

open source
hardware

Азбука ARMatурщика

ОСНОВЫ БЫТОВОЙ АВТОМАТИКИ, ЭЛЕКТРОНИКИ И СИСТЕМ УПРАВЛЕНИЯ

©

Консорциум хоббитов России

HackSpace «Чебураторный завод»

Дмитрий Понятов <dponyatov@gmail.com>

группа ruOpenWrt

Bill Collis (??)

Andreas Fester (V)

Joe Martin, Craig Libuse (38)

Оглавление

I О книге	2
От переводчика	5
1 1. Introduction to Practical Electronics	6
1.1 1.1 Your learning in Technology	6
1.2 1.2 Key Competencies from The NZ Curriculum	6
2 2 An introductory electronic circuit	7
2.1 2.1 Where to buy stuff?	8
2.2 2.2 Identifying resistors by their colour codes	8
2.3 2.3 LED's	8
2.4 2.4 Some LED Specifications	8
2.5 2.5 LED research task	8
2.6 2.6 Adding a switch to your circuit	8
2.7 2.7 Switch assignment	8
2.8 2.8 Important circuit concepts	8
2.9 2.9 Changing the value of resistance	8

2.10	2.10 Adding a transistor to your circuit	8
2.11	2.11 Understanding circuits	8
2.12	2.12 The input circuit — an LDR	8
2.13	2.13 Working darkness detector circuit	8
2.14	2.14 Protecting circuits — using a diode	8
2.15	2.15 Diode Research Task	8
2.16	2.16 Final darkness detector circuit	8
3	3 Introductory PCB construction	9
3.1	3.1 Eagle Schematic and Layout Editor Tutorial	9
3.2	3.2 An Introduction to Eagle	9
3.3	3.3 The Schematic Editor	9
3.4	3.4 The Board Editor	9
3.5	3.5 Making Negative Printouts	9
3.6	3.6 PCB Making	9
4	Soldering, solder and soldering irons	10
4.1	4.1 Soldering facts	11
4.2	4.2 Soldering Safety	11
4.3	4.3 Soldering wires to switches	11
4.4	4.4 Codes of practice	11
4.5	4.5 Good and bad solder joints	11
4.6	4.6 Short circuits	11
4.7	4.7 Soldering wires to LED's	11
5	5 Introductory Electronics Theory	12
5.1	5.1 Making electricity	13

5.2	5.2 ESD electrostatic discharge	13
5.3	5.3 Magnets, wires and motion	13
5.4	5.4 Group Power Assignment	13
5.5	5.5 Electricity supply in New Zealand	13
5.6	5.6 Conductors	13
5.7	5.7 Insulators	13
5.8	5.8 Choosing the right wire	13
5.9	5.9 Resistors	13
5.10	5.10 Resistor Assignment	13
5.11	5.11 Resistivity	13
5.12	5.12 Resistor prefixes	13
5.13	5.13 Resistor Values Exercises	13
5.14	5.14 Capacitors	13
5.15	5.15 Component symbols reference	13
5.16	5.16 Year 10/11 — Typical test questions so far	13

6 Introduction to microcontroller electronics

6.1	6.1 What is a computer?	15
6.2	6.2 What does a computer system do?	15
6.3	6.3 What does a microcontroller system do?	15
6.4	6.4 What exactly is a microcontroller?	15
6.5	6.5 Getting started with AVR Programming	15
6.6	6.6 Breadboard	15
6.7	6.7 Breadboard+Prototyping board circuit	15
6.8	6.8 Checking your workmanship	15
6.9	6.9 Getting started with Bascom & AVR	15
6.10	6.10 The compiler	15

6.11	6.11 The programmer	15
6.12	6.12 USBASP programming cable	15
6.13	6.13 Your first circuit	15
6.14	6.14 An introduction to flowcharts	15
6.15	6.15 Bascom output commands	15
6.16	6.16 Exercises	15
6.17	6.17 Two delays	15
6.18	6.18 Syntax errors -'bugs'	15
6.19	6.19 Microcontroller ports: write a Knightrider program using LED's	15
6.20	6.20 Knightrider v2	15
6.21	6.21 Knightrider v3	15
6.22	6.22 Commenting your programs	15
6.23	6.23 Learning review	15
6.24	6.24 What is a piezo and how does it make sound?	15
6.25	6.25 Sounding Off	15
6.26	6.26 Sound exercises	15
6.27	6.27 Amp it up	15

7	7 Microcontroller input circuits	16
7.1	7.1 Single push button switch	17
7.2	7.2 Pullup resistor theory	17
7.3	7.3 Switch in a breadboard circuit	17
7.4	7.4 Checking switches in your program	17
7.5	7.5 Program Logic – the 'If-Then' Switch Test	17
7.6	7.6 If-then exercises	17
7.7	7.7 Switch contact bounce	17
7.8	7.8 Reading multiple switches	17

7.9	7.9 Bascom debounce command	17
7.10	7.10 Different types of switches you can use	17
7.11	7.11 Reflective opto switch	17
8	8 Programming Review	18
8.1	8.1 Three steps to help you write good programs	19
8.2	8.2 Saving Programs	19
8.3	8.3 Organisation is everything	19
8.4	8.4 Programming template	19
8.5	8.5 What you do when learning to program	19
8.6	8.6 AVR microcontroller hardware	19
8.7	8.7 Power supplies	19
8.8	8.8 Programming words you need to be able to use correctly	19
8.9	8.9 Year10/11 typical test questions so far	19
9	9 Introduction to program flow	20
9.1	9.1 Pedestrian crossing lights controller	21
9.2	9.2 Pedestrian Crossing Lights schematic	21
9.3	9.3 Pedestrian Crossing Lights PCB Layout	21
9.4	9.4 Algorithm planning example – pedestrian crossing lights	21
9.5	9.5 Flowchart planning example – pedestrian crossing lights	21
9.6	9.6 Getting started code	21
9.7	9.7 Modification exercise for the pedestrian crossing	21
9.8	9.8 Traffic lights program flow	21
10	10 Introductory programming - using subroutines	22
10.1	10.1 Sending Morse code	22

10.2	10.2 LM386 audio amplifier PCB	22
10.3	10.3 LM386 PCB Layout	22
11	11 Introductory programming – using variables	23
11.1	11.1 Stepping or counting using variables	24
11.2	11.2 For-Next	24
11.3	11.3 Siren sound - programming using variables	24
11.4	11.4 Make a simple siren	24
11.5	11.5 Siren exercise	24
11.6	11.6 A note about layout of program code	24
11.7	11.7 Using variables for data	24
11.8	11.8 Different types of variables	24
11.9	11.9 Variables and their uses	24
11.10	11.10 Vehicle counter	24
11.11	11.11 Rules about variables	24
11.12	11.12 Examples of variables in use	24
11.13	11.13 Byte variable limitations	24
11.14	11.14 Random Numbers	24
11.15	11.15 The Bascom-AVR simulator	24
11.16	11.16 Electronic dice project	24
11.17	11.17 Programming using variables – dice	24
11.18	11.18 Dice layout stage 1	24
11.19	11.19 Dice layout stage 2	24
11.20	11.20 Dice Layout final	24
11.21	11.21 First Dice Program flowchart	24
11.22	11.22 A note about the Bascom Rnd command	24
11.23	11.23 Modified dice	24

11.24	11.24 Modified Knightrider	24
12	12 Basic displays	25
12.1	12.1 7 segment displays	25
12.2	12.2 Alphanumeric LED displays	25
13	13 TDA2822M Portable Audio Amplifier Project	26
13.1	13.1 Portfolio Assessment Schedule	27
13.2	13.2 Initial One Page Brief	27
13.3	13.3 TDA2822M specifications	27
13.4	13.4 Making a PCB for the TDA2822 Amp Project	27
13.5	13.5 Extra PCB making information	27
13.6	13.6 Component Forming Codes of Practice	27
13.7	13.7 TDA2811 wiring diagram	27
13.8	13.8 SKETCHUP Quick Start Tutorial	27
13.9	13.9 Creating reusable components in SketchUp	27
14	14 Basic programming logic	28
14.1	14.1 Quiz Game Controller	29
14.2	14.2 Quiz game controller system context diagram	29
14.3	14.3 Quiz game controller block diagram	29
14.4	14.4 Quiz game controller Algorithm	29
14.5	14.5 Quiz game schematic	29
14.6	14.6 Quiz game board veroboard layout	29
14.7	14.7 Quiz Controller flowchart	29
14.8	14.8 'Quiz Controller program code	29
14.9	14.9 Don't delay - use logic	29

15 15 Algorithm development – an alarm system	30
15.1 15.1 Simple alarm system – stage 1	31
15.2 15.2 Alarm System Schematic	31
15.3 15.3 A simple alarm system – stage 2	31
15.4 15.4 A simple alarm system – stage 3	31
15.5 15.5 A simple alarm system – stage 4	31
15.6 15.6 More complex alarm system	31
15.7 15.7 Alarm unit algorithm 5	31
15.8 15.8 Alarm 6 algorithm	31
16 16 Basic DC circuit theory	32
16.1 16.1 Conventional Current	33
16.2 16.2 Ground	33
16.3 16.3 Preferred resistor values	33
16.4 16.4 Resistor Tolerances	33
16.5 16.5 Combining resistors in series	33
16.6 16.6 Combining resistors in parallel	33
16.7 16.7 Resistor Combination Circuits	33
16.8 16.8 Multimeters	33
16.9 16.9 Multimeter controls	33
16.10 16.10 Choosing correct meter settings	33
16.11 16.11 Ohms law	33
16.12 16.12 Voltage & Current Measurements	33
16.13 16.13	33
16.14 16.14 Continuity	33
16.15 16.15 Variable Resistors	33
16.16 16.16 Capacitors	33

16.17	16.17 Capacitor Codes and Values	33
16.18	16.18 Converting Capacitor Values uF, nF , pF	33
16.19	16.19 Capacitor action in DC circuits	33
16.20	16.20 The Voltage Divider	33
16.21	16.21 Using semiconductors	33
16.22	16.22 Calculating current limit resistors for an LED	33
16.23	16.23 The Bipolar Junction Transistor	33
16.24	16.24 Transistor Specifications Assignment	33
16.25	16.25 Transistor Case styles	33
16.26	16.26 Transistor amplifier in a microcontroller circuit	33
16.27	16.27 Transistor Audio Amplifier	33
16.28	16.28 Speakers	33
16.29	16.29 Switch types and symbols	33
17	17 Basic project planning	34
17.1	17.1 System Designer	35
17.2	17.2 Project mind map	35
17.3	17.3 Project timeline	35
17.4	17.4 System context diagram	35
17.5	17.5 Block Diagram	35
17.6	17.6 Board Layouts	35
17.7	17.7 Algorithm design	35
17.8	17.8 Flowcharts	35
18	18 Example system design - hot glue gun timer	36
18.1	18.1 System context diagram	36
18.2	18.2 Hot glue gun timer block diagram	36

18.3	18.3 Hot glue gun timer algorithm	36
18.4	18.4 Hot glue gun timer flowchart	36
18.5	18.5 Hot glue gun timer program	36
19	19 Basic interfaces and their programming	37
19.1	19.1 Parallel data communications	38
19.2	19.2 LCDs (liquid crystal displays)	38
19.3	19.3 Alphanumeric LCDs	38
19.4	19.4 ATTINY461 Development PCB with LCD	38
19.5	19.5 Completing the wiring for the LCD	38
19.6	19.6 LCD Contrast Control	38
19.7	19.7 Learning to use the LCD	38
19.8	19.8 Repetition again - the 'For-Next' and the LCD	38
19.9	19.9 LCD Exercises	38
19.10	19.10 Defining your own LCD characters	38
19.11	19.11 LCD custom character program	38
19.12	19.12 A simple digital clock	38
19.13	19.13 Adding more interfaces to the ATTiny461 Development board	38
19.14	19.14 Ohms law in action – a multicoloured LED	38
II	Рабочая среда разработчика встраиваемых систем	39
19.15	Операционная система с набором типовых утилит	40
19.16	САПР электронных устройств (EDA CAD)	40
19.17	Пакет расчета и симуляции электронных схем: SPICE	40
19.18	САПР общего назначения	40
19.19	Система управления версиями: VCS	40

19.20 Текстовый редактор или интегрированная среда разработки (IDE)	41
19.21 ПО для программатора, JTAG-адаптера	41
19.22 Симулятор для отладки программ без железа	41
19.23 Система верстки документации	42
III Пакет компиляторов и утилит GNU toolchain	43
20 Make	45
21 binutils	46
22 Формат объектного файла GNU ELF	47
23 Ассемблер GNU as	49
24 Линкер GNU ld	50
24.1 Использование ld	51
24.2 Обзор	51
24.3 Вызов	51
24.3.1 Параметры командной строки	51
24.3.2 Опции, специфичные для целевой платформы i386 РЕ	51
24.3.3 Переменные среды	51
24.4 Скрипты линкера	51
24.5 Основные понятия скрипта линкера	51
24.6 Формат скрипта компоновщика	51
24.7 Пример простого скрипта компоновщика	51
24.8 Команды простого скрипта компоновщика	51

24.8.1 Установка точки входа	51
24.8.2 Команды работы с файлами	51
24.8.3 Работа с форматами объектный файлов	51
24.8.4 Другие команды сценария компоновщика	51
24.9 Присвоение значений символам	51
24.9.1 Простые назначения	51
24.9.2 PROVIDE	51
24.10 Команда SECTIONS	51
24.10.1 Описание выходных секций	53
24.10.2 Имя выходной секции	53
24.10.3 Описание выходных секций	56
24.10.4 Описание входных секций	56
24.10.5 Выходная секция данных	56
24.10.6 Ключевые слова выходных секций	56
24.10.7 Отброс выходных секций	56
24.10.8 Атрибуты выходных секций	56
24.10.9 Описание оверлея	56
24.11 Команда MEMORY	56
24.12 Команда PHDRS	56
24.13 Команда VERSION	56
24.14 Выражения в скриптах компоновщика	56
24.14.1 Константы	56
24.14.2 Имена символов	56
24.14.3 Счетчик адреса	56
24.14.4 Операторы	56
24.14.5 Вычисления	56
24.14.6 Часть выражения	56

24.14.7 Встроенные функции	56
24.15 Неявные скрипты компоновщика	56
24.16 Машинно-зависимые особенности	56
24.16.1 ld и H8/300	56
24.16.2 ld и семейство Intel 960	56
24.16.3 поддержка взаимодействия между кодом в режимах ARM и Thumb	56
24.16.4 ld и поддержка 32-разрядных ELF HPPA	56
24.16.5 ld и MMIX	56
24.16.6 ld и MSP430	56
24.16.7 поддержка различных версий TI COFF	56
24.16.8 ld и WIN32 (cygwin/mingw)	56
24.16.9 ld и процессоры Xtensa	56
24.17 BFD	56
24.17.1 Как это работает: очерк о BFD	56
24.17.2 Потеря информации	56
24.17.3 Канонический формат объектных файлов BFD	56
25 objdump	57
26 GNU Compiler Collection (GCC)	58
27 Компилятор GNU C	59
28 Компилятор GNU C++	60
29 Компилятор GNU Fortran	61

IV САПР электронных устройств KiCAD	62
29.1 Установка под Windows	64
29.2 Установка под Linux	64
29.3 Маршрут проектирования	66
29.4 Создание проекта в менеджере проектов kicad	67
29.5 Создание принципиальной схемы в eeschema (часть 1)	69
29.6 eeschema : редактор электрических схем	72
V Расчет схем и моделирование в ngSPICE	73
30 Доступные SPICE-пакеты	75
30.1 Установка ngSPICE под Windows	76
30.2 Установка LT-SPICE (только под Windows)	76
30.3 Установка ngSPICE под Linux	77
31 Пошаговый пример использования	78
31.1 Рисуем схему в KiCAD	78
31.2 Создание списка цепей	80
31.3 Запуск симуляции	82
31.4 Просмотр результата расчета	84
31.5 Расчет АЧХ по переменному току (AC симуляция)	88
31.6 Симуляция полноволнового выпрямителя	92
32 Настройка KiCAD для SPICE-моделирования	95
32.1 Библиотеки компонентов со SPICE-элементами	95
32.2 Настройка проекта	96

32.3 Как это работает	96
VI Разработка конструкции в САПР FreeCAD	98
33 Установка	101
33.1 Windows	101
33.2 Linux	102
VII Инструменты и электронное оборудование	103
34 Радиомонтажный инструмент	104
34.1 Pro'sKit	104
34.2 Инструмент до 1000 В	109
34.3 Хранение	110
34.4 Радиомонтаж	111
34.5 Прочие	115
35 Паяльное оборудование	116
35.1 Паяльник	116
35.2 Паяльная станция	118
36 Измерительное оборудование	121
36.1 Мультиметр	121
36.1.1 Mastech M838	122
36.1.2 Mastech M300	123
36.1.3 Mastech M320	124

36.2 Осциллограф	125
36.3 Логический анализатор	125
36.4 Генератор сигналов	125
36.5 Рыльцеметр RLC	125
37 Электроинструмент	126
37.1 Дрель	127
37.2 Лобзик	130
37.3 Жигатель	131
VIII Станочное оборудование	134
38 Настольные станки	137
39 Самодельная оснастка	138
40 Промышленные станки	139
40.1 1А616: станок токарно-винторезный	140
40.1.1 Назначение и области применения	141
40.1.2 Распаковка и транспортировка	141
40.1.3 Фундамент станка, монтаж и установка	141
40.1.4 Подготовка станка к первоначальному пуску	141
40.1.5 Паспортные данные	141
40.1.6 Описание основных узлов	141
40.1.7 Смазка	141
40.1.8 Первоначальный пуск	141
40.1.9 Указания по технике безопасности	141

40.1.10 Настройка	141
40.1.11 Регулирование	141
40.1.12 Ведомость комплектации	141

IX Разработка ПО для встраиваемых систем 142

41 IDE 143

41.1  ECLIPSE	146
41.1.1 Установка  ECLIPSE под Windows	146
41.1.2 Установка  ECLIPSE под Linux	146
41.1.3 Установка CDT	147
41.1.4 Установка PyDev	148
41.1.5 Установка TeXlipse	148
41.1.6 Редактирование файлов в формате XML и производных	149
41.1.7 Проверка орфографии	149
41.2 Code::Blocks	152
41.3 (g)Vim	152
41.3.1 Установка под Windows	152
41.3.2 Выход из (g)Vim	154
41.3.3 Выход с автосохранением	155
41.3.4 Переход в режим редактирования	155
41.3.5 Переход в режим команд	155
41.3.6 Запись редактируемого файла	155
41.3.7 Перезагрузка файла	156
41.3.8 Отмена последних изменений (undo)	156

42 Make: управление сборкой проектов	157
43 VCS: системы контроля версий	158
43.1 CVS	158
43.2 Subversion	158
43.3 Git	158
43.3.1 GitHub	158
44 Вспомогательные скрипты на языке Python	159
45 Основы Си и C ₊ ⁺	160
45.1 Установка MinGW (win32)	160
45.2 Особенности C ₊ ⁺ в embedded	160
45.3 Сборка кросскомпилятора GNU toolchain	160
46 Лексический и синтаксический анализ	161
46.1 Лексер и лексический анализ, утилита flex	162
46.2 Генератор синтаксических анализаторов bison	166
47 LLVM и разработка собственных компиляторов	169
47.1 Установка под Windows	169
47.2 Создание компилятора с помощью инфраструктуры LLVM	169
47.3 Инструменты llc и lli	170
47.4 Семантический анализ	171
47.5 Оптимизация	171
47.6 Кодогенерация	171
47.7 Транслятор Паскаля	172

X Микроконтроллеры Cortex-Mx

173

48 Производители

48.1 ST Microelectronix STM32	174
48.2 LPC	174
48.3 Миландр	174

49 Отладочные платы

49.1 LeafLabs Maple Mini: STM32F103 /Cortex-M3/	175
49.2 Серия STM32 STM32DISCOVERY	175
49.2.1 STM32DISCOVERY: STM32F103 /Cortex-M3/	175
49.2.2 STM32F4DISCOVERY: STM32F406 /Cortex-M4F/	175
49.2.3 STM32F0DISCOVERY: STM32F040 /Cortex-M0/	175

XI ПЛИС

176

XII USB

178

50 Стек протоколов USB

181

51 libUSB

51.1 драйвер для устройства USB	183
---	-----

52 Поддержка USB в Linux

184

52.1 Опции ядра	185
52.1.1 режимы host/client/otg и хост-контроллеры xHCI	185

52.1.2	data storage: носители данных	185
52.1.3	hid: клавиатура, мышь, джойстик	185
52.1.4	USB-периферия: сеть, звук,	185
52.2	Настройка hotplug и автомонтирования USB носителей	185
52.3	Сборка и настройка libusb	185
52.4	Примеры программ низкоуровневого ввода/вывода	185
XIII	Встраиваемый emLinux	186
53	Загрузчик syslinux	188
53.1	Закачка	188
53.2	Установка под Windows на флешку	189
53.3	syslinux.cfg	190
54	azLinux	193
54.1	Требования к системе сборки (BUILD -хост)	195
54.2	Понятие пакет	196
54.3	Клонирование проекта azLinux	198
54.4	Общий порядок сборки	199
54.5	Фиксация переменных	200
54.6	dirs : Создание дерева каталогов	200
54.7	gz : Загрузка архивов исходников	204
54.8	APP : Приложение	206
54.9	HW : Поддерживаемое железо	208
54.10	i386	209
54.10.1	qemu386: эмулятор QEMU	209

54.10.2 <code>eeepc701</code> : ASUS Eee PC 701	211
54.10.3 <code>gac1037</code> : Gigabyte GA-C1037UN-EU rev.2	212
54.11 ARM	213
54.11.1 <code>qemuARM</code> : эмулятор QEMU	213
54.11.2 <code>cubie1</code> : Cubie Board v.1	214
54.11.3 <code>rpi</code> : Raspberry Pi model B	214
54.11.4 <code>tion</code> : ТионПро270	215
54.11.5 <code>mb77</code> : Микрокомпьютер MB77.07 на базе СБИС K1879ХБ1Я	216
54.12 MIPS	217
54.12.1 <code>qemuMIPS</code> : эмулятор QEMU	217
54.12.2 <code>mr3020</code> : роутер MR3020	218
54.12.3 <code>vocore</code> : VoCore	219
54.12.4 <code>bswift</code> : BlackSwift	219
54.13 CPU: Конфигурации процессоров	220
54.13.1 <code>i386</code>	220
54.13.2 ARM	221
54.13.3 MIPS	221
54.14 Пакеты	221
54.14.1 <code>versions.mk</code> : Версии пакетов	221
54.14.2 <code>tc</code> : кросс-компилятор	221
54.14.3 <code>core</code> : ядро	222
54.14.4 <code>boot</code> : загрузчики	222
54.14.5 <code>libs</code> : библиотеки	222
54.14.6 <code>tc</code> : сборка кросс-компилятора	223
54.14.7 <code>binutils</code> : ассемблер, линкер и утилиты	226
54.14.8 <code>cclibs</code> : библиотеки для сборки <code>gcc</code>	227
54.14.9 <code>gcc0</code> : сборка минимального кросс-компилятора Си	228

54.14.10 gcc : пересборка полного кросс-компилятора Cи/C ⁺	228
54.14.11 core : сборка основной системы	228
54.14.12 kernel : ядро Linux	228
54.14.13 libc : библиотека uClibc	235
54.14.14 gcc : пересборка полного gcc	241
54.14.15 busybox : набор утилит busybox	241
54.14.16 libs : сборка библиотек \${LIBS}	242
54.14.17 apps : сборка прикладных пакетов \${APPS}	242
54.14.18 user : сборка пользовательского кода	242
54.14.19 root : формирование корневой файловой системы	242
54.14.20 boot : сборка загрузчика syslinux/grub/uboot	245
54.14.21 syslinux	246
54.14.22 grub	246
54.14.23 uboot	246
54.14.24 emu : запуск собранной системы в эмуляторе	246
54.15 netboot : Сетевая загрузка	249
54.16 Прошивка на устройство	249
54.17 RT-патч	249
54.18 SDK: расширения для on-board разработки	249
54.18.1 canadian : сборка binutils канадским крестом	250
54.18.2 binhost : binutils для хост-процессора	252
54.18.3 binavr8 : binutils для Atmel ATmega (AVR8)	252
54.18.4 bincmx : binutils для ARM Cortex-Mx	252
54.18.5 fpc : FreePascal	252
54.18.6 python : интерпретатор Python	254
54.18.7 gcl : GNU Common Lisp	254
54.18.8 maxima : система символьной математики Maxima	254

56 Библиотека libSDL

56.1 Инициализация и завершение SDL-программы	258
56.2 LazyFoo tutorial	258
56.2.1 Setting up SDL and Getting an Image on the Screen	258
56.2.2 Optimized Surface Loading and Blitting	264
56.2.3 Extension Libraries and Loading Other Image Formats	272
56.2.4 Event Driven Programming	273
56.2.5 Color Keying	282
56.2.6 Clip Blitting and Sprite Sheets	282
56.2.7 True Type Fonts	282
56.2.8 Key Presses	282
56.2.9 Mouse Events	282
56.2.10 Key States	282
56.2.11 Playing Sounds	282
56.2.12 Timing	282
56.2.13 Advanced Timers	282
56.2.14 Regulating Frame Rate	282
56.2.15 Calculating Frame Rate	282
56.2.16 Motion	282
56.2.17 Collision Detection	282
56.2.18 Per-pixel Collision Detection	282
56.2.19 Circular Collision Detection	282
56.2.20 Animation	282
56.2.21 Scrolling	282
56.2.22 Scrolling Backgrounds	282

56.2.23 Getting String Input	282
56.2.24 Game Saves	282
56.2.25 Joysticks	282
56.2.26 Resizable Windows and Window Events	282
56.2.27 Alpha Blending	282
56.2.28 Particle Engines	282
56.2.29 Tiling	282
56.2.30 Bitmap Fonts	282
56.2.31 Pixel Manipulation and Surface Flipping	282
56.2.32 Frame Independent Movement	282
56.2.33 Multithreading	282
56.2.34 Semaphores	282
56.2.35 Mutexes and Conditions	282
56.2.36 Using OpenGL with SDL	282
56.3 Примеры программ	282
56.3.1 Графический Hello World	283
56.3.2 Вывод случайных прямоугольников	285
56.4 Версия SDL2	286
57 BuildRoot	287
58 Особенности OpenWrt	288

для компьютера на i386 процессоре	289
59 Ресурсы	290
60 Структура	292
61 Процесс запуска	293
62 Сборка кросс-компилятора	294
63 Формат ELF32	300
64 multiboot	302
65 Микроядро	305
66 Драйвера	307
66.1 vga : текстовая консоль VGA 80×25	307
66.2 kbd : клавиатура	310
66.3 ide : жесткий диск IDE	310
66.3.1 fatfs : файловая система FAT16	310
XV Символьная и численная математика	311
67 Общие сведения о компьютерной математике	313
68 Пакет Maxima	316
68.1 Установка Maxima под Windows	316

68.2 Калькулятор	317
XVI Подготовка технической документации	318
69 Верстка в \LaTeX	319
70 Оформление листингов	320
71 Подготовка иллюстраций	321
71.1 GIMP	321
71.2 Inkscape	321
71.3 Graphviz	321
72 Замечания для соавторов “Абзуки ARМатурщика”	322
XVII Примерные учебные планы	323
73 Блондинко	324
74 Школотрон	325
75 Студень	326
76 Технический специалист	327

XVIII Куча	328
77 Автоматное программирование /фреймворк QuantumLeaps/	330
78 Запуск Linux на android устройстве в режиме паразита	331
Предметный указатель	344

Часть I

О книге

В текущем состоянии эта книга — конспект материалов, которые я сейчас собираю, в черновой верстке. Объем материала очень большой, фактически это целая специальность для приличного техникума, что-то типа “Технология цифрового производства”. Поэтому 146% пока составляет сырья копипаста, с редкими вкраплениями собственного бреда. В процессе адаптации, обкатки на студентах и доработки эта поделка должна принять более вменяемый вид. Но учитывая полное отсутствие обратной связи, этого никогда не случиться.

Это учебное пособие было создано для интересующихся любительской электроникой, самодельными цифровыми системами управления (Arduino, устройствами на микроконтроллерах и т.п.), и программистов-любителей. В связи с полной деградацией системы образования пособие также рекомендуется для применения при обучении в ВУЗах по специализациям, связанным с применением цифровой электроники и компьютерной техники.

Большой упор был сделан на использование открытого некоммерческого программного обеспечения, для удешевления учебного процесса, уменьшения себестоимости ваших проектов¹, и стимулирования вашего участия в развитии этих программных пакетов.

Книга очень объемна и разнообразна по материалу, и построена как справочник с группировкой материала по тематике. Для тех, кто только начинает, в разделе XVII расписаны **пошаговые учебные планы** с точки зрения параллельного изучения нескольких предметов с постепенным усложнением². Как известно, главная часть любого обучения — практическая. Особое внимание уделено набору лабораторных работ.

В качестве видеоматериала были использованы видеоуроки физики
© Ерюткин Е.С., учитель физики высшей категории ГБОУ СОШ №1360, г.Москва

Мы признательны Bill Collis за разрешение использовать материалы его книги «An Introduction to Practical Electronics, Microcontrollers and Software Design» [?] в русскоязычном варианте «Азбуки» (??), 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

² как это происходит при традиционном offline обучении

Так как для работы в области электроники необходимо владение технологиями изготовления конструктива, в книгу включен соответствующий раздел. Эти книги рекомендуются популярным поставщиком хоббийных настольных микро-станков Sherline Products. Так как от владельцев авторских прав не получено разрешение на полный официальный перевод, для этих книг сделан только перевод-подстрочник, который поможет вам читать оригинал:

- Joe Martin, Craig Libuse **Tabletop Machining** [?] (38)
- Doug Briney **Home Machinists Handbook** [?] (??)

Отечественных книг по использованию маленьких “часовых” и настольных станков просто не существует, хотя они и выпускались серийно. Исключение — книга Евгений Васильев **Маленькие станки** [?], но она имеет обзорный характер.

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: <https://github.com/ponyatov/Azбука>

Приглашаем всех желающих участвовать в развитии этого учебного пособия на форум ruOpenWrt и в группу <http://vk.com/samarahackerspace>, нам нужна обратная связь по качеству материала, результаты тестирования на вас или ваших студентах, дополнения и замечания.

От переводчика

Эта часть основана на переводе книги:

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. Introduction to Practical Electronics

1.1 1.1 Your learning in Technology

1.2 1.2 Key Competencies from The NZ Curriculum

Глава 2

2 An introductory electronic circuit

2.1 2.1 Where to buy stuff?

2.2 2.2 Identifying resistors by their colour codes

2.3 2.3 LED's

2.4 2.4 Some LED Specifications

2.5 2.5 LED research task

2.6 2.6 Adding a switch to your circuit

2.7 2.7 Switch assignment

Глава 3

3 Introductory PCB construction

3.1 3.1 Eagle Schematic and Layout Editor Tutorial

3.2 3.2 An Introduction to Eagle

3.3 3.3 The Schematic Editor

3.4 3.4 The Board Editor

3.5 3.5 Making Negative Printouts

3.6 3.6 PCB Making

Глава 4

Soldering, solder and soldering irons

4.1 4.1 Soldering facts

4.2 4.2 Soldering Safety

4.3 4.3 Soldering wires to switches

4.4 4.4 Codes of practice

4.5 4.5 Good and bad solder joints

4.6 4.6 Short circuits

4.7 4.7 Soldering wires to LED's

Глава 5

5 Introductory Electronics Theory

5.1 5.1 Making electricity

5.2 5.2 ESD electrostatic discharge

5.3 5.3 Magnets, wires and motion

5.4 5.4 Group Power Assignment

5.5 5.5 Electricity supply in New Zealand

5.6 5.6 Conductors

5.7 5.7 Insulators

Глава 6

Introduction to microcontroller electronics

6.1 6.1 What is a computer?

6.2 6.2 What does a computer system do?

6.3 6.3 What does a microcontroller system do?

6.4 6.4 What exactly is a microcontroller?

6.5 6.5 Getting started with AVR Programming

6.6 6.6 Breadboard

6.7 6.7 Breadboard+Prototyping board circuit

Глава 7

7 Microcontroller input circuits

7.1 7.1 Single push button switch

7.2 7.2 Pullup resistor theory

7.3 7.3 Switch in a breadboard circuit

7.4 7.4 Checking switches in your program

7.5 7.5 Program Logic – the ‘If-Then’ Switch Test

7.6 7.6 If-then exercises

7.7 7.7 Switch contact bounce

Глава 8

8 Programming Review

8.1 8.1 Three steps to help you write good programs

8.2 8.2 Saving Programs

8.3 8.3 Organisation is everything

8.4 8.4 Programming template

8.5 8.5 What you do when learning to program

8.6 8.6 AVR microcontroller hardware

8.7 8.7 Power supplies

Глава 9

9 Introduction to program flow

- 9.1 9.1 Pedestrian crossing lights controller
- 9.2 9.2 Pedestrian Crossing Lights schematic
- 9.3 9.3 Pedestrian Crossing Lights PCB Layout
- 9.4 9.4 Algorithm planning example – pedestrian crossing lights
- 9.5 9.5 Flowchart planning example – pedestrian crossing lights
- 9.6 9.6 Getting started code
- 9.7 9.7 Modification exercise for the pedestrian crossing

Глава 10

10 Introductory programming - using subroutines

10.1 10.1 Sending Morse code

10.2 10.2 LM386 audio amplifier PCB

10.3 10.3 LM386 PCB Layout

Глава 11

11 Introductory programming – using variables

11.1 11.1 Stepping or counting using variables

11.2 11.2 For-Next

11.3 11.3 Siren sound - programming using variables

11.4 11.4 Make a simple siren

11.5 11.5 Siren exercise

11.6 11.6 A note about layout of program code

11.7 11.7 Using variables for data

Глава 12

12 Basic displays

12.1 12.1 7 segment displays

12.2 12.2 Alphanumeric LED displays

Глава 13

13 TDA2822M Portable Audio Amplifier Project

13.1 13.1 Portfolio Assessment Schedule

13.2 13.2 Initial One Page Brief

13.3 13.3 TDA2822M specifications

13.4 13.4 Making a PCB for the TDA2822 Amp Project

13.5 13.5 Extra PCB making information

13.6 13.6 Component Forming Codes of Practice

Глава 14

14 Basic programming logic

14.1 14.1 Quiz Game Controller

14.2 14.2 Quiz game controller system context diagram

14.3 14.3 Quiz game controller block diagram

14.4 14.4 Quiz game controller Algorithm

14.5 14.5 Quiz game schematic

14.6 14.6 Quiz game board veroboard layout

14.7 14.7 Quiz Controller flowchart

Глава 15

15 Algorithm development – an alarm system

15.1 15.1 Simple alarm system – stage 1

15.2 15.2 Alarm System Schematic

15.3 15.3 A simple alarm system – stage 2

15.4 15.4 A simple alarm system – stage 3

15.5 15.5 A simple alarm system – stage 4

15.6 15.6 More complex alarm system

15.7 15.7 Alarm unit algorithm 5

Глава 16

16 Basic DC circuit theory

16.1 16.1 Conventional Current

16.2 16.2 Ground

16.3 16.3 Preferred resistor values

16.4 16.4 Resistor Tolerances

16.5 16.5 Combining resistors in series

16.6 16.6 Combining resistors in parallel

16.7 16.7 Resistor Combination Circuits

Глава 17

17 Basic project planning

17.1 17.1 System Designer

17.2 17.2 Project mind map

17.3 17.3 Project timeline

17.4 17.4 System context diagram

17.5 17.5 Block Diagram

17.6 17.6 Board Layouts

17.7 17.7 Algorithm design

Глава 18

18 Example system design - hot glue gun timer

18.1 18.1 System context diagram

18.2 18.2 Hot glue gun timer block diagram

18.3 18.3 Hot glue gun timer algorithm

18.4 18.4 Hot glue gun timer flowchart

18.5 18.5 Hot glue gun timer program

Глава 19

19 Basic interfaces and their programming

19.1 19.1 Parallel data communications

19.2 19.2 LCDs (liquid crystal displays)

19.3 19.3 Alphanumeric LCDs

19.4 19.4 ATTINY461 Development PCB with LCD

19.5 19.5 Completing the wiring for the LCD

19.6 19.6 LCD Contrast Control

19.7 19.7 Learning to use the LCD

Часть II

Рабочая среда разработчика встраиваемых
систем

19.15 Операционная система с набором типовых утилит

среда для запуска рабочих программ, просмотра электронной документации, поиска информации в Internet, запуска ПО поставляемого с измерительной аппаратурой (цифровые осциллографы, генераторы сигналов, логические и сигнальные анализаторы)

19.16 САПР электронных устройств (EDA CAD)

используется для разработки схем, моделирования работы устройства, разводки печатных плат (ПП) и межплатных соединителей, и подготовки технологических файлов для производства ПП

19.17 Пакет расчета и симуляции электронных схем: SPICE

выполняется симуляция работы схем, расчет рабочих режимов, подбираются номиналы элементов, и моделируется работа аналоговой части устройств

19.18 САПР общего назначения

создаются модели и чертежи конструкции устройств, прорабатывается компоновка, и проверяется работа электро-механических узлов

19.19 Система управления версиями: VCS

VCS предназначены для хранения полной истории изменений файлов проекта, и позволяют получить выгрузку проекта на любой момент времени, вести несколько веток разработки, получить историю изменений

конкретного файла, или сравнить две версии файла ([diff](#))

19.20 Текстовый редактор или интегрированная среда разработки (IDE)

редактирование текстов программ и скриптов сборки (компиляции) с цветовой подсветкой синтаксиса (в зависимости от языка файла), [автодополнением](#) и вызовом программ-утилит нажатием сочетаний клавиш. Также включает различные вспомогательные функции, например отладочный интерфейс и отображение объектов программ.

19.21 ПО для программатора, JTAG-адаптера

загрузка полученной прошивки в целевое устройство, редактирование памяти, внутрисхемная отладка в процессе работы устройства, прямое изменение сигналов на выводах процессора (граничное сканирование и тестирование железа).

19.22 Симулятор для отладки программ без железа

может использоваться как ограниченная замена реального железа для начального обучения, и для отладки программ, не связанных на работу железа.

19.23 Система верстки документации

Для документирования проектов и написания руководств нужна система верстки документации, выполняющая трансляцию текстов программ и файлов документации в выходной формат, чаще всего `.pdf` и `.html`.

Часть III

Пакет компиляторов и утилит GNU
toolchain

Пакет кросс-компилятора, ассемблера, линкера и других утилит типа make, objdump... для получения прошивок из исходных текстов программ.

Глава 20

Make

Глава 21

binutils

Глава 22

Формат объектного файла GNU ELF

формат файла elf32-i386

архитектура i386

HAS_SYMS в файл включена отладочная информация
об идентификаторах (“символах”)

start address 0x00100000 стартовый адрес загрузки ядра 1 Мб

Бинарный код делится на **секции**, или **сегменты**:

.text машинный код программы

.rodata данные: константы

.eh_frame

.data данные: инициализированные массивы, строки

.bss данные: пустые массивы под которые выделяется память при старте

.comment

CONTENTS

ALLOC

LOAD

READONLY

CODE

DATA

Глава 23

Ассемблер GNU `as`

Глава 24

Линкер GNU ld

24.1 Использование ld

24.2 Обзор

24.3 Вызов

24.3.1 Параметры командной строки

24.3.2 Опции, специфичные для целевой платформы i386 PE

24.3.3 Переменные среды

24.4 Скрипты линкера

The format of the SECTIONS command is:

```
SECTIONS
{
    sections-command
    sections-command
    ...
}
```

Each sections-command may of be one of the following:

- an ENTRY command (refer to Section 4.4.1 Setting the Entry Point)
- a symbol assignment (refer to Section 4.5 Assigning Values to Symbols)
- an output section description
- an overlay description

The ENTRY command and symbol assignments are permitted inside the SECTIONS command for convenience in using the location counter in those commands. This can also make the linker script easier to understand because you can use those commands at meaningful points in the layout of the output file.

Output section descriptions and overlay descriptions are described below.

If you do not use a SECTIONS command in your linker script, the linker will place each input section into an identically named output section in the order that the sections are first encountered in the input files. If all input sections are present in the first file, for example, the order of sections in the output file will match the order in the first input file. The first section will be at address zero.

24.10.1 Описание выходных секций

The full description of an output section looks like this:

```
section [address] [(type)] : [AT(lma)]
{
    output-section-command
    output-section-command
    ...
} [>region] [AT>lma_region] [:phdr :phdr ...] [=fillexp]
```

Most output sections do not use most of the optional section attributes.

The whitespace around section is required, so that the section name is unambiguous. The colon and the curly braces are also required. The line breaks and other white space are optional.

Each output-section-command may be one of the following:

- a symbol assignment (refer to Section 4.5 Assigning Values to Symbols)
- an input section description (refer to Section 4.6.4 Input Section Description)
- data values to include directly (refer to Section 4.6.5 Output Section Data)
- a special output section keyword (refer to Section 4.6.6 Output Section Keywords)

24.10.2 Имя выходной секции

The name of the output section is section. section must meet the constraints of your output format. In formats which only support a limited number of sections, such as a.out, the name must be one of the names supported by the format (a.out, for example, allows only .text, .data or .bss). If the output format supports any number of sections,

but with numbers and not names (as is the case for Oasys), the name should be supplied as a quoted numeric string. A section name may consist of any sequence of characters, but a name which contains any unusual characters such as commas must be quoted.

The output section name /DISCARD/ is special; refer to Section 4.6.7 Output Section Discarding.

- 24.10.3 Описание выходных секций
- 24.10.4 Описание входных секций
- 24.10.5 Выходная секция данных
- 24.10.6 Ключевые слова выходных секций
- 24.10.7 Отброс выходных секций
- 24.10.8 Атрибуты выходных секций
- 24.10.9 Описание оверлея
- 24.11 Команда MEMORY
- 24.12 Команда PHDRS
- 24.13 Команда VERSION
- 24.14 Выражения в скриптах компоновщика
 - 24.14.1 Константы
 - 24.14.2 Имена символов
 - 24.14.3 Счетчик адреса
 - 24.14.4 Операторы

Глава 25

objdump

Глава 26

GNU Compiler Collection (GCC)

Глава 27

Компилятор GNU C

Глава 28

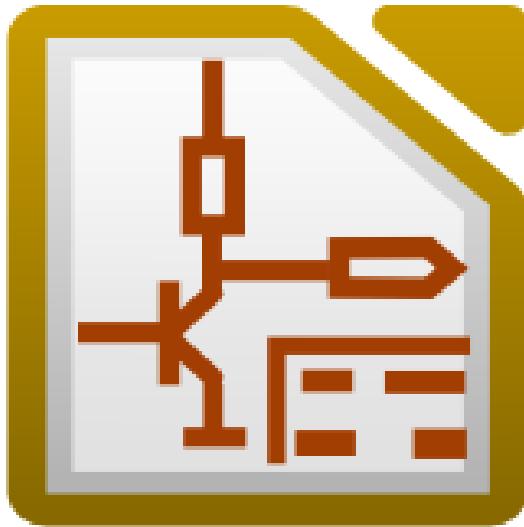
Компилятор GNU C++

Глава 29

Компилятор GNU Fortran

Часть IV

САПР электронных устройств KiCAD



1

2

KiCad — распространяемый по лицензии GNU GPL программный комплекс САПР EDA с открытыми исходными текстами, предназначенный для разработки электрических схем и печатных плат.

Кроссплатформенность компонентов KiCad обеспечивается использованием библиотеки wxWidgets. Поддерживаются операционные системы Linux, Windows NT 5.x, FreeBSD и Solaris.

Разработчик — Жан-Пьер Шарра (фр. Jean-Pierre Charras), исследователь в LIS (фр. Laboratoire des Images et des Signaux — Лаборатория Изображений и Сигналов) и преподаватель электроники и обработки изображений в фр. IUT de Saint Martin d'Hères (Франция).

3

¹ копипаста: <http://teholabs.com/knowledge/kicad.html>

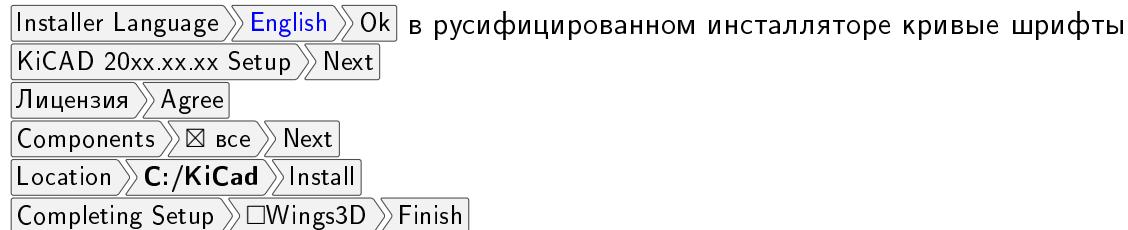
² копипаста: <http://ru.wikibooks.org/wiki/KiCad>

³ копипаста: <http://ru.wikibooks.org/wiki/KiCad/Miniurok>

Этот раздел познакомит Вас с основами использования системы KiCad. Он содержит информацию о всех шагах создания простой печатной платы: от рисования электрической схемы до печати готового рисунка платы. Вам будут представлены различные возможности KiCad и предложены эффективные пути решения различных задач.

Руководство пользователя, поставляемое вместе с KiCad, содержит значительно больше информации, чем этот урок. Ознакомьтесь с ним, чтобы узнать больше об использовании программы.

29.1 Установка под Windows



29.2 Установка под Linux

```
root# aptitude install kicad-doc-ru kicad
```

+++ ~/.blackbox/menu

```
1 [submenu] (CAD)
2 [exec] (KiCAD) {kicad}
```

Для добавления библиотек, поставляемых с этой книгой, сделайте [git checkout](#):

```
user:~$ git clone --depth=1 -o gh https://github.com/ponyatov/Azbuka.git Azbuka
```



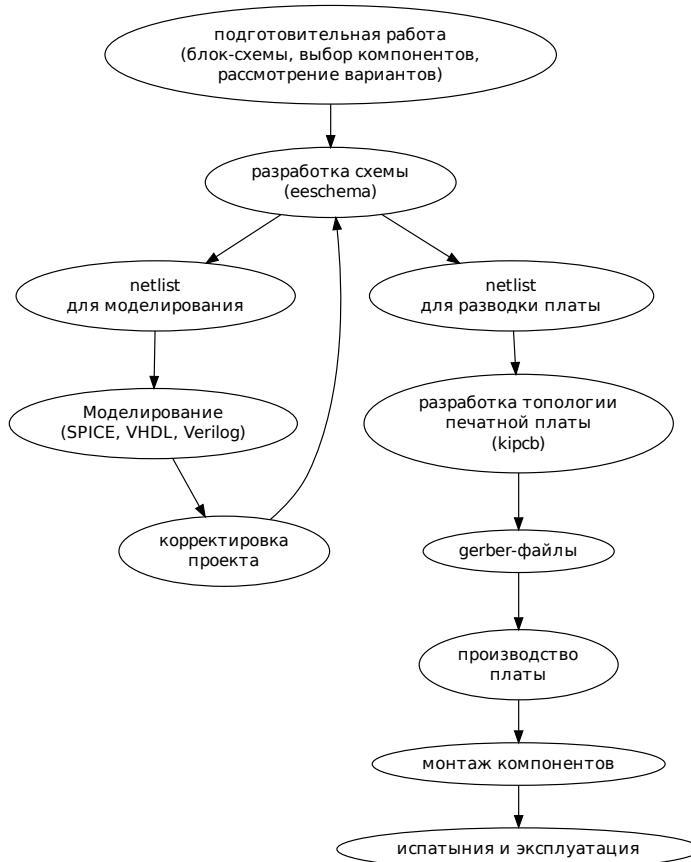
Для проверки работы библиотек можете открыть проект

kicad > Файл > Открыть > /Azbuka/bcollis/led1/led1.pro

или сразу схему

eeschema > Файл > Открыть > /Azbuka/bcollis/led1/led1.sch

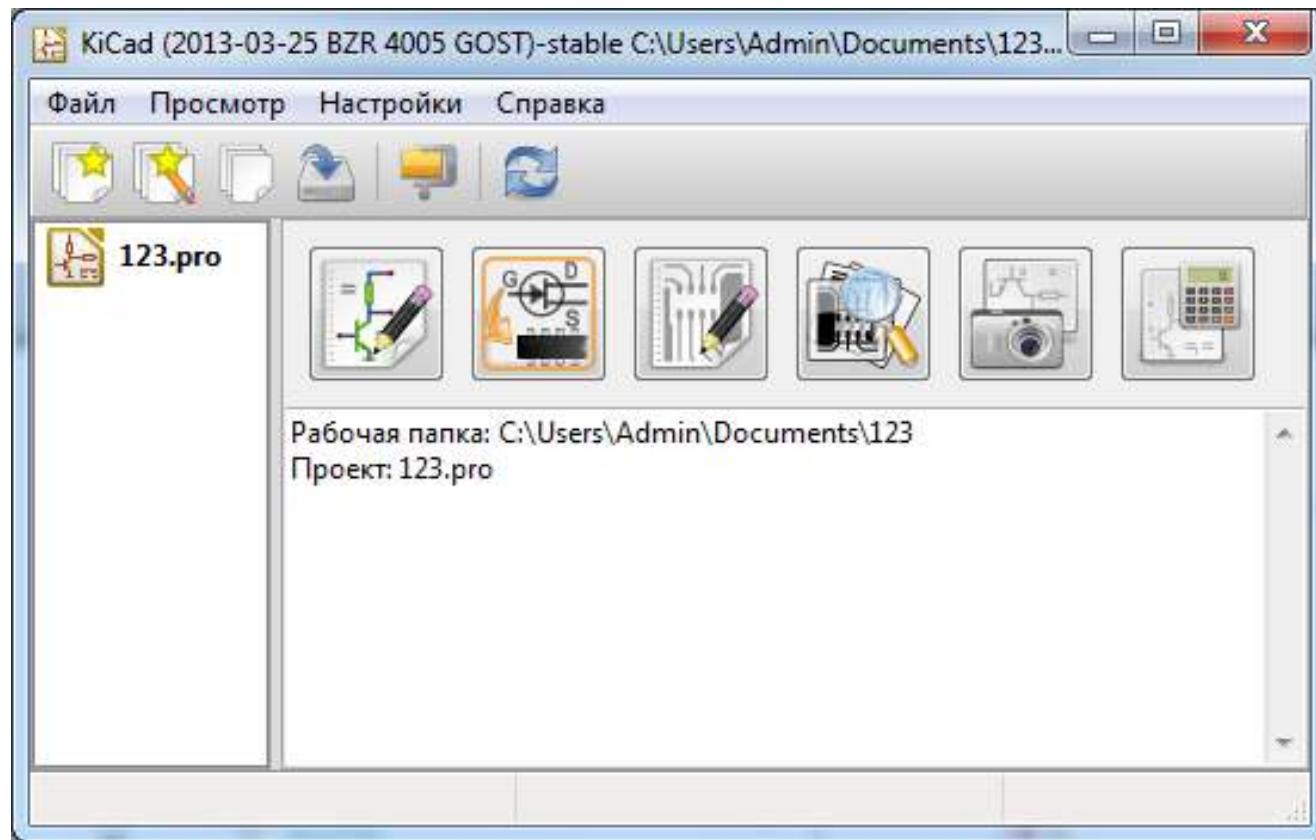
29.3 Маршрут проектирования



29.4 Создание проекта в менеджере проектов **kiCad**

Windows: Пуск > Программы > KiCAD > KiCAD

Linux: user@host\$ kicad



В верхней части панели менеджера проектов **kicad** имеются большие кнопки запуска компонентов KiCad:

-  **eeschema** — Редактор принципиальных схем
-  **pcbnew** — Редактор печатных плат
-  **cvpcb** — Программа редактирования **падстеков** (отверстий и площадок)

Каждая кнопка запускает соответствующую программу. Мы будем использовать эти программы по мере изучения.

-  **gerbview** — Программа просмотра фотошаблонов в формате Gerber
- **bitmap2component** — Создание компонента из черно-белого изображения (например логотипа)
-  **PcbCalculator** — Калькулятор для печатных плат
-  **PageLayout** — редактор формата листа схемы

Лучше всего для каждого проекта использовать раздельные папки; в противном случае система может сбиться с толку, если файлы из разных проектов будут лежать в одной папке. Проделайте следующие шаги:



1. Запустите программу KiCad



2. Создайте новый проект

- На панели инструментов KiCad выберите левую иконку с подсказкой **Начать новый проект**, используйте команду меню **Файл > Новый > Пустой** или сочетание клавиш **Ctrl + N**.

- Создайте папку проекта **DarkSensor**

- В диалоге **Создать новый проект** выберите созданную папку **DarkSensor** и введите имя проекта **DarkSensor** и нажмите **Сохранить**.

3. Если папка проекта содержит какие-то файлы, будет выведено окно выбора: создать подпапку с именем проекта **Yes**, или записать файл проекта в указанную папку как есть **No**. Нажмите **No**.

4. Сохраните проект кнопкой **Сохранить текущий проект**, **Файл > Сохранить** или **Ctrl + S**.

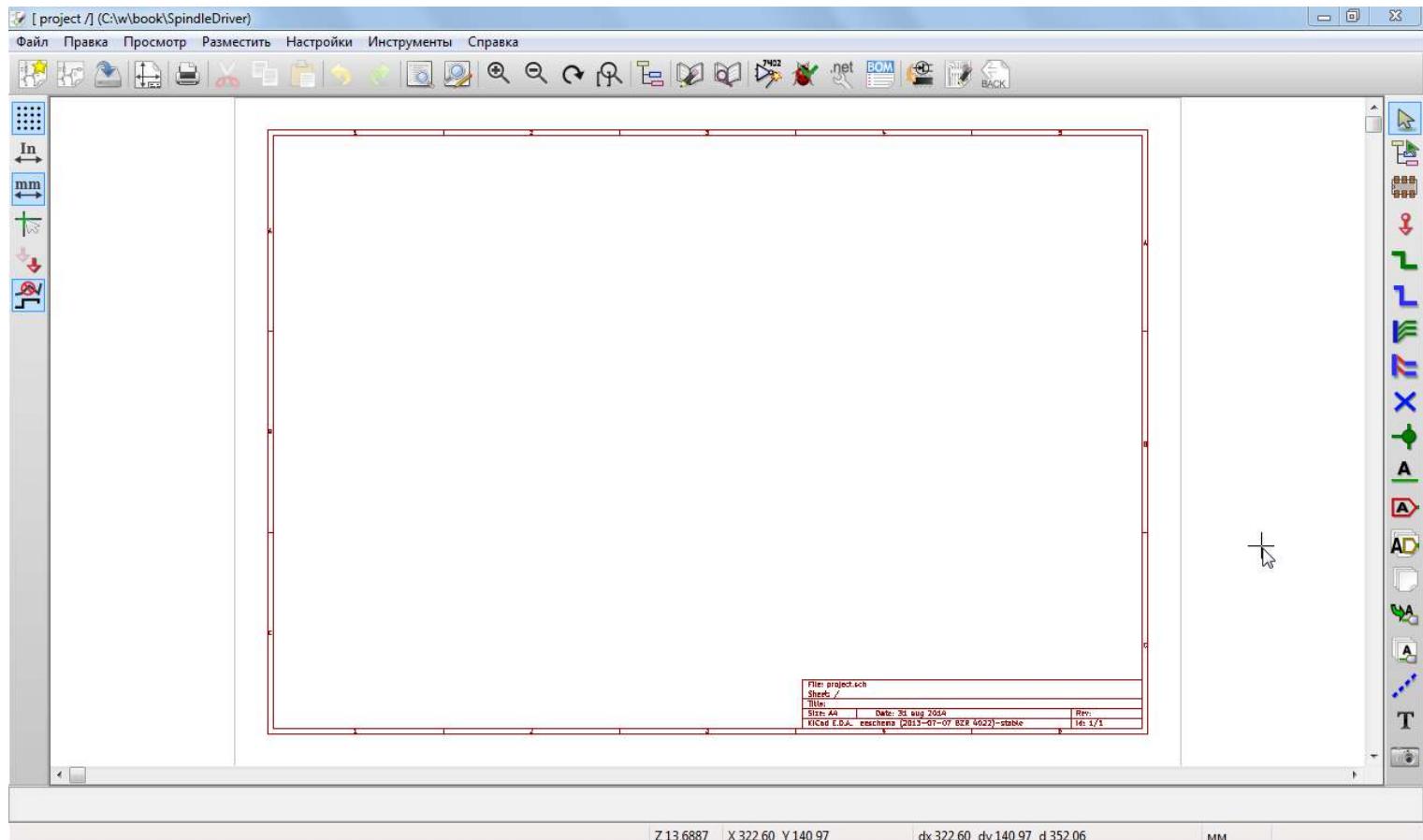
5. В папке появится файл **SpindleDriver.pro**, содержащий установки вашего проекта. Файл имеет текстовый формат, поэтому при необходимости его можно открыть в любом редакторе и вручную аккуратно подправить, например скорректировать настройки зазоров печатной платы.

29.5 Создание принципиальной схемы в **eeschema** (часть 1)

Запустите из менеджера проектов, графической оболочки или командной строки Linux модуль **eeschema**:

```
user@host$ eeschema &
```





На правом краю окна редактора схем есть вертикальная панель инструментов, которые мы и будем использовать для рисования схемы. Этими инструментами можно выбирать объекты, размещать компоненты, вводить связи и т.д.

29.6 **eeshema**: редактор электрических схем

обеспечивает:

- создание однолистовых и иерархических схем,
- проверку их корректности ERC (контроль электрических правил),
- создание списка электрических цепей netlist для редактора топологии платы pcbnew или для spice-моделирования схемы,
- доступ к документации на используемые в схеме электронные компоненты (datasheet).

Часть V

Расчет схем и моделирование в **ngSPICE**

на основе статьи

4

5

Electronic circuit simulation with gEDA and NG-Spice by Example

© Andreas Fester

SPICE: [S]imulation [P]rogram with [I]ntegrated [C]ircuit [E]mphasis — пакет программ симуляции и расчета электронных схем, была создана для моделирования [интегральных микросхем](#), расчета режимов работы, оптимизации, и предсказания поведения.

SPICE может выполнять несколько видо схемотехнических расчетов, самые важные из которых:

- Нелинейный анализ по постоянному току: вычисление передаточной характеристики по постоянному току
- Нелинейный анализ переходных процессов: вычисление токов и напряжений как функции времени в условиях большого сигнала
- Линейный АС анализ: вычисление выхода как функции от частоты. Выводится [bode plot](#)
- Анализ шума
- Расчет чувствительности
- Анализ искажений
- Фурье-анализ: вычисление и отображение частотных спектров
- Анализ Монте-Карло

⁴ копипаста: http://www.mithatkonar.com/wiki/doku.php/kicad/kicad_spice_quick_guide

⁵ копипаста: <http://physics.gmu.edu/~rubinp/courses/407/ngspice.pdf>

Глава 30

Доступные SPICE-пакеты

Первоначально SPICE был разработан в Университете Беркли. Другие версии SPICE являются форками этой реализации, и сейчас существует несколько вариантов:

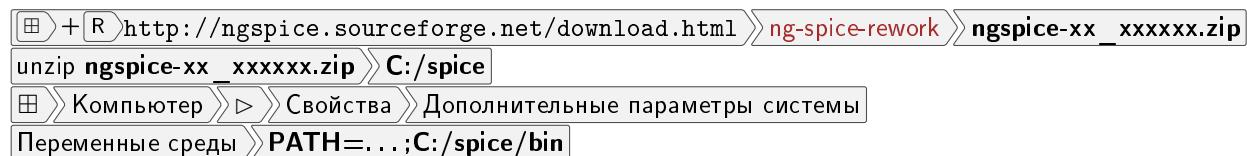
- **ngSPICE**: самый доступный бесплатный OpenSource SPICE-движок.
- <http://www.ngspice.com/> — on-line вариант **ngspice**, удобен для начального обучения
- **LT-SPICE**: Популярная бесплатная коммерческая версия от Linear Technology для Windows. Работает в **WINE**. Поставляется в комплекте с графической оболочкой, но требуется проверка файлов расчетных заданий, которые она создает.
- **gnucap**¹: Не совсем SPICE, но пытается быть синтаксически совместной.
- **SpiceOpus**: Коммерческая, но удобная, особенно в плане вывода графиков.

¹ уже включен в виндозную сборку KiCAD

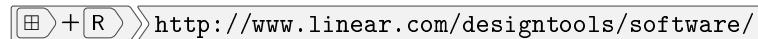
- PSpice: Windows-only, дорогой коммерческий пакет, стандарт de-facto для профессионального применения в USA в составе тяжелых EDA-продуктов. Они традиционно предоставляют обрезанную *gratis*-версию для студентов. Также имеет в комплекте GUI.
- MultiSim, OrCAD, DesignLab,.. коммерческие EDA-пакеты, содержит в составе одну из версий PSPICE
- Какие-то еще?

В этом разделе описан ngSPICE, который был написан с целью полностью переписать оригинальную реализацию Беркли SPICE. Сейчас он все еще содержит часть кода Беркли, но в этом коде было исправлено множество ошибок. Важно понять, что каждая реализация SPICE может вести себя особенно в некоторых случаях: некоторые из них более совместимы между собой, другие менее. Всегда важно прочитать документацию, которая поставляется с конкретной версией SPICE. Дистрибутив ngSPICE поставляется с детальным руководством пользователя, а этот раздел поможет вам начать. Хорошо иметь под рукой полное руководство при чтении этого раздела, так как интересующие вас команды там рассмотрены детальнее.

30.1 Установка ngSPICE под Windows



30.2 Установка LT-SPICE (только Windows)



30.3 Установка ngSPICE под Linux

```
root# aptitude install ngspice
```

Пакет **ngSPICE** также может быть легко собран и установлен компиляцией их исходниов. После загрузки архива, соберите пакет для вашего дистрибутива:

```
tar xvzf ng-spice-rework-15.tgz  
cd ng-spice-rework-15  
../configure --with-readline=yes  
make  
checkinstall
```

Убедитесь что в системе присутствует библиотека **GNU readline**, и она включена в опциях **configure**: ее использование делает интерфейс командной строки более комфортным. Подробнее сборка **ngSPICE** описана в его руководстве в составе дистрибутива.

Сборка **ngSPICE** для azLinux описана в разделе ??.

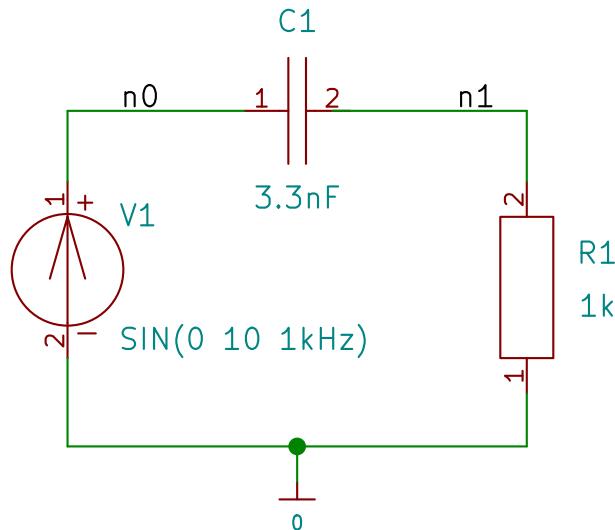
Глава 31

Пошаговый пример использования

Рассмотрим очень простой пример симуляции электронной схемы. В основном вы будете рисовать ваши схемы в [eeschema](#), но интерактивная симуляция пока не реализована.

31.1 Рисуем схему в KiCAD

Схема, которую мы хотим рассчитать — простой [RC-фильтр](#). Эта схема была выбрана, так как она содержит очень ограниченное количество элементов, и поэтому понятна: источник напряжения, резистор и конденсатор. Использование редактора схем вам уже должно быть знакомо из предыдущего раздела (29.6). Запустите [eeschema](#) и нарисуйте схему:



Простой RC-фильтр

Имена, указанные на проводниках — **имена цепей**. Они могут не указываться, если вам не нужно на них ссылаться в расчете. При экспорте списка цепей из **eeshema** им будут присвоены универсальные имена. Цепь может иметь любое имя, за некоторым исключением: **в списке должна быть одна цепь с именем [0]**, она подключается к общему проводу (**земле**). Рисуя схему, поместите на нее один или несколько элементов [0] из библиотеки **spice**.

V1 — **независимый источник напряжения**. Его значение задано в виде выражения **SIN(0 10 1kHz)**, создающего синусоидальный (**SIN**) сигнал со смещением (0) вольт, амплитудой (10) вольт и частотой (1kHz).

Рисуйте вашу схему, соблюдая несколько рекомендаций:

- Для именованных цепей используйте **глобальные метки** вместо локальных. В списке цепей глобальные идентификаторы цепей включаются как есть, а локальные метки модифицируются, что делает сложным последующие ссылки на них при SPICE-моделировании.

- Используйте компонент [0] из библиотеки **spice**, вместо обычного компонента [GND]: "0" официальное имя главной земли в файлах PSPICE. Некоторые SPICE-движки умеют транслировать *GND* → 0, другие нет.

31.2 Создание списка цепей

Входными данными для симуляции является **список цепей (netlist)**. Список цепей создается через экспорт из **eeschema**. Если вы используете программу рисования схем, посмотрите в ее документации, умеет ли она экспорт в SPICE(.cir), и как это сделать.

- Кликните кнопку или пункт меню **Сформировать список цепей**.
- Выберите вкладку **Spice**, и убедитесь что включен крыжик **Формат по умолчанию**. Вам нужно сделать это только один раз, настройки запоминаются.
- Нажмите кнопку **Сформировать**

Если вы хотите запускать **ngSPICE** из диалога экспорта:

- Заполните полный путь с программе симуляции, типа **C:/spice/bin/ngspice.exe** со всеми путями и расширениями, **KiCAD** пока не научился запускать симулятор через PATH.
- Нажмите кнопку **Запустить симулятор**.

В результате экспорта будет создан файл

RCfilter.cir

```
1* RC_filter
2
3*Sheet Name:/
4R1 0 /n1 1k
5C1 /n0 /n1 3.3nF
6V1 /n0 0 SIN(0 10 1kHz)
7
8.end
```

Формат нетлиста SPICE прост: каждая строка содержит один элемент схемы. Первый столбец каждой строки содержит имя элемента, затем идут имена цепей, которым подключен каждый вывод, последним идет значение элемента. Строки, начинающиеся с [*] — комментарии. В нашем примере строка, начинаящаяся с **V1**, описывает источник напряжения, подключенный к цепям **n0** (вывод 1) и **0** (вывод 2), значение **SIN(0 10 1kHz)**. Точно также заданы конденсатор и резистор. Так как цепи **n0,n1** заданы именами, перед ними стоит [/].

Первая и последняя строки имеют для ngSPICE особое значение: при чтении нетлиста ngSPICE считает первую строку названием схемы. Последняя строка должна содержать токен **.end**.

Так как формат файла нетлиста настолько прост, его можно легко создать вручную из любого текстового редактора. Некоторые статьи о SPICE-симуляции даже начинаются с такого способа, но он действительно неудобен для работы. Используя **eeschema** или другой редактор схем, намного проще изменять схему, и она может быть легко распечатана или включена как иллюстрация в документацию. Единственный реальный вариант, когда нужно работать напрямую в файлом — если вам вдруг понадобиться сформировать его автоматически, например при анализе паразитных емкостей и индуктивностей печатной платы, которые определяются формой печатных проводников.

31.3 Запуск симуляции

Теперь вы готовы симулировать схему. Прежде всего нам нужно решить, какие виды расчетов, которые умеет делать SPICE, нас интересуют:

- Анализ переходных процессов показывает поведение схемы во времени.
- Расчет по переменному току (AC) дает изменения работы схемы с изменением (входой) частоты.
- Параметрическая симуляция позволяет анализировать изменения в работе схемы при изменении одного или нескольких параметров, например изменении частоты источника и емкости конденсатора.

Для начала посмотрим как ведет себя входное напряжение во времени. Мы хотим выполнить анализ переходных процессов в схеме, и вывести напряжение между сетями `n0` и `0`.

Запускаем ngSPICE:

```
$ ngspice

spinit found in c:\spice\share\ngspice\scripts\spinit
*****
** ngspice-24 : Circuit level simulation program
** The U. C. Berkeley CAD Group
** Copyright 1985-1994, Regents of the University of California.
** Please get your ngspice manual from http://ngspice.sourceforge.net/docs.html
** Please file your bug-reports at http://ngspice.sourceforge.net/bugrep.html
** Creation Date: Jan 30 2012 22:58:51
*****
ngspice 1 ->
```

Теперь нам нужно загрузить сетлист (выводится заголовок: первая строка файла):

```
ngspice 1 -> source RCfilter.cir
```

```
Circuit: * eeschema netlist version 1.1 (spice format) creation date: 26.12.2014 16:15:26
```

Так как мы задали для входного напряжения частоту 1КГц, период $T = 1/F = 0.001\text{с} = 1\text{мс}$. Мы хотим увидеть как входное наряжение меняется за первые 5 периодов, т.е. 5 мс. Запускаем симуляцию следующей командой:

```
ngspice 2 -> tran 0.01ms 5ms
```

```
Doing analysis at TEMP = 27.000000 and TNOM = 27.000000
```

```
Warning: v1: no DC value, transient time 0 value used
```

```
Initial Transient Solution
```

```
-----
```

Node	Voltage
---	-----
/n1	0
/n0	0
v1#branch	0

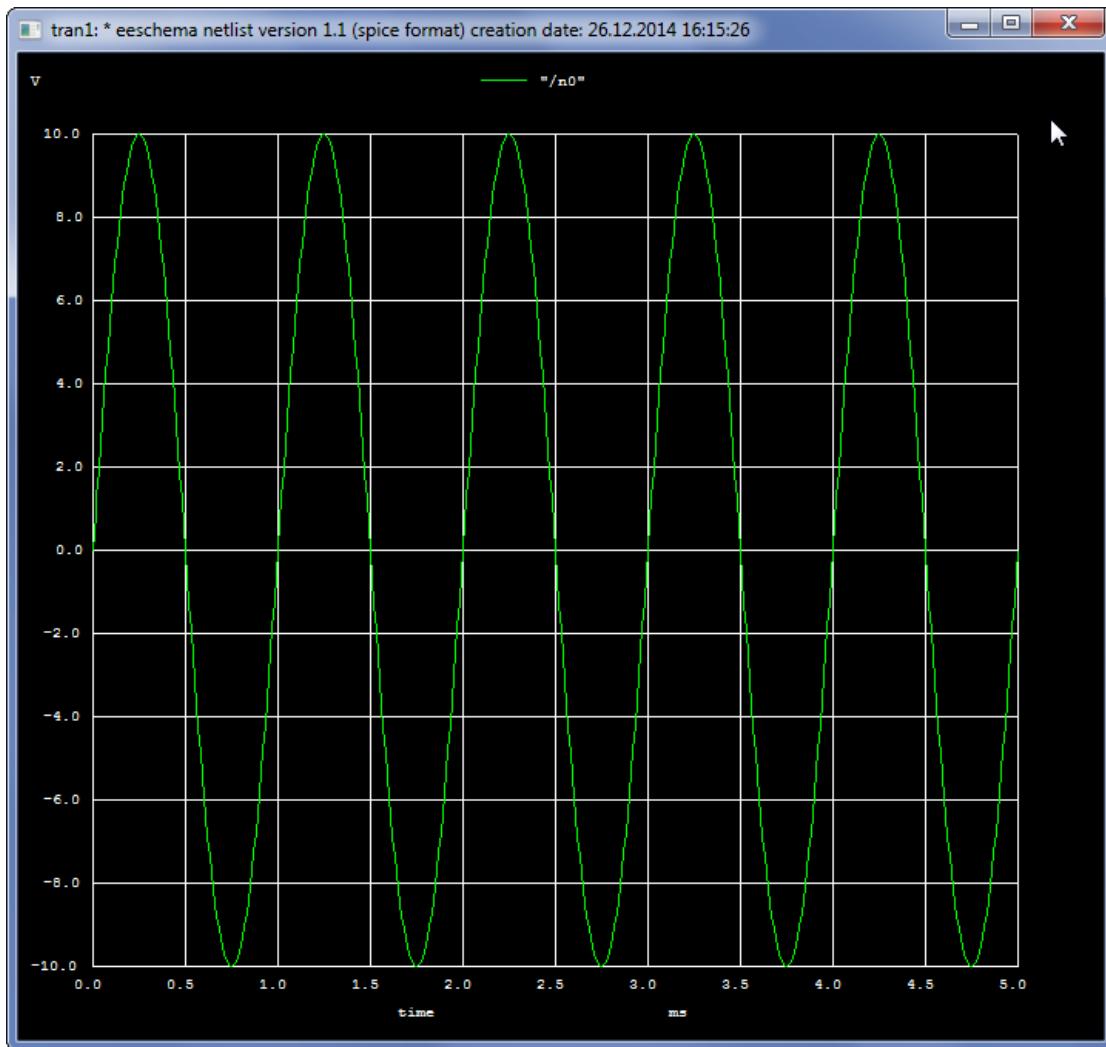
```
No. of Data Rows : 512
```

Первый параметр `tran` определяет [шаг расчета](#), второй — [конечное значение времени](#). Если не указан третий параметр, начальное время равно 0, иначе третий параметр указывает ненулевое [начальное время](#). Ну, это все 😊. Симуляция выполнена. Теперь нам нужно увидеть результат симуляции.

31.4 Просмотр результата расчета

SPICE создал таблицы с рассчитанными значениями: 512 значений для каждого узла схемы. Для простого просмотра чисел выполним команду (не забудьте про кавычки, без них не работает если первый символ [/]):

```
ngspice 3 -> plot "/n0"
```



Эта команда вывела напряжение при переходном процессе на цепи [n0].

Как вы заметили, диаграмма сигнала отображается на черном фоне. Если вам нужны другие цвета, например для вставки в документацию, их можно переопределить:

```
ngspice 12 -> set color0 = white  
ngspice 13 -> set color1 = black  
ngspice 14 -> set color2 = green  
ngspice 3 -> plot "/n0"
```

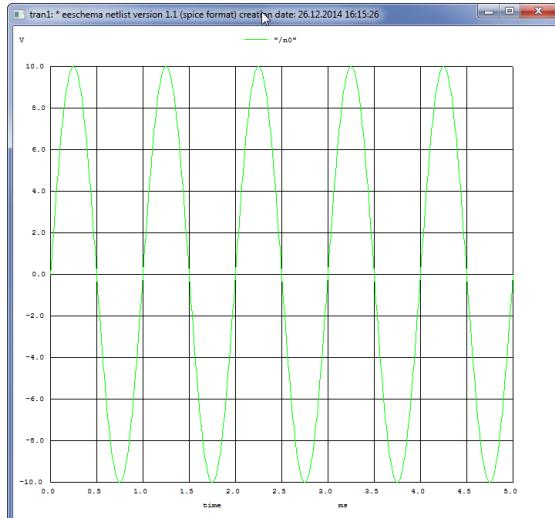
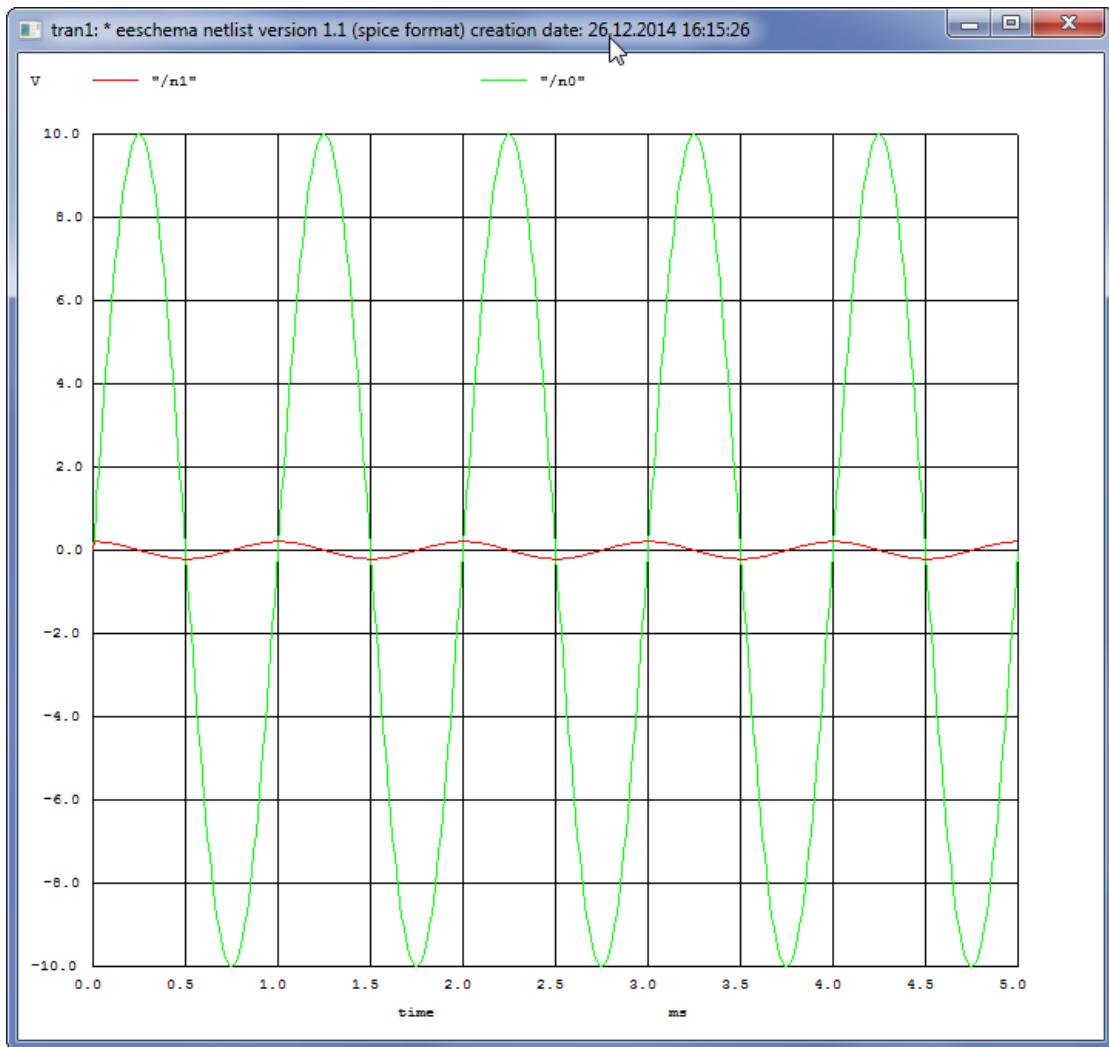


Диаграмма показывает форму входного сигнала, как мы и ожидали, но она нас мало интересует, так как мы ее и задали. Нам интереснее например [напряжение на резисторе](#), кроме того мы попробуем [сравнить два сигнала](#). Это легко сделать указав два имени цепи:

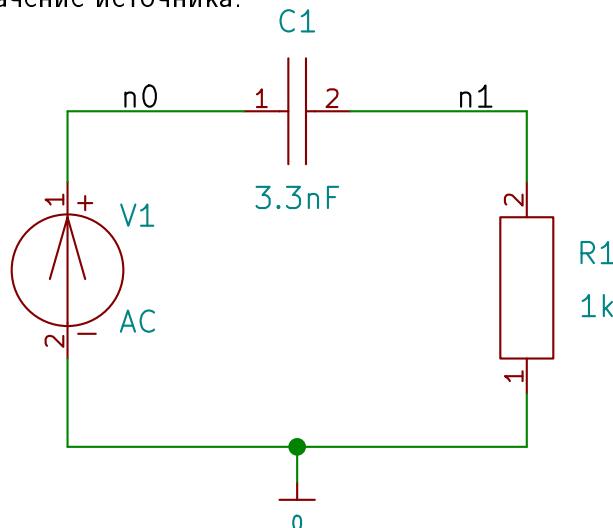
```
ngspice 31 -> plot "/n0" "/n1"
```



31.5 Расчет АЧХ по переменному току (AC симуляция)

Сигнала на резисторе почти не видно. Теперь вопрос: какие частоты попускает наш фильтр? Для определения этого теперь выполним расчет по переменному току (**AC симуляцию**). Команда для этого **ac (DEC j OCT j LIN)N FStart FEnd**.

FStart и **FEnd** — соответственно начальная и конечная частота. Необязательные параметры **DEC**, **OCT** или **LIN** указывают способ изменения частоты: декадно, октавно или линейно. Если выбрана октавная или декадная **вариация частоты**, то параметр **N** задает число частот на декаду или октаву. Для выполнения **AC анализа** должен быть изменен источник сигнала: сейчас он определен как синус с амплитудой 10 В и частотой 1 КГц. Для анализа это должен быть **источник переменного напряжения**. Снова запускаем **eeshema** и меняем значение источника:



Создаем нетлист, загружаем его и запускаем команду AC анализа:

```
$ ngspice ACanaliz.cir
```

```
ngspice 1 -> ac lin 1000 0.1 250kHz
```

```
Doing analysis at TEMP = 27.000000 and TNOM = 27.000000
```

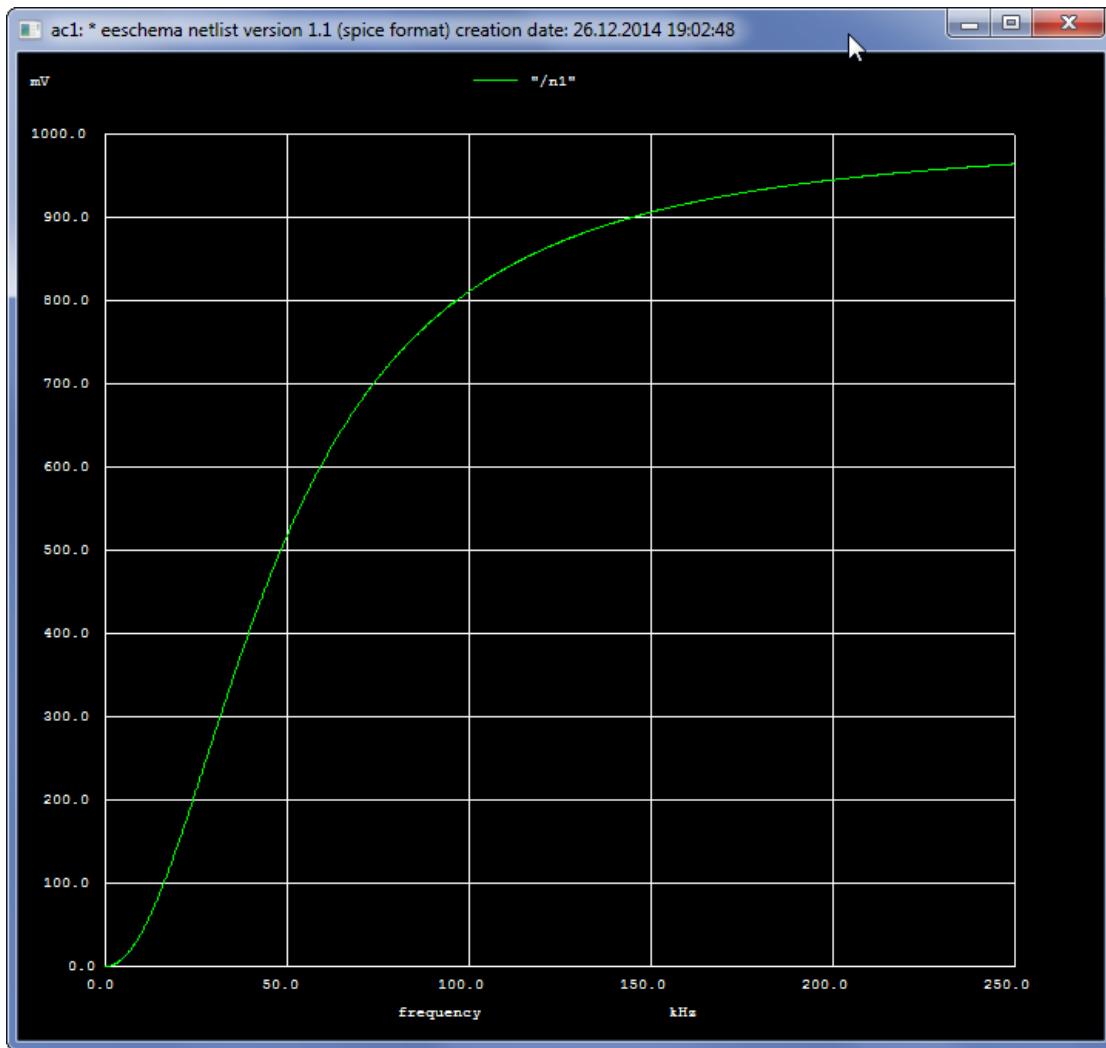
```
Warning: v1: has no value, DC 0 assumed
```

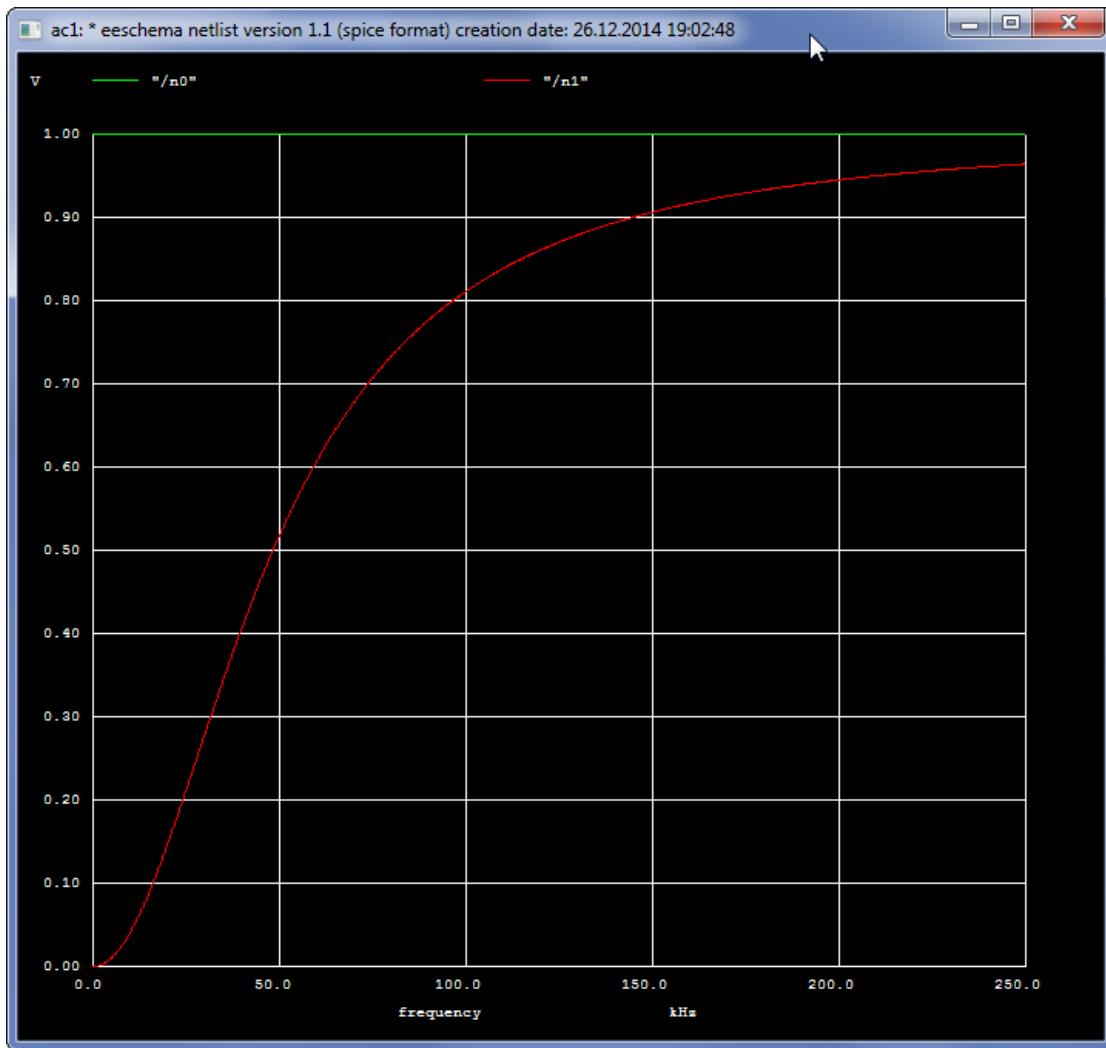
```
No. of Data Rows : 1000
```

```
ngspice 2 -> plot "/n1"
```

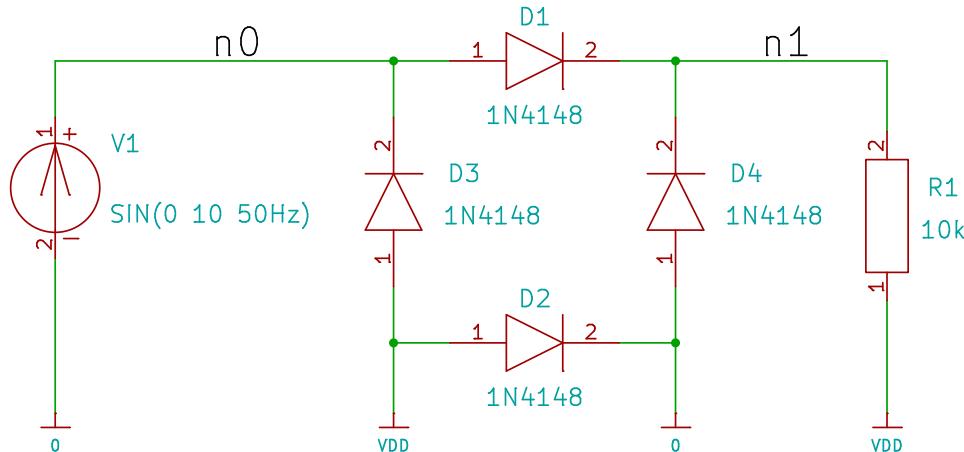
```
ngspice 3 -> plot "/n0" "/n1"
```

Эта команда выполняет **линейный АС анализ** от (почти) 0 Гц до 250 КГц. Результат можно увидеть как напряжение для источника, так и напряжение на **R1**:





31.6 Симуляция полноволнового выпрямителя



```
-PSPICE
* text before netlist

+PSPICE
* text after netlist
.control
tran 0.01ms 50ms
set hcopydevtype=postscript
set hcopypscolor=true
set color0=white
set color1=black
set color2=rgb:F/0/0
set color3=rgb:0/F/0
set color4=rgb:0/0/F
hardcopy RectifierPlot.eps "/n0" "/n1"
quit
.endc
```

Rectifier.cir

```
1 * Rectifier
2
3 * text before netlist
4
5 *Sheet Name:/
6 D4 0 /n1 1N4148
7 D2 0 0 1N4148
8 D3 0 /n0 1N4148
9 D1 /n0 /n1 1N4148
```

```
10 R1 0 /n1 10k
11 V1 /n0 0 SIN(0 10 50Hz)
12
13 * text after netlist
14 .control
15 tran 0.01ms 50ms
16 set hcopydevtype=postscript
17 set hcopypscolor=true
18 set color0=white
19 set color1=black
20 set color2=rgb:F/0/0
21 set color3=rgb:0/F/0
22 set color4=rgb:0/0/F
23 hardcopy RectifierPlot.eps "/n0" "/n1"
24 quit
25 .endc
26
27 .end
```

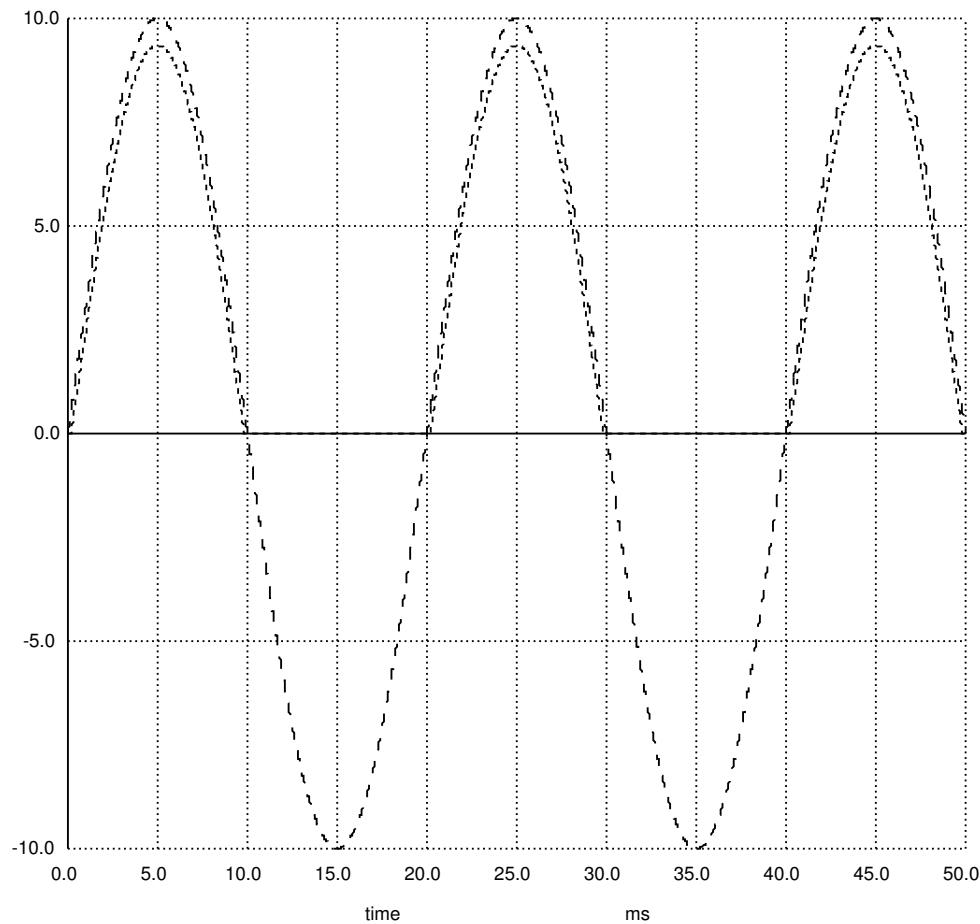
Подробнее использованные здесь приемы рисования схемы описаны в 32.

Кратко: была применена возможность вставки в нетлист текстовых блоков до и после списка элементов.

При запуске **ngspice** из **KiCAD** будет автоматически выполнен блок **.control/.endc**. Также для вывода графиков была использована команда **hardcopy**, предварительно настроенная на вывод в формате (**Encapsulated PostScript**). Текущая версия **ngspice** не умеет цветной ввод, он должен быть починен в следующих версиях.

V ---- "/n1"

— — "/n0"



Глава 32

Настройка KiCAD для SPICE-моделирования

32.1 Библиотеки компонентов со SPICE-элементами

- Библиотека базовых SPICE-компонентов поставляется с KiCAD. Эта библиотека — хороший вариант для начальных экспериментов. Библиотека не подключена по умолчанию, вы должны сделать это вручную сами через менеджер библиотек. На Debian Linux это файл `/usr/share/kicad/library/pspice.lib`¹
- Mithat Konar <webs@mithatkonar.com> разрабатывает (очень медленно) собственную библиотеку с некоторыми модификациями.
- В комплекте с этой книгой поставляются библиотеки, адаптированные для SPICE.

¹ PSpice — популярная коммерческая версия SPICE

32.2 Настройка проекта

1. Создайте новый проект как обычно.
2. Откройте **Eeschema** и удалите все библиотеки, подключаемые по умолчанию.
3. Вручную добавьте одну из SPICE-библиотек, или набор библиотек для этой книги. Обратите внимание, что SPICE-библиотека из поставки **KiCAD** по умолчанию не подключается к проекту.
4. Укажите расчетный SPICE-движок, который вы хотите использовать:

eeshema > Меню > Инструменты > Сформировать список цепей > Spice

Формат по умолчанию

Префикс обозначений

Использовать имена цепей

вкладка Spice > Команда симулятора: > **xterm -e ngspice**

Список цепей

32.3 Как это работает

1. Укажите режимы симуляции, которые вы хотите выполнить, и генерацию вывода, который хотите отобразить, добавив на схему текстовый блок (т.е. “комментарий”) с необходимыми директивами в синтаксисе SPICE и Nutmeg с некоторыми добавками. Например, для выполнения **расчета по постоянному току** и вывода сигнала в точке **vout**, добавьте блок:

1 +PSPICE

2 .control

```
3 ac dec 66 1kHz 120kHz
4 plot vdb(vout)
5 set units = degrees
6 plot vp(vout)
7 .endc
```

- Первая строка “+PSPICE ” указывает kicadу добавить текст [в конец](#) сформированного .cir-файла. В текущей версии KiCAD есть баг, который требует обязательного пробела после +SPICE.
- Соответственно строка “-PSPICE ” добавляет текст [в начало](#) .cir-файла.
- Для поборников OpenSource, не желающих видеть ссылка на коммерческий PSpice, предусмотрены директивы-синонимы ±“GNUCAP ”. Я думаю это то же самое что и ±“PSPICE ”, но не уверен на 100%, проверьте в документации.
- Да, вам потребуется немного изучить синтаксис SPICE and Nutmeg. Это нетрудно.

2. Запустите симуляцию:

[eeschema](#) >> Меню >> Инструменты >> Сформировать список цепей >> Spice
[Запустить симулятор](#)

Часть VI

Разработка конструкции в САПР FreeCAD



² В среде специалистов ряда отраслей известна проблема создания полноценной САПР в рамках OpenSource, и хотя FreeCAD ещё не является кандидатом на такую «полноту», 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 копипаста: [https://ru.wikipedia.org/wiki/FreeCAD_\(Juergen_Riegel%27s\)](https://ru.wikipedia.org/wiki/FreeCAD_(Juergen_Riegel%27s))

³ 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, многое 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 единственный 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 что не забывайте об этом, когда начинаете использовать FreeCAD, если вам он нравится, вы можете непосредственно влиять и помочь проекту!

³ копипаста: http://www.freecadweb.org/wiki/index.php?title=Getting_started

Глава 33

Установка

33.1 Windows





[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

33.2 Linux

Часть VII

Инструменты и электронное оборудование

Глава 34

Радиомонтажный инструмент

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

34.1 Pro'sKit

Отдельного обзора заслуживает инструмент и наборы Pro'sKit



PK-5308VM универсальный набор инструментов



1PK-616B Набор инструментов для электроники профессиональный



1PK-813B Набор базовых инструментов для электроники

По личному опыту: в 1РК-813В не хватает

- мелкого мультиметра,
- стриппера 1РК-3001Е,
- микрокусачек типа 8РК-30Д,
- канифоли,
- ножа,
- настроечную отвертку заменить индикаторной.

34.2 Инструмент до 1000 В

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



PM-911 Пассатижи 1 кВ



PM-917 Кусачки (бокорезы) 1 кВ

34.3 Хранение



103-132D Кассетница для деталей и компонентов



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

34.4 Радиомонтаж



8PK-30D Кусачки миниатюрные



1PK-709 Длинногубцы-кусачки



1PK-055S Длинногубцы изогнутые



1PK-29 Круглогубцы



1PK-101Т Пинцет прямой



1PK-3001Е Клещи для зачистки проводов
прецзионные (стриппер)



PD-374 Тиски на струбцине

34.5 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.



Глава 35

Паяльное оборудование

35.1 Паяльник

Паяльник — обязательен 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 ЭПЧН-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.

35.2 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.

Глава 36

Измерительное оборудование

36.1 Мультиметр

Мультиметр — обязательен, без него работать невозможно¹. Для совсем начинающего больше всего подойдет M32036.1.3 с автодиапазоном, когда освоитесь вторым прибором что-то из крупных серий M89x/MY6x с измерением температуры² или “рыльцеметр”36.5 (RLC).

¹ или собирать замену на паре измерительных головок тока/напряжения, и делителях

² иногда нужно для измерения температуры корпусов элементов, радиаторов, растворов если возитесь с электрохимией

36.1.1 Mastech M838



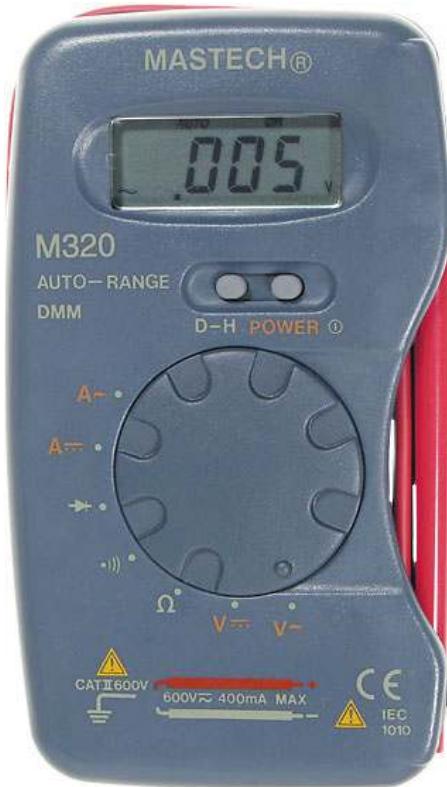
Простой, компактный, дешевый, с измерением температуры

36.1.2 Mastech M300



Простой, [очень компактный](#), дешевый, в чехле очень удачно умещается в набор инструментов.

36.1.3 Mastech M320



То же что и M30036.1.2, но с [автодиапазоном](#), т.е. не требует переключения диапазонов измерения вручную. На любителя, возможно [удобен для совсем начинающих](#), но слишком медленен если требуется измерение меняющегося тока/напряжения.

36.2 Осциллограф

36.3 Логический анализатор

36.4 Генератор сигналов

36.5 Рыльцеметр RLC

Глава 37

Электроинструмент

37.1 Дрель



Дрель ударная сетевая

Praktyl-R PID13D01 400 Вт (!)395 р.



Дрель безударная сетевая

Интерскол Д-11/530ЭР (с БЗП) 1120 р.

Дрель — одноразовая китайчатаина 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, запах горелой пластмассы через несколько минут работы на холостом ходу нет.

По надежности рекомендуется 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 37.1 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.



37.2 Лобзик



LEROY MERLIN
Твой Дом

Praktyl 350 Вт 356 р.



Makita 4329 2260 р.

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

37.3 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 37.1, 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 тоцилке¹.

Ограниченно доставаемые двигатели от стиральных машин, отличаются мощностью 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 шпинNELи постоянного тока с цанговыми патронами ER11. Требуют источник питания постоянного тока 9÷48 В. В магазинах не попадались, необходима прямая покупка с AliExpress² по почте.

¹ БЗП удобен, патрон с ключем дает лучший зажим и возможно точнее

² пользуйтесь английской версией — переводная жуткое УГ



Двигатель Вятка-Автомат 19?? г.



Двигатель печки Камаза



АИРЕ 56 В2, 0.2 кВт



Воздушный шпиндель с цангой ER11

Съемные фрезерные шпинNELи, 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 попате. 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.

Часть VIII

Станочное оборудование

Самые распространенные станки — **сверлильные**, т.к. имеют самую простую конструкцию, и минимальную стоимость. Предназначены для самой частой операции: изготовления перпендикулярных круглых дырок в различных материалах, топовые модели имеют также функцию нарезания резьбы. Для монтажа электроники и кустарного изготвления печатных плат часто используются очень маленькие настольные сверлилки, часто **самодельные**.

Наиболее многочисленную группу металлорежущих станков составляют **токарно-винторезные станки**, используются в механических, инструментальных и ремонтных цехах заводов, а также в ремонтных мастерских в основном для обработки деталей, имеющих форму тел вращения. При использовании соответствующей оснастки позволяют растачивать отверстия в призматических (прямоугольных) деталях, и фрезеровать небольшие детали. Самый ходовой тип детали — тела вращения с наружными и внутренними резьбами: валики, втулки, оси, болты, винты, шпильки, кольца, шайбы и т.д. К основным размерам, характеризующим токарный станок, относятся

- наибольший допустимый диаметр обрабатываемой заготовки,
- высота центров над станиной и
- расстояние между центрами.
- Часто обращают внимание на диаметр проходного отверстия шпинделя, определяющий максимальный диаметр **длинномерных заготовок**, что важно при изготовлении партий мелких деталей из длинных прутковых заготовок и нарезке резьб на трубах.

Значительную часть среди металлорежущих станков составляют **фрезерные станки**. Наибольшее распространение имеют консольно-фрезерные. Предназначены для выполнения различных фрезерных работ цилиндрическими, дисковыми, фасонными и другими **фрезами**, можно фрезеровать плоскости, пазы, фасонные поверхности, и т.д. Кроме этого, универсальные консольно-фрезерные станки с поворотным столом или делительной головкой позволяют фрезеровать различного рода винтовые канавки и зубья зубчатых колес. Для

самодельной электроники интересны универсальные малогабаритные фрезеры, способные работать в режимах вертикальной и горизонтальной фрезеровки, для изготовления самых разнообразных деталей, а при установке в горизонтальный шпиндель токарной оснастки и небольшую часть токарных работ.

Основными размерами фрезерных станков, по которым можно определить возможность установки и обработки конкретных заготовок с определенными габаритами, являются размеры рабочей поверхности стола (длина и ширина) и **рабочий ход стола/рабочая зона** в продольном, поперечном и вертикальном направлениях. Этими размерами, и типом шпинделя, также определяется возможность установки дополнительного оборудования, выпускаемого серийно: делительных столов, расточных головок, оснастки для нарезки зубчатых колес и т.п.

Общая рекомендация — берите самые большие станки с самой большой рабочей зоной, какие можете себе позволить по цене, помещению для установки, мощностью электропроводки, и стоимостью эксплуатации, обслуживания и расходных материалов. Чем больше станок, тем большую деталь вы сможете изготовить сами, и тем больше возможностей по использованию дополнительного оборудования. Хотя настольные станки дешевы и практически не требуют отдельного помещения, оснастку для них (например поворотный столик) вы для них не найдете, придется ее делать самому или где-то заказывать.

Глава 38

Настольные станки

На основе¹

© Joe Martin, illustration by Craig Libuse

Tabletop Machining

A basic approach to making small parts on miniature machine tools

© Джо Мартин, иллюстрации Craig Libuse

Базовые навыки изготовления мелких деталей на миниатюрных настольных станках

¹ копипаста: <http://rutracker.org/forum/viewtopic.php?t=3126529>

Глава 39

Самодельная оснастка

Глава 40

Промышленные станки

Иногда хоббиту удается получить доступ к старым промышленным станкам. Наиболее богаты в этом плане школы и другие учебные заведения, у них часто где-нибудь в углу или подвале стоят пара старых станков производства СССР. Несмотря на то, что стоят они годами без движения, добраться до них получается с большим гемором: нужно как минимум иметь официальный документ о наличии разряда токаря, фрезеровщика или станочника широкого профиля. Кроме того, без получения какого-то официального статуса, а соответственно и кучи ненужных обязанностей, допуск к станку вы тоже не получите.

Общественных открытых технологических площадок в России не существует в принципе, большая часть станочного оборудования установлена на закрытых территориях, или гниет в подвалах и школах.

40.1 1A616: станок токарно-винторезный



- 40.1.1 Назначение и области применения**
- 40.1.2 Распаковка и транспортировка**
- 40.1.3 Фундамент станка, монтаж и установка**
- 40.1.4 Подготовка станка к первоначальному пуску**
- 40.1.5 Паспортные данные**
- 40.1.6 Описание основных узлов**
- 40.1.7 Смазка**
- 40.1.8 Первоначальный пуск**
- 40.1.9 Указания по технике безопасности**
- 40.1.10 Настройка**
- 40.1.11 Регулирование**
- 40.1.12 Ведомость комплектации**

Часть IX

Разработка ПО для встраиваемых систем

Глава 41

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

¹ файлы не включающие непечатаемых символов и бинарных данных, которые можно прочитать простым выводом на экран командами типа `type`, `cat`, `more`

- цветовой и шрифтовой подсветкой 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 дерева классов и структур данных
 - 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** 41.1: 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  Luna работает заметно быстрее (особенно при запуске).
2. **Code::Blocks** 41.2: легкая среда для разработки на C/C₊⁺, для других языков может потребоваться написать свои модули или файлы описания синтаксиса
3. **(g)Vim** 41.3: 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 😊

41.1 eclipse



41.1.1 Установка eclipse под Windows

 <https://eclipse.org/> > Download > Eclipse Luna release for > 

Качаем архив базовой системы: [Eclipse IDE for Java Developers](#) > 

Или сразу сборку CDTECLIPSE: [Eclipse IDE for C/C++ Developers](#) > 

41.1.2 Установка eclipse под Linux

 <https://eclipse.org/> > Download > Eclipse Luna release for > 

Качаем архив базовой системы: [Eclipse IDE for Java Developers](#) > 

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
```

Распаковываем 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

41.1.3 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.



OK

Work with > CDT

CDT Main Features > C/C++ Development Tools

CDT Optional Features

Парсер файлов исходников наialecte 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

41.1.4 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 > Certificate > Restart Eclipse > Ok

41.1.5 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.

41.1.6 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

41.1.7 Проверка орфографии

2

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.

² копипаста: <http://www.simplecoding.org/proverka-orfografiyi-v-eclipse.html>

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.

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

[田] + R cmd

```
1 cd \GnuWin32\Aspell  
2 bin\aspell dump master en > en.dict  
3 bin\aspell dump master ru > ru.koi8  
4 iconv -f koi8-r -t utf-8 < ru.koi8 > ru.dict  
5 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

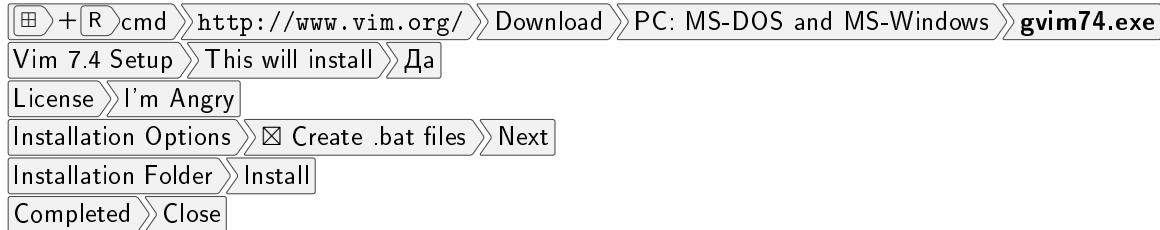
Apply ➤ OK

41.2 Code::Blocks

41.3 (g)Vim



41.3.1 Установка под Windows



Do you want to see README

» Да

Теперь можно настроить темную тему и выключение подсветки синтаксиса, по умолчанию после установки используется светлая тема и подсветка выключена:

меню » Правка » Настройка запуска

Переходим в конец файла и включаем режим вставки

Ctrl + Down Ins Enter Enter

```
1 syntax on
2 colorscheme pablo
```

Выходим в режим команд и принудительно сохраняем

Esc: w ! Enter Enter

Выходим из (g)Vim

Esc: q ! Enter

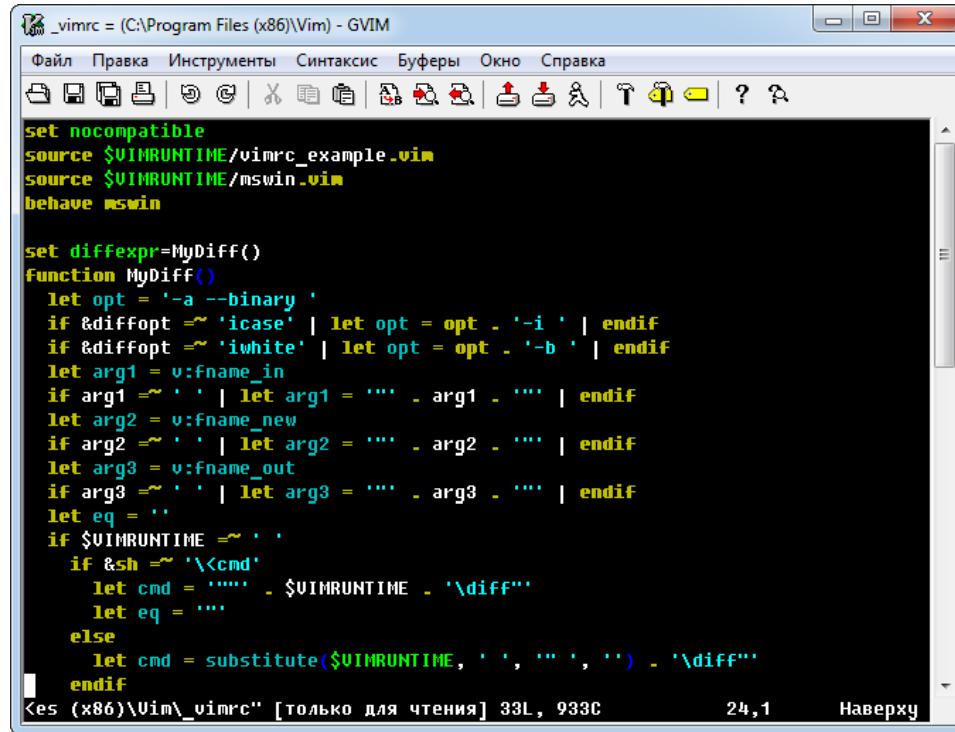
Если не получилось (под Windows 7):

Win + 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.



The screenshot shows a window titled '_vimrc = (C:\Program Files (x86)\Vim) - GVIM'. The menu bar includes 'Файл', 'Правка', 'Инструменты', 'Синтаксис', 'Буферы', 'Окно', and 'Справка'. Below the menu is a toolbar with various icons. The main text area contains Vim configuration code with syntax highlighting. The code defines a function MyDiff() which handles file comparison options like --binary, -i, -b, and -iwhite. It also sets up arguments for three files and constructs a command for the diff utility. The status bar at the bottom shows the path 'C:\Program Files (x86)\Vim_vimrc' [только для чтения] 33L, 933C, the line number '24,1', and the word 'Наверху'.

```
set nocompatible
source $VIMRUNTIME/vimrc_example.vim
source $VIMRUNTIME/mswin.vim
behave mswin

set diffexpr=MyDiff()
function MyDiff()
    let opt = '-a --binary '
    if &diffopt =~ 'icase' | let opt = opt . '-i ' | endif
    if &diffopt =~ 'iwhite' | let opt = opt . '-b ' | endif
    let arg1 = v:fname_in
    if arg1 =~ ' ' | let arg1 = "" . arg1 . ""'' | endif
    let arg2 = v:fname_new
    if arg2 =~ ' ' | let arg2 = "" . arg2 . ""'' | endif
    let arg3 = v:fname_out
    if arg3 =~ ' ' | let arg3 = "" . arg3 . ""'' | endif
    let eq =
if $VIMRUNTIME =~ ''
    if &sh =~ '\<cmd'
        let cmd = '''' . $VIMRUNTIME . '\diff'''
        let eq = ''
    else
        let cmd = substitute($VIMRUNTIME, ' ', '', '') . '\diff'''
    endif
es (x86)\Vim\_vimrc" [только для чтения] 33L, 933C 24,1 Наверху
```

41.3.2 Выход из (g)Vim

Esc : ! q Enter

41.3.3 Выход с автосохранением

[Esc] [Shift] + [Z] [Shift] + [Z]

41.3.4 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](#)

41.3.5 Переход в режим команд

[Esc]

41.3.6 Запись редактируемого файла

[Esc] : [w] [Enter]

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

[Esc] : [!] [w] [Enter]

41.3.7 Перезагрузка файла

Для перезагрузки возможно изменененного извне файла или отмены всех несохраненных изменений

`Esc : e Enter`

41.3.8 Отмена последних изменений (undo)

`Esc u u ...`

Глава 42

Make: управление сборкой проектов

Глава 43

VCS: системы контроля версий

43.1 CVS

43.2 Subversion

43.3 Git

43.3.1 GitHub

Глава 44

Вспомогательные скрипты на языке Python

Глава 45

Основы Си и C_+^+

45.1 Установка MinGW (win32)

45.2 Особенности C_+^+ в embedded

45.3 Сборка кросс-компилятора GNU toolchain

Глава 46

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 (на языке разметки, или `DDL: [D]ata [D]efinition [L]anguage`) описаны данные. И от вас требуется реализовать разбор такого файла, выделяя синтаксические структуры и элементы данных, чтобы в дальнейшем после их преобразования например записать текстовый файл в другом формате.

В таком виде хранятся результаты расчетных программ, работающих в пакетном режиме, данные с измерительных систем, поток данных с приемников GPS¹, очень популярный мета-формат XML со всеми его частными случаями типа HTML, XLIFF??, OpenDocument, тексты программ для станков с ЧПУ,...

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

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.

¹ протокол NMEA 0183

Подробнее эти техники рассмотрены в книгах??, особенно стоит отметить талмуд **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

<http://ds9a.nl/lex-yacc/cvs/lex-yacc-howto.html>

<http://alumni.cs.ucr.edu/~lgao/teaching/flex.html>

http://www.caps1.udel.edu/courses/cpeg421/2012/slides/Tutorial-Flex_Bison.pdf

46.1 Лексер и лексический анализ, утилита **flex**

Лéксер/сканер — программа или ее часть, которая

1. получает на вход исходные данные в виде сплошного потока одиночных символов,
2. группирует символы согласно набору правил (заданных **регулярными выражениями**) и
3. отдает на выходе символы, уже сгруппированные в **лексемы** или **токéны**.

Цель лексера — подготовить последовательность лексем для входа другой программы.

В самый простых случаях на лексер можно возложить простые преобразования текста.

Лексический анализ — процесс программного разбора входной последовательности символов² с целью получения на выходе последовательности групп символов — **токенов**, имеющих собственное смысловое значение³. Как правило, лексический анализ производится в соответствии набора правил определённого **формального, искусственного или компьютерного языка**.

Компьютерный 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 (лексёма), 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 потока⁴.

Рассмотрим обработку текстового файла: разделение текстового фрагмента на абзацы, запись в формате XLIFF для перевода в системе ABBYY SmartCAT, и обратной трансляции из XLIFF в \LaTeX -совместимое форматирование.

² например, такой как исходный код на одном из языков программирования

³ подобно группировке букв в слово

⁴ запись строки, числа и т. д.

Так как задача построения лексических анализаторов является стандартной задачей информатики, был разработан типовой инструмент: генератор лексических анализаторов **flex**. Эта программа транслирует описание лексера на своем высококоуровневом языке в фрагмент программы на языке Си/ C^+ или самостоятельную программу. Описание лексера прописывается в .lex-файле в формате:

определения, опции, декларации

%%

правила выделения токенов через регулярки

%%

сишный код

Комментарии поддерживаются сишные /* */ комментарии.

Опции %option

noyywrap отключает вызов лексером функции yywrap() при достижении конца текущего файла

main включение типовой функции main() вместо заданной пользователем

case-insensitive регистро-независимый лексический анализ, большие/маленькие буквы не различаются

yylineno в глобальной си-переменной yylineno доступен номер текущей строки

Формат определения имя определение:

digit [0-9]

number [\+\-\{}{0,1}{\}{digit}\+\.\{digit}\}*{}

Makefile

```
1 all: xliff.xliff Y.tex
2
3
4 xliff.xliff: X.tex tex2xliff
5 ____./tex2xliff < $< > $@ 
6 Y.tex: xliff.xliff xliff2tex
7 ____./xliff2tex < $< > $@ 
8
9 clean:
10 ____rm -f tex2xliff xliff2tex
11 ____rm -f xliff.xliff Y.tex
12
13 tex2xliff: tex2xliff.l tex2xliff.y
14 ____bison -d $@.y && \
15 ____flex -o $@.lex.c $@.l && \
16 ____g++ -o $@ $@.lex.c $@.tab.c
17
18 xliff2tex: xliff2tex.l xliff2tex.y
19 ____bison -d $@.y && \
20 ____flex -o $@.lex.c $@.l && \
21 ____g++ -o $@ $@.lex.c $@.tab.c
```

flex лексер

⁵ копипаста: <http://matt.might.net/articles/standalone-lexers-with-lex/>

```
1%option noyywrap
2
3%
4#include <iostream>
5#include <string>
6using namespace std;
7#define YYSTYPE string
8#include "tex2xliff.tab.h"
9%
10
11%%
12([a-zA-Z0-9]+\.)+(com|nz) { yyval = yytext; return URL; }
13
14\.[\n\ ]+ { yyval = "."; return SEP; }
15\& { yyval = "&"; return CHAR; }
16\n{2,22} { yyval = ""; return SEP; }
17\n { yyval = " "; return CHAR; }
18\\item { yyval = ""; return SEP; }
19.
{ yyval = yytext; return CHAR; }
20%%
```

46.2 Генератор синтаксических анализаторов **bison**

Синтаксический анализ — процесс анализа последовательности токенов с определением их грамматической структуры. На этом этапе выделяются синтаксические ошибки.

bison парсер

```
1
2 %{
3 #include <iostream>
4 #include <string>
5 using namespace std;
6 #define YYSTYPE string
7 #define YYINITDEPTH 0x10000
8 void yyerror(const char *str) { cerr << "\nerror:" << str << "\n\n"; }
9 extern int yylex();
10 %}
11
12 %token CHAR
13 %token SEP
14 %token URL
15
16 %%
17 BLOCK: CHARz | CHARz BLOCK ;
18 CHARz:
19     CHAR    { cout<<$$; } |
20     URL     { cout<<"<mrk mtype=\"protected\" comment=\"url\">>"<<$$<<"</mrk>"; } |
21     SEP     {
22         cout<<$$<<"</source></trans-unit>\n";
23         cout<<"<trans-unit><source>";
24     } ;
25 %%
26
27 int main () {
```

```
28 cout << "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n";
29 cout << "<xliff xmlns=\"urn:oasis:names:tc:xliff:document:1.2\" version=\"1.2\">\n";
30 cout << "<file>\n";
31 cout << "<body>\n<trans-unit><source>";
32 int yyp=yparse();
33 cout << "</source></trans-unit>\n</body>\n";
34 cout << "</file>\n";
35 cout << "</xliff>\n";
36 return yyp;
37 }
```

Глава 47

LLVM и разработка собственных компиляторов

47.1 Установка под Windows

http://llvm.org/ > Download > LLVM 3.2 > Experimental Clang Binaries for Mingw32/x86

Распакуйте clang+llvm-3.2-x86-mingw32-EXPERIMENTAL.tar.gz, переименовав в C:/LLMV, добавьте C:/LLMV/bin в PATH.

47.2 Создание компилятора с помощью инфраструктуры LLVM

¹ © IBM

¹ копипаста: <http://www.ibm.com/developerworks/ru/library/os-createcompilerllvm1/>

Инфраструктура LLVM³ предоставляет мощные возможности для создания оптимизирующих компиляторов вне зависимости от используемого языка программирования. Это весьма мощная инфраструктура компилятора, предназначенная для оптимизации программ, написанных на предпочтительном для разработчика языке программирования, на этапах компиляции, связывания и исполнения. Инфраструктура LLVM работает на нескольких разных платформах. Ее основное достоинство — генерация кода, который исполняется с высокой скоростью.

В основе инфраструктуры LLVM лежит хорошо документированное промежуточное представление⁴ программного кода. При наличии кодогенератора LLVM IR разработчику достаточно иметь т.н. фронтенд для своего языка программирования, чтобы получить полную систему парсер (фронтенд + генератор IR-кода + LLVM-бэкенд). Таким образом, построение собственного компилятора существенно упростилось.

Для разработчика компилятора важны две ключевые особенности LLVM:

1. LLVM содержит собственную батарею оптимизаторов, разрабатываемую и поддерживаемую большим сообществом компиляторщиков-суперпрофессионалов;
2. поддерживается генерация машинного кода для множества целевых архитектур, поэтому создание своего кодогенератора не нужно.

47.3 Инструменты **l2c** и **lli**

LLVM — это виртуальная машина и, как таковая, она имеет собственное представление программы в виде промежуточного байт-кода. В конечном итоге нам необходимо скомпилировать байт-код LLVM в код на языке

² копипаста: <http://habrahabr.ru/post/47878/>

³ [L]ow [L]evel [V]irtual [M]achine, низкоуровневая виртуальная машина

⁴ intermediate representation, IR

ассемблера для своей конкретной платформы. После этого мы сможем запустить этот код с помощью нативных ассемблера и компоновщика для этой платформы с целью генерации исполняемого кода, разделяемых библиотек и т. д. Для преобразования байт-кода LLVM в код на языке ассемблера для конкретной платформы применяется инструмент `llc`.

Возможность непосредственного исполнения порций байт-кода LLVM избавляет нас от необходимости дожидаться сбоев нативного исполняемого кода, чтобы найти ошибки в своей программе. Именно здесь оказывается полезным инструмент `lli`, поскольку он способен непосредственно выполнять байт-код (с помощью интерпретатора).

47.4 Семантический анализ

Семантический анализ — процесс выполнения семантических проверок: контроль типов, привязка объектов и т.д.

47.5 Оптимизация

Выполнение формальных преобразований структур данных, описывающих компилируемую программу, с целью построения более компактного/быстрого машинного кода.

47.6 Кодогенерация

Получение реального машинного кода в бинарном представлении, или в виде ассемблерных текстовых файлов.

47.7 Транслятор Паскаля

Часть X

Микроконтроллеры Cortex-Mx

Глава 48

Производители

48.1 ST Microelectronix STM32

48.2 LPC

48.3 Миландр

Глава 49

Отладочные платы

49.1 LeafLabs Maple Mini: STM32F103 /Cortex-M3/

49.2 Серия STM32 STM32DISCOVERY

49.2.1 STM32DISCOVERY: STM32F103 /Cortex-M3/

49.2.2 STM32F4DISCOVERY: STM32F406 /Cortex-M4F/

49.2.3 STM32F0DISCOVERY: STM32F040 /Cortex-M0/

Часть XI

ПЛИС

¹ копипаста: <http://habrahabr.ru/post/250511/>

Часть XII

USB

Многообразные порты, унаследованные от компьютеров IBM PC и PS/2, уходят в прошлое. Будущее, да и настоящее, принадлежит универсальным скоростным портам USB. Об удобствах, которые USB предоставляет простым пользователям, распространяться не приходится. Единый интерфейс для всех устройств, обладающий возможностями Plug'n'Play и продвинутого управления питанием — именно то, что нужно пользователям, для которых компьютер — часть бытовой техники. Другое дело — индивидуальные разработчики различных устройств и просто хакеры. Для этих категорий переход на USB представляет определенные сложности. Проблема заключается в том, что USB — интеллектуальный интерфейс.

Любое устройство, предназначенное для подключения к компьютеру через USB, должно поддерживать хотя бы небольшую часть спецификации протокола USB: уметь «представиться»² и адекватно реагировать на стандартные сообщения USB, посылаемые компьютером. В результате, даже устройство, все функции которого ограничиваются включением и выключением светодиода по сигналу с компьютера, при подключении через USB требует наличия микросхемы, которая умеет «разговаривать» с хостом.

Однако и для разработчиков собственных устройств переход на USB несет определенные преимущества. Прежде всего, упрощается процесс написания драйверов. Поскольку для общения с компьютером все USB устройства используют единый протокол, причем протокол этот абстрагирован от таких аппаратно-зависимых вещей как отображенные в память порты и прерывания, возникает возможность не писать свой собственный драйвер уровня ядра для каждого устройства. Вместо этого целые группы устройств могут использовать один и тот же драйвер уровня ядра, а специфичный код, учитывающий особенности конкретного устройства, может быть размещен на пользовательском уровне. При этом драйвер уровня ядра берет на себя такие функции как управление питанием устройства³, оставляя нам самое интересное — управление функциями устройства.

Перенос кода управления устройством в `userspace` не только упрощает отладку⁴, но и позволяют писать процедуры управления устройством на самых разных языках программирования, а не только на Си, как это делают те, кто пишет драйверы уровня ядра. Более того, пользовательская часть драйвера, не взаимодействуя

² предоставить информацию о себе и своих возможностях

³ весьма нетривиальная задача с учетом того, что сам компьютер может переключаться между несколькими энергосберегающими режимами

⁴ при падении приложения, скорее всего, не придется перезагружать машину

ющая напрямую с механизмами ядра операционной системы, может быть сделана кросс-платформенной, что мы и имеем в случае таких инструментов как `libusb`. Благодаря `libusb` даже многие устройства промышленного уровня могут обходиться без собственных драйверов на уровне ядра для каждой ОС, и иметь одну кодовую базу, которую проще модифицировать и сопровождать. Что уж говорить о любительских устройствах?

Глава 50

Стек протоколов USB

Протокол USB похож на стек сетевых протоколов, основанных на TCP/IP, точнее за основу принималась сетевая модель стека сетевых протоколов OSI/ISO¹:

Уровень	Layer	Тип данных	Функции
L1 Физический	Physical	Биты	Работа со средой передачи, сигналами и двоичными данными
L2 Канальный	Data link	Кадры	Физическая адресация
L3 Сетевой	Network	Пакеты/Датаграммы	Определение маршрута и логическая аресация
L4 Транспортный	Transport	Сегменты	Прямая связь между конечными пунктами и надежность
L5 Сеансовый	Session	Сеансы	Управление сеансом связи
L6 Представления	Presentation	Поток	Представление и шифрование данных
L7 Прикладной	Application	Данные	Доступ к сетевым службам

¹ ISO/IEC 7498-1, ГОСТ Р ИСО/МЭК 7498-1-99

На самом нижнем логическом уровне² устройства обмениваются пакетами данных³. Из пакетов формируются запросы, которые устройства посылают друг другу. Запросы составляют блоки запросов [U]SB [R]equest [B]lock, **URB**.

Протокол USB является “хостоцентричным” — процесс передачи данных всегда инициируется хостом (то есть, компьютером). Если у периферийного устройства появились данные для передачи хосту, оно должно ожидать запроса хоста на передачу данных. Соответственно мы имеем следующие типы USB устройств:

host хост-контроллер выполняет функции головного узла: инициирует передачу данных, управляет питанием
client

OTG [O]n-[T]he-[G)o, **OTG** — расширение спецификации USB 2.0, предназначенное для лёгкого соединения периферийных USB-устройств друг с другом без необходимости подключения к хосту. Например, цифровой фотоаппарат можно подключать к фотопринтеру напрямую, если они оба поддерживают стандарт USB OTG. К моделям КПК и коммуникаторов, поддерживающих USB OTG, можно подключать некоторые USB-устройства. Обычно это флэш-накопители, цифровые фотоаппараты, клавиатуры, мыши и другие устройства, не требующие дополнительных драйверов. При подключении через USB OTG ранг устройства (ведущий или ведомый) определяется наличием или, соответственно, отсутствием перемычки между контактами 4 (ID) и 5 (Ground) в штекере соединительного кабеля. В USB OTG кабеле такая перемычка устанавливается лишь в одном из двух разъёмов.

² спецификации физического уровня мы пока не рассматриваем

³ со встроенными механизмами коррекции ошибок, подтверждения получения и т.д

Глава 51

libUSB

51.1 драйвер для устройства USB

¹

¹ копипаста: <http://symmetrica.net/usb/usb1.htm>

Глава 52

Поддержка USB в Linux

52.1 Опции ядра

52.1.1 режимы host/client/otg и хост-контроллеры xHCI

52.1.2 data storage: носители данных

52.1.3 hid: клавиатура, мышь, джойстик

52.1.4 USB-периферия: сеть, звук,...

52.2 Настройка hotplug и автомонтирования USB носителей

52.3 Сборка и настройка libusb

52.4 Примеры программ низкоуровневого ввода/вывода

Часть XIII

Встраиваемый emLinux

Linux для встраиваемых систем¹ — 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 строковых 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 процессоров²;
 - Значительные требования по объему ОЗУ 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 устройств³, и устройств интенсивно использующих сетевые протоколы (роутеры, медиацентры).

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

¹ будем называть его emLinux

² 32-бит, необходим блок MMU

³ в т.ч. является основой Android

Глава 53

Загрузчик syslinux

Самый простой и удобный загрузчик для i386-систем, ставится на флешку из под Windows, работает с FAT-разделами, поддерживает загрузку с флешек, CDROM и по сети.

<http://www.syslinux.org/>

53.1 Закачка

.zip с бинарной сборкой **syslinux**:

<https://www.kernel.org/pub/linux/utils/boot/syslinux/syslinux-6.03.zip>

memtest86+ — полезная утилита для тестирования ОЗУ:

<http://www.memtest.org/download/5.01/memtest86+-5.01.zip>

Если планируете устанавливать рабочую станцию для сборки azLinux с флешки, нужно скачать полные или **netinst** установочные .iso-образы:

[Установочный образ Debian Linux i386:](#)

<http://cdimage.debian.org/debian-cd/7.7.0/i386/iso-cd/debian-7.7.0-i386-netinst.iso>

[Установочный образ Debian Linux amd64:](#)

<http://cdimage.debian.org/debian-cd/7.7.0/amd64/iso-cd/debian-7.7.0-amd64-netinst.iso>

[Сборка HDD-инсталлятора i386:](#)

<http://http.us.debian.org/debian/dists/wheezy/main/installer-i386/current/images/hd-media/initrd.gz>

<http://http.us.debian.org/debian/dists/wheezy/main/installer-i386/current/images/hd-media/vmlinuz>

[Сборка HDD-инсталлятора amd64:](#)

<http://http.us.debian.org/debian/dists/wheezy/main/installer-amd64/current/images/hd-media/initrd.gz>

<http://http.us.debian.org/debian/dists/wheezy/main/installer-amd64/current/images/hd-media/vmlinuz>

53.2 Установка под Windows на флешку

Распакуйте файлы из .zip

/bios/win32/syslinux.exe консольный инсталлятор

/bios/com32/menu/menu.c32 модуль текстового меню

/bios/com32/menu/vesamenu.c32 модуль графического меню (VESA)
служебные библиотеки syslinux

/bios/com32/libutil/libutil.c32

/bios/com32/lib/libcom32.c32

Для установки syslinux на флешку с FAT, на которую назначена буква F:, выполните батник:

/syslinux/syslinuxusb.bat

```
1 syslinux -i -m -a -d syslinux F:  
2 pause
```

-i install установить
-m MBR в MBR
-a active сделать раздел активным
-d directory в каталог **syslinux**

Если ставите Debian, распакуйте из **debian-netinst.iso** в **F:/Debian/**

debian-7.7.0-amd64-netinst.iso	.iso-образ установочного CD-ROM
debian-7.7.0-i386-netinst.iso	.iso-образ установочного CD-ROM
hd-media/install.amd/vmlinuz	ядро amd64 (x64)
hd-media/install.amd/initrd.gz	ramdisk с инсталляром
hd-media/install.386/vmlinuz	ядро i386 (x32)
hd-media/install.386/initrd.gz	ramdisk с инсталляром

53.3 **syslinux.cfg**

syslinux настраивается текстовым файлом **syslinux.cfg**.

/syslinux/syslinux.cfg

```
1 UI vesamenu.c32  
2 MENU RESOLUTION 640 480  
3 MENU TITLE azLinux  
4 MENU BACKGROUND /syslinux/splash640x480.png
```

```
5
6 DEFAULT azmicro
7 LABEL azmicro
8 MENU LABEL azLinux micro
9 KERNEL /azLinux/micro.kernel
10 INITRD /azLinux/micro.initrd
11 APPEND vga=none
12
13 LABEL azclock
14 MENU LABEL azLinux clock
15 KERNEL /azLinux/clock.kernel
16 INITRD /azLinux/clock.initrd
17 APPEND vga=ask
18
19 LABEL memtest
20 MENU LABEL memtest86+
21 KERNEL /syslinux/memtest.krn
22
23 LABEL debian64
24 MENU LABEL Debian GNU/Linux 7.7.0–amd64–netinst
25 KERNEL /Debian/amd64/vmlinuz
26 INITRD /Debian/amd64/initrd.gz
27 APPEND vga=none
28
29 LABEL debian32
30 MENU LABEL Debian GNU/Linux 7.7.0–i386–netinst
31 KERNEL /Debian/i386/vmlinuz
32 INITRD /Debian/i386/initrd.gz
```

В примере показана реализация с использованием графического VESA меню. Для использования более надежного текстового меню замените на UI menu.c32.

Обратите внимание на возможность включения нестандартных видеорежимов используя **[VESA]MENU RESOLUTION**: этот фильтр нужен для включения графики на ASUS EeePC 701: режим 800×480 недоступен для включения через параметр ядра **vga=**, поэтому приходится использовать возможности **syslinux**.

Глава 54

azLinux

Чтобы разобраться как можно собрать встраиваемый 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/Turbo-Pascal на что-то

- более переносимое: на энергоэффективное ARM/MIPS-железо, в т.ч. (типа)отечественного производства,
- стабильное: с полноценной многозадачностью, зашитой памяти и данных, и
- позволяющее использовать максимум возможностей аппаратуры: большая ОЗУ, 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 ОЗУ с любых носителей](#), в т.ч. [заблокированных на запись](#). Это важно для случаев, когда возможны внезапные выключения питания: вся система работает в ОЗУ-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 пользовательского 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:

- ядро (реального времени)
- μ libc
- урезанная командная оболочка ([busybox](#))

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

- 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 ресурсами². Интерактивная работа с пользователем также не предполагается, доступна только командная консоль³, 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??**.

54.1 Требования к системе сборки (BUILD-хост)

Требования жесткие — 2x-ядерный процессор, 2+ Гб ОЗУ, для 4+ Гб ОЗУ нужен 64x-битный дистрибутив Linux (рекомендую Debian), и естественно никаких виртуалок. Возможна установка системы на флешку, в этом случае требования к ОЗУ еще более ужесточаются — потребуется каталоги с временными файлами смонтировать как tmpfs:

добавить в /etc/fstab

```
1 tmpfs /home/user/Azbuka/azlin/tmp    tmpfs      auto , uid=user , gid=user  0 0
2 tmpfs /home/user/Azbuka/azlin/src    tmpfs      auto , uid=user , gid=user  0 0
3 tmpfs /home/user/.ccache           tmpfs      auto , uid=user , gid=user  0 0
```

Можно попытаться сделать билд-сервер и на худшем железе, но будьте готовы к тормозам или внезапному окончанию памяти — ресурсоемка сборка тяжелых библиотек типа libQt или крупных пакетов типа gcc.

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

³ ее можно считать сервисным режимом

Вы можете попробовать поставить Linux на виртуалку, на флешку, и на жесткий диск (если найдете место) и оценить возможности этих вариантов на сборке пакета `gcc0`. При сборке с флешки на ноутбуке с 2 ГБ ОЗУ мне для сборки `gcc0` пришлось временно размонтировать `cross/src`, сделать `./mk.rc && make gcc ramclean`, а потом примонтировать `tmpfs` опять на `src`.

Сборка под MinGW/Cygwin совершенно неживая. Если совсем никак без винды — используйте виртуалки, и будьте готовы ждать.

54.2 Понятие пакет

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.

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
2cat \
3    mk/head.mk \
4    mk/dirs.mk \
5    mk/versions.mk \
6    mk/packages.mk \
7    mk/commands.mk \
8    mk/clean.mk \
9    mk/gz.mk \
10   mk/src.mk \
11   mk/cfg.mk \
12   mk/cross.mk \
13   mk/core.mk \
14   mk/kernel.mk \
15   mk/ulibc.mk \
16   mk/busybox.mk \
17   mk/sdk/canadian.mk \
```

```
18  mk/sdk/python.mk \
19  mk/sdk/pascal.mk \
20  mk/sdk/math.mk \
21  mk/user.mk \
22  mk/libs.mk \
23  mk/apps.mk \
24  mk/emu.mk \
25  mk/root.mk \
26  mk/boot.mk \
27  mk/info.mk \
28 > Makefile
```

54.3 Клонирование проекта azLinux

При необходимости вносить правки⁴ 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
```

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

54.4 Общий порядок сборки

Каждый пакет собирается командой:

```
./mk.rc && make [HW=rpi] [APP=clock] <package>
```

1. `dirs` создание дерева каталогов 54.6
2. `gz` закачка архивов исходников 54.7
3. `tc` сборка кросс-компилятора 54.14.6
 - (a) `binutils` ассемблер, линкер и утилиты 54.14.7
 - (b) `cclibs` библиотеки для сборки `gcc` 54.14.8
 - (c) `gcc0` сборка минимального кросс-компилятора Си 54.14.9
4. `core` сборка основной системы 54.14.11
 - (a) `kernel` ядро Linux 54.14.12
 - (b) `ulibc` библиотека `uClibc` 54.14.13
 - (c) `busybox` набор утилит `busybox` 54.14.15
 - (d) `gcc` пересборка полного кросс-компилятора Си/ C_+ 54.14.14
5. `libs` сборка библиотек `LIBS` 54.14.16
6. `apps` сборка прикладных пакетов `APPS` 54.14.17
7. `user` сборка пользовательского кода 54.14.18

8. **root** формирование корневой файловой системы 54.14.19
9. **boot** сборка загрузчика 54.14.20 *syslinux/grub/uboot*
10. **emu** запуск собранной системы в эмуляторе 54.14.24
11. **netboot** сетевая загрузка 54.15
12. **firmware** прошивка на устройство 54.16

54.5 Фиксация переменных

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

```
./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.

54.6 **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
-rwxr-xr-x 1 user user    121 Дек 8 11:43 mk.rc
drwxr-xr-x 2 user user   4096 Дек 8 11:43 mk
drwxr-xr-x 2 user user   4096 Дек 8 11:43 app
drwxr-xr-x 2 user user   4096 Дек 8 11:43 hw
drwxr-xr-x 2 user user   4096 Дек 8 11:43 cpu
drwxr-xr-x 2 user user   4096 Дек 8 11:43 arch
drwxr-xr-x 2 user user   4096 Дек 8 13:26 gz
drwxr-xr-x 2 user user   4096 Дек 8 11:43 app
drwxr-xr-x 3 user user   4096 Дек 8 13:26 qemu386micro
drwxr-xr-x 2 user user   4096 Дек 8 13:26 qemu386micro.cross
-rw-r--r-- 1 user user    147 Дек 8 11:43 README.md
-rw-r--r-- 1 user user  17699 Дек 8 13:25 azlin.tex
drwxr-xr-x 2 user user   4096 Дек 8 13:26 src
drwxr-xr-x 2 user user   4096 Дек 8 13:26 tmp
-rw-r--r-- 1 user user   2087 Дек 8 13:26 Makefile
```

Пакет dirs прописан в файле

mk/dirs.mk

```
1 # directories processing
2 # sw sources mirror archive in .tar.[GZ]
3 GZ = $(PWD)/gz
4 # [S]ources [RC]ode unpacked
5 SRC = $(PWD)/src
6 # [T]e[MP] build dirs
7 TMP = $(PWD)/tmp
```

```
8 # build/target triplets
9 BUILD = $(shell gcc --dumpmachine)
10 # target root filesystem
11 ROOT = $(PWD)/$(HW)$(APP)
12 # cross-compiler [T]ool [C]hain
13 TC = $(ROOT).cross
14 BOOT = $(ROOT)/boot
15 ETC = $(ROOT)/etc
16 USR = $(ROOT)/usr
17 USRBIN = $(USR)/bin
18 USRLIB = $(USR)/lib
19 PACK = $(ROOT)/pack
20 ISO = $(TMP)/iso
21 DIRS = $(GZ) $(SRC) $(TMP) $(TC) $(ROOT) $(BOOT) $(USR) $(USRBIN) $(USRLIB) $(PACK)
22 .PHONY: dirs
23 dirs:
24     mkdir -p $(DIRS)
```

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

Каталог зеркала архивов исходных текстов программ

GZ

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

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

SRC

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

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

TMP

```
1 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

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

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

TC

```
1 TC = $(ROOT).cross
2 TC = $(ROOT)/etc
```

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 **HW54.9** и имени приложения **APP54.8**.

ROOT/BOOT

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

Список всех рабочих каталогов в одной переменной:

DIRS

```
1 DIRS = $(GZ) $(SRC) $(TMP) $(TC) $(ROOT) $(BOOT) $(USR) $(USRBIN) $(USRLIB) $(PACK)
```

54.7 gz: Загрузка архивов исходников

mk/gz.mk

```
1 # download sw packages sources to gz/
2
3 .PHONY: gz
4 gz:
5 #__exit -1
6 ____make gz_cc
7 ____make gz_core
8 ____make gz_libs
9 ____make gz_sdk
10 ____make gz_$(ARCH)
11 ____make gz_apps
12
13 .PHONY: gz_cc
14 gz_cc:
15 ____$(WGET) http://ftp.gnu.org/gnu/binutils/$(BINUTILS).tar.bz2
16 ____$(WGET) http://gcc.skazkaforyou.com/releases/$(GCC)/$(GCC).tar.bz2
17 ____$(WGET) ftp://ftp.gmplib.org/pub/gmp/$(GMP).tar.bz2
18 ____$(WGET) http://www.mpfr.org/mpfr-current/$(MPFR).tar.bz2
19 ____$(WGET) http://www.multiprecision.org/mpc/download/$(MPC).tar.gz
20
21 .PHONY: gz_core
22 gz_core:
23 ____$(WGET) https://www.kernel.org/pub/linux/kernel/v3.x/$(KERNEL).tar.xz
24 ____$(WGET) http://www.uclibc.org/downloads/$(ULIBC).tar.xz
25 ____$(WGET) http://busybox.net/downloads/$(BUSYBOX).tar.bz2
```

```
26
27 .PHONY: gz_libs
28
29 gz_libs:
30     $(WGET) https://www.libsdl.org/release/$(SDL).tar.gz
31     $(WGET) https://www.libsdl.org/projects/SDL_image/release/$(SDL_IMAGE).tar.gz
32     $(WGET) http://www.libsdl.org/projects/SDL_ttf/release/$(SDL_TTF).tar.gz
33     $(WGET) http://download.sourceforge.net/libpng/$(PNG).tar.xz
34     $(WGET) http://download.savannah.gnu.org/releases/freetype/$(FREETYPE).tar.bz2
35     $(WGET) http://zlib.net/$(ZLIB).tar.xz
36     $(WGET) http://cxx.uclibc.org/src/$(LIBCPP).tar.xz
37
38 .PHONY: gz_sdk
39
40 gz_sdk: gz_python
41     $(WGET) ftp://ftp.hu.freepascal.org/pub/fpc/dist/$(FPC_VER)/source/$(FPC).source.tar.gz
42     $(WGET) ftp://ftp.hu.freepascal.org/pub/fpc/dist/$(FPC_VER)/source/fpcbuild-$(FPC_VER).tar.gz
43     $(WGET) http://downloads.sourceforge.net/project/ecls/ecls/$(ECL_VER_A)/$(ECL).tgz
44
45 .PHONY: gz_python
46
47 gz_python:
48     exit -1
49     $(WGET) https://www.python.org/ftp/python/$(PYTHON_VER)/$(PYTHON).tar.xz
50     $(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/001-patch-001.patch
51     $(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/002-patch-002.patch
52     $(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/004-patch-004.patch
53     $(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/005-patch-005.patch
54     $(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/006-patch-006.patch
55     $(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/007-patch-007.patch
56     $(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/008-patch-008.patch
```

```
54 ____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/009--n
55 ____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/010-f
56 ____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/011-r
57
58 .PHONY: gz_i386
59 gz_i386:
60 ____$(WGET) https://www.kernel.org/pub/linux/utils/boot/syslinux/$(SYSLINUX).zip
61
62 .PHONY: gz_arm
63 gz_arm:
64 ____$(WGET) ftp://ftp.denx.de/pub/u-boot/$(UBOOT).tar.bz2
65
66 .PHONY: gz_apps
67 gz_apps:
68 ____$(WGET) http://elinks.or.cz/download/$(ELINKS).tar.bz2
```

54.8 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

```
1 APP = clock
```

Если вам нужно собрать другое приложение, вы можете переопределить значение из командной строки при запуске [всех](#) пакетов:

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

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

- **LIBS**: набор используемых библиотек
- **PACKS**: набор используемых программных пакетов 54.14

app/micro.mk

```
1 # app: micro
2 LIBS =
3 APPS = hello
```

app/clock.mk

```
1 # app: clock
2 LIBS = zlib png sdl
3 APPS = hello $(USRBIN)/sdl_hello $(USRBIN)/sdl_rect $(USRBIN)/sdl_clock
```

54.9 HW: Поддерживаемое железо

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

HW	CPU	ARCH	RAM	HD	SD	USB	Eth	WiFi	GPIO
qemu386 ??	i486sx	i386	32M+	<input type="checkbox"/> IDE		<input type="checkbox"/>	ne2k		
eeepc701 54.10.2	CeleronM	i386	512M+	<input type="checkbox"/> SSD	<input type="checkbox"/> SD	<input checked="" type="checkbox"/>	A??	<input type="checkbox"/> AR2425	
gac1037 ??	Celeron1037U	x86_64	1G+	<input type="checkbox"/> SATA		<input checked="" type="checkbox"/>	2×RTL8111		
qemuARM		arm	32M+						
cubie1 54.11.2	AllWinnerA10	armhf	1G		<input type="checkbox"/> μSD	<input checked="" type="checkbox"/>			
rpi 54.11.3	BCM2835	armel	512M		<input type="checkbox"/> SD	<input checked="" type="checkbox"/>			
tion 54.11.4	PXA270	arm??							
mb77 54.11.5	K1879ХБ1Я	arm??							
qemuMIPS		mips	32M+						
mr3020 ??	AR7240	mips	32M	4M		<input checked="" type="checkbox"/>		<input type="checkbox"/> AR9331	
vocore 54.12.3	RT5350	mips	32M	8M		<input type="checkbox"/>		<input type="checkbox"/> SoC	
bswift		AR9331	mips	64M	16M	<input type="checkbox"/>			20+

`mk/head.mk`

```
1 # [H]ard [W]are: qemu386 qemuARM qemuMIPS cubie1 rpi ...
2 HW = x86
3 # [APP]lication: micro clock
4 APP = clock
5 # load extra hw definitions: ARCH CPU ...
6 include hw/$(HW).mk
7 # load extra defs for CPU setted in $(HW).mk
8 include cpu/$(CPU).mk
9 # load extra defs for ARCHitecture setted in $(CPU).mk
```

```
10 include arch/$(ARCH).mk
11 # load extra app defs: LIBS APPS ...
12 include app/$(APP).mk
13
14 .PNONY: all
15 all:
16     make dirs
17     make gz
18     make cross0
19     make ramclean
20     make kernel
21     make ulibc
22     make busybox
23     make libs
24     make ramclean
25     make apps
26     make root
```

54.10 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.

54.10.1 qemu386: эмулятор QEMU

hw/qemu386.mk

```
1 # QEMU emulator: i386 mode
2 CPU = i486sx
3 QEMU_CFG = -m 64M -net none -localtime
4 #vga=ask
5 ##0x312 640x480x24
6 ##0x315 800x600x24
```

54.10.2 eeepc701: ASUS Eee PC 701



CPU	CeleronM
OЗУ	512
SSD	4G
видео	<input checked="" type="checkbox"/> Intel HDA, VGA DSUB
Ethernet	<input checked="" type="checkbox"/> 10/100M
WiFi	<input checked="" type="checkbox"/>
USB	<input checked="" type="checkbox"/> USB1.1 host
SD	<input checked="" type="checkbox"/>

hw/eeepc701.mk

1 # ASUS Eee PC 701

2 CPU = CeleronM

54.10.3 gac1037: Gigabyte GA-C1037UN-EU rev.2



CPU	Celeron 1037U
OЗУ	2 × DDR3, max 16G, 2 канала
чипсет	Intel NM70
сеть	2 × Realtek® GbE 1Gb (rtl8111)
HDD	2 × SATA2 (3Gb/s), 1 × SATA3 (6Gb/s), 1 × eSATA
USB	6 × USB3.0
видео	IntelGMA, выходы на VGA D-Sub и HDMI 1.4
аудио	Realtek ALC887 (HDA)
PCI	×1

В качестве варианта 64-битной платформы взята портативная материнка для неттопов: Gigabyte GA-C1037UN-EU. На ней возможны два варианта сборки:

1. x86_64/x64: нативный CPU=Celeron1037u
2. i686/x32: режим совместимости со старыми процессорами CPU=i686

На текущий момент эта материнка — оптимальный вариант для офисного, рабочего неигрового компьютера, или базы для изготовления мобильной рабочей станции в миникейсе: комплект из GA-C1037UN-EU и блока питания стоит порядка 5 тыс.руб, при этом возможна установка до 2×8G ОЗУ, что пока недоступно на дешевых ноутбуках. Но — **ОТВРАТИТЕЛЬНЫЙ** радиатор на мосте NM70, нагрев до 70°C , в обязательном порядке делать сквозную продувку корпуса до включения материнки.

материнская плата	GA-C1037UN-EU rev.2
CPU	Celeron 1037U (впаян)
охлаждение	отвратительное, обязательны кулера на все чипы и сквозная продувка корпуса,
OЗУ	1×8G DDR3 (в планах 2×8G)
HDD	нет, используется флешка 8G USB3 Transcend JF750 (возможно SATA SSD)
TFT	китайский 3.5" автомониторчик + конвертер HDMI2RCA на случай посмотреть видеовывод, обычно везде где я использую эту поделку, есть VGA монитор или хотя бы большой телевизор
электропитание	БП Hipro HPE-350W + автоинвертор батарея не требуется, под боком всегда есть 220 или 12 В по необходимости в транспорт грузится пара заряженных автоаккумуляторов

hw/gac1037.mk

```
1# Gigabyte GA-C1037UN-EU
2CPU = Celeron1037U
```

54.11 ARM

54.11.1 qemuARM: эмулятор QEMU

54.11.2 **cubie1: Cubie Board v.1**

54.11.3 **rpi: Raspberry Pi model B**

54.11.4 тион: ТионПро270

ЗАО “Завод Электрооборудования”

<http://www.zao-zeo.ru/catalog/sbc/67-tion-pro270>

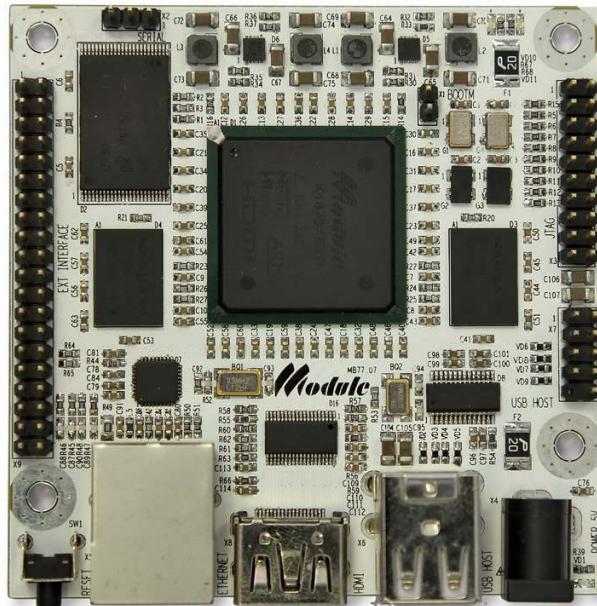


CPU	Marvell XScale PXA270 @ 416/520 МГц
OЗУ	64
Flash	32
видео	<input checked="" type="checkbox"/> VGA DSUB
Ethernet	<input checked="" type="checkbox"/> 10/100M
USB	<input checked="" type="checkbox"/> USB1.1 host
μ SD	<input checked="" type="checkbox"/>

54.11.5 mb77: Микрокомпьютер MB77.07 на базе СБИС K1879ХБ1Я

ЗАО НТЦ "Модуль"

http://www.module.ru/catalog/micro/micro_pc/



CPU

K1879ХБ1Я

ARM1176JZF-S @ 324 МГц

GPU

NeuroMatrix NMC3 @ 324 МГц

OЗУ

Flash

видео

HDMI

Ethernet

10/100M

USB

USB2 host

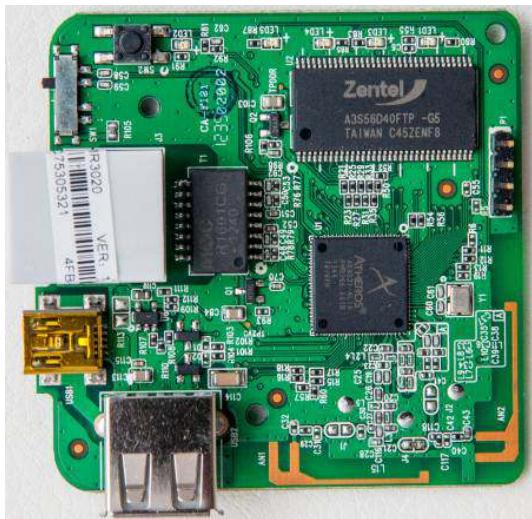
54.12 MIPS

54.12.1 qemuMIPS: эмулятор QEMU

hw/qemuMIPS.mk

54.12.2 mr3020: роутер MR3020

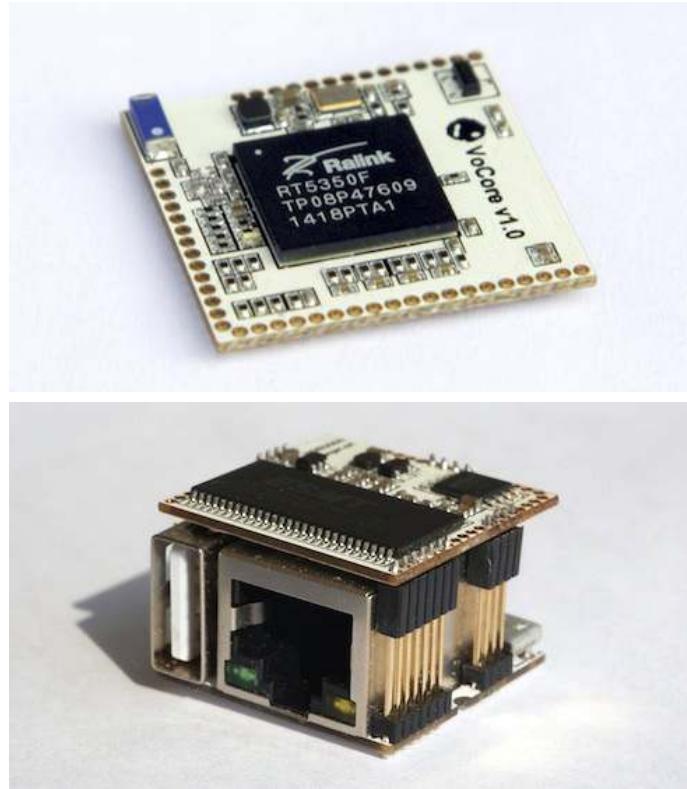
<http://wiki.openwrt.org/ru/toh/tp-link/tl-mr3020>



CPU	Atheros AR9330 rev.1 @ 400 MHz
O3Y	32
Flash	4
WiFi	<input checked="" type="checkbox"/> 802.11 b/g/n 150Mbps
Ethernet	<input checked="" type="checkbox"/> 10/100M
USB	<input checked="" type="checkbox"/> USB2

54.12.3 vocore: VoCore

vocore



CPU	SoC Ralink RT5350F /MIPS 24KEc/ @ 360 MHz
O3Y	32M
Flash	16M
WiFi	802.11n 1T/1R (1x1:1) 2.4 GHz 150Mbps MAC/BB/PA/RF
Ethernet	☒
USB	☒
GPIO	
μSD	☒

54.12.4 bswift: BlackSwift

bswift

54.13 CPU: Конфигурации процессоров

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

<code>ARCH</code>	архитектура целевой системы, используется при конфигурировании ядра
<code>TARGET</code>	триплет целевой системы, параметр задает тип целевой системы при сборке кросс-компилиатора и используется во всех скриптах <code>configure</code> при сборке остальных пакетов
<code>CFG_CPU</code>	параметры при сборке кросс-компилиатора
<code>CPU_FLAGS</code>	параметры <code>gcc</code> для оптимизации кода

54.13.1 i386

`cpu/i486sx.mk`

```
1 # i486sx cpu
2 # target arch is Intel x86 (32 bit)
3 ARCH = i386
4 # target triplet for embedded linux
5 TARGET = i486-linux-uclibc
6 # target CPU configure params
7 CFG_CPU = --disable-mmx --disable-3dnow --disable-sse
```

`cpu/CeleronM.mk`

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

`cpu/Celeron1037U.mk`

```
1# Intel Celeron 1037U
2ARCH = i386
3TARGET = celeronm-linux-uclibc
```

54.13.2 ARM

54.13.3 MIPS

cpu/AR7240.mk

```
1# Atheros AR7240 CPU (400Mhz)
2ARCH = mips
3TARGET = mips-linux-uclibc
```

cpu/RT5350.mk

```
1# Ralink RT5350 360MHz
2ARCH = mips
3TARGET = mips-linux-uclibc
```

54.14 Пакеты

54.14.1 **versions.mk**: Версии пакетов

54.14.2 **tc**: кросс-компилятор

mk/versions.mk

```
1 BINUTILS_VER = 2.24
2 GMP_VER = 5.1.3
3 MPFR_VER = 3.1.2
4 MPC_VER = 1.0.2
5 GCC_VER = 4.9.1
```

54.14.3 **core**: ядро

mk/versions.mk

```
1 KERNEL_VER = 3.17.6
2 ULIBC_VER = 0.9.33.2
3 BUSYBOX_VER = 1.22.1
```

54.14.4 **boot**: загрузчики

mk/versions.mk

```
1 SYSLINUX_VER = 6.03
2 UBOOT_VER = 2015.01
```

54.14.5 **libs**: библиотеки

```
1 SDL_VER = 1.2.15
2 SDL_IMAGE_VER = 1.2.12
3 SDL_TTF_VER = 2.0.11
```

54.14.6 **tc**: сборка кросс-компилиатора

```
1
2 CFG_BINUTILS0 = --target=$(TARGET) $(CFG_ARCH) $(CFG_CPU) \
3   --with-sysroot=$(ROOT) \
4   --with-native-system-header-dir=include \
5   --enable-lto --disable-multilib
6
7 CFG_CCLIBS0 = --disable-shared \
8   --with-gmp=$(TC) --with-mpfr=$(TC) --with-mpc=$(TC)
9
10 CFG_GMP0 = $(CFG_CCLIBS0)
11 CFG_MPFR0 = $(CFG_CCLIBS0)
12 CFG_MPC0 = $(CFG_CCLIBS0)
13
14 CFG_GCC0 = $(CFG_BINUTILS0) $(CFG_CCLIBS0) \
15   --disable-shared --disable-threads \
16   --without-headers --with-newlib \
17   --enable-languages="c"
18
19 CFG_GCC = $(CFG_BINUTILS0) $(CFG_CCLIBS0) \
```

```
20 ---enable-threads --enable-libgomp \
21 ---enable-languages="c,c++"
22
23 .PHONY: cross0
24 cross0: binutils0 cclibs0 gcc0
25
26 .PHONY: binutils0
27 binutils0: $(SRC)/$(BINUTILS)/README
28     rm -rf $(TMP)/$(BINUTILS) && mkdir $(TMP)/$(BINUTILS) && \
29     cd $(TMP)/$(BINUTILS) && \
30     $(SRC)/$(BINUTILS)/$(BCFG) $(CFG_BINUTILS0) && \
31     $(MAKE) && $(INSTALL)-strip
32
33 .PHONY: cclibs0
34 cclibs0: gmp0 mpfr0 mpc0
35
36 .PHONY: gmp0
37 gmp0: $(SRC)/$(GMP)/README
38     rm -rf $(TMP)/$(GMP) && mkdir $(TMP)/$(GMP) && \
39     cd $(TMP)/$(GMP) && \
40     $(SRC)/$(GMP)/$(BCFG) $(CFG_GMP0) && \
41     $(MAKE) && make install-strip
42
43 .PHONY: mpfr0
44 mpfr0: $(SRC)/$(MPFR)/README
45     rm -rf $(TMP)/$(MPFR) && mkdir $(TMP)/$(MPFR) && \
46     cd $(TMP)/$(MPFR) && \
47     $(SRC)/$(MPFR)/$(BCFG) $(CFG_MPFR0) &&
```

```
48 ____$(MAKE) && make install-strip
49
50 .PHONY: mpc0
51 mpc0: $(SRC)/$(MPC)/README
52 ____rm -rf $(TMP)/$(MPC) && mkdir $(TMP)/$(MPC) && \
53 ____cd $(TMP)/$(MPC) && \
54 ____$(SRC)/$(MPC)/$(BCFG) $(CFG_MPC0) && \
55 ____$(MAKE) && make install-strip
56
57 .PHONY: gcc0
58 gcc0: $(SRC)/$(GCC)/README
59 ____rm -rf $(TMP)/$(GCC) && mkdir $(TMP)/$(GCC) && \
60 ____cd $(TMP)/$(GCC) && \
61 ____$(SRC)/$(GCC)/$(BCFG) $(CFG_GCC0)
62 ____cd $(TMP)/$(GCC) && $(MAKE) all-gcc
63 ____cd $(TMP)/$(GCC) && $(MAKE) install-gcc
64 ____cd $(TMP)/$(GCC) && $(MAKE) all-target-libgcc
65 ____cd $(TMP)/$(GCC) && $(MAKE) install-target-libgcc
66
67 .PHONY: gcc
68 gcc: $(SRC)/$(GCC)/README
69 ____rm -rf $(TMP)/$(GCC) && mkdir $(TMP)/$(GCC) && \
70 ____cd $(TMP)/$(GCC) && \
71 ____$(SRC)/$(GCC)/$(BCFG) $(CFG_GCC)
72 ____cd $(TMP)/$(GCC) && $(MAKE) all-gcc
73 ____cd $(TMP)/$(GCC) && $(MAKE) install-gcc
74 ____cd $(TMP)/$(GCC) && $(MAKE) all-target-libgcc
75 ____cd $(TMP)/$(GCC) && $(MAKE) install-target-libgcc
```

54.14.7 **binutils**: ассемблер, линкер и утилиты

-target=\$(TARGET)	триплет целевой платформы
\$(CFG_ARCH)	параметры архитектуры
\$(CFG_CPU)	параметры процессора
-program-prefix	префикс <prefix>-(as ld ..)
-with-sysroot	каталог в котором находятся хедеры и библиотеки
-with-native-system-header-dir=/include	каталог с хедерами
-enable-lto	[L]ink[T]ime [O]ptimization

arch/i386.mk

```
1 # arch : ARCH_X86 i386  
2 CFG_ARCH = --disable-multilib
```

--disable-multilib выключить смешанный 32/64-битный режим

arch/arm.mk

```
1 # arch : ARCH_ARM arm  
2 TARGET = arm-linux-uclibcabi  
3 CFG_ARCH = --enable-interwork
```

--enable-interwork разрешить смешанный код ARM/THUMB

cpu/i486sx.mk

```
1 # i486sx cpu  
2 # target arch is Intel x86 (32 bit)  
3 ARCH = i386  
4 # target triplet for embedded linux  
5 TARGET = i486-linux-uclibc  
6 # target CPU configure params  
7 CFG_CPU = --disable-mmx --disable-3dnow --disable-sse
```

54.14.8 **cclibs**: библиотеки для сборки **gcc**

gmp mpfr mpc

54.14.9 **gcc0**: сборка минимального кросс-компилятора Си

При сборке **gcc0/gcc** отключайте **ccache**: кэш при разовых сборках не работает, но при монтировании как **tmpfs** потребляет слишком много ОЗУ:

```
./mk.rc && make CCACHE= gcc0
```

54.14.10 **gcc**: пересборка полного кросс-компилятора Си/ C_+

Пакет собирается [после сборки core](#).

54.14.11 **core**: сборка основной системы

mk/core.mk

```
1 .PHONY: core  
2 core: kernel ulibc gcc busybox
```

54.14.12 **kernel**: ядро Linux

mk/kernel.mk

```
1 CFG_KERNEL = ARCH=$(ARCH) INSTALL_HDR_PATH=$(ROOT)  
2  
3 .PHONY: kernel  
4 kernel: $(SRC)/$(KERNEL)/README  
5 # 1  
6 cd $(SRC)/$(KERNEL) && make $(CFG_KERNEL) distclean
```

```
7 ____cd $(SRC)/$(KERNEL) && make $(CFG_KERNEL) allnoconfig
8 ____# 2
9 ____cat kernel/all >> $(SRC)/$(KERNEL)/.config
10 ____cat kernel/arch/$(ARCH) >> $(SRC)/$(KERNEL)/.config
11 ____cat kernel/cpu/$(CPU) >> $(SRC)/$(KERNEL)/.config
12 ____cat kernel/hw/$(HW) >> $(SRC)/$(KERNEL)/.config
13 ____cat kernel/app/$(APP) >> $(SRC)/$(KERNEL)/.config
14 ____# 3
15 ____echo "CONFIG CROSS_COMPILE=\"$(TARGET)-\" >> $(SRC)/$(KERNEL)/.config
16 ____echo "CONFIG LOCALVERSION=\"-$(HW)$(APP)\" >> $(SRC)/$(KERNEL)/.config
17 ____echo "CONFIG DEFAULT_HOSTNAME=\"$(HW)$(APP)\" >> $(SRC)/$(KERNEL)/.config
18 ____# 4
19 ____cd $(SRC)/$(KERNEL) && make $(CFG_KERNEL) menuconfig
20 ____# 5
21 ____cd $(SRC)/$(KERNEL) && $(MAKE) $(CFG_KERNEL)
22 ____# 6
23 ____make kernel-$(ARCH)-fix
24 ____cp $(SRC)/$(KERNEL)/arch/$(ARCH)/boot/zImage $(BOOT)/$(HW)$(APP).kernel
25 ____# 7
26 ____cd $(SRC)/$(KERNEL) && make $(CFG_KERNEL) headers_install
27
28.PHONY: kernel-i386-fix
29kernel-i386-fix:
30____cp \
31____ $(SRC)/$(KERNEL)/arch/$(ARCH)/boot/bzImage \
32____ $(SRC)/$(KERNEL)/arch/$(ARCH)/boot/zImage
33
34.PHONY: kernel-arm-fix
```

35 kernel-arm-fix :

ARCH архитектура: `src/linux-x.x.x/arch/*`
INSTALL_HDR_PATH путь установки **хедеров ядра**

1. подготовка к сборке с пустым **конфигом** `src/linux-x.x.x/.config`
2. накатываем на пустой **.config** файлы-модификаторы, содержащие переопределения переменных конфигурации:
 - **all** универсальные параметры ядра для всех платформ
 - **arch** параметры, специфичные для архитектуры
 - **cpu** параметры, адаптирующие ядро к конкретному процессору (эмулатор FPU, возможности управления тактовой частотой и потреблением, поддержка многоядерности и т.п.)
 - **hw** параметры чипсета и периферии материнской платы (**модули ядра** = драйвера)
 - **app** параметры, в основном **отключаемые** для конкретного **приложения** (отключение графики, режим реального времени, спец.параметры)
3. установка параметров кросс-компиляции и имени сборки
4. запуск интерактивного меню конфигурирования, выйти с сохранением
5. сборка ядра
6. копирование готового файла ядра в **`\${BOOT}`**
7. генерация **хедеров** в **`\${ROOT}`**

Все параметры ядра идут с префиксом **CONFIG_**:

- **core.mk**
 - **CROSS_COMPILE** префикс кросс-компилятора <pfx>-(gcc|as|ld..)
 - **LOCALVERSION** суффикс ядра linux-x.x.x-suffix
 - **DEFAULT_HOSTNAME** имя компьютера <host>.<domain>
- **all** для всех платформ
 - **NOHIGHMEM=y** сборка в режиме $\leq 1\text{Gb}$ ОЗУ
 - **KERNEL_GZIP=y** ядро сжимать алгоритмом **gzip**⁵
 - корневая файловая система в **initrd/ramdisk**
 - * **BLK_DEV_INITRD=y** rootfs в ОЗУ(initrd)
 - * **PROC_FS=y** файловая подсистема /proc
 - * **SYSFS=y** файловая подсистема /sys
 - * **DEVTMPFS=y** файловая подсистема /dev (интерфейсы драйверов устройств)
 - * **DEVTMPFS_MOUNT=y** автоматически монтировать **devfs**
 - режим реального времени⁶ ??
 - * **PREEMPT=y** вытесняющая многозадачность в режиме ядра
 - * **HZ_1000=y** системный таймер 1KHz
 - исполняемые форматы файлов

⁵ важно для загрузчиков на не-i386 системах

⁶ повышенная или **гарантированная** отзывчивость системы на внешние события

- * **BINFMT_ELF**=y бинарные файлы в формате **ELF**
- * **BINFMT_SCRIPT**=y файлы скриптов с `#!/bin/sh` в заголовке
- * **COREDUMP**=n не писать **корки** при сбоях
- **поддержка консоли** `tty`
 - * **TTY**=y последовательные порты и командная консоль
 - * **UNIX98_PTYS**=n псевдотерминалы UNIX98
 - * **LEGACY_PTYS**=n псевдотерминалы UNIX/BSD
- **клавиатура**
 - * **INPUT_KEYBOARD**=y ввод с аппаратной клавиатуры
- **мышь**
 - * **INPUT_MOUSE**=y
 - * **INPUT_MOUSEDEV**=y
 - * **INPUT_MOUSEDEV_PSAUX**=n не создавать `/dev/psaux`
- **USB HID: устройства ввода без драйверов** (клавиатура, мышь, джойстик, самодельные кнопки)
включение поддержки USB на i386 не требуется
 - * **HID**=y [H]uman [I]nterface [D]evice
 - * **HID_GENERIC**=y универсальный драйвер USB HID Class
- **Графический режим** `FrameBuffer`
 - * **FB**=y
 - * **FRAMEBUFFER_CONSOLE**=y командная консоль
 - * **LOGO**=y вывод пингвина при запуске
 - * **LOGO_LINUX_MONO**=y выводить только черно-белый вариант

- * **LOGO_LINUX_VGA16=n**
- * **LOGO_LINUX_CLUT224=n**
- Отладка и printk лог ядра
 - * **DEBUG_KERNEL=y** включение отладочных функций ядра
 - * **EARLY_PRINTK=y** вывод пусковой части ядра
 - * **PRINTK=y** вывод главного системного лога ядра **printk**
 - * **PRINTK_TIME=y** выводить метки времени (для отладки времени запуска)
 - * **SCHED_DEBUG=n** не отлаживать планировщик
 - * **DEBUG_PREEMPT=_n** не отлаживать вытесняющий режим
- **arch i386**
 - **X86_GENERIC=y** универсальная оптимизация для всех i386-процессоров
 - **UART/COM/RS232** последовательные порты
 - * **SERIAL_8250=y** UART 8250/16550
 - * **SERIAL_8250_CONSOLE=y** командандная консоль на COM-портах
 - **VGA_CONSOLE=y** командандная консоль на VGA 80×25
 - **FrameBuffer**
 - * **FB_VESA=y** универсальный видеодрайвер VESA 2.0+
 - **клава**
 - * **KEYBOARD_ATKBD=y** клавиатура PC AT / PS2
 - **мышь**
 - * **MOUSE_PS2=y** мышь PS/2

- * **MOUSE_SERIAL=y** (старая) мышь на RS232
- **NVRAM=y** доступ к памяти CMOS
- [отладка](#)
 - * **MAGIC_SYSRQ=y** волшебная кнопка **Alt**+**SysRq**
 - * **X86_VERBOSE_BOOTUP=y** доп.сообщения при пуске ядра
- [сри i386 i486sx](#)
 - **M486=y**
 - **MATH_EMULATION=y** включить программную эмуляцию мат.сопроцессора (FPU) в ядре
- [hw qemu386 qemu386](#)
 - **MATH_EMULATION=n** выключить эмуляцию FPU: **QEMU** запускается на процессорах Pentium и старше, которые всегда имеют аппаратный модуль плавающей точки
 - **HZ_1000=n** в режиме эмуляции быстрый таймер не имеет смысла,
 - **HZ_100=y** переключаемся на 100 Hz
- [app micro](#)
 - **FB=n** выключаем поддержку FrameBuffer,..
 - **INPUT_MOUSE=n** мыши

54.14.13 ulibc: библиотека uClibc

mk/ulibc.mk

```
1 CFG_ULIBC = CROSS=$(TARGET)- ARCH=$(ARCH) PREFIX=$(ROOT)
2 #__UCLIBC_EXTRA_CFLAGS=""  
3  
4 .PHONY: ulibc  
5 ulibc: $(SRC)/$(ULIBC)/README  
6     # 1  
7     cd $(SRC)/$(ULIBC) && $(MAKE) $(CFG_ULIBC) distclean  
8     cd $(SRC)/$(ULIBC) && $(MAKE) $(CFG_ULIBC) allnoconfig  
9     # 2  
10    cat ulibc/all >> $(SRC)/$(ULIBC)/.config  
11    cat ulibc/arch/$(ARCH) >> $(SRC)/$(ULIBC)/.config  
12    cat ulibc/cpu/$(CPU) >> $(SRC)/$(ULIBC)/.config  
13    cat ulibc/app/$(APP) >> $(SRC)/$(ULIBC)/.config  
14     # 3  
15    echo "CROSS_COMPILER_PREFIX=\"$(TARGET)-\" >> $(SRC)/$(ULIBC)/.config  
16    echo "KERNEL_HEADERS=\"$(ROOT)/include\" >> $(SRC)/$(ULIBC)/.config  
17     # 4  
18    cd $(SRC)/$(ULIBC) && $(MAKE) $(CFG_ULIBC) menuconfig  
19     # 5  
20    cd $(SRC)/$(ULIBC) && $(MAKE) $(CFG_ULIBC)  
21     # 6  
22    cd $(SRC)/$(ULIBC) && $(MAKE) $(CFG_ULIBC) install  
23     # 7  
24    cd $(SRC)/$(ULIBC) && $(MAKE) $(CFG_ULIBC) install_utils  
25     # 8
```

```
26 cd $(SRC)/$(ULIBC) && $(MAKE) $(CFG_ULIBC) hostutils
27 cp $(SRC)/$(ULIBC)/utils/ldd.host $(TC)/bin/$(TARGET)-ldd
28 cp $(SRC)/$(ULIBC)/utils/ldconfig.host $(TC)/bin/$(TARGET)-ldconfig
29 cp $(SRC)/$(ULIBC)/utils/getconf.host $(TC)/bin/$(TARGET)-getconf
30 # 9 (in root package)
31 # ldconfig in root.mk
```

1. подготовка к сборке с пустым конфигом `src/uClibc-x.x.x/.config`
2. накатываем на пустой `.config` файлы-модификаторы, содержащие переопределения переменных конфигурации:
 - `all` универсальные параметры `ulibc` для всех платформ
 - `arch` параметры, специфичные для архитектуры
 - `crt` параметры, адаптирующие `ulibc` к конкретному процессору (набор команд и оптимизация)
 - `app` параметры, в основном отключаемые для конкретного приложения (отключение shared библиотек и неиспользуемых групп функций)
3. установка параметров кросс-компиляции и [пути к хедерам ядра](#)
4. запуск интерактивного меню конфигурирования, выйти с сохранением
5. сборка `ulibc`
6. инсталляция в `ROOT`
7. инсталляция утилит для работы с shared библиотеками [на целевой системе](#)
8. инсталляция утилит для `BUILD`-системы

9. создание /etc/ld.so.cache (выполняется в пакете root)

- - ARCH_HAS_MMU=y всегда включено
 - ARCH_USE_MMU=y всегда включено
 - UCLIBC_HAS_FPU=y всегда включено: используется эмулятор в ядре
 - UCLIBC_HAS_FLOATS=y функции плавающей точки
 - DO_C99_MATH=n функции float math C99
 - UCLIBC_CTOR_DTOR=y конструктор/деструктор
 - UCLIBC_HAS_LFS=y большие файлы
- разделяемые shared библиотеки
 - HAVE_SHARED=y
 - LDSO_CACHE_SUPPORT=y /etc/ld.so.conf
 - LDSO_PRELOAD_ENV_SUPPORT=n переменная LD_PRELOAD содержит список библиотек, загружаемых первыми
 - LDSO_LD_LIBRARY_PATH=n отключить переменную LD_LIBRARY_PATH со списком каталогов поиска shared библиотек
- многопоточные threads программы
 - LINUXTHREADS_OLD=y старый вариант потоков, новый вызывает segfault при использовании ключей gcc -lpthread и использовании shared ulibc

- параметры установки

- **DOSTrip=y**
 - **RUNTIME_PREFIX=**
 - **DEVEL_PREFIX=**

- опции необходимые **busybox**

- **UCLIBC_HAS_CTYPE_TABLES=y**
 - **UCLIBC_HAS_NETWORK_SUPPORT=y**
 - **UCLIBC_HAS_FNMATCH=y**
 - **UCLIBC_HAS_GNU_GETOPT=y**
 - **UCLIBC_HAS_REGEX=y**
 - **UCLIBC_SUSV3_LEGACY=y**

-

ulibc/all

```
1 # CPU
2
3 ARCH_HAS_MMU=y
4 ARCH_USE_MMU=y
5
6 # FPU/floats
7
8 UCLIBC_HAS_FPU=y
```

```
9 UCLIBC_HAS_FLOATS=y
10 DO_C99_MATH=n
11 UCLIBC_HAS_LONG_DOUBLE_MATH=n
12
13 # shared libs and c++ stuff
14
15 HAVE_SHARED=y
16 LDSO_CACHE_SUPPORT=y
17 LDSO_PRELOAD_ENV_SUPPORT=n
18 LDSO_LD_LIBRARY_PATH=n
19
20 UCLIBCCTOR_DTOR=y
21 LINUXTHREADS_OLD=y
22
23 UCLIBC_LINUX_SPECIFIC=y
24 UCLIBC_LINUX_MODULE_26=n
25 UCLIBC_HAS_LFS=y
26
27 UCLIBC_HAS_REALTIME=y
28 UCLIBC_HAS_STRING_GENERIC_OPT=y
29 UCLIBC_HAS_STRING_ARCH_OPT=y
30
31 DOSTRIP=y
32 RUNTIME_PREFIX=""
33 DEVEL_PREFIX=""
34
35 # net
36 UCLIBC_HAS_NETWORK_SUPPORT=y
```

```
37 UCLIBC_HAS_RESOLVER_SUPPORT=y
38 UCLIBC_HAS_IPV4=y
39 UCLIBC_HAS_IPV6=n
40
41 # i18n disabled
42 UCLIBC_HAS_WCHAR=n
43
44 # BB needs
45 UCLIBC_HAS_CTYPE_TABLES=y
46 UCLIBC_HAS_FNMATCH=y
47 UCLIBC_HAS_GNU_GETOPT=y
48 UCLIBC_HAS_REGEX=y
49 UCLIBC_SUSV3_LEGACY=y
50
51 # Python needs
52 DO_C99_MATH=y
53 #UCLIBC_HAS_LONG_DOUBLE_MATH=y
54 UCLIBC_HAS_WCHAR=y
55 UCLIBC_SUSV4_LEGACY=y
56 #UCLIBC_HAS_PTY=y
57 #UNIX98PTY_ONLY=y
58 #UCLIBC_HAS_LIBUTIL=y
59
60 #DOMULTI=y
61 #UCLIBC_HAS_COMPAT_RES_STATE=n
62 #UCLIBC_HAS_SYSLOG=y
63 ## ALSA needs
64 #UCLIBC_HAS_BSD_ERR=y
```

```
65  
66 # canadian cross needs  
67 #UCLIBC_HAS_WCHAR=y
```

ulibc/arch/i386

```
1 TARGET_i386=y
```

ulibc/cpu/i486sx

```
1 CONFIG_486=y
```

54.14.14 **gcc**: пересборка полного **gcc**

После сборки **ulibc** необходимо еще раз пересобрать **gcc** с полными настройками.

54.14.15 **busybox**: набор утилит **busybox**

mk/busybox.mk

```
1 CFG_BUSYBOX = \  
2 ____CONFIG_PREFIX=$(ROOT) \  
3 ____CROSS_COMPILE=$(TARGET)- \  
4 ____SYSROOT=$(ROOT)  
5  
6 .PHONY: busybox  
7 busybox: $(SRC)/$(BUSYBOX)/README  
8 # 1
```

```
9 cd $(SRC)/$(BUSYBOX) && $(MAKE) $(CFG_BUSYBOX) distclean
10 cd $(SRC)/$(BUSYBOX) && $(MAKE) $(CFG_BUSYBOX) allnoconfig
11 # 2
12 cp app/${APP}.bb $(SRC)/$(BUSYBOX)/.config
13 cd $(SRC)/$(BUSYBOX) && $(MAKE) $(CFG_BUSYBOX) menuconfig
14 cp $(SRC)/$(BUSYBOX)/.config app/${APP}.bb
15 # 3
16 cd $(SRC)/$(BUSYBOX) && $(MAKE) $(CFG_BUSYBOX) install
```

1. подготовка к сборке с пустым конфигом
2. интерактивное меню конфигурирования с сохранением конфига в `app/${APP}.bb`
3. сборка и инсталляция в ROOT

Особенность `busybox` — скрипты конфигурации не умеют отрабатывать переписывание переменных конфигурации, поэтому приходиться держать в `app/*.bb` настройки для каждого приложения полностью. Если вы создаете свое приложение, скопируйте `app/micro.bb` в качестве базовой версии вашего `app/userapp.bb`.

54.14.16 libs: сборка библиотек `${LIBS}`

54.14.17 apps: сборка прикладных пакетов `${APPS}`

54.14.18 user: сборка пользовательского кода

54.14.19 root: формирование корневой файловой системы

mk/root.mk

```
1 ROOTREX = " ./(boot|pack)"  
2  
3 .PHONY: root  
4 root:  
5     # 1  
6     rm -rf $(ETC) ; cp -r etc $(ROOT)/  
7     cat app/$(APP).rcS >> $(ROOT)/etc/init.d/rcS  
8     cp README.md $(ETC)/  
9     chmod +x $(ETC)/init.d/* $(ETC)/dhcp.rc  
10    # 2  
11    $(LDCONFIG) -v -r $(ROOT)  
12    # 3  
13    ln -fs /sbin/init $(ROOT)/init  
14    # 4  
15    cp -r share $(ROOT)/  
16    # 5  
17    cd $(ROOT) && find . | egrep -v $(ROOTREX) > $(PACK)/allfiles  
18    # 6  
19    python ./pack.py $(ROOT)  
20    # cat $(PACK)/allfiles > $(PACK)/rootfiles  
21    # 7  
22    cd $(ROOT) && cat $(PACK)/rootfiles | cpio -o -H newc > $(BOOT)/$(HW)$(APP).cpio  
23    cat $(BOOT)/$(HW)$(APP).cpio | gzip -9 > $(BOOT)/$(HW)$(APP).rootfs
```

1. пересоздание /etc/

2. генерация кеша динамических библиотек /etc/ld.so.cache

3. /sbin/init → /init

4. /share/

5. создание **initrd** используя фильтрацию файлов **регулярным выражением** в переменной **ROOTREX**

/etc/inittab

```
1 :: sysinit:/etc/init.d/rcS
2 :: shutdown:/etc/init.d/rcD
3 :: ctrlaltdel:/sbin/reboot
4 tty1::askfirst:/bin/sh
5 tty2::askfirst:/bin/sh
6 tty3::askfirst:/bin/sh
7 tty4::askfirst:/bin/sh
8 #ttyS1::respawn:/bin/sh
```

/etc/init.d/rcS

```
1#!/bin/sh
2# system dirs
3mkdir /tmp
4mkdir /proc ; mount -t proc proc /proc
5mkdir /sys ; mount -t sysfs sysfs /sys
6# hotplug device manager (automount, firmware boot, config)
7mdev -s && echo /sbin/mdev > /proc/sys/kernel/hotplug
8# about
9uname -a
10cat /etc/README.md
```

1. создание системных каталогов
2. запуск **демона mdev**, обрабатывающего **devfs**: создание/удаление устройств в **/dev**, автомонтирование, загрузка прошивок,..
3. вывод **README**

/etc/init.d/rcD

```
1#!/bin/sh
2sync
3umount -a
```

54.14.20 **boot**: сборка загрузчика **syslinux/grub/uboot**

mk/boot.mk

```
1
2UNZISO = unzip -jn $(GZ)/$(SYSLINUX).zip -d $(ISO)
3
4.PHONY: iso
5iso : $(BOOT)/$(HW)$(APP).iso
6$(BOOT)/$(HW)$(APP).iso : $(BOOT)/$(HW)$(APP).kernel $(BOOT)/$(HW)$(APP).rootfs \
7syslinux/isolinux.cfg
8____rm -rf $(ISO) && mkdir $(ISO)
9____cp $(BOOT)/$(HW)$(APP).kernel $(ISO)/kernel
10____cp $(BOOT)/$(HW)$(APP).rootfs $(ISO)/rootfs
11____cp syslinux/memtest.krn $(ISO)/
```

```
12 cp share/splash640x480.png $(ISO)/splash.png
13 $(UNZISO) bios/com32/elflink/ldlinux/ldlinux.c32
14 $(UNZISO) bios/com32/lib/libcom32.c32
15 $(UNZISO) bios/com32/libutil/libutil.c32
16 $(UNZISO) bios/com32/menu/menu.c32
17 $(UNZISO) bios/com32/menu/vesamenu.c32
18 $(UNZISO) bios/core/isolinux.bin
19 cp syslinux/isolinux.cfg $(ISO)/syslinux.cfg
20 $(MKISO) -no-emul-boot -boot-info-table -b isolinux.bin \
21 -o $(BOOT)/$(HW)$(APP).iso $(ISO)
22 #-r -J
```

54.14.21 syslinux

54.14.22 grub

54.14.23 uboot

54.14.24 **emu**: запуск собранной системы в эмуляторе

Для отладки кода, не связанного жестко с железом, для которого допускается выполнение в эмуляторе, используется **QEMU**.

mk/emu.mk

```
1 include app/$(APP).qemu
2
3 QEMU_ALL = -m 64M -net nic -net user -localtime -append "vga=0x312"
4
```

```

5 .PHONY: emu
6 emu: $(BOOT)/$(HW)$(APP).kernel $(BOOT)/$(HW)$(APP).rootfs
7     qemu-system-$(ARCH) $(QEMU_CFG) $(QEMU_ALL) \
8     --kernel $(BOOT)/$(HW)$(APP).kernel \
9     --initrd $(BOOT)/$(HW)$(APP).rootfs
10
11 .PHONY: emuk
12 emuk: $(BOOT)/$(HW)$(APP).kernel
13     qemu-system-$(ARCH) $(QEMU_CFG) $(QEMU_ALL) \
14     --kernel $(BOOT)/$(HW)$(APP).kernel
15
16 .PHONY: emuiso
17 emuiso: $(BOOT)/$(HW)$(APP).iso
18     qemu-system-$(ARCH) $(QEMU_CFG) \
19     --boot d --cdrom $<
20
21 .PHONY: bochs
22 bochs: $(BOOT)/$(HW)$(APP).iso
23     bochs -f syslinux/bochs.rc

```

hw/qemu386.mk

```

1 # QEMU emulator: i386 mode
2 CPU = i486sx
3 QEMU_CFG = -m 64M -net none -localtime
4 #vga=ask
5 ##0x312 640x480x24
6 ##0x315 800x600x24

```

Переменная QEMU_CFG задает параметры запуска **QEMU**:

-m	объем ОЗУ на эмулируемой системе
-net none	отключить сеть и iPXE boot
-kernel	прямая загрузка ядра Linux из файла
-initrd	образ корневой файловой системы (initrd)
-append	параметры ядра:
vga=ask	запрос списка видеорежимов при загрузке и выбор режима
vga=None	std. VGA text 80×25, FrameBuffer отключен
vga=0x315	VESA 800×600×24bit
vga=0x317	VESA 1024×768×16bit

Mode 0x0301: 640x480 (+640), 8 bits
Mode 0x0303: 800x600 (+800), 8 bits
Mode 0x0305: 1024x768 (+1024), 8 bits
Mode 0x0311: 640x480 (+1280), 16 bits
Mode 0x0312: 640x480 (+2560), 24 bits
Mode 0x0314: 800x600 (+1600), 16 bits
Mode 0x0315: 800x600 (+3200), 24 bits
Mode 0x0317: 1024x768 (+2048), 16 bits
Mode 0x0318: 1024x768 (+4096), 24 bits

54.15 **netboot**: Сетевая загрузка

54.16 Прошивка на устройство

54.17 RT-патч

54.18 SDK: расширения для on-board разработки

Иногда на целевой системе требуется набор средств программирования, для выполнения

- сложных скриптов (Python 54.18.6),
- математических вычислений⁷ (GCL 54.18.7/Maxima 54.18.8),
- или компиляции неизвестных заранее исходников:
 - обновления или отладка прошивок для периферийных контроллеров
 - пересборки программ в хост-системе
 - отладки программ, активно работающих со специфичным железом

Для этого в состав azLinux включен набор расширений SDK⁸.

⁷ в т.ч. и символьных (аналитических, не численных)

⁸ [S]oftware [D]evelopment [K]it

54.18.1 canadian: сборка binutils канадским крестом

Канадский крест — метод кросс-компиляции `binutils/gcc`, когда явно указываются три триплета:

`-build` платформа, на которой выполняется сборка, т.е. ваш билд-сервер: `x86_64` или `i686`

`-host` платформа, на которой будет выполняться собранный кросс-компилятор, т.е. TARGET нашей встраиваемой системы: `i386-linux-uclibc`, `arm-linux-gnueabihf` или что-то подобное

`-target` платформа или микропроцессор, для которого кросс-компилятор будет генерировать код, например `Cortex-M3`.

mk/sdk/canadian.mk

```
1 # canadian cross
2
3 CFG_CAN_BIN = \
4     --with-native-system-header-dir=include \
5     --enable-lto --disable-multilib
6 CFG_GCC = $(CFG_CAN_BIN) \
7     --enable-threads --enable-libgomp \
8     --enable-languages="c,c++"
9 ##--program-prefix=$(P)
10
11 .PHONY: canadian
12 canadian: $(SRC)/$(BINUTILS)/README $(SRC)/$(GCC)/README
13 ##--# binutils
14 ##--rm -rf $(TMP)/$(BINUTILS) && mkdir $(TMP)/$(BINUTILS) &&
15 ##--cd $(TMP)/$(BINUTILS) && \
16 ##-- $(XPATH) $(SRC)/$(BINUTILS)/$(CACFG) \
```

```
17 #____$_($CFG_CAN_BIN) --target=$($T) $(O) --prefix=$($PFX) --with-sysroot=$($SR) && \
18 #____$_($MAKE) && $_($PINSTALL)-strip && \
19 #____grep $(ROOT) $(PACK)/ .strace > $(PACK)/ binutils.$(PK).strace && rm $(PACK)/ .strace
20 #____# gcc
21 ____rm -rf $(TMP)/$(GCC) && mkdir $(TMP)/$(GCC) && \
22 ____cd $(TMP)/$(GCC) && \
23 ____$_($XPATH) $(SRC)/$(GCC)/$(CACFG) \
24 ____$_($CFG_CAN_GCC) --target=$($T) $(O) --prefix=$($PFX) --with-sysroot=$($SR)
25 ____cd $(TMP)/$(GCC) && $_($MAKE) all-gcc
26 #____cd $(TMP)/$(GCC) && $_($MAKE) install-gcc
27 #____cd $(TMP)/$(GCC) && $_($MAKE) all-target-libgcc
28 #____cd $(TMP)/$(GCC) && $_($MAKE) install-target-libgcc
29
30 .PHONY: binhost
31 binhost:
32 ____make canadian T=$(TARGET) PFX=$(USR) SR=/ PK=host O=
33 #____ O=$(CFG_ARCH) $(CFG_CPU)"
34
35 .PHONY: bin486
36 bin486:
37 ____make canadian T=i486-elf O=
38 #____ O="--with-cpu=i486"
39
40 .PHONY: binavr
41 binavr:
42 ____make canadian T=avr O=
43 #____ O=
44
```

```
45 .PHONY: bincmx
46 bincmx:
47     make canadian T=arm-none-eabi O=
48 #__P=cmx_ \
49 #__O="--enable-interwork --disable-multilib"
```

54.18.2 **binhost**: **binutils** для хост-процессора

Ассемблер и линкер для компиляции и (пере)сборки бинарников для процессора хост-системы.

54.18.3 **binavr8**: **binutils** для Atmel ATmega (AVR8)

Компиляция и прошивка периферийных контроллеров на популярных микропроцессорах Atmel ATmega (AVR8).

54.18.4 **bincmx**: **binutils** для ARM Cortex-Mx

Компиляция и прошивка периферийных контроллеров на микропроцессорах перспективной линейки Cortex-Mx.

54.18.5 **fpc**: FreePascal

mk/sdk/pascal.mk

```
1 FPC_CFG = INSTALL_PREFIX=$(PWD)/pascal \
2 BINUTILSPREFIX=$(TARGET)- \
3 OS_TARGET=linux CPU_TARGET=i386
```

```
4
5 #FPC=ppcx64
6 #OS_TARGET=linux
7 #CPU_TARGET=i386
8 #CROSSOPT="-Xd -Xt"
9
10 .PHONY: fpc
11 #fpc: $(SRC)/fpcbuild-$(FPC_VER)/ Makefile
12 ##__cd $(SRC)/fpcbuild-$(FPC_VER)/ fpcsrc && \
13 ##__make distclean && \
14 ##__make all $(FPC_CFG) && \
15 ##__make install $(FPC_CFG)
16 ##__ CPU_TARGET=i386 FPC=ppc386
17 #
18 #$(SRC)/fpcbuild-$(FPC_VER)/ Makefile:
19 ##__cd $(SRC) ; tar zx < $(GZ)/fpcbuild-$(FPC_VER).tar.gz
20
21 .PHONY: fpc
22 fpc: $(SRC)/$(FPC)/ Makefile
23 ##__cd $(SRC)/$(FPC) && \
24 ##__make $(FPC_CFG) distclean && \
25 ##__$(XPATH) make $(FPC_CFG) build
26
27 $(SRC)/$(FPC)/ Makefile:
28 ##__cd $(SRC) ; tar zx < $(GZ)/$(FPC).source.tar.gz
```

54.18.6 **python**: интерпретатор Python

54.18.7 **gcl**: GNU Common Lisp

54.18.8 **maxima**: система символьной математики Maxima

Глава 55

Применение Linux для новой “железки”

В последние несколько лет идет взрывной рост применения встраиваемых плат на базе процессоров архитектур ARM и MIPS, в комплекте к которым идет та или иная сборка emLinux. В большой степени это обусловлено массовым выпуском мобильных устройств и роутеров SOHO-сегмента. С другой стороны, производители железа учитывают высокую популярность Linux в среде разработчиков.

В этом разделе рассмотрим случай, с которым вы обязательно столкнетесь: **в какой-то момент вам потребуется самостоятельно запустить и освоить новую “железку”, в комплекте к которой идет какая-нибудь сборка emLinux.**

В 54 приведено полное описание системы сборки для создания пакета кросс-компиляции. Но этот вариант (полнейшей пересборки прошивки) подходит вам только в случае, если вы полностью определяете состав ПО.

Очень часто требуется сохранить прошивку и сборку Linux-системы от поставщика или производителя оборудования, т.к. в нее могут входить части, поставляемые в бинарном виде, vendor-specific патчи на ядро, и подобное закрытое ПО в виде так называемых “бинарных блобов”. Чаще всего этим страдают мобильные аппаратные платформы, в т.ч. Raspberry Pi: производитель SoC¹ предоставляет часть драйверов в виде за-

¹ [S]ystem-[o]n-[C]hip: процессор и периферия, иногда даже RAM и Flash, объединенные в одном корпусе

крытого firmware. В клиническом случае предоставляется ядро собственной модификации и набор утилит для управления железом только в бинарном виде без возможности пересборки пользователем.

Другой типичный случай: вам нужно запустить на железке всего 1-2 ваших программы. При этом [вендорная сборка](#) Linux устройства слишком громоздка, чтобы полностью ее пересобирать: X Window, полный мультимедийный комплект ПО и библиотек, тяжелый браузер,... Или вы хотите сохранить возможность установки сторонних пакетов и регулярного обновления какой-нибудь популярной сборки, например Raspbian.

В качестве примера разберем добавление конфигурации для двух железок:

1. VoCore 54.12.3
2. ТионПро270 54.11.4

Глава 56

Библиотека libSDL

Библиотека SDL предоставляет такие средства, как быстрый вывод 2D-графики, обработку ввода, проигрывание звука, вывод 3D через OpenGL и другие операции, причем делает это кроссплатформенно. Список платформ обширный: Linux, Windows, Windows CE, BeOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX и QNX — и вдобавок есть неофициальные порты на другие системы.

Сама библиотека написана на Си и поддерживает C^+ , однако есть биндинги к большинству популярных языков. Автор [libsdl](#) был нанят компанией Valve, программные продукты которой активно используют библиотеку. К тому же, теперь библиотека выходит под лицензией zlib, а не LGPLv2, как было раньше, и SDL 2.0 можно использовать в любых своих приложениях, в т.ч. коммерческих. Скорее всего сделано это было для того, чтобы Valve смогла включить ее в Steam для Linux.

Использование SDL позволяет писать графические и мультимедийные приложения для emLinux, не включая в систему достаточно тяжелую X Window System.

В книге рассмотрена версия 1.2.15, последняя версия из ветки [SDL1](#), т.к. она используется в сборке azLinux 54. Сейчас активно развивается ветка [SDL2](#), ее особенности кратко рассмотрены в разделе 56.4.

56.1 Инициализация и завершение SDL-программы

```
int main() {  
    ...  
    SDL_Init(SDL_INIT_xxx));  
    ...  
    SDL_Quit();  
}
```

SDL_INIT_EVERYTHING все подсистемы, не работает с отключенным звуком

SDL_INIT_VIDEO только графическая подсистема

56.2 LazyFoo tutorial

¹

56.2.1 Setting up SDL and Getting an Image on the Screen

Since SDL is a third party library, you have to install it yourself. Here you'll get a step by step guide on how to set it up.

If you have any problems try consulting the [SDL Development FAQ](#).

Once you set up SDL, you can move on the second half of the tutorial and learn to load and show images on the screen.

¹ копипаста: http://lazyfoo.net/SDL_tutorials/

■ Windows

An important note for Visual Studio users: The latest version of SDL for visual studio comes with two sets of library and binary files: x86 for 32bit, x64 for 64bit. If you're on a 64bit operating system, Visual Studio still compiles in 32bit by default. When you set the library directory, it should point to the x86 folder inside of the lib folder.

Установка для следующих сред разработки описана здесь:

1. Dev C++ 4.9.9.2
2. Code::Blocks 8.02
3. MinGW Developer Studio 2.05
4. Eclipse 3.1
5. Command Line (MinGW)
6. Visual Studio.NET 2010 Express
7. Visual Studio.NET 2005/2008 Express
8. Visual Studio.NET 2003

Linux

Пакеты ставятся из вашего дистрибутива.

Для включения в azLinux добавьте пакет `sdl` в переменную конфигурирования `LIBS += sdl`. Сборка пакета описана в [??](#).

Getting an Image on the Screen

This tutorial covers how to do Hello World SDL style.

Now that you have SDL set up, it's time to make a bare bones graphics application that loads and displays an image on the screen.

```
// Include SDL functions and datatypes
#include "SDL/SDL.h"
```

At the top of the source file we include the SDL header file so we can use the SDL functions and data types.

Remember that some of you (like Visual Studio users) are going to include SDL like this:

```
#include "SDL.h"
```

So if the compiler is complaining that it can't find "SDL/SDL.h" then it's either because you're including the wrong path or you forgot to put `SDL.h` in the right place.

```
int main( int argc, char* args[] )
{
    //The images
    SDL_Surface* hello = NULL;
    SDL_Surface* screen = NULL;
```

At the top of the `main()` function, two `SDL_Surface` pointers are declared. An `SDL_Surface` is an image, and in this application we're going to be dealing with two images. The surface "`hello`" is the image we're going to be loading and showing. The "`screen`" is what is visible on the screen.

Whenever you're dealing with pointers, you should always remember to initialize them.

Also, when using SDL, you must have your `main()` function declared like it is above. You can't use `void main()` or anything like that.

```
//Start SDL
SDL_Init( SDL_INIT_EVERYTHING );

//Set up screen
screen = SDL_SetVideoMode( 640, 480, 32, SDL_SWSURFACE );

//Load image
hello = SDL_LoadBMP( "hello.bmp" );
```

The first function we call in the main() function is `SDL_Init()`. This call of `SDL_Init()` initializes all the SDL subsystems so we can start using SDL's graphics functions.

Next `SDL_SetVideoMode()` is called to set up a 640 pixel wide, 480 pixel high window that has 32 bits per pixel. The last argument (`SDL_SWSURFACE`) sets up the surface in software memory. After `SDL_SetVideoMode()` executes, it returns a pointer to the window surface so we can use it.

After the window is set up, we load our image using `SDL_LoadBMP()`. `SDL_LoadBMP()` takes in a path to a bitmap file as an argument and returns a pointer to the loaded `SDL_Surface`. This function returns `NULL` if there was an error in loading the image.

```
//Apply image to screen
SDL_BlitSurface( hello, NULL, screen, NULL );

//Update Screen
SDL_Flip( screen );

//Pause
SDL_Delay( 2000 );
```

Now that we have our window set up and our image loaded, we want to apply the loaded image onto the screen. We do this using `SDL_BlitSurface()`. The first of `SDL_BlitSurface()` argument is the source surface. The third argument

is the destination surface. `SDL_BlitSurface()` sticks the source surface onto the destination surface. In this case, it's going to apply our loaded image onto the screen. You'll find out what the other arguments do in later tutorials.

Now that our image is applied to screen, we need to update the screen so we can see it. We do this using `SDL_Flip()`. If you don't call `SDL_Flip()`, you'll only see an unupdated blank screen.

Now that the image is applied to the screen and it's made visible, we have to make the window stay visible so it doesn't just flash for a split second and disappear. We'll make the window stay by calling `SDL_Delay()`. Here we delay the window for 2000 milliseconds (2 seconds). You'll learn a better way to make the window stay in place in tutorial 4.

```
//Free the loaded image
SDL_FreeSurface( hello );

//Quit SDL
SDL_Quit();

return 0;
}
```

Now that we're not going to use the loaded image anymore in our program, we need to remove it from memory. You can't just use `delete`, you have to use `SDL_FreeSurface()` to remove the image from memory. At the end of our program, we call `SDL_Quit()` to shut down SDL.

You may be wondering why we never deleted the screen surface. Don't worry, `SDL_Quit()` cleans it up for you. Congratulations, you've just made your first graphics application.

Troubleshooting your SDL application

If the compiler complains that it can't find 'SDL/SDL.h', it means you forgot to set up your header files. Your compiler/IDE should be looking for the SDL header files, so make sure that it's configured to look in the SDL include

folder.

If you're using Visual Studio and the compiler complains 'SDL/SDL.h': No such file or directory, go to the top of the source code and make sure it says #include "SDL.h".

If your program compiles, but linker complains it can't find some lib files, make sure your compiler/IDE is looking in the SDL lib folder. If your linker complains about an undefined references to a bunch of SDL functions, make sure you linked against SDL in the linker.

If your linker complains about entry points, make sure that you have the main function declared the right way and that you only have one main function combined in your source code.

If the program compiles, links, and builds, but when you try to run it it complains that it can't find SDL.dll, make sure you put SDL.dll in the same directory as the compiled executable. Visual Studio users will need to put the dll file in the same directory as your vcproj file. Windows users can also put the dll inside of the system32 directory.

If you run the program and the images don't show up or the window flashes for a second and you find in stderr.txt:

Fatal signal: Segmentation Fault (SDL Parachute Deployed)

It's because the program tried to access memory it wasn't supposed to. Odds are its because it tried to access NULL when SDL_BlitSurface() was called. This means you need to make sure the bitmap files are in the same directory as the program. Visual Studio users will need to put the bitmap file in the same directory as your vcproj file.

Also if you're using Visual Studio and you get the error "The application failed to start because the application configuration is incorrect. Reinstalling the application may fix this problem. it's because you don't have the service pack update installed. Do not forget to have the latest version of your compiler/IDE with the service pack update for your compiler/IDE or SDL will not work with Visual Studio.

Some Linux users will run and get a blank screen. Try running the program from the command line.

If you had to create a project to compile an SDL program, remember that you will need to create a project for every SDL program you create. Or, better yet, you can reuse the SDL project you made the first time. Download the media and source code for this tutorial [here](#).

56.2.2 Optimized Surface Loading and Blitting

Now that you got an image on the screen in part 2 of the last tutorial, it's time do your surface loading and blitting in a more efficient way.

```
//The headers
#include "SDL/SDL.h"
#include <string>
```

Here are our headers for this program.

SDL.h is included because obviously we're going to need SDL's functions.

The string header is used because ... eh I just like std::string over char*

```
//The attributes of the screen
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_BPP = 32;
```

Here we have the various attributes of the screen.

I'm pretty sure you can figure out what SCREEN_WIDTH and SCREEN_HEIGHT are. SCREEN_BPP is the bits per-pixel. In all of the tutorials, 32-bit color will be used.

```
//The surfaces that will be used
SDL_Surface *message = NULL;
SDL_Surface *background = NULL;
SDL_Surface *screen = NULL;
```

These are the three images that are going to be used.

"background" is obviously going to be the background image, "message" is the bitmap that says "Hello" and "screen" is obviously the screen.

Remember: its a good idea to always set your pointers to NULL if they're not pointing to anything.

```
SDL_Surface *load_image( std::string filename )
{
    //Temporary storage for the image that's loaded
    SDL_Surface* loadedImage = NULL;

    //The optimized image that will be used
    SDL_Surface* optimizedImage = NULL;
```

Here we have our image loading function.

What this function does is load the image, then returns a pointer to the optimized version of the loaded image.

The argument "filename" is the path of the image to be loaded. "loadedImage" is the surface we get when the image is loaded. "optimizedImage" is the surface that is going to be used.

```
//Load the image
loadedImage = SDL_LoadBMP( filename.c_str() );
```

First the image is loaded using `SDL_LoadBMP()`.

But it shouldn't be used immediately because the bitmap is 24-bit. The screen is 32-bit and it's not a good idea to blit a surface onto another surface that is a different format because SDL will have to change the format on the fly which causes slow down.

```
//If nothing went wrong in loading the image
if( loadedImage != NULL )
```

```
{  
    //Create an optimized image  
    optimizedImage = SDL_DisplayFormat( loadedImage );  
  
    //Free the old image  
    SDL_FreeSurface( loadedImage );  
}
```

Next we check if the image was loaded properly. If there was an error, `loadedImage` will be `NULL`.

If the image loaded fine, `SDL_DisplayFormat()` is called which creates a new version of "loadedImage" in the same format as the screen. The reason we do this is because when you try to stick one surface onto another one of a different format, `SDL` converts the surface so they're the same format.

Creating the converted surface every time you blit wastes processing power which costs you speed. Because we convert the surface when we load it, when you want to apply the surface to the screen, the surface is already the same format as the screen. Now `SDL` won't have to convert it on the fly.

So now we have 2 surfaces, the old loaded image and the new optimized image.

`SDL_DisplayFormat()` created a new optimized surface but didn't get rid of the old one.

So we call `SDL_FreeSurface()` to get rid of the old loaded image.

```
//Return the optimized image  
return optimizedImage;  
}
```

Then the newly made optimized version of the loaded image is returned.

```
void apply_surface( int x, int y, SDL_Surface* source, SDL_Surface* destination )  
{  
    //Make a temporary rectangle to hold the offsets
```

```
SDL_Rect offset;  
  
//Give the offsets to the rectangle  
offset.x = x;  
offset.y = y;
```

Here we have our surface blitting function.

It takes in the coordinates of where you want to blit the surface, the surface you're going to blit and the surface you're going to blit it to.

First we take the offsets and put them inside an `SDL_Rect`. We do this because `SDL_BlitSurface()` only accepts the offsets inside of an `SDL_Rect`.

An `SDL_Rect` is a data type that represents a rectangle. It has four members representing the X and Y offsets, the width and the height of a rectangle. Here we're only concerned about x and y data members.

```
//Blit the surface  
SDL_BlitSurface( source, NULL, destination, &offset );  
}
```

Now we actually blit the surface using `SDL_BlitSurface()`.

The first argument is the surface we're using.

Don't worry about the second argument, we'll just set it to `NULL` for now.

The third argument is the surface we're going to blit on to.

The fourth argument holds the offsets to where on the destination the source is going to be applied.

```
int main( int argc, char* args[] )  
{
```

Now we start the main function.

When using SDL, you should always use:

```
int main( int argc, char* args[] )
```

or

```
int main( int argc, char** args )
```

Using `int main()`, `void main()`, or any other kind won't work.

```
//Initialize all SDL subsystems
if( SDL_Init( SDL_INIT_EVERYTHING ) == -1 )
{
    return 1;
}
```

Here we start up SDL using `SDL_Init()`.

We give `SDL_Init()` `SDL_INIT_EVERYTHING`, which starts up every SDL subsystem. SDL subsystems are things like the video, audio, timers, etc that are the individual engine components used to make a game.

We're not going to use every subsystem but it's not going to hurt us if they're initialized anyway.

If SDL can't initialize, it returns -1. In this case we handle the error by returning 1, which will end the program.

```
//Set up the screen
screen = SDL_SetVideoMode( SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_BPP, SDL_SWSURFACE );
```

Now it's time to make our window and get a pointer to the window's surface so we can blit images to the screen.

You already know what the first 3 arguments do. The fourth argument creates the screen surface in system memory.

```
//If there was an error in setting up the screen
if( screen == NULL )
{
    return 1;
}
```

If there was a problem in making the screen pop up, screen will be set to NULL.

```
//Set the window caption
SDL_WM_SetCaption( "Hello World", NULL );
```

Here the caption is set to "Hello World".

The caption is this part of the window:

```
//Load the images
message = load_image( "hello.bmp" );
background = load_image( "background.bmp" );
```

Now the images are loaded using the image loading function we made.

```
//Apply the background to the screen
apply_surface( 0, 0, background, screen );
```

Now it's time to apply the background with the function we made.

Before we blitted the background, the screen looked like this:

But now that we blitted the background image, the screen looks like this in memory:

When you blit, you copy the pixels from one surface onto another. So now the screen has our background image in the top left corner, but we want to fill up the entire screen. Does that mean we have to load the background image 3 more times?

```
apply_surface( 320, 0, background, screen );
apply_surface( 0, 240, background, screen );
apply_surface( 320, 240, background, screen );
```

Nope. We can just blit the same surface 3 more times.

```
//Apply the message to the screen
apply_surface( 180, 140, message, screen );
```

Now we're going to apply the message surface onto the screen at x offset 180 and y offset 140.

The thing is SDL coordinate system doesn't work like this:

SDL's coordinate system works like this:

So the origin (0,0) is at the top left corner instead of the bottom left.

So when you blit the message surface, it's going to blit it 180 pixels right, and 140 pixels down from the origin in the top left corner:

SDL's coordinate system is awkward at first but you'll get used to it.

```
//Update the screen
if( SDL_Flip( screen ) == -1 )
{
    return 1;
}
```

Even though we have applied our surfaces, the screen we see is still blank.

Now we have to update the screen using `SDL_Flip()` so that the screen surface we have in memory matches the one shown on the screen.

If there's an error it will return -1.

```
//Wait 2 seconds  
SDL_Delay( 2000 );
```

We call `SDL_Delay()` so that the window doesn't just flash on the screen for a split second. `SDL_Delay()` accepts time in milliseconds, or 1/1000 of a second.

So the window will stay up for 2000/1000 of a second or 2 seconds.

```
//Free the surfaces  
SDL_FreeSurface( message );  
SDL_FreeSurface( background );
```

```
//Quit SDL  
SDL_Quit();
```

```
//Return  
return 0;
```

```
}
```

Now we do the end of the program clean up.

`SDL_FreeSurface()` is used to get rid of the surfaces we loaded since we're not using them anymore. If we don't free the memory we used, we will cause a memory leak.

Then `SDL_Quit()` is called to quit SDL. Then we return 0, ending the program.

You may be asking yourself "why aren't we freeing the screen surface?". Don't worry. `SDL_Quit()` will take care of that for us. If you run the program and the images don't show up or the window flashes for a second and you find in `stderr.txt`:

```
Fatal signal: Segmentation Fault (SDL Parachute Deployed)
```

It's because the program tried to access memory it wasn't supposed to. Odds are it's because it tried to access NULL when `apply_surface()` was called. This means you need to make sure the bitmap files are in the same directory as the program.

If the window pops up and the image doesn't show up, again make sure the bitmaps are in the same folder as the program or in the project directory.

If you're using Visual Studio and the compiler complains about 'SDL/SDL.h': No such file or directory, go to the top of the source code and make sure it says `#include "SDL.h"`.

Also if you're using Visual Studio and you get the error "The application failed to start because the application configuration is incorrect. Reinstalling the application may fix this problem." it's because you don't have the service pack update installed. Do not forget to have the latest version of your compiler/IDE with the service pack update for your compiler/IDE or SDL will not work with Visual Studio.

Download the media and source code for this tutorial [here](#).

56.2.3 Extension Libraries and Loading Other Image Formats

SDL only supports .bmp files natively, but using the `SDL_image` extension library, you'll be able to load BMP, PNM, XPM, LBM, PCX, GIF, JPEG, TGA and PNG files.

Extension libraries allow you to use features that basic SDL doesn't support natively. Setting up an SDL extension library isn't hard at all, I'd even say it's easier to set up than basic SDL. This tutorial will teach you how to use the `SDL_image` extension library.

▪ Windows

Linux

56.2.4 Event Driven Programming

Up until this point you're probably used to command driven programs using cin and cout. This tutorial will teach you how to check for events and handle events.

An event is simply something that happens. It could be a key press, movement of the mouse, resizing the window or in this case when the user wants to X out the window.

```
//The headers
#include "SDL/SDL.h"
#include "SDL/SDL_image.h"
#include <string>

//Screen attributes
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_BPP = 32;

//The surfaces
SDL_Surface *image = NULL;
SDL_Surface *screen = NULL;
```

Here we have the same story as before, we have our headers, constants and surfaces.

```
//The event structure that will be used
SDL_Event event;
```

Now this is new. A `SDL_Event` structure stores event data for us to handle.

```
SDL_Surface *load_image( std::string filename )
{
    //The image that's loaded
    SDL_Surface* loadedImage = NULL;

    //The optimized image that will be used
    SDL_Surface* optimizedImage = NULL;

    //Load the image
    loadedImage = IMG_Load( filename.c_str() );

    //If the image loaded
    if( loadedImage != NULL )
    {
        //Create an optimized image
        optimizedImage = SDL_DisplayFormat( loadedImage );

        //Free the old image
        SDL_FreeSurface( loadedImage );
    }

    //Return the optimized image
    return optimizedImage;
}
```

```
void apply_surface( int x, int y, SDL_Surface* source, SDL_Surface* destination )
{
    //Temporary rectangle to hold the offsets
    SDL_Rect offset;

    //Get the offsets
    offset.x = x;
    offset.y = y;

    //Blit the surface
    SDL_BlitSurface( source, NULL, destination, &offset );
}
```

Here we have our surface loading and blitting functions. Nothing has changed from the previous tutorial.

```
bool init()
{
    //Initialize all SDL subsystems
    if( SDL_Init( SDL_INIT_EVERYTHING ) == -1 )
    {
        return false;
    }

    //Set up the screen
    screen = SDL_SetVideoMode( SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_BPP, SDL_SWSURFACE );

    //If there was an error in setting up the screen
```

```
if( screen == NULL )
{
    return false;
}

//Set the window caption
SDL_WM_SetCaption( "Event test", NULL );

//If everything initialized fine
return true;
}
```

Here is the initialization function. This function starts up SDL, sets up the window, sets the caption and returns false if there are any errors.

```
bool load_files()
{
    //Load the image
    image = load_image( "x.png" );

    //If there was an error in loading the image
    if( image == NULL )
    {
        return false;
    }

    //If everything loaded fine
```

```
    return true;  
}
```

Here is the file loading function. It loads the images, and returns false if there were any problems.

```
void clean_up()  
{  
    //Free the image  
    SDL_FreeSurface( image );  
  
    //Quit SDL  
    SDL_Quit();  
}
```

Here we have the end of the program clean up function. It frees up the surface and quits SDL.

```
% int main( int argc, char* args[] )  
% {  
%     //Make sure the program waits for a quit  
%     bool quit = false;
```

Now we enter the main function.

Here we have the quit variable which keeps track of whether the user wants to quit. Since the program just started we set it to false or the program will end immediately.

```
%     //Initialize  
%     if( init() == false )  
%     {
```

```
%         return 1;
%
%     }
%
%     //Load the files
%     if( load_files() == false )
%     {
%         return 1;
%     }
```

Now we call the initialization and file loading functions we made earlier.

```
%     //Apply the surface to the screen
%     apply_surface( 0, 0, image, screen );
%
%     //Update the screen
%     if( SDL_Flip( screen ) == -1 )
%     {
%         return 1;
%     }
```

Then we show the image on the screen.

```
%     //While the user hasn't quit
%     while( quit == false )
%     {
```

Now we start the main loop. This loop will keep going until the user sets quit to true.

```
%        //While there's an event to handle
%        while( SDL_PollEvent( &event ) )
%
{
```

In SDL whenever an event happens, it is put on the event queue. The event queue holds the event data for every event that happens.

So if you were to press a mouse button, move the mouse around, then press a keyboard key, the event queue would look something like this:

What `SDL_PollEvent()` does is take an event from the queue and sticks its data in our event structure:
What this code does is keep getting event data while there's events on the queue.

```
%        //If the user has Xed out the window
%        if( event.type == SDL_QUIT )
%
%        {
%            //Quit the program
%            quit = true;
%
%        }
%
%    }
```

When the user Xs out the window, the event type will be `SDL_QUIT`.

But when the user does that it does not end the program, all it does inform us the user wants to exit the program.

Since we want the program to end when the user Xs the window, we set `quit` to true and it will break the loop we are in.

```
%        //Free the surface and quit SDL
%        clean_up();
%
```

```
%     return 0;  
% }
```

Finally, we do the end of the program clean up.

There are other ways to handle events like `SDL_WaitEvent()` and `SDL_PeepEvents()`. You can find out more about them in the SDL documentation. Download the media and source code for this tutorial here.

On a side note, now would also be a good time to learn to use the SDL error functions. I don't have a tutorial on them, but I touch on them in article 5. The SDL documentation should explain `SDL_GetError()`, and the `SDL_image` documentation should explain `IMG_GetError()`. `SDL_ttf` and `SDL_mixer` also have their error functions so remember to look those up in their documentations.

56.2.5 Color Keying

56.2.6 Clip Blitting and Sprite Sheets

56.2.7 True Type Fonts

56.2.8 Key Presses

56.2.9 Mouse Events

56.2.10 Key States

56.2.11 Playing Sounds

56.2.12 Timing

56.2.13 Advanced Timers

56.2.14 Regulating Frame Rate

56.2.15 Calculating Frame Rate

56.2.16 Motion

56.2.17 Collision Detection

56.2.18 Per-pixel Collision Detection

56.2.19 Circular Collision Detection

56.2.20 Animation

56.3.1 Графический Hello World

user/sdl_hello.c

```
1 #define SPLASH_DELAY 3000
2
3 #include "SDL/SDL.h"
4
5 int main() {
6     // init SDL
7     if (SDL_Init(SDL_INIT_VIDEO)) { // SDL_INIT_EVERYTHING) {
8         fprintf(stderr, "\nSDL_GetError: %s\n", SDL_GetError());
9         abort();
10    }
11    // start window/fullscreen
12    SDL_Surface* screen = SDL_SetVideoMode( 640, 480, 0, SDL_SWSURFACE );
13    // hide mouse cursor
14    SDL_ShowCursor(0);
15    // load hello image
16    SDL_Surface* hello = SDL_LoadBMP( "/share/hello.bmp" );
17    // copy image to screen
18    SDL_BlitSurface( hello, NULL, screen, NULL );
19    // update sdl window
20    SDL_Flip( screen );
21    // delay 5 seconds
22    SDL_Delay( SPLASH_DELAY );
23    // quit
24    SDL_Quit();
25    return 0;
```


56.3.2 Вывод случайных прямоугольников

user/sdl_rect.c

```
1 // #include <stdio.h>
2 // #include <stdlib.h>
3 // #include <unistd.h>
4 #include <assert.h>
5 // #include <time.h>
6
7 #include <SDL/SDL.h>
8
9 int main() {
10    srand(time(0));
11    if (SDL_Init(SDL_INIT_VIDEO)) {
12        fprintf(stderr, "\nSDL_GetError:%s\n", SDL_GetError());
13        abort();
14    }
15    SDL_Surface* screen = SDL_SetVideoMode(0, 0, 0, SDL_HWSURFACE);
16    assert(screen!=NULL);
17    SDL_Rect rect = screen->clip_rect;
18    int R,G,B,X,Y,U,V;
19    SDL_Rect quad;
20
21    SDL_FillRect(screen,&screen->clip_rect,
22                  SDL_MapRGB(screen->format,0,0,0));
23    SDL_Flip(screen );
24
25    // wait keypress
```

```
26     SDL_Event event;
27     int done=0;
28     for (done=0;done==0;) {
29         R=rand()%0x100; G=rand()%0x100; B=rand()%0x100;
30         X=rand()%rect.w; Y=rand()%rect.h;
31         U=rand()%rect.w; V=rand()%rect.h;
32         quad.x=X; quad.y=Y; quad.w=U; quad.h=V;
33         SDL_FillRect(screen,&quad,SDL_MapRGB(screen->format,R,G,B));
34         SDL_Flip(screen);
35     while (SDL_PollEvent(&event)) {
36         switch (event.type) {
37             case SDL_QUIT:
38             case SDL_KEYDOWN:
39                 done=1;
40             }
41     }
42 }
43     SDL_Quit();
44     return 0;
45 }
```

56.4 Версия SDL2

Глава 57

BuildRoot

Глава 58

Особенности OpenWrt

Часть XIV

x86os: простая операционная система
для компьютера на i386 процессоре

Глава 59

Ресурсы

http://www.jamesmolloy.co.uk/tutorial_html/index.html

<http://wiki.osdev.org>

Библиотека системного программиста © Александр Фролов, Григорий Фролов:

- Операционная система MS-DOS. Том 1, книги 1-2
- Операционная система MS-DOS. Том 1, книга 3
- Аппаратное обеспечение IBM PC. Том 2, книга 1
- Программирование видеоадаптеров CGA, EGA и VGA
- Защищенный режим процессоров Intel 80286/80386/80486. Том 4

Графика:

- Уилтон Р.
Видеосистемы персональных компьютеров IBM PC и PS/2.
Радио и связь, 1994

- С.А.Васильев Программирование видеосистем, ТГТУ, 2003
- Е.В. Шикин, А.В. Боресков
 - Компьютерная графика. Динамика, реалистические изображения. Диалог-МИФИ, 1996
 - Компьютерная графика. Полигональные модели. Диалог-МИФИ, 2001
 - Начала компьютерной графики. Диалог-МИФИ, 1993

Глава 60

Структура

- кросс-компилятор
- загрузчик
- микроядро
- драйвера
- библиотеки
- прикладные программы

Глава 61

Процесс запуска

1. BIOS
2. boot0 первый сектор загрузочного диска, считывает boot1
3. boot1 основная часть загрузчика, в файле на загрузочном диске
4. ядро ОС
5. запуск драйверов
6. запуск системных демонов (сервисы)
7. запуск GUI (если есть) или пользовательской консоли

Глава 62

Сборка кросс-компилиатора

```
git clone --depth=1 -o gh https://github.com/ponyatov/x86os.git
cd x86os
make dirs
make gz

make cross
```

cfg.mk

```
1 CPU = i486
2 TCFLAGS = -g0 -O0 -nostdlib -ffreestanding -Tx86os.ld -std=c99
```

mk/versions.mk

```
1 # cross compiler
```

```
2 BINUTILS_VER = 2.24
3 GMP_VER = 5.1.3
4 MPFR_VER = 3.1.2
5 MPC_VER = 1.0.2
6 GCC_VER = 4.9.1
7 # libc
8 NEWLIB_VER = 2.2.0-1
9 # loader
10 SYSLINUX_VER = 6.03
```

mk/packages.mk

```
1 # cross compiler
2 BINUTILS = binutils-$(BINUTILS_VER)
3 GMP = gmp-$(GMP_VER)
4 MPFR = mpfr-$(MPFR_VER)
5 MPC = mpc-$(MPC_VER)
6 GCC = gcc-$(GCC_VER)
7 # libc
8 NEWLIB = newlib-$(NEWLIB_VER)
9 # loader
10 SYSLINUX = syslinux-$(SYSLINUX_VER)
```

mk/dirs.mk

```
1 GZ = $(PWD)/gz
2 TC = $(PWD)/$(CPU).cross
3 SRC = $(PWD)/src
4 TMP = $(PWD)/tmp
5
```

```
6 DIRS = $(GZ) $(TC) $(SRC) $(TMP)
7 .PHONY: dirs
8 dirs:
9   mkdir -p $(DIRS)
```

mk/src.mk

```
1 .PHONY: gz
2 gz:
3   $(WGET) http://ftp.gnu.org/gnu/binutils/$(BINUTILS).tar.bz2
4   $(WGET) http://gcc.skazkaforyou.com/releases/$(GCC)/$(GCC).tar.bz2
5   $(WGET) https://gmplib.org/download/gmp/$(GMP).tar.bz2
6   $(WGET) http://www.mpfr.org/mpfr-current/$(MPFR).tar.bz2
7   $(WGET) http://www.multiprecision.org/mpc/download/$(MPC).tar.gz
8   $(WGET) ftp://sourceware.org/pub/newlib/$(NEWLIB).tar.gz
9   $(WGET) https://www.kernel.org/pub/linux/utils/boot/syslinux/$(SYSLINUX).tar.xz
10
11 $(SRC)/%/README: $(GZ)/%.tar.gz
12   cd $(SRC) && zcat $< | tar x && touch $@
13 $(SRC)/%/README: $(GZ)/%.tar.bz2
14   cd $(SRC) && bzcat $< | tar x && touch $@
15 $(SRC)/%/README: $(GZ)/%.tar.xz
16   cd $(SRC) && xzcat $< | tar x && touch $@
```

mk/cross.mk

```
1 TARGET = $(CPU)-elf
2 CCACHE = ccache
3
4 CFG = configure --prefix=$(TC) --disable-nls --disable-werror \
```

```
5 ____CC=$(CCACHE) gcc -pipe" \
6 ____--infodir=$(TMP)/info --mandir=$(TMP)/man --docdir=$(TMP)/doc
7
8 .PHONY: cross
9 cross: binutils cclibs gcc
10
11 CFG_BINUTILS = --target=$(TARGET) --enable-lto \
12 ____--with-sysroot=$(PWD) --with-native-system-header-dir=/include
13
14 .PHONY: binutils
15 binutils: $(SRC)/$(BINUTILS)/README
16 ____rm -rf $(TMP)/$(BINUTILS) && mkdir $(TMP)/$(BINUTILS) && \
17 ____cd $(TMP)/$(BINUTILS) && \
18 ____$(SRC)/$(BINUTILS)/$(CFG) $(CFG_BINUTILS) && \
19 ____$(MAKE) && $(INSTALL)-strip
20
21 include mk/cclibs.mk
22
23 CFG_GCC = $(CFG_BINUTILS) $(CFG_CCLIBS) \
24 ____--without-headers --with-newlib \
25 ____--enable-languages="c,c++"
26
27 .PHONY: gcc
28 gcc: $(SRC)/$(GCC)/README
29 ____rm -rf $(TMP)/$(GCC) && mkdir $(TMP)/$(GCC) && \
30 ____cd $(TMP)/$(GCC) && \
31 ____$(SRC)/$(GCC)/$(CFG) $(CFG_GCC)
32 ____cd $(TMP)/$(GCC) && $(MAKE) all-gcc
```

```
33 ____cd $(TMP)/$(GCC) && $(MAKE) install-gcc
34 ____cd $(TMP)/$(GCC) && $(MAKE) all-target-libgcc
35 ____cd $(TMP)/$(GCC) && $(MAKE) install-target-libgcc
```

mk/cclibs.mk

```
1 CFG_CCLIBS = --disable-shared --with-gmp=$(TC)
2 .PHONY: cclibs
3 cclibs: gmp mpfr mpc
4
5 CFG_GMP = $(CFG_CCLIBS)
6 .PHONY: gmp
7 gmp: $(SRC)/$(GMP)/README
8 ____rm -rf $(TMP)/$(GMP) && mkdir $(TMP)/$(GMP) && \
9 ____cd $(TMP)/$(GMP) && \
10 ____$(SRC)/$(GMP)/$(CFG) $(CFG_GMP) && \
11 ____$(MAKE) && $(INSTALL)-strip
12
13 CFG_MPFR = $(CFG_CCLIBS)
14 .PHONY: mpfr
15 mpfr: $(SRC)/$(MPFR)/README
16 ____rm -rf $(TMP)/$(MPFR) && mkdir $(TMP)/$(MPFR) && \
17 ____cd $(TMP)/$(MPFR) && \
18 ____$(SRC)/$(MPFR)/$(CFG) $(CFG_MPFR) && \
19 ____$(MAKE) && $(INSTALL)-strip
20
21 CFG_MPC = $(CFG_CCLIBS)
22 .PHONY: mpc
23 mpc: $(SRC)/$(MPC)/README
```

```
24 rm -rf $(TMP)/$(MPC) && mkdir $(TMP)/$(MPC) && \
25 cd $(TMP)/$(MPC) && \
26 $(SRC)/$(MPC)/$(CFG) $(CFG_MPC) && \
27 $(MAKE) && $(INSTALL)-strip
```

Как вы можете заметить, структура файлов и каталогов похожа на [azlin /54/](#).

Глава 63

Формат ELF32

Подробнее см. формат ELF 22 и скрипты линкера 24.4

Собранный кросс-компилятор генерирует исполняемые файлы, т.е. файл ядра, в формате ELF. Содержание можно посмотреть в файле `kernel.objdump`, который создается при выполнении команды

```
make kernel
```

формат файла elf32-i386

архитектура i386

HAS_SYMS в файл включена отладочная информация
об идентификаторах (“символах”)

start address 0x00100000 стартовый адрес загрузки ядра 1 Мб

Бинарный код делится на секции, или сегменты:

.text машинный код программы
.rodata данные: константы
.eh_frame
.data данные: инициализированные массивы, строки
.bss данные: пустые массивы под которые выделяется память при старте
.comment

CONTENTS

ALLOC

LOAD

READONLY

CODE

DATA

Глава 64

multiboot

Для запуска ОС не нужно писать свой загрузчик: с этим легко справится любой современный универсальный загрузчик, поддерживающий стандарт [multiboot](#): GRUB или SysLinux.

загрузчик SysLinux

Спецификация Multiboot 0.6.96

Перевод Multiboot Specification

Для упрощения некоторые файлы, в т.ч. multiboot, были заимствованы из исходных текстов syslinux:

`src/syslinux-6.03/gpxe/src/arch/i386/include/multiboot.h`

`kernel/x86os.ld`

1 ENTRY(_multiboot)

2

```
3 SECTIONS {
4
5 . = 1M;      /* load @1Mb high mem */
6
7 .text BLOCK(4K) : ALIGN(4K) {
8     *(.multiboot)
9     *(.text)
10}
11
12.stack BLOCK(4K) : ALIGN(4K) { *(.stack) }
13.rodata BLOCK(4K) : ALIGN(4K) { *(.rodata) }
14.data BLOCK(4K) : ALIGN(4K) { *(.data) }
15.bss BLOCK(4K) : ALIGN(4K) { *(.bss) }
16
17/DISCARD/ : { *(.eh_frame) *(.comment) }
18
19}
```

kernel/multiboot.S

```
1 // multiboot-compliant assembly start-up
2
3.set MAGIC,    0x1BADB002      // multiboot signature
4.set ALIGN,    1<<0           // page align memory segments
5.set MEMINFO,  1<<1           // memory map
6.set FLAGS,    ALIGN | MEMINFO
7.set CHECKSUM, -(MAGIC + FLAGS)
8
9.section .multiboot
```

```
10 .align 4
11 .long MAGIC
12 .long FLAGS
13 .long CHECKSUM
14
15 // .text
16 .global _multiboot
17 _multiboot:
18     cli
19     movl $stack_top, %esp
20     call _start
21     cli
22 .halt:
23     hlt
24     jmp .halt
25
26 .section .stack, "aw", @nobits
27 .skip 16*1024
28 stack_top:
```

Глава 65

Микроядро

```
make kernel  
make emu
```

kernel/kernel.mk

```
1 KERNEL = kernel/multiboot.S  
2 KERNEL += kernel/kernel.c  
3  
4 .PHONY: kernel  
5 kernel: kernel.elf  
6 kernel.elf: $(KERNEL) $(DRIVER) $(USER) $(HEADER) cfg.mk x86os.ld  
7 $(TCC) $(TCFLAGS) -o $@ $(KERNEL) $(DRIVER) $(USER)  
8 $(OBJDUMP) -xd $@ > kernel.objdump
```

kernel/kernel.c

```
1 #include <portio.h>
2 #include <driver/vga.h>
3 #include <user/user.h>
4
5 void __start() {
6     vga_init();
7     vga_write("x86os@https://github.com/ponyatov/Azbuka/blob/master/Azbuka.pdf?raw=true");
8     user();
9 }
```

Глава 66

Драйвера

66.1 **vga**: текстовая консоль VGA 80×25

include/driver/vga.h

```
1 #ifndef _H_VGA
2 #define _H_VGA
3
4 #define VGA_COLS 80
5 #define VGA_ROWS 25
6
7 #define VGA_ADDR 0xB8000
8
9 #define COLOR_BLACK 0
10 #define COLOR_BLUE 1
11 #define COLOR_GREEN 2
```

```
12#define COLOR_CYAN 3
13#define COLOR_RED 4
14#define COLOR_MAGENTA 5
15#define COLOR_BROWN 6
16#define COLOR_LIGHT_GREY 7
17#define COLOR_DARK_GREY 8
18#define COLOR_LIGHT_BLUE 9
19#define COLOR_LIGHT_GREEN 10
20#define COLOR_LIGHT_CYAN 11
21#define COLOR_LIGHT_RED 12
22#define COLOR_LIGHT_MAGENTA 13
23#define COLOR_LIGHT_BROWN 14
24#define COLOR_WHITE 15
25
26void vga_init();
27
28#endif
```

driver/vga.c

```
1 // Std.VGA 80x25 text console driver
2
3#include <stdint.h>
4#include <driver/vga.h>
5
6const uint8_t vga_cols = VGA_COLS;
7const uint8_t vga_rows = VGA_ROWS;
8
9uint8_t *vga_buf = (uint8_t *) (VGA_ADDR);
```

```
10
11 uint8_t vga_fg = COLOR_WHITE;
12 uint8_t vga_bg = COLOR_DARK_GREY;
13
14 uint8_t vga_cursor_row = 0;
15 uint8_t vga_cursor_col = 0;
16
17 void vga_init() {
18     vga_cursor_row = 0;
19     vga_cursor_col = 0;
20     for (int i = 0; i < VGA_COLS * VGA_ROWS; i++)
21         vga_buf[i * 2 + 1] = COLOR_LIGHT_GREY | vga_bg << 4;
22 }
23
24 void vga_char(char c) {
25     if (c == '\n') {
26         while (vga_cursor_col > 0)
27             vga_char(' ');
28     } else {
29         int ptr = vga_cursor_row * VGA_COLS + vga_cursor_col;
30         vga_buf[ptr * 2 + 0] = c;
31         vga_buf[ptr * 2 + 1] = vga_fg | vga_bg << 4;
32         vga_cursor_col++;
33         if (vga_cursor_col > VGA_COLS) {
34             vga_cursor_col = 0;
35             vga_cursor_row++;
36             if (vga_cursor_row > VGA_ROWS)
37                 vga_cursor_row = 0;
```

```
38     }
39 }
40 }
41
42 void vga_write( char *msg) {
43     for (int i = 0; i <= VGA_COLS * VGA_ROWS; i++) {
44         if (msg[i] == 0x00)
45             break;
46         else
47             vga_char(msg[ i ]);
48     }
49 }
```

66.2 **kbd**: клавиатура

66.3 **ide**: жесткий диск IDE

66.3.1 **fatfs**: файловая система FAT16

Часть XV

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 → 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 67

Общие сведения о компьютерной математике

12

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 запятой³. 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 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.

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

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 понадобиться 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

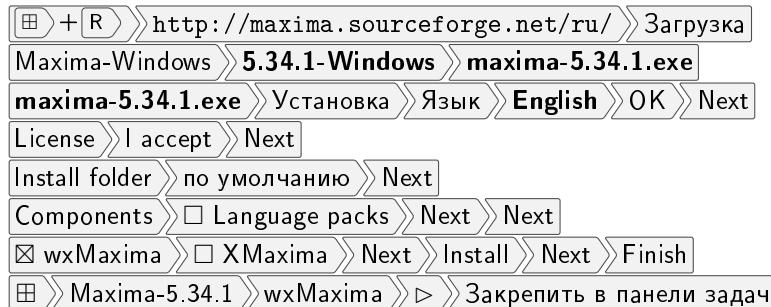
и для ВУЗа, и для студентов и преподавателей иметь в своем распоряжении легальные копии такого ПО без больших, и даже каких-либо денежных затрат.

Глава 68

Пакет Maxima

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

68.1 Установка Maxima под Windows



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 [?] получена из файла .ps с помощью сервиса <http://ps2pdf.com/convert.htm>.

68.2 Калькулятор

Часть XVI

Подготовка технической документации

Глава 69

Верстка в LATEX

Глава 70

Оформление листингов

Глава 71

Подготовка иллюстраций

71.1 GIMP

71.2 Inkscape

71.3 Graphviz

Глава 72

Замечания для соавторов “Абзуки
АРМатурщика”

Часть XVII

Примерные учебные планы

Глава 73

Блондинко

Глава 74

Школотрон

1. ??; ??

постараться понять разделы

??

??

??

??

Текст может оказаться сложноватым, но эти понятия необходимы для понимания электроники, как минимум нужно понимать что такое ток, напряжение, сопротивление и мощность.

2. ??; ??

Умение хотя бы немного пользоваться простейшим измерительным прибором необходимо: проверить батарейку, посмотреть ток через элемент, определить ноги светодиода и т.п.

Глава 75

Студень

1. ??

Глава 76

Технический специалист

1. ?? прочитать по диагонали

Часть XVIII

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.

Глава 77

Автоматное программирование /фреймворк QuantumLeaps/

Глава 78

Запуск Linux на android устройстве в режиме паразита

¹ © Mikael Q Kuisma

How to run Debian or Ubuntu GNU/Linux on your Android

Integrating GNU/Linux with Android The Matrix Way The most seamless way extending your Android device with a full blown GNU/Linux such as Debian (or Ubuntu) is running the Android system in a chroot environment in the Debian file system. This way you can access the Android system from Debian without restrictions at the same time no modifications to the Android system itself are needed.

¹ копипаста: <http://whiteboard.ping.se/Android/Debian>

This description requires general computer skills such as GNU/Linux but not necessary specific knowledge about Android - but it sure helps. You will have to install the Android SDK toolkit, and if you are not comfortable running pre-compiled binaries from dubious sources, you may have to get at least parts of the Android OS source code as well.

A new init procedure mounting a new root file system, transferring control to the Android init in a chroot environment is implemented as described here below. The pros compared to other methods are many.

Features

- Full GNU/Linux Debian installation with lots of apt-get:able packages
- Full control of the Android environment from Debian
- Simultaneous use of Debian as well as Android
- Access the Android file system from your workstation desktop via ssh/sftp
- No need to unmount/remount the SDcard accessing it via ssh/sftp
- Makes it easy to backup both the Android as well as the Debian system
- Android system untouched and unaware of any modifications
- Android root file system no longer volatile, edits are kept between reboots
- Critical file systems kept on SDcard for easy access in case of major f**k up
- Graphic X11 Windows user interface, both client and server, local and remote, native, over SSH or VNC
- Zero performance impact
- Easy to modify your Android ROM selectively, without the need to reflash the entire device.
- Manage your Android device as any other GNU/Linux system

Requirements (click on each topic for more info)

- Android SDK toolkit
- Your Android device boot image and the ability to flash your device (or your favorite ROM)
- A static linked binary of busybox for the ARM architecture
- An SDcard, preferable fast (class 10) and large capacity (32GB)
- A GNU/Linux machine with an SDcard reader
- Root access on the Android device (makes things smoother, but in the end you'll get it anyway)

Steps

- Partition an SDcard into two partitions, one FAT, one GNU/Linux (e.g. ext3 or ext4)
- Create a new initramfs to flash the device with
- Create a Debian root file system on the second partition of the SDcard
- Copy the original Android root file system to /android in the Debian file system tree

Disclaimer - The instructions here are not for your device explicit, and you can not follow them by the letter, but have to adjust them for your telephone or tablet. Most often I've highlighted what you may need to change. If you're not experienced flashing your phone there's also a risk you render it useless, becoming the proud owner of an expensive brick. This solution is primarily intended for the experienced GNU/Linux hacker, system administrator or app developer wanting full control over the Android device using a standard GNU/Linux environment. For the novice wanting to run GNU/Linux on his mobile device for the fun of it, there are other less powerful solutions I'd recommend before this one.

Partition the SDcard

Get a large capacity SDcard and create two partitions. Make sure it's fast as well (class 10). Make the first partition the standard FAT file system used by various apps. Make the second a GNU/Linux partition for the Debian root filesystem. Use ext4 if your Android kernel supports it, else chose the best supported. Look in /proc/filesystems, /proc/config.gz or so on your device. Partitions the SDcard on your ordinary GNU/Linux machine using fdisk. Keep the sector boundaries aligned with the factor as the first partition on the card when shipped. This will ensure partition alignment for best performance. Some solid state disks gets terrible performance unaligned.

At your desktop computer

```
root@workstation:~# fdisk -cu /dev/sdf
```

```
Command (m for help): p
```

```
Disk /dev/sdf: 32.0 GB, 32018268160 bytes
170 heads, 53 sectors/track, 6940 cylinders, total 62535680 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdf1		53	13839359	6919653+	c	W95 FAT32 (LBA)
/dev/sdf2		13839360	62529399	24345020	83	Linux

Here the alignment is 53 sectors, i.e. even tracks, so I make sure the start sector of both partitions is a multiple of this. I chose to split 32G into 8G FAT plus 24G ext4. The Debian base environment including X11 is just above 1GB in size, so 24G may be overkill.

Create new file systems with mkfs.

At your desktop computer

```
# mkfs -t vfat /dev/sdf1  
# mkfs -t ext4 /dev/sdf2
```

Creating the initramfs

Replace the initramfs shipped with your device with your own modified. Use an init mounting a new root file system from the SDcards GNU/Linux partition and transfer control to this.

Below is an example of the /init of the initramfs file system. It must be named /init because this is hardcoded into the Android kernel to execute upon boot. You most certainly need to change /dev/mmcblk1p2 to the name of your SDcard second partition.

On The ASUS Transformer TF101 the second partition is named /dev/mmcblk1p2 but look at your device to see what your is called.

From your desktop shell to your Android device

Ok, here on the Sony Ericsson Xperia Active we see the FAT partition /mnt/sdcard is device 179,1 hence the next partition must be 179,2 and it's named mmcblk0p2. Note that during our init, busybox creates the device node directly in /dev without the /block/ directory Android uses.

/init in your new initramfs

Precompiled busybox from busybox.net

The file system of this initramfs is very minimalistic and only contains the /sbin/busybox and the mount points /proc, /sys, /dev and /mnt/root. Or to be on the safe side, use the original initramfs and just add /sbin/busybox, a mount point /mnt/root and replace init with the script above.

We'll need the systems base address, i.e. where the RAM begins. To get it from your original kernel zImage, check for /proc/config.gz in your running kernel or use the extract-ikconfig script on the kernel binary. This script is included in the kernel source.

The kernel zImage is your original kernel. mkbootimg is a part of Android OS build but can be found pre-compiled at various sites. You can download your original boot image from your device, from your vendor or from less official sources on the Internet, all depending on your type of phone and its openness. One feature of having a locked bootloader, is that if an image is flashable, it must be genuine (i.e. its signature verifies). XDA Forum is a good starting point, whatever method you chose.

If you prefer running a custom ROM (e.g. CyanogenMod), you can of course use its boot image instead of the device's original ROM.

- Utility to unpack the boot image (gives you the base address as well)
- XDA Forums
- Howto Unpack, Edit, and Re-Pack Boot Images
- How to compile mkbootimg
- Finding the base address

We sit on our newly created image my-boot.img for a while now, while finishing the rest. Do not flash it yet.

Creating the Debian root file system

Mount your SDcard, if not already mounted. I assume you've mounted it as /mnt/debian. If you prefer Ubuntu or some other Debian based distribution, the steps are the same. Replace the mirrors accordingly.

Chose a Debian mirror close to you, and begin to create the Debian root filesystem.

At your desktop computer

This is only half of the Debian installation. Now we need to complete the other half. Either run this in an emulator (qemu-system-arm or Android Virtual Device (AVD)) or maybe easier, directly at your Android device as root. Here I'm using /root as a temporary mount point on the device since it happens to be unused in Android (actually, read-only).

On your Android device

Here you'd actually ran Debian on your device! But chroot:ed below Android, we want the reverse. But now we got a complete GNU/Linux system with SSH server and all. Still some tinkering needs to be done. If apt-get update didn't work, check your /etc/named.conf.

Here below you find a ready-made root file system up to this point described, i.e. with a ssh-server installed. The root password is "root". Feel free to change it. ☺

- Ready made root file system (128MB)

Note that this file system is as of up to this point in this tutorial. It still needs work, e.g. installing your Android root in /android, adding the init scripts etc. This can't be pre-made, since they differs from device to device.

Creating the new Android root file system

Mount the SDcard in your ordinary GNU/Linux machine again.

Unpack the original boot image initramfs to /android on SDcard GNU/Linux partition. This is the new Android root. Create directory /android/log. Note that since the new Android root here isn't a mount point but a subdirectory, Android will not succeed re-mounting it as read-only. If you believe this is a problem, you can instead create the Android root on a separate partition on the SDcard, mounting it as /mnt/root/android in the init on the initramfs above directly after mounting /mnt/root. Note that in this case, /android/log may not be used for boot logs by /etc/init below, since it's read-only. You may solve this by mounting a tmpfs or simply remove the logging by /etc/init.

Android normally only accepts four partitions on the SDcard (void limitation). If you don't want to waste one of them for the small root file system, you can loopback mount (-bind) /android to /mnt/android making it a mount point. This mount point you then can set to read-only using remount. Note that you must do a remount, because a bind-mount can not change the flags of the original file system initially. You'll have to do this remount explicit yourself in init.stage2 using /bin/mount in this case. But initially I suggest you just let the root be writeable until you get everything up and running. This can be done later – or not at all.

Making a boot image is done with the Android OS build kit mkbootimg. There's no official tool splitting such an image, but it's quite trivial and lots of scripts available to do this. The image is basically just a concatenation of the kernel zImage and initramfs.cpio.gz.

- My utility to unpack the boot image
- More info about unpacking an Android boot image

Some finishing scripts to tie all together

Our new initramfs transfer init control to /etc/init on the GNU/Linux partition. Use this script below. You also need to copy the busybox to /sbin.

/etc/init of SDcard ext4 filesystem

Also make sure you copy the busybox to /sbin in this file system as well. Note that log from init.stage2 is stored in the Android file tree so you can access it from Android in case the Debian-level ssh server didn't start due to some mistake done in for example /etc/rc.local.

What this script does, is forking of a secondary delayed script the Debian environment executes once the Android init is done. It then transfers control to the Android original init, still running as pid 1 of course.

The secondary script init.stage2

/etc/init.stage2 of the SDcard ext4 filesystem

Basically this only waits on Android init, then sets up everything necessary for Debian such as devices, proc and sys mounts, and executes /etc/rc.local.

You see we mount /dev loopback from the Android root. Because of this, you must remove any devices in /dev populated by debootstrap, or else this mount will fail.

My /etc/rc.local looks like below.

/etc/rc.local of the SDcard ext4 filesystem

Note that init makes sure everything here is logged to /android/log/boot.log. This is in the case the ssh-server does not start, you may see why in the file /log/boot.log by adb shell to Android.

Install it

If everything went well so far, it's time to install your customised boot image. Here below I assume you have an unlocked bootloader supporting fastboot, but you might have to use some other tool to flash your phone depending on model.

First enter fastboot mode on your Android device. This is done with some magic key combination during power off and is phone specific. You may try VolumeUp or VolumeDown as you either turn on the phone or connect its USB cord to the computer - or Google your phone model plus "fastboot".

On you desktop computer

The marked i 0x0fc" tells fastboot the vendor of the device to flash and you must change this matching your phone. You don't want to flash wrong Android device. If you are sure only the right one is connected (see with "fastboot devices") you may exclude this parameter.

All done, you now run Debian integrated with Android The Matrix Way. Run ssh to it as user root with the password you specified.

Additional tinkering

/etc/group

The Android environment is quite restricted. If you plan to run as non-root in the Debian environment, you'll need to add yourself to some Android groups to get access to network and such. The groups of the Android user shell serves as a template. Most important are the inet group 3003 to get network access and 1015 to write on the SDcard.

On an android device as user shell

The complete set of Android user uid and group gid can by found in system/core/include/private/android_filesystem_(yes, it's hard-coded).

/etc/mtab

To make df happy, make this a symlink to /proc/mounts

Still, df will produce a somewhat confusing output due to the double mounts of devices in the different roots. Not to worry, this is only cosmetic.

locales

You don't get any localised locale installed by default. If you'd like that, apt-get install locales, edit /etc/locale.gen to select what locale you'd like, then run locale-gen.

Setting the system default time zone

Using the GNU/Linux Debian environment

Connectivity

To get a Debian terminal, download the ConnectBot from Google Play and ssh-connect to localhost. Note that if you use the "local" connection in ConnectBot, you'll enter The Matrix, i.e. the chroot Android environment, and can see no signs of the Debian environment whatsoever.

Connect using SSH to localhost

Connect using local connection

X11 Window

If you want to run X11 on your device, apt-get tightvncserver and get the free android app android-vnc-viewer from Google Play.

First apt-get some desktop environment. You can use gnome-desktop-environment if you got the hardware for it, but for smaller systems I'd recommend lxde instead. Both are included in the Debian ARM distribution.

On your Android device in the Debian system via SSH

Running Debian GNU/Linux with Gnome on Android using the android-vnc-viewer app

ASUS Transformer TF101 running Debian GNU/Linux with Gnome desktop environment (click to enlarge)

For better ergonomics, run ssh (optionally with X11 forwarding) from your favourite computer. Running the mouse pointer with the index finger over the touch screen, can be somewhat challenging. :-) Still, Gnome on the ASUS Transformer TF101 runs surprisingly well.

File access

One of the reasons motivating me to implement this is the ability to access the Android files without have to unmount/remount the SDcard.

In Gnome (Nautilus) at my workstation

As a user in the `sdcard_rw` group you have full access to the SDcard and as the root user all the files in the filesystem. This also makes backups easy. My devices are backed up nightly via BackupPC running tar over ssh.

The Android Media Scanner normally runs automatically each time Android remounts the SDcard. Since you are now transferring the media to the SDcard using ssh/sftp, not remounting the card, the media scanner won't run except for at boot. Download an app to start the Media Scanner manually from Google Play if you need - there are lots of them.

Search Google Play for Media Scanners

The Media Scanner is an index service used to catalogue media files such as MP3's and images for Android apps. If you transfer a media file using ssh/sftp but can't find it in your app, initiate a Media Scan.

Modifying the system

With the real GNU/Linux distribution on top, it's trivial to customize the device to your likings. For example the Sony Xperia Active only got 420MB internal storage for apps and data. This we'll change simply by moving `/data` from the internal partition to a new partition we create on the SDcard. Create the partition, copy `/data` to it, edit Android's `init.rc` (or `init.vendor.rc`) to mount the new one instead, and then restart. All done, no need to re-flash the device or anything.

Before

After

Total space from 420MB to 4.66GB - not bad at all. Just keep in mind Android's vold does not like more than four partitions on the SDcard by default.

Rooted

Please note that this way, the Android environment is not "rooted". This is trivial to achieve, but very much less needed, unless you have some app needing root privileges you still need to use. Myself I find giving away root privileges to apps far too dangerous.

To enter the Android Matrix from the Debian world, use chroot.

This is seldom needed, since you'll perform all the work (e.g. edits) of the Android file system directly from the Debian environment, using the full set of tools GNU/Linux provides you.

Happy hacking!

Caveats

Environment variables and file descriptors

When ssh:ing into the device, remember that neither the ssh server nor your login shell is a child of Android's init. Therefore you've got no access to neither file descriptors nor environment variables created by init, especially not ANDROID_PROPERTY_WORKSPACE with corresponding file descriptor. Because of this, you can't use getprop/setprop or any command relying Android properties from the ssh session (e.g. restart adbd). To do this, you must enter the Android world via a child of init, e.g. adbd or a local ssh-server in the Android root (e.g. dropbear).

You can always enter the Android world only to adb shell back to itself.

Note that this device is not rooted (ro.secure is 1), hence me ending up as the shell user.

Running apt-get upgrade, many installations scripts restarts their corresponding daemons. Since no daemons except for them you start in rc.local are supposed to be running in the Debian environment, it might be a good idea to restart the system after the upgrade.

Some warnings

Although Debian is the root, both systems are heavily dependent on the Android system and its init, since it is the "owner" of the hardware (i.e. runs init). If Android init fails, you will not be able to ssh into the Debian machine. Even if the root is transferred to the SDcard, the Android init mounts internal partitions, /system most important. If you do some change on this partition, you might lock yourself out. This can be solved by a backup copy of the Android environment so you can restore it to the SDcard and edit /android/init.*.rc to not mount /system from internal flash but use the one you restored to the SDcard instead. Running /system from the SDcard to begin with may be a good idea if you plan to change it frequently. This way the original system partition can be left untouched. This of course goes for all the Android partitions. Use can easily increase the size for your apps by changing /data to a partition on the SDcard of whatever size you like.

Or to conclude, always keep a backup.

Of course you can implement some fail-safe in the init scripts, populating /dev, mounting /proc, /sys, setting IP

address etc if the Android init fails, but I find it more practical to run /system from SDcard instead.

The Android Java machine (Dalvik) on the other hand is quite non-critical at the operating system level, so removing bundled apps (bloatware) on the /data partition is quite harmless. If you happen to remove e.g. the Home Application, you'll still be able to ssh into the Debian system restoring it from your backup.

Common mistakes

Make sure you set execute permissions for busybox, init scripts etc.

Make sure all mount points are there.

Make sure /dev is a mount point and not populated with device nodes.

About Performance

Note that this is not emulation or virtualization but a runtime environment. Because of this no performance penalty whatsoever occur in neither the Android nor the Debian environment, not counting the extra one and a half second to boot the device.

If moving partitions (eg. /system and/or /data) to the SDcard for safety or to increase the size, the speed of the SDcard may affect the performance. My benchmarks shows that a class 10 card gives about the same I/O performance as the internal nand disk, though. Don't expect more than 15-20 MB/s. USB disks may give you more.

Don't expect laptop performance, though. If it's primarily a GNU/Linux workstation you want, get an x86 based machine instead. The Android platform is design with resource conservation in mind, not high performance.

pstree

A typical ps tree showing the process hierarchy. Note the sshd running this pstree and a sftp server in the Debian root. All the other processes are chroot:ed to the Android root.

Realize there's no connection between the process tree and the chroot file system structure. Here the init process lives in the chroot environment despite of being top of the process tree, and the ssh server sshd in the genuine top root, despite of being below init in the process tree.

Devices verified

Implemented successfully on the following Android devices.

- LG Optimus P700

- Sony Ericsson Xperia Active ST17
- ASUS Transformer TF101
- HTC Desire
- Samsung Galaxy S II (SGH-T989) (requires custom mkbootimg, see link)
- Hisense E860
- Sony Xperia NEO

Got it up on some other device? Send me a mail and I'll add it to this list. Feedback? Questions? Suggestions?
This tutorial too basic? Too complicated? Please feel free to send me a mail!

/By Mikael Q Kuisma

Предметный указатель

программы

memtest86+, 188

syslinux, 188

термины

), 170

бэкенд компилятора, 170

фронтенд компилятора, 170

грамматика, 163

канадский крест, 250

кросс-компиляция, 187

кросс-тулчейн, 187

лексема, 162

лексер, 162

лексический анализ, 163

лексический анализатор, 164

падстек, 68

пакет, 196

промежуточное представление, 170

регулярное выражение, 162

сборка, 187

сборка пакета, 196

сегмент, 47, 300

секция ELF, 47, 300

семантический анализ, 171

синтаксический анализ, 166

сканер, 162

токен, 162

учебный план, 3

вендорная сборка, 256

Data Definition Language, 161

DDL, 161

ELF, 300

emLinux, 187

multiboot, 302

plain text, 161

USB

хост-контроллер, 182

On-The-Go, 182

OTG, 182
URB, 182
VCS, 40

azLinux

настройка
приложение, 207

пакет
dirs, 202

переменная
APP, 207
BOOT, 203
BUILD, 203
DIRS, 203
GZ, 202
ROOT, 203
SRC, 203
TC, 203
TMP, 203

приложение
clock, 207
micro, 207

скрипт
mk.rc, 198

железо, 208
AR7240, 221
ASUS Eee PC 701, 211

BlackSwift, 219
Celeron1037U, 221
CeleronM, 220
Gigabyte GA-C1037UN-EU, 213
i386, 209
i486sx, 220
QEMU, 210, 214, 217
RT5350, 221
VoCore, 219