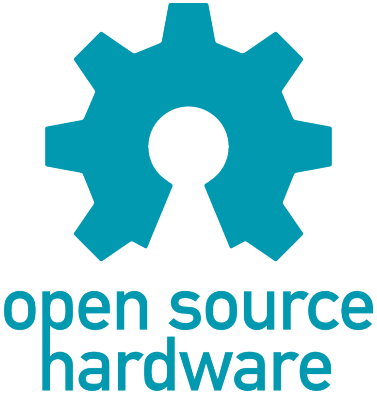


Азбука халтурщика-ARMатурщика
разработка встраиваемых систем

основы бытовой автоматики,
систем управления и сбора данных

© ruOpenWrt
© HackSpace «Чебураторный завод»
© Консорциум хоббитов России
© Bill Collis (Часть 1)

Оглавление

Введение **2**

I Введение в практическую электронику **3**

An Introduction to Practical Electronics, Microcontrollers and Software Design © Bill Collis 4

1 1 Введение в практическую электронику 13 **5**

1.1 Ваше обучение по специальности «Технология» 7

1.1.1 Цели обучения технологиям Ново-Зеландской программы 7

1.2 Ключевые компетенции Ново-Зеландской программы 8

2 2 Вводная электронная схема 15 **10**

2.1 2.1 Где купить комплектующие? 15 10

2.2 2.2 Определение сопротивления резистора по цветовому коду 16 13

2.3 2.3 Светодиоды 17 13

2.4 2,4 Некоторые технические характеристики светодиода 17 . . . 13
2.5 2.5 Задание на исследование светодиода 17 . . . 13
2.6 2.6 Добавление выключателя в схему 18 . . . 13
2.7 2.7 Задание на установку выключателя 18 . . . 14
2.8 2,8 Важные понятия схемы 19 . . . 14
2.9 2.9 Изменение величины сопротивления 19 . . . 14
2.10 2.10 Добавление транзистора в схему 20 . . . 14
2.11 2.11 Чтение схем 21 . . . 14
2.12 2.12 Входная цепь — LDR 22 . . . 15
2.13 2.13 Рабочая схема датчика темноты 23 . . . 15
2.14 2,14 Защитные цепи - использование диода 24 . . . 15
2.15 2.15 Задача исследования диода 24 . . . 15
2.16 2.13 Финальная схема датчика темноты 23 . . . 15

3 3 Вводное конструирование печатной платы 26 16

4 4 Пайка, припой и паяльники 41 17

5 5 Введение в теорию электроники 49 18

6 6 Введение в электронику микроконтроллера 63 19

7 7 Входные цепи микроконтроллера 91 20

8 8 Обзор программирования 104 21

9 9 Введение в поток выполнения программы 112 22

10 10 Вводное программирование: использование подпрограмм 126 23

11 11 Вводное программирование: Использование переменных 134 24

12 12 Основные дисплеи 161 25

13 13 Проект портативного аудиоусилителя на TDA2822M 174 26

14 14 Основы логического программирования 187 27

15 15 Разработка алгоритма: Система сигнализации 202 28

16 16 Основы теории цепей постоянного тока 215 29

17 17 Основы планирования проекта 236 30

18 18 Пример дизайна системы: Таймер клеевого пистолета 268 31

19 19 Основные интерфейсы и их программирование 273 32

20 20 Основы интерфейса аналого-цифрового преобразования 295 33

21 21 Основы проектирования системы 314 34

22 22 Основы проектирования системы: Тайм-трекер 317 35

23 23 Основы вычислений времени 330 36

24 24 Основы строковых переменных 340 37

25 25 Силовые интерфейсы 353 38

26 26 Теория источников питания 370 39

27 27 Типичные вопросы тестирования 2011/12/13 годов 395 40

28 28 Расширенное программирование: Массивы 397 41

29 29 Подтягивающие резисторы AVR 402 42

30 30 Дополнительноподключение клавиатуры 403 43

31 31 Тонкости циклов Do-Loop & While-Wend 417 44

32 32 Подключение двигателя постоянного тока 423 45

33 33 Пример расширенной системы: Будильник 452 46

34 34 Резистивный сенсорный экран 468 47

35 35 Пример проектирования системы: Регулятор температуры 475 48

36 36 Расширенное программирование: Машины состояний 478 49

37 37 Переработанный проект будильника 501 50

38 38 Студенческий проект: Расширенный оконный контроллер 514 51

39 39 Альтернативные техники кодирования машин состояния 524 52

40 40 Сложно: последовательная связь 526 53

41 41 Цифровой радиоканал 597 54

42 42 Введение в I2C 617 55

43 43 Студенческий проект: Таймер полива теплицы 631 56

44 44 Проект Велосипедного аудиоусилителя 642 57

45 45 Графические LCD 648 58

46 46 Проект Отслеживания температуры GLCD 660 59

47 47 Прерывания 672 60

48 48 Таймеры/Счётчики 692 61

49 49 Проект скроллинга графического LED дисплея: массивы и таймеры 698 62

50 50 Проект медицинского прибора: реализация таймера 709 63

51 Проект часов на 7-сегментном индикаторе реализация на сдвоенном таймере 64

52 52 ИС драйвера дисплея MAX 7219/7221 739 65

53 53 Подключение через мобильную связь: ADH8066 744 66

54 54 Передача данных через Internet 778 67

55 55 Задание: математика в реальном мире 816 68

56 56 Цветной графический LCD на основе SSD1928 825 69

57 57 Светофор: помощь и решение 865 70

58 58 Компьютерное программирование: низкоуровневые детали 869 71

59 59 USB-программатор: USBASP 876 72

60 60 Программатор USBTinyISP 877 73

61 61 Программирование на Си и AVR 881 74

62 62 Объектно-Ориентированное Программирование (ООП) на C_+^+ и AVR 929 75

63 63 Современные (2014) отладочные платы на AVR 953 76

64 64 Eagle: создание собственной библиотеки 970 77

65 65 Практические методы 979 78

66 66 ЧПУ 990 79

67 67 Индекс 1008 80

II Основы электроники 81

68 Линейные схемы на пассивных элементах, основы электротехники 83

69 Симуляция и расчет схем в ngSPICE 84

70 KiCAD 85
70.1 Отрисовка схем в KiCAD 85
70.2 Библиотеки элементов 85
70.3 Передача схемы в ngSPICE 85

71 Простейшие полупроводниковые элементы 86
71.1 Оптоэлектроника 86
71.2 Схемы на биполярных транзисорах 86
71.3 Схемы на на полевых транзисорах 86

72 Операционные усилители 87

73 Источники питания **88**
73.1 Батарейное питание 88
73.2 Линейные стабилизаторы 88
73.3 Импульсные преобразователи на ШИМ-контроллерах 88
73.4 Цепи защиты и гашения кондуктивных помех 88

74 Цифровая электроника **89**

75 Компьютерные интерфейсы **90**
75.1 Поколение 90х: COM, LPT, ISA 91
75.1.1 Резервный программатор AVR “пять проводков” 91
75.2 Сеть CAN 91
75.3 Интерфейсные модули USB 91
75.3.1 Универсальный высокоскоростной конвертер FTDI FT2232H 91
75.3.2 JTAG-адаптер 91
75.3.3 Отладочный модуль CAN 91
75.4 Интерфейсные модули Ethernet 91

76 ПЛИС **92**

77 Датчики **93**

78 Электропривод и исполнительные устройства **94**

III Основы конструирования РЭС 95

79 Пакеты моделирования на основе OpenFOAM 96

80 Обеспечение теплового режима 97

81 Электромагнитная совместимость 98

81.1 Кондуктивные помехи . . . 98

81.2 Компоновочные модели и оптимизация кабельной сети . . . 98

IV Технология РЭС 99

82 Инструменты и оборудование 100

82.1 Паяльное оборудование . . . 100

82.1.1 Паяльник . . . 100

82.1.2 Паяльная станция . . . 102

82.2 JTAG-адаптер . . . 105

82.3 Отладочные платы . . . 105

82.3.1 Arduino /Atmel Mega AVR8/ . . . 105

82.3.2 Cortex-Mx . . . 105

82.3.3 CubieBoard /Cortex-A8 AllWinner A10/ . . . 105

82.3.4 Raspberry Pi /ARM11 BCM3032/ . . . 105

82.3.5 BlackSwift /MIPS/ . . . 105

82.3.6 VoCore /MIPS/ . . . 105

82.4 Радиомонтажный инструмент . . . 105

82.4.1 Pro'sKit . . . 106
82.5 Измерительное оборудование . . . 118
82.5.1 Тестер . . . 118
82.5.2 Осциллограф . . . 118
82.5.3 Логический анализатор . . . 118
82.5.4 Генератор сигналов . . . 118
82.5.5 Рыльцеметр . . . 118
82.6 Электроинструмент . . . 118
82.6.1 Дрелъ . . . 119
82.6.2 Лобзик . . . 122
82.6.3 Жвигатель . . . 123

83 Трассировка плат и подготовка производства в KiCAD 126
83.1 Технология ЛУТ (Лазерный УТюг) . . . 126
83.2 Технология фоторезиста . . . 126
83.3 Формат Gerber и подготвка промышленного производства . . . 126

84 FreeCAD 127
84.1 Установка под ⊞Windows . . . 130
84.2 Чертеж . . . 132
84.3 Эскиз . . . 132
84.4 Деталь . . . 132
84.5 Сборка . . . 132
84.6 Автогенерация конструкторской докуметации . . . 132
84.7 Скрипты и пользовательские расширения . . . 132

85 Эксплуатация станочного оборудования **133**

86 Основы ЧПУ и цифрового производства **134**
86.1 CAM-пакеты для FreeCAD 134

V Основы теории систем автоматического управления **135**

87 Математический аппарат **136**
87.1 Передаточная функция 136
87.2 Устойчивость САУ 136
87.3 Сети Петри 136
87.4 Автоматы Маркова 136

88 Релейное управление **137**

89 Пропорциональные САУ **138**

90 ПИДn-регуляторы **139**

VI Разработка ПО для встраиваемых систем **140**

91 Архитектура программных систем **141**
91.1 Литература 141
91.2 Ортогональность 141

91.2.1 Что такое ортогональность? . . . 142
91.2.2 Неортогональная система . . . 142
91.2.3 Преимущества ортогональности . . . 143
91.2.4 Увеличение производительности . . . 144
91.2.5 Снижение риска . . . 144
91.2.6 Проектные группы . . . 145
91.2.7 Проектирование . . . 146
91.2.8 Инструментарии и библиотеки . . . 147
91.2.9 Написание текста программы . . . 149
91.2.10 Тестирование . . . 150
91.2.11 Документация . . . 151
91.2.12 Жизнь в условиях ортогональности . . . 151

92 Вспомогательные скрипты на языке Python 153
92.1 Установка под ⊞Windows . . . 155
92.2 Запуск . . . 156
92.3 Дополнительные материалы . . . 161

93 Make: управление сборкой проектов 162

94 VCS: системы контроля версий 163
94.1 CVS . . . 163
94.2 Subversion . . . 163
94.3 Git . . . 163

94.3.1 GitHub 163

95 Основы Си и C_+^+ **164**
95.0.2 Установка MinGW (win32) 164
95.1 Особенности C_+^+ в embedded 164

96 LLVM и разработка собственных компиляторов **165**

97 Лексический и синтаксический анализ **166**
97.1 Лексер и лексический анализ, утилита flex 167
97.2 Компилятор Паскаля 170
97.3 Дополнительная литература 170

98 Сборка кросс-компилятора GNU toolchain **171**

VII Микроконтроллеры Cortex-Mx **172**

99 Отладочные платы **173**
99.1 STM32DISCOVERY /Cortex-M3 STM32F103/ 173
99.2 STM32F4DISCOVERY /Cortex-M4 STM32F407/ 173

VIII Периферия 174

IX Встраиваемый emLinux 175

100 azLinux 178
100.1 Пакеты . . . 180
100.2 Загрузка . . . 182
100.3 Фиксация переменных . . . 182
100.4 dirs: Создание дерева каталогов . . . 183
100.5 APP: Приложение . . . 186
100.6 HW: Поддерживаемое железо . . . 188
100.6.1 i386 . . . 189
100.6.2 ARM . . . 190
100.6.3 MIPS . . . 190
100.7 CPU: Конфигурации процессоров . . . 191
100.7.1 i386 . . . 191
100.7.2 ARM . . . 192
100.7.3 MIPS . . . 192
100.8 Пакеты . . . 192

101 cross 193

102 BuildRoot 194

103 Особенности OpenWrt 195

104 Библиотека SDL **196**
104.1 Реализация microGUI 196

105 Приложения для X Window **197**

106 Программирование сетевых приложений **198**

107 Сборка кросс-компиляторя GNU мальтийским крестом **199**

X IDE **200**

108 eclipse **203**
108.1 Установка ECLIPSE под Windows 205
108.2 Установка ECLIPSE под Linux 205
108.3 Установка CDT 206
108.4 Установка PyDev 207
108.5 Установка TeXlipse 207
108.6 Редактирование файлов в формате XML и производных 208
108.7 Проверка орфографии 208

109 Code::Blocks **211**

110 (g)Vim **212**
110.1 Установка под Windows 214
110.2 Выход из (g)Vim 217

110.2.1 Выход с автосохранением . . . 217
110.3 Переход в режим редактирования . . . 217
110.4 Переход в режим команд . . . 217
110.5 Запись редактируемого файла . . . 218
110.6 Перезагрузка файла . . . 218
110.7 Отмена последних изменений (undo) . . . 218

XI Замечания для авторов 219
110.8 Набор репозиториев на GitHub . . . 220
110.9 Верстка в LaTeX . . . 220

XII Подготовка публикаций в LaTeX 222
110.10 Установка MikTeX под ⊞Windows . . . 225
110.11 Структура документа . . . 225
110.11.1 Заголовочный файл или блок . . . 225
110.11.2 Стили документа . . . 225
110.11.3 Пакеты . . . 225
110.11.4 Автор и название . . . 225
110.11.5 Верстка титульных страниц . . . 225
110.11.6 Оглавление . . . 225
110.12 Верстка слайдов . . . 225
110.13 Список литературы и цитирование . . . 225
110.14 Команды секционирования: часть, глава, раздел,.. . . . 228

110.15 Таблицы . . . 228
110.16 Формулы . . . 228
110.17 Перекрестные ссылки и гипессылки . . . 228
110.18 Листинги скриптов и текстовых данных . . . 228
110.19 Подготовка иллюстраций . . . 228
110.19.1 Графики GNUPLOT . . . 229
110.19.2 Схемы и графы в GraphViz . . . 229
110.19.3 PGF/TikZ . . . 230
110.19.4 GLE . . . 230
110.19.5 Xy-pic . . . 231
110.20 Верстка электронных изданий . . . 231

XIII Символьная и численная математика 232

111 Общие сведения о компьютерной математике 234

112 Пакет Maxima 237
112.1 Установка Maxima под ⊞Windows . . . 237
112.2 Калькулятор . . . 238

XIV Куча 239

Список литературы 240

Введение

Первоначально этот материал задумывался как комплект документации к платам BlackSwift и VoCore, но постепенно превратился в толстенный учебник для студентов ВУЗов и научных работнков по специализациям, связанным с применением цифровой электроники и компьютерной техники.

Большой упор был сделан на использование открытого некоммерческого программного обеспечения, с целью удешевления учебного процесса, уменьшения себестоимости ваших проектов[1], и стимулирования вашего участия в развитии этих программных пакетов.

Лицензия на эту книгу пока не выбрана, так что она пока просто пишется в духе OpenSource: любой может использовать ее часть, изменять или дополнять, до тех пор, пока не накладываются какие-либо административные, финансовые или юридические ограничения на распространение и развитие оригинальной версии или ее открытых форков.

Приглашаем всех желающих участвовать в развитии этого учебного пособия на форум ruOpenWrt, нам нужна обратная связь по качеству материала, результаты тестирования на вас или ваших студентах, дополнения и замечания.

Мы признательны Bill Collis за разрешение использовать материалы его книги «An Introduction to Practical Electronics, Microcontrollers and Software Design»[**bcollis**] в русскоязычном варианте «Азбуки» (Часть I), и конечно он вполне заслуженно включен в основные соавторы этой книги.

[1] вряд ли ли у вас окажется лишняя пачка килобаксов на покупку пары коммерческих САПР, по крайней мере пока ваш стартап не взлетит в Top$100K

Часть I

Введение в практическую электронику

Эта часть основана на книге:

An Introduction to Practical Electronics, Microcontrollers and Software Design
Second Edition, 01 May-2014
© Bill Collis
www.techideas.co.nz

Мы признательны автору за разрешение использовать материалы его книги в русскоязычном варианте «Азбуки», и конечно он вполне заслуженно включен в основные соавторы этой книги.

We are grateful to the author for permission to use materials of his book in the russian version of «Azbuka», and of course he was deservedly included in the main co-authors of this book.

```
From: Bill Collis <Bill.Collis@..........nz>
Date: 2014-11-24 0:53 GMT+04:00
Subject: Electronis Book
To: "dponyatov@gmail.com" <dponyatov@gmail.com>

Hi Dmitry
thanks for your email.
I am looking at the future of the book myself and thinking I will open source
it. If you will only be in using it in Russian language then that is ok and you
need to reference the original book.

Thanks
Bill
```

Глава 1

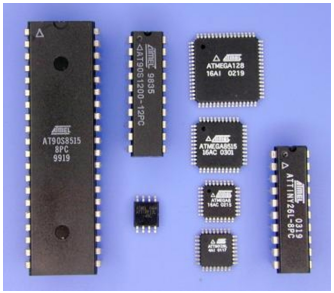
1 Введение в практическую электронику 13

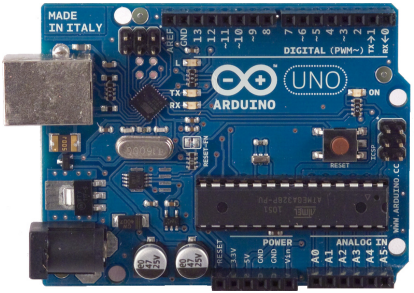
Эта книга ©[1] имеет слеующий ряд основных направлений:

- Распознавание электронных компонентов и их правильное использование
- Наработка цельного набора компетенций по основам электроники
- Использование макетных плат
- Навыки ручной пайки

[1] оригинал: [**bcollis**] B.Collis The Introduction to Practical Electronics...

- Использование закона Ома для выбора токоограничивающих резисторов
- Делитель напряжения
- Использование EDA CAD[2] для разработки и подготовки производства печатных плат
- Программирование микроконтроллеров и их сопряжение с внешними устройствами
- Использование транзистора в режиме ключа
- Теория источников питания
- Принципы и схемы электропривода
- Навыки отладки схем, их тестирования и испытаний
- Следование принципам обучения через практику
- Безопасные приемы работы





[2] [E]lectronic [D]esign [A]utomation, САПР автоматизации проектирования электроники

1.1 Ваше обучение по специальности «Технология»

1.1.1 Цели обучения технологиям Ново-Зеландской программы

- **Технологическая практика**
  - **Чоткость**: разработка ясных описаний для ваших технологических проектов.
  - **Планирование**: думать прежде чем делать, и использовать во время работы документацию: блок-схемы, принципиальные схемы, чертежи разводки плат, диаграммы и эскизы.
  - **Наработка навыков**: сборка, отладка и тестирование электронных схем, проектирование и изготовление печатных плат, написание программ для микроконтроллеров.
- **Технологические знания**
  - **Моделирование**: прежде чем строить готовое электронное устройство, сначала важно понять как оно работает путем моделирования и/или макетирования аппаратного и программного обеспечения.
  - **Технологические продукты**: знания о компонентах и их характеристиках.
  - **Технологические системы**: электронное устройство является более, чем набором компонентов, это функционирующая система с входами, выходами и контролирующим процессом.
- **Природа технологии**
  - **Значение технологических достижений**: знания об электронных компонентах, особенно микроконтроллерах, как основе современных технологий.

– **Роль технологии в обществе**: электронные устройства в настоящее время играют центральную роль в инфраструктуре нашего современного общества; подчинили ли они нас себе, как они изменили нашу жизнь?

1.2 Ключевые компетенции Ново-Зеландской программы

- **Знания**: для меня предметом технологии является все что относится к знанию. Моя цель: заставить студентов понимать технологии, заложенные в электронные устройства. Для достижения этого понимания студенты должны активно учиться[3] в работе на самом раннем этапе, чтобы они могли построить собственное понимание предмета и пойти дальше, чтобы стать хорошими решалами проблем. В начале обучения электронике это требует от студентов восприимчивости к инструкциям, которые им дают, и поиск ясности, когда они не понимают их.

  Для этого на занятиях рассматриваются много новых и различных элементов знаний, и студентам выдаются задания на решение проблем, чтобы помочь им мыслить логически. Копирование чужого ответа наказывается, но приветствуется совместная работа. В основе обучения лежит построение правильных концептуальных моделей и анализ в контексте "большой картины".

- **Взаимодействие**: работа в парах и группах, это важно как в классе, так и в любой другой ситуации в жизни; мы все должны договариваться и разделять ресурсы и оборудование с другими людьми; поэтому крайне важно активное общение и помощь друг другу.

- **Использование языка символов и текстов**: сердцем нашего предмета является язык, который мы используем для обмена информацией в электронных схемах, планах, алгоритмах и синтаксисе

[3] в оригинале **enage**, англо-калька с себуанского, NZ

компьютерных языков программирования; так что способность распознавать и правильно использовать символы и диаграммы для работы, которую мы делаем, имеет критическое значение.

- **Самоконтроль**: студенты принимают на себя личную ответственность за собственное обучение; они принимают вызов, надеясь найти ответы в книгах или найти учителя, способного объяснить им, что делать. Это значит, что студенты должны взаимодействовать с рабочим материалом.

  Иногда ответы приходят легко, иногда нет; часто наша тема требует много проб и ошибок (в основном ошибок). Студенты должны знать, что у них будут трудные времена, пока не будет изучена большая часть. И не сдаться в поиске понимания.

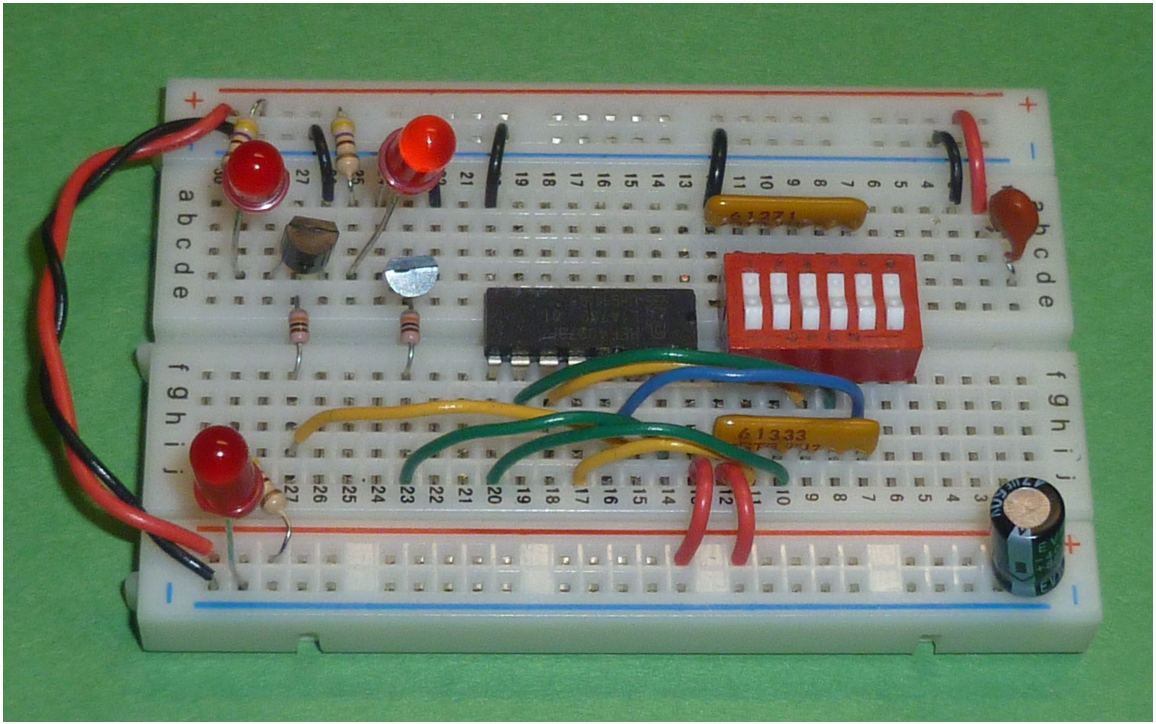
- **Участие и содействие**: мы живем в мире, который невероятно зависит от технологии, особенно электроники; студенты должны развивать осознание важности этой области человеческого творчества в нашей повседневной жизни, и понимать, что наши проекты имеют и социальную функцию, а не только техническую.

Глава 2

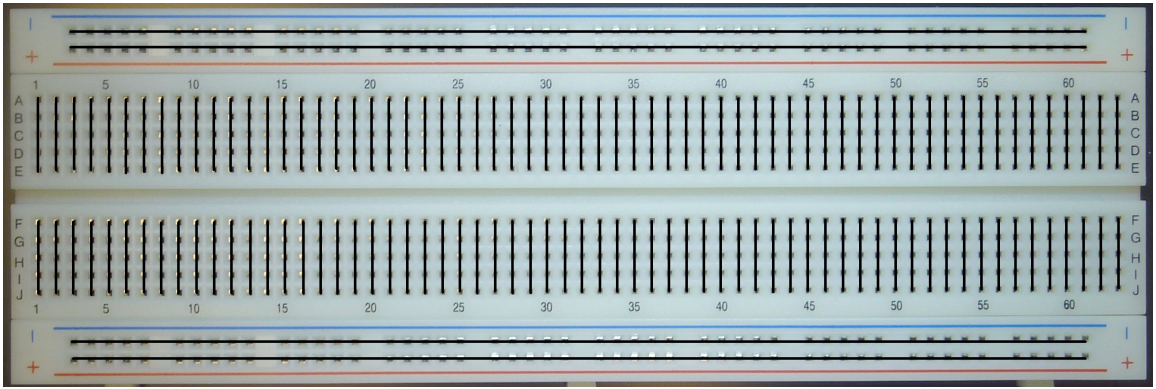
2 Вводная электронная схема 15

2.1 2.1 Где купить комплектующие? 15

В Новой Зеландии есть некоторое количество отличных поставщиков компонентов с разумными ценами, включающих www.surplustronics.co.nz, и www.activecomponents.com. Зарубежные поставщики, которых я использую, включают www.digikey.co.nz, www.sparkfun.com, ebay.com и aliexpress.com



Макетная плата (breadboard) — пластмассовый блок с отверстиями и металлическими полосковыми металлическими зажимами, создающими соединения между элементами схемы. Отверстия расположены так, что компоненты могут быть соединены вместе формируя схему. Верхние и нижние ряды, как правило, используется для шин питания, красный сверху для плюса, и внизу черный/синий для для минуса (общий провод).



Эта схема 2.1 может быть собрана вот так 2.1, обратите внимание, что светодиод должен находиться в правильном положении. Если у вас есть светодиод и резистор, соединенные в замкнутый контур, светодиод должен загореться.

Принципиальная схема

Компоновка

The LED requires 2V the battery is 9V, if you put the LED across the battery it would stop working! So a 1k (1000ohm) resistor is used to reduce the voltage to the LED and the current through it, get a multimeter and measure the voltage across the resistor, is it close to 7V? If you disconnect any wire within the circuit it stops working, a circuit needs to be complete before electrons can flow.

2.2 2.2 Определение сопротивления резистора по цветовому коду 16

.

2.3 2.3 Светодиоды 17

.

2.4 2,4 Некоторые технические характеристики светодиода 17

.

2.5 2.5 Задание на исследование светодиода 17

.

2.6 2.6 Добавление выключателя в схему 18

.

2.7 2.7 Задание на установку выключателя 18

.

2.8 2,8 Важные понятия схемы 19

.

2.9 2.9 Изменение величины сопротивления 19

.

2.10 2.10 Добавление транзистора в схему 20

.

2.11 2.11 Чтение схем 21

.

2.12 2.12 Входная цепь — LDR 22

.

2.13 2.13 Рабочая схема датчика темноты 23

.

2.14 2,14 Защитные цепи - использование диода 24

.

2.15 2.15 Задача исследования диода 24

.

2.16 2.13 Финальная схема датчика темноты 23

.

Глава 3

3 Вводное конструирование печатной платы 26

Глава 4

4 Пайка, припой и паяльники 41

Глава 5

5 Введение в теорию электроники 49

Глава 6

6 Введение в электронику микроконтроллера 63

Глава 7

7 Входные цепи микроконтроллера 91

Глава 8

8 Обзор программирования 104

Глава 9

9 Введение в поток выполнения программы 112

Глава 10

10 Вводное программирование: использование подпрограмм 126

Глава 11

11 Вводное программирование: Использование переменных 134

Глава 12

12 Основные дисплеи 161

Глава 13

13 Проект портативного аудиоусилителя на TDA2822M 174

Глава 14

14 Основы логического программирования 187

Глава 15

15 Разработка алгоритма: Система сигнализации 202

Глава 16

16 Основы теории цепей постоянного тока 215

Глава 17

17 Основы планирования проекта 236

Глава 18

18 Пример дизайна системы: Таймер клеевого пистолета 268

Глава 19

19 Основные интерфейсы и их программирование 273

Глава 20

20 Основы интерфейса аналого-цифрового преобразования 295

Глава 21

21 Основы проектирования системы 314

Глава 22

22 Основы проектирования системы: Тайм-трекер 317

Глава 23

23 Основы вычислений времени 330

Глава 24

24 Основы строковых переменных 340

Глава 25

25 Силовые интерфейсы 353

Глава 26

26 Теория источников питания 370

Глава 27

27 Типичные вопросы тестирования 2011/12/13 годов 395

Глава 28

28 Расширенное программирование: Массивы 397

Глава 29

29 Подтягивающие резисторы AVR 402

Глава 30

30 Дополнительноподключение клавиатуры 403

Глава 31

31 Тонкости циклов Do-Loop & While-Wend 417

Глава 32

32 Подключение двигателя постоянного тока 423

Глава 33

33 Пример расширенной системы: Будильник 452

Глава 34

34 Резистивный сенсорный экран 468

Глава 35

35 Пример проектирования системы: Регулятор температуры 475

Глава 36

36 Расширенное программирование: Машины состояний 478

Глава 37

37 Переработанный проект будильника 501

Глава 38

38 Студенческий проект: Расширенный оконный контроллер 514

Глава 39

39 Альтернативные техники кодирования машин состояния 524

Глава 40

40 Сложно: последовательная связь 526

Глава 41

41 Цифровой радиоканал 597

Глава 42

42 Введение в I2C 617

Глава 43

43 Студенческий проект: Таймер полива теплицы 631

Глава 44

44 Проект Велосипедного аудиоусилителя 642

Глава 45

45 Графические LCD 648

Глава 46

46 Проект Отслеживания температуры GLCD 660

Глава 47

47 Прерывания 672

Глава 48

48 Таймеры/Счётчики 692

Глава 49

49 Проект скроллинга графического LED дисплея: массивы и таймеры 698

Глава 50

50 Проект медицинского прибора: реализация таймера 709

Глава 51

Проект часов на 7-сегментном индикаторе реализация на сдвоенном таймере

Глава 52

52 ИС драйвера дисплея MAX 7219/7221 739

Глава 53

53 Подключение через мобильную связь: ADH8066 744

Глава 54

54 Передача данных через Internet 778

Глава 55

55 Задание: математика в реальном мире 816

Глава 56

56 Цветной графический LCD на основе SSD1928 825

Глава 57

57 Светофор: помощь и решение 865

Глава 58

58 Компьютерное программирование: низкоуровневые детали 869

Глава 59

59 USB-программатор: USBASP 876

Глава 60

60 Программатор USBTinyISP 877

Глава 61

61 Программирование на Си и AVR 881

Глава 62

62 Объектно-Ориентированное Программирование (ООП) на C_{+}^{+} и AVR 929

Глава 63

63 Современные (2014) отладочные платы на AVR 953

Глава 64

64 Eagle: создание собственной библиотеки 970

Глава 65

65 Практические методы 979

Глава 66

66 ЧПУ 990

Глава 67

67 Индекс 1008

Часть II

Основы электроники

Здесь идет список ссылок на онлайн лекции в edX, Coursera, и т.п.

Глава 68

Линейные схемы на пассивных элементах, основы электротехники

Глава 69

Симуляция и расчет схем в ngSPICE

Глава 70

KiCAD

70.1 Отрисовка схем в KiCAD

70.2 Библиотеки элементов

70.3 Передача схемы в ngSPICE

Глава 71

Простейшие полупроводниковые элементы

71.1 Оптоэлектроника

71.2 Схемы на биполярных транзисорах

71.3 Схемы на на полевых транзисорах

Глава 72

Операционные усилители

Глава 73

Источники питания

73.1 Батарейное питание

73.2 Линейные стабилизаторы

73.3 Импульсные преобразователи на ШИМ-контроллерах

73.4 Цепи защиты и гашения кондуктивных помех

Глава 74

Цифровая электроника

Глава 75

Компьютерные интерфейсы

75.1 Поколение 90x: COM, LPT, ISA

75.1.1 Резервный программатор AVR “пять проводков”

75.2 Сеть CAN

75.3 Интерфейсные модули USB

75.3.1 Универсальный высокоскоростной конвертер FTDI FT2232H

75.3.2 JTAG-адаптер

75.3.3 Отладочный модуль CAN

Глава 76

ПЛИС

Глава 77

Датчики

Глава 78

Электропривод и исполнительные устройства

Часть III

Основы конструирования РЭС

Глава 79

Пакеты моделирования на основе OpenFOAM

Глава 80

Обеспечение теплового режима

Глава 81

Электромагнитная совместимость

81.1 Кондуктивные помехи

81.2 Компоновочные модели и оптимизация кабельной сети

Часть IV

Технология РЭС

Глава 82

Инструменты и оборудование

82.1 Паяльное оборудование

82.1.1 Паяльник

Паяльник — обязателен дешевый сетевой мощностью не менее 20 Вт, типа ЭПСН-25/220. Ограничитель мощности или регулятор температуры легко собрать самостоятельно.

Для сборки электроники хорошо также иметь маленький монтажный 12 В 8 Вт от паяльной станции ZD-927 (∼100 р), без самой станции. Если не жалко 500 р, берите станцию ZD-927 целиком, внутри простейший регулятор мощности, и вам не понадобится источник питания на 12 В, который вы еще не сделали.

Паяльник ЭПСН-25/220

Паяльник 220В 25Вт, СВЕТОЗАР, SV-55310-25 230 р.

Паяльник 220В 25Вт ZD-721N 175 р.

Паяльник для станции ZD-927 12 В 8 Вт 85 р.

82.1.2 Паяльная станция

Из всего разнообразия для хоббита оптимальным являются паяльные станции Lukey 702/853D (3000+ р). Для работы или регулярного хобби паяльная станция с феном, а может даже и встроенным источником питания, вещь незаменимая, и не такая уж дорогая.

Паяльная станция ZD-927 520 р.

Паяльная станция LUKEY 702 3100 р.

Паяльная станция LUKEY 853D с источником питания 5200 р.

82.2 JTAG-адаптер

82.3 Отладочные платы

Прежде чем начать работать с отдельными МК, устанавливая их на плату собственной разработки, для быстрого старта используют отладочные платы[1]

82.3.1 Arduino /Atmel Mega AVR8/

82.3.2 Cortex-Mx

82.3.3 CubieBoard /Cortex-A8 AllWinner A10/

82.3.4 Raspberry Pi /ARM11 BCM3032/

82.3.5 BlackSwift /MIPS/

82.3.6 VoCore /MIPS/

82.4 Радиомонтажный инструмент

Пара надфилей, заточной камень на дрель, комплект сверел и несколько листов наждачки.

[1] development board, demo board

82.4.1 Pro'sKit

Отдельного обзора заслуживает инструмент и наборы Pro'sKit

PK-5308BM универсальный набор инструментов

1PK-616B Набор инструментов для электроники профессиональный

1PK-813B Набор базовых инструментов для электроники

По личному опыту: в 1PK-813B не хватает

- мелкого мультиметра,
- стриппера 1PK-3001E,
- микрокусачек типа 8PK-30D,
- канифоли,
- ножа,
- настроечную отвертку заменить индикаторной.

Инструмент до 1000 В

Для электромонтажных работ обязательно приобретите комплект высоковольтного инструмента до 1000 В:

РМ-911 Пассатижи 1 кВ

РМ-917 Кусачки (бокорезы) 1 кВ

Хранение

103-132D Кассетница для деталей и компонентов

SB-3428SB Портативная кассетница для саморезов и т.п.

Радиомонтаж

8PK-30D Кусачки миниатюрные

1PK-709 Длинногубцы-кусачки

1PK-055S Длинногубцы изогнутые

1PK-29 Круглогубцы

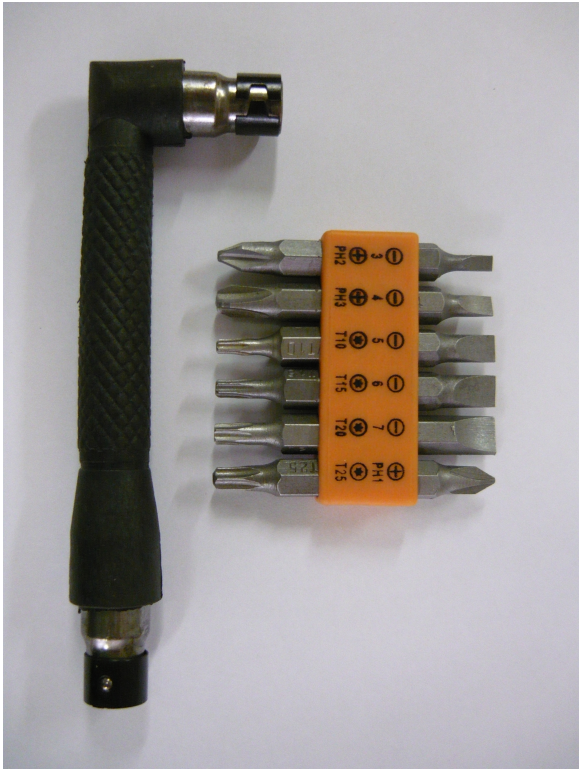
1PK-101T Пинцет прямой

1PK-3001E Клещи для зачистки проводов прецизионные (стриппер)

PD-374 Тиски на струбцине

Прочие

Попалась интересная недорогая отвертка: аиксация четкая, исполнение очень неплохое, позволяет добраться до узких мест. Из минусов: ручка похоже не цельнометаллическая, при изломе есть риск распороть руку.



82.5 Измерительное оборудование

82.5.1 Тестер

82.5.2 Осциллограф

82.5.3 Логический анализатор

82.5.4 Генератор сигналов

82.5.5 Рыльцеметр

82.6 Электроинструмент

82.6.1 Дрель

Дрель ударная сетевая Praktyl-R PID13D01 400 Вт (!)395 р.

Дрель безударная сетевая Интерскол Д-11/530ЭР (с БЗП) 1120 р.

Дрель — одноразовая китайчатина от 400 р. Продаются уже брендированные на Леруа Мерлен, наклейка «PID13D01 Ударная дрель 400 Вт, 13 мм». Скорость регулируется глубиной нажатия курка, крутилка

на курке ограничивает глубину механически, фиксатор держит скорость близко к минимальной, запаха горелой пластмассы через несколько минут работы на холостом ходу нет.

По надежности рекомедуется Интерскол 1100+ р. Надежность Интерскола — не «китай», классика ДУ-580ЭР работает в хвост и гриву, используется криворукими студентами, лежит в подвале в пыли от точила, и никаких вопросов даже со щетками.

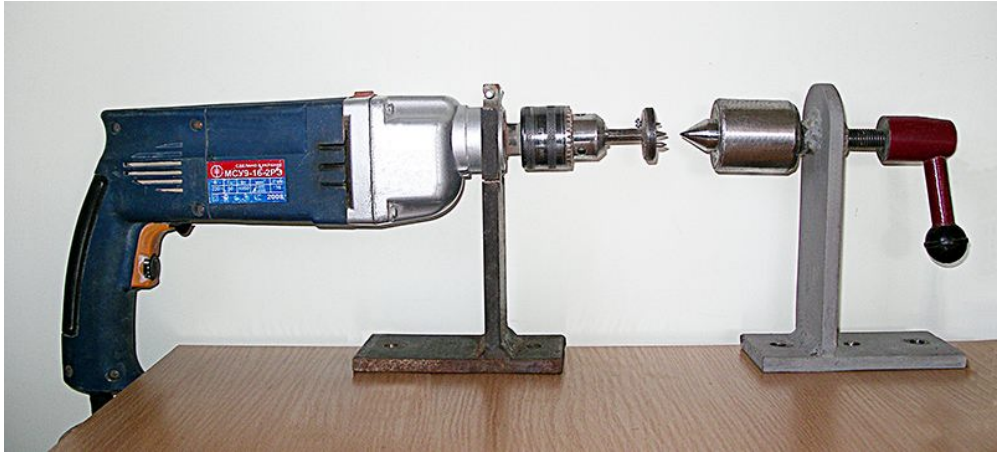
Если не планируете много сверлить бетон, **берите дрель без ударного механизма**: отсутствуют лишние продольные перемещения, что может быть важно при использовании в качестве шпинделя сверлильного станка, и механизации других технологических поделок.

У шуруповерта нет 43 мм шейки для фиксации, поэтому как средство электропривода он практически бесполезен, и нужен собственно для заворачивания большого количества саморезов. Хотя наличие ограничителя крутящего момента и малые габариты удобны при сверлении и сборке поделок.

Имея некоторое количество поделочного материала, кривые руки и особенно доступ к станочному оборудованию, можно сколкозить некоторое подобие настольных станочков /стр.121/ для механизации некоторых работ, используя дрель в качестве привода.

Главным элементом такой оснастки — зажим на шейку дрели 43 мм. Особых требований по его точности и качеству нет, т.к. сама шейка обычно пластиковая, и никакой доводки по круглости и параллельности оси инструмента не проходит.

makita


МСУЭ-16-2Р

82.6.2 Лобзик

Praktyl 350 Вт 356 р.

Makita 4329 2260 р.

Лобзик полезен при разделке стеклотекстолита, и изготовлении технологической мебели (стеллажи, рабочие столы и т.п.).

82.6.3 Ж двигатель

Если у вас возникло желание механизировать изготовление механических деталей, а свободного доступа к настоящему станочному оборудованию нет, есть смысл рассмотреть изготовление самодельной механизированной оснастки типа /стр.121/, или даже самодельных станочков. В этом случае надо рассмотреть применения универсального привода.

Первый кандидат на место универсального электропривода достается той самой дрели, не забываем об обязательном наличии 43 мм монтажной шейки. Достоинство дрели как привода — прямое подключение к сети, встроенный редуктор, есть модели с простой регулировкой оборотов, есть резьба и отверстие под винт на валу, в комплекте есть патрон для зажима мелких деталей в точилке[2].

Ограниченно доставаемые двигатели от стиральных машин, отличаются мощностю и оборотистостью, особенно от старых моделей. Часто доступны сразу с готовым шкивом на валу, который иногда проще использовать, чем снять.

Автозапчасти: привод печки Камаза, двигатель постоянного тока 24 В 50 Вт

Новые асинхронные двигатели АИРЕ 56 В2/В4 (3000/1500 об.) с заводским конденсатором, подключается к сети ∼220 В, цена от 2500 р. С ростом размеров и мощности цена резко повышается. Следует обратить внимание на возможность монтажа на дополнительный фланцевый подшипниковый щит, (?) с моделями АИРЕ 80.

Для самодельных серлилок и микроинструмента хороши китайские воздушные шпиндели постоянного тока с цанговыми патронами ER11. Требуют источник питания постоянного тока 9÷48 В. В магазинах не попадались, необходима прямая покупка с AliExpress[3] по почте.

---

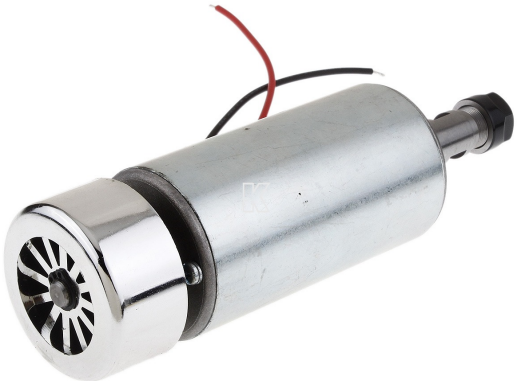
[2] БЗП удобен, патрон с ключем дает лучший зажим и возможно точнее

[3] пользуйтесь английской версией — переводная жуткое УГ

Жвигатель Вятка-Автомат 19?? г.

Двигатель печки Камаза

АИРЕ 56 В2, 0.2 КВт



Воздушный шпиндель с цангой ER11

Съемные фрезерные шпиндели, поставляются отдельно или в комплекте с насадкой ручного фрезера по дереву. Лучшие, со стальной шейкой — Kress, активно применяются хобби-ЧПУшниками. Попроще и сильно дешевле делал Интерскол, иногда попадается noname. Недостаток как универсального привода — они высокоскоростные, возникают проблемы с понижающими передачами. Применение — приводной высокоскоростной инструмент: боры, фрезы по дереву, микроинструмент для граверов (микродиски, шарошки). Цанга 8 мм. Для некоторых моделей бывают наборы цанг на мелкий инструмент.

KRESS 530/800/1050 FM(E)
5600+ р.

Интерскол ФМ-30/750
/снят с производства/

Интерскол ФМ-55/1000 Э
5050 р.

Глава 83

Трассировка плат и подготовка производства в KiCAD

83.1 Технология ЛУТ (Лазерный УТюг)

83.2 Технология фоторезиста

83.3 Формат Gerber и подготовка промышленного производства

Глава 84

FreeCAD

[1] В среде специалистов ряда отраслей известна проблема создания полноценной САПР в рамках OpenSource, и хотя FreeCAD ещё не является кандидатом на такую «полноценность», этот продукт может рассматриваться как одна из попыток создания базы для решения этой проблемы. Разработчик FreeCAD Юрген Ригель, работающий в корпорации DaimlerChrysler, позиционирует свою программу как первый бесплатный инструмент проектирования механики (сравнивая свой продукт с такими развитыми проприетарными системами как CATIA версий 4 и 5, SolidWorks), созданный на основе библиотеки **Open CASCADE**. Цель программы — предоставить базовый инструментарий этой библиотеки в интерактивном режиме.

Следует отметить, что имеет место ещё один программный продукт имеющий название freeCAD, его разработчик — Aik-Siong Koh, и он не связан с FreeCAD'ом Юргена Ригеля.

[2] FreeCAD — CAD/CAE приложение трёхмерного параметрического моделирования. Оно в основном сделано для механического проектирования, но также может быть использовано для любых других случаев, в которых вам нужно точно моделировать трёхмерные объекты с контролем над историей моделирования.

FreeCAD все еще находится в ранней стадии разработки, так что, хотя он уже предлагает Вам большой (и растущий) список функций, многого еще не хватает, особенно если сравнивать его с коммерческими решениями, и вы можете не найти его достаточно развитым для использования в производственной среде. Тем не менее, есть быстрорастущее сообщество пользователей-энтузиастов, и вы уже можете найти много примеров качественных проектов, разработанных с FreeCAD.

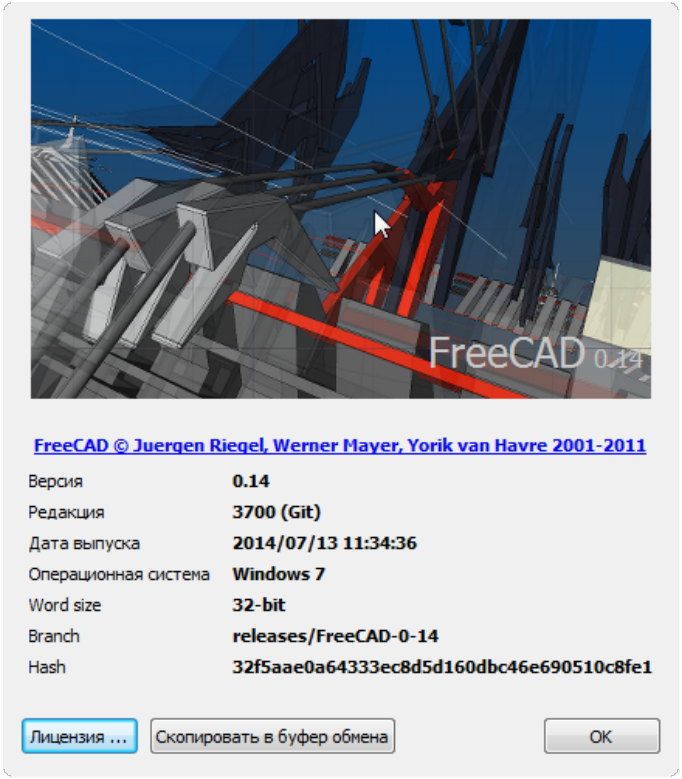
Как и все проекты с открытым исходным кодом, проект FreeCAD не единственый способ работы обеспеченный Вам его разработчиками. Это во многом зависит от роста его сообществу пользователей и разработчиком, доработки функций и стабилизации кода (да здравствует исправление ошибок!). Так

[1] копипаста https://ru.wikipedia.org/wiki/FreeCAD_(Juergen_Riegel%27s)
[2] копипаста http://www.freecadweb.org/wiki/index.php?title=Getting_started

что не забывайте об этом, когда начинаете использовать FreeCAD, если вам он нравится, вы можете непосредственно влиять и помочь проекту!

84.1 Установка под ⊞Windows

⊞ + R » http://www.freecadweb.org/ » Download » ⊞Windows » **FreeCAD 0.14** » **. . . _setup.exe**
FreeCAD 0.14.3700_x86_setup » FreeCAD 0.14 License » I agree
Distination Folder » **C:/FreeCAD** » Install » Completed » Close
⊞ » Программы » FreeCAD 0.14 » FreeCAD » ▷ » Закрепить в панели задач


FreeCAD 0.14
FreeCAD © Juergen Riegel, Werner Mayer, Yorik van Havre 2001-2011
Версия 0.14
Редакция 3700 (Git)
Дата выпуска 2014/07/13 11:34:36
Операционная система Windows 7
Word size 32-bit
Branch releases/FreeCAD-0-14
Hash 32f5aae0a64333ec8d5d160dbc46e690510c8fe1
Лицензия ...
Скопировать в буфер обмена
OK

84.2 Чертеж

84.3 Эскиз

84.4 Деталь

84.5 Сборка

84.6 Автогенерация конструкторской докуметации

84.7 Скрипты и пользовательские расширения

Глава 85

Эксплуатация станочного оборудования

Глава 86

Основы ЧПУ и цифрового производства

86.1 CAM-пакеты для FreeCAD

Часть V

Основы теории систем автоматического управления

Глава 87

Математический аппарат

87.1 Передаточная функция

87.2 Устойчивость САУ

87.3 Сети Петри

87.4 Автоматы Маркова

Глава 88

Релейное управление

Глава 89

Пропорциональные САУ

Глава 90

ПИДn-регуляторы

Часть VI

Разработка ПО для встраиваемых систем

Глава 91

Архитектура программных систем

91.1 Литература

[**proghant**] Программист-прагматик. Путь от подмастерья к мастеру Хант Э., Томас Д., Лори/Питер, 2004, 2007

91.2 Ортогональность

[1][**proghant**]

[1] копипаста http://ru.wikibooks.org/wiki/%D0%9E%D1%80%D1%82%D0%BE%D0%B3%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C

Ортогональность очень важна, если вы хотите создавать системы, которые легко поддаются проектированию, сборке, тестированию и расширению. Однако этому принципу редко обучают непосредственно. Часто он является лишь скрытым достоинством других разнообразных методик, которые вы изучаете. Это неправильно. Как только вы научитесь непосредственно применять принципы ортогональности, вы сразу заметите, как улучшилось качество создаваемых вами систем.

91.2.1 Что такое ортогональность?

Термин "ортогональность" заимствован из геометрии. Две линии являются ортогональными, если они пересекаются под прямым углом, например, оси координат на графике. В терминах векторной алгебры две такие линии перемещения являются независимыми. Если двигаться параллельно оси X вдоль одной из линий, то проекция движущейся точки на другую линию не меняется. Этот термин был введен в информатике для обозначения некой разновидности независимости или несвязанности. В грамотно спроектированной системе программа базы данных будет ортогональной к интерфейсу пользователя: вы можете менять интерфейс пользователя без воздействия на базу данных и менять местами базы данных, не меняя интерфейса. Перед тем как рассмотреть преимущества ортогональных систем, познакомимся с неортогональной системой.

91.2.2 Неортогональная система

Предположим, вы находитесь в экскурсионном вертолете, совершающем полет над Гранд-Каньоном, когда пилот, который совершил ошибку, наевшись рыбы за обедом внезапно вскрикивает и теряет сознание. По счастливой случайности это происходит, когда вы парите на высоте 30 метров. Вы догадываетесь, что рычаг управления общим шагом несущего винта обеспечивает подъем машины, так что, если его слегка опустить, вертолет начнет плавно снижаться. Однако когда вы пытаетесь сделать это, то осознае-

те, что жизнь — не такая уж простая штука. Вертолет клюет носом, и вас начинает вращать по спирали влево.

Внезапно вы понимаете, что управляете системой, в которой каждое воздействие имеет побочные эффекты. При нажатии на левый рычаг вам придется сделать уравновешивающее движение назад правым рычагом и нажать на правую педаль. Но при этом каждое из этих действий вновь повлияет на все органы управления. Неожиданно вам приходится жонглировать невероятно сложной системой, в которой любое изменение влияет на все остальные управляющие воздействия. Вы испытываете феноменальную нагрузку: ваши руки и ноги находятся в постоянном движении, пытаясь уравновесить все взаимодействующие силы. Органы управления вертолетом определенно не являются ортогональными.

91.2.3 Преимущества ортогональности

Как показывает пример с вертолетом, неортогональные системы сложнее изменять и контролировать. Если составляющие системы отличаются высокой степенью взаимозависимости, то невозможно устранить какую-либо неисправность лишь на локальном уровне.

Исключайте взаимодействие между объектами, не относящимися друг к другу

Мы хотим спроектировать компоненты, которые являются самодостаточным независимыми, с единственным, четким назначением. Когда компоненты изолированы друг от друга, вы уверены, что можно изменить один из них, не заботясь об остальных. Пока внешние интерфейсы этого компонента остаются неизменными можете быть спокойны, что не создадите проблем, которые распространятся по всей системе. С созданием ортогональных систем у вас появятся два больших преимущества: увеличение производительности и снижение риска.

91.2.4 Увеличение производительности

- Изменения в системе локализуются, поэтому периоды разработки и тестирования сократятся. Легче написать относительно небольшие, самодостаточные компоненты, чем один большой программный модуль. Простые компоненты могут быть спроектированы, запрограммированы, протестированы и затем забыты — не нужно непрерывно менять существующий текст по мере того, как к нему добавляются новые фрагменты.

- Ортогональный подход также способствует многократному использованию компонентов. Если компоненты имеют определенную, четкую сферу ответственности, они могут комбинироваться с новыми компонентами способами, которые не предполагались при их первоначальной реализации. Чем меньше связанность в системах, тем легче их перенастроить и провести их обратное проектирование.

- При комбинировании ортогональных компонентов происходит заметное увеличение производительности. Предположим, что один компонент способен осуществлять M, а второй — N различных операций. Если эти компоненты ортогональны и комбинируются, то в сумме они способны осуществить $M \times N$ различных операций. Но если два компонента не являются ортогональными, они будут перекрываться, и результат их действия будет меньшим по сравнении с ортогональными компонентами. Вы получаете большее количество функциональных возможностей в пересчете на единичное усилие, если комбинирует между собой ортогональные компоненты.

91.2.5 Снижение риска

Ортогональный подход приводит к снижению уровня риска, присущего любой разработке.

- Ошибочные фрагменты текста программы изолируются. Если модуль содержит ошибку, то вероятность ее распространения на всю систему уменьшается. Кроме того, ошибочный фрагмент может быть извлечен и заменен новым (исправленным).

- Конечный продукт (система) становится менее хрупким. Проблемы, появляющиеся при внесении небольших изменений и устранении недочетов на определенном участке, не проходят дальше этого участка.

- Ортогональная система способствует повышению качества тестирования, поскольку облегчается проектирование и тестирование отдельных ее компонентов.

- Вы не будете слишком сильно привязаны к определенному субподрядчику, программному продукту или платформе, поскольку интерфейсы между компонентами, производимыми фирмами-субподрядчиками, не будут играть главенствующей роли в проекте.

91.2.6 Проектные группы

Приходилось ли вам замечать, насколько эффективно работают проектные команды, все члены которых знают, что делать, и полностью отдают себя делу, тогда как в других командах сотрудники постоянно препираются между собой и не собираются ни в чем уступать друг другу? Зачастую это не что иное, как проблема ортогональности. Если команды организованы с большим числом перекрытий, то сотрудники путают свои должностные обязанности. Для любого изменения необходимо собирать всю команду, поскольку оно, может быть, затронет каждого. Как разбить команду на группы с четкими обязанностями и минимальным перекрытием? На этот вопрос нет простого ответа. В некоторой степени это зависит от проекта и вашего анализа областей, которые в перспективе могут измениться. Это же зависит от людей, находящихся в вашем распоряжении.

Мы предпочитаем отделять инфраструктуру от приложения. Каждому из основных инфраструктурных компонентов[2] приписывается только ему принадлежащая группа. Подобным образом производится разделение функциональных возможностей приложения. После этого мы изучаем людей, которые имеются в нашем распоряжении на данный момент[3], и сообразно этому корректируем состав групп. Вы можете неформально определить уровень ортогональности структуры проектной команды. Для этого просто посмотрите, скольких людей нужно привлечь к обсуждению каждого изменения, требуемого со стороны. Чем больше эта цифра, тем ниже уровень ортогональности группы. Отсюда ясно, что ортогональная команда работает более эффективно.[4]

91.2.7 Проектирование

Большинство разработчиков знакомо с потребностью в проектировании ортогональных систем, хотя они наверняка используют термины "модульный "компонентно-ориентированный"и "многоуровневый"для описания конкретного процесса. Системы должны быть скомпонованы из набора взаимодействующих модулей каждый из которых реализует функциональные возможности независимо от других. Иногда эти компоненты объединены в уровни, каждый из которых обеспечивает некий уровень абстракции. Данный многоуровневый подход является мощным методом проектирования ортогональных систем. Поскольку на каждом уровне используются только абстракции, обеспеченные на низших уровнях, можно легко изменить основные реализации, не затрагивая самой программы. Иерархическое представление также уменьшает риск появления неконтролируемых зависимостей между модулями. Существует простой тест на ортогональность проектирования. Как только вы составили схему компонентов, спросите себя: "Сколько

[2] база данных, интерфейс связи, промежуточное программное обеспечение и т.д.

[3] или планируем их появление в будущем

[4] Высказав это, мы тем самым поощряем стремление сотрудников более мелких подразделений постоянно общаться друг с другом.

модулей подвергнутся воздействию, если я резко изменю требования по конкретной функции?"В ортогональной системе ответ должен быть "один". Перемещение кнопки на панели графического интерфейса пользователя не должно требовать внесения изменений в схему базы данных. Добавление контекстно-зависимой справки не должно изменить подсистему выставления счетов. Рассмотрим сложную систему контроля и управления отопительной установкой. Первоначально требовалось наличие графического интерфейса, но затем требования были изменены, с тем чтобы добавить систему речевого ответа и управления установкой при помощи телефона с тональным набором. В ортогонально спроектированной системе для этого вам пришлось бы изменить только модули, связанные с интерфейсом пользователя, а основная логика управления предприятием остается неизменной. На самом деле, если вы тщательно структурируете систему, то у вас должна быть возможность поддержки обоих интерфейсов при наличии одной и той же программной базы. Стоит спросить себя, как защитить вашу конструкцию от изменений в окружающем мире, например, вы пользуетесь номером телефона в качестве идентификатора заказчика. Что произойдет, если телефонная станция изменит коды междугородней связи? Не полагайтесь на свойства предметов, которыми не можете управлять.

91.2.8 Инструментарии и библиотеки

Будьте внимательным, чтобы сохранить ортогональность вашей системы при введении инструментариев и библиотек, произведенных фирмами-субподрядчиками. Проявите мудрость при выборе технологии. Однажды авторы работали над проектом, в котором требовалось, чтобы некий фрагмент программы на языке Java выполнялся автономно — на сервере и в удаленном режиме — на клиентской машине. В этом случае возможными вариантами распределения классов были технологии RMI и CORBA. Если удаленный доступ к классу обеспечивался при помощи RMI, то в этом случае каждое обращение к удаленному методу в этом классе могло бы привести к генерации исключения, означающей, что эта наивная реализация потребовала бы от нас обработки этого исключения всякий раз при использовании удален-

ных классов. В данном случае использование RMI явно не ортогонально: программа, обращающаяся к удаленным классам, не должна зависеть от их физического расположения. Альтернативный способ — технология CORBA — не налагает подобного ограничения: мы можем написать программу, для которой не имеет значения, где физически находятся классы. Когда вы используете инструментарий (или даже библиотеку, созданную другими разработчиками), вначале спросите себя, не заставит ли он внести в вашу программу изменения, которых там быть не должно. Если схема долговременного хранения объекта прозрачна, то она ортогональна. Если же при этом требуется создание объектов или обращение к ним каким-либо особым образом, то она неортогональна. Отделение этих подробностей от вашей программы дает дополнительное преимущество, связанное с возможностью смены субподрядчиков в будущем. Интересным примером ортогональности является система Enterprise Java Beans. В большинстве диалоговых систем обработки запросов прикладная программа должна обозначать начало и окончание каждой транзакции. В системе EJB эта информация выражена описательно в виде метаданных вне любых программ. Та же прикладная программа может работать в различных транзакционных средах без каких-либо изменений. Вероятно, это станет прообразом многих операционных сред будущего. Другой интересной проверкой на ортогональность является технология Aspect Oriented Programming — исследовательский проект фирмы Xerox Parc. Технология AOP позволяет выразить в одном-единственном месте линию поведения, которая в противном случае была бы распределена по всему исходному тексту программы. Например, журнальные сообщения обычно генерируются путем явных обращений к некоторой функции записи в журнал по всему исходному тексту. Используя технологию AOP, вы реализуете процедуру записи в журнал ортогонально к записываемым данным. Используя версию AOP для языка Java можно записать сообщение журнала при входе в любой метод класса Fred, запрограммировав аспект:

```
aspect Trace {
advise * Fred.*(..) {
static before {
```

```
Log.write("-> Entering + thisJoinPoint.methodName); }
}
}
```

При вплетении этого аспекта в текст вашей программы будут генерироваться трассировочные сообщения. Если этого не сделать, не будет и сообщений. В обоих случаях исходный текст остается неизменным.

91.2.9 Написание текста программы

Всякий раз, когда вы пишете программу, вы подвергаетесь риску снижения уровня ортогональности вашего приложения. Если вы постоянно не отслеживаете не только то, что вы делаете, но и весь контекст приложения, то существует опасность неумышленного дублирования функциональных возможностей в некотором другом модуле или выражения существующих знаний дважды. Есть ряд методик, которые можно использовать для поддержки ортогональности:

- Сохраните вашу программу "несвязанной". Напишите "скромную" программу — модули, которые не раскрывают ничего лишнего для других модулей и не полагаются на их внедрение. Попробуйте применить закон Деметера , который обсуждается в разделе "Несвязанности и закон Деметера". При надобности изменения состояния объекта это должен делать сам объект. В таком случае программа остается изолированной от реализации другой программы, а вероятность того, что система останется ортогональной, увеличивается.

- Избегайте глобальных данных. Всякий раз, когда ваша программа ссылается на глобальные данные, она привязывается к другим компонентам, использующим эти данные. Даже глобальные переменные, которые вы собираетесь использовать только для чтения, могут вызвать проблемы (например,

если вам нужно срочно изменить программу, сделав ее многопоточной). Вообще программа станет проще в понимании и сопровождении, если вы явно перешлете любой требуемый контекст в ваши модули. В объектно-ориентированных приложениях контекст часто пересылается как параметр к конструктора! объектов. В другой программе вы можете создать конструкции, содержащие контекст, и обходить ссылки на них.

- Подобные функции. Зачастую вы сталкиваетесь с набором функций, похожих друг на друга; возможно, они используют общий фрагмент в начале и конце программы, но в ее середине каждая пользуется своим алгоритмом. Дублированная программа является признаком структурных проблем. Для того чтобы составить программу лучше, следует обратить внимание на шаблон Strategy в книге "Design Patterns". Пусть постоянное критическое отношение к вашей программе войдет у вас в привычку. Ищите любые возможности реорганизации для усовершенствования ее конструкции и повышения уровня ортогональности. Этот процесс называется реорганизацией.

91.2.10 Тестирование

Систему, спроектированную и реализованную ортогональным образом, намного проще тестировать. Поскольку взаимодействие между компонентами системы формализовано и ограничено, большая часть тестирования может осуществляться на уровне отдельных модулей. Это хорошо, поскольку подобное тестирование значительно легче поддается спецификации и выполнению, чем интеграционное тестирование. Мы предлагаем, чтобы каждый модуль был снабжен своим собственным встроенным тестом, и эти тесты выполнялись автоматически как часть обычной процедуры сборки. Процедура сборки модульного теста сама по себе является интересным тестом на ортогональность. Что требуется, чтобы собрать и скомпоновать тест модуля? Должны ли вы задействовать большую часть системы только для того, чтобы скомпилировать или скомпоновать тест? В этом случае модуль очень хорошо связан с оставшейся

частью системы. Момент устранения ошибки также подходит для оценки ортогональности системы в целом. Когда вы сталкиваетесь с проблемой, оцените, насколько локален процесс ее устранения. Нужно изменить лишь один модуль, или изменения должны происходить по всей системе? Когда вы меняете что-либо, устраняются ли при этом ошибки или происходит загадочное появление новых? Это удачный момент для внедрения автоматизации. Если вы применяете систему управления исходным текстом, комментируйте устранение ошибок, когда вы осуществляете возвращение модуля в библиотеку после тестирования. Затем вы можете генерировать ежемесячные отчеты, где анализируются тенденции в ряде исходных файлов, в которых производилось устранение ошибок.

91.2.11 Документация

Что удивительно, ортогональность применима и к документации. Координатами являются содержание и представление. Если документация действительно ортогональна, вы можете существенно изменить внешний вид, не изменяя содержания. Современные текстовые процессоры содержат стили и макрокоманды, которые помогают в этом.

91.2.12 Жизнь в условиях ортогональности

Ортогональность тесно связана с принципом DRY (анг. «don't repeat yourself» — «не повторяй самого себя»). DRY — это базовый принцип заявленный в книге «Программист-прагматик» Эндрю Ханта и Дэйва Томаса (The Pragmatic Programmer). Авторы расматривали этот принцип в контексте баз данных, тестовых планов, проектирования программных систем, а также документирования систем[1] Используя этот принцип, можно свести к минимуму дублирование в пределах системы, а при помощи ортогональности уменьшить взаимозависимость между компонентами системы.

Звучит неуклюже, но если вы используете принцип ортогональности в тесной связи с принципом DRY, вы обнаружите, что разрабатываемые вами системы становятся более гибкими, более понятными и более простыми в отладке, тестировании и сопровождении. Когда вы присоединяетесь к проекту, в котором люди ведут отчаянную борьбу за внесение изменений, а каждое изменение приводит к появлению четырех новых проблем, вспомните кошмар с вертолетом. Вероятно, проект сконструирован и запрограммирован неортогонально.

Глава 92

Вспомогательные скрипты на языке Python

Название языка произошло вовсе не от вида пресмыкающихся. Автор назвал язык в честь популярного британского комедийного телешоу 1970-х «Летающий цирк Монти Пайтона». Впрочем, всё равно название языка чаще ассоциируют именно со змеёй, нежели с передачей — пиктограммы файлов в KDE или в Microsoft Windows и даже эмблема на сайте `http://www.python.org` (до выхода версии 2.5) изображают змеиные головы.

Python[1] — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода.

Python удобно применять для написания различных вспомогательных скриптов. Часто его используют при разработке сложных программных систем для написания первых версий. В процессе работы над большими программами часто перерабатываются большие объемы кода, поэтому для ускорения разработки требуется максимально высокоуровневый язык. После того как архитектура программы стабилизируется, узким местом становится производительность, и программу переписывают на более низкоуровневом компилируемом языке, чаще всего C_{+}^{+}.

Написание программ упрощают:

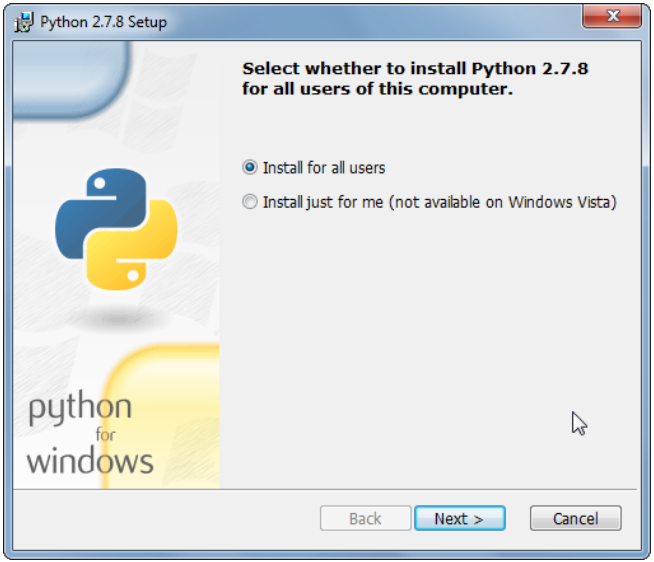
- **объектно-ориентированное программирование** облегчает разработку программ, позволяет переопределить стандартные операторы для пользовательских типов данных, упрощая синтаксис
- **динамическая типизация** не требуется заранее упределять переменные, они создаются простым присваиванием
- **обработка исключений** для секции кода можно определить обработчик ошибок
- **высокоуровневые структуры данных** — списки, словари (набор элементов ключ:значение), очереди
- богатая стандартная библиотека и множество дополнительных библиотек на все случаи

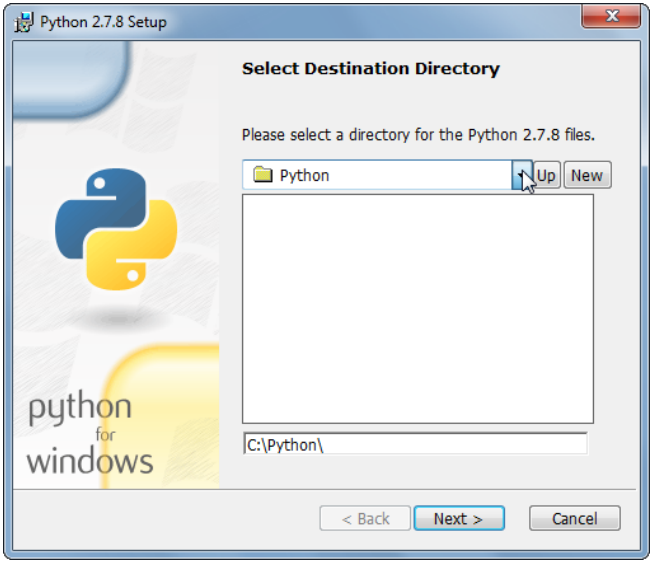
[1] в оригинале читается **па́йтон**, но давно русифицировался как **пито́н**

92.1 Установка под ⊞Windows

⊞ + R » http://www.python.org » Downloads » Python 2.7.8

python-2.7.8.msi » Setup » for all users/for me
Destination Directory » **C:/Python/** » Next





Customize » Python » Add python.exe to PATH » Next » Finish

92.2 Запуск

Из командной строки: ⊞ + R » cmd » python

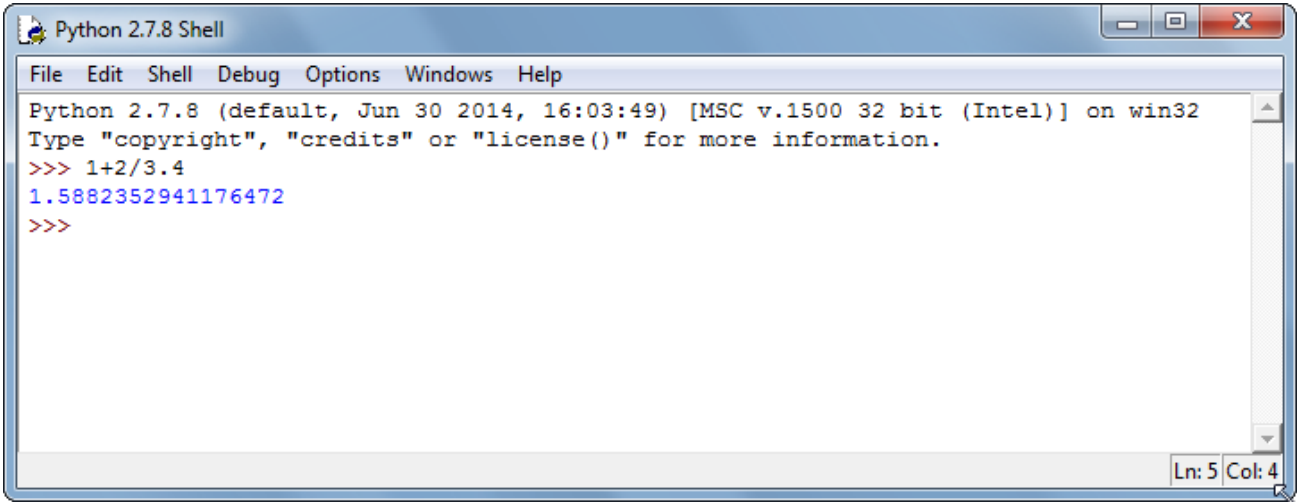
```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Users\dmitry>python
Python 2.7.8 (default, Jun 30 2014, 16:03:49) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+2/3.4
1.5882352941176472
>>> _
```

Простейшая среда IDLE[2]:

⊞ Программы Python 2.7 IDLE (Python GUI)
Панель задач IDLE ▷ Закрепить в панели задач ◁

[2] на GUI-библиотеке Tkinter, идущей в комплекте



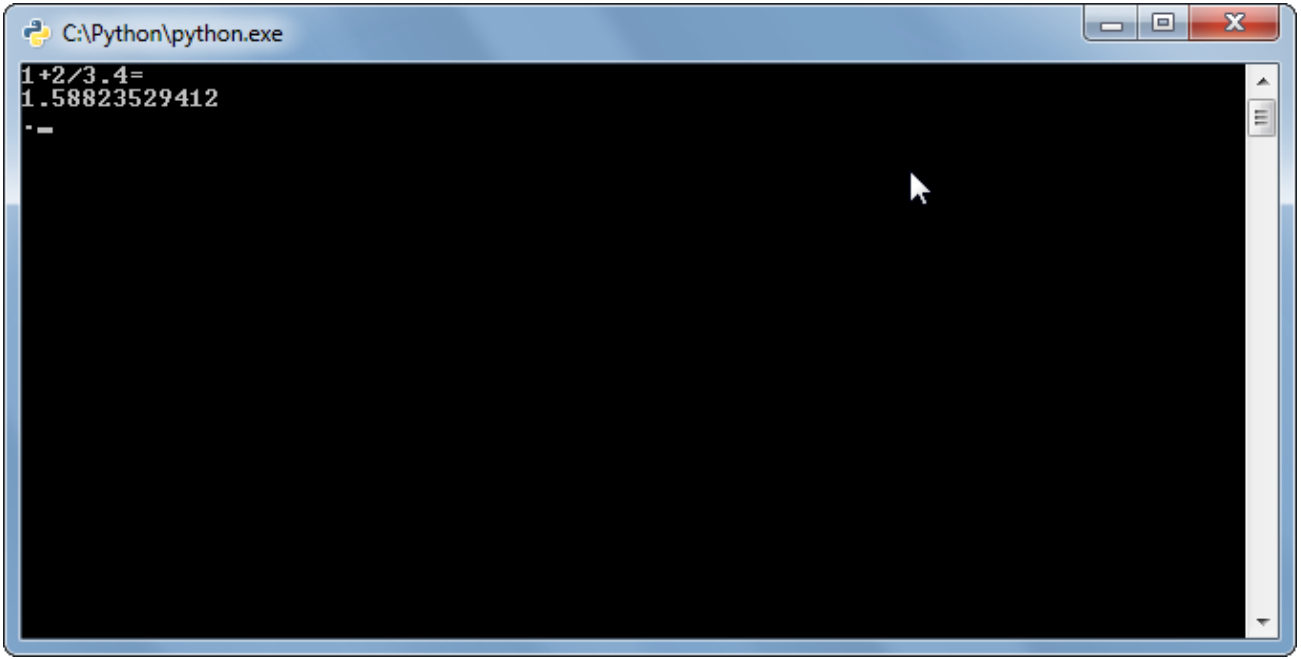
◁◁ по файлу скрипта:

⊞+R » notepad **/tmp/py.py**

/tmp/py.py

```
1 print "1+2/3.4="
2 print 1+2/3.4
3
4 raw_input('.')
```

⊞+R » /tmp/py.py



Открытием файла скрипта в IDLE:

⊞ + R /tmp/
py.py ▷ Edit with IDLE
меню Run Run Module F5

Python 2.7.8: py.py - C:\w\Azbuka\python\install\py.py

File Edit Format Run Options Windows Help

```
print "1+2/3.4="
print 1+2/3.4

raw_input('.')
```

Ln: 6 Col: 0

Python 2.7.8 Shell

File Edit Shell Debug Options Windows Help

```
Python 2.7.8 (default, Jun 30 2014, 16:03:49) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1+2/3.4=
1.58823529412
.
>>>
```

Ln: 8 Col: 4

92.3 Дополнительные материалы

[**pyotkidach**] Г. Россум, Ф.Л.Дж. Дрейк, Д.С. Откидач, Язык программирования Python
[**pythink**] Аллен Дауни Думать на языке Python: Думать как компьютерный специалист

Глава 93

Make: управление сборкой проектов

Глава 94

VCS: системы контроля версий

94.1 CVS

94.2 Subversion

94.3 Git

94.3.1 GitHub

Глава 95

Основы Си и C_+^+

95.0.2 Установка MinGW (win32)

95.1 Особенности C_+^+ в embedded

Глава 96

LLVM и разработка собственных компиляторов

Глава 97

Лексический и синтаксический анализ

Очень часто в практике возникает необходимость работы с данными в текстовых форматах — plain text файлы, в которых в каком-либо формате[1] описаны данные. И от вас требуется реализовать разбор такого файла, выделяя элементы данных, чтобы в дальнейшем после их преобразования например записать текстовый файл в другом формате.

В таком виде хранятся результаты расчетных программ, работащих в пакетном режиме, данные с измерительных систем, поток данных с приемников GPS[2], очень популярный мета-формат XML со всеми его частными случаями типа HTML, XLIFF??, OpenDocument, тексты программ для станков с ЧПУ,...

С некоторыми хинтами точно так же можно работать и с бинарными файлами, преобразовав их сначала в текстовую форму (в простейшем случае просто сделав hex dump).

[1] на языке разметки, или DDL [D]ata [D]efinition [L]anguage

[2] протокол NMEA 0183

В некоторых случаях необходимо написание трансляторов форматов (текстовых) данных, или даже интерпретаторов/компиляторов языков программирования.

Все эти техники с использованием стандартных утилит flex и bison будут кратко описаны в этой главе. Подробнее эти техники рассмотрены в книгах97.3, особенно стоит отметить талмуд **DragonBook**97.3.

97.1 Лексер и лексический анализ, утилита flex

Лексер — программа или ее часть, которая

1. получает на вход исходные данные в виде сплошного потока одиночных сиволов,
2. группирует символы согласно набору правил, и
3. отдает на выходе символы, уже сгруппированные в лексéмы или токéны.

Цель лексера — подготовить последовательность лексем для входа другой программы.

Лексический анализ — процесс программного разбора входной последовательности символов[3] с целью получения на выходе последовательности групп символов — токенов, имеющих собственное смысловое значение[4]. Как правило, лексический анализ производится в соответствии набора правил определённого формального, искуственного или компьютерного языка.

[3] например, такой как исходный код на одном из языков программирования

[4] подобно группировке букв в слово

Язык, а точнее его грамматика, задаёт определённый набор лексем, которые могут встретиться на входе лексера, и набор правил, по которым их следует группировать.

Традиционно принято организовывать процесс лексического анализа, рассматривая входную последовательность символов как поток одиночных символов. При такой организации лексер самостоятельно управляет выборкой отдельных символов из входного потока.

Распознавание лексем с учетом грамматики обычно производится путём их идентификации согласно идентификаторам токенов, определяемых грамматикой языка. При этом любая последовательность символов входного потока (лексема), которая согласно грамматике не может быть идентифицирована как токен языка, обычно рассматривается как специальный токен-ошибка.

Каждый выделенный токен можно представить в виде парной структуры, содержащей

1. идентификатор токена и
2. саму последовательность символов лексемы, выделенной из входного потока[5].

Рассморим обработку текстового файла: описания простой графической фигуры в текством формате IGES: универсальном формате обмена данными для САПР. IGES-файл состоит из 80-символьных ASCII-записей[6]. Текстовые строки представлены в «Холлерит»-формате – число символов в строке, плюс латинская буква [H] и сама строка, например — «4HSLOT». Рассмотрим очень короткий IGES-файл 1987 года, включающий в себя лишь сущности пары точек (POINT, тип 116), пары полуокружностей

[5] запись строки, числа и т. д.

[6] длина записи произошла из эры перфокарт, типовая ширина вывода терминала или принтера

(CIRCULAR ARC, тип 100) и двух линий (LINE, тип 110). Можно его считать эскизом паза под шпонку на валу, или шаблоном некруглого монтажного отверстия.

Файл разделен на 5 секций, обозначенных буквами (S, G, D, P, или T) в колонке 73. Характеристики и геометрическая информация для каждой сущности поделены между двумя секциями; одна — в формате фиксированной длины[7], другая в составной записи, с фиксированной точкой[8]. При отображении мы должны видеть две желтые точки, одна из которых в начале координат [0,0,0], две красных полуокружности и две зеленые линии.

sample.iges

```
                                                                        S      1
1H,,1H;,4HSLOT,37H$1$DUA2:[IGESLIB.BDRAFT.B2I]SLOT.IGS;,                G      1
17HBravo3  BravoDRAFT,31HBravo3->IGES  V3.002  (02-Oct-87),32,38,6,38,15,  G      2
4HSLOT,1.,1,4HINCH,8,0.08,13H871006.192927,1.E-06,6.,                   G      3
31HD. A. Harrod, Tel. 313/995-6333,24HAPPLICON - Ann Arbor, MI,4,0;     G      4
     116       1       0       1       0       0       0       0      1D      1
     116       1       5       1       0                              0D      2
     116       2       0       1       0       0       0       0      1D      3
     116       1       5       1       0                              0D      4
     100       3       0       1       0       0       0       0      1D      5
     100       1       2       1       0                              0D      6
     100       4       0       1       0       0       0       0      1D      7
     100       1       2       1       0                              0D      8
     110       5       0       1       0       0       0       0      1D      9
     110       1       3       1       0                              0D     10
```

[7] [D]irectory [E]ntry секция
[8] [P]arameter [D]ata секция

```
16      110      6      0      1      0      0      0      0      1D     11
17      110      1      3      1      0                             0D     12
18 116,0.,0.,0.,0,0,0;                                               1P      1
19 116,5.,0.,0.,0,0,0;                                               3P      2
20 100,0.,0.,0.,0.,1.,0.,-1.,0,0;                                    5P      3
21 100,0.,5.,0.,5.,-1.,5.,1.,0,0;                                    7P      4
22 110,0.,-1.,0.,5.,-1.,0.,0,0;                                      9P      5
23 110,0.,1.,0.,5.,1.,0.,0,0;                                       11P      6
24 S      1G      4D     12P      6                                   T      1
```

97.2 Компилятор Паскаля

97.3 Дополнительная литература

[**dragonbook**] **Книга Дракона**: Ахо, Сети, Ульман Принципы построения компиляторов.

Habr: Компиляция. 1: лексер
Habr: Компиляция. 2: грамматики
Habr: Компиляция. 3: бизон
Habr: Компиляция. 4: игрушечный ЯП
Habr: Компиляция. 5: нисходящий разбор
Habr: Компиляция. $5\frac{1}{2}$: llvm как back-end
Habr: Компиляция. 6: промежуточный код

Глава 98

Сборка кросс-компилятора GNU toolchain

Часть VII

Микроконтроллеры Cortex-Mx

Глава 99

Отладочные платы

99.1 STM32DISCOVERY /Cortex-M3 STM32F103/

99.2 STM32F4DISCOVERY /Cortex-M4 STM32F407/

Часть VIII

Периферия

Часть IX

Встраиваемый emLinux

Linux для встраиваемых систем[1] — популярный метод быстрого создания комплекса ПО для больших сложных приложений, работающих на достаточно мощном железе, особенно предполагающих интенсивное использование сетевых технологий.

За счет использования уже существующей и очень большой базы исходных текстов ядра, библиотек и программ для Linux, бесплатно доступных в т.ч. и для коммерческих приложений, можно на порядки сократить стоимость разработки собственных программных компонентов, и при этом получить готовую команду бесплатных стронних разработчиков, уже знакомых с созданием ПО для Linux.

Из недостатков можно отметить:

- Отсутствие полноценной поддержки режима жесткого реального времени;
- Тяжелое ядро;
  - Поддерживаются только мощные семейства процессоров[2];
  - Значительные требования по объему RAM и общей производительности;
- Дремучесть техспециалистов, контуженных ТурбоПаскалем и Windowsом;

Для сборки emLinux-системы используется метод кросс-компиляции, когда используется кросс-тулчейн, компилирующий весь комплект ПО для компьютера с другой архитектурой. Типичный пример — сборка ПО на ПК с процессором Intel i7 для Raspberry Pi или планшета на процессоре AllWinner/Tegra/. . . .

emLinux очень широко применяется на рынке мобильных устройств[3], и устройств интенсивно использующих сетевые протоколы (роутеры, медиацентры).

[1] будем называть его emLinux
[2] 32-бит, необходим блок MMU
[3] в т.ч. является основой Android

В качестве примера применения рассмотрим относительно простое приложение: многофункциональные настенные часы с синхронизацией времени через Internet, с будильником, медиапроигрывателем, блэкджеком и плюшками.

Глава 100

azLinux

Чтобы разобраться как можно собрать встраивамый Linux, в этом разделе описан набор Make-файлов и файлов конфигурации для сборки минимального emLinux. Это обрезанный форк проекта Cross Linux, подробнее описанного в разделе ??. Ограничено количество поддерживаемого железа, упрощены конфигурационные файлы, минимизировано количество библиотек и программных пакетов.

Изначально идея создания этой системы появилась из желания заменить тухлую связку x86/DOS/TurboPascal на что-то

- более переносимое: на энергоэффективное ARM/MIPS-железо, в т.ч. (типа)отечественного производства,
- стабильное: с полноценной многозадачностью, защитой памяти и данных, и

- позволяющее использовать максимум возможностей аппаратуры: большая RAM, ECC, gcc-оптимизированный 32/64-битный код, USB, CAN, Ethernet, WiFi, разнообразные носители данных, аппаратный watchdog.

- Также большой интерес представляют десятки готовые библиотек сжатия и кодирования данных, численных методов, ЦОС, и обработки изображений, а также

- множество готовых программ, доступных в исходных кодах[1] для выполнения различных полезных функций: сетевые серверы, символьная математика, обработка данных,. . .

- Еще одна ключевая фича — способность Linux полностью загружаться в RAM с **любых** носителей, в т.ч. заблокированных на запись. Это важно для случаев, когда возможны внезапные выключения питания: вся система работает в RAM-диске, а корректность записи данных на изменяемые носители можно гибко контролировать программно. При запуске после аварийного выключения никаких проверок файловых систем не требуется, ОС стартует сразу, а проверку/починку разделов данных возможно выполнять в фоновом режиме.

- Время запуска системы — на x86 удалось экспериментально получить время запуска 0.2 сек от загрузки ядра до начала выполнения пользователького кода. Используя модульное ядро, возможно выполнить критический к времени запуска пользовательский код до инициализации USB, сети, внешних носителей данных и тяжелых сервисов.

Почему не BuildRoot Эта система сборки создавалась как максимально облегченный пакет для решения узких задач, и для освоения технологии кросс-компиляции. Предполагается что функциональное наполнение не будет развиваться шире набора:

[1] для использования как есть, изучения принципов работы и модификации под собственные нужды

- ядро (реального времени)
- libc
- урезанная командная оболочка (busybox)
- несколько прикладных библиотек поддержки (сжатие, кодирование, базовая графика)
- пользовательский узкоспециализированный код на Си/C_+^+

Расширять функционал, добавляя libQt, X Window, Apache, MySQL,. . . , Gnome/KDE и т.д. не планируется в принципе — это система для решения узких прикладных задач на аппаратуре с минимальными ресурсами[2]. Интерактивная работа с пользователем также не предполагается, доступна только командная консоль[3], и очень ограниченные графические и мультимедийные возможности. Если ваши хотелки выходят за этот функционал, рекомендую сразу уходить на использование широко известной системы кросс-сборки Linux-систем под названием **BuildRoot??**.

100.1 Пакеты

Прежде чем продолжить, введем понятие пакет. В azLinux пакетом называется одна или несколько частей скриптов сборки, обозначаемых именем. В чем-то это похоже на бинарные пакеты обычных дистрибутивов Linux — чтобы добавить в систему какой-то функционал, мы устанавливаем бинарный пакет. Но есть и отличие: пакет дистрибутива это реальный архивный файл, содержащий в себе файлы программ, данных; в azLinux пакет — виртуальная штука с именем.

[2] особенно интересны процессорные модули в DIMM форм-факторе, только CPU, RAM, NAND и GPIO гребенка

[3] ее можно считать сервисным режимом

Просматривая файлы в каталоге **mk/**, легко найти имена пакетов по шаблону:

```
.PHONY: somename
somename: [зависимые файлы]
    [команда1]
    ...
```

Если вы запустите команду:

```
cd ~/az ; ./mk.rc && make somename
```

запустится сборка пакета somename.

Но не нужно забывать, что кроме этой секции в **.mk**, существуют зависимости между файлами, при работе команд сборки динамически создаются и изменяются файлы, иногда что-то скачивается из Interneta — все эти процессы тоже входят в пакет.

Часть пакетов не связана со сборкой программ, а выполняют служебные функции, поэтому для них правильнее будет фраза запуск пакета.

Все действия выполняются с помощью команды make. Обратите особое внимание на то, что **Makefile** собирается скриптом mk.rc из частей в каталоге **mk/**, поэтому если вы что-то меняете в скриптах, не забудьте сначала запустить ./mk.rc.

mk.rc

```
1 #!/bin/sh
2 cat \
```

```
3     mk/head.mk \
4     mk/dirs.mk \
5     mk/versions.mk \
6     mk/packages.mk \
7     mk/commands.mk \
8     mk/gz.mk \
9> Makefile
```

100.2 Загрузка

При необходимости вносить правки[4] работайте с вашим собственным форком на GitHub.

Получите клон пакета из репозитория:

```
cd ~ ; git clone --depth=1 -o gh https://github.com/user/azlin az
```

При необходимости обновитесь:

```
cd ~/az ; git pull
```

100.3 Фиксация переменных

Если вам требуется собрать систему со значениями переменных, отличающихся от тех, которые прописаны в make-файлах, при запуске всех пакетов нужно указывать трюбуемые значения в командной строке. Это позволить легко выбрать нужный вам вариант сборки.

[4] что естественно — вам потребуется добавлять свои пакеты и поддержку железа

```
./mk.rc && make HW=rpi APP=clock distclean dirs tc core libs apps boot root
```

При таком указании все переназначения для этих переменных игнорируются, поэтому возможны некоторые сложности с указанием например опций оптимизации.

100.4 dirs: Создание дерева каталогов

После загрузки или обновления запустите пакет dirs:

```
$ ./mk.rc && make dirs
mkdir -p /home/user/Azbuka/azlin/gz /home/user/Azbuka/azlin/src
/home/user/Azbuka/azlin/tmp /home/user/Azbuka/azlin/x86_64-linux-gnu
/home/user/Azbuka/azlin/qemu386-clock /home/user/Azbuka/azlin/qemu386-clock/boot

$ ls -la
итого 72
drwxr-xr-x 2 user user  4096 Дек  8 11:43 app
drwxr-xr-x 2 user user  4096 Дек  8 11:43 arch
-rw-r--r-- 1 user user 17699 Дек  8 13:25 azlin.tex
drwxr-xr-x 2 user user  4096 Дек  8 11:43 cpu
drwxr-xr-x 2 user user  4096 Дек  8 13:26 gz
drwxr-xr-x 2 user user  4096 Дек  8 11:43 hw
-rw-r--r-- 1 user user  2087 Дек  8 13:26 Makefile
drwxr-xr-x 2 user user  4096 Дек  8 11:43 mk
-rwxr-xr-x 1 user user   121 Дек  8 11:43 mk.rc
```

```
drwxr-xr-x 3 user user  4096 Дек  8 13:26 qemu386-clock
-rw-r--r-- 1 user user   147 Дек  8 11:43 README.md
drwxr-xr-x 2 user user  4096 Дек  8 13:26 src
drwxr-xr-x 2 user user  4096 Дек  8 13:26 tmp
drwxr-xr-x 2 user user  4096 Дек  8 13:26 x86_64-linux-gnu
```

Пакет dirs прописан в файле

mk/dirs.mk

```
# directories processing
# sw sources mirror archive in .tar.[GZ]
GZ = $(PWD)/gz
# [S]ou[RC]e codes unpacked
SRC = $(PWD)/src
# [T]e[MP] build dirs
TMP = $(PWD)/tmp
# build/target triplets
BUILD = $(shell gcc -dumpmachine)
# cross-compiler [T]ool[C]hain
TC = $(PWD)/$(BUILD)
# target root filesystem
ROOT = $(PWD)/$(HW)-$(APP)
BOOT = $(ROOT)/boot
DIRS = $(GZ) $(SRC) $(TMP) $(TC) $(ROOT) $(BOOT)
.PHONY: dirs
dirs:
    mkdir -p $(DIRS)
```

Встроенная переменная **PWD** содержит полное имя каталога, из которого был запущен make.

Каталог зеркала архивов исхоных текстов программ

GZ

```
GZ = $(PWD)/gz
```

Каталог распаковки исходных текстов: некоторые пакеты должна собираться в дереве исходников

SRC

```
SRC = $(PWD)/src
```

Каталог out-of-tree сборки: остальные пакеты умеют собираться вне дерева исходников, если хватает RAM этот каталог удобно монировать как **tmpfs**

TMP

```
TMP = $(PWD)/tmp
```

Переменная **BUILD** задает триплет системы, на которой вы собираете: x86_64-linux-gnu, i686-linux-gnu или что-то подобное. Для получения триплета используется подстановка строки, выдаваемой запуском gcc.

BUILD

```
BUILD = $(shell gcc -dumpmachine)
```

Каталог в который собирается кросс-компилятор. Используется триплет рабочей Linux-системы.

TC

```
TC = $(PWD)/$(BUILD)
```

Каталог целевой **rootfs**. Отдельно прописан загрузочный каталог, в который будет записываться собранное ядро, образ initrd, бинарники и конфиги загрузчика. Имя **ROOT** создается из двух переменных, описанных далее: имя аппаратной платформы **HW**100.6 и имени приложения **APP**100.5.

ROOT/BOOT

```
ROOT = $(PWD)/$(HW)-$(APP)
BOOT = $(ROOT)/boot
```

Список всех рабочих каталогов в одной переменной:

DIRS

```
DIRS = $(GZ) $(SRC) $(TMP) $(TC) $(ROOT) $(BOOT)
```

100.5 APP: Приложение

Приложение — короткое кодовое название вашего варианта сборки системы в целом.
Приложение задается в файле **mk/head.mk** через переменную **APP**, доступные значения:

1. micro: минимальная версия системы, только командная консоль
2. clock: простые Linux-powered электронные часы

Значение переменной **APP** по умолчанию задано в **mk/head.mk**:

APP @ hw/head.mk

```
APP = clock
```

Если вам нужно собрать другое приложение, вы можете переопределить значение из командной строки при запуске всех пакетов:

```
./mk.rc && make APP=micro distclean dirs tc core libs apps
```

В **app/${APP}.mk** в переменных задаются:

- **LIBS**: набор используемых библиотек
- **PACKS**: набор используемых программных пакетов 100.8

app/micro.mk

```
# app: micro
LIBS =
APPS =
```

app/clock.mk

```
# app: clock
LIBS = sdl
APPS =
```

100.6 HW: Поддерживаемое железо

Конфигурация целевого железа задается в файле **mk/head.mk** через переменную **HW**, доступные значения приведены в таблице:

HW	CPU	ARCH	RAM	HDD/NAND	SD	USB	WiFi	GPIO
qemu386	i486sx	i386	32M+	☐ IDE/SATA		☐		
eeepc701	CeleronM	i386	512M+	☐ SSD 4G	☐ SD	☒	☐ Atheros AR2425	
qemuARM		arm	32M+					
cubie1	AllWinnerA10	armhf	1G		☐ μSD	☒		
rpi	BCM2835	armel	512M		☐ SD	☒		
qemuMIPS		mips	32M+					
mr3020	AR7240	mips	32M	4M		☒	☐ Atheros AR9331	
vocore	RT5350	mips	32M	8M		☐	☐ SoC	
bswift	AR9331	mips	64M	16M NOR		☐ host		20+

mk/head.mk

```
# [H]ard [W]are: qemu386 qemuARM qemuMIPS cubie1 rpi ...
HW = qemu386
# [APP]lication: micro clock
APP = clock
# load extra hw definitions: ARCH CPU ...
include hw/$(HW).mk
# load extra defs for CPU setted in $(HW).mk
include cpu/$(CPU).mk
# load extra defs for ARCHitecture setted in $(CPU).mk
include arch/$(ARCH).mk
```

```
# load extra app defs: LIBS APPS ...
include app/$(APP).mk
```

100.6.1 i386

Персоналки с архитектурой **i386** — самое сложное семейство с точки зрения поддержки. Комбинации процессоров, материнских плат и плат расширений дают сотни вариантов конфигураций, в т.ч. десятки моделей компьютеров в формате PC/104 и промышленных панелей. Особенно доставляет тот факт, что 95% периферии имеет закрытые бинарные драйвера только для ⊞Windows, поэтому будьте аккуратны с выбором железа.

qemu386: эмулятор QEMU

hw/qemu386.mk

```
# QEMU emulator: i386 mode
CPU = i486sx
```

eeepc701: ASUS Eee PC 701

hw/eeepc701.mk

```
# ASUS Eee PC 701
CPU = CeleronM
```

100.6.2 ARM

qemuARM: эмулятор QEMU

hw/qemuARM.mk

---

cubie1: Cubie Board v.1

rpi: Raspberry Pi model B

100.6.3 MIPS

qemuMIPS: эмулятор QEMU

hw/qemuMIPS.mk

---

mr3020: роутер MR3020

mr3020

vocore: VoCore

vocore

bswift: BlackSwift

bswift

100.7 CPU: Конфигурации процессоров

Настройки на процессор задаются в файле **cpu/${CPU}.mk**.

100.7.1 i386

cpu/i486sx.mk

```
# i486sx cpu
# target arch is Intel x86 (32 bit)
ARCH = i386
# target triplet for embedded linux
TARGET = i486-linux-uclibc
```

cpu/CeleronM.mk

```
# Intel Celeron M ULV 353
ARCH = i386
TARGET = celeronm-linux-uclibc
```

100.7.2 ARM

100.7.3 MIPS

cpu/AR7240.mk

```
# Atheros AR7240 CPU (400Mhz)
ARCH = mips
TARGET = mips-linux-uclibc
```

cpu/RT5350.mk

```
# Ralink RT5350 360MHz
ARCH = mips
TARGET = mips-linux-uclibc
```

100.8 Пакеты

Глава 101

cross

Глава 102

BuildRoot

Глава 103

Особенности OpenWrt

Глава 104

Библиотека SDL

104.1 Реализация microGUI

Глава 105

Приложения для X Window

Глава 106

Программирование сетевых приложений

Глава 107

Сборка кросс-компиляторя GNU мальтийским крестом

Часть X

IDE

IDE — Integrated Development Environment, интегрированная среда разработки.
Программный пакет, включающий

- средства управления проектом,
- отслеживание зависимостей между файлами (в т.ч. с анализом исходного текста программ на конструкции типа `#include`, `module`, `uses`),
- автозапуском компиляторов для изменившихся файлов,
- GUI для отладчиков (gdb),
- специализированный редактор plain text[1] файлов с
  - цветовой и шрифтовой подстветкой синтаксиса,
  - автодополнением: дописываются имена объектов программ, синтаксические конструкции и параметры функций,
  - автоформатированием: фрагмент текста переформатируется в соответствии с синтаксисом языка редактируемого файла, проставляются отступы в зависимости от вложенности синтаксических конструкций типа циклов и условных блоков)
  - выделением строк, на которые указывают сообщения об ошибках компиляторов,
  - маркеры точек останова отладчика
- отображение структуры программ, например деревья классов и структур данных

[1] файлы не включающие непечатаемых символов и бинарных данных, которые можно причитать простым выводом на экран командами типа **type**, **cat**, **more**

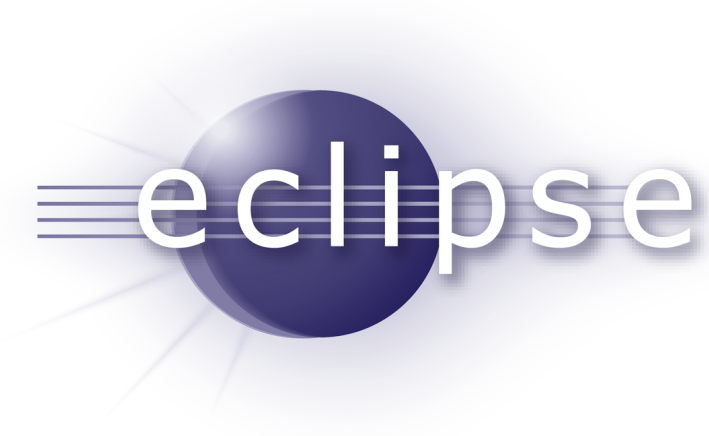
- контекстные справочники по используемым языкам программирования, автоматический вывод списка параметров при вводе имени функции
- отображение дизассемблерных листингов для компилируемых языков
- отображение браузера как вкладки или MDI окна
- отображение вывода статических анализаторов программ с кликабельными ссылками
- вывод компиляторов и трансляторов с цветовым выделением и переход на ошибочную строку в редакторе при щелчке на ошибке
- ...

В этой книге рассмотрены три бесплатных мультиплатформенных OpenSource IDE, в порядке навороченности, универсальности, и требуемым ресурсам для работы самой среды:

1. ECLIPSE 108: самая навороченная и ресурсоемкая IDE, написана на Java, имеет десятки дополнительных модулей на все случаи, умеет работать со всеми распространенными языками программирования, жрет память, и требует современного компьютера минимум с 2+ Гб ОЗУ. Последний релиз ECLIPSE Luna работает заметно быстрее (особенно при запуске).
2. Code::Blocks 109: легкая среда для разработки на C/C_+^+, для других языков модет потребоваться написать свои модули или файлы описания синтаксиса
3. (g)Vim 110: самый легкий и портабельный универсальный текстовый редактор с расширенными функциями, работает на всех существующих платформах (кроме совсем уж embedded), использует минимум ресурсов, но требует некоторого обучения даже чтобы выйти из `vim` ☺

Глава 108

eclipse



108.1 Установка eclipse под Windows

⊞ + R » https://eclipse.org/ » Download » Eclipse Luna release for » ⊞Windows
Качаем архив базовой системы: Eclipse IDE for Java Developers » ⊞Windows 32/64 Bit
Или сразу сборку CDT ECLIPSE: Eclipse IDE for C/C++ Developers » ⊞Windows 32/64 Bit

108.2 Установка eclipse под Linux

⊞ + R » https://eclipse.org/ » Download » Eclipse Luna release for » Linux
Качаем архив базовой системы: Eclipse IDE for Java Developers » Linux 32/64 Bit
Или сразу сборку CDT ECLIPSE: Eclipse IDE for C/C++ Developers » Linux 32/64 Bit

Пока качается, параллельно устанавливаем в систему Java-рантайм:

```
sudo aptitude install openjdk-7-jre
```

Распаковывем полученный архив **eclipse-java-luna-SR1-linux-gtk-x86_64.tar.gz** в **$HOME**:

```
cd ~
tar zx < Downloads/eclipse-java-luna-SR1-linux-gtk-x86_64.tar.gz
ls -la eclipse/eclipse
-rwxr-xr-x 1 user user 74675 Авг 13 16:12 eclipse/eclipse
```

Прописываем запуск ECLIPSE в ваш оконный менеджер или **.blackboxmenu** с параметром **-noSplash** для лечения глюка с запуском на x64-битных системах:

.blackbox.menu

```
[begin] (Linux)
[exec] (eclipse) {eclipse/eclipse -noSplash}
```

108.3 Установка CDT

CDT — расширение ECLIPSE для разработки на Си/C_{+}^{+}, редактирования make-файлов. Это расширение критически важно для вашей работы, поэтому ставить его обязательно, или сразу качать сборку CDT ECLIPSE.

ECLIPSE » Help » Install New Software...
Work with » Add » Add repository
Name » CDT
Location » http://download.eclipse.org/tools/cdt/releases/8.5
OK
Work with » CDT
CDT Main Features » ☒ C/C++ Development Tools
CDT Optional Features
Парсер файлов исходников на диалекте C99: ☒ C99 LR Parser
Поддержка gcc в режиме кросс-компиляции: ☒ GCC Cross Compiler Support
Аппаратная отладка через gdb: ☒ GDB Hardware Debugging
Next » Next » Accept » Finish

108.4 Установка PyDev

PyDev — расширение для разработки на Python:

ECLIPSE > Help > Install New Software...
Work with > Add > Add repository
Name > PyDev
Location > http://pydev.org/updates
OK
Work with > PyDev
PyDev > ☒ PyDev for Eclipse
Next > Next > Accept > Finish > Certitificate > Restart Eclipse > Ok

108.5 Установка TeXlipse

Если планируете работать с документацией в формате LaTeX, установите расширение TeXlipse:

ECLIPSE > Help > Install New Software...
Work with > Add > Add repository
Name > TeXlipse
Location > http://texlipse.sourceforge.net/
OK
Work with > TeXlipse

Это расширение поддерживает подсветку синтаксиса, автодополнение, построение динамического оглавления, автокомпиляцию по сохранению, и несколько визардов создания проекта.

108.6 Редактирование файлов в формате XML и производных

Установите пакет ECLIPSE WST:
Help > Install New Software
Work with: > Luna - http://download.eclipse.org/releases/luna
Filter: > WST > Eclipse WST > Next > Next > Restart > OK

108.7 Проверка орфографии

[1]

То, что проверка орфографии очень удобная вещь вряд ли нужно объяснять. Есть конечно люди, которые не обращают на неё внимание, но это чаще всего из-за экономии времени и отсутствия удобных средств проверки.

Действительно, удобная автоматическая проверка орфографии есть в офисных пакетах, но мне сложно представить разработчика, который будет переносить комментарии в Word и обратно ☺.

Поэтому очень удобно иметь проверку правописания прямо в IDE. И ECLIPSE в этом смысле полностью соответствует ожиданиям.

Долго объяснять что к чему нет смысла. Проверка орфографии встроена в ECLIPSE и если вы пишите только на английском, то может быть не захотите ничего менять.

Кроме того, есть статья Aaron'a (en) в которой автор рассказывает о подключении дополнительных словарей и плагине **eSpell**.

[1] копипаста http://www.simplecoding.org/proverka-orfografii-v-eclipse.html

Но русских словарей в дистрибутиве нет, а при подключении внешних есть нюансы. Поэтому мы максимально подробно рассмотрим подготовку и добавление русских словарей.

Первый вопрос. В каком виде должны быть словари и где их взять?

Тут всё просто. Формат словаря — обычный текстовый файл, в котором каждое слово начинается с новой строки. И нам вполне подойдут свободно распространяемые словари **aSpell**.

Установка состоит из 4 шагов:

1. качаем aSpell и словари для нужных языков

   ⊞ + R » http://aspell.net/win32/

   Binaries » Full installer

   Precompiled dictionaries » English

   Precompiled dictionaries » Russian

2. устанавливаем сначала **aSpell**, потом отдельно каждый словарь

   Aspell-0-50-3-3-Setup.exe » Setup GNU Aspell » Next » License » Next

   Directory » **C:/GnuWin32/Aspell** » Next » Next

   Additional » Next » Install » Next » ☐ View manual » Finish

   Aspell-en-0.50-2-3.exe » Aspell English Dictionary » Next » License » Next

   Directory » **C:/GnuWin32/Aspell** » Next » Next » Install » Finish

   Aspell-ru-0.50-2-3.exe » Aspell Russian Dictionary » Next » License » Next

   Directory » **C:/GnuWin32/Aspell** » Next » Next » Install » Finish

3. делаем дамп словарей, перекодируем из koi8r в utf8 и объединяем

   ⊞ + R cmd

```
cd \GnuWin32\Aspell
bin\aspell dump master en > en.dict
bin\aspell dump master ru > ru.koi8
iconv -f koi8-r -t utf-8 < ru.koi8 > ru.dict
copy en.dict + ru.dict enru.dict
```

4. настраиваем spell-checker ECLIPSE

   ECLIPSE > Window > Preferences > Editors > Text editors > Spelling

   User defined dictionary > **C:/GnuWin32/Aspell/enru.dict**

   Encoding > UTF-8

   Apply > OK

Глава 109

Code::Blocks

Глава 110

(g)Vim

110.1 Установка под ⊞Windows

⊞ + R cmd » http://www.vim.org/ » Download » PC: MS-DOS and MS-Windows » **gvim74.exe**
Vim 7.4 Setup » This will install » Да
License » I'm Angry
Installation Options » ☒ Create .bat files » Next
Installation Folder » Install
Completed » Close
Do you want to see README » **Да**

Теперь можно настроить темную тему и выключение подстветки синтаксиса, по умолчанию после установки используется светлая тема и подстветка выключена:
меню » Правка » Настройка запуска

Переходим в конец файла и включаем режим вставки
Ctrl + Down Ins Enter Enter

```
syntax on
colorscheme pablo
```

Выходим в режим команд и принудительно сохраняем
Esc : w ! Enter Enter

Выходим из (g)Vim
Esc : q ! Enter

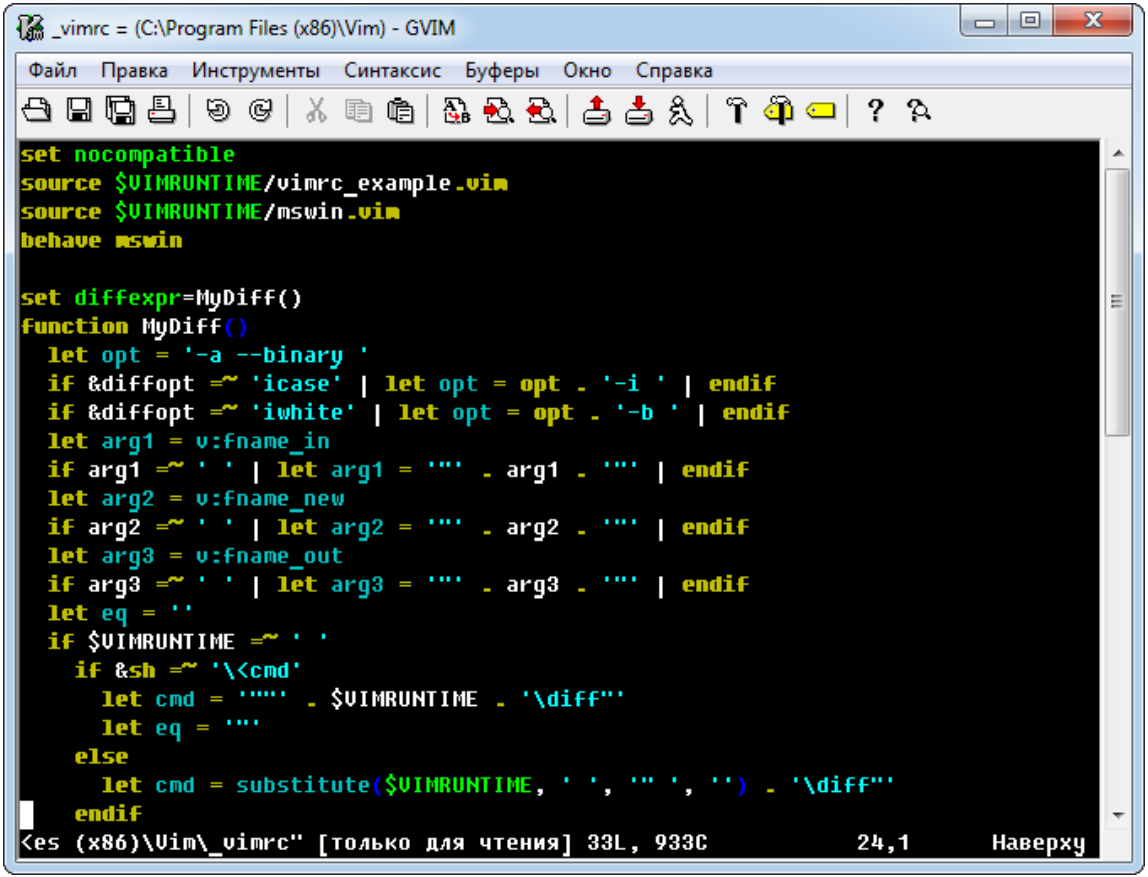
Если не получилось (под Windows 7):

⊞ + R cmd » **/Program Files (x86)/Vim/**

Копируем файл **_vimrc** в любой каталог, например в **/tmp/**, затем ▷▷ » Edit with Vim, и повторяем редактирование еще раз.

Затем копируем **_vimrc** обратно в **/Program Files (x86)/Vim/** с заменой.

Если теперь открыть на редактирование тот же файл, или любой другой текстовый, получим более удобный вид: для файлов известных типов будет работать подсветка синтаксиса.



110.2 Выход из (g)Vim

Esc : ! q Enter

110.2.1 Выход с автосохранением

Esc Shift + Z Shift + Z

110.3 Переход в режим редактирования

(g)Vim запускается в командном режиме, для перехода в режим редактирования используются следующие клавиатурные команды:

- Ins или i: включение режима вставки по текущему положению курсора
- Ins Ins или r: включение режима перезаписи поверх текста после курсора
- Shift + A: включение режима вставки в конец текущей строки

110.4 Переход в режим команд

Esc

110.5 Запись редактируемого файла

Esc : w Enter

Если выводится предупреждение типа “файл защищен от записи” или подобное, может сработать принудительная запись:

Esc : ! w Enter

110.6 Перезагрузка файла

Для перезагрузки возможно измененнного извне файла или отмены всех несохраненных изменений

Esc : e Enter

110.7 Отмена последних изменений (undo)

Esc u u . . .

Часть XI

Замечания для авторов

110.8 Набор репозиториев на GitHub

https://github.com/ponyatov/Azbuka	основная репа
https://github.com/ponyatov/bib	библиографические базы данных
https://github.com/ponyatov/scratcher	журнал, используются некоторые материлы

Для работы с проектом сделайте собственный форк основной репы, библиографическую базу и журнал можете клонироввать напрямую. Создайте каталог и склонируйте репы:

```
D:
cd \
mkdir w
cd \w\
git clone --depth=1 -o gh git@github.com:username/Azbuka.git

git clone --depth=1 -o gh git@github.com:ponyatov/bib.git
git clone --depth=1 -o gh git@github.com:ponyatov/scratcher.git
```

110.9 Верстка в LaTeX

Было много вопросов по выбору языка разметки, и даже предложения некоторые материалы просто навордятить. Но все же используется LaTeX, т.к. это самая наворченная система подготовки больших изданий, широко распространенная (в узких кругах), и прежде всего как имеющая богатейший набор пакетов-расширений для всевозможных вывертов.

LaTeX не предназначен для верстки полноцвета, журнальной верстки или ручного таскания блоков по листу. LaTeX изначально был заточен на подготовку научно-технической и учебной многостраничной литературы **с логической разметкой**.

Как профессиональный инструмент, LaTeX требует обучения. Начерно навордятить на нем текст, накидав как попало картинок, и наляпав шрифтов[1] не получиться. И это хорошо.

Но — материалы на добавление принимаются в любых форматах, группой авторов, способных их доварить до нужного качества. Единственное ограничение: наличие бесплатных средств просмотра на трех основных платформах: ⊞Windows, Linux и MacOS, или в онлайне (Google Docs, M$ облака, и прочие сетевые болота).

Также приветствуетсся использование различных более простых языков разметки (SPHINX, Wiki, DocBook, .md, ...). Для первоначального сбора и группировки материала они проще для освоения, чаще всего рендер-движки для этих языков заточены под веб-редактиврование в т.ч. групповое, и хорошо подходят для простой по оформлению документации на программные пакеты, или как сборник ссылок на другие ресурсы.

Для включения таких материалов в основную верстку несложно написать TeX-транслятор (если нет сразу готового), который создаст .tex файлы нужного вида с минимумом ручной доработки.

[1] про существование стилей многие вордятники даже не слышали

Часть XII

Подготовка публикаций в LaTeX

Подробное описание этого слона см. [Ivovsky].

Внимание: LaTeX не предназначен для верстки полноцветных и журнальных изданий, его область — научно-техническая, учебная и художественная литература, и пакетная верстка материалов, сформированных автоматически (отчеты работы программ, графики и анализ экспериментальных данных, и т.п.)

[2]

LaTeX (по-русски произносится **латéх**) — наиболее популярный набор макрорасширений (или макропакет) системы компьютерной вёрстки TeX, который облегчает набор сложных документов. В типографском наборе форматируется как LaTeX.

Главная идея LaTeX состоит в том, что авторы должны думать о содержании, о том, что они пишут, не беспокоясь о конечном визуальном облике (печатный вариант, текст на экране монитора или что-то другое). Готовя свой документ, автор указывает логическую структуру текста (разбивая его на главы, разделы, таблицы, изображения), а LaTeX решает вопросы его отображения. Так содержание отделяется от оформления. Оформление при этом или определяется заранее (стандартное), или разрабатывается для конкретного документа.

В практическом смысле использование LaTeX позволяет (в порядке уменьшения важности):

- с помощью макросов и TeX-программирования реализовывать любые стили и сложную верстку, существует множество готовых пакетов для верстки графических химических формул, разнообразных схем, транскрипционных знаков, внезапно электронных схем, цветных листингов и т.п.
- автоматизировать работу с документами: пересобирать выходные файлы через Make, генерировать части документов с помощью своих скриптов[3]

[2] копипаста https://ru.wikipedia.org/wiki/LaTeX

[3] отчеты, стандартные формы, результаты работы любых программ

- получить выходой документ в .pdf .html .txt .PostScript .djvu ... с кликабельными ссылками, анимированными, а иногда и интерактивными элементами
- не использовать файлы документов в закрытом формате
- легко держать набор файлов в VCS
- не покупать текстовый процессор

Особенно важен пункт про сложную верстку: она всегда нужна в крупных технических публикациях, особенно в учебной литературе, или отчетных работах. Вам обязательно понадобиться вставлять графики экспериментальных данных, тематически специфичные схемы, листинги, выходные данные работы ваших программ и т.п.

Традиционно LaTeX любим математиками, и всеми кто готовит публикации с большим количеством формул и перекрестных ссылок: после небольшого обучения формулы вводятся с листа со скоростью набора текста, особенно если ваш редактор умеет автодополнение, и никакой мышиной возьни.

Естественно всякие чисто автоматические вещи типа автонумерации ссылок и формул, сборки оглавлений и индексов, цветовая подсветка синтаксиса в листингах программ, размещение плавающих иллюстраций и т.п. выполняются автоматически TeX-процессором в пакетном режиме, и на выходе получается красивый печатный или электронный (.pdf) документ.

Единственная область, не удобная в LaTeX-верстке — создание сложных таблиц. Для этого были созданы визуальные редакторы, позволяющие отрисовать структуру таблицы мышью, а затем заполнить готовый шаблон данными.

110.10 Установка MikTeX под ⊞Windows

⊞ + R » http://miktex.org/ » Download » Recommended » Basic MiKTEX Installer

110.11 Структура документа

110.11.1 Заголовочный файл или блок

110.11.2 Стили документа

110.11.3 Пакеты

110.11.4 Автор и название

110.11.5 Верстка титульных страниц

110.11.6 Оглавление

110.12 Верстка слайдов

110.13 Список литературы и цитирование

LATEX умеет мощную подсистему управления цитированием и списками литературы. В простейшем случае, например при написании единственной статьи, раздел библиографии можно создать в том же документе, добавив в конец `thebibliography`:

```
\documentclass{article}

\input{header}

\author{Вася Пупкин}
\title{Пример статьи с цитатами}

\begin{document}
\maketitle

В статье используются книги: \cite{A} и \cite{B}

\begin{thebibliography}{99}

\bibitem{A} Книга A

\bibitem{B} Книга B

\end{thebibliography}
\end{document}
```

Но если вы регулярно работаете с документацией, или часто пишете статьи, возникает естественное желание вынести весь список литературы в отдельную базу данных, прописать авторов, названия, издательства и т.п. Это делается с помощью программы **biber** и пакета **biblatex**.

Пример использования этой системы вы легко найдете в исходниках этой книги:

- файл **header.tex** содержит секцию подключения пакета и подгрузки библиофайлов:

```
% books bib management
\usepackage{biblatex}
\addbibresource{../bib/python.bib}
\addbibresource{../bib/eskd.bib}
...
```

- библиофайлы хранятся в **соседнем** репозитории **../bib**, склонированном с https://github.com/ponyatov/bib.
- порядок вызова **pdflatex** и **biber** см. **Makefile**

Для оформления библиографии в нужном стиле см. примеры [**bibiso**].

110.14 Команды секционирования: часть, глава, раздел,..

110.15 Таблицы

110.16 Формулы

110.17 Перекрестные ссылки и гипессылки

110.18 Листинги скриптов и текстовых данных

110.19 Подготовка иллюстраций

Подготовка иллюстраций — одна из самых геморных тех в создании документации, и ее верстке для бумажных и электронных изданий.

Предпочтение нужно отдавать векторным форматам, за исключеним фотоиллюстраций. В идеале скриншоты также хорошо бы переводить в векторыный формат, но пока инструмент для этого не найден, поэтому выходные файлы будут пухнуть в объеме.

Для подготовки векторных иллюстраций: схем, графиков, диаграмм и т.п. используйте пакеты, принимающие на вход программы на специализированном языке программрования, легко читаемым человеком. В этом случае у вас сохраниться отслеживать изменения, читая логи VCS.

Обратите внимание на возможность использования стилевых файлов на весь проект (для всех иллюстраций в книге например). Их использование даст профессиональный вид продукту, при этом со-

храниться возможность взять и переформатировать 100500 схем в 10-томнике, поменяв шрифт, цвета, толщины линий, зазоры между элементами и т.п.

Пользуйтесь только относительными единицами размеров, и привязывайтесь к размерам шрифтов, это даст возможность использовать готовую иллюстрацию в нескольких проектах с разными размерами бумаги и наборами используемых шрифтов.

110.19.1 Графики GNUPLOT

Самый постой способ получит график простой аналитической функции или экспериментальных данных — воспользоваться утилитой GNUPLOT.

Оценить возможности можно вот по этому видео

Примеры на википедии

Примеры выложенные вместе с текстом на языке gnuplota

110.19.2 Схемы и графы в GraphViz

Для отрисовки графов и схем, легко к ним сводящихся, можно использовать пакет **GraphViz** и язык **Dot**.

110.19.3 PGF/TikZ

Сложные графики можно рисовать с помощью пакета **PGF/TikZ**, но для его работы нужна установленная LaTeX-система. Этот пакет предназначен прежде всего для набора и верстки изданий с множеством сложных схем.

110.19.4 GLE

GLE — универсальный язык описания векторных графических объектов с элементами языка программирования. Поддерживает вычисления, типовые конструкции программирования (циклы, условия, рекурсию).

- графики
- 3D графики
- диаграммы
- фракталы
- электронные схемы
- исчо

110.19.5 Xy-pic

110.20 Верстка электронных изданий

Для электронных изданий, предназначенных для чтения с различных экранов как компьютера, так и портативных устройств, существует ряд ограничений и рекомендаций, из-за особенностей экранов: малый размер, низкое разрешение, поддержка цвета (TFT vs e-Ink) и т.п.: [**ebooktex**]

Установка полей в .PDF:

```
\hypersetup{
pdftitle={Азбука халтурщика-ARMатурщика},
pdfauthor={ruOpenWrt, HackSpace Чебураторный завод, Bill Collis (part 1)}
}
```

Часть XIII

Символьная и численная математика

В практике любого инженера математика занимают главную роль. Без хорошего знания математики, причем практически всех областей, от школьной до дифференциального исчисления, работать в этой области практически невозможно.

Прежде всего свободное знание математики, физики, и химии необходимо для чтения любой литературы, если вам нужно разобраться в какой-либо прикладной области. Очень часто приходится реализовывать некоторые численные методы вычислений, выполняющиеся в вашем устройстве в реальном времени, для управления процессами, обработки сигналов с датчиков, принятия решений о включении исполнительных устройств и т.п. Ну и конечно вы не сможете создать само устройство, не понимая принципы его работы ☺. Это конечно не относится к различным простейшим устройствам типа таймеров или простой автоматики, но стоимость заказов такого типа $\to 0$.

Если вы хотите поднять или восстановить свой уровень знания базовых наук (а заодно и английского), удобно воспользоваться ресурсом `https://www.khanacademy.org/`: это знаменитая on-line академия **Khan Academy**, имеющая как набор видеолекций по базовым техническим наукам, так и большую батарею тестов для проверки ваших знаний. Не забывайте периодически проходить все тесты, чтобы поддерживать свои знания рабочими. Из недостатков — отвратнейшая реализация на мобильных устройствах, часть тестов просто не работает, а ввод ответов крайне неудобен из-за необходимости постоянно пользоваться (полно)экранной клавиатурой и переключения на числовой ввод.

На русском языке ресурсов такого класса к сожалению пока не попадалось. Кое-что есть кусочками, но по большей части только лекции в стиле «книжкой по башке», похоже навыков вводного обучения в России просто не существует. Если есть силы и желание, можете сами реализовать проект по созданию онлайн системы базового образования ☺.

В этом разделе собраны примеры проектов, требующие некоторых базовых знаний, а также рассмотрено использование OpenSource программ для вычислений и обработки данных.

Глава 111

Общие сведения о компьютерной математике

[1][2]

Для начала пару слов о том, что из себя представляют эти самые символьные или, как их еще называют, аналитические вычисления, в отличие от численных расчетов. Компьютеры, как известно, оперируют с числами, целыми и с плавающей запятой[3]. К примеру, решения уравнения $x^2 = 2x + 1$ можно получить как -0.41421356 и 2.41421356, а $3x = 1$ — как 0.33333333. А ведь хотелось бы увидеть

[1] копипаста http://maxima.sourceforge.net/ru/maxima-tarnavsky-1.html

[2] Тихон Тарнавский. Maxima — максимум свободы символьных вычислений

[3] на самом деле настоящую "плавучку" поддерживают только достаточно мощные процессоры, не хуже i486dx, встраиваемые не-DSP CPU/MCU аппаратно работают только с целыми числами: ± 127, $\pm 2^{16-1}$ и $\pm 2^{32-1}$ в зависимости от разрядности ядра 8- 16- или 32-бит

не приближенную цифровую запись, а точную величину, т. е. $1 \pm \sqrt{2}$ в первом случае и $1/3$ во втором. С этого простейшего примера и начинается разница между численными и символьными вычислениями.

Но кроме этого, есть еще задачи, которые вообще невозможно решить численно или наоборот аналитически.

Например, параметрические уравнения, где в виде решения нужно выразить неизвестное через параметр; или нахождение производной от функции; да практически любую достаточно общую задачу можно решить только в символьном виде.

Наоборот, для многих задач не существует точного аналитического решения, и приходится применять численные методы их решения.

В некоторых случаях нужно получение простого и быстрого приближенного решения — это может понадобится в системах управления, когда микроконтроллер не успевает за управляемым процессом, если пытается получить точное численное решение. При обработке сигналов например не требуется точное решение, достаточно результата, получаемого численными методами.

Для решения аналитических задач давно появились компьютерные программы, оперирующие любыми математическими объектами, от чисел любого типа, векторов и матриц до тензоров, от функций до интегро-дифференциальных уравнений и т. д. — они имеют общее название [C]omputer [A]lgebra [S]ystem.

Среди математического ПО для аналитических (символьных) вычислений наиболее широко известны коммерческие CAS-пакеты Maple, Mathematica и MathCAD. Для символьных вычислений предназначен пакет MatLab. Это очень мощные и очень дорогие инструменты для ученых и инженеров, позволяющие автоматизировать наиболее рутинную и требующую повышенного внимания часть работы, оперируя при этом аналитической записью данных, т. е. почти математическими формулами.

Такие программы можно назвать средой программирования, с той разницей, что в качестве элементов языка программирования выступают привычные человеку математические обозначения.

Для преподавателей, аспирантов, и студентов предоставляются академические более дешевые лицензии, но для хоббитов и коммерческого применения требуется покупка полной лицензии, имеющей

зачастую космическую стоимость. Неплохим вариантом может послужить использование бесплатного и свободного OpenSource программного обеспечения, описанного далее — пакетов **Maxima** и **Octave**.

С другой стороны, основное направление, кроме научных разработок, где такие программы востребованы — высшее образование; а использование для учебных нужд именно свободного ПО — реальная возможность и для ВУЗа, и для студентов и преподавателей иметь в своем распоряжении легальные копии такого ПО без больших, и даже каких-либо денежных затрат.

Глава 112

Пакет Maxima

Maxima – пакет CAS: символьной математики, но также включает функционал численных вычислений и визуализации.

112.1 Установка Maxima под ⊞Windows

⊞ + R » http://maxima.sourceforge.net/ru/ » Загрузка
Maxima-Windows » **5.34.1-Windows** » **maxima-5.34.1.exe**
maxima-5.34.1.exe » Установка » Язык » **English** » OK » Next
License » I accept » Next
Install folder » по умолчанию » Next
Components » ☐ Language packs » Next » Next

☒ wxMaxima ☐ XMaxima Next Install Next Finish
⊞ Maxima-5.34.1 wxMaxima ▷ Закрепить в панели задач

Дополнительная документация: http://maxima.sourceforge.net/ru/documentation.html

PDF для книги Ильина В.А., Силаев П.К. Система аналитических вычислений Maxima для физиков-теоретиков [**maxphis**] получена из файла **.ps** с помощью сервиса http://ps2pdf.com/convert.htm.

112.2 Калькулятор

Часть XIV

Куча

В этот раздел собраны все материалы, не вошедшие в основную часть потому что слишком сложны для начинающих, не попадают не в один раздел по тематике, или не вписались по каким-то другим параметрам.

Все новые материалы также сначала попадают сюда, а потом принимается решение об их переносе в основную часть.

Часто сюда пишут статьи те, кто принимает участие в создании книги эпизодически, или те, у кого нет достаточно времени заниматься их подготовкой.