

Азбука ARMатурщика

ОСНОВЫ БЫТОВОЙ АВТОМАТИКИ, ЭЛЕКТРОНИКИ И СИСТЕМ УПРАВЛЕНИЯ

© Консорциум хоббитов России
HackSpace «Чебураторный завод»
Дмитрий Понятов <dponyatov@gmail.com>
группа ruOpenWrt
Bill Collis (II)
Andreas Fester (VI)
Joe Martin, Craig Libuse (16)

Оглавление

I	О книге	2
II	Введение в практическую электронику	5
	An Introduction to Practical Electronics, Microcontrollers and Software Design © Bill Collis	6
1	Введение в практическую электронику	7
2	Ваше обучение по специальности «Технология»	9
2.1	Цели обучения технологиям Ново-Зеландской программы	9
2.2	Ключевые компетенции Ново-Зеландской программы	10
3	Вводная электронная схема	13
3.1	Где купить комплектующие?	13
3.2	Макетная плата: breadboard	14
3.3	Простейшая схема	15
3.4	Ток и напряжение, сечение проводника, плотность тока	16
3.5	Проводники и изоляторы, сопротивление и проводимость	18

3.6	Диэлектрическая и тепловая прочность изоляции	20
3.7	Тепловое действие тока. Мощность	22
3.8	Масштабные множители	22
3.9	Использование мультиметра	23
3.10	Определение сопротивления резистора по цветовому коду	26
3.11	Светодиоды	27
3.12	Некоторые характеристики светодиодов	28
3.13	Задание на исследование светодиода	29
3.14	Добавление выключателя в схему	30
3.15	Задание на исследование выключателя	30
3.16	Важные понятия схемотехники	31
3.17	Изменение величины сопротивления	32
3.18	Добавление в схему транзистора	32
3.19	Понимание схем	34
3.20	Входная цепь — фоторезистор (LDR)	35
3.21	Рабочая схема датчика темноты	37
3.22	Защитные цепи — использование диода	37
3.23	Задача исследования диода	38
3.24	Финальная схема датчика темноты	39
4	Вводное конструирование печатной платы	41
4.1	Учебник по редакторам схем и печатных плат	41
4.2	Введение в KiCAD	42
4.2.1	Менеджер проектов kicad	43
4.2.2	Создание схемы в модуле eeschema	45
4.2.3	Сохранение схемы	47
4.3	The Schematic Editor	47

4.3.1	The Toolbox	47
4.3.2	Using parts libraries	48
4.3.3	Using Components from within libraries	49
4.3.4	Different component packages	50
4.3.5	Moving parts	50
4.3.6	Wiring parts together	50
4.3.7	Zoom Controls	51
4.3.8	Junctions	51
4.3.9	ERC	51
4.4	The Board Editor	52

III Рабочая среда разработчика встраиваемых систем 53

4.5	Операционная система с набором типовых утилит	54
4.6	САПР электронных устройств (EDA CAD)	54
4.7	Пакет расчета и симуляции электронных схем: SPICE	54
4.8	САПР общего назначения	54
4.9	Система управления версиями: VCS	54
4.10	Текстовый редактор или интегрированная среда разработки (IDE)	55
4.11	ПО для программатора, JTAG-адаптера	55
4.12	Симулятор для отладки программ без железа	55
4.13	Система верстки документации	56

IV Пакет компиляторов и утилит GNU toolchain 57

5	Make	59
---	-------------	----

6	binutils	60
6.1	Формат объектного файла GNU ELF	60
6.2	Ассемблер GNU as	61
6.3	Линкер GNU ld	61
6.3.1	Скрипты линкера	61
6.4	objdump	61
7	GNU Compiler Collection (GCC)	62
7.1	Компилятор GNU C	62
7.2	Компилятор GNU C++	62
7.3	Компилятор GNU Fortran	62
V	САПР электронных устройств KiCAD	63
7.4	Установка под ☐Windows	65
7.5	Установка под Linux	65
7.6	Маршрут проектирования	67
7.7	Создание проекта в менеджере проектов kicad	68
7.8	Создание принципиальной схемы в eeschema (часть 1)	70
7.9	eeschema : редактор электрических схем	73
VI	Расчет схем и моделирование в ngSPICE	74
8	Доступные SPICE-пакеты	76
8.1	Установка ngSPICE под ☐Windows	77
8.2	Установка LT-SPICE (только ☐Windows)	77

8.3	Установка ngSPICE под Linux	78
9	Пошаговый пример использования	79
9.1	Рисуем схему в KiCAD	79
9.2	Создание списка цепей	81
9.3	Запуск симуляции	83
9.4	Просмотр результата расчета	85
9.5	Расчет АЧХ по переменному току (АС симуляция)	89
9.6	Симуляция полноволнового выпрямителя	93
10	Настройка KiCAD для SPICE-моделирования	96
10.1	Библиотеки компонентов со SPICE-элементами	96
10.2	Настройка проекта	97
10.3	Как это работает	97
VII	Разработка конструкции в САПР FreeCAD	99
11	Установка	102
11.1	Windows	102
11.2	Linux	103
VIII	Инструменты и электронное оборудование	104
12	Радиомонтажный инструмент	105
12.1	Pro'sKit	105
12.2	Инструмент до 1000 В	110

12.3	Хранение	111
12.4	Радиомонтаж	112
12.5	Прочие	116
13	Паяльное оборудование	117
13.1	Паяльник	117
13.2	Паяльная станция	119
14	Измерительное оборудование	122
14.1	Мультиметр	122
14.1.1	Mastech M838	123
14.1.2	Mastech M300	124
14.1.3	Mastech M320	125
14.2	Осциллограф	126
14.3	Логический анализатор	126
14.4	Генератор сигналов	126
14.5	Рыльцемер RLC	126
15	Электроинструмент	127
15.1	Дрель	128
15.2	Лобзик	131
15.3	Жвигатель	132
IX	Станочное оборудование	135
16	Настольные станки	138

17	Самодельная оснастка	139
-----------	-----------------------------	------------

18	Промышленные станки	140
-----------	----------------------------	------------

18.1	1A616: станок токарно-винторезный	141
18.1.1	Назначение и области применения	142
18.1.2	Распаковка и транспортировка	142
18.1.3	Фундамент станка, монтаж и установка	142
18.1.4	Подготовка станка к первоначальному пуску	142
18.1.5	Паспортные данные	142
18.1.6	Описание основных узлов	142
18.1.7	Смазка	142
18.1.8	Первоначальный пуск	142
18.1.9	Указания по технике безопасности	142
18.1.10	Настройка	142
18.1.11	Регулирование	142
18.1.12	Ведомость комплектации	142

X	Разработка ПО для встраиваемых систем	143
----------	--	------------

19	IDE	144
-----------	------------	------------

19.1	⊕ECLIPSE	147
19.1.1	Установка ⊕ECLIPSE под ☐Windows	147
19.1.2	Установка ⊕ECLIPSE под Linux	147
19.1.3	Установка CDT	148
19.1.4	Установка PyDev	149
19.1.5	Установка TeXlipse	149

19.1.6	Редактирование файлов в формате XML и производных	150
19.1.7	Проверка орфографии	150
19.2	Code::Blocks	153
19.3	(g)Vim	153
19.3.1	Установка под Windows	153
19.3.2	Выход из (g)Vim	155
19.3.3	Выход с автосохранением	156
19.3.4	Переход в режим редактирования	156
19.3.5	Переход в режим команд	156
19.3.6	Запись редактируемого файла	156
19.3.7	Перезагрузка файла	157
19.3.8	Отмена последних изменений (undo)	157
20	Make: управление сборкой проектов	158
21	VCS: системы контроля версий	159
21.1	CVS	159
21.2	Subversion	159
21.3	Git	159
21.3.1	GitHub	159
22	Вспомогательные скрипты на языке Python	160
23	Основы Си и C₊⁺	161
23.1	Установка MinGW (win32)	161
23.2	Особенности C ₊ ⁺ в embedded	161
23.3	Сборка кросс-компилятора GNU toolchain	161

24	Лексический и синтаксический анализ	162
24.1	Лексер и лексический анализ, утилита flex	163
24.2	Генератор синтаксических анализаторов bison	167
25	LLVM и разработка собственных компиляторов	170
25.1	Установка под Windows	170
25.2	Создание компилятора с помощью инфраструктуры LLVM	170
25.3	Инструменты llic и llic	171
25.4	Семантический анализ	172
25.5	Оптимизация	172
25.6	Кодогенерация	172
25.7	Транслятор Паскаля	173

XI Микроконтроллеры Cortex-Mx **174**

26	Производители	175
26.1	ST Microelectronix STM32	175
26.2	LPC	175
26.3	Миландр	175
27	Отладочные платы	176
27.1	LeafLabs Maple Mini: STM32F103 /Cortex-M3/	176
27.2	Серия STM32 STM32DISCOVERY	176
27.2.1	STM32DISCOVERY: STM32F103 /Cortex-M3/	176
27.2.2	STM32F4DISCOVERY: STM32F406 /Cortex-M4F/	176
27.2.3	STM32F0DISCOVERY: STM32F040 /Cortex-M0/	176

XII ПЛИС	177
-----------------	------------

XIII USB	179
-----------------	------------

28 Стек протоколов USB	182
-------------------------------	------------

29 libUSB	184
------------------	------------

29.1 драйвер для устройства USB	184
---	-----

30 Поддержка USB в Linux	185
---------------------------------	------------

30.1 Оptions ядра	186
-----------------------------	-----

30.1.1 режимы host/client/otg и хост-контроллеры xHCI	186
---	-----

30.1.2 data storage: носители данных	186
--	-----

30.1.3 hid: клавиатура, мышь, джойстик	186
--	-----

30.1.4 USB-периферия: сеть, звук,...	186
--	-----

30.2 Настройка hotplug и автомонтирования USB носителей	186
---	-----

30.3 Сборка и настройка libusb	186
--	-----

30.4 Примеры программ низкоуровневого ввода/вывода	186
--	-----

XIV Встраиваемый emLinux	187
---------------------------------	------------

31 Загрузчик syslinux	189
------------------------------	------------

31.1 Закачка	189
------------------------	-----

31.2 Установка под Windows на флешку	190
--	-----

31.3 syslinux.cfg	191
-----------------------------	-----

32 azLinux	194
32.1 Требования к системе сборки (BUILD -хост)	196
32.2 Понятие пакет	197
32.3 Клонирование проекта azLinux	199
32.4 Общий порядок сборки	199
32.5 Фиксация переменных	201
32.6 dirs: Создание дерева каталогов	201
32.7 gz: Загрузка архивов исходников	205
32.8 APP: Приложение	206
32.9 HW: Поддерживаемое железо	207
32.10i386	209
32.10.1 qemu386: эмулятор QEMU	209
32.10.2 eeeepc701: ASUS Eee PC 701	209
32.10.3 gac1037: Gigabyte GA-C1037UN-EU rev.2	210
32.11ARM	211
32.11.1 qemuARM: эмулятор QEMU	211
32.11.2 cubie1: Cubie Board v.1	212
32.11.3 rpi: Raspberry Pi model B	212
32.12MIPS	212
32.12.1 qemuMIPS: эмулятор QEMU	212
32.12.2 mr3020: роутер MR3020	212
32.12.3 vocore: VoCore	212
32.12.4 bswift: BlackSwift	212
32.13CPU: Конфигурации процессоров	213
32.13.1 i386	213
32.13.2 ARM	214
32.13.3 MIPS	214

32.14	Пакеты	214
32.14.1	versions.mk : Версии пакетов	214
32.14.2	tc : кросс-компилятор	214
32.14.3	core : ядро	215
32.14.4	boot : загрузчики	215
32.14.5	libs : библиотеки	215
32.14.6	tc : сборка кросс-компилятора	216
32.14.7	binutils : ассемблер, линкер и утилиты	219
32.14.8	cclibs : библиотеки для сборки gcc	220
32.14.9	gcc0 : сборка минимального кросс-компилятора Си	220
32.14.10	gcc : пересборка полного кросс-компилятора Си/C ⁺	221
32.14.11	core : сборка основной системы	221
32.14.12	kernel : ядро Linux	221
32.14.13	libc : библиотека uClibc	227
32.14.14	gcc : пересборка полного gcc	233
32.14.15	busybox : набор утилит busybox	233
32.14.16	libs : сборка библиотек #{LIBS}	235
32.14.17	apps : сборка прикладных пакетов #{APPS}	235
32.14.18	user : сборка пользовательского кода	235
32.14.19	root : формирование корневой файловой системы	235
32.14.20	boot : сборка загрузчика syslinux/grub/uboot	237
32.14.21	syslinux	238
32.14.22	grub	238
32.14.23	uboot	238
32.14.24	emu : запуск собранной системы в эмуляторе	238
32.15	netboot : Сетевая загрузка	240
32.16	Прошивка на устройство	240

32.17RT-патч	240
32.18SDK: расширения для on-board разработки	240
32.18.1 canadian : сборка binutils канадским крестом	241
32.18.2 binhost : binutils для хост-процессора	242
32.18.3 binavr8 : binutils для Atmel ATmega (AVR8)	243
32.18.4 bincmx : binutils для ARM Cortex-Mx	243
32.18.5 fpc : FreePascal	243
32.18.6 python : интерпретатор Python	244
32.18.7 gcl : GNU Common Lisp	244
32.18.8 maxima : система символьной математики Maxima	244

33 Библиотека libSDL 245

33.1 Инициализация и завершение SDL-программы	246
33.2 LazyFoo tutorial	246
33.2.1 Setting up SDL and Getting an Image on the Screen	246
33.2.2 Optimized Surface Loading and Blitting	252
33.2.3 Extension Libraries and Loading Other Image Formats	260
33.2.4 Event Driven Programming	261
33.2.5 Color Keying	270
33.2.6 Clip Blitting and Sprite Sheets	270
33.2.7 True Type Fonts	270
33.2.8 Key Presses	270
33.2.9 Mouse Events	270
33.2.10 Key States	270
33.2.11 Playing Sounds	270
33.2.12 Timing	270
33.2.13 Advanced Timers	270

33.2.14	Regulating Frame Rate	270
33.2.15	Calculating Frame Rate	270
33.2.16	Motion	270
33.2.17	Collision Detection	270
33.2.18	Per-pixel Collision Detection	270
33.2.19	Circular Collision Detection	270
33.2.20	Animation	270
33.2.21	Scrolling	270
33.2.22	Scrolling Backgrounds	270
33.2.23	Getting String Input	270
33.2.24	Game Saves	270
33.2.25	Joysticks	270
33.2.26	Resizable Windows and Window Events	270
33.2.27	Alpha Blending	270
33.2.28	Particle Engines	270
33.2.29	Tiling	270
33.2.30	Bitmap Fonts	270
33.2.31	Pixel Manipulation and Surface Flipping	270
33.2.32	Frame Independent Movement	270
33.2.33	Multithreading	270
33.2.34	Semaphores	270
33.2.35	Mutexes and Conditions	270
33.2.36	Using OpenGL with SDL	270
33.3	Примеры программ	270
33.3.1	Графический Hello World	271
33.3.2	Вывод случайных прямоугольников	272
33.4	Версия SDL2	273

34 BuildRoot	274
35 Особенности OpenWrt	275
XV x86os: простая операционная система для компьютера на i386 процессоре	276
36 Ресурсы	277
37 Структура	279
38 Процесс запуска	280
39 Сборка кросс-компилятора	281
40 Формат ELF32	287
41 multiboot	293
42 Микроядро	296
43 Драйвера	298
43.1 vga : текстовая консоль VGA 80×25	298
43.2 kbd : клавиатура	300
43.3 ide : жесткий диск IDE	300
43.3.1 fatfs : файловая система FAT16	300

XVI	Символьная и численная математика	301
44	Общие сведения о компьютерной математике	303
45	Пакет Maxima	306
45.1	Установка Maxima под Windows	306
45.2	Калькулятор	307
XVII	Подготовка технической документации	308
46	Верстка в L ^A T _E X	309
47	Оформление листингов	310
48	Подготовка иллюстраций	311
48.1	GIMP	311
48.2	Inkscape	311
48.3	Graphviz	311
49	Замечания для соавторов “Абзуки ARMатурщика”	312
XVIII	Примерные учебные планы	313
50	Блондинко	314
51	Школотрон	315

52 Студень	316
53 Технический специалист	317
XIX Куча	318
54 Автоматное программирование /фреймворк QuantumLeaps/	320
55 Запуск Linux на android устройстве в режиме паразита	321
Предметный указатель	334

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 сырая копипада, 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 XVIII 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» (II), и конечно он вполне заслуженно включен в основные соавторы этой книги.

¹ вряд 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** [?] (16)
- 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/Azbuka>

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.

Часть II

Введение в практическую электронику

Эта часть основана на книге:

An Introduction to Practical Electronics, Microcontrollers and Software Design

Second Edition, 01 May-2014

© Bill Collis

www.techideas.co.nz

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

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

From: Bill Collis <Bill.Collis@.....nz>

Date: 2014-11-24 0:53 GMT+04:00

Subject: Electronis Book

To: "dponyatov@gmail.com" <dponyatov@gmail.com>

Hi Dmitry

thanks for your email.

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

Thanks

Bill

Глава 1

Введение в практическую электронику

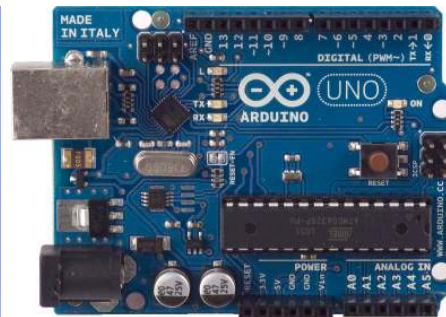
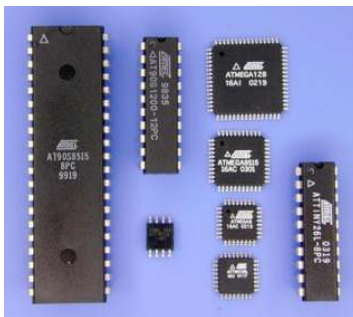
Эта книга ©¹ имеет следующий ряд основных направлений:

- Распознавание электронных компонентов и их правильное использование
- Нарботка цельного набора компетенций по основам электроники
- Использование макетных плат
- Навыки ручной пайки
- Использование закона Ома для выбора токоограничивающих резисторов
- Делитель напряжения
- Использование EDA CAD² для разработки и подготовки производства печатных плат

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

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

- Программирование микроконтроллеров и их сопряжение с внешними устройствами
- Использование транзистора в режиме ключа
- Теория источников питания
- Принципы и схемы электропривода
- Навыки отладки схем, их тестирования и испытаний
- Следование принципам обучения через практику
- Безопасные приемы работы



Глава 2

Ваше обучение по специальности «Технология»

2.1 Цели обучения технологиям Ново-Зеландской программы

- Технологическая практика

- **Чоткость:** разработка ясных описаний для ваших технологических проектов.
- **Планирование:** думать прежде чем делать, и использовать во время работы документацию: блок-схемы, принципиальные схемы, чертежи разводки плат, диаграммы и эскизы.
- **Наработка навыков:** сборка, отладка и тестирование электронных схем, проектирование и изготовление печатных плат, написание программ для микроконтроллеров.

- Технологические знания

- **Моделирование:** прежде чем строить готовое электронное устройство, сначала важно понять как оно работает путем моделирования и/или макетирования аппаратного и программного обеспечения.
- **Технологические продукты:** знания о компонентах и их характеристиках.
- **Технологические системы:** электронное устройство является более, чем набором компонентов, это функционирующая система с входами, выходами и контролирующим процессом.

• Природа технологии

- **Значение технологических достижений:** знания об электронных компонентах, особенно микроконтроллерах, как основе современных технологий.
- **Роль технологии в обществе:** электронные устройства в настоящее время играют центральную роль в инфраструктуре нашего современного общества; подчинили ли они нас себе, как они изменили нашу жизнь?

2.2 Ключевые компетенции Ново-Зеландской программы

- **Знания:** для меня предметом технологии является все что относится к знанию. Моя цель: заставить студентов понимать технологии, заложенные в электронные устройства. Для достижения этого понимания студенты должны активно учиться¹ в работе на самом раннем этапе, чтобы они могли построить собственное понимание предмета и пойти дальше, чтобы стать хорошими решателями проблем. В начале обучения электронике это требует от студентов восприимчивости к инструкциям, которые им дают, и поиск ясности, когда они не понимают их.

Для этого на занятиях рассматриваются много новых и различных элементов знаний, и студентам выдаются задания на решение проблем, чтобы помочь им мыслить логически. Копирование чужого ответа

¹ в оригинале **enage**, англо-калька с [себуанского](#), NZ

наказывается, но приветствуется совместная работа. В основе обучения лежит построение правильных концептуальных моделей и анализ в контексте "большой картины".

- **Взаимодействие:** работа в парах и группах, это важно как в классе, так и в любой другой ситуации в жизни; мы все должны договариваться и разделять ресурсы и оборудование с другими людьми; поэтому крайне важно активное общение и помощь друг другу.
- **Использование языка символов и текстов:** сердцем нашего предмета является язык, который мы используем для обмена информацией в электронных схемах, планах, алгоритмах и синтаксисе компьютерных языков программирования; так что способность распознавать и правильно использовать символы и диаграммы для работы, которую мы делаем, имеет критическое значение.
- **Самоконтроль:** студенты принимают на себя личную ответственность за собственное обучение; они принимают вызов, надеясь найти ответы в книгах или найти учителя, способного объяснить им, что делать. Это значит, что студенты должны взаимодействовать с рабочим материалом.

Иногда ответы приходят легко, иногда нет; часто наша тема требует много проб и ошибок (в основном ошибок). Студенты должны знать, что у них будут трудные времена, пока не будет изучена большая часть. И не сдаваться в поиске понимания.
- **Участие и содействие:** мы живем в мире, который невероятно зависит от технологии, особенно электроники; студенты должны развивать осознание важности этой области человеческого творчества в нашей повседневной жизни, и понимать, что наши проекты имеют и социальную функцию, а не только техническую.



Глава 3

Вводная электронная схема

3.1 Где купить комплектующие?

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

Для России можно отметить сеть магазинов “Болтмастер”¹.

Для модулей пока что доступна поставка из Китая по почте: AliExpress с оплатой с виртуальной карты VISA платежной системы QIWI. Доставка занимает от 2х недель до 2х месяцев. Удобно покупать модули в виде микросхем, уже запаянных с обвязкой на кусок текстолита (**breakout board**): Arduino Mini, Maple Mini, датчики, контроллеры двигателей. Также интересны наборы (магазины) элементов: пачки резисторов, конденсаторов и т.п. в выводном исполнении, по несколько десятков номиналов.

¹ Самара

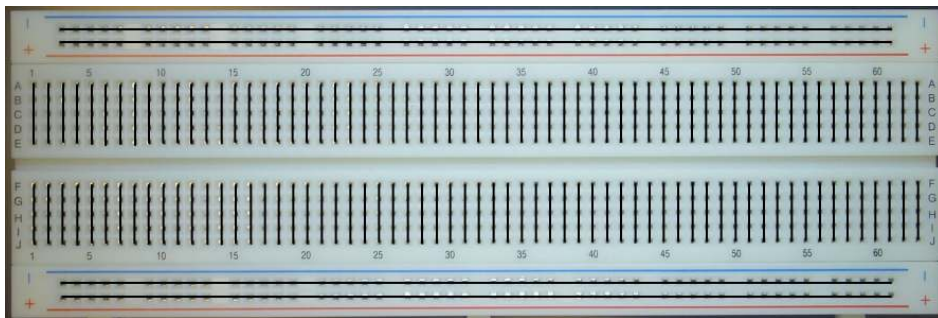
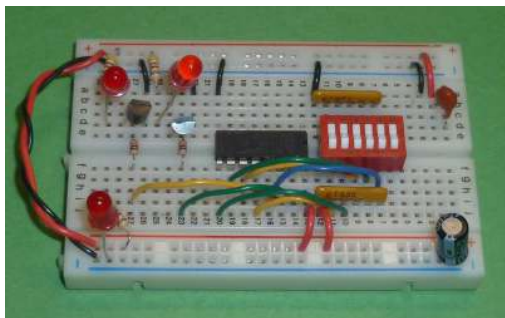
3.2 Макетная плата: breadboard



<https://www.youtube.com/watch?v=vQdUSE1auz8>

<https://www.youtube.com/watch?v=k9jcHB9tWko>

https://www.youtube.com/watch?v=2wvn8_23phE



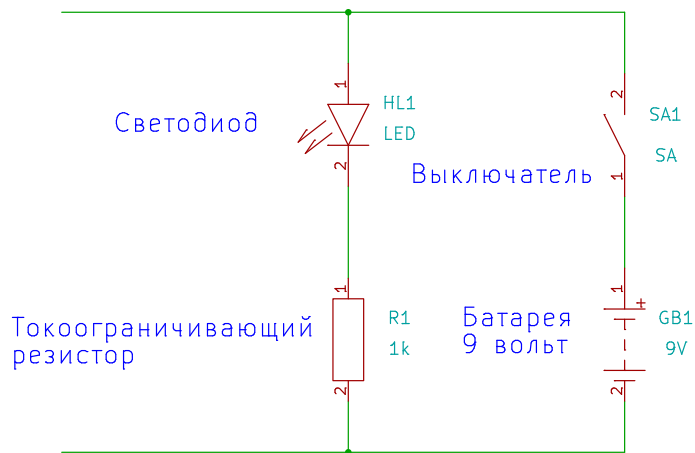
Breadboard ([б]еспаяечная [м]акетная [п]лата , БМП, “вафля”) — пластмассовый блок с отверстиями и металлическими полосковыми зажимами, создающими соединения между **элементами схемы**. Отверстия расположены так, что **компоненты** и отрезки провода могут быть соединены вместе формируя схему, без использования паяльника. Верхние и нижние ряды, как правило, используются для **шин питания**, красный сверху для плюса, и внизу черный/синий для минуса (**общий провод**, или “земля”). На длинных вафлях шины питания поделены на отдельные сегменты, и требуют соединения короткими перемычками.

3.3 Простейшая схема

Эта схема может быть собрана вот так 3.3, обратите внимание, что **светодиод** должен находиться в правильном положении. Если у вас есть светодиод и **резистор**, соединенные в **замкнутый контур**, светодиод должен загореться.

Принципиальная схема

Компоновка



Светодиоду требуется для включения около 2 В^2 , батарея на 9 В , так что с напряжением все нормально. Но если вы подключите светодиод напрямую к батарее, он сгорит! Для светодиодов главный рабочий параметр — допустимый рабочий ток, обычно он не превышает $10..20\text{ mA}^3$. Так что 1 k^4 резистор используется для ограничения тока через светодиод.

² [В]ольт

³ $1\text{ [м]иллиАмпер} = 0.001\text{ [А]мпер}$

⁴ $1\text{ [К]илоОм} = 1000\text{ Ом}$

Недопустимо подавать на светодиоды напряжение обратной полярности. Светодиоды имеют невысокое (несколько вольт) обратное пробивное напряжение. В схемах, где возможно появление обратного напряжения, светодиод должен быть защищён параллельно включенным обычным диодом в противоположной полярности.

Если вы отключите любой провод в схеме, она перестает работать, схема должна быть завершённой, чтобы электроны могли течь по проводникам из **источника питания**.

3.4 Ток и напряжение, сечение проводника, плотность тока



<https://www.youtube.com/watch?v=Dq4fSp6wz-o>

1. https://www.youtube.com/watch?v=3ZeveDL1_bg Электрический ток Источники тока
2. <https://www.youtube.com/watch?v=mt86jYUjPFI> Ток в металлах Действия электрического тока
3. <https://www.youtube.com/watch?v=42CEi94hGgA> Электрический ток. Сила тока
4. <https://www.youtube.com/watch?v=SNP05e9wZWU> Амперметр Измерение силы тока
5. <https://www.youtube.com/watch?v=Yt91aIAwp68> Электрическое напряжение

В предыдущем разделе были использованы два важных понятия электроники — **ток** и **напряжение**. Необходимо объяснить эти понятия, не залезая глубоко в физику. Проще всего объяснить какое-то явление на примере другого явления, с которым человек регулярно сталкивается в обыденной жизни, и ощущает его собственными органами чувств. Для электрических явлений лучше всего подходит **гидравлическая модель**⁵. В ГМ электрические явления заменяются условными водопроводными: **проводник** — труба, **электрический ток** — поток воды в этой трубе, **резистор** — сужение сечения трубы, препятствующее течению тока, **диод** — клапан, открывающийся только в одну сторону, и т.п. В результате маловразумительные абстрактные понятия

⁵ пример применения гидравлической модели, осторожно, г***осайт

физики электрических явлений превращаются в почти ощутимые потоки и давление воды: все мы ежедневно пользуемся водопроводом, даже в глухих регионах найдется бочка или хотя бы дырявое ведро.

Таким образом, любому школьнику можно объяснить эти понятия:

Электрическое **напряжение** — давление “электрической воды” в проводнике-трубе.
Измеряется в [В]ольтах.

Как и обычное давление, **напряжение — разностная величина, так как измеряется между двумя точками** электрической цепи. Для обычного давления за опорную величину принимают атмосферное давление⁶. Для электричества напряжение измеряют тоже между двумя точками, или между точкой и **общим проводом**, принимаемым за 0. В физике напряжение определяют также как **разность потенциалов** между двумя точками.

Если взять проводник, и разделить его условной плоскостью, перпендикулярной проводнику, в области пересечения этой плоскости и проводника получается область — **сечение проводника**.

Сечение проводника легко увидеть и реально — достаточно взять толстый одножильный сплошной провод, и аккуратно отпилить (не откусить) его точно поперек. Поверхность такого отпила и будет сечением.

Электрический **ток** — количество “электрической воды”, проходящей через сечение проводника-трубы в единицу времени. **Сила тока** измеряется в [А]мперах.

$$j = \frac{I}{S} \quad (3.1)$$

⁶ не ощущаемое человеком, так как оно уравновешено таким же внутренним давлением крови

где j плотность тока, А/м²
 I сила тока, Ампер
 S площадь сечения проводника, м²

Чем выше плотность тока, тем больше нагревается проводник (подробнее см ??), поэтому она учитывается при выборе провода, и ширины проводников печатных плат. В этом случае часто используется численно другая величина — А/мм².

3.5 Проводники и изоляторы, сопротивление и проводимость



1. <https://www.youtube.com/watch?v=q4I1hZ5YQ2w> Электрическое сопротивление проводника.
2. <https://www.youtube.com/watch?v=Slx0EEaQ4Cg> Удельное сопротивление
<https://www.youtube.com/watch?v=FCkR-3YE5Ac>

Используя гидравлическую модель, точно так же просто можно объяснить понятие **проводимости**. Любой материал можно представить как кусок вещества, имеющий пористую структуру, например как поролон или песок. Через этот пористый материал течет “электрическая вода”, т.е. ток. Это течение вызвано действием напряжения, **приложенного** к концам проводника (источником питания). Напряжение проталкивает электричество через материал, поэтому чем больше напряжение, тем выше сила проходящего тока. Каждый материал имеет свою “пористость”, чем она больше, тем больше **проводимость**:

$$I = G U \quad (3.2)$$

где I сила тока, А
 G проводимость, См (сиemens)
 U напряжение, В

Чаще вместо проводимости используют обратную величину — **сопротивление**:

$$R = \frac{1}{G} = G^{-1} \quad (3.3)$$

где G проводимость, См (сиemens)
 R сопротивление, Ом

Соответственно, формула 3.2 превращается в описанный в любом школьном учебнике физики **закон Ома для постоянного тока**:

$$I = \frac{R}{U} \quad (3.4)$$

В зависимости от проводимости или сопротивления материалы делят на несколько видов:

	G , проводимость	R , сопротивление
сверхпроводник	∞	0
проводник	большая	низкое
изолятор, диэлектрик	низкая	высокое

В зависимости от внешних условий: температуры, давления, агрегатного состояния вещества⁷, примесей — одно и то же вещество может быть как проводником, так и изолятором.

Типичные проводники: металлы, ионизированный газ (плазма), растворы солей в воде (электролиты), влажное дерево

Типичные диэлектрики: пластики, стекло, керамика, сухое дерево, вакуум, газы в т.ч. воздух.

В электронике очень часто нужно иметь в определенном месте схемы заданное сопротивление, для этого выпускаются специальные элементы с точно калиброванным сопротивлением между выводами: резисторы, иногда применяют устаревшее название сопротивление (часто проволочное). Их делают из керамики, на которую наматывают проволоку, или напыляют тонкую пленку из специальных металлических сплавов с высоким сопротивлением⁸. Иногда используется только кусочек такого сплава без керамики. Для качественной пайки выводы резисторов делают из хорошо проводящего металла, хорошо смачиваемого расплавленным припоем.

3.6 Диэлектрическая и тепловая прочность изоляции

Для диэлектриков существует специальный параметр, характеризующий их способность работать как изолятор — диэлектрическая прочность.

Чем выше диэлектрическая прочность, тем большее напряжение способен выдерживать изолятор без разрушения.

⁷ плазма, газ, жидкость, твердое тело

⁸ часто в состав входят никель, хром, вольфрам

При повышении температуры диэлектрическая прочность уменьшается

Именно поэтому



Запрещается использовать любые удлинители в свернутом состоянии, или использовать электрический провод, свернутый бухтой (катушкой)



Не допускается нагрев любых частей электронной схемы или элементов конструкции выше $40..50^{\circ}\text{C}$

Для любого диэлектрика существует некоторое значение напряжения, при котором он начинает пропускать ток. При прохождении тока материал нагревается (??). В результате нагрева проходящим током или от соседних соприкасающихся поверхностей у диэлектрика увеличивается проводимость, что приводит к еще большему проходящему через него току, и дополнительному нагреву.

Если тепло не отводится во внешнюю среду (например в середине катушки провода смотанного удлинителя), температура продолжает повышаться. В некоторый момент температура достигает значения, при котором слой изоляции размягчается, утрачивает механическую прочность, и соседние проводники сближаются или соприкасаются с плохим контактом (происходит **короткое замыкание**).

В месте плохого контакта или большого тока температура резко повышается до значений, когда диэлектрик воспламеняется или резко меняет химический состав, распадаясь **с выделением ядовитых соединений**⁹ и полностью утрачивая изолирующие свойства как в электрическом, так и в механическом смысле.

⁹ для типичной изоляции из ПВХ — сложные токсичные соединения с содержанием хлора

Самый опасный вариант, и типичный случай **выгорания проводки** — **ток короткого замыкания** слишком маленький для срабатывания защитных устройств (**предохранитель, автомат, устройство защитного отключения, УЗО**), но достаточный для нагрева изоляции до температуры химического распада или воспламенения. Аварийного выключения не происходит, а проводка продолжает гореть до победного конца.

3.7 Тепловое действие тока. Мощность



1. <https://www.youtube.com/watch?v=2LWAI0HRI8s> Нагревание проводников электрическим током
Закон Джоуля Ленца
2. <https://www.youtube.com/watch?v=Mbt40TgBRuw> **Мощность** электрического тока

3.8 Масштабные множители



<https://www.youtube.com/watch?v=i3ABWmCl1EI> Перевод единиц измерения

Практически для всех единиц измерения необходимо использование **масштабных множителей**, когда величины численно или слишком большие (много цифр до запятой), или слишком маленькие (много цифр после запятой):

пико	p	10^{-12}
напо	n	10^{-9}
микро	μ	10^{-6}
милли	m	10^{-3}
кило	k, K	10^3
мега	M	10^6
гига	G	10^9

$$10 \text{ mA} = 10 \times 10^{-3} \text{ A} = 10 \times 0.001 \text{ A} = 0.010 \text{ A}$$

$$5 \text{ КОм} = 5 \times 10^3 \text{ Ом} = 5 \times 1000 \text{ Ом} = 5000 \text{ Ом}$$

3.9 Использование мультиметра



<https://www.youtube.com/watch?v=BEgvm4o-u2Q>

<https://www.youtube.com/watch?v=UMwYLwsPgCY>

1. <https://www.youtube.com/watch?v=GBuGSj1uPGk>

2. <https://www.youtube.com/watch?v=VJ3RBS42IVY>

<https://www.youtube.com/watch?v=q3R4s6WE1cI>

Возьмите мультиметр (14.1) и измерьте напряжение на резисторе, близко ли оно к 7 В ? Также измерьте ток через диод, не превышает ли он допустимый ?

Напряжение измеряется в [В]ольтах подключением измерительного прибора (мультиметра) параллельно элементу, при этом нужно

- режим измерения включить на режим измерения постоянного напряжения $V- / DCV$ (или переменного, обозначается как $V \sim / ACV$) а
- диапазон измерения выставить на максимальное значение напряжения
- Последовательно уменьшая диапазон измерения, найдите диапазон, в котором мультиметр показывает наибольшее количество знаков после запятой.

Если диапазон слишком большой, прибор покажет значение в районе 0, а если слишком низкий — выведет [1]. Обычно работают в диапазоне, соответствующем максимальному напряжению питания (в нашем случае $20 V^{10}$), иногда для слабых напряжений переключаясь на одну..две ступени ниже. Но — возможны случаи¹¹ когда напряжение в части схемы на порядки (в десятки..тысячи раз) превосходит напряжение питания.

Ток измеряется в [А]мперах подключением измерительного прибора (мультиметра) последовательно с элементом в разрыв цепи, при этом нужно

- режим измерения включить на режим измерения постоянного тока $A- / DCA$ (или переменного, обозначается как $A \sim / ACA$) а
- диапазон измерения выставить на максимальное значение тока
- Последовательно уменьшая диапазон измерения, найдите диапазон, в котором мультиметр показывает наибольшее количество знаков после запятой.

¹⁰ [V]olt = [B]ольт

¹¹ в любых схемах содержащих индуктивности: катушки провода, трансформаторы, электродвигатели, динамики и т.п.

Обратите внимание, что напряжение измеряется **вольтметром** на полностью собранной схеме, а для измерения тока нужно изменять схему, включая в нее **амперметр**. Если измерения тока нужно проводить на готовом устройстве, иногда ставят 2хконтактный **джампер** (как на материнских платах компьютеров): при измерении амперметр подключают к его контактам, а потом джампер замыкают специальной съемной проводящей перемычкой. Этим прием вы можете использовать в своих устройствах для регулярного измерения **тока потребления**, и расчета **потребляемой мощности**:

$$W_{\text{потребляемая}}[\text{Ватт}] = U_{\text{питания}}[\text{Вольт}] \times I_{\text{устройства/элемента}}[\text{Ампер}] \quad (3.5)$$

Переключив мультиметр в **режим прозвонки (тестера)**, можно проверить

- наличие электрического соединения между двумя точками **обязательно отключенной от источника питания** схемы,
- исправность диода или светодиода,
- исправность конденсатора большой **емкости** и
- любые другие случаи когда нужно определить что две точки электрически соединены между собой, постоянно или временно.

Если между **щупами** мультиметра в режиме прозвонки¹² **сопротивление** протекающему току не превышает 1 КилоОм¹³, раздается звуковой сигнал.

¹² = тестера

¹³ см. паспорт на прибор

Если щупы подключить к исправному **разряженному** конденсатору большой емкости (“электролиту”), раздается короткий щелчок или даже “пик”.

Тестером также можно определить тип и **цоколёвку**¹⁴ транзистора; как это сделать описано в ??.

Диоды проверяются двумя подключениями **в разной полярности** — при **красном щупе** на **аноде (+)** диода (в **прямой полярности включения**) диод проводит ток, а в **обратной полярности** нет (красный щуп на **катоде (-)**).

Проверьте **светодиод**: отключите его из схемы, и проверьте как диод. Если светодиод исправен, в **прямой полярности** светодиод будет **очень слабо светиться**.

3.10 Определение сопротивления резистора по цветовому коду

Когда берете резистор, проверьте его **номинал** (значение). В наших схемах каждый резистор имеет свою цель, и значение выбирается в зависимости от того, хотим ли мы больший или меньший ток в этой части цепи. Чем выше **номинал** резистора, тем меньше ток. Чем ниже номинал резистора, тем выше ток. Для **выводных** резисторов, которые мы будем использовать на макетках, маркировка наносится на корпус в виде набора цветных полосок:

Цветовая маркировка на 5 цветных полосок:

цифра	цифра	цифра	множитель	точность
-------	-------	-------	-----------	----------

Цифра: 0 1 2 3 4 5 6 7 8 9

■	■	■	■	■	■	■	■	■	□
---	---	---	---	---	---	---	---	---	---

Множитель: ■ $10^0 = 1$ ■ $10^1 = 10$ ■ $10^2 = 100$ ■ $10^3 = 1000$
■ $10^4 = 10000$ ■ $10^5 = 100000$ ■ $10^6 = 1000000$
■ $10^{-1} = 0.1$ ■ $10^{-2} = 0.01$

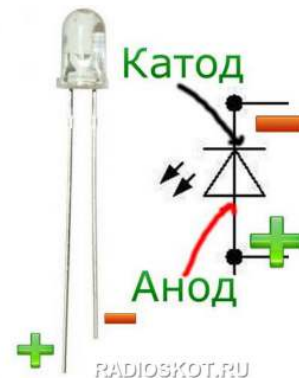
¹⁴ “раскопытку”

Точность: ■ ±1% ■ ±2% ■ ±5% ■ ±10%

-[■ ■ ■ ■ ■]-	100 × 10 ⁴ ± 1%	1M	1 миллион Ом	1M Ω	1 000 000 Ом
1 0 0 4 1					
-[■ ■ ■ ■ ■]-	100 × 10 ² ± 2%	10k	10 тысяч ом	10,000 Ом	10k Ω
1 0 0 2 2					
-[■ ■ ■ ■ ■]-	100 × 10 ¹ ± 5%	1k	1 тысяча ом	1000 Ом	1k Ω
1 0 0 1 5					
-[■ □ ■ ■ ■]-	390 × 10 ⁰ ± 1%	390R	390 Ом	390Ω	
3 9 0 0 1					
-[■ ■ ■ ■ ■]-	100 × 10 ⁰ ± 1%	100R	1000 Ом	100Ω	
1 0 0 0 1					
-[■ ■ ■ ■ ■]-	470 × 10 ⁻¹ ± 1%	47R	47 Ом	47Ω	
4 7 0 -1 1					

3.11 Светодиоды

В настоящее время светоизлучающие диоды используются в индикаторах и дисплеях на различном оборудовании, однако они все больше и больше используются в качестве замены для галогенных ламп и ламп накаливания¹⁵ во многих других приложениях. Они включают в себя ходовые огни на транспорте, светофоры, большие уличные TV-экраны.





По сравнению с **лампами накаливания**, светодиоды почти не дают тепла, и поэтому очень эффективны. Они также имеют гораздо большее время жизни, например 10 лет, по сравнению с 10 месяцами для ламп накаливания. Таким образом, в некоторых ситуациях например сигналов светофора, если там установлены светодиоды, они дают значительную экономию как по потребляемой мощности, так и по стоимости технического обслуживания. Хотя есть небольшая проблема со светодиодными светофорами — они не плавят снег, который на них налипает!

3.12 Некоторые характеристики светодиодов

Интенсивность : измеряется в мКд (милликанделах)

Угол излучения : угол от оси, до линии, где интенсивность падает до 50%

Прямое напряжение U_F : напряжение необходимое для получения полной яркости светодиода

Прямой ток: рабочий ток, дающий максимальную яркость

Пиковая длина волны : самая яркая спектральная линия излучаемого света

3.13 Задание на исследование светодиода

Пользуясь сайтом одного из поставщиков, найдите информацию и характеристики/атрибуты для двух светодиодов: обычный красный 5 мм и 5 мм высокой интенсивности.

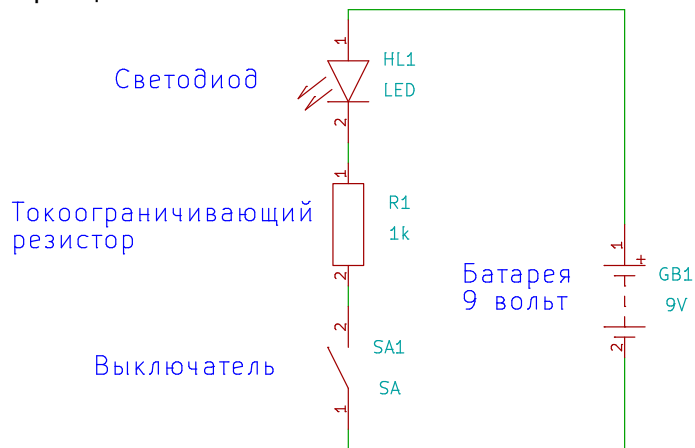
Светодиод	Красный 5 мм	Высокая интенсивность 5 мм
Поставщик		
Номер детали		
Стоимость (\$)		
Яркость (мКд)		
Прямое напряжение (U_F)		
Длина волны (нм)		
Прямой ток (I_F)		

3.14 Добавление выключателя в схему

Выключатель — способ для ручного управления схемой пользователем

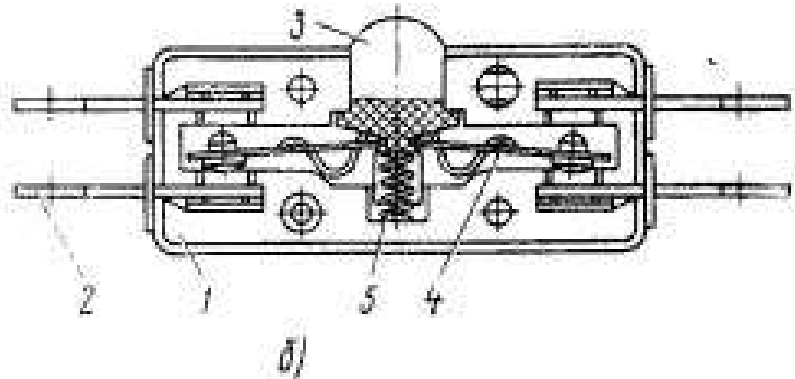
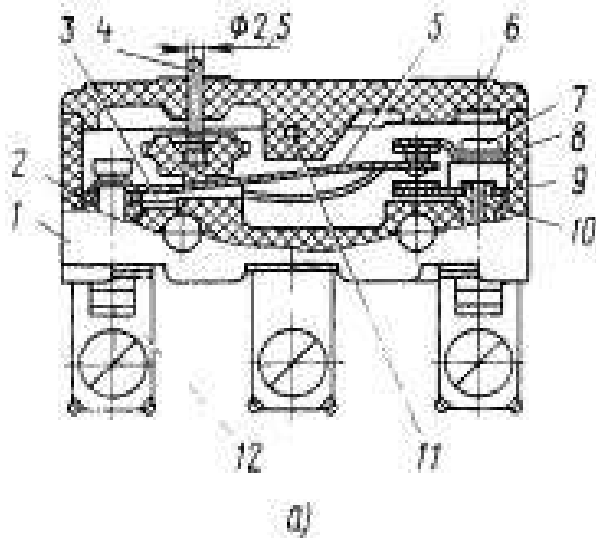
Принципиальная схема

Компоновка



3.15 Задание на исследование выключателя

Найдите маленький переключатель, аккуратно разберите его, нарисуйте конструкцию, и объясните как он работает. Убедитесь, что вы поняли цель пружин(ы). Вот упрощенные рисунки небольшого переключателя, когда он находится в разных позициях. Когда переключатель включен, электричество может течь, когда он разомкнут, цепь разорвана.



3.16 Важные понятия схемотехники

Кратко повторим:

Схема состоит из нескольких компонентов и источника питания, соединенных проводами.

Поток электронов (часто называют **носители заряда**) течет в цепи; однако если нет **полной (замкнутой) цепи**, электроны не могут течь.

Напряжение (U) является мерой энергии в цепи, оно используется в качестве меры энергии, подаваемой от батареи или энергии (напряжения) через часть схемы.

Ток (I) представляет собой поток электронов от батареи по контуру и обратно к батарее. Ток измеряется в амперах (обычно мы будем использовать миллиампер или мА). Обратите внимание, что **электрический ток**

это не ток электронов или ток зарядов. Так же, как течение реки не эквивалентно потоку воды.

Сопротивление работает на уменьшение тока, резисторы в схеме оказывают сопротивление току.

Проводники, такие как провода, соединяющие компоненты вместе, не имеют (теоретически) никакого сопротивления току.

Действительно важное понятие, для ясного понимания:

Напряжение прикладывается параллельно компонентам, а ток течет через компоненты.

3.17 Изменение величины сопротивления

Каков эффект изменения значения сопротивления в нашей цепи?

Резистор контролирует ток, чем выше значение резистора, тем меньше ток. Как будет выглядеть резистор на 10K?

390R	большой ток	яркое свечение светодиода
1k	слабый ток	тусклое свечение
1M	нет тока	нет свечения

3.18 Добавление в схему транзистора

Транзистор — электронный элемент, выполняющий функцию усиления: управление сильным сигналом под контролем слабого сигнала.

Один из видов — Полевой транзистор (FET, [F]ield [E]ffect [T]ransistor). Управление выполняется слабым входным напряжением, которое управляет выходным током, изменяя проводимость (сопротивление) части транзистора в выходной цепи.

Главная особенность полевого транзистора — во входной цепи, по которой подается управляющее напряжение, течет почти нулевой **входной ток**, т.е. **полевой транзистор имеет бесконечное входное сопротивление**. Эта особенность важна при подключении источников слабого сигнала (датчиков), генерирующих слабое сигнальное напряжение, но не способных выдать скольнибудь заметный ток — электро-динамический микрофон, индуктивный звукосниматель электрогитары и т.п. Сигнал от таких датчиков усиливается **предварительным усилителем** на полевом транзисторе, а затем усиленный сигнал подается в остальную часть схемы для дальнейшей обработки.

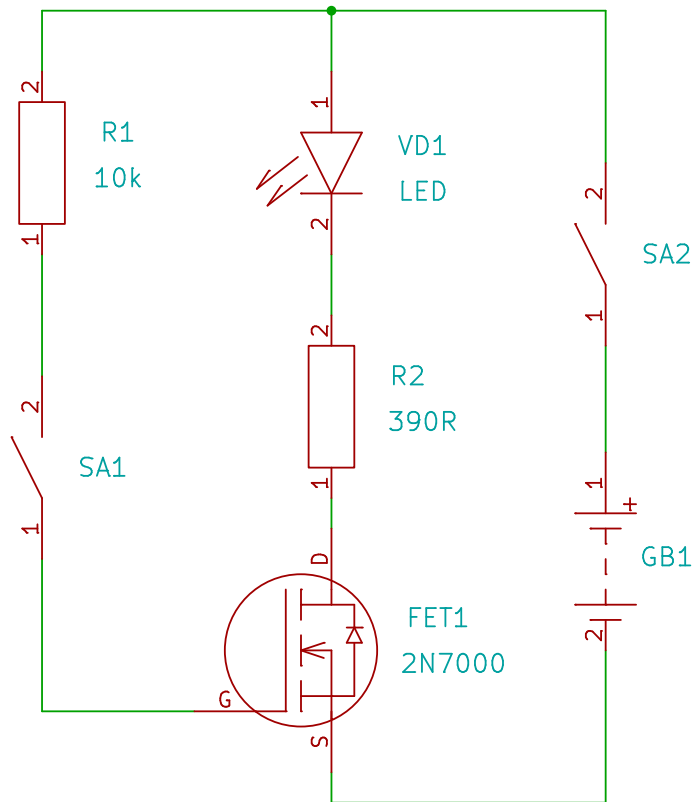
У FET-транзистора три ноги:

S Source исток

G Gate затвор

D Drain сток

Слабый сигнал на затворе способен управлять бóльшим током, текущим с истока на сток. Ток стока тот же самый, который течет через светодиод. Резистор 390 Ом ограничивает этот ток на приемлемом для работы светодиода уровне.



3.19 Понимание схем

Электроника — все что относится к управлению в физическом мире. Физические объекты имеют свойства, такие как температура, сила, движение, и звуковые/радио/световые колебания, связанные с ними.

Электронные устройства имеют **входные цепи**, чтобы преобразовать физический мир (свет, звук и т.д.) в различные уровни напряжения.

Они включают в себя **преобразующие схемы**, которые преобразуют, манипулируют и изменяют информацию (информация закодирована в виде различных напряжений).

Они имеют **выходные цепи** для преобразования различных уровней сигнала обратно в физический мир, где мы можем почувствовать результат процесса (свет, звук и т.д.)

Рассмотрим пример, например телевизор: радиосигнал из физического мира на входе преобразуется электронной схемой в уровень напряжения, и преобразуется в свет, который мы видим, и звук, который можно слышать.

3.20 Входная цепь — фоторезистор (LDR)

LDR¹⁶ или фоторезистор — типичный компонент, используемый в цепях измерения уровня освещенности. LDR изменяет сопротивление пропорционально интенсивности света, падающего на него. Фоторезисторы изготавливаются из полупроводников, таких как селен, оксид таллия и сульфид кадмия. Когда фотоны света сталкиваются с атомами в материале LDR, проводимость увеличивается, и электроны могут течь через цепь. Это означает, что по мере увеличения уровня освещенности, сопротивление уменьшается.

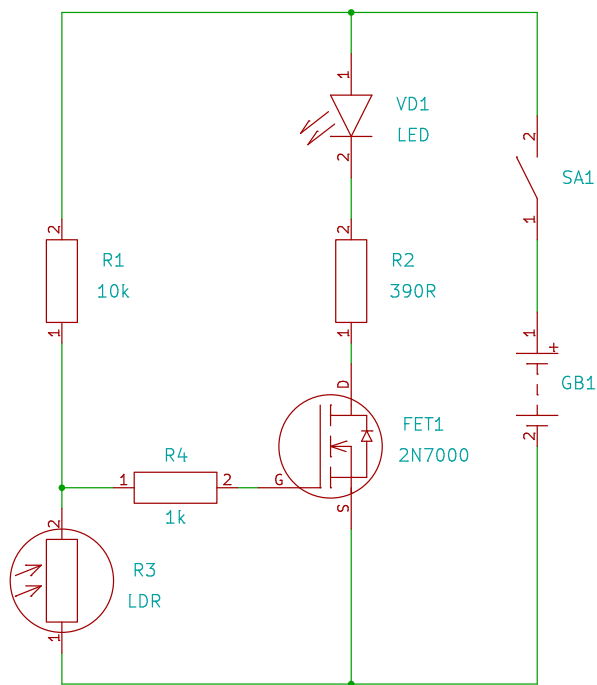
Фоторезисторы можно использовать только с маленьким протекающим через него током порядка 5 мА, если превышает допустимый ток, LDR перегревается и сгорает. Обычно они используются в схеме делителя напряжения последовательно с обычным резистором.



Найдите фоторезистор и измерьте его сопротивление:

сопротивление LDR при полном дневном свете _____

сопротивление LDR в темноте _____



Компоненты на схеме: резистор R1 от 10K (10 000) Ом до 1M (1 000 000) Ом, R3 фоторезистор, GB1 батарея. R1, R3 — схема делителя напряжения (питания), резистор R2 ограничивает ток через светодиод VD1 и цепь исток/сток полевого транзистора.

В темноте LDR имеет высокое сопротивление, и напряжение на выходе высокое.

На свету LDR имеет низкое сопротивление, и напряжение падает.

3.21 Рабочая схема датчика темноты

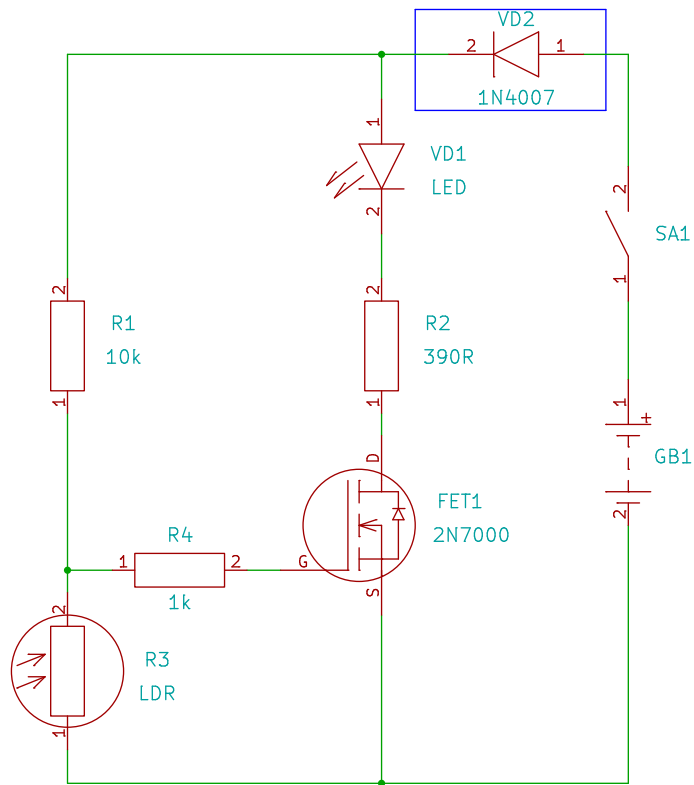
Экспериментируя с резистором последовательно с LDR, можно регулировать чувствительность схемы на различных уровнях освещенности.

3.22 Защитные цепи — использование диода

Диоды очень часто используемые компоненты, они бывают всех форм и размеров.

Диод — электронный прибор, пропускающий ток только в одном направлении





Основная характеристика диода — то что ток течет только в одном направлении, так что вы не можете его пустить в схеме в обратном направлении, и можете быть уверены, что это работает. В этой модифицированной схеме питание подается от батареи. Фраза "цепь защищена диодом" означает, что если батарея ошибочно подключена в неправильной полярности, тока не будет, потому что диод блокирует его. Это обычно используется в мастерской для защиты наших схем от подачи питания обратной полярности.

Конечно, диод несовершенен, и если напряжение источника питания превысит **максимальное обратное напряжение диода**, то диод будет сломан: обратный ток диода будет очень быстро расти, и сожжет диод. 1N4004 допускает обратное напряжение 400 В. Диоды также могут проводить определенный **ток в прямом направлении**, иначе они перегреваются и сгорают. 1N4004 имеет максимальный прямой ток 1 А.

3.23 Задача исследования диода

Исследуйте спецификации для этих двух распространенных диодов (мы их часто используем в классе) и узнайте, что значит каждый параметр спецификации.

	Описание	1N4007	1N4148
Пиковое обратное напряжение			
Максимальный прямой ток			

3.24 Финальная схема датчика темноты

Функция **входной** части схемы — определение уровня освещенности.

Функция **преобразующей** части схемы (транзистор) — усиление небольшого изменения напряжения из-за легких изменений освещенности.

Функция **выходной** части цепи — индикация для пользователя.

Функция **источника питания** — надежно обеспечить энергию для работы схемы.

Когда темно, светодиод включается, когда свет присутствует, светодиод выключен. Эта схема может использоваться как игрушка для ребенка, чтобы ориентироваться в темноте, и найти дверь в темной комнате.

Диод, светодиод и транзистор поляризованные элементы, т.е. имеют положительные и отрицательные выводы и, следовательно, требуют правильного включения в схему, или они не будут работать.

Вы можете определить полярность светодиода найдя спил на корпусе (отрицательный вывод, катод) или по длинной ноге (положительный вывод, или анод)

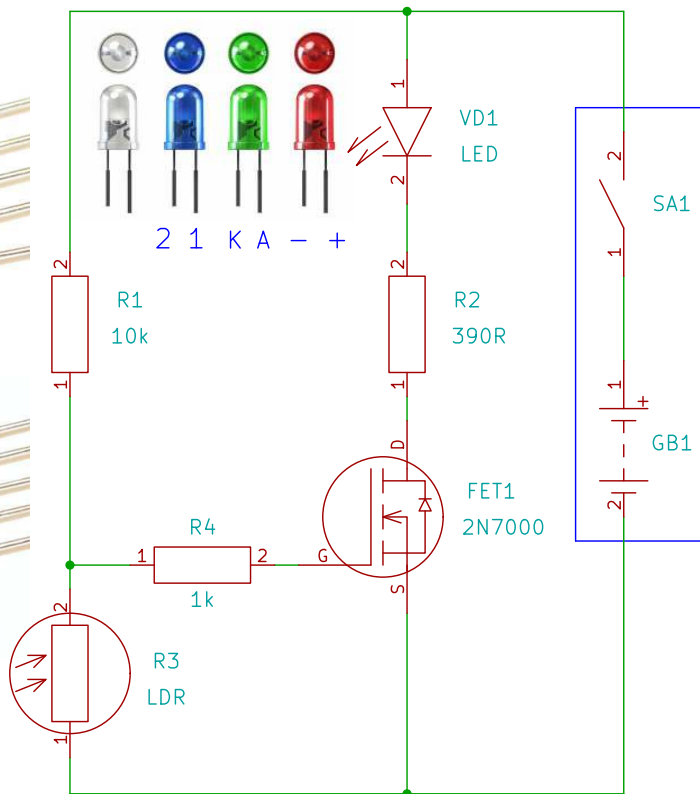
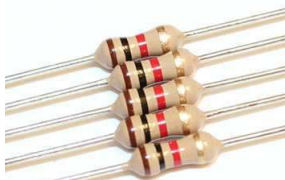
Полярность **транзистора** можно определить за счет формы корпуса и зная расположение трех выводов.

Нарисуйте линии от компонентов к их символам на схеме, чтобы помочь вам запомнить их. Помните, что резистор в выходной цепи был установлен с более низким значением (уменьшен от 1k до 390 Ом) чтобы в окончательной схеме сделать светодиод ярче.

10k



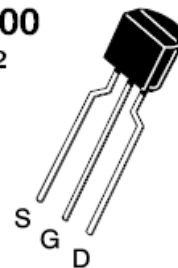
1k



батарейный блок
со встроенным
выключателем



2N7000
TO-92



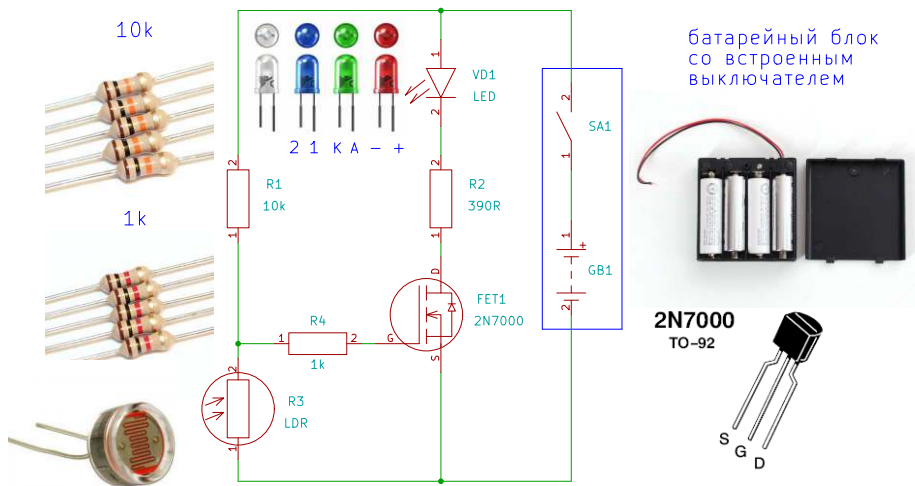
Глава 4

Вводное конструирование печатной платы

Мы берем короткий перерыв в теории электроники, чтобы представить вам новую тему в конструировании — изготовление печатных плат.

4.1 Учебник по редакторам схем и печатных плат

Схема типа детектора тьмы не очень подходит нам в виде сборки на "вафле" нам нужна надежная конечная конструкция, так что мы сделаем ее на печатной плате.



4.2 Введение в KiCAD

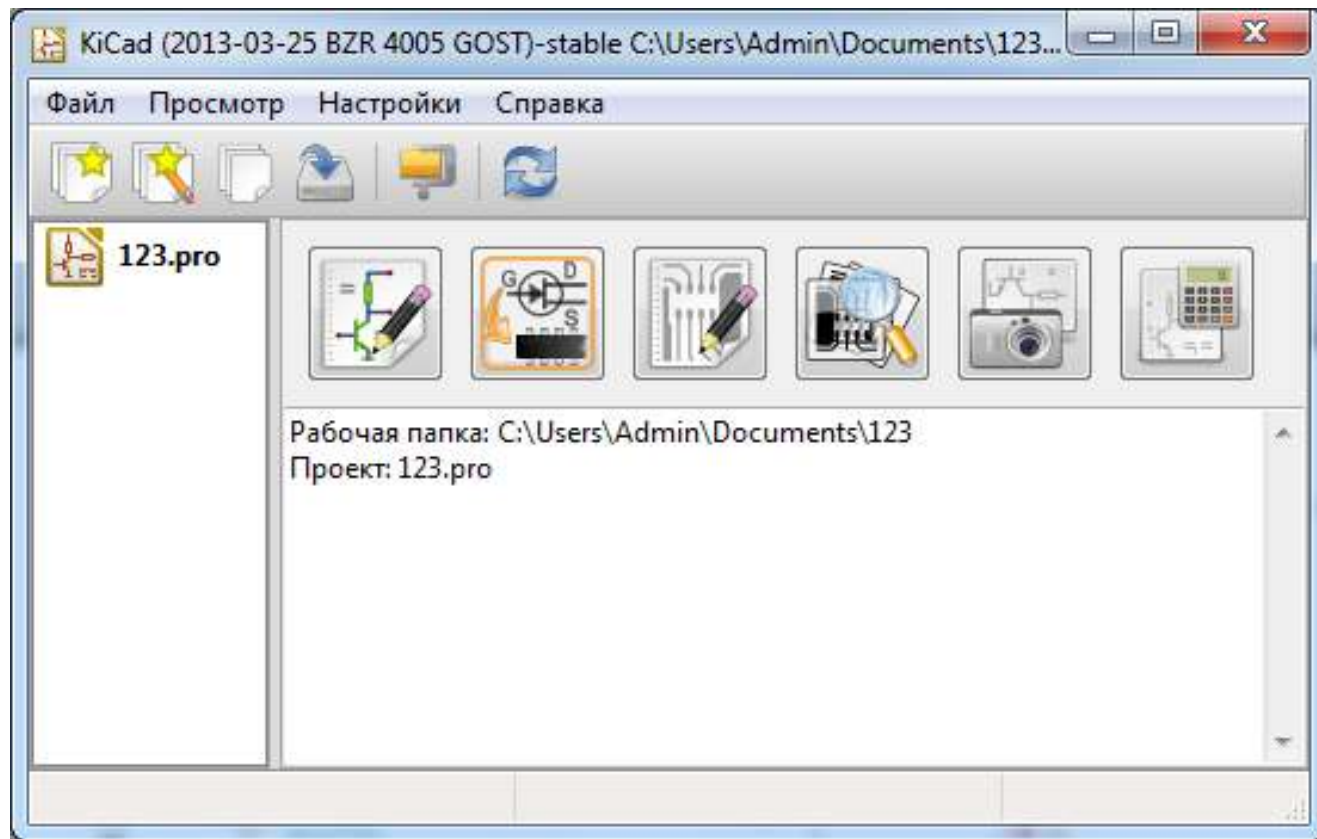
Подробно пакет **KiCAD** рассмотрен в разделе V. В этом разделе очень кратко продублирован минимум информации об этой программе¹, чтобы вы легко и быстро смогли начать разрабатывать печатные платы для своих простых поделок. В оригинальной книге [?] рассматривается пакет **Eagle**, но бесплатная лицензия на **Eagle** запрещает использовать ее для коммерческих проектов, и размер плат ограничен до 100 мм × 80 мм, поэтому была сделана замена САПР. EDA САПР позволяет пользователю рисовать схемы для электронных цепей, и затем на их основе **трассировать** печатные платы. Используя этот очень краткий пошаговый учебник, вы **разведёте** плату для схемы усилителя на микросхеме TDA2822.

¹ САПР электронных устройств, EDA CAD




4.2.1 Менеджер проектов **kiCad**

Windows:  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 **падстегов** (отверстий и площадок)

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

Лучше всего для каждого проекта использовать отдельные папки; в противном случае система может сбиться с толку, если файлы из разных проектов будут лежать в одной папке. Прделайте 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**

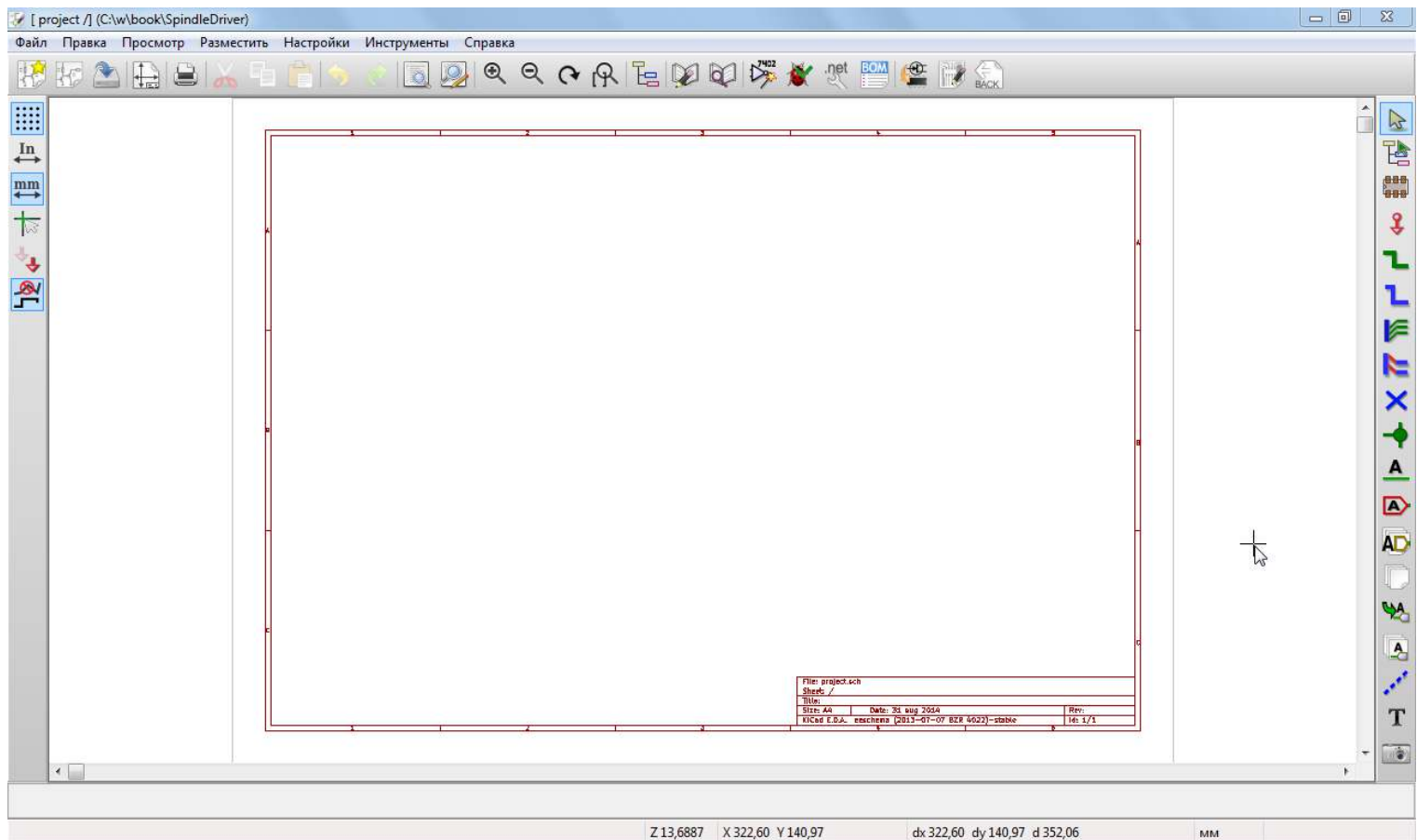
- В диалоге выберите созданную папку и нажмите .
3. Если папка проекта содержит какие-то файлы, будет выведено окно выбора: создать подпапку с именем проекта , или записать файл проекта в указанную папку как есть . Нажмите No.
 4. Сохраните проект кнопкой , или + .
 5. В папке появится файл **SpindleDriver.pro**, 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.

4.2.2 Создание схемы в модуле **eeschema**

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.

4.2.3 Сохранение схемы

- Не забывайте сохранять ваши данные в процессе работы
- САПР создает много временных файлов, так что вы должны держать ваши папки в аккуратности.
- Если в первый раз запустили **eeschema**, создайте папку для проекта.
- Создайте папку в соответствии с названием проекта, например, **DarkDetector**
- Сохраните схему как **DarkDetector.sch** в папке **DarkDetector**.

4.3 The Schematic Editor

The first part of the process in creating a PCB is drawing the schematic.

1. Parts will be added from libraries
2. and joined together using 'nets' to make the circuit

4.3.1 The Toolbox

As you point to the tools in the TOOLBOX their names will appear in a popup and also their description will appear in the status bar at the bottom of the window

Find the following tools:

- ADD A PART
- MOVE AN OBJECT

- DELETE AN OBJECT
- DEFINE THE NAME OF AN OBJECT
- DEFINE THE VALUE OF AN OBJECT
- DRAW NETS (connections)
- ERC (electrical rule check)

4.3.2 Using parts libraries

Selecting parts libraries to use.

Parts are stored within libraries and there are a large number of libraries in Eagle.

It is not hard to create your own library and modify the parts within it. The cls.lbr has many already modified components within it. If Eagle is not setup to use the cls library you will need to do it now.

1. From your internet browser save the file cls.lbr into your Eagle folder.
2. In Eagle's control panel from the menu select options then directories
3. In the new window that appears make sure the directories for the libraries are highlighted
4. Click on browse and find your Eagle.directory
5. Next highlight the directories for Projects
6. Click on browse and find your Eagle directory again.
7. Choose OK.

8. You might need to close EAGLE and restart it to make sure it reads the libraries ok.
9. To use a library right click on it from within the Control Panel
10. Make sure Use is highlighted. It will have a green dot next to it if it is selected
11. At this time right click on the other lbr folder and select Use none.

NOTE THE IMPORTANCE OF THE GREEN DOT NEXT TO THE LIBRARY,
if its not there you will not see the library in the schematic editor!

4.3.3 Using Components from within libraries

From your schematic Click the ADD button in the toolbox
A new window will open (it may take a while)

- Find the CLS library
- Open it by double clicking on it or by
- clicking the + sign
- Open the R-EU_ section (ResistorEuropean)
- Here you will find the 0204/10 resistor.
- Select it and then click OK

Add 2 more resistors of the same type.

Add all of the following parts

LIBRARY	PART	Qty
cls	REU-0204/10	3
cls	LDR	1
cls	2,54/0,8 (wirepads)	2
cls	led 5MM	1
cls	1N4148 D41-10	1
cls	2N7000	1
cls	GND	3

A wirepad allows us to connect wires to the PCB (such as wires to switches and batteries)

4.3.4 Different component packages

There are several different types of resistors; they all have the same symbol however resistors come in different physical packages so we must choose an appropriate one. The 0204/7 is suitable for us but any of the 4 smallest ones would be OK.

4.3.5 Moving parts

Move the parts around within the schematic editor so that they are arranged as per the schematic below. Keep the component identifiers (numbers like R1, R2, R3) in the same places as those below.

4.3.6 Wiring parts together

These form the electrical connections that make up the circuit. Select the net button from the toolbox.

Left click on the very end of a component and draw in a straight line either up, down, left or right.

Left click again to stop at a point and draw before drawing in another direction.

Double left click at another component to finish the wire.

4.3.7 Zoom Controls

There are a number of zoom controls that can be used to help you work in your circuit.

Find these on the toolbar and identify what each does.

Nets

Nets are the wire connections between the components, each has a unique name.

Find the info button in the toolbox and check the names and details of the components and nets/wires.

When you want to connect a new net to an existing net, Eagle will prompt you as to which name to give the combined net.

If one of the nets has a proper name i.e. VCC, V+,V-, ground... use that name, otherwise choose the net with the smallest number

4.3.8 Junctions

Junctions are the dots at joins in the circuit, they are there to make sure that the wires are electrically connected. Generally you will NOT need to add these to your circuit as the net tool puts them in place automatically

4.3.9 ERC

The ERC button causes Eagle to test the schematic for electrical errors.

Errors such as pins overlapping, and components unconnected are very common.

The ERC gives a position on the circuit as to where the error is; often zooming in on that point and moving components around will help identify the error.

You must correct all errors before going on.

4.4 The Board Editor

Часть III

Рабочая среда разработчика встраиваемых систем

4.5 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)

4.6 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

4.7 Пакет расчета и симуляции электронных схем: SPICE

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

4.8 САПР общего назначения

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

4.9 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**)

4.10 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.

4.11 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).

4.12 Симулятор для отладки программ без железа

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

4.13 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 IV

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 5

Make

Глава 6

binutils

6.1 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	машиинный 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

6.2 Ассемблер GNU **as**

6.3 Линкер GNU **ld**

6.3.1 Скрипты линкера

6.4 **objdump**

Глава 7

GNU Compiler Collection (GCC)

7.1 Компилятор GNU C

7.2 Компилятор GNU C++

7.3 Компилятор GNU Fortran

Часть V

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 фпр. IUT de Saint Martin d'Hères (Франция).

3

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

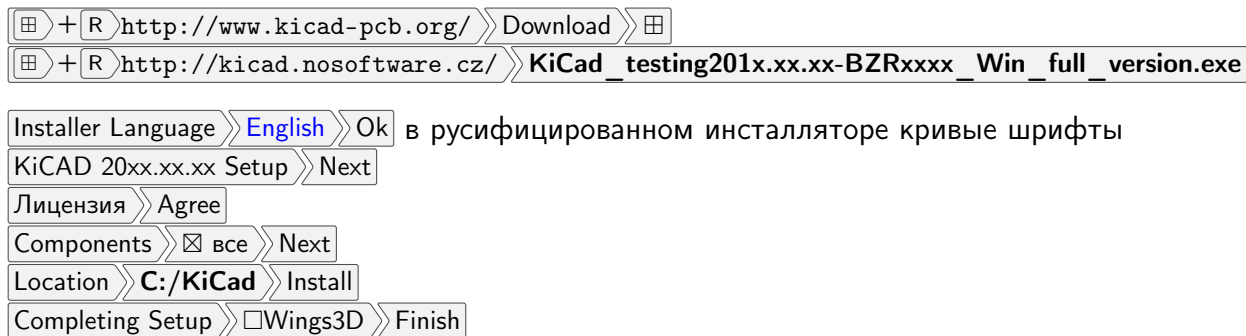
² копияста: <http://ru.wikibooks.org/wiki/KiCad>

³ копияста: <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.

7.4 Установка под Windows



7.5 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
```

kicad >> eeschema >> Настройки >> Библиотека
Пользовательские пути поиска >> Добавить >> /home/user/Azbuka/kicad/lib
Файлы библиотек >> Добавить >> R,L,C,SPICE,DA_POWER,.. >> Открыть >> OK

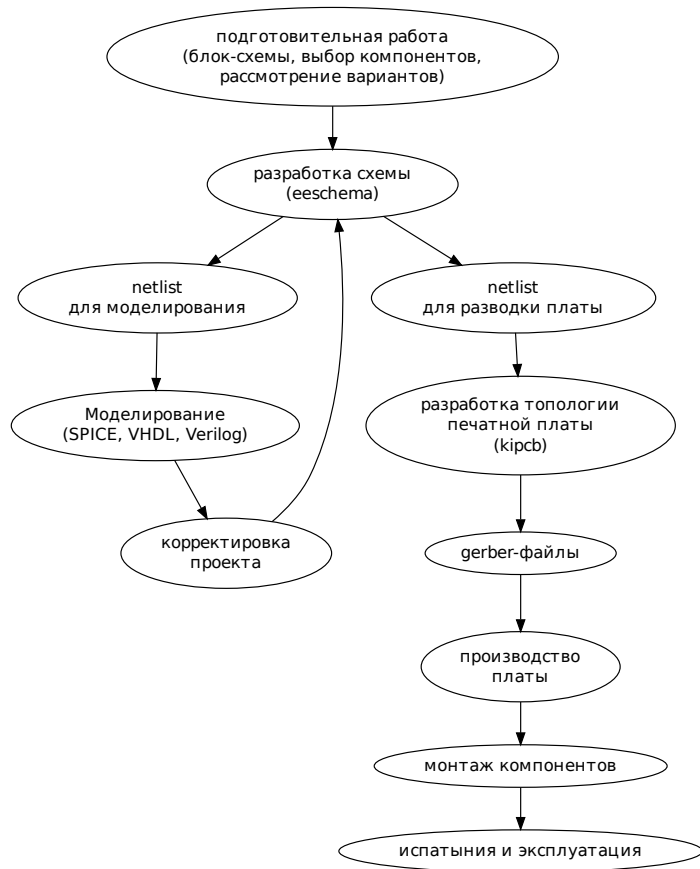
Для проверки работы библиотек можете открыть проект

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


или сразу схему

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

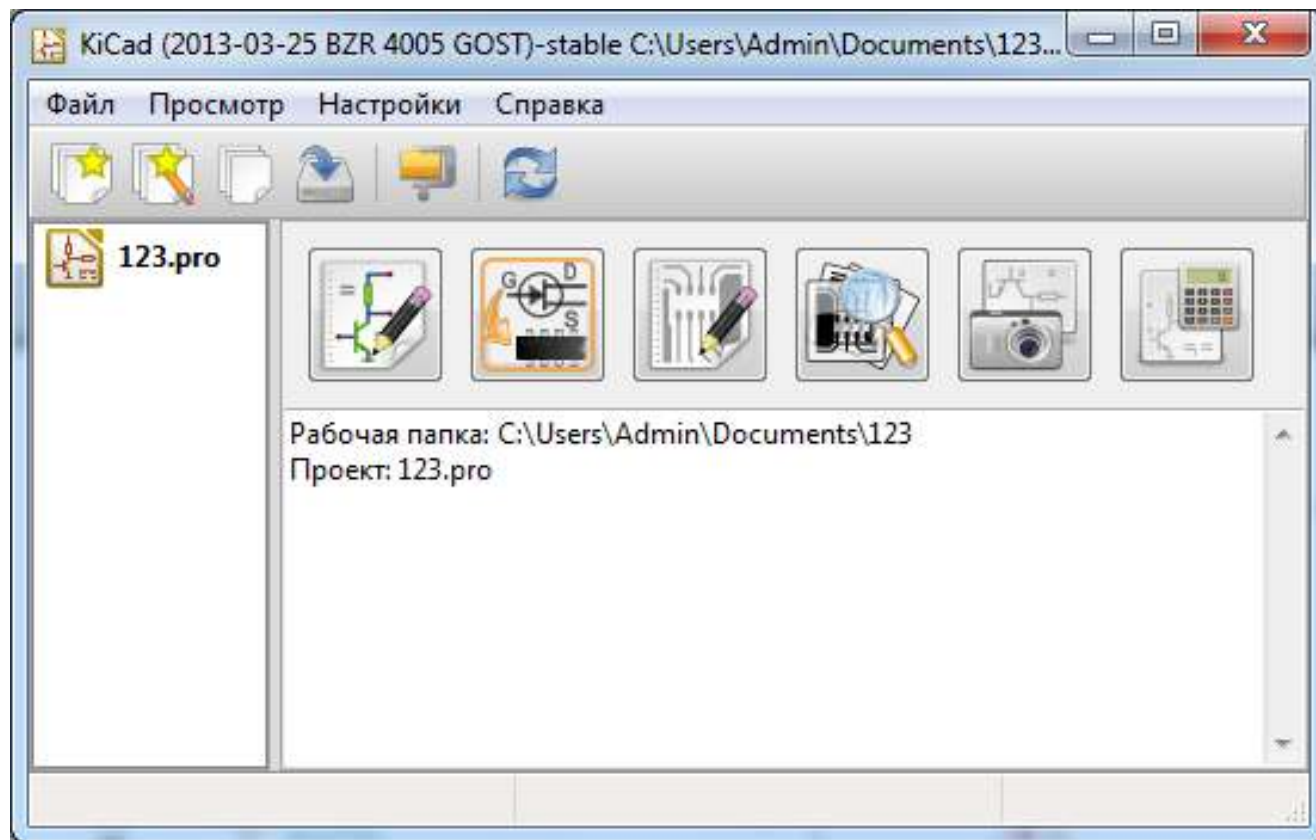
7.6 Маршрут проектирования






7.7 Создание проекта в менеджере проектов **kicad**

Windows:  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 **падстечков** (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 отдельные папки; в противном случае система может сбиться с толку, если файлы из разных проектов будут лежать в одной папке. Прodelайте следующие шаги:



1. Запустите программу KiCad



2. Создайте новый проект

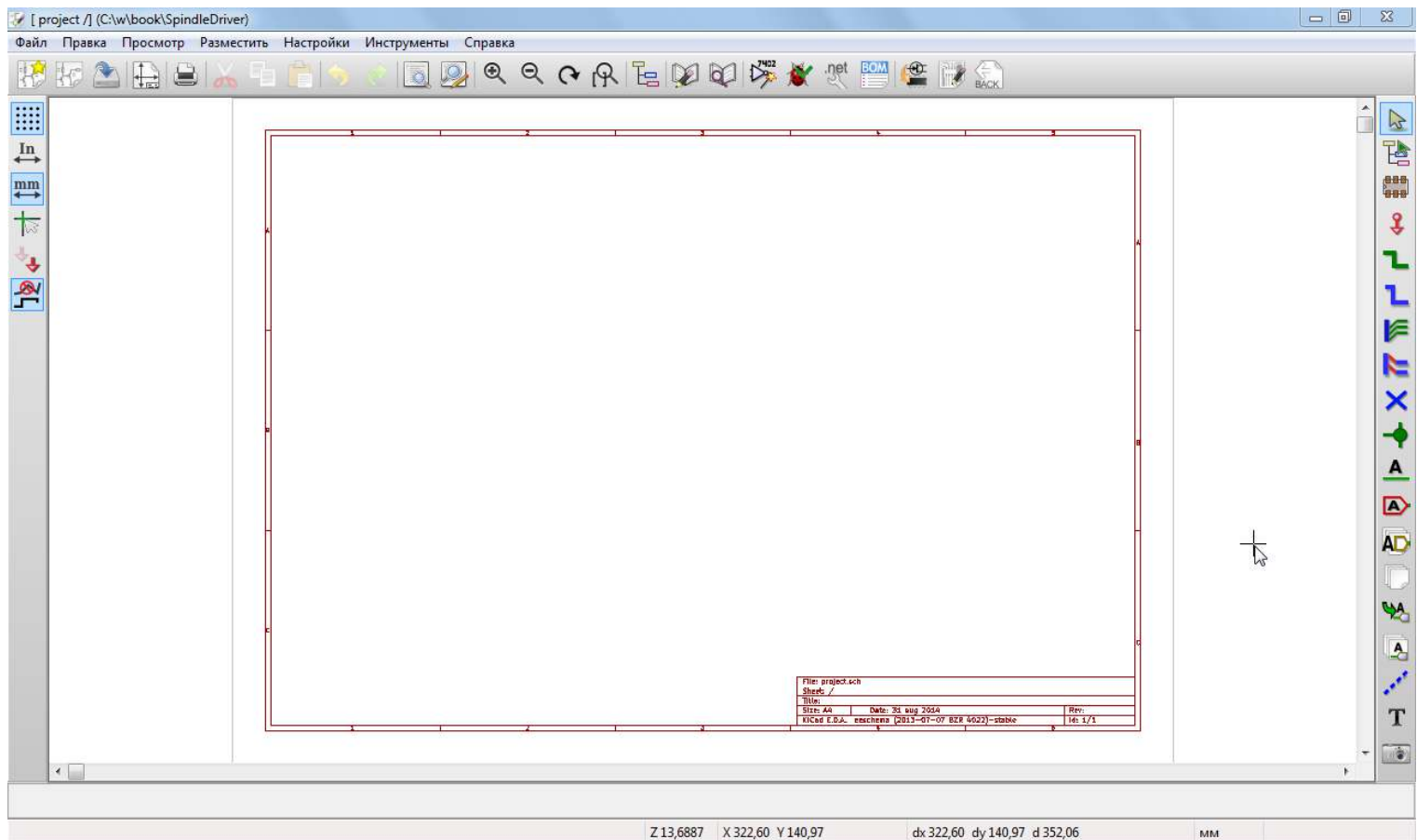
- На панели инструментов KiCad выберите левую иконку с подсказкой **Начать новый проект**, используйте команду меню **Файл** > **Новый** > **Пустой** или сочетание клавиш **Ctrl** + **N**.
 - Создайте папку проекта **DarkSensor**
 - В диалоге **Создать новый проект** выберите созданную папку выберите только что 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.

7.8 Создание принципиальной схемы в **eescema** (часть 1)

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

```
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.

7.9 **eeschema**: 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 VI

Расчет схем и моделирование в ngSPICE

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 видов 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 копия: http://www.mithatkonar.com/wiki/doku.php/kicad/kicad_spice_quick_guide

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

Глава 8

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 совместимой.
- **SpiceOpus**: Коммерческая, но удобная, особенно в плане вывода графиков.

¹ уже включен в винدوزную 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.

8.1 Установка **ngSPICE** под Windows

```

[Win] + [R] > http://ngspice.sourceforge.net/download.html > ng-spice-rework > ngspice-xx_xxxxxx.zip
unzip ngspice-xx_xxxxxx.zip > C:/spice
[Win] > Компьютер > Свойства > Дополнительные параметры системы
Переменные среды > PATH=...;C:/spice/bin

```

8.2 Установка **LT-SPICE** (только Windows)

```

[Win] + [R] > http://www.linear.com/designtools/software/

```

8.3 Установка ngSPICE под Linux

```
root# aptitude install ngspice
```

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

```
tar xvfz 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 ??.

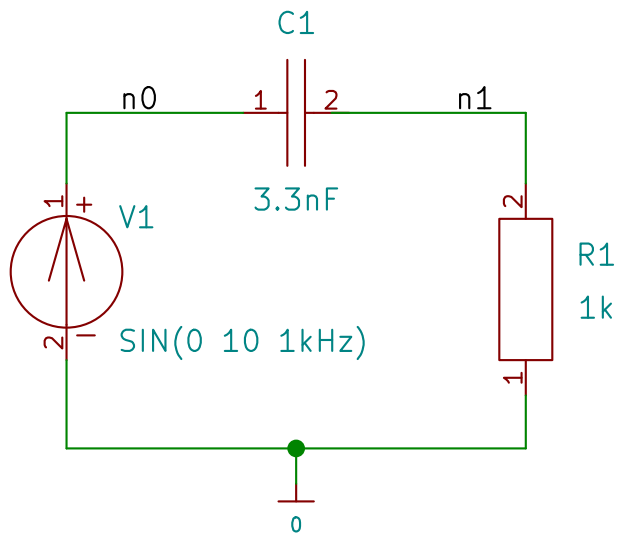
Глава 9

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.

9.1 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 (7.9). 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 **eescema** им будут присвоены уникальные 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 $GND \rightarrow 0$, другие нет.

9.2 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

```
1 * RC_filter
2
3 *Sheet Name:/
4 R1  0 /n1 1k
5 C1  /n0 /n1 3.3nF
6 V1  /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, начинающаяся 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 понадобится сформировать его автоматически, например при анализе паразитных емкостей и индуктивностей печатной платы, которые определяются формой печатных проводников.

9.3 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 (входной) 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 нетлист (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 1 КГц, период $T = 1/F = 0.001 \text{ с} = 1 \text{ мс}$. Мы хотим увидеть как входное напряжение 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
```

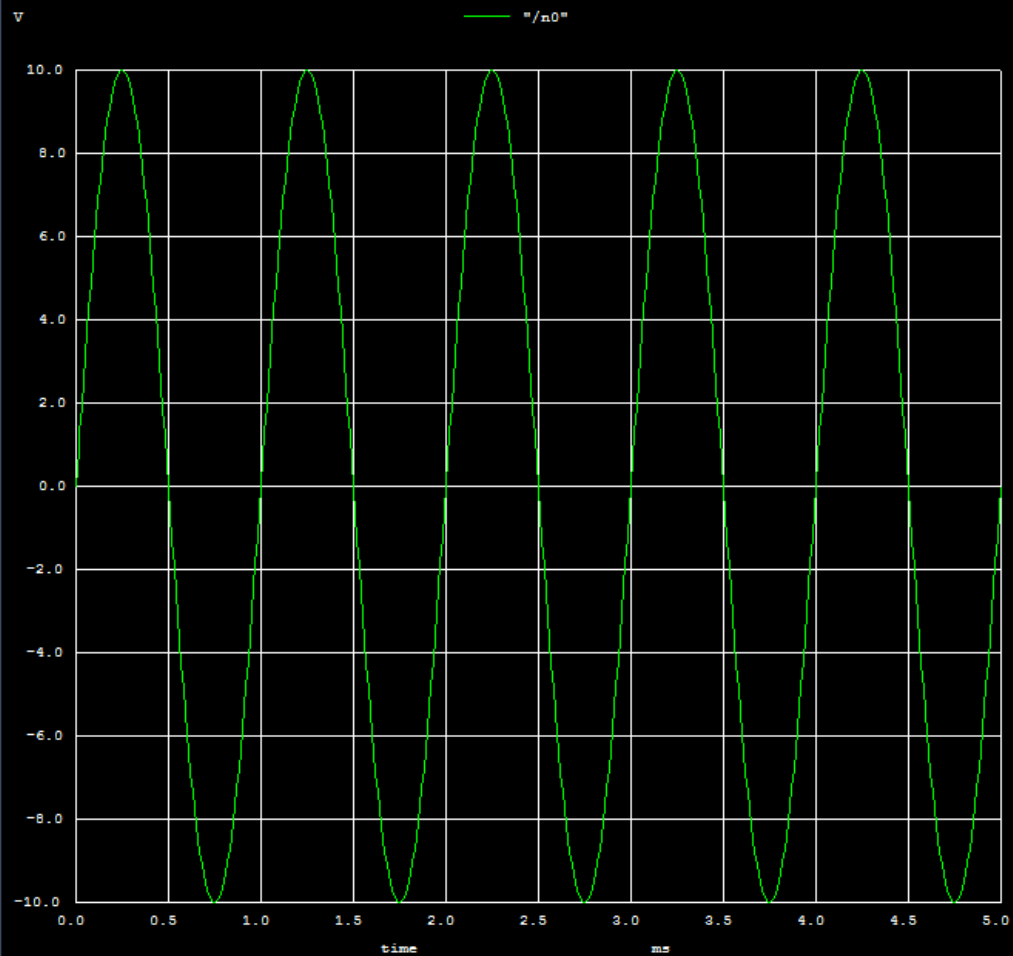
Первый параметр tran определяет **шаг расчета**, второй — **конечное значение времени**. Если не указан третий параметр, начальное время равно 0, иначе третий параметр указывает ненулевое **начальное время**. Ну, это все ☺. Симуляция выполнена. Теперь нам нужно увидеть результат симуляции.

9.4 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"
```

tran1: * eeschema netlist version 1.1 (spice format) creation date: 26.12.2014 16:15:26



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"
```

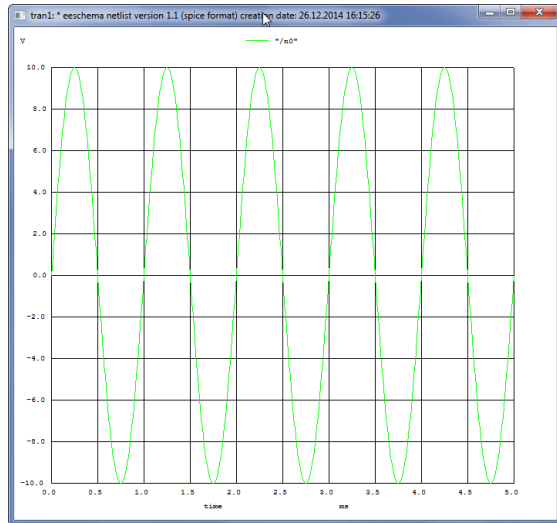


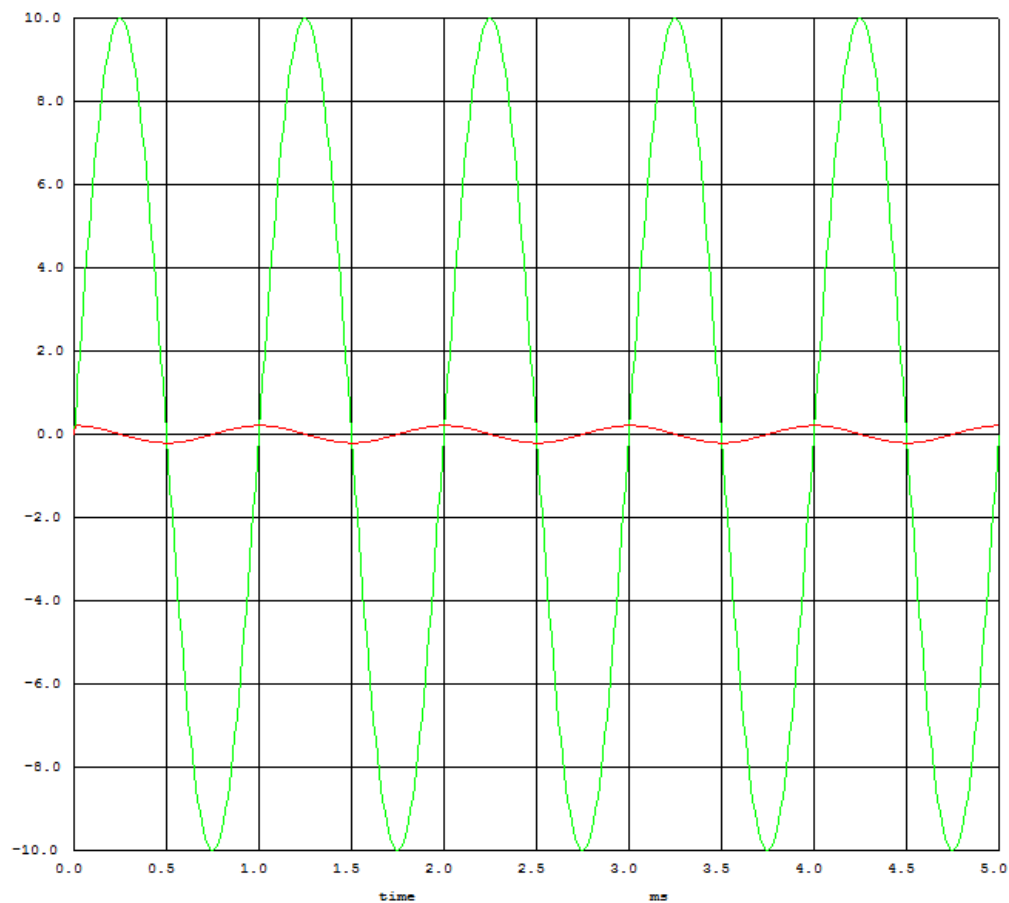
Диаграмма показывает форму входного сигнала, как мы и ожидали, но она нас мало интересует, так как мы ее и задали. Нам интереснее например [напряжение на резисторе](#), кроме того мы попробуем [сравнить два сигнала](#). Это легко сделать указав два имени цепи:

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


tran1: * eeschema netlist version 1.1 (spice format) creation date: 26.12.2014 16:15:26

V ———— "/n1"

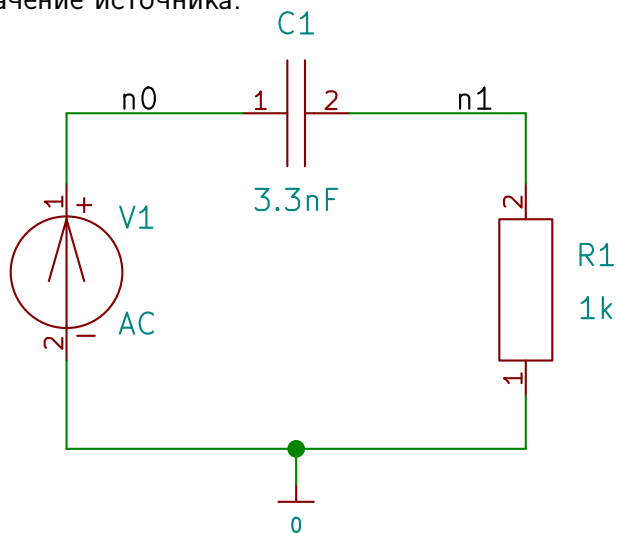
————— "/n0"



9.5 Расчет АЧХ по переменному току (АС симуляция)

Сигнала на резисторе почти не видно. Теперь вопрос: какие частоты попускает наш фильтр? Для определения этого теперь выполним **расчет по переменному току (АС 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 **АС 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 **eeschema** и меняем значение источника:



Создаем нетлист, загружаем его и запускаем команду АС анализа:

```
$ ngspice ACanaliz.cir
```

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

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

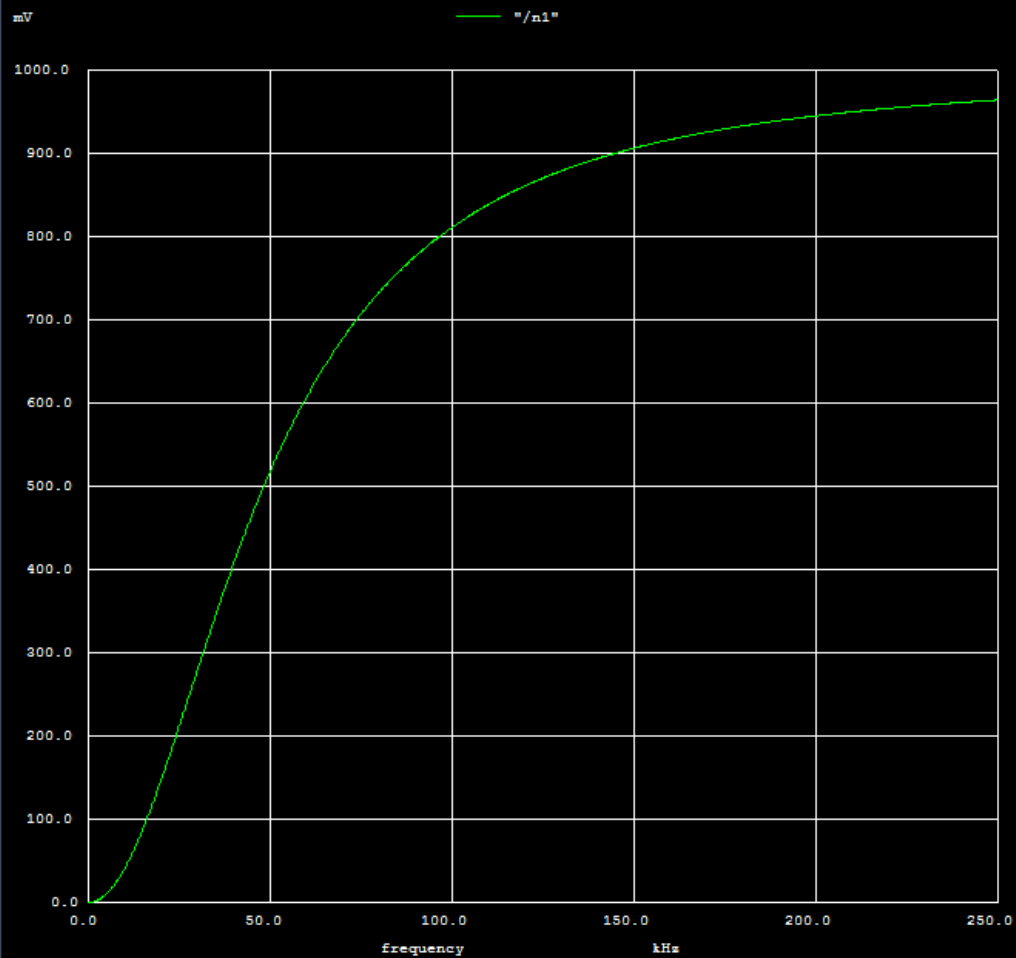
```
Warning: v1: has no value, DC 0 assumed
```

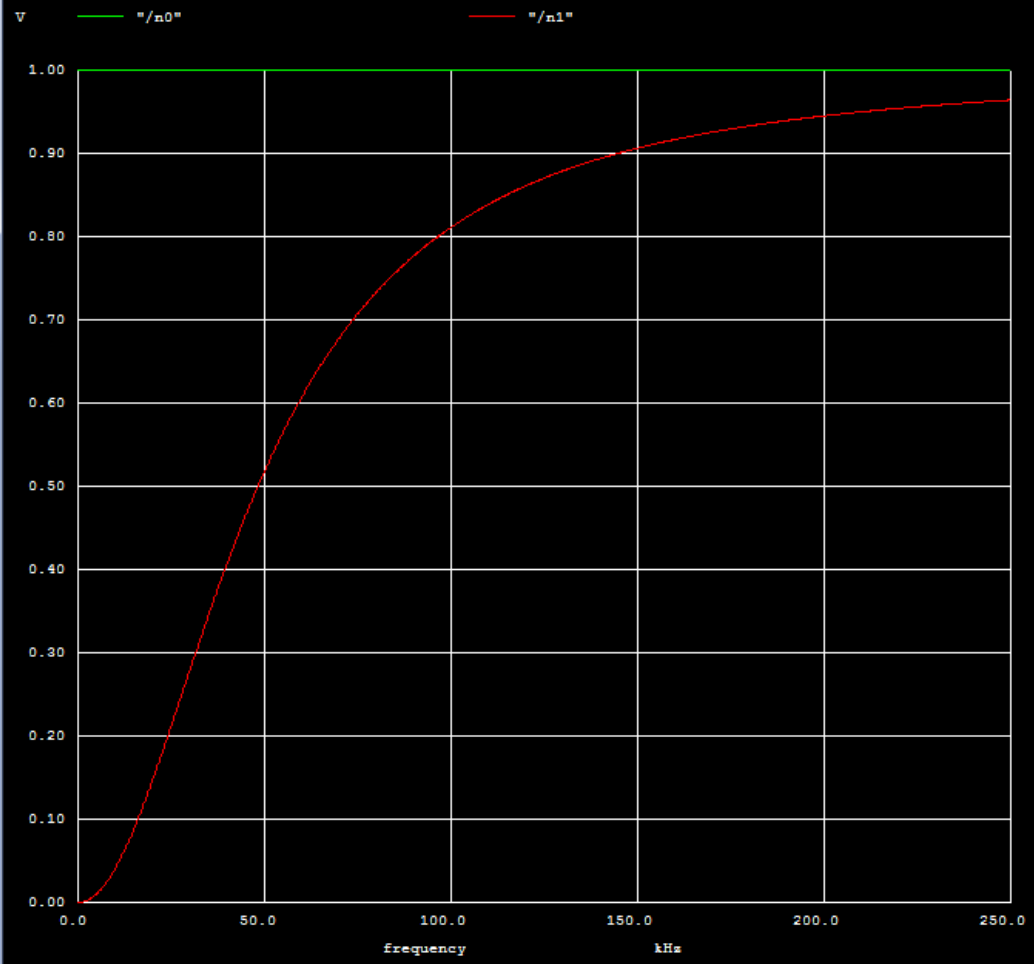
```
No. of Data Rows : 1000
```

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

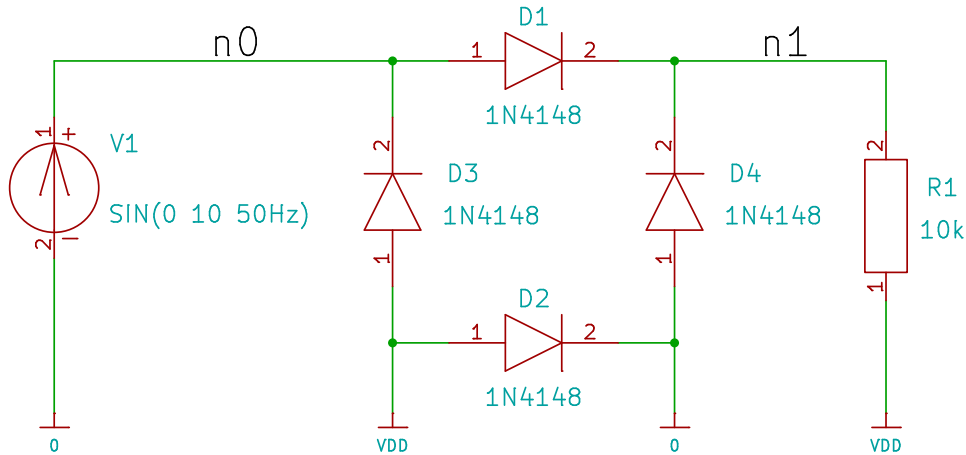
```
ngspice 3 -> plot "/n0" "/n1"
```

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**:





9.6 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

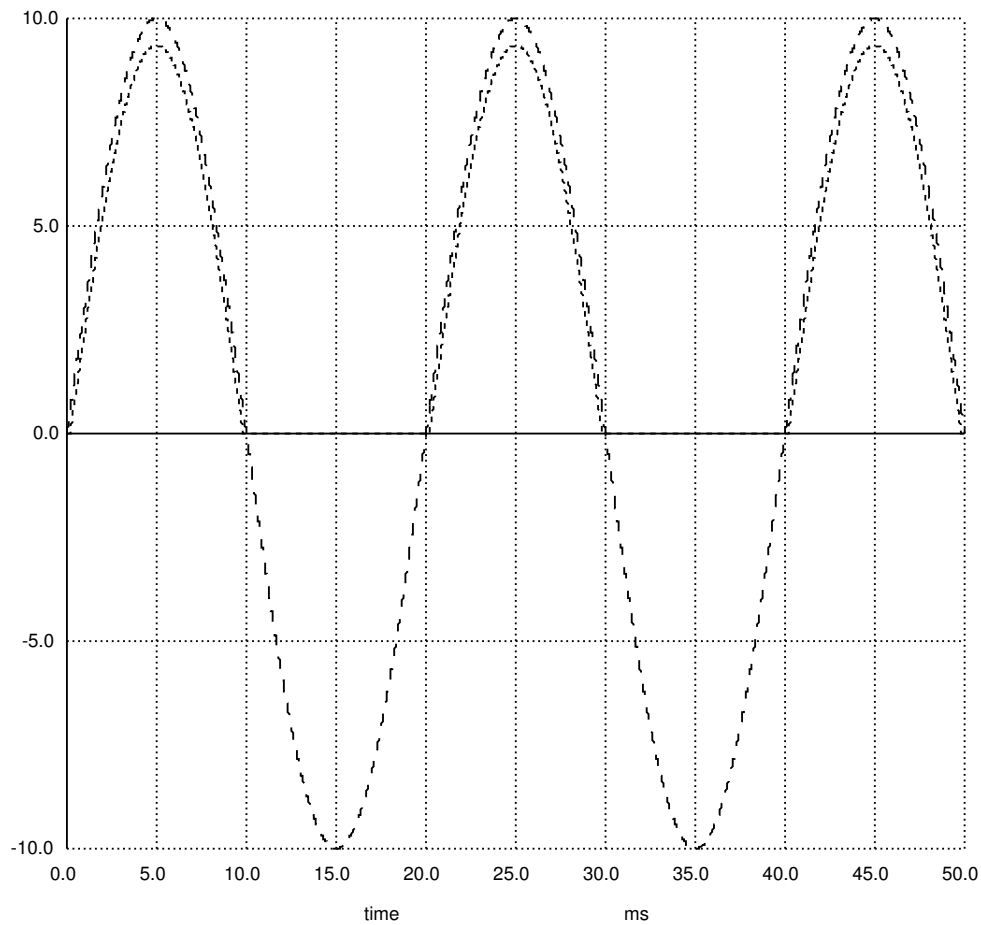
```
1 * Rectifier
2
3 * text before netlist
4
5 *Sheet Name:/
6 D4  0 /n1 1N4148
7 D2  0 0 1N4148
8 D3  0 /n0 1N4148
9 D1  /n0 /n1 1N4148
```

```
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 10.

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 10

Настройка KiCAD для SPICE-моделирования

10.1 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

10.2 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:

eeschema » Меню » Инструменты » Сформировать список цепей » Spice

☒ Формат по умолчанию

☐ Префикс обозначений

☒ Использовать имена цепей

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

Список цепей

10.3 Как это работает

1. Укажите режимы сиуляции, 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 `out`, 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 \pm "GNUCAP ". Я думаю это то же самое что и \pm "PSPICE ", но не уверен на 100%, проверьте в документации.
- Да, вам потребуется немного изучить синтакис SPICE and Nutmeg. Это нетрудно.

2. Запустите симуляцию:



Часть VII

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 «полноценность», этот продукт может рассматриваться как одна из попыток создания базы для решения этой проблемы. Разработчик FreeCAD Юрген Ригель, работающий в корпорации DaimlerChrysler, позиционирует свою программу как первый бесплатный инструмент проектирования механики (сравнивая свой продукт с такими развитыми проприетарными системами как CATIA версий 4 и 5, SolidWorks), созданный на основе библиотеки **Open CASCADE**. Цель программы — предоставить базовый инструментарий этой библиотеки в интерактивном режиме.

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

² копияста: [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, многого еще не хватает, особенно если сравнивать его с коммерческими решениями, и вы можете не найти его достаточно развитым для использования в производственной среде. Тем не менее, есть быстрорастущее сообщество пользователей-энтузиастов, и вы уже можете найти много примеров качественных проектов, разработанных с FreeCAD.

Как и все проекты с открытым исходным кодом, проект FreeCAD не единственный способ работы обеспеченный Вам его разработчиками. Это во многом зависит от роста его сообществу пользователей и разработчиков, 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 копия: http://www.freecadweb.org/wiki/index.php?title=Getting_started

Глава 11

Установка

11.1 ☐ 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

11.2 Linux

Часть VIII

Инструменты и электронное оборудование

Глава 12

Радиомонтажный инструмент

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

12.1 Pro'sKit

Отдельного обзора заслуживает инструмент и наборы Pro'sKit



PK-5308BM универсальный набор инструментов



1PK-616B Набор инструментов для электроники профессиональный



1PK-813B Набор базовых инструментов для электроники

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

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

12.2 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

12.3 Хранение



103-132D Кассетница для деталей и компонентов



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

12.4 Радиомонтаж



8PK-30D Кусачки миниатюрные



1PK-709 Длинногубцы-кусачки



1PK-055S Длинногубцы изогнутые



1PK-29 Круглогубцы



1PK-101T Пинцет прямой



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



PD-374 Тиски на струбцине

12.5 Прочие

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



Глава 13

Паяльное оборудование

13.1 Паяльник

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

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



Паяльник ЭПСН-25/220



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



Паяльник 220В 25Вт ZD-721N 175 р.



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

13.2 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 14

Измерительное оборудование

14.1 Мультиметр

Мультиметр — обязателен, без него работать невозможно¹. Для совсем начинающего больше всего подойдет M32014.1.3 с автодиапазоном, когда освоитесь возьмете вторым прибором что-то из крупных серий M89х/MY6х с измерением температуры² или “рыльцетр”14.5 (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

14.1.1 Mastech M838



Простой, компактный, дешевый, с измерением температуры

14.1.2 Mastech M300



Простой, **очень компактный**, дешевый, в чехле очень удачно помещается в набор инструментов.

14.1.3 Mastech M320



То же что и M30014.1.2, 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.

14.2 Осциллограф

14.3 Логический анализатор

14.4 Генератор сигналов

14.5 Рыльцеметр RLC

Глава 15

Электроинструмент

15.1 Дрель



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



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

Дрель — одноразовая китайчатина от 400 р. Продаются уже брендированные на Леруа Мерлен, наклейка «PID13D01 Ударная дрель 400 Вт, 13 мм». Скорость регулируется глубиной нажатия курка, крутилка на курке ограничивает глубину механически, фиксатор держит скорость близко к минимальной, запаха горелой пластмассы через несколько минут работы на холостом ходу нет.

По надежности 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 сколхозить некоторое подобие настольных станочков 15.1 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.



15.2 Лобзик



Praktyl 350 Вт 356 р.



Makita 4329 2260 р.

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

15.3 Жвигатель

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 15.1, 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 точилке¹.

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 шпиндели 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



Жвигатель Вятка-Автомат 19?? г.



Двигатель печи Камаза



АИРЕ 56 В2, 0.2 КВт



Воздушный шпиндель с цангой ER11

Съемные фрезерные шпиндели, поставляются отдельно или в комплекте с насадкой ручного фрезера по дереву. Лучшие, со стальной шейкой — Kress, активно применяются хобби-ЧПУшниками. Попроще и сильно дешевле делал Интерскол, иногда попадаете пополам. Недостаток как универсального привода — они высокоскоростные, возникают проблемы с понижающими передачами. Применение — приводной высокоскоростной инструмент: боры, фрезы по дереву, микроинструмент для гравиров (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 IX

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 изготовления 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 16

Настольные станки

На основе ¹

© 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 копия: <http://rutracker.org/forum/viewtopic.php?t=3126529>

Глава 17

Самодельная оснастка

Глава 18

Промышленные станки

Иногда хоббиту удастся 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.

18.1 1A616: станок токарно-винторезный



- 18.1.1 Назначение и области применения
- 18.1.2 Распаковка и транспортировка
- 18.1.3 Фундамент станка, монтаж и установка
- 18.1.4 Подготовка станка к первоначальному пуску
- 18.1.5 Паспортные данные
- 18.1.6 Описание основных узлов
- 18.1.7 Смазка
- 18.1.8 Первоначальный пуск
- 18.1.9 Указания по технике безопасности
- 18.1.10 Настройка
- 18.1.11 Регулирование
- 18.1.12 Ведомость комплектации

Часть X

Разработка ПО для встраиваемых систем

Глава 19

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 text¹ файлов с

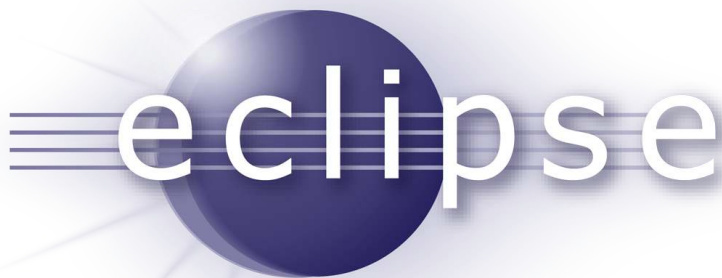
¹ файлы не включающие непечатаемых символов и бинарных данных, которые можно причитать простым выводом на экран командами типа **type**, **cat**, **more**

- 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 деревья 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** 19.1: 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  **ECLIPSE** Luna работает заметно быстрее (особенно при запуске).
2. **Code::Blocks** 19.2: легкая среда для разработки на C/C₊⁺, для других языков может потребоваться написать свои модули или файлы описания синтаксиса
3.  **Vim** 19.3: самый легкий и **портатбельный** 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 ☺

19.1 ☰eclipse



19.1.1 Установка ☰eclipse под ☰Windows

☰+R >> <https://eclipse.org/> >> Download >> Eclipse Luna release for ☰Windows

Качаем архив базовой системы: Eclipse IDE for Java Developers >> ☰Windows 32/64 Bit

Или сразу сборку CDT☰ECLIPSE: Eclipse IDE for C/C++ Developers >> ☰Windows 32/64 Bit

19.1.2 Установка ☰eclipse под Linux

☰+R >> <https://eclipse.org/> >> Download >> Eclipse Luna release for >> Linux

Качаем архив базовой системы: Eclipse IDE for Java Developers >> Linux 32/64 Bit

Или сразу сборку CDT⊖ECLIPSE: Eclipse IDE for C/C++ Developers >> Linux 32/64 Bit

Пока качается, параллельно устанавливаем в систему Java-рантайм:

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

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

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

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

`.blackbox.menu`

19.1.3 Установка CDT

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

⊖ECLIPSE >> Help >> Install New Software...
Work with >> Add >> Add repository
Name >> CDT
Location >> <http://download.eclipse.org/tools/cdt/releases/8.5>

OK
Work with > CDT
CDT Main Features > ☒ C/C++ Development Tools
CDT Optional Features
Парсер файлов исходников на диалекте C99: ☒ C99 LR Parser
Поддержка **gcc** в режиме кросс-компиляции: ☒ GCC Cross Compiler Support
Аппаратная отладка через **gdb**: ☒ GDB Hardware Debugging
Next > Next > Accept > Finish

19.1.4 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

19.1.5 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.

19.1.6 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

19.1.7 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](http://www.simplecoding.org/proverka-orfografii-v-eclipse.html). И ☹ECLIPSE в этом смысле полностью соответствует ожиданиям.

² копияста: <http://www.simplecoding.org/proverka-orfografii-v-eclipse.html>

Долго объяснять что к чему нет смысла. Проверка орфографии встроена в ECLIPSE и если вы пишете 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

 + 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

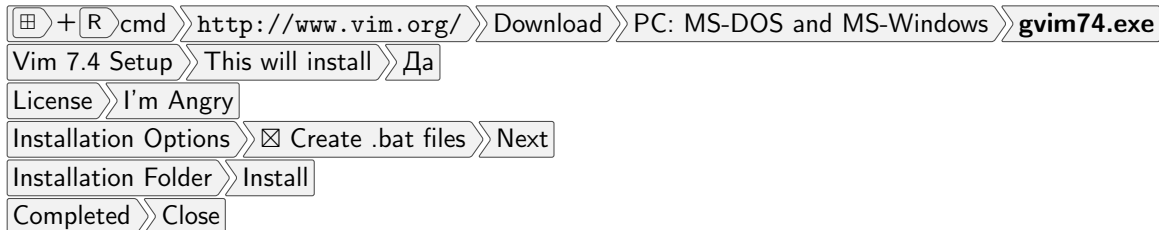
Apply >> OK

19.2 Code::Blocks

19.3 (g)Vim



19.3.1 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

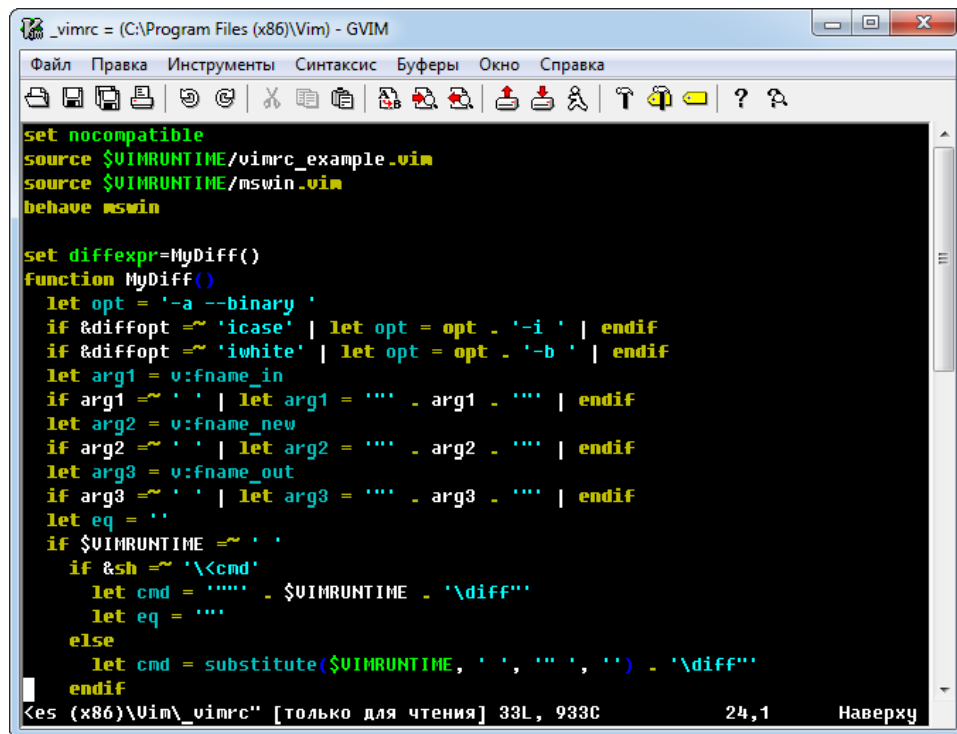
Если не получилось (под Windows 7):

⌘ + 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 the GVIM editor window titled "_vimrc = (C:\Program Files (x86)\Vim) - GVIM". The menu bar includes "Файл", "Правка", "Инструменты", "Синтаксис", "Буферы", "Окно", and "Справка". The toolbar contains icons for file operations, editing, and navigation. The main text area displays a Vim script with syntax highlighting: keywords like "set", "source", "function", "let", "if", "endif", and "else" are in green, while comments and other text are in yellow. The script includes settings for compatibility, sourcing runtime files, and a custom diff function. The status bar at the bottom shows the file path, a read-only warning, line and column numbers, and a "Наверх" (Up) button.

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

set diffoptions=MyDiff()
function MyDiff()
  let opt = '-a --binary '
  if &diffoptions =~ 'icase' | let opt = opt . '-i ' | endif
  if &diffoptions =~ '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
  endif
endfunction
<es (x86)\Vim\_vimrc" [только для чтения] 33L, 933C          24,1      Наверх
```

19.3.2 Выход из (g)Vim

Esc : ! q Enter

19.3.3 Выход с автосохранением

`Esc` `Shift` + `Z` `Shift` + `Z`

19.3.4 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](#)

19.3.5 Переход в режим команд

`Esc`

19.3.6 Запись редактируемого файла


`Esc` : `w` `Enter`

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

`Esc` : `!` `w` `Enter`

19.3.7 Перезагрузка файла

Для перезагрузки возможно измененного извне файла или отмены всех несохраненных изменений

 :  

19.3.8 Отмена последних изменений (undo)

   . . .

Глава 20

Make: управление сборкой проектов

Глава 21

VCS: системы контроля версий

21.1 CVS

21.2 Subversion

21.3 Git

21.3.1 GitHub

Глава 22

Вспомогательные скрипты на языке Python

Глава 23

Основы Си и C_{+}^{+}

23.1 Установка MinGW (win32)

23.2 Особенности C_{+}^{+} в embedded

23.3 Сборка кросс-компилятора GNU toolchain

Глава 24

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 хинтами 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.capsl.udel.edu/courses/cpeg421/2012/slides/Tutorial-Flex_Bison.pdf

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

Лёксер/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 **лексёмы** или **токёны**.

Цель лексера — подготовить последовательность лексем для входа другой программы.

В самых 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 (лексэма), 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 высокоуровневом 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

yywrap	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}*

Декларации на Си прописываются в скобках %{ }%

5

Makefile

```
1
2 all: xliff.xliff Y.tex
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

```

1 %option noyywrap
2
3 %{
4 #include <iostream>
5 #include <string>
6 using namespace std;
7 #define YYSTYPE string
8 #include "tex2xiff.tab.h"
9 %}
10
11 %%
12 ([a-z0-9]+\.)+(com|nz) { yylval = yytext; return URL; }
13
14 \.[\n\ ]+      { yylval = "."; return SEP; }
15 \&             { yylval = "&"; return CHAR; }
16 \n{2,22}      { yylval = ""; return SEP; }
17 \n            { yylval = " "; return CHAR; }
18 \\item        { yylval = ""; return SEP; }
19 .             { yylval = yytext; return CHAR; }
20 %%

```

24.2 Генератор синтаксических анализаторов **bison**

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


```
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>\n<source>";
32 int yyp=yyparse();
33 cout << "</source></trans-unit>\n</body>\n";
34 cout << "</file>\n";
35 cout << "</xliff>\n";
36 return yyp;
37 }
```

Глава 25

LLVM и разработка собственных компиляторов

25.1 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`.

25.2 Создание компилятора с помощью инфраструктуры LLVM

¹ © IBM

¹ копия: <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 представление**⁴ 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.

25.3 Инструменты **llc** 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 копия: <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 сбоя нативного исполняемого кода, чтобы найти ошибки в своей программе. Именно здесь оказывается полезным инструмент **lli**, поскольку он способен непосредственно исполнять байт-код (с помощью интерпретатора).

25.4 Семантический анализ

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

25.5 Оптимизация

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

25.6 Кодогенерация

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

25.7 Транслятор Паскаля

Часть XI

Микроконтроллеры Cortex-Mx

Глава 26

Производители

26.1 ST Microelectronix STM32

26.2 LPC

26.3 Миландр

Глава 27

Отладочные платы

27.1 LeafLabs Maple Mini: STM32F103 /Cortex-M3/

27.2 Серия STM32 STM32DISCOVERY

27.2.1 STM32DISCOVERY: STM32F103 /Cortex-M3/

27.2.2 STM32F4DISCOVERY: STM32F406 /Cortex-M4F/

27.2.3 STM32F0DISCOVERY: STM32F040 /Cortex-M0/

Часть XII

ПЛИС

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

Часть XIII

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 позволяет 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 взаимодейству-

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 28

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 адресация
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 29

libUSB

29.1 драйвер для устройства USB

1

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

Глава 30

Поддержка USB в Linux

30.1 Опции ядра

30.1.1 режимы `host/client/otg` и хост-контроллеры xHCI

30.1.2 `data storage`: носители данных

30.1.3 `hid`: клавиатура, мышь, джойстик

30.1.4 USB-периферия: сеть, звук,...

30.2 Настройка `hotplug` и автомонтирования USB носителей

30.3 Сборка и настройка `libusb`

30.4 Примеры программ низкоуровневого ввода/вывода

Часть XIV

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 сторонних 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;
- Дремуемость техспециалистов, контуженных ТурбоПаскалем и Windowsom;

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, медиacentры).

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

¹ будем называть его **emLinux**

² 32-бит, необходим блок MMU

³ в т.ч. является основой Android

Глава 31

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/>

31.1 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>

31.2 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:

```
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

31.3 syslinux.cfg

syslinux настраивается текстовым файлом **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 финт нужен для включения графики на ASUS EeePC 701: режим 800×480 недоступен для включения через параметр ядра `vga=`, поэтому приходится использовать возможности `syslinux`.

Глава 32

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, защитой 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 фишка — [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??**.

32.1 Требования к системе сборки (BUILD-хост)

Требования жесткие — 2х-ядерный процессор, 2+ Гб ОЗУ, для 4+ Гб ОЗУ нужен 64х-битный дистрибутив Linux (рекомендую Debian), и естественно никаких виртуалок. Возможна установка системы на флешку, в этом случае требования к ОЗУ еще более ужесточаются — потребуется каталог 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.

32.2 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
2 cat \
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/pascal.mk \
19 mk/user.mk \
20 mk/libs.mk \
21 mk/apps.mk \
22 mk/emu.mk \
23 mk/root.mk \
24 mk/boot.mk \
25 mk/info.mk \
26 > Makefile
```

32.3 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
```

32.4 Общий порядок сборки

Каждый пакет собирается командой:

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


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

1. **dirs** создание дерева каталогов 32.6
2. **gz** загрузка архивов исходников 32.7
3. **tc** сборка кросс-компилятора 32.14.6
 - (a) **binutils** ассемблер, линкер и утилиты 32.14.7
 - (b) **cclibs** библиотеки для сборки **gcc** 32.14.8
 - (c) **gcc0** сборка минимального кросс-компилятора Си 32.14.9
4. **core** сборка основной системы 32.14.11
 - (a) **kernel** ядро Linux 32.14.12
 - (b) **ulibc** библиотека **uClibc** 32.14.13
 - (c) **busybox** набор утилит **busybox** 32.14.15
 - (d) **gcc** пересборка полного кросс-компилятора Си/ C_+^+ 32.14.14
5. **libs** сборка библиотек **\${LIBS}** 32.14.16
6. **apps** сборка прикладных пакетов **\${APPS}** 32.14.17
7. **user** сборка пользовательского кода 32.14.18
8. **root** формирование корневой файловой системы 32.14.19
9. **boot** сборка загрузчика 32.14.20 **syslinux/grub/uboot**

10. **emu** запуск собранной системы в эмуляторе 32.14.24

11. **netboot** сетевая загрузка 32.15

12. **firmware** прошивка на устройство 32.16

32.5 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.

32.6 **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]ou[RC]e codes 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 DIRS = $(GZ) $(SRC) $(TMP) $(TC) $(ROOT) $(BOOT) $(USR) $(USRBIN) $(USRLIB) $(PACK)
21 .PHONY: dirs
22 dirs:
23 ____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 мониторить 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 **HW32.9** и имени приложения **APP32.8**.

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)
```

32.7 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
11 .PHONY: gz_cc
12 gz_cc:
13 ____$(WGET) http://ftp.gnu.org/gnu/binutils/$(BINUTILS).tar.bz2
14 ____$(WGET) http://gcc.skazkaforyou.com/releases/$(GCC)/$(GCC).tar.bz2
15 ____$(WGET) ftp://ftp.gmplib.org/pub/gmp/$(GMP).tar.bz2
16 ____$(WGET) http://www.mpfr.org/mpfr-current/$(MPFR).tar.bz2
17 ____$(WGET) http://www.multiprecision.org/mpc/download/$(MPC).tar.gz
18
19 .PHONY: gz_core
20 gz_core:
21 ____$(WGET) https://www.kernel.org/pub/linux/kernel/v3.x/$(KERNEL).tar.xz
22 ____$(WGET) http://www.uclibc.org/downloads/$(ULIBC).tar.xz
23 ____$(WGET) http://busybox.net/downloads/$(BUSYBOX).tar.bz2
24
25 .PHONY: gz_libs
```

```

26 gz_libs:
27 ____$(WGET) https://www.libsdl.org/release/$(SDL).tar.gz
28 ____$(WGET) https://www.libsdl.org/projects/SDL_image/release/$(SDL_IMAGE).tar.gz
29 ____$(WGET) http://www.libsdl.org/projects/SDL_ttf/release/$(SDL_TTF).tar.gz
30 ____$(WGET) http://download.sourceforge.net/libpng/$(PNG).tar.xz
31 ____$(WGET) http://download.savannah.gnu.org/releases/freetype/$(FREETYPE).tar.bz2
32 ____$(WGET) http://zlib.net/$(ZLIB).tar.xz
33
34 .PHONY: gz_sdk
35 gz_sdk:
36 ____$(WGET) ftp://ftp.hu.freepascal.org/pub/fpc/dist/$(FPC_VER)/source/$(FPC).source.tar.g
37 ____$(WGET) ftp://ftp.hu.freepascal.org/pub/fpc/dist/$(FPC_VER)/source/fpcbuild-$(FPC_VER).

```

32.8 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 **vsx** пакетов:

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

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

- **LIBS**: набор используемых библиотек
- **PACKS**: набор используемых программных пакетов 32.14

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 $(USBIN)/sdl_hello $(USBIN)/sdl_rect $(USBIN)/sdl_clock
```

32.9 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	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	AllWinnerA10	armhf	1G		<input type="checkbox"/> μ SD	<input checked="" type="checkbox"/>			
rpi	BCM2835	armel	512M		<input type="checkbox"/> SD	<input checked="" type="checkbox"/>			
qemuMIPS		mips	32M+						
mr3020	AR7240	mips	32M	4M		<input checked="" type="checkbox"/>		<input type="checkbox"/> AR9331	
vocore	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 = qemu386
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

```

32.10 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.

32.10.1 qemu386: эмулятор QEMU

hw/qemu386.mk

```
1 # QEMU emulator: i386 mode
2 CPU = i486sx
3 QEMU_CFG = -m 64M -net none \
4 ____-append "vga=0x312"
5 #vga=ask
6 #0x312 640x480x24
7 #0x315 800x600x24
```

32.10.2 eeepc701: ASUS Eee PC 701

hw/eeepc701.mk

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

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



CPU	Celeron 1037U
ОЗУ	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

<http://www.ixbt.com/news/hard/index.shtml?17/33/31>

На текущий момент эта материнка — оптимальный вариант для офисного, рабочего неигрового компьютера, или базы для изготовления мобильной рабочей станции в миникеесе: 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,
ОЗУ	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
2 CPU = Celeron1037U
```

32.11 ARM

32.11.1 qemuARM: эмулятор QEMU

32.11.2 cubie1: Cubie Board v.1

32.11.3 rpi: Raspberry Pi model B

32.12 MIPS

32.12.1 qemuMIPS: эмулятор QEMU

32.12.2 mr3020: роутер MR3020

mr3020

32.12.3 vocore: VoCore

vocore

32.12.4 bswift: BlackSwift

bswift

32.13 CPU: Конфигурации процессоров

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

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

32.13.1 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
2 ARCH = i386
3 TARGET = celeronm-linux-uclibc
```

32.13.2 ARM

32.13.3 MIPS

cpu/AR7240.mk

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

cpu/RT5350.mk

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

32.14 Пакеты

32.14.1 versions.mk: Версии пакетов

32.14.2 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
```

32.14.3 **core:** ядро

mk/versions.mk

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

32.14.4 **boot:** загрузчики

mk/versions.mk

32.14.5 **libs:** библиотеки

mk/versions.mk

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


32.14.6 tc: сборка кросс-компилятора

```
1
2 CFG_BINUTILS = --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_BINUTILS) $(CFG_CCLIBS) \
15 ____--disable-shared --disable-threads \
16 ____--without-headers --with-newlib \
17 ____--enable-languages="c"
18
19 CFG_GCC = $(CFG_BINUTILS) $(CFG_CCLIBS) \
20 ____--enable-threads --enable-libgomp \
21 ____--enable-languages="c,c++"
22
23 .PHONY: cross
24 cross: binutils cclibs gcc0
25
```

```
26 .PHONY: binutils
27 binutils: $(SRC)/$(BINUTILS)/README
28 ____rm -rf $(TMP)/$(BINUTILS) && mkdir $(TMP)/$(BINUTILS) &&\
29 ____cd $(TMP)/$(BINUTILS) &&\
30 ____$(SRC)/$(BINUTILS)/$(BCFG) $(CFG_BINUTILS) &&\
31 ____$(MAKE) && 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
```

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

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

arch/i386.mk

```
1 CFG_ARCH = --disable-multilib
```

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

arch/arm.mk

```
1 TARGET = arm-linux-uclibceabihf
```

```
2 CFG_ARCH = --with-float=hard
```

```
3 #--disable-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
```

32.14.8 **cclibs**: библиотеки для сборки **gcc**

gmp **mpfr** **mpc**

32.14.9 **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
```

32.14.10 gcc: пересборка полного кросс-компилятора Си/C₊

Пакет собирается **после сборки core**.

32.14.11 core: сборка основной системы

mk/core.mk

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

32.14.12 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 $(CFG_KERNEL) headers_install
27
28 .PHONY: kernel-i386-fix
29 kernel-i386-fix:
30 ____cp \
31 ____$(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 (эмулятор 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 `<px>-(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 ≤ 1 Gb ОЗУ
 - KERNEL_GZIP=y ядро сжимать алгоритмом gzip⁵
 - 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 1 KHz
 - исполняемые форматы файлов
 - * 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

⁵ важно для загрузчиков на не-i386 системах

⁶ повышенная или гарантированная отзывчивость системы на внешние события

- * **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** командная консоль на COM-портах
- **VGA_CONSOLE=y** командная 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

- **cpu 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

32.14.13 **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
 - **cpu** 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/декструктор
- `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 ARCH_HAS_MMU=y
2 ARCH_USE_MMU=y
3 UCLIBC_HAS_FPU=y
4 UCLIBC_HAS_FLOATS=y
5 #DO_C99_MATH=y
6
7 HAVE_SHARED=y
8 LDSO_CACHE_SUPPORT=y
9 LDSO_PRELOAD_ENV_SUPPORT=n
10 LDSO_LD_LIBRARY_PATH=n
11
12 UCLIBC_CTOR_DTOR=y
13 LINUXTHREADS_OLD=y
14
```



```
15 UCLIBC_LINUX_SPECIFIC=y
16 UCLIBC_LINUX_MODULE_26=n
17 UCLIBC_HAS_LFS=y
18
19 UCLIBC_HAS_LFS=y
20 UCLIBC_HAS_REALTIME=y
21 UCLIBC_HAS_STRING_GENERIC_OPT=y
22 UCLIBC_HAS_STRING_ARCH_OPT=y
23
24 DOSTRIP=y
25 RUNTIME_PREFIX=""
26 DEVEL_PREFIX=""
27
28 # BB needs
29 UCLIBC_HAS_CTYPE_TABLES=y
30 UCLIBC_HAS_NETWORK_SUPPORT=y
31 UCLIBC_HAS_FNMATCH=y
32 UCLIBC_HAS_GNU_GETOPT=y
33 UCLIBC_HAS_REGEX=y
34 UCLIBC_SUSV3_LEGACY=y
35
36 ## Python needs
37
38 #DO_C99_MATH=y
39 #UCLIBC_HAS_LONG_DOUBLE_MATH=n
40 #UCLIBC_HAS_WCHAR=y
41 #UCLIBC_SUSV4_LEGACY=y
42 #UCLIBC_HAS_PTY=y
```

```
43 #UNIX98PTY_ONLY=n
44 #UCLIBC_HAS_LIBUTIL=y
45
46 #DOMULTI=y
47 #UCLIBC_HAS_IPV4=y
48 #UCLIBC_HAS_COMPAT_RES_STATE=n
49 #UCLIBC_HAS_SYSLOG=y
50 ## ALSA needs
51 #UCLIBC_HAS_BSD_ERR=y
52
53 # canadian cross needs
54 UCLIBC_HAS_WCHAR=y
```

ulibc/arch/i386

```
1 TARGET_i386=y
```

ulibc/cpu/i486sx

```
1 CONFIG_486=y
```

32.14.14 gcc: пересборка полного gcc

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

32.14.15 busybox: набор утилит busybox

```

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 приходится 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`.

32.14.16 **libs**: сборка библиотек $\${LIBS}$

32.14.17 **apps**: сборка прикладных пакетов $\${APPS}$

32.14.18 **user**: сборка пользовательского кода

32.14.19 **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     cp README.md $(ETC)/
8     chmod +x $(ETC)/init.d/*
9     # 2
10    $(LDCONFIG) -v -r $(ROOT)
11    # 3
12    ln -fs /sbin/init $(ROOT)/init
13    # 4
14    cp -r share $(ROOT)/
15    # 5
16    cd $(ROOT) && find . -type f | egrep -v $(ROOTREX) > $(PACK)/allfiles
17    pack/pack.py $(PACK)
18    cd $(ROOT) && cat $(PACK)/rootfiles | cpio -o -H newc > $(BOOT)/$(HW)$(APP).cpio
19    cat $(BOOT)/$(HW)$(APP).cpio | gzip -9 > $(BOOT)/$(HW)$(APP).rootfs
```

```
20 ____# 6
21 ____sh $(PACK)/mkpacks.rc
```

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
3 mkdir /tmp
```

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

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

/etc/init.d/rcD

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

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

mk/boot.mk

32.14.21 **syslinux**

32.14.22 **grub**

32.14.23 **uboot**

32.14.24 **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 .PHONY: emu
4 emu: $(BOOT)/$(HW)$(APP).kernel $(BOOT)/$(HW)$(APP).rootfs
5 ____qemu-system-$(ARCH) $(QEMU_CFG) \
6 ____-kernel $(BOOT)/$(HW)$(APP).kernel \
7 ____-initrd $(BOOT)/$(HW)$(APP).rootfs
```

hw/qemu386.mk

```
1 # QEMU emulator: i386 mode
2 CPU = i486sx
3 QEMU_CFG = -m 64M -net none \
4 ____-append "vga=0x312"
5 #vga=ask
6 #0x312 640x480x24
7 #0x315 800x600x24
```

Переменная QEMU_CFG задает параметры запуска QEMU:

-m	объем ОЗУ на эмулируемой системе
-net none	отключить сеть и iPXE boot
<hr/>	
-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

32.15 **netboot**: Сетевая загрузка

32.16 Прошивка на устройство

32.17 RT-патч

32.18 SDK: расширения для on-board разработки

Иногда на целевой системе требуется набор средств программирования, для выполнения

- сложных скриптов (Python 32.18.6),
- математических вычислений⁷ (GCL 32.18.7/Maxima 32.18.8),
- 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

32.18.1 **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-gnueabi 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 = --prefix=$(USR)
4 #--enable-lto
5 CFG_CAN_GCC = $(CFG_CAN_BIN) \
6 ____--enable-threads --enable-libgomp \
7 ____--enable-languages="c"
8
9 .PHONY: canadian
10 canadian: $(SRC)/$(BINUTILS)/README
11 ____# binutils
12 ____rm -rf $(TMP)/$(BINUTILS) && mkdir $(TMP)/$(BINUTILS) &&\
13 ____cd $(TMP)/$(BINUTILS) &&\
14 ____$(XPATH) $(SRC)/$(BINUTILS)/$(TCFG) \
15 ____$(CFG_CAN_BIN) --target=$(T) $(O) --program-prefix=$(P) &&\
16 ____$(MAKE) && $(PACKINSTALL) -strip
```

```

17 #___# gcc
18 #___rm -rf $(TMP)/$(GCC) && mkdir $(TMP)/$(GCC) &&\
19 #___cd $(TMP)/$(GCC) &&\
20 #___$(XPATH) $(SRC)/$(GCC)/$(TCFG) \
21 #___$(CFG_CAN_GCC) --target=$(T) $(O) --program-prefix=$(P)
22 #___cd $(TMP)/$(GCC) &&$(XPATH) $(MAKE) all-gcc
23
24 .PHONY: binhost
25 binhost:
26 ___make canadian T=$(TARGET) P= O="$(CFG_ARCH) $(CFG_CPU)"
27
28 .PHONY: bin486
29 bin486:
30 ___make canadian T=i486-elf P=i486- O="--with-cpu=i486"
31
32 .PHONY: binavr
33 binavr:
34 ___make canadian T=avr P=avr- O=
35
36 .PHONY: bincmx
37 bincmx:
38 ___make canadian T=arm-none-eabi P=cmx- \
39 ___O="--enable-interwork --disable-multilib"

```

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

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

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

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

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

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

32.18.5 **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
```

32.18.6 **python:** интерпретатор Python

32.18.7 **gcl:** GNU Common Lisp

32.18.8 **maxima:** система символьной математики **Maxima**

Глава 33

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 32. Сейчас активно развивается ветка **SDL2**, ее особенности кратко рассмотрены в разделе 33.4.

33.1 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

33.2 LazyFoo tutorial

1

33.2.1 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 копияста: 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](#).

33.2.2 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](#).

33.2.3 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

33.2.4 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.

- 33.2.5 Color Keying
- 33.2.6 Clip Blitting and Sprite Sheets
- 33.2.7 True Type Fonts
- 33.2.8 Key Presses
- 33.2.9 Mouse Events
- 33.2.10 Key States
- 33.2.11 Playing Sounds
- 33.2.12 Timing
- 33.2.13 Advanced Timers
- 33.2.14 Regulating Frame Rate
- 33.2.15 Calculating Frame Rate
- 33.2.16 Motion
- 33.2.17 Collision Detection
- 33.2.18 Per-pixel Collision Detection
- 33.2.19 Circular Collision Detection
- 33.2.20 Animation

33.3.1 Графический Hello World

user/sdl_hello.c

```
1 #include "SDL/SDL.h"
2
3 int main() {
4     // init SDL
5     if (SDL_Init(SDL_INIT_VIDEO)) { // SDL_INIT_EVERYTHING) {
6         fprintf(stderr, "\nSDL_GetError: %s\n", SDL_GetError());
7         abort();
8     }
9     // start window/fullscreen
10    SDL_Surface* screen = SDL_SetVideoMode( 640, 480, 0, SDL_SWSURFACE );
11    // hide mouse cursor
12    SDL_ShowCursor(0);
13    // load hello image
14    SDL_Surface* hello = SDL_LoadBMP( "/share/hello.bmp" );
15    // copy image to screen
16    SDL_BlitSurface( hello, NULL, screen, NULL );
17    // update sdl window
18    SDL_Flip( screen );
19    // delay 5 seconds
20    SDL_Delay( 5000 );
21    // quit
22    SDL_Quit();
23    return 0;
24 }
```


33.3.2 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 }

```

33.4 Версия SDL2

Глава 34

BuildRoot

Глава 35

Особенности OpenWrt

Часть XV

x86os: простая операционная система
для компьютера на i386 процессоре

Глава 36

Ресурсы

<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 37

Структура

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

Глава 38

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 39

Сборка кросс-компилятора

```
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 -std=c99 -Tx86os.ld
```

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) && bzipcat $< | 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) $(CFG_BINUTILS) &&\br/>19 ____$(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) &&\br/>30 ____cd $(TMP)/$(GCC) &&\br/>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 /32/](#).

Глава 40

Формат ELF32

Подробнее см. формат ELF 6.1 и скрипты линкера 6.3.1

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

kernel.objdump

```
1
2 kernel.elf:        file format elf32-i386
3 kernel.elf
4 architecture: i386, flags 0x00000112:
5 EXEC_P, HAS_SYMS, D_PAGED
6 start address 0x0010000c
7
8 Program Header:
9    LOAD off        0x00001000 vaddr 0x00100000 paddr 0x00100000 align 2**12
10        filesz 0x0000006f memsz 0x0000006f flags r-x
11    LOAD off        0x00002000 vaddr 0x00101000 paddr 0x00101000 align 2**12
12        filesz 0x00000000 memsz 0x00004000 flags rw-
13    LOAD off        0x00002000 vaddr 0x00105000 paddr 0x00105000 align 2**12
```

```

14      filesz 0x00000002 memsz 0x00000002 flags r—
15  LOAD off    0x00003000 vaddr 0x00106000 paddr 0x00106000 align 2**12
16      filesz 0x00000004 memsz 0x00001004 flags rw—
17  LOAD off    0x00003004 vaddr 0x00107004 paddr 0x00107004 align 2**12
18      filesz 0x00000089 memsz 0x00000089 flags r—

```

20 Sections:

Idx	Name	Size	VMA	LMA	File off	Algn
22	0 .text	0000006f	00100000	00100000	00001000	2**12
23		CONTENTS, ALLOC, LOAD, READONLY, CODE				
24	1 .stack	00004000	00101000	00101000	00002000	2**12
25		ALLOC				
26	2 .rodata	00000002	00105000	00105000	00002000	2**12
27		CONTENTS, ALLOC, LOAD, READONLY, DATA				
28	3 .data	00000004	00106000	00106000	00003000	2**12
29		CONTENTS, ALLOC, LOAD, DATA				
30	4 .bss	00000004	00107000	00107000	00003004	2**12
31		ALLOC				
32	5 .ignore	00000089	00107004	00107004	00003004	2**2
33		CONTENTS, ALLOC, LOAD, READONLY, DATA				

34 SYMBOL TABLE:

```

35 00100000 | d .text 00000000 .text
36 00101000 | d .stack 00000000 .stack
37 00105000 | d .rodata 00000000 .rodata
38 00106000 | d .data 00000000 .data
39 00107000 | d .bss 00000000 .bss
40 00107004 | d .ignore 00000000 .ignore
41 00000000 | df *ABS* 00000000 /tmp/ccOJlDa.o

```

```

42 1badb002 |      *ABS* 00000000 MAGIC
43 00000001 |      *ABS* 00000000 ALIGN
44 00000002 |      *ABS* 00000000 MEMINFO
45 00000003 |      *ABS* 00000000 FLAGS
46 e4524ffb |      *ABS* 00000000 CHECKSUM
47 00105000 |      .stack 00000000 stack_top
48 00100018 |      .text 00000000 .halt
49 00000000 | df *ABS* 00000000 kernel.c
50 00000000 | df *ABS* 00000000 vga.c
51 00000000 | df *ABS* 00000000 com.c
52 00000000 | df *ABS* 00000000 lpt.c
53 00000000 | df *ABS* 00000000 kbd.c
54 00000000 | df *ABS* 00000000 ide.c
55 00000000 | df *ABS* 00000000 fatfs.c
56 00000000 | df *ABS* 00000000 timer.c
57 00000000 | df *ABS* 00000000 user.c
58 00000000 | df *ABS* 00000000
59 00107000 g   O .bss 00000004 vga_ptr
60 0010002d g   F .text 0000003d vga_init
61 0010006a g   F .text 00000005 user
62 00105001 g   O .rodata 00000001 vga_rows
63 0010001b g   F .text 00000012 __start
64 0010000c g   .text 00000000 __multiboot
65 00106000 g   O .data 00000004 vga_buf
66 00105000 g   O .rodata 00000001 vga_cols

```

```

70 Disassembly of section .text:
71
72 00100000 <_multiboot-0xc>:
73   100000:  02 b0 ad 1b 03 00      add     0x31bad(%eax),%dh
74   100006:  00 00                  add     %al,(%eax)
75   100008:  fb                      sti
76   100009:  4f                      dec     %edi
77   10000a:  52                      push    %edx
78   10000b:  e4 fa                  in      $0xfa,%al
79
80 0010000c <_multiboot>:
81   10000c:  fa                      cli
82   10000d:  bc 00 50 10 00        mov     $0x105000,%esp
83   100012:  e8 04 00 00 00        call    10001b <_start>
84   100017:  fa                      cli
85
86 00100018 <.halt>:
87   100018:  f4                      hlt
88   100019:  eb fd                  jmp     100018 <.halt>
89
90 0010001b <_start>:
91   10001b:  55                      push    %ebp
92   10001c:  89 e5                  mov     %esp,%ebp
93   10001e:  83 ec 08              sub     $0x8,%esp
94   100021:  e8 07 00 00 00        call    10002d <vga_init>
95   100026:  e8 3f 00 00 00        call    10006a <user>
96   10002b:  c9                      leave
97   10002c:  c3                      ret

```

```

98
99 0010002d <vga_init>:
00 10002d: 55          push    %ebp
01 10002e: 89 e5      mov     %esp,%ebp
02 100030: 83 ec 10   sub     $0x10,%esp
03 100033: c7 05 00 70 10 00 00  movl    $0x0,0x107000
04 10003a: 00 00 00
05 10003d: c7 45 fc 00 00 00 00  movl    $0x0,-0x4(%ebp)
06 100044: eb 1c      jmp     100062 <vga_init+0x35>
07 100046: 8b 0d 00 60 10 00   mov     0x106000,%ecx
08 10004c: a1 00 70 10 00     mov     0x107000,%eax
09 100051: 8d 50 01     lea     0x1(%eax),%edx
10 100054: 89 15 00 70 10 00   mov     %edx,0x107000
11 10005a: 01 c8      add     %ecx,%eax
12 10005c: c6 00 ff     movb    $0xff,(%eax)
13 10005f: ff 45 fc     incl    -0x4(%ebp)
14 100062: 83 7d fc 77   cmpl    $0x77,-0x4(%ebp)
15 100066: 7e de      jle     100046 <vga_init+0x19>
16 100068: c9         leave
17 100069: c3         ret
18
19 0010006a <user>:
20 10006a: 55          push    %ebp
21 10006b: 89 e5      mov     %esp,%ebp
22 10006d: 5d         pop     %ebp
23 10006e: c3         ret

```

Глава 41

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 .ignore : { *(.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 42

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) $(USER) cfg.mk x86os.ld  
7 ____$(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     user();
8 }
```

Глава 43

Драйвера

43.1 **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 enum vga_color
10 {
11     COLOR_BLACK = 0,
```

```
12  COLOR_BLUE = 1 ,
13  COLOR_GREEN = 2 ,
14  COLOR_CYAN = 3 ,
15  COLOR_RED = 4 ,
16  COLOR_MAGENTA = 5 ,
17  COLOR_BROWN = 6 ,
18  COLOR_LIGHT_GREY = 7 ,
19  COLOR_DARK_GREY = 8 ,
20  COLOR_LIGHT_BLUE = 9 ,
21  COLOR_LIGHT_GREEN = 10 ,
22  COLOR_LIGHT_CYAN = 11 ,
23  COLOR_LIGHT_RED = 12 ,
24  COLOR_LIGHT_MAGENTA = 13 ,
25  COLOR_LIGHT_BROWN = 14 ,
26  COLOR_WHITE = 15 ,
27 };
28
29 void vga_init ();
30
31 #endif
```

driver/vga.c

```
1 // Std.VGA 80x25 text console driver
2
3 #include <driver/vga.h>
4
5 const unsigned char vga_cols = VGA_COLS;
6 const unsigned char vga_rows = VGA_ROWS;
```

```
7
8 unsigned char *vga_buf = (unsigned char *) (VGA_ADDR);
9 unsigned int vga_ptr = 0;
10
11 void vga_init() {
12     vga_ptr = 0;
13     for (int i = 0; i < VGA_COLS * 3 / 2; i++)
14         vga_buf[vga_ptr++] = 0xFF;
15 }
```

43.2 **kbd**: клавиатура

43.3 **ide**: жесткий диск IDE

43.3.1 **fatfs**: файловая система FAT16

Часть XVI

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 $\rightarrow 0$.

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 на навыки **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 44

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 копия: <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

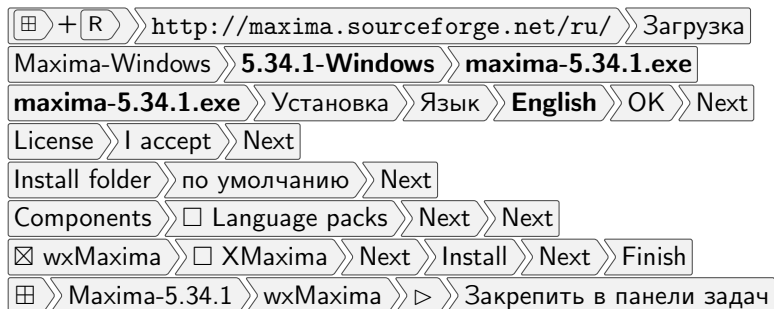
и для ВУЗа, и для студентов и преподавателей иметь в своем распоряжении легальные копии такого ПО без больших, и даже каких-либо денежных затрат.

Глава 45

Пакет Maxima

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

45.1 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>.

45.2 Калькулятор

Часть XVII

Подготовка технической документации

Глава 46

Верстка в \LaTeX

Глава 47

Оформление листингов

Глава 48

Подготовка иллюстраций

48.1 GIMP

48.2 Inkscape

48.3 Graphviz

Глава 49

Замечания для соавторов “Абзуки
ARMатурщика”

Часть XVIII

Примерные учебные планы

Глава 50

Блондинко

Глава 51

Школотрон

1. Введение в практическую электронику: Вводная электронная схема

постараться понять разделы

Ток и напряжение, сечение проводника, плотность тока

Проводники и изоляторы, сопротивление и проводимость

Диэлектрическая и тепловая прочность изоляции

Масштабные множители

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

2. Введение в практическую электронику: Использование мультиметра

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

Глава 52

Студень

1. Введение в практическую электронику

Глава 53

Технический специалист

1. Введение в практическую электронику прочитать по диагонали

Часть XIX

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 54

Автоматное программирование
/фреймворк QuantumLeaps/

Глава 55

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 копияста: <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 you phone there's also a risk you render it useless, becoming the proud owner of an expensive brick. This solution is primary 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. Partition 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 aliment 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 and 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 (gemu-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 (vold 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 in 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 make 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 backedup 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 Androids `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 Androids 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.

ps tree

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+, 189

syslinux, 189

термины

), 171

Ампер, 17

БМП, 14

ГМ, 16

УЗО, 22

Вольт, 17

автомат, 22

бэкенд компилятора, 171

диэлектрическая прочность, 20

диэлектрик, 19

диод, 16

допустимый рабочий ток, 15

электрический ток, 16

элемент, 14

фронтенд компилятора, 171

гидравлическая модель, 16

грамматика, 164

интенсивность, 28

источник питания, 16

изолятор, 19

канадский крест, 241

компонент, 14

короткое замыкание, 21

кросс-компиляция, 188

кросс-тулчейн, 188

лампа

галогенная, 27

накаливания, 27

лексема, 163

лексер, 163

лексический анализ, 164

лексический анализатор, 165

макетная плата, 14

масштабные множители, 22

мощность, 22
напряжение, 16, 17
номинал, 26
носители заряда, 31
обратное напряжение диода, 38
общий провод, 14, 17
падстек, 44, 69
пакет, 197
пиковая длина волны, 29
площадь сечения проводника, 18
плотность тока, 18
полевой транзистор, 32
предохранитель, 22
предварительный усилитель, 33
промежуточное представление, 171
проводимость, 18
проводник, 16, 19
прямое напряжение
 светодиода, 28
прямой ток
 светодиода, 28
прямой ток диода, 38
разность потенциалов, 17
разводка печатной платы, 42
регулярное выражение, 163
резистор, 15, 16, 20
сборка, 188

сборка пакета, 197
сечение проводника, 17
сегмент, 60, 287
секция ELF, 60, 287
семантический анализ, 172
сила тока, 17
синтаксический анализ, 167
сканер, 163
сопротивление, 19, 20
светодиод, 15
шина питания, 14
ток, 16, 17
ток короткого замыкания, 22
токен, 163
транзистор, 32
трассировка печатной платы, 42
учебный план, 3
угол излучения, 28
усиление, 32
устройство защитного отключения, 22
вафля, 14
выгорание проводки, 22
выходная цепь, 35
вывод элемента, 20
выводной корпус, 26
входная цепь, 35
входное сопротивление, 33

- входной ток, 33
- закон Джоуля Ленца, 22
- закон Ома
 - для постоянного тока, 19
- замкнутый контур, 15
- земля, 14
- breadboard, 14
- Data Definition Language, 162
- DDL, 162
- ELF, 287
- emLinux, 188
- FET, 32
- multiboot, 293
- plain text, 162
- USB
 - хост-контроллер, 183
 - On-The-Go, 183
 - OTG, 183
 - URB, 183
- VCS, 54

azLinux

- настройка
 - приложение, 207
- пакет
 - dirs, 203
- переменная

- APP, 207
- BOOT, 204
- BUILD, 204
- DIRS, 204
- GZ, 203
- ROOT, 204
- SRC, 203
- TC, 204
- TMP, 204
- приложение
 - clock, 207
 - micro, 207
- скрипт
 - mk.rc, 199
- железо, 207
 - AR7240, 214
 - ASUS Eee PC 701, 209
 - BlackSwift, 212
 - Celeron1037U, 214
 - CeleronM, 213
 - Gigabyte GA-C1037UN-EU, 211
 - i386, 209
 - i486sx, 213
 - QEMU, 209, 212
 - RT5350, 214
 - VoCore, 212