения ввиду работы без обратной связи
- . потребление энергии не уменьшается даже без нагрузки
- . затруднена работа на высоких скоростях
- . невысокая удельная мощность
- . относительно сложная схема управления

разработчики, будучи не в состоянии спроектировать редуктор, применяют шаговые двигатели неоправданно часто.

Виды шаговых двигателей**

Существуют три основных типа шаговых двигателей:

- двигатели с переменным магнитным сопротивлением
- двигатели с постоянными магнитами
- гибридные двигатели

В шаговом двигателе вращающий момент создается магнитными потоками статора и ротора, которые соответствующим образом ориентированы друг относительно друга. Статор изготовлен из материала с высокой магнитной проницаемостью и имеет несколько полюсов. Полюс можно определить как некоторую область намагниченного тела, где магнитное поле сконцентрировано. Полюса имеют как статор, так и ротор. Для уменьшения потерь на вихревые токи магнитопроводы собраны из отдельных пластин, подобно сердечнику трансформатора. Вращающий момент пропорционален величине магнитного поля, которая пропорциональна току в обмотке и количеству витков. Таким образом, момент зависит от параметров обмоток. Если хотя бы одна обмотка шагового двигателя запитана, ротор принимает определенное положение. Он будет находиться в этом положении до тех пор, пока внешний приложенный момент не превысит

некоторого значения, называемого моментом удержания. После этого ротор повернется и будет стараться принять одно из следующих положений равновесия.

Способы управления ШД и практическая реализация

Существует несколько способов управления фазами шагового двигателя. Первый способ обеспечивается попеременной коммутацией фаз, при этом они не перекрываются, в один момент времени включена только одна фаза. Этот способ называют «one phase on» full step или wave drive mode. Точки равновесия ротора для каждого шага совпадают с «естественными» точками равновесия ротора у незапитанного двигателя. Недостатком этого способа управления является то, что для биполярного двигателя в один и тот же момент времени используется 50% обмоток, а для униполярного – только 25%. Это означает, что в таком режиме не может быть получен полный момент (см. рисунок 9):

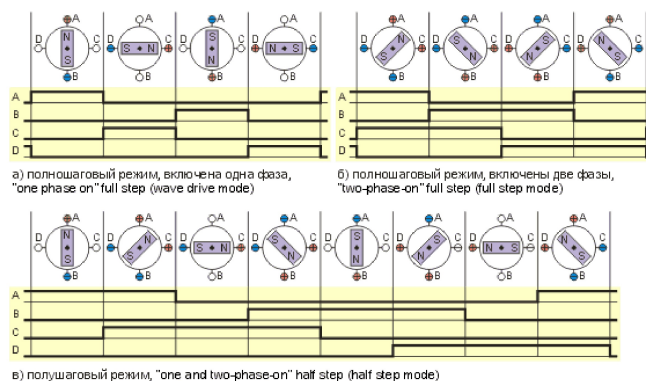


Рис. 9. Различные способы управления фазами шагового двигателя.

** Комментарий редакции.

Определить тип двигателя можно даже на ощупь: при вращении вала обесточенного двигателя с постоянными магнитами (или гибридного) чувствуется переменное сопротивление вращению, двигатель вращается как бы щелчками. В то же время вал обесточенного двигателя с переменным магнитным сопротивлением вращается свободно. Гибридные двигатели являются дальнейшим усовершенствованием двигателей с постоянными магнитами и по способу управления ничем от них не отличаются. Определить тип двигателя можно также по конфигурации обмоток. Двигатели с переменным магнитным сопротивлением обычно имеют три (реже четыре) обмотки с одним общим выводом. Двигатели с постоянными магнитами чаще всего имеют две независимые обмотки. Эти обмотки могут иметь отводы от середины. Иногда двигатели с постоянными магнитами имеют 4 отдельных обмотки.

Второй способ – управление фазами с перекрытием: две фазы включены в одно и то же время. Его называют «two-phase-on» full step или просто full step mode. При этом способе управления ротор фиксируется в промежуточных позициях между полюсами статора и обеспечивается примерно на 40% больший момент, чем в случае одной включенной фазы. Этот способ управления обеспечивает такой же угол шага, как и первый способ, но положение

точек равновесия ротора смещено на пол-шага.

Третий способ является комбинацией первых двух и называется полушаговым режимом, «one and two-phase-on» half step или просто half step mode, когда двигатель делает шаг в половину основного. Этот метод управления достаточно распространен, так как двигатель с меньшим шагом стоит дороже и очень заманчиво получить от 100-шагового двигателя 200 шагов на оборот. На каждый второй шаг запитана лишь одна фаза, а в остальных случаях запитаны две. В результате угловое перемещение ротора составляет половину угла шага для первых двух способов управления. Полушаговый режим обычно не позволяет получить полный момент, хотя наиболее совершенные драйверы реализуют модифицированный полушаговый режим, в котором двигатель обеспечивает практически полный момент, при этом рассеиваемая мощность не превышает номинальной.

Еще один способ управления называется микрошаговым режимом или micro stepping mode. При этом способе управления ток в фазах нужно менять небольшими шагами, обеспечивая таким образом дробление половинного шага на еще меньшие микрошаги. Когда одновременно включены две фазы, но их токи не равны, то положение равновесия ротора будет лежать не в середине шага, а в другом месте, определяемом соотношением токов фаз. Меняя это соотношение, можно обеспечить некоторое количество микрошагов внутри одного шага.

Рассмотрим управление ШД при следующей схеме подключения (см. рисунок 10):

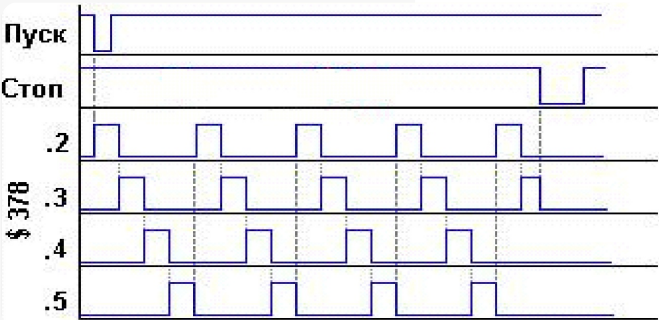


Рис. 10. Диаграмма управляющих сигналов и схема подключения

Программа формирования импульсов для ШД с использованием цикла будет следующей:

```
{ используемые переменные }
var flag_pusk : boolean;      // флаг запуска импульсов
flag_on_off : boolean;       // флаг переключений импульс-пауза
count_CPU : integer;         // кол-во циклов для одного импульса-паузы
count : integer;             // счетчик циклов

// процедура формирования импульсов
procedure Impuls();
begin
  while flag_pusk do begin
    ApplicationProcessMessages;
    if flag_on_off then Out32($378, 1) // импульс
    else Out32($378, 0);              // пауза
    Inc ( count ) ;
    if count < count_CPU then CONTINUE; // повтор цикла
    flag_on_off := not flag_on_off
  end
end ;

// старт формирования импульсов
procedure TForm1.Button1Click( Sender : TObject);
begin
  flag_pusk := true;
  Impuls ()
end;

// стоп формирования импульсов
procedure TForm1.Button2Click(Sender : TObject);
begin
  flag_pusk := false
end;
```

В ресурсах к статье приведен пример проекта программы с использованием цикла – управление 4-х фазным шаговым двигателем <Упр_ШД_4_фаз> (см. рисунок 11).

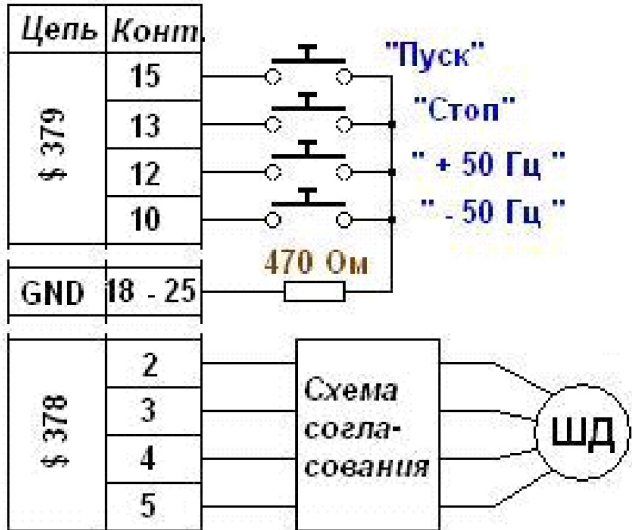




Рис. 11. Утилита управления ШД

Подведем итоги***

Итак, применение персонального компьютера в качестве управляющего контроллера для различных устройств вполне возможно, не составляет особенной сложности и требует только определенных навыков в программировании и практической электронике. Но надо ли это?

В большинстве современных устройств, как промышленных, так и бытовых, применяются специально разработанные для таких целей микроконтроллеры (PIC - контроллеры от MicroChip, ATME1 и др.). Несмотря на их низкую стоимость, разработка управляющих устройств на их основе требует специальных навыков, в любом

случае необходима разработка и изготовление интерфейсных согласующих устройств, устройств ввода и отображения информации. Кроме этого, требуется специальный программатор для ввода рабочей программы и т.п. Также широко применяются программируемые промышленные контроллеры - устройства, имеющие микропроцессорный блок управления, расширяющие устройства ввода и вывода сигналов. Однако стоимость таких устройств сопоставимо со стоимостью обычного компьютера, а зачастую и намного превышает ее. В то же время сейчас как на предприятиях, так и дома уже накопилось достаточное количество устаревших компьютеров (PII, PIII), 14-и 15-и дюймовых мониторов, которые вполне можно приспособить для решения указанных задач.

Кроме всего этого, несомненным достоинством персонального компьютера является большая гибкость при разработке программного обеспечения, создание на мониторе удобного интерфейса для пользователей (пример такого монитора <Kuasy.exe> найдете в ресурсах к статье), возможность быстрого оперативного изменения параметров и настроек, сохранения всевозможных данных, создание отчетов и многое другое.

Конечно, применение компьютера для управления светодиодной гирляндой или вентилятором вряд ли можно назвать экономически целесообразным, но при модернизации, разработке и изготовлении единичных производственных установок применение персональных компьютеров может значительно упростить процесс и принести значительный экономический эффект.

Ресурсы

- В. Дегтярь. Работа с LPT портом в Дельфи или компьютер в роли управляющего контроллера. Часть 1. ПРОграммист. - Клуб ПРОграммистов, 2010, №3 <http://programmersclub.ru/pro/pro3.zip>
- Модули и проекты, использованные в статье <http://programmersclub.ru/pro/pro4.zip>

*** Важное замечание.

В данной статье использованы следующие модули и проекты:

- Библиотека функций для работы с LPT портом в Windows <inout32.dll>
- Модуль для работы с LPT портом в Дельфи <CtrlBitLPT.pas>
- Установка состояний регистров LPT и мониторинг LPT <Упр_Сост_LPT.exe>
- Демо-версия монитора модернизированного ТПА «KUASY 170» <Kuasy.exe>
- Проект формирования импульсов с MMTimer <Имп_MMTimer>
- Проект формирования импульсов с использованием цикла <Упр_ШД_4_фаз>

Все они приведены в виде ресурсов на <http://www.programmersforum.ru> в разделе «Журнал клуба программистов. Четвертый выпуск» или непосредственно в архиве с журналом.

Данная статья рассчитана в помощь программистам и инженерам-разработчикам в области промышленной автоматизации и АСУТП. Поскольку материал объемный, то было решено разделить статью на части, в первой рассмотрим схемотехнику и конструктив модулей передачи и приема сигналов с датчиков приращений, алгоритм работы энкодера, а в остальных – его реализацию на ПЛИС, методику программирования и непосредственно практическое создание тестовой утилиты визуализации состояний датчика приращений...



Сергей Бадло

by raхр <http://raхр.radioliga.com>

*Каждый компонент системы не должен быть наилучшим, а ровно таким, чтобы обеспечить требуемый уровень функционирования всей системы...
/ принцип системотехники*



Для передачи импульсных сигналов с датчика приращений на относительно большие расстояния в условиях промышленных помех можно использовать как радиоканал, так и проводной вариант*, к примеру, дифференциальные интерфейсы на основе RS-485 и LVDS. Реализация радиоканала и RS-485 видится избыточной, так как необходимо наличие «упаковщика» как минимум трех сигналов с шифратора приращений (А, В и строба), т.е. наличие контроллера. Как обойти эту проблему?

Краткий экскурс...

Вспомним про формат LVDS. LVDS используется в таких компьютерных шинах как FireWire, USB 3.0, PCI Express, DVI, Serial ATA. Но среди прочего, данный интерфейс получил распространение и для передачи сигналов на больших скоростях на расстояния до сотни метров. Да, именно так, уже существуют и такие

LVDS драйверы. Кроме того, сигналы с шифратора приращений необходимо декодировать и привести в удобоваримый вид. А значит, возникает необходимость создания аппаратного энкодера для определения таких параметров как: положение ротора, направление вращения, обрыв в канале и т.п.

Для реализации алгоритма энкодера видится два пути:

1. Использование микроконтроллера
2. Использование ПЛИС

Хороши оба варианта. Но для меня как разработчика более удобен вариант представления в виде схемы. Достоинства программируемых логических интегральных схем (ПЛИС) хорошо известны, это и наличие множества готовых библиотек от простейших логических элементов до микропроцессоров и возможность многократного перепрограммирования для изменения схемы, без внесения изменений в печать, и наличие подобного языка VHDL и возможность просто нарисовать поведение схемы. Хотя последние вряд-ли можно назвать преимуществами, так эти свойства присущи и микроконтроллерам...

Таким образом, задачу разработки энкодера для шифраторов приращений можно разбить на следующие этапы:

1. Разработка модуля связи (передачи и приема) для передачи импульсных сигналов на большие расстояния в условиях промышленных помех

Список сокращений, использованных в статье:

PDF – датчик углового положения,

EEPROM – перепрограммируемая ПЗУ,

ПЛИС – перепрограммируемые логические матрицы с сохранением памяти CPLD и без сохранения FPGA (работа схемы ограничивается наличием питающего напряжения),

ТИ – тактовые импульсы,

LVDS (Low-Voltage Differential Signaling) – интерфейс передачи информации дифференциальными сигналами малых напряжений,

Шифратор приращений – преобразователь, на выходе которого в цифровой форме представляются воспринимаемые ими перемещения. Различают* поворотные и абсолютные шифраторы.

- 2. Аппаратная реализация алгоритма энкодера на существующих ПЛИС
- 3. Создание тестовой программы визуализации состояния датчиков приращений

Практика. Реализация модуля передачи и приема сигналов с шифратора приращений

Дифференциальный метод передачи используется в LVDS, поскольку обладает меньшей чувствительностью к общим помехам, чем простая однопроводная схема. Применение источников сигнала с дифференциальным токовым выходом и приемников с низкоомным дифференциальным входом (см. рисунок 2) обеспечивает минимальные индуктивные наводки, поскольку информация передается в форме тока, а емкостная наводка мала, так как при хорошей симметрии линии передачи она является синфазной и подавляется входным дифференциальным приемником. Дополнительной защитой линии является ее экранирование.

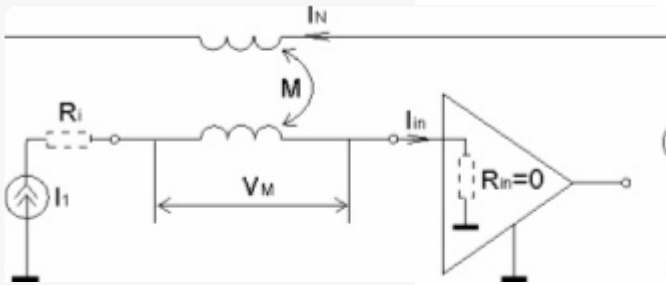


Рис. 2. Канал передачи сигнала с помощью тока нечувствителен к индуктивным наводкам

Поскольку дифференциальные технологии, в том числе и LVDS, менее чувствительны к шумам, то в них возможно использование меньших

* Комментарий автора.

Поворотные шифраторы - генерируют выходные импульсы, которые подсчитываются реверсивным счетчиком, поэтому их показания соответствуют тому, как далеко диск продвинулся с начала отсчета. Здесь в основном применяются два чувствительных элемента, расположенных в преобразователях таким образом, что их выходы сдвинуты относительно друг друга на 90° по фазе.

Абсолютные шифраторы - реализуют кодированный выход, который индицирует абсолютное положение контролируемого объекта, причем кодирование производится в двоичном коде, а его длина соответствует длине кода измерительной системы. Как правило, снабжены интерфейсами: SSI (Synchronous Serial Interface), CAN, PROFIBUS, RS-485.

** Комментарий редакции

Следует упомянуть про параллельно-последовательные преобразователи со встроенным антидребезгом и нормализацией сигналов цифровых входов и преобразования их в единый поток данных, передаваемый по SPI интерфейсу. К примеру, ИМС SN65HVS88х от Texas Instruments. Однако это потребует минимум еще трех корпусов со стороны передатчика и приемника, что приведет к удорожанию модуля связи. Да и сам интерфейс SPI не предназначен для таких расстояний. Кроме того, существуют кодеры на основе сдвигающих регистров (HT12E/HT12D или MC145026/28). Но их применение оправдано в случае использования радиоканала.

перепадов напряжения, до 350 мВ. что позволяет по сравнению с другими способами передачи сигналов значительно снизить потребляемую мощность. Например, статическая мощность, рассеиваемая на нагрузочном резисторе LVDS, составляет всего 1.2 мВт, по сравнению с 90 мВт, рассеиваемыми на нагрузочном резисторе интерфейса RS-422. На рисунке 3 представлена схема организации канала связи между шифратором приращений и контроллером энкодера с использованием дифференциальных линий передачи:

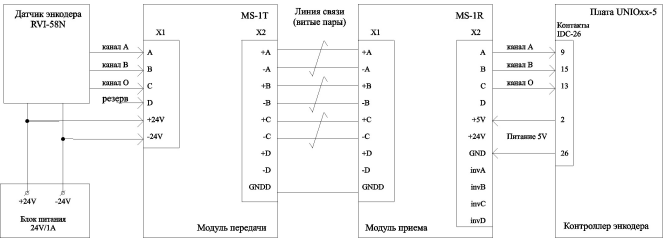


Рис. 3. Схема организации канала передачи сигналов энкодера

Заданным условиям удовлетворяют нижеприведенные схемы (см. рисунок 4 и 5) на основе дифференциальных приемо-передатчиков сигнала LVDS формата SN65LVDS31 и SN65LVDS32 фирмы Texas Instruments [1].

Модуль связи энкодера конструктивно состоит из двух модулей: передачи и приема, разнесенных в пространстве. Модуль передачи размещается рядом с энкодером, а модуль приема непосредственно около контроллера энкодера.

Плата модуля передачи имеет в своем составе:

- узлы согласования входного уровня для типов датчиков приращений с открытым коллектором или потенциальным выходом

- узел формирования питающего напряжения 3.3В для ИМС дифференциального передатчика – U1
- дифференциальный передатчик с согласующими сопротивлениями – U2
- клеммник для подключения входных сигналов с датчика приращений – X1
- клеммник выходных сигналов – X2

Плата модуля приема включает:

- дифференциальный приемник с согласующими сопротивлениями – U1
- узел формирования питающего напряжения 3.3В для ИМС дифференциального приемника с гальванической развязкой – U2
- узлы согласования выходного уровня по типу СК или открытый коллектор
- клеммник для подключения входных сигналов с линии связи – X1
- клеммник выходных сигналов для энкодера – X2

Конструктив модуля связи

Платы модулей [3], габаритами 100x85 мм, выполнены из стеклотекстолита и разведены в пакете OrCad (см. рисунки 6 и 7). На плате передатчика перемычки X1...X4 (PLS-2) предназначены для конфигурации платы под датчики с открытым коллектором и

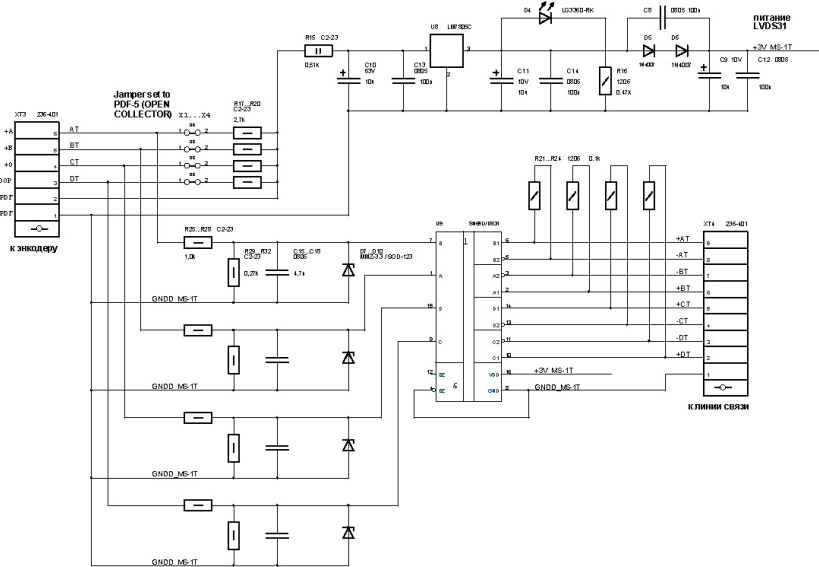


Рис. 4. Схема электрическая принципиальная модуля передачи сигналов датчика приращений

потенциальным выходом. Суппрессоры D7...D10 служат для ограничения входных уровней и дополнительной защиты входных цепей микросхемы дифференциального передатчика U9. R21...R24 обеспечивают согласование длинной линии на основе витых пар***.

На плате приемника предусмотрен вариант, как потенциальных выходов, так и с открытым коллектором. Для питания цепей приемника использован DC-DC преобразователь PUS-0505 (U5). Допустимо использование любого 5-

***** Комментарий автора.**
В качестве кабеля связи рекомендуется использовать следующие марки кабеля:
· FTP 4x2xAWG 24/1,
· S- FTP 4x2xAWG 24/1,
· S- STP 4x2xAWG 24/1.

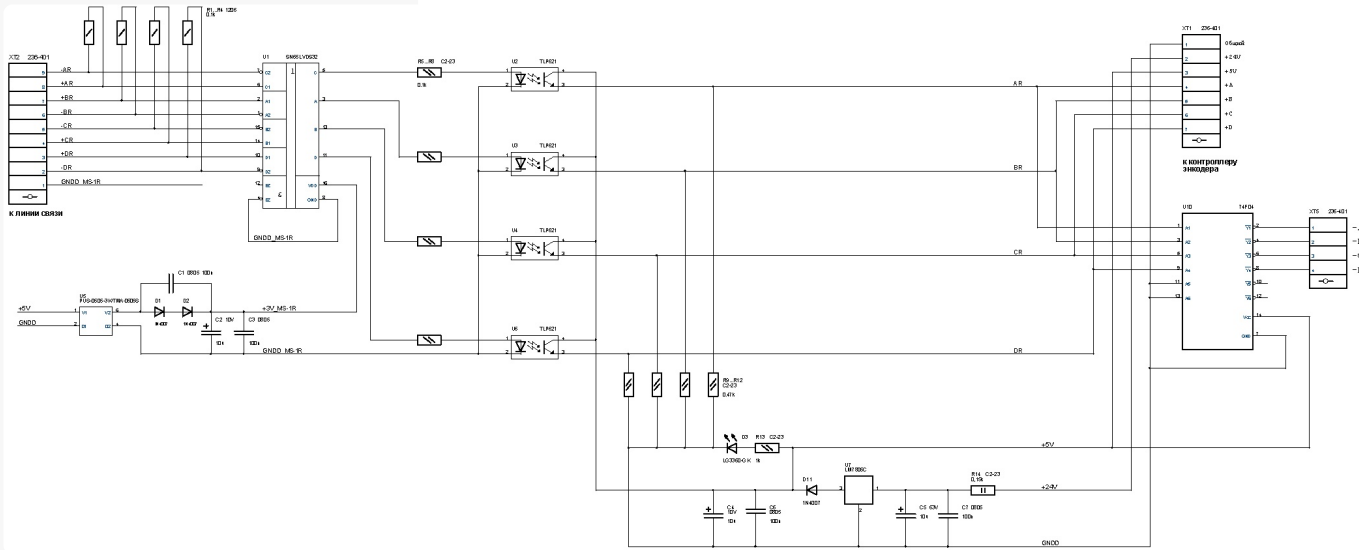
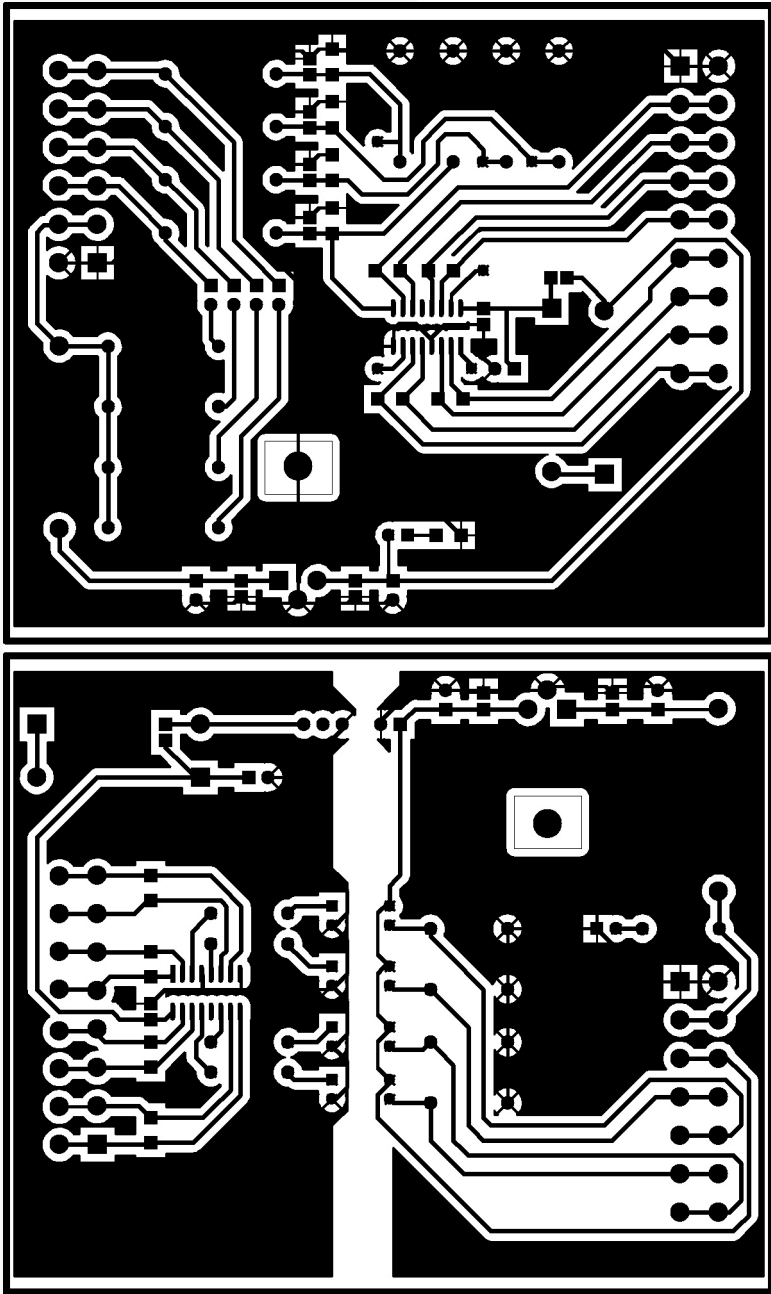


Рис. 5. Схема электрическая принципиальная модуля приема сигналов датчика приращений



DRILL CHART				
SYM	DIAM	TOL	QTY	NOTE
x	0.800 mm		56	
+	1.479 mm		107	
▣	1.800 mm		8	
⌘	3.800 mm		2	
TOTAL			173	

Рис. 6. Печатная плата “вид снизу” модуля передачи и приема (сверху-вниз)

вольтового в корпусе SIP мощностью не менее полуватта. Гальваническую развязку сигналов обеспечивают оптроны TLP621 (U2, U3, U4, U6). Предусмотрено питание схемы как от 5В, так и 24В источника. Элементы R1...R4 обеспечивают согласование длинной линии на основе витых пар. При необходимости инверсии выходных

сигналов предусмотрена схема НЕ на ИМС SN74F04 (U10).

Все резисторы, для упрощения разводки платы, корпусные. Конденсаторы типоразмера 0805. Используемые входные и выходные клеммники – WAGO 236-401 с защелками под отвертку. Платы установлены в корпуса из поликарбоната PHOENIX CONTACT с креплением на DIN рейку. Непосредственно габариты и вся конструкция в сборе представлена на рисунке 8.

Аппаратная часть. Краткое описание контроллера энкодера

Аппаратной основой энкодера датчика приращений служит модуль DIC110 (UNIOxx-5) фирмы Fastwell [2, 3], выполненный в формате MicroPC и установленный в 8-ми слотовое промышленное шасси на базе 486 процессора под управлением DOS. Шасси позволяет установку как ISA, так и PCI плат периферийного ввода-вывода.

Модули UNIOxx-5 имеют 5 разделяемых линий прерываний, канал прямого доступа к памяти (DMA) и светодиод обращения к плате. Внутренняя структура представлена на рисунке 9. Платы UNIO, в зависимости от загруженной прошивки, могут выполнять цифровой/частотный ввод-вывод, аналоговый ввод-вывод (через модули Grayhill), измерение частоты и многие другое. Прошивка изменяется программно, благодаря чему разработчики получают уникальную возможность решать с помощью одной платы множество задач. Если взглянуть на структуру, то сразу видно, что подобные возможности реализуются в основном благодаря наличию ПЛИС. Каждая матрица обслуживает 24 канала ввода-вывода. Загрузка схем матриц производится при включении питания или аппаратном сбросе (RESET) из электрически перепрограммируемого постоянного запоминающего устройства (EEPROM). Изменение варианта загружаемой

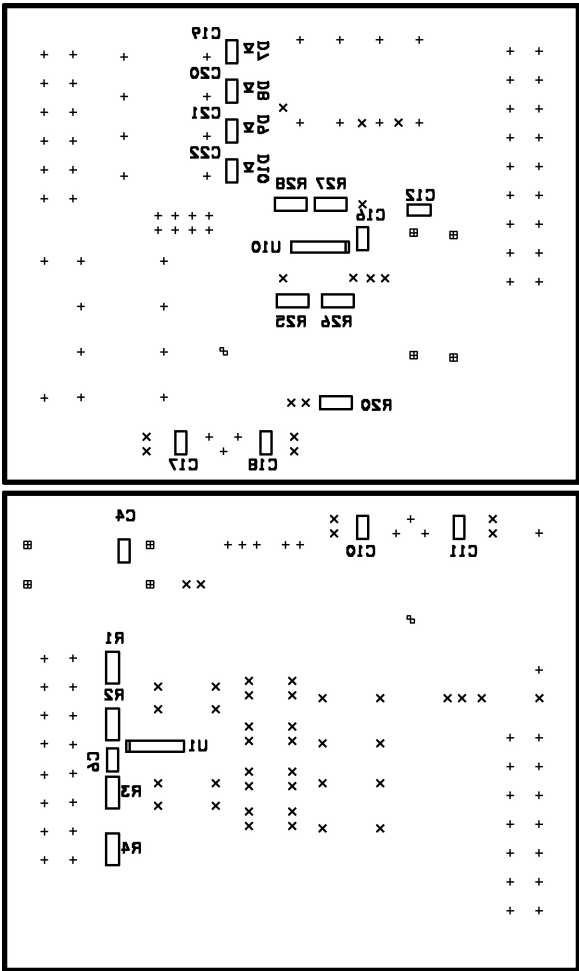
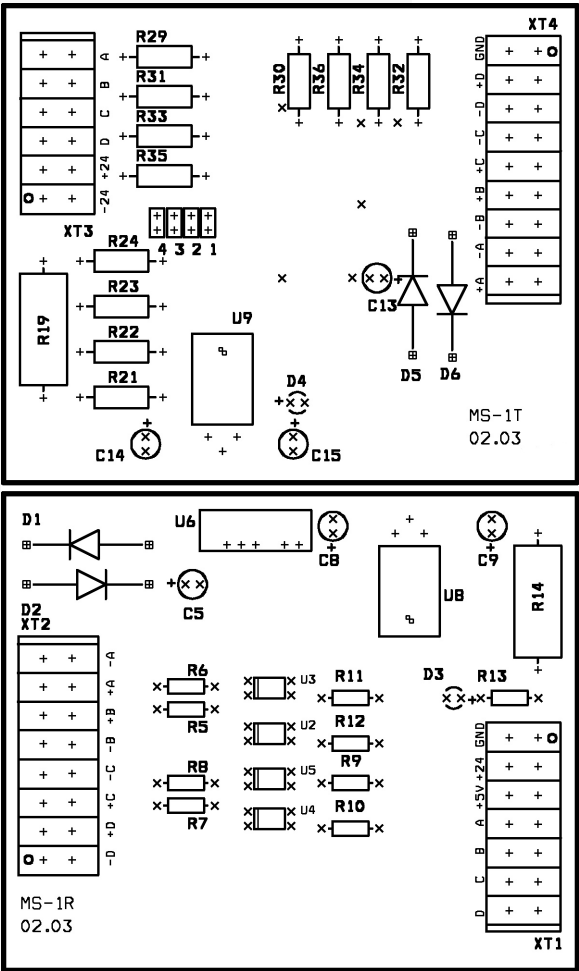


Рис. 7. Расположение элементов “вид сверху и снизу” модуля передачи и приема (сверху-вниз)



Рис. 8. Конструктив модуля передачи и приема импульсных сигналов

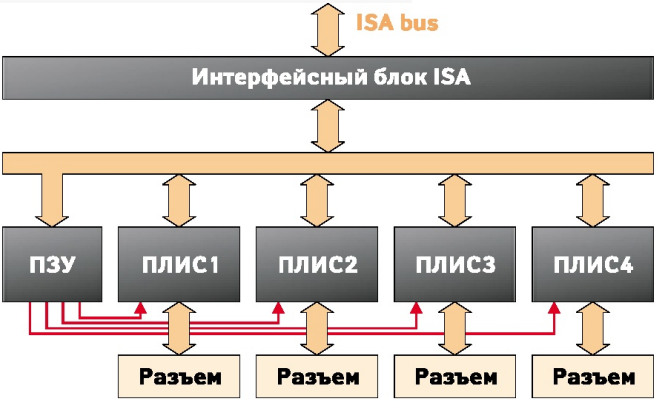


Рис. 9. Структура модуля UNIOxx

схемы и, следовательно, способа обработки сигналов осуществляется перепрограммированием EEPROM непосредственно в системе.

Разработка ПО. Алгоритм энкодера

Итак, приступим к основной задаче. Для работы нам понадобится следующее:

- Turbo C++ IDE ver.3.0 от Borland
- IDE среда Xilinx Foundation Series 3.1i / 6.2i для ПЛИС [4]
- Модуль передачи-приема импульсных сигналов датчика приращений
- Промышленная плата UNIOxx-5 от Fastwell
- JTAG.XILINX программатор из материала [5] или утилита внутреннего загрузчика ISP от FastWell [6]

Энкодер предназначен для декодирования сигналов угловых датчиков положения и передачи информации об угловом положении, скорости, направления вращения, количества

оборо-
тов, а
также
допол-
ните-
льных
сигна-
лов (в
виде
приз-
наков
отказа
кана-
лов, на-
личия

Таблица. Процесс считывания по ISA

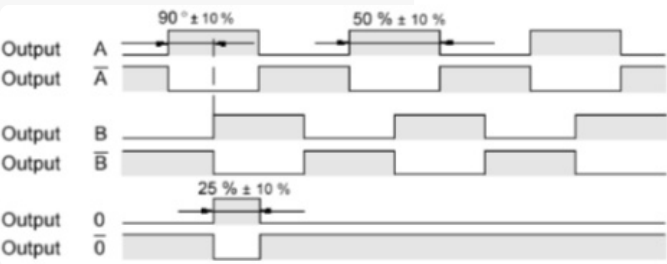
Adress/ # контакта			CSn	WR	D0	D1	D2	D3	D4	D5	D6	D7	Обозначе- ние
A0/1	A1/2	A2/6	63	74	73	72	71	68	64	60	58	56	
IN					OUT (считывание ←)								
0	0	0	x	┐	D ^A ₀	D ^A ₁	D ^A ₂	D ^A ₃	D ^A ₄	D ^A ₅	D ^A ₆	D ^A ₇	угловое положение по каналу А
0	0	1	x	┐	D ^A ₈	D ^A ₉	D ^A ₁₀	D ^A ₁₁	OA	OB	HB	OQ	признаки OA, OB, HB, OQ
0	1	0	x	┐	D ^Q ₀	D ^Q ₁	D ^Q ₂	D ^Q ₃	D ^Q ₄	D ^Q ₅	D ^Q ₆	D ^Q ₇	номер оборота

строба) в виде 8-ми разрядной цифровой последовательности для дальнейшей расшифровки программным путем.

(коммутации) 6-ти внутренних регистров тактовые импульсы (меандр) частотой 50 МГц от внешнего генератора платы UNIOxx-5

Входные сигналы контроллера

Сигналы с каждого датчика шифратора приращений представляют последовательности импульсов, смещенных на 90 градусов по периоду следования. Частота следования импульсов определяет скорость вращения. Максимальная частота – 200 кГц. Максимальное количество импульсов за 1- оборот – 6000. Каждый оборот идентифицируется стробирующим импульсом. На рисунке 10 представлена последовательность импульсов от датчика, поступающих на вход контроллера:



А – прямой и инверсный канал А, В – прямой и инверсный канал В, О – прямой и инверсный строб О (длительность равна четверти периода импульса по каналу А или В ±10%)

Рис. 10. Входные сигналы энкодера

Управляющие сигналы:

- сигнал чтения READ по шине ISA платы UNIOxx-5
- адресация (A0...A2) для организации чтения

Выходные сигналы контроллера

Выходные сигналы для каждой микросхемы FPGA передаются по шине ISA в последовательности, описанной в таблице. Дальнейшая работа с шиной ISA определяется принципиальной схемой и протоколом обмена с микросхемами FPGA. Сформируем основные требования к энкодеру:

- работоспособность при входных частотах канальных импульсов от 0.0 до 200 кГц
- выставление бита направления вращения
- запись данных в буферные регистры по поступлению: сигнала READ и опросе нулевого адреса из шины ISA, и хранение данных до поступления следующего сигнала READ
- измерение количества импульсов, соответствующих угловому положению датчика и реверсом счетчика количества импульсов по направлению вращения
- антидребезговый прием дискретных сигналов каналов (прямого и инверсного А, прямого и инверсного В)
- динамическое изменение ширины защитного интервала для каждого из каналов в пределах длительности канального импульса;
- хранение результата по предыдущему сигналу READ и выдачу текущего значения относительного углового положения, признаков канальных отказов, наличия

строба, количества оборотов, относительной скорости

выдача должна осуществляться путем последовательного выбора (адресации по шине адреса A0...A3) внутренних регистров хранения и записи в виде битовой последовательности DO0...DO7 по портам платы модуля UNIOxx-5

Теперь перейдем к программной части...

Скажем пару слов о среде САПР Xilinx****. Программные средства Xilinx Foundation представляют собой систему сквозного проектирования, которая реализует полный цикл разработки цифровых устройств на базе ПЛИС, включая программирование кристалла. Оболочка Project Navigator предоставляет пользователю интерфейс для работы с проектом и управления всеми процессами проектирования и программирования ПЛИС. Исходные описания проектируемых устройств могут быть представлены в текстовой форме с использованием языков HDL (Hardware Description Language), в виде диаграмм состояний или принципиальных схем. В состав пакета включен схемотехнический редактор и комплект библиотек.

Согласно спецификации на модуль UNIOxx-5 [2, 3], используются ПЛИС FPGA – XC5204. Теперь, запустив IDE Xilinx, создадим новый проект и выберем соответствующую матрицу (см. рисунок 11):

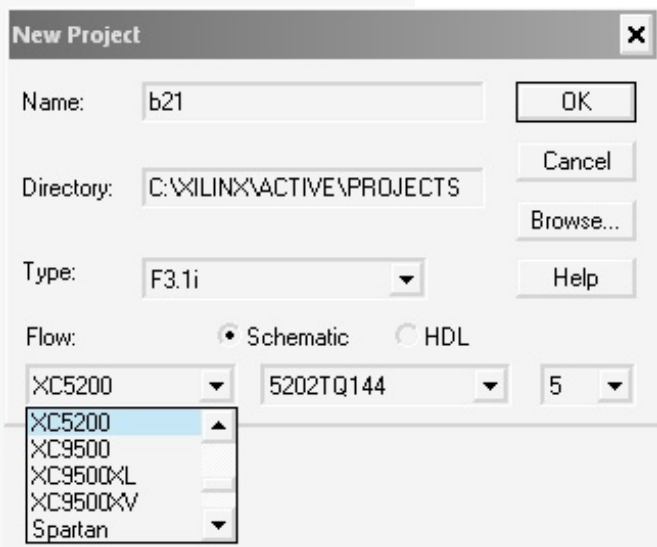


Рис. 11. Создание проекта и выбор параметров в среде Xilinx

Заключение

Следует обратить ваше внимание, что при необходимости передачи импульсных сигналов с датчика приращений на расстояния более чем несколько сот метров (но менее километра), нужно использовать дифференциальные приемопередатчики с драйверами предкоррекции сигналов и компенсацией потерь. В качестве таких приемопередатчиков можно рекомендовать микросхемы LVDS CLC001 и CLC012 фирмы National Semiconductor [8, 9].

**** Комментарий автора.

Если вы впервые столкнулись с данной средой, то сперва рекомендуется ознакомиться с кратким руководством [7] по работе с пакетом.

Продолжение смотрите в следующем выпуске нашего журнала...

Ресурсы

- Спецификация на ИМС SN65LVDS31 (SO-16) <http://www.alldatasheet.com/datasheet-pdf/pdf/28186/TI/SN65LVDS31D.html>
- и спецификация на ИМС SN65LVDS32 <http://www.alldatasheet.com/datasheet-pdf/pdf/28217/TI/SN65LVDS32A.html>
- Fastwel Micro PC compatible UNIOxx-5. Программируемые модули ввода-вывода. Руководство пользователя. – Doc. UNIOxx-5 ver.02.02
- Спецификация на модуль ввода-вывода UNIOXX-5 <http://www.fastwel.ru/content/ecmsfiles/239170.pdf>
- Сайт производителя Xilinx <http://www.xilinx.com>
- С. Бадло. JTAG.XILINX программатор. – Радиолюбитель, Минск, 2008, №7, с.38 <http://raxp.radioliga.com/cnt/s.php?p=jtag.pdf>
- Печатные платы модуля связи GERBER RS-274X <http://programmersclub.ru/pro/pro4.zip>
- Руководство по работе с Xilinx Foundation Series 3.1 <http://raxp.radioliga.com/cnt/s.php?p=x3.zip>
- Спецификация на драйвер CLC001 <http://national.com/pf/CL/CLC001.html>
- Спецификация на драйвер CLC012 <http://national.com/pf/CL/CLC012.html>

Все, так или иначе, сталкивались с капчей. CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) – полностью автоматизированный публичный тест Тьюринга для различения компьютеров и людей. Представляет собой компьютерный тест, используемый для того, чтобы определить, кем является пользователь системы: человеком или компьютером. В этой статье я хочу рассказать о двух методах защиты форума на базе движка phpBB2 от спамеров и их ботов...



by **Arigato**

www.programmersforum.ru/member.php?u=19542

Технология CAPTCHA была создана в 2000 году учеными из университета Карнеги-Меллона, и сегодня используется в Интернете практически повсеместно. Основная идея теста CAPTCHA: предложить пользователю такую задачу, которую легко решает человек, но которую крайне трудно решить компьютеру. Как правило, это задачи на распознавание зрительных образов. Наиболее часто эта технология используется в различных Интернет-сервисах, в частности – хранилищах файловых архивов и форумах.

Уязвимости защиты CAPTCHA

При недостаточной степени защиты скриптов спамбот может пройти тест CAPTCHA и без распознавания картинок. В этом случае он либо подменяет идентификатор сессии, либо парсит информацию, содержащуюся на WEB странице и определяет то, что изображено на картинке. Если количество вариантов ответов невелико, спамбот может «угадать» ответ. Как правило, используется несколько параллельных потоков, благодаря чему скорость перебора зависит от полосы доступного канала. Кроме того, возможно и накопление базы вопросов и ответов, и рано или поздно вся она будет у него.

Существуют программы, распознающие конкретные реализации CAPTCHA, к примеру PWNtcha. Да и никто не мешает спамеру подключать модули программ распознавания текста, тот же FineReader. Различают «сильную» (сильно размытая и не контрастная картинка) и «слабую» CAPTCHA. Что же делать и как защититься от спамботов?

Методы защиты

Первое, что нужно сделать, чтобы прекратить автоматическую регистрацию ботов и массовый спам в темах – это закрыть возможность писать сообщение гостям, поставить картинку (капчу) на форму регистрации и установить активацию учетной записи по E-Mail. Рассмотрим подробнее...

1. Изменение картинки при регистрации на форумах phpBB2

В ходе эксплуатации форума быстро выясняется, что штатных методов для защиты не достаточно. Боты умеют регистрироваться, читать почту и, более того, спокойно распознают картинку, предоставляемую движком форума phpBB2. Если же изменить картинку на более сложную, то можно остановить подавляющее большинство автоматически регистрируемых ботов.

Для этого нужно проделать следующее. В папке `<includes>` форума изменить файл **usercp_confirm.php** и удалить всю часть файла после строчки:

```
// output six seperate original pngs ... first way is preferable!
```

или же строчки (в разных версиях форума могут встречаться несколько отличные друг от друга комментарии):

```
// Thanks to DavidMJ for emulating zlib within the code :)
```

и до конца файла. Вместо удаленной части следует вставить следующее:

```
// Изменение картинки в форме регистрации
// Автор мода: Arigato, 2006
list($usec, $sec) = explode(' ', microtime());
mt_srand($sec * $usec);

$font = getcwd() . "/includes/font.ttf";

$img = ImageCreate (320, 50);

$color = array();
$color[] = ImageColorAllocate ($img, 0, 0, 0);
$color[] = ImageColorAllocate ($img, 255, 0, 0);
$color[] = ImageColorAllocate ($img, 0, 255, 0);
$color[] = ImageColorAllocate ($img, 255, 255, 0);
$color[] = ImageColorAllocate ($img, 255, 0, 255);
$color[] = ImageColorAllocate ($img, 0, 255, 255);
$color[] = ImageColorAllocate ($img, 255, 255, 255);

$sx = ImageSX ($img) - 1;
$sy = ImageSY ($img) - 1;
$sc = count ($color) - 1;

// Фооновый шум:
for ($i = 0; $i < 1024; $i++)
{
    $x = mt_rand (0, $sx);
    $y = mt_rand (0, $sy);
    $c = $color[mt_rand(1,$sc)];
    ImageSetPixel ($img, $x, $y, $c);
}

// Вывод кода:
$xpos = mt_rand (8, 32);
$height = $sy - mt_rand (0, $sy / 4);
for ($i = 0; $i < strlen($code); $i++)
{
    $angle = mt_rand (0, 30) - 15;
    $size = mt_rand (0, 8) + 32;
    $ypos = $sy - mt_rand (8, $sy - $height - 8);
    $c = $color[mt_rand(1,$sc)];
    $rect = ImageTTFtext ($img, $size, $angle, $xpos, $ypos, $c,
    $font, $code[$i]);
    $width = $rect[2] - $rect[0];
    $height = $rect[1] - $rect[7];
    $xpos += $width + mt_rand (4, 48);
}

// Передний шум:
for ($i = 0; $i < 256; $i++)
{
    $x = mt_rand (0, $sx);
    $y = mt_rand (0, $sy);
    $c = $color[mt_rand(0,$sc)];
    ImageSetPixel ($img, $x, $y, $c);
}
for ($i = 0; $i < mt_rand (2, 8); $i++)
{
    $x1 = mt_rand (0, $sx);
    $y1 = mt_rand (0, $sy);
    $x2 = mt_rand (0, $sx);
```

```
$y2 = mt_rand (0, $sy);
$c = $color[mt_rand(0,$sc)];
ImageLine ($img, $x1, $y1, $x2, $y2, $c);
}

header ("Content-type: image/png");
header ("Cache-control: no-cache, no-store");
ImagePng ($img);
ImageDestroy ($img);
?>
```

Кроме этого нужно поместить в папку includes файл со шрифтом font.ttf, архив с которым можно скачать ниже. При желании можно использовать любой другой TrueType шрифт **[1]**, содержащий латинские буквы и цифры. Как показал опыт применения данного метода на многих форумах, автоматическая регистрация ботов после такой модификации прекращается полностью (см. рисунок):



Рис. Вариант замены капчи

2. Разрешить отправлять личные сообщения только пользователям с 20 и более сообщениями на форуме phpBB2 (защита от спама в личку)

Да, сегодня уже и такой вид спама имеет место. Хотя он еще не получил большого распространения, о защите уже пора задуматься. Итак, что нужно делать? Открыть файл **privmsg.php** и найти строчку: `if ($mode == 'newpm')` и добавить выше следующий код:

```
// НАЧАЛО: отправка личных сообщений только для пользователей с
// 20 сообщениями на форуме
// Автор мода: Arigato, 2007
if ( $mode == 'post' || $mode == 'reply' || $mode == 'quote' )
{
    $mod_mes_count = 20;
    if ( $userdata['user_posts'] < $mod_mes_count )
    {
        message_die(GENERAL_MESSAGE, "<b>$userdata[username]</b>,"
        у Вас на форуме <b>$userdata[user_posts]</b>
        сообщений<br/>Отправлять личные сообщения Вы сможете,
        когда наберете <b>$mod_mes_count</b> или более сообщений
        на форуме");
    }
} // КОНЕЦ: отправка личных сообщений только для пользователей с
// 20 сообщениями на форуме
```


Переменная `$mod_mes_count` определяет количество сообщений на форуме, после которого пользователю разрешается отправлять ЛС. Читать ЛС может любой пользователь. Предупреждение о запрете отправки ЛС выводится по-русски, многоязычность не поддерживается.

Заключение

Мы уже выяснили, как убрать основную уязвимость – фиксированный шрифт и фиксированное положение символов. Но, как всегда добавим «ложку дегтя». Дело в том, что сегодня получили распространение сервисы по типу CaptchaExchange Server. Эти сервисы направлены на «обход» картинок CAPTCHA, путем ручного «человеческого» распознавания символов. Принцип работы основан на системе баллов, которые пользователь может заработать, распознавая в свободное время картинки для других пользователей. Набранные баллы пользователь может потратить позже, в любое удобное время, запустив программу автоматического скачивания файлов с файлообменного сервера или для регистраций. К сожалению и второй, рассмотренный нами, метод не дает 100% гарантию, но основная цель будет достигнута – это снижение нагрузки на сервис и относительное спокойствие пользователей.

Ресурсы

- Вариант шрифта для замены на форуме <http://www.programmersforum.ru/attachment.php?attachmentid=5029&d=1217234938> или в архиве с 4-м номером нашего журнала <http://programmersclub.ru/pro/pro4.zip>
- Защита PHPBB от спам-ботов путем изменения каптчи http://frolov.cc/post_1196993870.html
- Защищаем PHPBB от спама и ботов или как изменить стандартное визуальное подтверждение (captcha) в PHPBB <http://handynotes.ru/2007/04/phpbb-captcha-phpbb.html>
- Обсуждение на официальном форуме phpBB. Preventing SPAM - Bots and Humans (на английском языке) <http://www.phpbb.com/community/viewtopic.php?t=427852>
- Вариант защиты с помощью генерации элементов формы на стороне клиента DOM-javascript <http://ir2.ru/dhtml4spam.aspx>



Здравствуйте! В этой статье я хотел бы рассказать о создании анимации на веб-страницах с использованием скриптового языка программирования JavaScript, далее JS. Статья ориентирована прежде всего на новичков...



Алексей Шульга

by **Levsha100** www.programmersforum.ru

По правде говоря я не являюсь поклонником JS, да и веб-программирования тоже. Но родина сказала: «Надо!». Я немного отошел от темы нашего сайта, ибо работы над ним еще ведутся и это, видимо, будет еще долго. Поэтому было решено написать цикл статей по JS и различным интересным трюкам.

Шаблон

Для начала нам необходимо создать заготовку- то место, куда мы будем внедрять скрипты, Вот она:

```
<html><head>
<title>Тестовая страничка</title>
</head>

<body></body>

</html>
```

Это обыкновенная **HTML**- страничка. Я сохранил ее под именем `<Test.html>`.

Встраивание JS-скрипта

Итак, шаблон у нас есть, и теперь мы можем смело пробовать оживлять страничку с помощью скриптов. Добавить JS-скрипт можно двумя основными способами: добавлением его непосредственно в веб-страницу, либо вынести его в отдельный файл. Для наглядности, я выбрал первый способ. Добавим скрипт в секцию `<body>`, после чего она примет вид:

```
<body>

    <script type="text/javascript">
        // Тут находится JS-код
    </script>

</body>
```

Для проверки можно заменить за-

комментированную строку такой строкой:

```
alert("Hello, world");
```

Этот код выведет **модальное** окно с текстом **"Hello, world"**.

Добавление объектов

Прежде чем мы займемся анимацией, нужно чтобы было то, что будет анимироваться. Поэтому займемся созданием анимируемых объектов. Для примера, возьмем квадратные DIV-вы со стороной 10 пикселей. Прежде чем создавать собственно объекты, для них необходимо задать стиль. Это делается с помощью CSS, например:

```
<head><title>Тестовая страничка</title>

<style type="text/css">
    .obj1style{position:absolute; width:10; height:10;
        background-color: #00FF00}
</style>

</head>
```

Наиболее важным свойством – является свойство **position** и именно его значение **absolute** позволяет размещать объект, к которому применен данный стиль, в любой точке веб-страницы. Свойства **width** и **height**, как не трудно догадаться, отвечают за высоту и ширину объекта соответственно. Цвет фона объекта – зеленый.

Теперь займемся собственно добавлением объектов, для этого необходимо заменить строку:

```
// Тут находится JS-код;
```

на строки:

```
var obj_count=10; // Сколько у нас будет объектов
// Цикл от 1 до количества создаваемых объектов
for (i=1;i<=obj_count;i++){
    document.write('<div class="obj1style" style="left:');
    document.write(i*20+50);
```

```
document.write('<div style="position: absolute; top: 50px; left: 0px; width: 100px; height: 100px; background-color: #cccccc;" id="obj1"></div>');
document.write(i);
document.write('<br>');

```

```
// вот то самое место, именно тут мы вписываем в страницу 10
// DIV-ов с классом obj1style идентификаторами с именами от
// obj1-1 до obj1-10. Все элементы располагаются на одной
// высоте, отступ от левой стороны страницы высчитывается по
// формуле i * 20 + 50, где i - это номер элемента [1..10].
}

```

Вместо данного кода мы могли бы написать что-то вроде:

```
<div class="obj1style" style="left: 70px; top: 50px;" id="obj1-1"></div>
...
<div class="obj1style" style="left: 250px; top: 50px;" id="obj1-10"></div>

```

Но это неэффективно и некрасиво, да и лень мне писать так много.

I like to move it или анимируем!

Самое интересное. Итак, для начала зададимся целью, что необходимо сделать. Я решил начать с простого – объекты будут летать в некоем сосуде и при ударе об стенку они будут отскакивать*. Для начала, после строки:

```
var obj_count=10;

```

добавим следующий код:

```
var VX = new Array(obj_count);
var VY = new Array(obj_count);

document.write('<div style="position: absolute; left: 45px; top: 40px; width: 300px; height: 300px; background-color: #bbbbbb;" id="Area"></div>');

```

Первые две строки создают два массива длиной 10 элементов, которые хранят значение скорости для каждого объекта. Третья же строка размещает на страничке «сосуд», в котором будут летать наши «молекулы». Тут, в принципе, я думаю все знакомо и просто. Далее, после строк начальной инициализации:

```
document.write('<div style="position: absolute; top: 50px; left: 0px; width: 100px; height: 100px; background-color: #cccccc;" id="obj1"></div>');
document.write(i);
document.write('<br>');

```

добавляем код, который будет задавать случайную

* Комментарий автора.

...кстати, данную модель можно было бы назвать «Модель идеального газа», ибо данная анимация действительно в некой степени моделирует поведение частиц идеального газа.

скорость для каждого объекта:

```
VX[i] = Math.round(Math.random()*6)-3; if(VX[i]==0){VX[i]=1;}
VY[i] = Math.round(Math.random()*6)-3; if(VY[i]==0){VY[i]=1;}

```

Условия «if» нужны для того, чтобы не было нулевых скоростей, то есть, чтобы не было неподвижных объектов. Теперь перейдем к самому сердцу нашего аниматора – функции **Timer()**. Она периодически, с интервалом 50мс, вызывает функцию **Animate()**, но о ней позже.

Функция выглядит так:

```
function Timer()
{
    setTimeout(function(){Animate();
    setTimeout(arguments.callee, 50);}, 0);
}

```

Эта функция должна вызываться (стартовать) при загрузке страницы. Для того, чтобы обеспечить такую возможность – мы подправим тег <body>. Итоговый вид после изменений показан ниже:

```
<body onload="Timer()">

```

Вроде разобрались, пора браться за последнюю в данном уроке и самую массивную функцию – функцию **Animate()**. Вот полный ее текст:

```
function Animate()
{
    for(i=1;i<=obj_count;i++) {
        document.getElementById("obj1-" + i).style.left=parseInt(document.getElementById("obj1-" + i).style.left)+VX[i];
        document.getElementById("obj1-" + i).style.top=parseInt(document.getElementById("obj1-" + i).style.top)+VY[i];
        if (parseInt(document.getElementById("obj1-" + i).style.left) <=
            parseInt(document.getElementById("Area").style.left)) {
            document.getElementById("obj1-" + i).style.left=document.getElementById("Area").style.left;
            VX[i]*=-1;
        }
        if (parseInt(document.getElementById("obj1-" + i).style.left)+10
            >=
            parseInt(document.getElementById("Area").style.left)+parseInt(

```

```
document.getElementById("Area").style.width)) {
document.getElementById("obj1-"+i).style.left=parseInt(
document.getElementById("Area").style.left)+
parseInt(document.getElementById("Area").style.width)-10;
VX[i]*=-1;
}
if (parseInt(document.getElementById("obj1-"+i).style.top) <=
parseInt(document.getElementById("Area").style.top)){
document.getElementById("obj1-
"+i).style.top=document.getElementById("Area").style.top;
VY[i]*=-1;
}
if (parseInt(document.getElementById("obj1-"+i).style.top)+10 >=
parseInt(document.getElementById("Area").style.top)+
parseInt(document.getElementById("Area").style.height)){
document.getElementById("obj1-"+i).style.top=parseInt(
document.getElementById("Area").style.top)+
parseInt(document.getElementById("Area").style.height)-10;
VY[i]*=-1;
}
}
}
```

Как она работает? Очень просто! В цикле мы пробегаем по всем элементам, при этом выполняем следующие действия:

1. Прибавляем к X координате объекта его горизонтальную скорость, которую берем из массива VX.
2. Прибавляем к Y координате объекта его вертикальную скорость, которую берем из массива VY.
3. Проверяем, не вышел ли левый край объекта за левый край сосуда, если вышел, то ставим объект на границу и инвертируем скорость. Т.е., если летел, к примеру, шарик и он вылетел за левую границу, то мы возвращаем шар на край сосуда и пинаем его в обратную сторону, так, что модуль его горизонтальной скорости не меняется, меняется только знак.
4. Аналогично, только мы проверяем вылет за правую границу.
5. В принципе тоже самое, только здесь все происходит в вертикальной плоскости, мы определяем вылет за верхнюю границу и если таковой факт был, то инвертируем вертикальную составляющую скорости.
6. Аналогично действию №5, за исключением того, что проверка происходит на вылет за нижнюю грань.

Смело копируем эту функцию перед функцией **Timer()** и пробуем запустить нашу маленькую демонстрацию (см. рисунок):

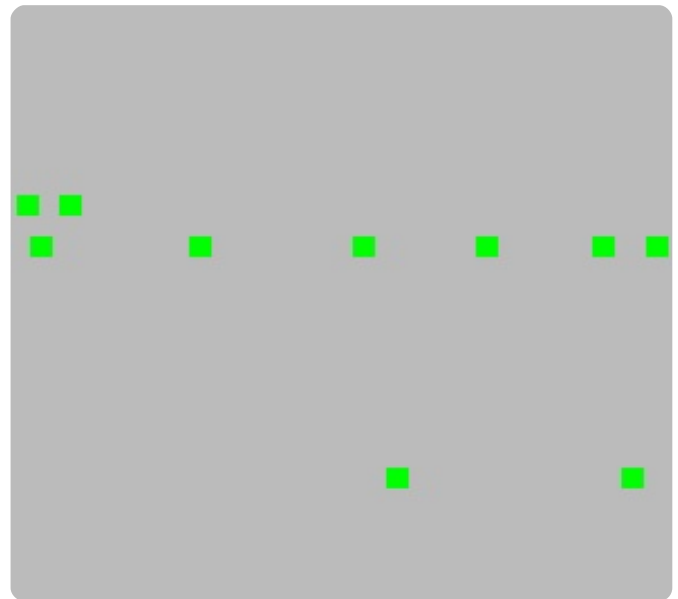


Рис. Тестовая анимация

Заключение

В этой статье я попытался объяснить базовые принципы анимации с помощью JavaScript. Если Вам что-то непонятно, то не расстраивайтесь. В следующих уроках курса, я, на более интересных и практичных примерах, покажу как делать вещи, которые будет не стыдно поставить на свой сайт. А пока – экспериментируйте!

Также хотелось-бы написать статью об оптимизации кода, ибо наш вариант далеко не идеален, поэтому – ждите.

Если возникнут какие-либо вопросы, то обращайтесь на форум www.programmersforum.ru, либо пишите на почтовый ящик редакции. Я или кто-либо другой попытаемся Вам помочь.

Здравствуйте читатели нашего журнала. Сегодня мы хотим напомнить вам, что продолжается прием заявок на участие в конкурсе по созданию лучшего бота для игры в Fortress2 с денежным призом. Организатор конкурса – Форум программистов www.programmersforum.ru...



Руслан Аблязов

by **груЗиН** <http://pblog.ru/?author=8>

Почему надо участвовать в этом конкурсе?

Во-первых, это интересно, вы можете поучаствовать в конкурсе, где не нужно загружать данные из текстового файла и сохранять их туда! Во-вторых, можно получить денежный приз, толстовку или футболку от клуба. В-третьих, вы получите опыт в создании ИИ для игры, и сможете сказать «Я разрабатывал бота для игры!»

Призы как денежные – **3000 рублей**, так и сувениры от клуба на **2000 рублей**. Конкурс рассчитан на 2-4 месяца. Первая битва ботов состоится **15 июля** 2010 года.

Ключевые понятия игры

Итак, вы хотите написать бота для этой игры, но не знаете как. Что пригодится для написания бота для игры:

1. Компилятор, который позволяет компилировать DLL файлы и разумеется знание языка этого компилятора
2. Знание правил игры

* Комментарий автора.

Для тех, кто не в курсе: бот представляет собой DLL с тремя экспортируемыми функциями. Документация по созданию бота находится в файле Fortress 2 Bot Specification

- . Скачать Fortress 2 build 2025 beta + Документация + исходник SimpleBot v1.0 <http://programmersforum.ru/attachment.php?attachmentid=23688&d=1270876754>
- . Скачать исходники SimpleBot v1.0 на C++ (CodeBlocks+MinGW) <http://programmersforum.ru/attachment.php?attachmentid=23689&d=1270876964>
- . Скачать документацию по созданию ботов <http://pkonkurs.ru/wp-content/uploads/2010/06/Fortress-2-Bot-Specification.zip>
- . Скачать исходник бота на C++ (CodeBlocks+MinGW) <http://pkonkurs.ru/wp-content/uploads/2010/06/SimpleBotCpp.zip>
- . Исходник бота на Delphi поставляется в комплекте с игрой.

3. Знание, какие функции должны присутствовать в DLL, для того чтобы бот смог работать

С компилятором я думаю, проблем не будет он может быть любой, главное условие, чтобы он смог компилировать функции по соглашению stdcall. Сначала поясню как вообще происходит игра. играют двое, у каждого игрока есть база, есть три типа ресурсов, есть щит, есть проекты. Проекты бывают разные: атака вражеской базы, развитие своей базы, ремонт базы, шпионаж. Всего проектов 50. В начале игры игрок выбирает 15 проектов, которыми он будет играть. Потом игроки по очереди выбирают проекты, игрок может выбрать только тот проект, на который хватает ресурсов. Проигрывает тот игрок, броня базы которого станет равной нулю.

Какова ее физика с точки зрения ботописателя? Читаем внимательно!

Мы выбираем двух ботов и нажимаем начать игру, ядро игры вызывает функции StartGame ботов участвующих в игре. Задача функции StartGame это выбор набора проектов, которыми будет играть бот, набор проектов надо сохранить в массиве, указатель на который будет передан функции StartGame. Проекты задаются числами (полный набор проектов указан в документации). После выбора проектов начинается игра. В ходе игры по очереди вызываются функции GetTurn каждого бота. В случае, если боту недоступен ни один проект, то функция GetTurn вызывается с нулевым указателем на массив доступных проектов. В конце игры еще раз вызывается функция StartGame с нулевым указателем на массив проектов. Нулевой указатель на массив свидетельствует о том, что происходит уведомление бота о конце игры.

Количество ресурсов (энергия, металл,

электроэлементы) зависит от количества объектов их добывающих (батареи, рудники, лаборатории). Количество батарей и рудников у врага можно уменьшить используя специальные шпионские-проекты. Внимание, нововведение в Fortress 2 : введено ограничение на количество лабораторий. По-умолчанию оно равно 5. Как только броня базы становится больше 50, к лимиту прибавляется одна лаборатория за каждые 5 единиц брони свыше 50. Например:

```
Base = 30, LabLimit = 5;
Base = 54, LabLimit = 5;
Base = 56, LabLimit = 6;
Base = 63, LabLimit = 7;
Base = 75, LabLimit = 8;
Base = 81, LabLimit = 9;
```

...и т.д. В случае использования проектов, которые увеличивают количество лабораторий и при этом уже достигнут лимит количества лабораторий, проект считается использованным (ресурсы расходуются), а количество лабораторий не изменяется.

Отличия от первой версии игры

1. Добавлены уведомления о пропуске хода и о конце игры и ее результатах
2. Бот может узнать имя противника (см. структуру TAdditionalGameInfo)

Возможности улучшения бота SimpleBot v1.1 на C++

Покажем, как можно изменить игру бота SimpleBot v1.1 в лучшую сторону, всего лишь добавив несколько строчек кода. Итак, приступим. Что мы сделаем в первую очередь? Изменим набор проектов. Смотрим строку в коде, которая содержит список проектов [1, 2]:

```
const int Projects[MaxProjectsToPlayer] = {5, 6, 11, 13, 16, 18,
20, 28, 33, 34, 35, 44, 45, 46, 50};
```

Что тут можно изменить? В принципе можно тут изменить все, но заморачиваться мы не будем, просто удалим проект номер 50 и вставим (29) СуперАтака 2. En 16, Me 14, El 6 : Base-20, Shield-

15. В итоге получаем такой список проектов:

```
const int Projects[MaxProjectsToPlayer] = {5, 6, 11, 13, 16, 18,
20, 28, 29, 33, 34, 35, 44, 45, 46};
```

Теперь смотрим функцию хода бота:

```
if (!aAvailProjects) return 0;
int OPP = GetOtherPlayer(aPlayerNumber);

if ((IsProjectAvail(20,aAvailProjects)) and
(aGame[aPlayerNumber].Base<25))
return 20;

if ((IsProjectAvail(18,aAvailProjects)) and
(aGame[aPlayerNumber].Base<20))
return 18;

if ((IsProjectAvail(16,aAvailProjects)) and
(aGame[aPlayerNumber].Shield<5))
return 16;
```

Первые две строки это проверка на уведомление о пропуске хода и получение индекса противника. Потом идет проверка доступности проектов 20, 18, 16:

```
(16) Ремонт 2. En 9, El 2 : SS+10
(18) Ремонт 4. Me 7, El 2 : SB+9
(20) Ремонт 6. Me 8, El 5 : SB+12
```

т.е. у бота приоритет : сначала проверить состояние базы, если состояние плохое выбираем ремонтный проект. изменять здесь ничего не будем. Смотрим далее. Далее идут при проверки и использование проектов увеличения количества батарей, рудников и лабораторий:

```
if (IsProjectAvail(34,aAvailProjects) and
(aGame[aPlayerNumber].Mines<3) and
(aGame[aPlayerNumber].Base>35) and
(aGame[aPlayerNumber].Shield>2))
return 34;

if (IsProjectAvail(33,aAvailProjects) and
(aGame[aPlayerNumber].Battery<4) and
(aGame[aPlayerNumber].Base>35) and
(aGame[aPlayerNumber].Shield>2))
return 33;

if (IsProjectAvail(35,aAvailProjects) and
(aGame[aPlayerNumber].Labs<3) and
(aGame[aPlayerNumber].Base>35) and
(aGame[aPlayerNumber].Shield>2))
return 35;
```

т.е. если состояние базы нормальное то можно

увеличить количество батарей, рудников и лабораторий. Но как видно что количество батарей, рудников и лабораторий будет не больше 4,3,3 соответственно. Эти строчки мы менять не будем, добавим дополнительные строчки отвечающие за усиленное развитие базы:

```
if (IsProjectAvail(34,aAvailProjects) and
(aGame[aPlayerNumber].Mines<6) and
(aGame[aPlayerNumber].Base>45) and
(aGame[aPlayerNumber].Shield>10))
return 34;

if (IsProjectAvail(33,aAvailProjects) and
(aGame[aPlayerNumber].Battery<7) and
(aGame[aPlayerNumber].Base>45) and
(aGame[aPlayerNumber].Shield>10))
return 33;

if (IsProjectAvail(35,aAvailProjects) and
(aGame[aPlayerNumber].Labs<5) and
(aGame[aPlayerNumber].Base>45) and
(aGame[aPlayerNumber].Shield>5))
return 35;
```

Теперь количество батарей, рудников и лабораторий будет увеличиваться до 7,6,5 соответственно. Развитие базы будет осуществляться только в том случае если состояние базы очень хорошее. В набор проектов мы добавили проект номер 29, вопрос куда вставить его обработку чтобы не ухудшить игру бота. После проверок на развитие базы есть такие строки:

```
if (IsProjectAvail(46,aAvailProjects)) return 46;
if (IsProjectAvail(28,aAvailProjects)) return 28;
```

в первую очередь обрабатывается наличие проекта номер 46 а потом 28. Вставим проверку проекта номер 29 между ними:

```
if (IsProjectAvail(46,aAvailProjects)) return 46;
if (IsProjectAvail(29,aAvailProjects)) return 29;
if (IsProjectAvail(28,aAvailProjects)) return 28;
```

Компилируем бота, запускаем игру, ставим 500 игр и смотрим результат (см. рисунок 1). Результат очевиден, выигрыш в 84% игр. Но нет предела совершенству.

Возможности улучшения бота SimpleBot v1.2 на Delphi

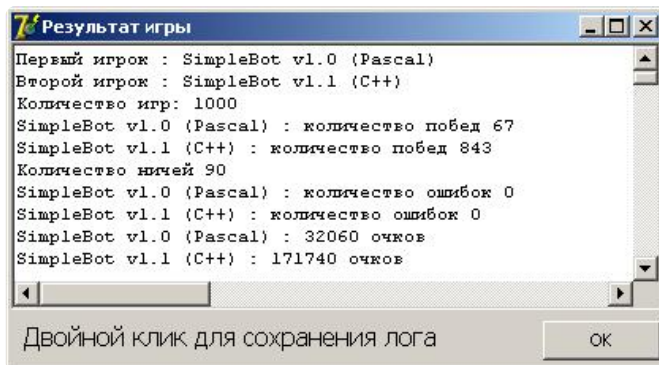


Рис. 1. Результаты игры SimpleBot v1.1

Рассмотрим поведение бота... В начале, бот проверяет состояние базы и если оно плохое, то выбирает проект ремонта базы, проверяет условия начального развития (когда батарей, рудников и лабораторий совсем мало). Затем проверяет условия усиленного развития базы и атакующие проекты, и в самом конце, если ни одно условие не выполнилось, то случайно выбирает проект. Что можно изменить в первую очередь? Очевидно, что условия начального развития базы. Базу сделаем не менее 35 единиц и счит не нулевым. Также слишком жесткие условия усиленного развития базы. Вот их нам и надо изменить. Вот что у меня получилось [3]:

```
if IsProjectAvail(BuildMineProject,aAvailProjects) and
(aGame[aPlayerNumber].Mines<3) and
(aGame[aPlayerNumber].Base>25) then
begin
Result:=BuildMineProject;
exit;
end;

if IsProjectAvail(BuildBatteryProject,aAvailProjects) and
(aGame[aPlayerNumber].Battery<4) and
(aGame[aPlayerNumber].Base>25) then
begin
Result:=BuildBatteryProject;
exit;
end;

if IsProjectAvail(BuildLabProject,aAvailProjects) and
(aGame[aPlayerNumber].Labs<3) and
(aGame[aPlayerNumber].Base>25) then
begin Result:=BuildLabProject;
exit;
end;

if IsProjectAvail(BuildBatteryProject,aAvailProjects) and
(aGame[aPlayerNumber].Battery<7) and
(aGame[aPlayerNumber].Base>40) and
```

```

(aGame[aPlayerNumber].Shield>10) then
begin
  Result:=BuildBatteryProject;
  exit;
end;

if IsProjectAvail(BuildMineProject,aAvailProjects) and
(aGame[aPlayerNumber].Mines<6) and
(aGame[aPlayerNumber].Base>40) and
(aGame[aPlayerNumber].Shield>10) then
begin
  Result:=BuildMineProject;
  exit;
end;

if IsProjectAvail(BuildLabProject,aAvailProjects) and
(aGame[aPlayerNumber].Labs<5) and
(aGame[aPlayerNumber].Base>40) then
begin
  Result:=BuildLabProject;
  exit;
end;

```

Немного хотелось бы сказать про количество лабораторий. Их количество увеличивается только до 5, так как строительство больше пяти лабораторий требует дополнительного строительства базы. Вообще проекты спроектированы так, что на шпионские проекты надо много электроэлементов и если у вас мало электроэлементов вы не сможете использовать сильные шпионские проекты. Правильное развитие на лаборатории и использование шпионских проектов может стать решающим фактором в победе вашего бота.

Теперь посмотрим список проектов выбираемых ботом. Удалим 28 проект и добавим (31) СуперАтака 4. En 26, Me 16, El 10 : ->49 в итоге получаем такой список проектов:

```

const STRATEGY:array[0..MaxProjectsToPlayer-1] of integer =
(5, 6, 11, 13, 16, 18, 20, 28, 31, 33, 34, 35, 44, 45, 46);

```

И вставим проверку использования 31 проекта перед всеми атакующими проектами, т.е. его наличие будет обрабатываться в первую очередь:

```

if IsProjectAvail(31,aAvailProjects) then
begin
  Result:=31;
  exit;
end;

```

```

if IsProjectAvail(46,aAvailProjects) then
begin
  Result:=46;
  exit;
end;

if IsProjectAvail(29,aAvailProjects) then
begin
  Result:=29;
  exit;
end;

```

Проверку наличия проектов 45 и 44 оставим без изменения, а вот в конце при случайном выборе проекта изменим число 15 на 8, таким образом, случайно будут выбираться только первые 8 проектов из списка доступных проектов, а именно проекты мелких атак и ремонта базы:

```

repeat
  Result:=aAvailProjects^[random(8)];
until Result<>0;

```

Компилируем и смотрим результаты (см. рисунок 2). Результат – выигрыш более 60% игр:

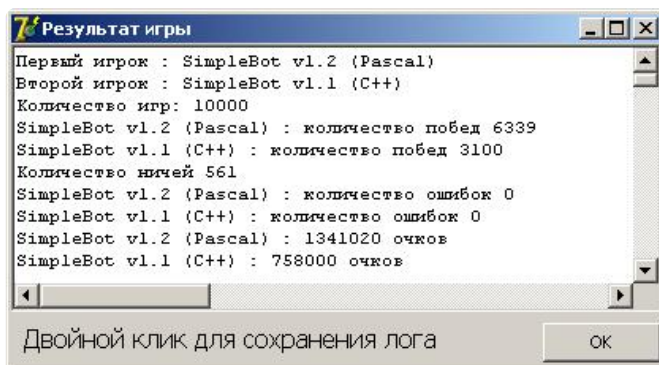


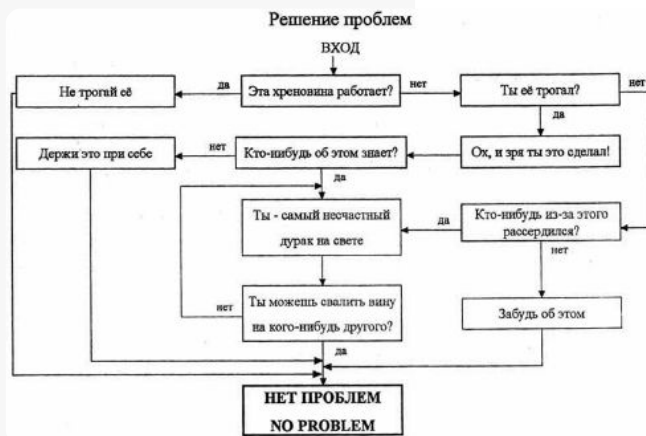
Рис. 2. Результаты игры SimpleBot v1.2

Ресурсы

- Скачать исходники SimpleBot v1.1. (C++) http://pkonkurs.ru/wp-content/uploads/2010/06/SimpleBotCpp_v11.zip
- Скачать Fortress 2 build 2026 + FortUI build 1004 <http://pkonkurs.ru/wp-content/uploads/2010/06/Fortress-2-2026-+-FortUI-1004.zip>
- Скачать Fortress 2 build 2026 beta + SimpleBot v1.2 <http://pkonkurs.ru/wp-content/uploads/2010/06/Fortress-2-2026-+-FortUI-1004-+-SimpleBot-v1.2.zip>

Под редакцией Сергея Бадло...

Эта схема позволяет решить любые проблемы в максимально короткие сроки:



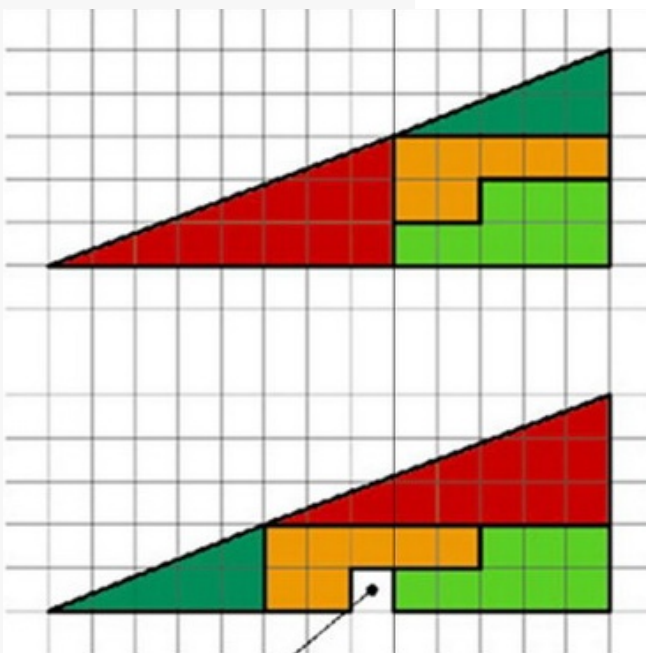
А как Вы решаете свои проблемы?

Веселые стихи (читать надо вслух):

2 15 42
42 15
37 08 5
20 20 20!

38 46
0 4 20
7 08 33
20 20 20!

45 108 2
47 16
3 4 502



Как это объяснить?

20 20 20!

7 14 100 0

0 0 0 13

37 08 5

20 20 20!

Философия на пальцах

1. Платонизм. Я вспомнил! У меня есть пальцы!
2. Неоплатонизм. У меня есть пальцы! Но это вспомнил не я...
3. Атомизм. Пальцы есть, но только очень маленькие, и их очень много.
4. Киники. Пальцы есть. Но зачем?
5. Стоицизм. Пальцы неизбежны.
6. Иудаизм. Мои пальцы - всем пальцам пальцы!
7. Зороастризм. Есть пальцы левые, есть пальцы правые и их поровну.
8. Индуизм. Каждому пальцу - по карме!
9. Буддизм. Пальцы бренны - так на фиг они нужны?
10. Конфуцианство. Пальцы. Просто пальцы.
11. Даосизм. От пальцев никуда не денешься.
12. Христианство. Пальцев пять, но ладонь-то одна!
13. Христианская ересь. А пальцев-то не пять!
14. Средневековая философия. Пальцы непостижимы.
15. Философия Возрождения. А пальцы-то есть!
16. Ислам. Нет пальцев кроме моих.
17. Сенсуализм. Если ударить по пальцам и будет больно, то они есть, а если не больно - то их нет.
18. Идеализм. Пальцы есть, потому что я думаю, что они есть.
19. Субъективный идеализм. Вот перестану думать о пальцах - и они исчезнут!
20. Агностицизм. Пальцы-то есть, но вот, поди, это докажи...
21. Материализм. Пальцы есть, потому-то я о них и думаю.
22. Диалектический материализм. Единство и борьба правых и левых пальцев.
23. Рационализм. Пальцы есть. Их не может не быть.
24. Скептицизм. Поди, разберись в этих пальцах!
25. Детерминизм. Это, смотря какие пальцы.

26. Просвещение. А что ты сделал для своих пальцев?!
27. Гегельянство. Пальцы есть! Но непонятно – как?!
28. Ницшеанство. Не стоит долго глядеть на свои пальцы, иначе однажды они взглянут на тебя.
29. Марксизм. Это как два пальца.
30. Марксизм - ленинизм. Это как два пальца об асфальт.
31. Иррационализм. А есть ли пальцы?
32. Позитивизм. Пальцы пальцами, однако...
33. Экзистенциализм. Где-то у меня были пальцы...

Самолет летит со скоростью 800 м/сек. Его догоняет вражеский истребитель. А мы в хвосте установили пулемет и стрельнули по врагу. Скорость вылета пули из ствола, допустим, 750 м/сек. Что будет? По отношению к Земле получается, что пуля должна двигаться вслед за нашим самолетом со скоростью 50 м/сек., т.е. лететь «задом-наперед» ?



...прозванивается тестером и показывает от 300 Ом до 1.5 кОм, в зависимости от точек приложения. Материал хрупкий, при попытке пилить ножовкой крошится. Не горит, бытовой дозиметр наличие радиоактивности не показывает. Что это может быть?

Слепая печать - зло! Сижу, перепечатаваю книгу (сестре для учебы надо)

Минусы УПС-а: дома электричество отключили, пля, только через 20 минут заметил...



- Из-за вашего дурацкого аджепишного акселя у меня скажишник писайный с идешником райдовским бошками сйерориваются.

- Что лучше, AMD или Intel ?

- Смотря куда совать...

Такое ощущение, что наши разработчики боевой техники и вооружения немножко издеваются над своими зарубежными коллегами. В смысле названий создаваемой ими техники. Вот у Германии есть танк «Леопард». У Израиля - «Меркава» (Боевая колесница). У Америки танк «Абрамс», у Франции «Леклерк», оба в честь знаменитых генералов.

А у нас - Т-72Б «Рогатка». В честь рогатки. Не понятно почему, зато понятно, что КВН мог родиться только у нас.

Или, например, берут американцы и называют свою самоходную гаубицу «Паладин». А англичане свою называют «Арчер» (Лучник). Все путем. Тут подходят наши и говорят: смотрите сюда. Вот самоходные гаубицы 2С1 «Гвоздика», 2С3 «Акация», самоходный миномет 2С4 «Тюльпан» и дальнобойные самоходные пушки 2С5 «Гиацинт» и 2С7 «Пион», способные стрелять ядерными снарядами. Нюхайте, пожалуйста, букет. А чтоб вам совсем стало непонятно и страшно, была у нас еще ракета под названием «Кромка».

А чтоб вы еще больше задумались, тяжелую боевую машину поддержки танков мы назвали

«Рамка». А чтоб у вас башка закружилась, новейший ракетный комплекс береговой обороны мы назвали «Бал».

А чтоб у вас идиотская улыбка на репе образовалась, наш самый мощный в мире 30-ствольный самоходный огнемет ТОС-1 называется «Буратино».

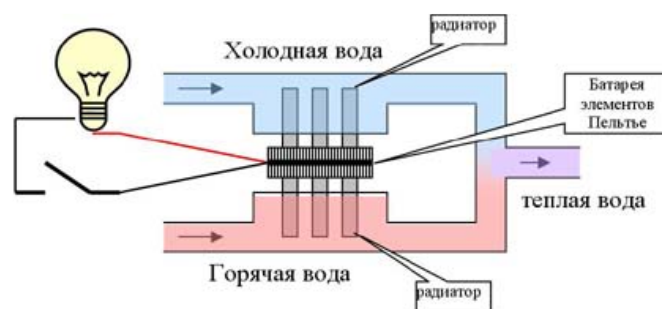
А чтоб вас прям сегодня же в дурдом увезли - наш подствольный гранатомет ГП-30 имеет название «Обувка».

А ежели что, то есть еще 82-мм автоматический миномет 2Б9 Василек», ротный миномет 2Б14 «Поднос», миномет 2С12 «Сани», межконтинентальная баллистическая ракета «Курьер» с ядерным зарядом, межконтинентальная баллистическая ракета РТ-23 УТТХ «Молодец» с десятью ядерными зарядами, атомная подлодка проекта 705 «Лира», система управления артиллерийским огнем «Капустник», контейнерная система управления ракетами «Фантазмагория», самоходное орудие «Конденсатор» и граната для подствольного гранатомета 7П24 «Подкидыш». «Капюшон» - самонаводящийся боевой противотанковый элемент для кассетной БЧ в артиллерии. «Полуфинал» - бесконтактный взрыватель 9Э343. «Окаменелость» - что-то из войск связи или ПВО... Программно-технический радиолокационный комплекс активного воздействия на гидрометеорологические процессы «Хмарка».

Ну и просто нетривиальные ассоциации: артиллерийская АСУ полкового уровня «Успех», бесшумный АГК «Канарейка» (6С1), САУ 2С9-1 «Свиристелка», противопожарная система «Иней» (ЗЭЦ13, кстати, она действительно настолько эффективна, как по названию можно подумать) и противонапалмовая система «Сода», радиостанция «Баян» (Р-135 на шасси Урал-375Д), 125-мм БПС «Заколка» (ЗБМ22)...

Холодная и горячая вода омывает радиаторы приделанные к разным сторонам батареи

элементов Пельтье. Температуры холодной и горячей воды на входе постоянны.



Что будет с температурой воды на выходе, если включить лампочку:

- повысится?
- понизится?
- не изменится?
- произойдет что-то другое?

А что будет с этой температурой, если закоротить элемент Пельтье?



...шарф сисадмина.

Надпись на магазине электротоваров: «Энергосберегающие лампочки без ГМО»