

Азбука АРМАТУРЩИКА

# Основы бытовой автоматики, электроники и систем управления



Консорциум хоббитов России  
HackSpace «Чебураторный завод»  
Дмитрий Понятов <[dponyatov@gmail.com](mailto:dponyatov@gmail.com)>  
Bill Collis (??)  
Andreas Fester (V)  
Joe Martin, Craig Libuse (95)

# Оглавление

О книге	46
<b>1 1. Introduction to Practical Electronics</b>	
<b>Введение в практическую электронику</b>	48
<b>От переводчика</b> . . . . .	49
1.1 1.1 Your learning in Technology	
<b>Ваше обучение по специальности «Технология»</b> . . . . .	52
1.1.1 Technology Achievement Objectives from the NZ Curriculum	
<b>Цели обучения технологиям Ново-Зеландской программы</b> . . . . .	52
1.2 1.2 Key Competencies from The NZ Curriculum	
<b>Ключевые компетенции Ново-Зеландской программы</b> . . . . .	53
<b>2 2 An introductory electronic circuit</b>	57
2.1 2.1 Where to buy stuff? . . . . .	58
2.2 2.2 Identifying resistors by their colour codes . . . . .	58
2.3 2.3 LED's . . . . .	58
2.4 2.4 Some LED Specifications . . . . .	58

2.5	2.5 LED research task . . . . .	58
2.6	2.6 Adding a switch to your circuit . . . . .	58
2.7	2.7 Switch assignment . . . . .	58
2.8	2.8 Important circuit concepts . . . . .	58
2.9	2.9 Changing the value of resistance . . . . .	58
2.10	2.10 Adding a transistor to your circuit . . . . .	58
2.11	2.11 Understanding circuits . . . . .	58
2.12	2.12 The input circuit — an LDR . . . . .	58
2.13	2.13 Working darkness detector circuit . . . . .	58
2.14	2.14 Protecting circuits — using a diode . . . . .	58
2.15	2.15 Diode Research Task . . . . .	58
2.16	2.16 Final darkness detector circuit . . . . .	58
<b>3</b>	<b>3 Introductory PCB construction</b>	<b>59</b>
3.1	3.1 Eagle Schematic and Layout Editor Tutorial . . . . .	59
3.2	3.2 An Introduction to Eagle . . . . .	59
3.3	3.3 The Schematic Editor . . . . .	59
3.4	3.4 The Board Editor . . . . .	59
3.5	3.5 Making Negative Printouts . . . . .	59
3.6	3.6 PCB Making . . . . .	59
<b>4</b>	<b>Soldering, solder and soldering irons</b>	<b>60</b>
4.1	4.1 Soldering facts . . . . .	61
4.2	4.2 Soldering Safety . . . . .	61
4.3	4.3 Soldering wires to switches . . . . .	61
4.4	4.4 Codes of practice . . . . .	61
4.5	4.5 Good and bad solder joints . . . . .	61

4.6	4.6 Short circuits . . . . .	61
4.7	4.7 Soldering wires to LED's . . . . .	61
<b>5</b>	<b>5 Introductory Electronics Theory</b>	<b>62</b>
5.1	5.1 Making electricity . . . . .	63
5.2	5.2 ESD electrostatic discharge . . . . .	63
5.3	5.3 Magnets, wires and motion . . . . .	63
5.4	5.4 Group Power Assignment . . . . .	63
5.5	5.5 Electricity supply in New Zealand . . . . .	63
5.6	5.6 Conductors . . . . .	63
5.7	5.7 Insulators . . . . .	63
5.8	5.8 Choosing the right wire . . . . .	63
5.9	5.9 Resistors . . . . .	63
5.10	5.10 Resistor Assignment . . . . .	63
5.11	5.11 Resistivity . . . . .	63
5.12	5.12 Resistor prefixes . . . . .	63
5.13	5.13 Resistor Values Exercises . . . . .	63
5.14	5.14 Capacitors . . . . .	63
5.15	5.15 Component symbols reference . . . . .	63
5.16	5.16 Year 10/11 — Typical test questions so far . . . . .	63
<b>6</b>	<b>Introduction to microcontroller electronics</b>	<b>64</b>
6.1	6.1 What is a computer? . . . . .	65
6.2	6.2 What does a computer system do? . . . . .	65
6.3	6.3 What does a microcontroller system do? . . . . .	65
6.4	6.4 What exactly is a microcontroller? . . . . .	65
6.5	6.5 Getting started with AVR Programming . . . . .	65

6.6	6.6 Breadboard . . . . .	65
6.7	6.7 Breadboard+Prototyping board circuit . . . . .	65
6.8	6.8 Checking your workmanship . . . . .	65
6.9	6.9 Getting started with Bascom & AVR . . . . .	65
6.10	6.10 The compiler . . . . .	65
6.11	6.11 The programmer . . . . .	65
6.12	6.12 USBASP programming cable . . . . .	65
6.13	6.13 Your first circuit . . . . .	65
6.14	6.14 An introduction to flowcharts . . . . .	65
6.15	6.15 Bascom output commands . . . . .	65
6.16	6.16 Exercises . . . . .	65
6.17	6.17 Two delays . . . . .	65
6.18	6.18 Syntax errors -'bugs' . . . . .	65
6.19	6.19 Microcontroller ports: write a Knightrider program using LED's . . . . .	65
6.20	6.20 Knightrider v2 . . . . .	65
6.21	6.21 Knightrider v3 . . . . .	65
6.22	6.22 Commenting your programs . . . . .	65
6.23	6.23 Learning review . . . . .	65
6.24	6.24 What is a piezo and how does it make sound? . . . . .	65
6.25	6.25 Sounding Off . . . . .	65
6.26	6.26 Sound exercises . . . . .	65
6.27	6.27 Amp it up . . . . .	65
<b>7</b>	<b>7 Microcontroller input circuits</b> . . . . .	<b>66</b>
7.1	7.1 Single push button switch . . . . .	67
7.2	7.2 Pullup resistor theory . . . . .	67
7.3	7.3 Switch in a breadboard circuit . . . . .	67

7.4	7.4 Checking switches in your program . . . . .	67
7.5	7.5 Program Logic – the ‘If-Then’ Switch Test . . . . .	67
7.6	7.6 If-then exercises . . . . .	67
7.7	7.7 Switch contact bounce . . . . .	67
7.8	7.8 Reading multiple switches . . . . .	67
7.9	7.9 Bascom debounce command . . . . .	67
7.10	7.10 Different types of switches you can use . . . . .	67
7.11	7.11 Reflective opto switch . . . . .	67
<b>8</b>	<b>8 Programming Review</b>	<b>68</b>
8.1	8.1 Three steps to help you write good programs . . . . .	69
8.2	8.2 Saving Programs . . . . .	69
8.3	8.3 Organisation is everything . . . . .	69
8.4	8.4 Programming template . . . . .	69
8.5	8.5 What you do when learning to program . . . . .	69
8.6	8.6 AVR microcontroller hardware . . . . .	69
8.7	8.7 Power supplies . . . . .	69
8.8	8.8 Programming words you need to be able to use correctly . . . . .	69
8.9	8.9 Year10/11 typical test questions so far . . . . .	69
<b>9</b>	<b>9 Introduction to program flow</b>	<b>70</b>
9.1	9.1 Pedestrian crossing lights controller . . . . .	71
9.2	9.2 Pedestrian Crossing Lights schematic . . . . .	71
9.3	9.3 Pedestrian Crossing Lights PCB Layout . . . . .	71
9.4	9.4 Algorithm planning example – pedestrian crossing lights . . . . .	71
9.5	9.5 Flowchart planning example – pedestrian crossing lights . . . . .	71
9.6	9.6 Getting started code . . . . .	71

9.7	9.7 Modification exercise for the pedestrian crossing . . . . .	71
9.8	9.8 Traffic lights program flow . . . . .	71
<b>10</b>	<b>10 Introductory programming - using subroutines</b>	<b>72</b>
10.1	10.1 Sending Morse code . . . . .	72
10.2	10.2 LM386 audio amplifier PCB . . . . .	72
10.3	10.3 LM386 PCB Layout . . . . .	72
<b>11</b>	<b>11 Introductory programming – using variables</b>	<b>73</b>
11.1	11.1 Stepping or counting using variables . . . . .	74
11.2	11.2 For-Next . . . . .	74
11.3	11.3 Siren sound - programming using variables . . . . .	74
11.4	11.4 Make a simple siren . . . . .	74
11.5	11.5 Siren exercise . . . . .	74
11.6	11.6 A note about layout of program code . . . . .	74
11.7	11.7 Using variables for data . . . . .	74
11.8	11.8 Different types of variables . . . . .	74
11.9	11.9 Variables and their uses . . . . .	74
11.10	11.10 Vehicle counter . . . . .	74
11.11	11.11 Rules about variables . . . . .	74
11.12	11.12 Examples of variables in use . . . . .	74
11.13	11.13 Byte variable limitations . . . . .	74
11.14	11.14 Random Numbers . . . . .	74
11.15	11.15 The Bascom-AVR simulator . . . . .	74
11.16	11.16 Electronic dice project . . . . .	74
11.17	11.17 Programming using variables – dice . . . . .	74
11.18	11.18 Dice layout stage 1 . . . . .	74

11.19	11.19 Dice layout stage 2 . . . . .	74
11.20	11.20 Dice Layout final . . . . .	74
11.21	11.21 First Dice Program flowchart . . . . .	74
11.22	11.22 A note about the Bascom Rnd command . . . . .	74
11.23	11.23 Modified dice . . . . .	74
11.24	11.24 Modified Knightrider . . . . .	74
<b>12</b>	<b>12 Basic displays</b>	<b>75</b>
12.1	12.1 7 segment displays . . . . .	75
12.2	12.2 Alphanumeric LED displays . . . . .	75
<b>13</b>	<b>13 TDA2822M Portable Audio Amplifier Project</b>	<b>76</b>
13.1	13.1 Portfolio Assessment Schedule . . . . .	77
13.2	13.2 Initial One Page Brief . . . . .	77
13.3	13.3 TDA2822M specifications . . . . .	77
13.4	13.4 Making a PCB for the TDA2822 Amp Project . . . . .	77
13.5	13.5 Extra PCB making information . . . . .	77
13.6	13.6 Component Forming Codes of Practice . . . . .	77
13.7	13.7 TDA2811 wiring diagram . . . . .	77
13.8	13.8 SKETCHUP Quick Start Tutorial . . . . .	77
13.9	13.9 Creating reusable components in SketchUp . . . . .	77
<b>14</b>	<b>14 Basic programming logic</b>	<b>78</b>
14.1	14.1 Quiz Game Controller . . . . .	79
14.2	14.2 Quiz game controller system context diagram . . . . .	79
14.3	14.3 Quiz game controller block diagram . . . . .	79
14.4	14.4 Quiz game controller Algorithm . . . . .	79

14.5	14.5 Quiz game schematic . . . . .	79
14.6	14.6 Quiz game board veroboard layout . . . . .	79
14.7	14.7 Quiz Controller flowchart . . . . .	79
14.8	14.8 'Quiz Controller program code . . . . .	79
14.9	14.9 Don't delay - use logic . . . . .	79
<b>15</b>	<b>15 Algorithm development – an alarm system</b>	<b>80</b>
15.1	15.1 Simple alarm system – stage 1 . . . . .	81
15.2	15.2 Alarm System Schematic . . . . .	81
15.3	15.3 A simple alarm system – stage 2 . . . . .	81
15.4	15.4 A simple alarm system – stage 3 . . . . .	81
15.5	15.5 A simple alarm system – stage 4 . . . . .	81
15.6	15.6 More complex alarm system . . . . .	81
15.7	15.7 Alarm unit algorithm 5 . . . . .	81
15.8	15.8 Alarm 6 algorithm . . . . .	81
<b>16</b>	<b>16 Basic DC circuit theory</b>	<b>82</b>
16.1	16.1 Conventional Current . . . . .	83
16.2	16.2 Ground . . . . .	83
16.3	16.3 Preferred resistor values . . . . .	83
16.4	16.4 Resistor Tolerances . . . . .	83
16.5	16.5 Combining resistors in series . . . . .	83
16.6	16.6 Combining resistors in parallel . . . . .	83
16.7	16.7 Resistor Combination Circuits . . . . .	83
16.8	16.8 Multimeters . . . . .	83
16.9	16.9 Multimeter controls . . . . .	83
16.10	16.10 Choosing correct meter settings . . . . .	83

16.11	16.11 Ohms law . . . . .	83
16.12	16.12 Voltage & Current Measurements . . . . .	83
16.13	16.13 . . . . .	83
16.14	16.14 Continuity . . . . .	83
16.15	16.15 Variable Resistors . . . . .	83
16.16	16.16 Capacitors . . . . .	83
16.17	16.17 Capacitor Codes and Values . . . . .	83
16.18	16.18 Converting Capacitor Values uF, nF , pF . . . . .	83
16.19	16.19 Capacitor action in DC circuits . . . . .	83
16.20	16.20 The Voltage Divider . . . . .	83
16.21	16.21 Using semiconductors . . . . .	83
16.22	16.22 Calculating current limit resistors for an LED . . . . .	83
16.23	16.23 The Bipolar Junction Transistor . . . . .	83
16.24	16.24 Transistor Specifications Assignment . . . . .	83
16.25	16.25 Transistor Case styles . . . . .	83
16.26	16.26 Transistor amplifier in a microcontroller circuit . . . . .	83
16.27	16.27 Transistor Audio Amplifier . . . . .	83
16.28	16.28 Speakers . . . . .	83
16.29	16.29 Switch types and symbols . . . . .	83
<b>17</b>	<b>17 Basic project planning</b> . . . . .	<b>84</b>
17.1	17.1 System Designer . . . . .	85
17.2	17.2 Project mind map . . . . .	85
17.3	17.3 Project timeline . . . . .	85
17.4	17.4 System context diagram . . . . .	85
17.5	17.5 Block Diagram . . . . .	85
17.6	17.6 Board Layouts . . . . .	85

17.7	17.7 Algorithm design . . . . .	85
17.8	17.8 Flowcharts . . . . .	85
<b>18</b>	<b>18 Example system design - hot glue gun timer</b>	<b>86</b>
18.1	18.1 System context diagram . . . . .	86
18.2	18.2 Hot glue gun timer block diagram . . . . .	86
18.3	18.3 Hot glue gun timer algorithm . . . . .	86
18.4	18.4 Hot glue gun timer flowchart . . . . .	86
18.5	18.5 Hot glue gun timer program . . . . .	86
<b>19</b>	<b>19 Basic interfaces and their programming</b>	<b>87</b>
19.1	19.1 Parallel data communications . . . . .	88
19.2	19.2 LCDs (liquid crystal displays) . . . . .	88
19.3	19.3 Alphanumeric LCDs . . . . .	88
19.4	19.4 ATTINY461 Development PCB with LCD . . . . .	88
19.5	19.5 Completing the wiring for the LCD . . . . .	88
19.6	19.6 LCD Contrast Control . . . . .	88
19.7	19.7 Learning to use the LCD . . . . .	88
19.8	19.8 Repetition again - the 'For-Next' and the LCD . . . . .	88
19.9	19.9 LCD Exercises . . . . .	88
19.10	19.10 Defining your own LCD characters . . . . .	88
19.11	19.11 LCD custom character program . . . . .	88
19.12	19.12 A simple digital clock . . . . .	88
19.13	19.13 Adding more interfaces to the ATTiny461 Development board . . . . .	88
19.14	19.14 Ohms law in action – a multicoloured LED . . . . .	88

<b>20</b>	<b>20 Basic analog to digital interfaces</b>	<b>89</b>
20.1	20.1 ADC - Analog to Digital conversion . . . . .	90
20.2	20.2 Light level sensing . . . . .	90
20.3	20.3 Voltage dividers review . . . . .	90
20.4	20.4 AVR ADC connections . . . . .	90
20.5	20.5 Select-Case . . . . .	90
20.6	20.6 Reading an LDR's values . . . . .	90
20.7	20.7 Marcus' year10 night light project . . . . .	90
20.8	20.8 Temperature measurement using the LM35 . . . . .	90
20.9	20.9 A simple temperature display . . . . .	90
20.10	20.10 LM35 temperature display . . . . .	90
20.11	20.11 Force Sensitive Resistors . . . . .	90
20.12	20.12 Piezo sensor . . . . .	90
20.13	20.13 Multiple switches and ADC . . . . .	90
<b>21</b>	<b>21 Basic System Design</b>	<b>91</b>
21.1	21.1 Understanding how systems are put together . . . . .	92
21.2	21.2 Food Processor system block diagram . . . . .	92
21.3	21.3 Subsystems . . . . .	92
21.4	21.4 Food Processor system functional attributes - algorithm . . . . .	92
21.5	21.5 Food Processor system flowchart . . . . .	92
21.6	21.6 Toaster Design . . . . .	92
21.7	21.7 Toaster - system block diagram . . . . .	92
21.8	21.8 Toaster Algortihm . . . . .	92
<b>22</b>	<b>22 Basic System development - Time Tracker</b>	<b>93</b>
22.1	22.1 System context diagram and brief . . . . .	94

22.2	22.2 Time tracker block diagram . . . . .	94
22.3	22.3 Algorithm development . . . . .	94
22.4	22.4 Schematic . . . . .	94
22.5	22.5 Time tracker flowchart and program version 1 . . . . .	94
22.6	22.6 Time Tracker stage 2 . . . . .	94
22.7	22.7 Time Tracker stage 3 . . . . .	94
22.8	22.8 Time Tracker stage 4 . . . . .	94
<b>23</b>	<b>23 Basic maths time</b>	<b>95</b>
23.1	23.1 Ohms law calculator . . . . .	95
23.2	23.2 more maths - multiplication . . . . .	95
23.3	23.3 Algorithms for multiplication of very large numbers . . . . .	95
23.4	23.4 Program ideas - algorithm and flowchart exercises . . . . .	95
<b>24</b>	<b>24 Basic string variables</b>	<b>96</b>
24.1	24.1 Strings assignment . . . . .	96
24.2	24.2 ASCII Assignment . . . . .	96
24.3	24.3 Time in a string . . . . .	96
24.4	24.4 Date in a string . . . . .	96
24.5	24.5 Scrolling message assignment . . . . .	96
24.6	24.6 Some LCD programming exercises . . . . .	96
<b>25</b>	<b>25 Advanced power interfaces</b>	<b>97</b>
25.1	25.1 Microcontroller power limitations . . . . .	98
25.2	25.2 Power . . . . .	98
25.3	25.3 Power dissipation in resistors . . . . .	98
25.4	25.4 Diode characteristics . . . . .	98

25.5	25.5 Using Zener diodes . . . . .	98
25.6	25.6 How diodes work . . . . .	98
25.7	25.7 How does a LED give off light? . . . . .	98
25.8	25.8 LCD Backlight Data . . . . .	98
25.9	25.9 Transistors as power switches . . . . .	98
25.10	25.10 High power loads . . . . .	98
25.11	25.11 AVR Power matters . . . . .	98
25.12	25.12 Darlington transistors - high power . . . . .	98
25.13	25.13 ULN2803 Octal Darlington Driver . . . . .	98
25.14	25.14 Connecting a FET backlight control to your microcontroller . . . . .	98
25.15	25.15 FET backlight control . . . . .	98

## **26 26 Advanced Power Supply Theory** 99

26.1	26.1 Typical PSUs . . . . .	100
26.2	26.2 The four stages of a PSU (power supply unit) . . . . .	100
26.3	26.3 Stage 1: step down transformer . . . . .	100
26.4	26.4 Stage 2: AC to DC Conversion . . . . .	100
26.5	26.5 Stage 3: Filtering AC component . . . . .	100
26.6	26.6 Stage 4: Voltage Regulation . . . . .	100
26.7	26.7 Ripple (decibel & dB) . . . . .	100
26.8	26.8 Line Regulation . . . . .	100
26.9	26.9 Load Regulation . . . . .	100
26.10	26.10 Current Limit . . . . .	100
26.11	26.11 Power, temperature and heatsinking . . . . .	100
26.12	26.12 Typical PSU circuit designs . . . . .	100
26.13	26.13 PSU block diagram . . . . .	100
26.14	26.14 PSU Schematic . . . . .	100

26.15	26.15 Practical current limit circuit . . . . .	100
26.16	26.16 Voltage measurement using a voltage divider . . . . .	100
26.17	26.17 Variable power supply voltmeter program . . . . .	100
<b>27</b>	<b>27 Year11/12/13 typical test questions so far</b>	<b>101</b>
<b>28</b>	<b>28 Advanced programming -arrays</b>	<b>102</b>
<b>29</b>	<b>29 AVR pull-up resistors</b>	<b>103</b>
<b>30</b>	<b>30 AVR pull-up resistors</b>	<b>104</b>
<b>31</b>	<b>31 Advanced programming - state machines</b>	<b>105</b>
31.1	31.1 Daily routine state machine . . . . .	106
31.2	31.2 Truck driving state machine . . . . .	106
31.3	31.3 Developing a state machine . . . . .	106
31.4	31.4 A state machine for the temperature alarm system . . . . .	106
31.5	31.5 Using System Designer software to design state machines . . . . .	106
31.6	31.6 State machine to program code . . . . .	106
31.7	31.7 The power of state machines over flowcharts . . . . .	106
31.8	31.8 Bike light – state machine example . . . . .	106
31.9	31.9 Bike light program version 1b . . . . .	106
31.10	31.10 Bike light program version 2 . . . . .	106
<b>32</b>	<b>32 Advanced keypad interfacing</b>	<b>107</b>
32.1	32.1 Keypad program 1 . . . . .	107
32.2	32.2 Keypad program 2 . . . . .	107
32.3	32.3 Keypad program 3 – cursor control . . . . .	107

32.4	32.4 Keypad texter program V1 . . . . .	107
32.5	32.5 Keypad texter program 1a . . . . .	107
32.6	32.6 ADC keypad interface . . . . .	107
<b>33</b>	<b>33 Do-Loop &amp; While-Wend subtleties</b>	<b>108</b>
33.1	33.1 While-Wend or Do-Loop-Until or For-Next? . . . . .	108
<b>34</b>	<b>34 DC Motor interfacing</b>	<b>109</b>
34.1	34.1 H-Bridge . . . . .	110
34.2	34.2 H-Bridge Braking . . . . .	110
34.3	34.3 L293D H-Bridge IC . . . . .	110
34.4	34.4 L298 H-Bridge IC . . . . .	110
34.5	34.5 LMD18200 H-Bridge IC . . . . .	110
34.6	34.6 LMD18200 program . . . . .	110
34.7	34.7 Darlington H-Bridge . . . . .	110
34.8	34.8 Stepper motors . . . . .	110
34.9	34.9 PWM - pulse width modulation . . . . .	110
34.10	34.10 PWM outputs . . . . .	110
34.11	34.11 Uses for PWM . . . . .	110
34.12	34.12 ATMEL AVR's PWM pins . . . . .	110
34.13	34.13 PWM on any port . . . . .	110
34.14	34.14 PWM internals . . . . .	110
<b>35</b>	<b>35 Advanced System Example – Alarm Clock</b>	<b>111</b>
35.1	. . . . .	111
35.2	35.2 Analogue seconds display on an LCD . . . . .	111
35.3	35.3 LCD big digits . . . . .	111

<b>36</b>	<b>36 Resistive touch screen</b>	<b>112</b>
36.1	36.1 Keeping control so you dont lose your 'stack' . . . . .	112
<b>37</b>	<b>37 System Design Example – Temperature Controller</b>	<b>113</b>
<b>38</b>	<b>38 Alarm clock project re-developed</b>	<b>114</b>
38.1	38.1 System Designer to develop a Product Brainstorm . . . . .	114
38.2	38.2 Initial block diagram for the alarm clock . . . . .	114
38.3	38.3 A first (simple) algorithm is developed . . . . .	114
38.4	38.4 A statemachine for the first clock . . . . .	114
38.5	38.5 Alarm clock state machine and code version 2 . . . . .	114
38.6	38.6 Token game – state machine design example . . . . .	114
<b>39</b>	<b>39 Advanced window controller student project</b>	<b>115</b>
39.1	39.1 Window controller state machine #1 . . . . .	115
39.2	39.2 Window controller state machine #3 . . . . .	115
39.3	39.3 Window controller state machine #5 . . . . .	115
39.4	39.4 Window controller program . . . . .	115
<b>40</b>	<b>40 Alternative state machine coding techniques</b>	<b>116</b>
<b>41</b>	<b>41 Complex — serial communications</b>	<b>117</b>
41.1	41.1 Simplex and duplex . . . . .	118
41.2	41.2 Synchronous and asynchronous . . . . .	118
41.3	41.3 Serial communications, Bascom and the AVR . . . . .	118
41.4	41.4 RS232 serial communications . . . . .	118
41.5	41.5 Build your own RS232 buffer . . . . .	118
41.6	41.6 Talking to an AVR from Windows XP . . . . .	118

41.7	41.7 Talking to an AVR from Win7 . . . . .	118
41.8	41.8 First Bascom RS-232 program . . . . .	118
41.9	41.9 Receiving text from a PC . . . . .	118
41.10	41.10 BASCOM serial commands . . . . .	118
41.11	41.11 Serial IO using Inkey() . . . . .	118
41.12	41.12 Creating your own software to communicate with the AVR . . . . .	118
41.13	41.13 Microsoft Visual Basic 2008 Express Edition . . . . .	118
41.14	41.14 Stage 1 — GUI creation . . . . .	118
41.15	41.15 Stage 2 — Coding and understanding event programming . . . . .	118
41.16	41.16 Microsoft Visual C# commport application . . . . .	118
41.17	41.17 Microcontroller with serial IO . . . . .	118
41.18	41.18 PC software (C#) to communicate with the AVR . . . . .	118
41.19	41.19 Using excel to capture serial data . . . . .	118
41.20	41.20 PLX-DAQ . . . . .	118
41.21	41.21 StampPlot . . . . .	118
41.22	41.22 Serial to parallel . . . . .	118
41.23	41.23 Keyboard interfacing — synchronous serial data . . . . .	118
41.24	41.24 Keyboard as asynchronous data . . . . .	118
41.25	41.25 GPS . . . . .	118

<b>42</b>	<b>42 Radio Data Communication</b> . . . . .	<b>119</b>
42.1	42.1 An Introduction to data over radio . . . . .	120
42.2	42.2 HT12E Datasheet, transmission and timing . . . . .	120
42.3	42.3 HT12 test setup . . . . .	120
42.4	42.4 HT12E Program . . . . .	120
42.5	42.5 HT12D datasheet . . . . .	120
42.6	42.6 HT12D Program . . . . .	120

42.7	42.7 Replacing the HT12E encoding with software . . . . .	120
<b>43</b>	<b>43 Introduction to I2C</b>	<b>121</b>
43.1	43.1 I2C Real Time Clocks . . . . .	122
43.2	43.2 Real time clocks . . . . .	122
43.3	43.3 Connecting the RTC . . . . .	122
43.4	43.4 Connecting the RTC to the board . . . . .	122
43.5	43.5 Internal features . . . . .	122
43.6	43.6 DS1307 RTC code . . . . .	122
43.7	43.7 DS1678 RTC code . . . . .	122
<b>44</b>	<b>44 Plant watering timer student project</b>	<b>123</b>
44.1	44.1 System block diagram . . . . .	123
44.2	44.2 State machine . . . . .	123
44.3	44.3 Program code . . . . .	123
<b>45</b>	<b>45 Bike audio amplifier project</b>	<b>124</b>
<b>46</b>	<b>46 Graphics LCDs</b>	<b>125</b>
46.1	46.1 The T6963 controller . . . . .	125
46.2	46.2 Graphics LCD (128x64) — KS0108 . . . . .	125
46.3	46.3 Generating a negative supply for a graphics LCD . . . . .	125
<b>47</b>	<b>47 GLCD Temperature Tracking Project</b>	<b>126</b>
47.1	47.1 Project hardware . . . . .	127
47.2	47.2 Project software planning . . . . .	127
47.3	47.3 Draw the graph scales . . . . .	127
47.4	47.4 Read the values . . . . .	127

47.5	47.5 Store the values . . . . .	127
47.6	47.6 Plot the values as a graph . . . . .	127
47.7	47.7 Full software listing . . . . .	127
<b>48</b>	<b>48 Interrupts</b>	<b>128</b>
48.1	48.1 Switch bounce problem investigation . . . . .	129
48.2	48.2 Keypad- polling versus interrupt driven . . . . .	129
48.3	48.3 Improving the HT12 radio system by using interrupts . . . . .	129
48.4	48.4 Magnetic Card Reader . . . . .	129
48.5	48.5 Card reader data structure . . . . .	129
48.6	48.6 Card reader data timing . . . . .	129
48.7	48.7 Card reader data formats . . . . .	129
48.8	48.8 Understanding interrupts in Bascom- trialling . . . . .	129
48.9	48.9 Planning the program . . . . .	129
48.10	48.10 Pin Change Interrupts PCINT0-31 . . . . .	129
<b>49</b>	<b>49 Timer/Counters</b>	<b>130</b>
49.1	49.1 Timer2 (16 bit) Program . . . . .	130
49.2	49.2 Timer0 (8 bit) Program . . . . .	130
49.3	49.3 Accurate tones using a timer (Middle C) . . . . .	130
49.4	49.4 Timer1 Calculator Program . . . . .	130
49.5	49.5 Timer code to make a siren by varying the preload value . . . . .	130
<b>50</b>	<b>50 LED dot matrix scrolling display project — arrays and timers</b>	<b>131</b>
50.1	50.1 Scrolling text code . . . . .	131
50.2	50.2 Scrolling text — algorithm design . . . . .	131
50.3	50.3 Scrolling test — code . . . . .	131

<b>51 51 Medical machine project – timer implementation</b>	<b>132</b>
51.1 51.1 Block diagram . . . . .	132
51.2 51.2 Blower - state machine . . . . .	132
51.3 51.3 Blower program code . . . . .	132
<b>52 52 Multiple 7-segment clock project — dual timer action</b>	<b>133</b>
52.1 52.1 Understanding the complexities of the situation . . . . .	134
52.2 52.2 Hardware understanding . . . . .	134
52.3 52.3 Classroom clock — block diagram . . . . .	134
52.4 52.4 Classroom clock — schematic . . . . .	134
52.5 52.5 Classroom clock — PCB layout . . . . .	134
52.6 52.6 Relay Circuit Example . . . . .	134
52.7 52.7 Classroom clock — flowcharts . . . . .	134
52.8 52.8 Classroom clock — program . . . . .	134
<b>53 53 The MAX 7219/7221 display driver IC's</b>	<b>135</b>
53.1 53.1 AVR clock/oscillator . . . . .	135
<b>54 54 Cellular Connectivity-ADH8066</b>	<b>136</b>
54.1 54.1 ADH prototype development . . . . .	137
54.2 54.2 ADH initial test setup block diagram . . . . .	137
54.3 54.3 Process for using the ADH . . . . .	137
54.4 54.4 ADH communications . . . . .	137
54.5 54.5 Initial state machine . . . . .	137
54.6 54.6 Status flags . . . . .	137
54.7 54.7 Second state machine . . . . .	137
54.8 54.8 StateMachine 3 . . . . .	137

54.9	54.9 Sending an SMS text . . . . .	137
54.10	54.10 Receiving an SMS text . . . . .	137
54.11	54.11 Splitting a large string (SMS message) . . . . .	137
54.12	54.12 Converting strings to numbers . . . . .	137
54.13	54.13 Full Program listing for SM3 . . . . .	137
<b>55</b>	<b>55 Data transmission across the internet</b>	<b>138</b>
55.1	55.1 IP address . . . . .	139
55.2	55.2 MAC (physical) address . . . . .	139
55.3	55.3 Subnet mask . . . . .	139
55.4	55.4 Ping . . . . .	139
55.5	55.5 Ports . . . . .	139
55.6	55.6 Packets . . . . .	139
55.7	55.7 Gateway . . . . .	139
55.8	55.8 DNS . . . . .	139
55.9	55.9 WIZNET812 . . . . .	139
55.10	55.10 Wiznet 812 Webserver V1 . . . . .	139
55.11	55.11 Transmitting data . . . . .	139
55.12	55.12 Wiznet Server2 (version1) . . . . .	139
55.13	55.13 'Main do loop . . . . .	139
55.14	55.14 process any messages received from browser . . . . .	139
55.15	55.15 Served webpage . . . . .	139
<b>56</b>	<b>56 Assignment – maths in the real world</b>	<b>140</b>
56.1	56.1 Math assignment - part 1 . . . . .	141
56.2	56.2 Math assignment - part 2 . . . . .	141
56.3	56.3 Math assignment - part 3 . . . . .	141

56.4	56.4 Math assignment - part 4 . . . . .	141
56.5	56.5 Math assignment - part 5 . . . . .	141
56.6	56.6 Math assignment - part 6 . . . . .	141
56.7	56.7 Extension exercise . . . . .	141
<b>57</b>	<b>57 SSD1928 based colour graphics LCD</b> . . . . .	<b>142</b>
57.1	57.1 System block diagram . . . . .	143
57.2	57.2 TFT LCDs . . . . .	143
57.3	57.3 System memory requirements . . . . .	143
57.4	57.4 System speed . . . . .	143
57.5	57.5 SSD and HX ICs . . . . .	143
57.6	57.6 Colour capability . . . . .	143
57.7	57.7 SSD1928 and HX8238 control requirements . . . . .	143
57.8	57.8 SSD1928 Software . . . . .	143
57.9	57.9 SSD1928 microcontroller hardware interface . . . . .	143
57.10	57.10 Accessing SSD control registers . . . . .	143
57.11	57.11 SSD1928_Register_routines.bas . . . . .	143
57.12	57.12 Accessing the HX8238 . . . . .	143
57.13	57.13 SSD1928_GPIO_routines.bas . . . . .	143
57.14	57.14 LCD timing signals . . . . .	143
57.15	57.15 HX setups . . . . .	143
57.16	57.16 SSD setups . . . . .	143
57.17	57.17 SSD line / HSync timing . . . . .	143
57.18	57.18 SSD row / VSync/ frame timing . . . . .	143
57.19	57.19 HX and SSD setup routine . . . . .	143
57.20	57.20 'SSD1928_HardwareSetup_Routines.bas . . . . .	143
57.21	57.21 SSD1928_Window_Control_Routines.bas . . . . .	143

57.2257.22	Colour data in the SSD memory . . . . .	143
57.2357.23	Accessing the SSD1928 colour memory . . . . .	143
57.2457.24	'SSD1928_Memory_Routines.bas . . . . .	143
57.2557.25	Drawing simple graphics . . . . .	143
57.2657.26	'SSD1928_Simple_Graphics_Routines.bas . . . . .	143
57.2757.27	SSD1928_text_routines . . . . .	143
<b>58 58</b>	<b>Traffic Light help and solution</b>	<b>144</b>
<b>59 59</b>	<b>Computer programming — low level detail</b>	<b>145</b>
59.1	59.1 Low level languages: . . . . .	146
59.2	59.2 AVR Internals — how the microcontroller works . . . . .	146
59.3	59.3 1. The 8 bit data bus . . . . .	146
59.4	59.4 2. Memory . . . . .	146
59.5	59.5 3. Special Function registers . . . . .	146
59.6	59.6 A simple program to demonstrate the AVR in operation . . . . .	146
59.7	59.7 Bascom keyword reference . . . . .	146
<b>60 60</b>	<b>USB programmer — USBASP</b>	<b>147</b>
<b>61 61</b>	<b>USBTinyISP programmer</b>	<b>148</b>
<b>62 62</b>	<b>C-Programming and the AVR</b>	<b>149</b>
62.1	62.1 Configuring a programmer . . . . .	150
62.2	62.2 First program . . . . .	150
62.3	62.3 Output window . . . . .	150
62.4	62.4 Configuring inputs & outputs . . . . .	150
62.5	62.5 Making a single pin an input . . . . .	150

62.6	62.6 Making a single pin an output . . . . .	150
62.7	62.7 Microcontroller type . . . . .	150
62.8	62.8 Includes . . . . .	150
62.9	62.9 Main function . . . . .	150
62.10	62.10 The blinkyelled program . . . . .	150
62.11	62.11 Counting your bytes . . . . .	150
62.12	62.12 Optimising your code . . . . .	150
62.13	62.13 Reading input switches . . . . .	150
62.14	62.14 Macros . . . . .	150
62.15	62.15 Auto-generated config from System Designer . . . . .	150
62.16	62.16 Writing your own functions . . . . .	150
62.17	62.17 AVR Studio editor features . . . . .	150
62.18	62.18 AVR hardware registers . . . . .	150
62.19	62.19 Character LCD programming in C . . . . .	150
62.20	62.20 CharLCD.h Header file . . . . .	150
62.21	62.21 Manipulating AVR register addresses . . . . .	150
62.22	62.22 Writing to the LCD . . . . .	150
62.23	62.23 Initialise the LCD . . . . .	150
62.24	62.24 lcd commands . . . . .	150
62.25	62.25 Writing text to the LCD . . . . .	150
62.26	62.26 Program Flash and Strings . . . . .	150
62.27	62.27 LCD test program1 . . . . .	150
62.28	62.28 CharLCD.h . . . . .	150
62.29	62.29 CharLCD.c . . . . .	150
63	63 Object Oriented Programming (OOP) in CPP and the AVR . . . . .	151
63.1	63.1 The black box concept . . . . .	152

63.2	63.2 The concept of a class . . . . .	152
63.3	63.3 First CPP program . . . . .	152
63.4	63.4 Creating an AVR CPP program in Atmel Studio 6 . . . . .	152
63.5	63.5 Adding our class files to the project . . . . .	152
63.6	63.6 First Input and output program . . . . .	152
63.7	63.7 Class OutputPin . . . . .	152
63.8	63.8 Class InputPin . . . . .	152
63.9	63.9 Inheritance . . . . .	152
63.10	63.10 Class IOPin . . . . .	152
63.11	63.11 Encapsulation . . . . .	152
63.12	63.12 Access within a class . . . . .	152
63.13	63.13 Class Char_LCD . . . . .	152
63.14	63.14 Exercise — create your own Led class . . . . .	152
<b>64</b>	<b>64 Current (2014) AVR development PCBS</b>	<b>153</b>
64.1	64.1 Year 9 ATTiny 45 Board . . . . .	154
64.2	64.2 Year10 ATTiny461 V4a development board . . . . .	154
64.3	64.3 Year11 ATMega48 (or 88 or 168 or 328) V4 . . . . .	154
64.4	64.4 Year11 ATMega48 (or 88 or 328) v3 development board . . . . .	154
64.5	64.5 Year 12 ATMega 20x4 Character LCD v6A . . . . .	154
64.6	64.6 Year 13 ATMega GLCD 128x64 (2014) Veroboard . . . . .	154
64.7	64.7 Year 13 ATMega GLCD (older pin connections) . . . . .	154
64.8	64.8 ATMEGA microcontroller pin connections . . . . .	154
64.9	64.9 ATMEGA16/644 40pin DIP package — pin connections . . . . .	154
<b>65</b>	<b>65 Eagle — creating your own library</b>	<b>155</b>
65.1	65.1 Autorouting PCBS . . . . .	155

<b>66 66 Practical Techniques</b>	<b>156</b>
66.1 66.1 PCB Mounting . . . . .	157
66.2 66.2 Countersink holes and joining MDF/wood . . . . .	157
66.3 66.3 MDF . . . . .	157
66.4 66.4 Plywood . . . . .	157
66.5 66.5 Acrylic . . . . .	157
66.6 66.6 Electrogalv . . . . .	157
66.7 66.7 Choosing fasteners . . . . .	157
66.8 66.8 Workshop Machinery . . . . .	157
66.9 66.9 Glues/Adhesives . . . . .	157
66.10 66.10 Wood Joining techniques . . . . .	157
66.11 66.11 Codes of Practice for student projects . . . . .	157
66.12 66.12 Fitness for purpose definitions and NZ legislation . . . . .	157
<b>67 67 CNC</b>	<b>158</b>
67.1 67.1 Machine overview . . . . .	159
67.2 67.2 Starting the CNC machine . . . . .	159
67.3 67.3 CamBam . . . . .	159
67.4 67.4 CamBam options . . . . .	159
67.5 67.5 Drawing shapes in CamBam . . . . .	159
67.6 67.6 Machining commands . . . . .	159
67.7 67.7 A Box of Pi . . . . .	159
67.8 67.8 Holding Tabs . . . . .	159
67.9 67.9 Engraving . . . . .	159
67.10 67.10 Polylines . . . . .	159
<b>Index</b>	<b>160</b>

I Рабочая среда разработчика встраиваемых систем	161
67.11 Операционная система с набором типовых утилит . . . . .	162
67.12 САПР электронных устройств (EDA CAD) . . . . .	162
67.13 Пакет расчета и симуляции электронных схем: SPICE . . . . .	162
67.14 САПР общего назначения . . . . .	162
67.15 Система управления версиями: VCS . . . . .	162
67.16 Текстовый редактор или интегрированная среда разработки (IDE) . . . . .	163
67.17 ПО для программатора, JTAG-адаптера . . . . .	163
67.18 Симулятор для отладки программ без железа . . . . .	163
67.19 Система верстки документации . . . . .	164
II Пакет компиляторов и утилит GNU toolchain	165
68 Make	167
69 binutils	168
70 Формат объектного файла GNU ELF	169
71 Ассемблер GNU <code>as</code>	171
72 Линкер GNU <code>ld</code>	172
72.1 Использование <code>ld</code> . . . . .	173
72.2 Обзор . . . . .	173
72.3 Вызов . . . . .	173
72.3.1 Параметры командной строки . . . . .	173
72.3.2 Опции, специфичные для целевой платформы i386 PE . . . . .	173

72.3.3 Переменные среды . . . . .	173
72.4 Скрипты линкера . . . . .	173
72.5 Основные понятия скрипта линкера . . . . .	173
72.6 Формат скрипта компоновщика . . . . .	173
72.7 Пример простого скрипта компоновщика . . . . .	173
72.8 Команды простого скрипта компоновщика . . . . .	173
72.8.1 Установка точки входа . . . . .	173
72.8.2 Команды работы с файлами . . . . .	173
72.8.3 Работа с форматами объектный файлов . . . . .	173
72.8.4 Другие команды сценария компоновщика . . . . .	173
72.9 Присвоение значений символам . . . . .	173
72.9.1 Простые назначения . . . . .	173
72.9.2 PROVIDE . . . . .	173
72.10 Команда SECTIONS . . . . .	173
72.10.1 Описание выходных секций . . . . .	175
72.10.2 Имя выходной секции . . . . .	175
72.10.3 Описание выходных секций . . . . .	178
72.10.4 Описание входных секций . . . . .	178
72.10.5 Выходная секция данных . . . . .	178
72.10.6 Ключевые слова выходных секций . . . . .	178
72.10.7 Отброс выходных секций . . . . .	178
72.10.8 Атрибуты выходных секций . . . . .	178
72.10.9 Описание оверлея . . . . .	178
72.11 Команда MEMORY . . . . .	178
72.12 Команда PHDRS . . . . .	178
72.13 Команда VERSION . . . . .	178
72.14 Выражения в скриптах компоновщика . . . . .	178

72.14.1 Константы . . . . .	178
72.14.2 Имена символов . . . . .	178
72.14.3 Счетчик адреса . . . . .	178
72.14.4 Операторы . . . . .	178
72.14.5 Вычисления . . . . .	178
72.14.6 Часть выражения . . . . .	178
72.14.7 Встроенные функции . . . . .	178
72.15 Неявные скрипты компоновщика . . . . .	178
72.16 Машино-зависимые особенности . . . . .	178
72.16.1 ld и H8/300 . . . . .	178
72.16.2 ld и семейство Intel 960 . . . . .	178
72.16.3 поддержка взаимодействия между кодом в режимах ARM и Thumb . . . . .	178
72.16.4 ld и поддержка 32-разрядных ELF HPPA . . . . .	178
72.16.5 ld и MMIX . . . . .	178
72.16.6 ld и MSP430 . . . . .	178
72.16.7 поддержка различных версий TI COFF . . . . .	178
72.16.8 ld и WIN32 (cygwin/mingw) . . . . .	178
72.16.9 ld и процессоры Xtensa . . . . .	178
72.17 BFD . . . . .	178
72.17.1 Как это работает: очерк о BFD . . . . .	178
72.17.2 Потеря информации . . . . .	178
72.17.3 Канонический формат объектных файлов BFD . . . . .	178
<b>73 objdump</b>	<b>179</b>
<b>74 GNU Compiler Collection (GCC)</b>	<b>180</b>

75 Компилятор GNU C	181
76 Компилятор GNU C++	182
77 Компилятор GNU Fortran	183
<b>III САПР электронных устройств KiCAD</b>	<b>184</b>
<b>78 Установка</b>	<b>187</b>
78.1 Windows . . . . .	187
78.2 Linux . . . . .	188
78.3 Настройка библиотек . . . . .	188
78.4 Дотфайлы . . . . .	189
78.5 Глобальные шаблоны . . . . .	192
<b>79 Маршрут проектирования</b>	<b>197</b>
<b>80 Создание проекта в менеджере проектов <i>kicad</i></b>	<b>199</b>
<b>81 Создание принципиальной схемы в <i>eeschema</i> (часть 1)</b>	<b>203</b>
<b>82 <i>eeschema</i>: редактор электрических схем</b>	<b>205</b>
<b>83 Библиотеки компонентов</b>	<b>206</b>
83.1 Создание УГО для схем . . . . .	207
83.2 Модель печатной платы . . . . .	213
83.3 Создание падстека . . . . .	214

84 <b>gerbview</b> : просмотр фотошаблонов	215
85 Программа <b>Wings3D</b> для создания 3D моделей	216
85.1 Установка <b>Wings3D</b> под Windows . . . . .	216
<b>IV Библиотека универсальных электронных модулей «Одурино»</b>	218
<b>86 USB</b>	220
86.1 Стек протоколов USB . . . . .	221
86.2 libUSB . . . . .	223
86.2.1 драйвер для устройства USB . . . . .	223
86.3 Поддержка USB в Linux . . . . .	224
86.3.1 Опции ядра . . . . .	224
86.3.2 Настройка hotplug и автомонтирования USB носителей . . . . .	224
86.3.3 Сборка и настройка libusb . . . . .	224
86.3.4 Примеры программ низкоуровневого ввода/вывода . . . . .	224
<b>V Расчет схем и моделирование в <b>ngSPICE</b></b>	225
<b>87 Доступные SPICE-пакеты</b>	227
87.1 Установка <b>ngSPICE</b> под Windows . . . . .	228
87.2 Установка <b>LT-SPICE</b> (только под Windows) . . . . .	228
87.3 Установка <b>ngSPICE</b> под Linux . . . . .	229
<b>88 Пошаговый пример использования</b>	230
88.1 Рисуем схему в <b>KiCAD</b> . . . . .	230

88.2 Создание списка цепей . . . . .	232
88.3 Запуск симуляции . . . . .	234
88.4 Просмотр результата расчета . . . . .	236
88.5 Расчет АЧХ по переменному току (АС симуляция) . . . . .	240
88.6 Симуляция полноволнового выпрямителя . . . . .	244
<b>89 Настройка KiCAD для SPICE-моделирования</b>	<b>247</b>
89.1 Библиотеки компонентов со SPICE-элементами . . . . .	247
89.2 Настройка проекта . . . . .	248
89.3 Как это работает . . . . .	248
<b>VI Разработка конструкции в САПР FreeCAD</b>	<b>250</b>
<b>90 Установка</b>	<b>253</b>
90.1 Windows . . . . .	253
90.2 Linux . . . . .	254
<b>VII Инструменты и электронное оборудование</b>	<b>255</b>
<b>91 Радиомонтажный инструмент</b>	<b>256</b>
91.1 Pro'sKit . . . . .	256
91.2 Инструмент до 1000 В . . . . .	261
91.3 Хранение . . . . .	262
91.4 Радиомонтаж . . . . .	263
91.5 Прочие . . . . .	267

<b>92 Паяльное оборудование</b>	<b>268</b>
92.1 Паяльник . . . . .	268
92.2 Паяльная станция . . . . .	270
<b>93 Измерительное оборудование</b>	<b>273</b>
93.1 Мультиметр . . . . .	273
93.1.1 Mastech M838 . . . . .	274
93.1.2 Mastech M300 . . . . .	275
93.1.3 Mastech M320 . . . . .	276
93.2 Осциллограф . . . . .	277
93.3 Логический анализатор . . . . .	277
93.4 Генератор сигналов . . . . .	277
93.5 Рыльцеметр RLC . . . . .	277
<b>94 Электроинструмент</b>	<b>278</b>
94.1 Дрель . . . . .	279
94.2 Лобзик . . . . .	282
94.3 Жигатель . . . . .	283
<b>VIII Станочное оборудование</b>	<b>286</b>
<b>95 Настольные станки</b>	<b>289</b>
<b>96 Самодельная оснастка</b>	<b>290</b>
<b>97 Промышленные станки</b>	<b>291</b>
97.1 1A616: станок токарно-винторезный . . . . .	292

97.1.1 Назначение и область применения . . . . .	293
97.1.2 Распаковка и транспортировка . . . . .	293
97.1.3 Фундамент станка, монтаж и установка . . . . .	293
97.1.4 Подготовка станка к первоначальному пуску . . . . .	293
97.1.5 Паспортные данные . . . . .	293
97.1.6 Описание основных узлов . . . . .	293
97.1.7 Смазка . . . . .	293
97.1.8 Первоначальный пуск . . . . .	293
97.1.9 Указания по технике безопасности . . . . .	293
97.1.10 Настройка . . . . .	293
97.1.11 Регулирование . . . . .	293
97.1.12 Ведомость комплектации . . . . .	293
<b>IX Разработка ПО для встраиваемых систем</b>	<b>294</b>
<b>98 IDE</b>	<b>295</b>
98.1  ECLIPSE . . . . .	298
98.1.1 Установка  ECLIPSE под  Windows . . . . .	298
98.1.2 Установка  ECLIPSE под Linux . . . . .	298
98.1.3 Установка CDT . . . . .	299
98.1.4 Установка PyDev . . . . .	300
98.1.5 Установка TeXlipse . . . . .	300
98.1.6 Редактирование файлов в формате XML и производных . . . . .	301
98.1.7 Проверка орфографии . . . . .	301
98.2 Code::Blocks . . . . .	304
98.3 Vim . . . . .	304

98.3.1 Установка под Windows . . . . .	304
98.3.2 Выход из Vim . . . . .	306
98.3.3 Выход с автосохранением . . . . .	307
98.3.4 Переход в режим редактирования . . . . .	307
98.3.5 Переход в режим команд . . . . .	307
98.3.6 Запись редактируемого файла . . . . .	307
98.3.7 Перезагрузка файла . . . . .	308
98.3.8 Отмена последних изменений (undo) . . . . .	308
<b>99 Make: управление сборкой проектов</b>	<b>309</b>
<b>100 VCS: системы контроля версий</b>	<b>310</b>
100.1 CVS . . . . .	310
100.2 Subversion . . . . .	310
100.3 Git . . . . .	310
100.3.1 GitHub . . . . .	310
<b>101 Вспомогательные скрипты на языке Python</b>	<b>311</b>
<b>102 Основы Си и C<sub>+</sub><sup>+</sup></b>	<b>312</b>
102.1 Установка MinGW (win32) . . . . .	312
102.2 Особенности C <sub>+</sub> <sup>+</sup> в embedded . . . . .	312
102.3 Сборка кросс-компилятора GNU toolchain . . . . .	312
<b>103 Лексический и синтаксический анализ</b>	<b>313</b>
103.1 Лексер и лексический анализ, утилита <b>flex</b> . . . . .	314
103.2 Генератор синтаксических анализаторов <b>bison</b> . . . . .	318

<b>104 LLVM и разработка собственных компиляторов</b>	<b>321</b>
104.1 Установка под Windows . . . . .	321
104.2 Создание компилятора с помощью инфраструктуры LLVM . . . . .	321
104.3 Инструменты <i>llc</i> и <i>lli</i> . . . . .	322
104.4 Семантический анализ . . . . .	323
104.5 Оптимизация . . . . .	323
104.6 Кодогенерация . . . . .	323
104.7 Транслятор Паскаля . . . . .	324
<b>X Микроконтроллеры Cortex-Mx</b>	<b>325</b>
<b>105 Производители</b>	<b>326</b>
105.1 ST Microelectronics STM32 . . . . .	326
105.2 LPC . . . . .	326
105.3 Миландр . . . . .	326
<b>106 Отладочные платы</b>	<b>327</b>
106.1 LeafLabs Maple Mini: STM32F103 /Cortex-M3/ . . . . .	327
106.2 Серия STM32 STM32DISCOVERY . . . . .	327
106.2.1 STM32DISCOVERY: STM32F103 /Cortex-M3/ . . . . .	327
106.2.2 STM32F4DISCOVERY: STM32F406 /Cortex-M4F/ . . . . .	327
106.2.3 STM32F0DISCOVERY: STM32F040 /Cortex-M0/ . . . . .	327

<b>XI ПЛИС</b>	<b>328</b>
<b>XII USB</b>	<b>330</b>
<b>107Стек протоколов USB</b>	<b>333</b>
<b>108ibUSB</b>	<b>335</b>
108.1драйвер для устройства USB . . . . .	335
<b>109Поддержка USB в Linux</b>	<b>336</b>
109.1Опции ядра . . . . .	337
109.1.1режимы host/client/otg и хост-контроллеры xHCI . . . . .	337
109.1.2data storage: носители данных . . . . .	337
109.1.3hid: клавиатура, мышь, джойстик . . . . .	337
109.1.4USB-периферия: сеть, звук,. . . . .	337
109.2Настройка hotplug и автомонтирования USB носителей . . . . .	337
109.3Сборка и настройка libusb . . . . .	337
109.4Примеры программ низкоуровневого ввода/вывода . . . . .	337
<b>XIII Встраиваемый emLinux</b>	<b>338</b>
<b>110Загрузчик syslinux</b>	<b>340</b>
110.1Закачка . . . . .	340
110.2Установка под Windows на флешку . . . . .	341
110.3syslinux.cfg . . . . .	342

<b>111azLinux</b>	<b>345</b>
111.1 Требования к системе сборки ( <b>BUILD</b> -хост) . . . . .	347
111.2 Понятие <b>пакет</b> . . . . .	348
111.3 Клонирование проекта <b>azLinux</b> . . . . .	350
111.4 Общий порядок сборки . . . . .	351
111.5 Фиксация переменных . . . . .	353
111.6 <b>dirs</b> : Создание дерева каталогов . . . . .	353
111.7 <b>gz</b> : Загрузка архивов исходников . . . . .	357
111.8 <b>APP</b> : Приложение . . . . .	361
111.9 <b>HW</b> : Поддерживаемое железо . . . . .	363
111.10 <b>386</b> . . . . .	364
111.10.1 <b>qemu386</b> : эмулятор QEMU . . . . .	364
111.10.2 <b>eepc701</b> : ASUS Eee PC 701 . . . . .	366
111.10.3 <b>gac1037</b> : Gigabyte GA-C1037UN-EU rev.2 . . . . .	367
111.11 <b>ARM</b> . . . . .	368
111.11.1 <b>qemuARM</b> : эмулятор QEMU . . . . .	368
111.11.2 <b>cubie1</b> : Cubie Board v.1 . . . . .	369
111.11.3 <b>pi</b> : Raspberry Pi model B . . . . .	369
111.11.4 <b>tion</b> : ТионПро270 . . . . .	370
111.11.5 <b>mb77</b> : Микрокомпьютер MB77.07 на базе СБИС K1879ХБ1Я . . . . .	371
111.12 <b>MIPS</b> . . . . .	372
111.12.1 <b>qemuMIPS</b> : эмулятор QEMU . . . . .	372
111.12.2 <b>mr3020</b> : роутер MR3020 . . . . .	373
111.12.3 <b>VoCore</b> . . . . .	374
111.12.4 <b>swift</b> : BlackSwift . . . . .	374
111.13 <b>CPU</b> : Конфигурации процессоров . . . . .	375
111.13.1 <b>i386</b> . . . . .	375

111.13.2	ARM	376
111.13.3	MIPS	376
111.1	Пакеты	376
111.14.1	<code>mk/versions.mk</code> : Версии пакетов	376
111.14.2	<code>cross0</code> : кросс-компилятор	376
111.14.3	<code>core</code> : ядро	377
111.14.4	<code>boot</code> : загрузчики	377
111.14.5	<code>libs</code> : библиотеки	378
111.14.6	<code>cross0</code> : сборка кросс-компилятора	378
111.14.7	<code>binutils</code> : ассемблер, линкер и утилиты	383
111.14.8	<code>clibs</code> : библиотеки для сборки <code>gcc</code>	384
111.14.9	<code>gcc0</code> : сборка минимального кросс-компилятора Си	385
111.14.10	<code>gcc</code> : пересборка полного кросс-компилятора Си/ $C^+$	385
111.14.11	<code>core</code> : сборка основной системы	385
111.14.12	<code>kernel</code> : ядро Linux	385
111.14.13	<code>libc</code> : библиотека <code>uClibc</code>	391
111.14.14	<code>gcc</code> : пересборка полного <code>gcc</code>	396
111.14.15	<code>busybox</code> : набор утилит <code>busybox</code>	397
111.14.16	<code>libs</code> : сборка библиотек <code> \${LIBS} </code>	398
111.14.17	<code>apps</code> : сборка прикладных пакетов <code> \${APPS} </code>	398
111.14.18	<code>user</code> : сборка пользовательского кода	398
111.14.19	<code>root</code> : формирование корневой файловой системы	398
111.14.20	<code>root</code> : сборка загрузчика <code>syslinux/grub/uboot</code>	400
111.14.21	<code>syslinux</code>	401
111.14.22	<code>grub</code>	401
111.14.23	<code>boot</code>	401
111.14.24	<code>run</code> : запуск собранной системы в эмуляторе	401

11.1.1 <code>betboot</code> : Сетевая загрузка . . . . .	404
11.1.2 Прошивка на устройство . . . . .	404
11.1.3 RT-патч . . . . .	404
11.1.8 <code>math</code> : компьютерная математика . . . . .	404
11.1.18.1 <code>blas</code> : библиотека <b>BLAS</b> . . . . .	404
11.1.18.2 <code>lapack</code> : библиотека <b>LAPACK</b> . . . . .	404
11.1.18.3 <code>Octave</code> : система численных методов <b>Octave</b> . . . . .	404
11.1.18.4 <code>maxima</code> : система аналитической математики <b>Maxima</b> . . . . .	404
11.1.9 <code>SDK</code> : расширения для on-board разработки . . . . .	404
11.1.19.1 <code>canadian</code> : сборка binutils канадским крестом . . . . .	405
11.1.19.2 <code>binhost</code> : binutils для хост-процессора . . . . .	407
11.1.19.3 <code>binavr8</code> : binutils для Atmel ATmega (AVR8) . . . . .	407
11.1.19.4 <code>bincmx</code> : binutils для ARM Cortex-Mx . . . . .	408
11.1.19.5 <code>fpc</code> : FreePascal . . . . .	408
11.1.19.6 <code>python</code> : интерпретатор Python . . . . .	409
11.1.19.7 <code>gcl</code> : GNU Common Lisp . . . . .	409
<b>11.2 Применение Linux для новой “железки”</b>	<b>410</b>
<b>11.3 Библиотека libSDL</b>	<b>412</b>
11.3.1 Инициализация и завершение SDL-программы . . . . .	413
11.3.2 LazyFoo tutorial . . . . .	413
11.3.2.1 Setting up SDL and Getting an Image on the Screen . . . . .	413
11.3.2.2 Optimized Surface Loading and Blitting . . . . .	419
11.3.2.3 Extension Libraries and Loading Other Image Formats . . . . .	427
11.3.2.4 Event Driven Programming . . . . .	428
11.3.2.5 Color Keying . . . . .	437

113.2.6 Clip Blitting and Sprite Sheets . . . . .	437
113.2.7 True Type Fonts . . . . .	437
113.2.8 Key Presses . . . . .	437
113.2.9 Mouse Events . . . . .	437
113.2.10 Key States . . . . .	437
113.2.11 Playing Sounds . . . . .	437
113.2.12 Timing . . . . .	437
113.2.13 Advanced Timers . . . . .	437
113.2.14 Regulating Frame Rate . . . . .	437
113.2.15 Calculating Frame Rate . . . . .	437
113.2.16 Motion . . . . .	437
113.2.17 Collision Detection . . . . .	437
113.2.18 Per-pixel Collision Detection . . . . .	437
113.2.19 Circular Collision Detection . . . . .	437
113.2.20 Animation . . . . .	437
113.2.21 Scrolling . . . . .	437
113.2.22 Scrolling Backgrounds . . . . .	437
113.2.23 Getting String Input . . . . .	437
113.2.24 Game Saves . . . . .	437
113.2.25 Joysticks . . . . .	437
113.2.26 Resizable Windows and Window Events . . . . .	437
113.2.27 Alpha Blending . . . . .	437
113.2.28 Particle Engines . . . . .	437
113.2.29 Tiling . . . . .	437
113.2.30 Bitmap Fonts . . . . .	437
113.2.31 Pixel Manipulation and Surface Flipping . . . . .	437
113.2.32 Frame Independent Movement . . . . .	437

113.2.31	Multithreading . . . . .	437
113.2.34	Semaphores . . . . .	437
113.2.35	Mutexes and Conditions . . . . .	437
113.2.36	Using OpenGL with SDL . . . . .	437
113.3	Примеры программ . . . . .	437
113.3.1	Графический Hello World . . . . .	438
113.3.2	Вывод случайных прямоугольников . . . . .	440
113.4	Версия SDL2 . . . . .	441
114	BuildRoot	442
115	Особенности OpenWrt	443
<b>XIV</b>	<b>x86os: простая операционная система для компьютера на i386 процессоре</b>	<b>444</b>
116	Ресурсы	445
117	Структура	447
118	Процесс запуска	448
119	Сборка кросс-компилятора	449
120	Формат ELF32	455
121	multiboot	457

<b>122</b>	<b>Микроядро</b>	460
<b>123</b>	<b>Драйвера</b>	462
123.1	<b>vga</b> : текстовая консоль VGA 80×25 . . . . .	462
123.2	<b>kbd</b> : клавиатура . . . . .	465
123.3	<b>ide</b> : жесткий диск IDE . . . . .	465
123.3.1	<b>fatfs</b> : файловая система FAT16 . . . . .	465
<b>XV</b>	<b>Символьная и численная математика</b>	466
<b>124</b>	<b>Общие сведения о компьютерной математике</b>	468
<b>125</b>	<b>Пакет Octave</b>	471
<b>126</b>	<b>Пакет Maxima</b>	472
126.1	Установка Maxima под Windows . . . . .	473
<b>127</b>	<b>Параллельные вычисления</b>	474
127.1	Многоядерные архитектуры с разделяемой памятью . . . . .	474
127.2	Кластер архитектуры Beowulf . . . . .	474
127.3	Вычисления на GPU . . . . .	474
127.4	Средства параллелизации $C_+$ . . . . .	474
127.5	BLAS/LAPACK/MPI/ScalaPack . . . . .	474
127.6	Средства измерения производительности . . . . .	474

<b>XVI Подготовка технической документации</b>	<b>475</b>
128 Верстка в L <sup>A</sup> T <sub>E</sub> X	476
129 Оформление листингов	477
<b>130 Подготовка иллюстраций</b>	<b>478</b>
130.1 GIMP . . . . .	478
130.2 Inkscape . . . . .	478
130.3 Graphviz . . . . .	478
<b>131 Замечания для соавторов “Абзуки АРМатурщика”</b>	<b>479</b>
<b>XVII Примерные учебные планы</b>	<b>480</b>
132 Блондинко	481
133 Школотрон	482
134 Студень	483
135 Технический специалист	484
<b>XVIII Куча</b>	<b>485</b>
136 Автоматное программирование /фреймворк QuantumLeaps/	487

137 Запуск Linux на android устройстве

в режиме паразита

488

Предметный указатель

501

# О книге

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

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

Большой упор был сделан на использование открытого некоммерческого программного обеспечения, для удешевления учебного процесса, уменьшения себестоимости ваших проектов<sup>1</sup>, и стимулирования вашего участия в развитии этих программных пакетов.

Книга очень объемна и разнообразна по материалу, и построена как справочник с группировкой материала по тематике. Для тех, кто только начинает, в разделе XVII расписаны **пошаговые учебные планы** с точки зрения параллельного изучения нескольких предметов с постепенным усложнением<sup>2</sup>. 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

<sup>1</sup> вряд ли ли у вас окажется лишняя пачка килобаксов на покупку пары коммерческих САПР, по крайней мере пока ваш стартап не взлетит в Топ\$100K

<sup>2</sup> как это происходит при традиционном offline обучении

*Electronics, Microcontrollers and Software Design* [?] в русскоязычном варианте «Азбуки» (??), и конечно он вполне заслуженно включен в основные соавторы этой книги.

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** [?] (95)
- Doug Briney **Home Machinists Handbook** [?] (??)

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

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

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

# Глава 1

## 1. Introduction to Practical Electronics

Введение в практическую электронику

# От переводчика

Эта часть основана на переводе книги:

## An Introduction to Practical Electronics, Microcontrollers and Software Design

Second Edition, 01 May-2014

© Bill Collis

[www.techideas.co.nz](http://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

This book has a number of focus areas.

Эта книга ©<sup>1</sup> имеет следующий ряд основных направлений:

- Electronic component recognition and correct handling

Распознавание электронных компонентов и их правильное использование

- Developing a solid set of conceptual understandings in basic electronics.

Наработка цельного набора компетенций по основам электроники

- Electronic breadboard use

Использование макетных плат

- Hand soldering skills

Навыки ручной пайки

- Use of Ohm's law for current limiting resistors

Использование закона Ома для выбора токоограничивающих резисторов

- The voltage divider

Делитель напряжения

- CAD PCB design and manufacture

Использование EDA CAD<sup>2</sup> для разработки и подготовки производства печатных плат

- Microcontroller programming and interfacing

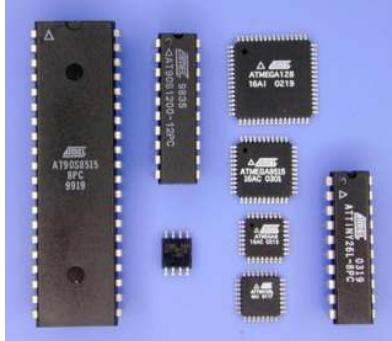
Программирование микроконтроллеров и их сопряжение с внешними устройствами

---

<sup>1</sup> оригинал: [?] B.Collis The Introduction to Practical Electronics...

<sup>2</sup> [E]lectronic [D]esign [A]utomation, САПР автоматизации проектирования электроники

- The transistor as a switch  
Использование транзистора в режиме ключа
- Power supply theory  
Теория источников питания
- Motor driving principles and circuits  
Принципы и схемы электропривода
- Modelling solutions through testing and trialing  
Навыки отладки схем, их тестирования и испытаний
- Following codes of practice  
Следование принципам обучения через практику
- Safe workshop practices  
Безопасные приемы работы



## 1.1 1.1 Your learning in Technology

### Ваше обучение по специальности «Технология»

#### 1.1.1 Technology Achievement Objectives from the NZ Curriculum

##### Цели обучения технологиям Ново-Зеландской программы

- **Technological Practice**

###### Технологическая практика

- **Brief:** develop clear specifications for your technology projects.

**Чёткость:** разработка ясных описаний для ваших технологических проектов.

- **Planning:** thinking about things before you start making them and using drawings such as flowcharts, circuit diagrams, pcb layouts, statecharts and sketchup plans while working.

**Планирование:** думать прежде чем делать, и использовать во время работы документацию: блок-схемы, принципиальные схемы, чертежи разводки плат, диаграммы и эскизы.

- **Outcome Development:** trialling, testing and building electronic circuits, designing and making PCBs, writing programs for microcontrollers.

**Наработка навыков:** сборка, отладка и тестирование электронных схем, проектирование и изготовление печатных плат, написание программ для микроконтроллеров.

- **Technological Knowledge**

###### Технологические знания

- **Technological Modelling:** before building an electronic device, it is important to find out how well it works first by modelling and/or trialling its hardware and software.

**Моделирование:** прежде чем строить готовое электронное устройство, сначала важно понять как оно работает путем моделирования и/или макетирования аппаратного и программного обеспечения.

- **Technological Products**: getting to know about components and their characteristics.  
**Технологические продукты**: знания о компонентах и их характеристиках.
- **Technological Systems**: an electronic device is more than a collection of components it is a functioning system with inputs, outputs and a controlling process.  
**Технологические системы**: электронное устройство является более, чем набором компонентов, это функционирующая система с входами, выходами и контролирующим процессом.

- **Nature of Technology**

**Природа технологии**

- **Characteristics of Technological Outcomes**: knowing about electronic components especially micro-controllers as the basis for modern technologies.  
**Значение технологических достижений**: знания об электронных компонентах, особенно микроконтроллерах, как основе современных технологий.
- **Characteristics of Technology**: electronic devices now play a central role in the infrastructure of our modern society; are we their masters, how have they changed our lives?  
**Роль технологии в обществе**: электронные устройства в настоящее время играют центральную роль в инфраструктуре нашего современного общества; подчинили ли они нас себе, как они изменили нашу жизнь?

## 1.2 1.2 Key Competencies from The NZ Curriculum

**Ключевые компетенции Ново-Зеландской программы**

- **Thinking**: to me the subject of technology is all about thinking. My goal is to have students understand the technologies embedded within electronic devices. To achieve this students must actively engage with their work at the earliest stage so that they can construct their own understandings and go on to become good problem

solvers. In the beginning of their learning in electronics this requires students to make sense of the instructions they have been given and search for clarity when they do not understand them.

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

After that there are many new and different pieces of knowledge introduced in class and students are given problem solving exercises to help them think logically. The copying of someone else's answer is flawed but working together is encouraged. At the core of learning is building correct conceptual models and to have things in the context of the 'big picture'.

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

- **Relating to others:** working together in pairs and groups is as essential in the classroom as it is in any other situation in life; we all have to share and negotiate resources and equipment with others; it is essential therefore to actively communicate with each other and assist one other.

**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 помочь друг другу.

- **Using language symbols and texts:** At the heart of our subject is the language we use for communicating electronic circuits, concepts, algorithms and computer programming syntax; so the ability to recognise and using

---

<sup>3</sup> в оригинале **enage**, англо-калька с *себуанского*, NZ

symbols and diagrams correctly for the work we do is vital.

**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.

- **Managing self:** This is about students taking personal responsibility for their own learning; it is about challenging students who expect to read answers in a book or have a teacher tell them what to do. It means that students need to engage with the material in front of them.

**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.

Sometimes the answers will come easily, sometimes they will not; often our subject involves a lot of trial and error (mostly error). Students should know that it is in the tough times that the most is learnt. And not to give up keep searching for understanding.

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.

- **Participating and contributing:** We live in a world that is incredibly dependent upon technology especially electronics, students need to develop an awareness of the importance of this area of human creativity to our daily lives and to recognise that our projects have a social function as well as a technical one.

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





# Глава 2

## 2 An introductory electronic circuit

2.1 2.1 Where to buy stuff?

2.2 2.2 Identifying resistors by their colour codes

2.3 2.3 LED's

2.4 2.4 Some LED Specifications

2.5 2.5 LED research task

2.6 2.6 Adding a switch to your circuit

2.7 2.7 Switch assignment

# Глава 3

## 3 Introductory PCB construction

3.1 3.1 Eagle Schematic and Layout Editor Tutorial

3.2 3.2 An Introduction to Eagle

3.3 3.3 The Schematic Editor

3.4 3.4 The Board Editor

3.5 3.5 Making Negative Printouts

3.6 3.6 PCB Making



# Глава 4

## Soldering, solder and soldering irons

4.1 4.1 Soldering facts

4.2 4.2 Soldering Safety

4.3 4.3 Soldering wires to switches

4.4 4.4 Codes of practice

4.5 4.5 Good and bad solder joints

4.6 4.6 Short circuits

4.7 4.7 Soldering wires to LED's



# Глава 5

## 5 Introductory Electronics Theory

5.1 5.1 Making electricity

5.2 5.2 ESD electrostatic discharge

5.3 5.3 Magnets, wires and motion

5.4 5.4 Group Power Assignment

5.5 5.5 Electricity supply in New Zealand

5.6 5.6 Conductors

5.7 5.7 Insulators



# Глава 6

## Introduction to microcontroller electronics

6.1 6.1 What is a computer?

6.2 6.2 What does a computer system do?

6.3 6.3 What does a microcontroller system do?

6.4 6.4 What exactly is a microcontroller?

6.5 6.5 Getting started with AVR Programming

6.6 6.6 Breadboard

6.7 6.7 Breadboard+Prototyping board circuit



# Глава 7

## 7 Microcontroller input circuits

7.1 7.1 Single push button switch

7.2 7.2 Pullup resistor theory

7.3 7.3 Switch in a breadboard circuit

7.4 7.4 Checking switches in your program

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

7.6 7.6 If-then exercises

7.7 7.7 Switch contact bounce



# Глава 8

## 8 Programming Review

8.1 8.1 Three steps to help you write good programs

8.2 8.2 Saving Programs

8.3 8.3 Organisation is everything

8.4 8.4 Programming template

8.5 8.5 What you do when learning to program

8.6 8.6 AVR microcontroller hardware

8.7 8.7 Power supplies



# Глава 9

## 9 Introduction to program flow

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

# Глава 10

## 10 Introductory programming - using subroutines

10.1 10.1 Sending Morse code

10.2 10.2 LM386 audio amplifier PCB

10.3 10.3 LM386 PCB Layout



# Глава 11

## 11 Introductory programming – using variables

11.1 11.1 Stepping or counting using variables

11.2 11.2 For-Next

11.3 11.3 Siren sound - programming using variables

11.4 11.4 Make a simple siren

11.5 11.5 Siren exercise

11.6 11.6 A note about layout of program code

11.7 11.7 Using variables for data

# Глава 12

## 12 Basic displays

12.1 12.1 7 segment displays

12.2 12.2 Alphanumeric LED displays



# Глава 13

## 13 TDA2822M Portable Audio Amplifier Project

13.1 13.1 Portfolio Assessment Schedule

13.2 13.2 Initial One Page Brief

13.3 13.3 TDA2822M specifications

13.4 13.4 Making a PCB for the TDA2822 Amp Project

13.5 13.5 Extra PCB making information

13.6 13.6 Component Forming Codes of Practice



# Глава 14

## 14 Basic programming logic

14.1 14.1 Quiz Game Controller

14.2 14.2 Quiz game controller system context diagram

14.3 14.3 Quiz game controller block diagram

14.4 14.4 Quiz game controller Algorithm

14.5 14.5 Quiz game schematic

14.6 14.6 Quiz game board veroboard layout

14.7 14.7 Quiz Controller flowchart



# Глава 15

## 15 Algorithm development – an alarm system

15.1 15.1 Simple alarm system – stage 1

15.2 15.2 Alarm System Schematic

15.3 15.3 A simple alarm system – stage 2

15.4 15.4 A simple alarm system – stage 3

15.5 15.5 A simple alarm system – stage 4

15.6 15.6 More complex alarm system

15.7 15.7 Alarm unit algorithm 5



# Глава 16

## 16 Basic DC circuit theory

16.1 16.1 Conventional Current

16.2 16.2 Ground

16.3 16.3 Preferred resistor values

16.4 16.4 Resistor Tolerances

16.5 16.5 Combining resistors in series

16.6 16.6 Combining resistors in parallel

16.7 16.7 Resistor Combination Circuits



# Глава 17

## 17 Basic project planning

17.1 17.1 System Designer

17.2 17.2 Project mind map

17.3 17.3 Project timeline

17.4 17.4 System context diagram

17.5 17.5 Block Diagram

17.6 17.6 Board Layouts

17.7 17.7 Algorithm design

# Глава 18

## 18 Example system design - hot glue gun timer

18.1 18.1 System context diagram

18.2 18.2 Hot glue gun timer block diagram

18.3 18.3 Hot glue gun timer algorithm

18.4 18.4 Hot glue gun timer flowchart

18.5 18.5 Hot glue gun timer program



# Глава 19

## 19 Basic interfaces and their programming

19.1 19.1 Parallel data communications

19.2 19.2 LCDs (liquid crystal displays)

19.3 19.3 Alphanumeric LCDs

19.4 19.4 ATTINY461 Development PCB with LCD

19.5 19.5 Completing the wiring for the LCD

19.6 19.6 LCD Contrast Control

19.7 19.7 Learning to use the LCD



# Глава 20

## 20 Basic analog to digital interfaces

20.1 20.1 ADC - Analog to Digital conversion

20.2 20.2 Light level sensing

20.3 20.3 Voltage dividers review

20.4 20.4 AVR ADC connections

20.5 20.5 Select-Case

20.6 20.6 Reading an LDR's values

20.7 20.7 Marcus' year10 night light project



# Глава 21

## 21 Basic System Design

- 21.1 21.1 Understanding how systems are put together
- 21.2 21.2 Food Processor system block diagram
- 21.3 21.3 Subsystems
- 21.4 21.4 Food Processor system functional attributes - algorithm
- 21.5 21.5 Food Processor system flowchart
- 21.6 21.6 Toaster Design
- 21.7 21.7 Toaster - system block diagram



## Глава 22

# 22 Basic System development - Time Tracker

22.1 22.1 System context diagram and brief

22.2 22.2 Time tracker block diagram

22.3 22.3 Algorithm development

22.4 22.4 Schematic

22.5 22.5 Time tracker flowchart and program version 1

22.6 22.6 Time Tracker stage 2

22.7 22.7 Time Tracker stage 3

# Глава 23

## 23 Basic maths time

23.1 23.1 Ohms law calculator

23.2 23.2 more maths - multiplication

23.3 23.3 Algorithms for multiplication of very large numbers

23.4 23.4 Program ideas - algorithm and flowchart exercises

# Глава 24

## 24 Basic string variables

24.1 24.1 Strings assignment

24.2 24.2 ASCII Assignment

24.3 24.3 Time in a string

24.4 24.4 Date in a string

24.5 24.5 Scrolling message assignment

24.6 24.6 Some LCD programming exercises



# Глава 25

## 25 Advanced power interfaces

25.1 25.1 Microcontroller power limitations

25.2 25.2 Power

25.3 25.3 Power dissipation in resistors

25.4 25.4 Diode characteristics

25.5 25.5 Using Zener diodes

25.6 25.6 How diodes work

25.7 25.7 How does a LED give off light?



# Глава 26

## 26 Advanced Power Supply Theory

26.1 26.1 Typical PSUs

26.2 26.2 The four stages of a PSU (power supply unit)

26.3 26.3 Stage 1: step down transformer

26.4 26.4 Stage 2: AC to DC Conversion

26.5 26.5 Stage 3: Filtering AC component

26.6 26.6 Stage 4: Voltage Regulation

26.7 26.7 Ripple (decibel & dB)

## Глава 27

27 Year11/12/13 typical test questions so far

# Глава 28

## 28 Advanced programming -arrays

Глава 29

29 AVR pull-up resistors

# Глава 30

## 30 AVR pull-up resistors



# Глава 31

## 31 Advanced programming - state machines

31.1 31.1 Daily routine state machine

31.2 31.2 Truck driving state machine

31.3 31.3 Developing a state machine

31.4 31.4 A state machine for the temperature alarm system

31.5 31.5 Using System Designer software to design state machines

31.6 31.6 State machine to program code

31.7 31.7 The power of state machines over flowcharts

# Глава 32

## 32 Advanced keypad interfacing

32.1 32.1 Keypad program 1

32.2 32.2 Keypad program 2

32.3 32.3 Keypad program 3 – cursor control

32.4 32.4 Keypad texter program V1

32.5 32.5 Keypad texter program 1a

32.6 32.6 ADC keypad interface

# Глава 33

## 33 Do-Loop & While-Wend subtleties

### 33.1 33.1 While-Wend or Do-Loop-Until or For-Next?



# Глава 34

## 34 DC Motor interfacing

34.1 34.1 H-Bridge

34.2 34.2 H-Bridge Braking

34.3 34.3 L293D H-Bridge IC

34.4 34.4 L298 H-Bridge IC

34.5 34.5 LMD18200 H-Bridge IC

34.6 34.6 LMD18200 program

34.7 34.7 Darlington H-Bridge

# Глава 35

## 35 Advanced System Example – Alarm Clock

35.1

35.2 35.2 Analogue seconds display on an LCD

35.3 35.3 LCD big digits

# Глава 36

## 36 Resistive touch screen

36.1 36.1 Keeping control so you dont lose your 'stack'

# Глава 37

## 37 System Design Example – Temperature Controller

# Глава 38

## 38 Alarm clock project re-developed

- 38.1 38.1 System Designer to develop a Product Brainstorm
- 38.2 38.2 Initial block diagram for the alarm clock
- 38.3 38.3 A first (simple) algorithm is developed
- 38.4 38.4 A statemachine for the first clock
- 38.5 38.5 Alarm clock state machine and code version 2
- 38.6 38.6 Token game – state machine design example

# Глава 39

## 39 Advanced window controller student project

39.1 39.1 Window controller state machine #1

39.2 39.2 Window controller state machine #3

39.3 39.3 Window controller state machine #5

39.4 39.4 Window controller program

# Глава 40

## 40 Alternative state machine coding techniques



# Глава 41

## 41 Complex — serial communications

41.1 41.1 Simplex and duplex

41.2 41.2 Synchronous and asynchronous

41.3 41.3 Serial communications, Bascom and the AVR

41.4 41.4 RS232 serial communications

41.5 41.5 Build your own RS232 buffer

41.6 41.6 Talking to an AVR from Windows XP

41.7 41.7 Talking to an AVR from Win7



# Глава 42

## 42 Radio Data Communication

42.1 42.1 An Introduction to data over radio

42.2 42.2 HT12E Datasheet, transmission and timing

42.3 42.3 HT12 test setup

42.4 42.4 HT12E Program

42.5 42.5 HT12D datasheet

42.6 42.6 HT12D Program

42.7 42.7 Replacing the HT12E encoding with software



# Глава 43

## 43 Introduction to I2C

43.1 43.1 I2C Real Time Clocks

43.2 43.2 Real time clocks

43.3 43.3 Connecting the RTC

43.4 43.4 Connecting the RTC to the board

43.5 43.5 Internal features

43.6 43.6 DS1307 RTC code

43.7 43.7 DS1678 RTC code

# Глава 44

## 44 Plant watering timer student project

44.1 44.1 System block diagram

44.2 44.2 State machine

44.3 44.3 Program code

Глава 45

45 Bike audio amplifier project

# Глава 46

## 46 Graphics LCDs

46.1 46.1 The T6963 controller

46.2 46.2 Graphics LCD (128x64) — KS0108

46.3 46.3 Generating a negative supply for a graphics LCD



# Глава 47

## 47 GLCD Temperature Tracking Project

47.1 47.1 Project hardware

47.2 47.2 Project software planning

47.3 47.3 Draw the graph scales

47.4 47.4 Read the values

47.5 47.5 Store the values

47.6 47.6 Plot the values as a graph

47.7 47.7 Full software listing



# Глава 48

## 48 Interrupts

48.1 48.1 Switch bounce problem investigation

48.2 48.2 Keypad- polling versus interrupt driven

48.3 48.3 Improving the HT12 radio system by using interrupts

48.4 48.4 Magnetic Card Reader

48.5 48.5 Card reader data structure

48.6 48.6 Card reader data timing

48.7 48.7 Card reader data formats

# Глава 49

## 49 Timer/Counters

49.1 49.1 Timer2 (16 bit) Program

49.2 49.2 Timer0 (8 bit) Program

49.3 49.3 Accurate tones using a timer (Middle C)

49.4 49.4 Timer1 Calculator Program

49.5 49.5 Timer code to make a siren by varying the preload value

# Глава 50

## 50 LED dot matrix scrolling display project — arrays and timers

50.1 50.1 Scrolling text code

50.2 50.2 Scrolling text — algorithm design

50.3 50.3 Scrolling test — code

# Глава 51

## 51 Medical machine project – timer implementation

51.1 51.1 Block diagram

51.2 51.2 Blower - state machine

51.3 51.3 Blower program code



# Глава 52

## 52 Multiple 7-segment clock project — dual timer action

52.1 52.1 Understanding the complexities of the situation

52.2 52.2 Hardware understanding

52.3 52.3 Classroom clock — block diagram

52.4 52.4 Classroom clock — schematic

52.5 52.5 Classroom clock — PCB layout

52.6 52.6 Relay Circuit Example

# Глава 53

## 53 The MAX 7219/7221 display driver IC's

### 53.1 53.1 AVR clock/oscillator



# Глава 54

## 54 Cellular Connectivity-ADH8066

54.1 54.1 ADH prototype development

54.2 54.2 ADH initial test setup block diagram

54.3 54.3 Process for using the ADH

54.4 54.4 ADH communications

54.5 54.5 Initial state machine

54.6 54.6 Status flags

54.7 54.7 Second state machine



# Глава 55

## 55 Data transmission across the internet

55.1 55.1 IP address

55.2 55.2 MAC (physical) address

55.3 55.3 Subnet mask

55.4 55.4 Ping

55.5 55.5 Ports

55.6 55.6 Packets

55.7 55.7 Gateway



# Глава 56

## 56 Assignment – maths in the real world

56.1 56.1 Math assignment - part 1

56.2 56.2 Math assignment - part 2

56.3 56.3 Math assignment - part 3

56.4 56.4 Math assignment - part 4

56.5 56.5 Math assignment - part 5

56.6 56.6 Math assignment - part 6

56.7 56.7 Extension exercise



# Глава 57

## 57 SSD1928 based colour graphics LCD

57.1 57.1 System block diagram

57.2 57.2 TFT LCDs

57.3 57.3 System memory requirements

57.4 57.4 System speed

57.5 57.5 SSD and HX ICs

57.6 57.6 Colour capability

57.7 57.7 SSD1928 and HX8238 control requirements

## Глава 58

### 58 Traffic Light help and solution



# Глава 59

## 59 Computer programming — low level detail

59.1 59.1 Low level languages:

59.2 59.2 AVR Internals — how the microcontroller works

59.3 59.3 1. The 8 bit data bus

59.4 59.4 2. Memory

59.5 59.5 3. Special Function registers

59.6 59.6 A simple program to demonstrate the AVR in operation

59.7 59.7 Bascom keyword reference

## Глава 60

### 60 USB programmer — USBASP

# Глава 61

## 61 USBTinyISP programmer



# Глава 62

## 62 C-Programming and the AVR

62.1 62.1 Configuring a programmer

62.2 62.2 First program

62.3 62.3 Output window

62.4 62.4 Configuring inputs & outputs

62.5 62.5 Making a single pin an input

62.6 62.6 Making a single pin an output

62.7 62.7 Microcontroller type



# Глава 63

## 63 Object Oriented Programming (OOP) in CPP and the AVR

63.1 63.1 The black box concept

63.2 63.2 The concept of a class

63.3 63.3 First CPP program

63.4 63.4 Creating an AVR CPP program in Atmel Studio 6

63.5 63.5 Adding our class files to the project

63.6 63.6 First Input and output program



# Глава 64

## 64 Current (2014) AVR development PCBS

64.1 64.1 Year 9 ATTiny 45 Board

64.2 64.2 Year10 ATTiny461 V4a development board

64.3 64.3 Year11 ATMega48 (or 88 or 168 or 328) V4

64.4 64.4 Year11 ATMega48 (or 88 or 328) v3 development board

64.5 64.5 Year 12 ATMega 20x4 Character LCD v6A

64.6 64.6 Year 13 ATMega GLCD 128x64 (2014) Veroboard

64.7 64.7 Year 13 ATMega GLCD (older pin connections)

# Глава 65

## 65 Eagle — creating your own library

### 65.1 65.1 Autorouting PCBs



# Глава 66

## 66 Practical Techniques

66.1 66.1 PCB Mounting

66.2 66.2 Countersink holes and joining MDF/wood

66.3 66.3 MDF

66.4 66.4 Plywood

66.5 66.5 Acrylic

66.6 66.6 Electrogalv

66.7 66.7 Choosing fasteners



# Глава 67

## 67 CNC

67.1 67.1 Machine overview

67.2 67.2 Starting the CNC machine

67.3 67.3 CamBam

67.4 67.4 CamBam options

67.5 67.5 Drawing shapes in CamBam

67.6 67.6 Machining commands

67.7 67.7 A Box of Pi

# Index

## Часть I

# Рабочая среда разработчика встраиваемых систем

## **67.11 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)

## **67.12 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

## **67.13 Пакет расчета и симуляции электронных схем: SPICE**

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

## **67.14 САПР общего назначения**

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

## **67.15 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](#))

## 67.16 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.

## 67.17 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).

## 67.18 Симулятор для отладки программ без железа

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

## **67.19 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 II

# 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 68

Make

# Глава 69

## binutils

# Глава 70

# Формат объектного файла GNU ELF

формат файла elf32-i386

архитектура i386

**HAS\_SYMS** в файл включена отладочная информация об идентификаторах (“символах”)

start address 0x00100000

стартовый адрес загрузки ядра 1 Мб

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

.text            машинный код программы

## .rodata      данные: константы

.eh frame

.data        данные: инициализированные массивы, строки

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

## comment

CONTENTS

ALLOC

LOAD

READONLY

CODE

DATA

# Глава 71

## Ассемблер GNU `as`



# Глава 72

## Линкер GNU ld

### 72.1 Использование ld

### 72.2 Обзор

### 72.3 Вызов

#### 72.3.1 Параметры командной строки

#### 72.3.2 Опции, специфичные для целевой платформы i386 РЕ

#### 72.3.3 Переменные среды

### 72.4 Скрипты линкера

The format of the SECTIONS command is:

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

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

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

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

Output section descriptions and overlay descriptions are described below.

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

## 72.10.1 Описание выходных секций

The full description of an output section looks like this:

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

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

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

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

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

## 72.10.2 Имя выходной секции

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

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

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



- 72.10.3 Описание выходных секций
- 72.10.4 Описание входных секций
- 72.10.5 Выходная секция данных
- 72.10.6 Ключевые слова выходных секций
- 72.10.7 Отброс выходных секций
- 72.10.8 Атрибуты выходных секций
- 72.10.9 Описание оверлея
- 72.11 Команда MEMORY
- 72.12 Команда PHDRS
- 72.13 Команда VERSION
- 72.14 Выражения в скриптах компоновщика
  - 72.14.1 Константы
  - 72.14.2 Имена символов
  - 72.14.3 Счетчик адреса
  - 72.14.4 Операторы

## Глава 73

**objdump**

## Глава 74

# GNU Compiler Collection (GCC)

Глава 75

Компилятор GNU C

# Глава 76

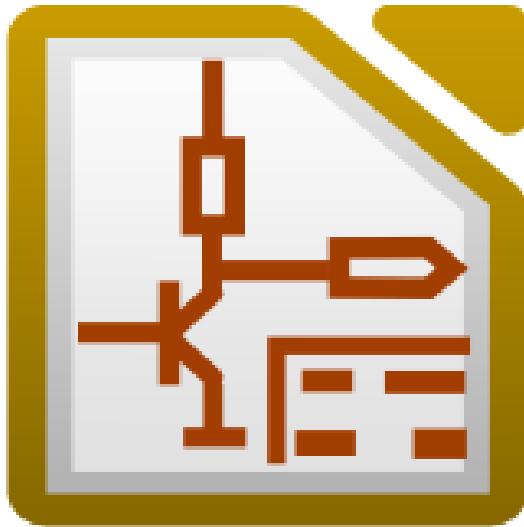
## Компилятор GNU C++

Глава 77

Компилятор GNU Fortran

## Часть III

САПР электронных устройств KiCAD



1

2

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

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

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

3

---

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

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

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

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

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

# Глава 78

## Установка

### 78.1 Windows

  
Installer Language > English > Ok в русифицированном инсталляторе кривые шрифты  
KiCAD 20xx.xx.xx Setup > Next  
Лицензия > Agree  
Components >  все > Next  
Location > C:/KiCad > Install  
Completing Setup >  Wings3D > Finish

## 78.2 Linux

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

+++ ~/.blackbox/menu

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

## 78.3 Настройка библиотек

Для добавления библиотек, поставляемых с этой книгой, сделайте *git clone* или *git pull*:

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



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

## 78.4 Дотфайлы

Посколько программа изначально писалась как юниксовая, пользовательские настройки хранятся в dot-файлах:

/.kicad

```
1 KicadFramePos_x=9
2 KicadFramePos_y=35
3 KicadFrameSize_x=843
4 KicadFrameSize_y=505
5 KicadFrameMaximized=0
6 LeftWinWidth=178
7 ShowPageLimits=1
8 WorkingDir=/home/ponyatov/deep
9 BgColor=15
10 file1=/home/ponyatov/deep/deep.pro
11 file2=
12 file3=
13 file4=
14 file5=
15 file6=
16 file7=
17 file8=
18 file9=
```

**KicadFrame\*** размеры и положение окна менеджера проектов

**WorkingDir** каталог с текущим рабочим проектом

/.eeschema

```
1 SchematicFramePos_x=203
2 SchematicFramePos_y=130
3 SchematicFrameSize_x=600
4 SchematicFrameSize_y=400
5 SchematicFrameMaximized=0
6 SchematicFrameAutoSaveInterval=600
7 SchematicFrameCursorShape=0
8 SchematicFrameShowGrid=1
9 SchematicFrameGridColor=8
10 SchematicFrame_LastGridSize=5
11 ColWire=2
12 ColorBus=1
13 ColorConn=2
14 ColorLlab=0
15 ColorHlab=6
16 ColorGllab=4
17 ColorPinF=5
18 ColPinN=4
19 ColorPNam=3
20 ColorField=5
21 ColorRef=3
22 ColorValue=3
23 ColorNote=9
24 ColorBody=4
25 ColorBodyBg=23
```

```
26 ColorNetN=8
27 ColorPin=4
28 ColorSheet=5
29 ColorSheetFileName=6
30 ColorSheetName=3
31 ColorSheetLab=6
32 ColorNoCo=1
33 ColorErcW=2
34 ColorErcE=4
35 ColorGrid=8
36 PrintMonochrome=1
37 PrintSheetReferenceAndTitleBlock=1
38 DefaultDrawLineWidth=6
39 ShowHiddenPins=0
40 HorizVertLinesOnly=1
41 PreviewFramePositionX=-1
42 PreviewFramePositionY=-1
43 PreviewFrameWidth=-1
44 PreviewFrameHeight=-1
45 PrintDialogPositionX=-1
46 PrintDialogPositionY=-1
47 PrintDialogWidth=-1
48 PrintDialogHeight=-1
49 SpiceUseNetNames=0
50 SimCmdLine=
51 FindDialogPositionX=-1
52 FindDialogPositionY=-1
53 FindDialogWidth=-1
```

```
54 FindDialogHeight=-1
55 LastFindReplaceFlags=1
56 LastFindString=
57 LastReplaceString=
58 FieldNames=(templatefields)
59 AutoPAN=1
60 ShowPageLimits=1
61 WorkingDir=/home/ponyatov/deep
62 BgColor=15
63 file1=/home/ponyatov/deep/deep.sch
64 file2=
65 file3=
66 file4=
67 file5=
68 file6=
69 file7=
70 file8=
71 file9=
```

**SchematicFrame\*** размеры и положение окна eeschema

fileN список последних схем ( **Файл >> Последние файлы** )

## 78.5 Глобальные шаблоны

После установки **KiCAD** можно скорректировать файлы глобальных шаблонов, чтобы новые проекты создавались сразу с нужными настройками, прежде всего с нужным нам набором библиотек:

```
sudo vim /usr/share/kicad/template/kicad.pro
```

/usr/share/kicad/template/kicad.pro

```
1 update=19/09/2011 08:20:40
2 version=1
3 last_client=pcbnew
4 [general]
5 version=1
6 RootSch=
7 BoardNm=
8 [eeschema]
9 version=1
10 LibDir=/home/ponyatov/odurino/lib
11 NetFmt=1
12 HPGLSpd=20
13 HPGLDm=15
14 HPGLNum=1
15 offX_A4=0
16 offY_A4=0
17 offX_A3=0
18 offY_A3=0
19 offX_A2=0
20 offY_A2=0
21 offX_A1=0
22 offY_A1=0
23 offX_A0=0
24 offY_A0=0
25 offX_A=0
```

```
26 offY_A=0
27 offX_B=0
28 offY_B=0
29 offX_C=0
30 offY_C=0
31 offX_D=0
32 offY_D=0
33 offX_E=0
34 offY_E=0
35 RptD_X=0
36 RptD_Y=100
37 RptLab=1
38 LabSize=60
39 [eeschema/libraries]
40 LibName1=R
41 LibName2=L
42 LibName3=C
43 LibName4=POWER
44 LibName5=DA_POWER
45 LibName6=FET
46 LibName7=SPICE
47 LibName8=VD
48 LibName9=SWITCH
49 LibName10=FTDI
50 LibName11=Atmel
51 LibName12=MAXIM
52 LibName13=STMicro
53 [cvpcb]
```

```
54 version=1
55 NetExt=net
56 [ cvpcb / libraries ]
57 EquName1=devcms
58 [ pcbnew ]
59 version=1
60 PadDrlX=320
61 PadDimH=600
62 PadDimV=600
63 BoardThickness=630
64 TxtPcbV=800
65 TxtPcbH=600
66 TxtModV=600
67 TxtModH=600
68 TxtModW=120
69 VEgarde=100
70 DrawLar=150
71 EdgeLar=150
72 TxtLar=120
73 MSegLar=150
74 LastNetListRead=
75 [ pcbnew / libraries ]
76 LibDir=/home/ponyatov/odurino/lib
```

**LibDir** каталог библиотек, установите на свой или корпоративный/групповой

**LibNameN** задается список библиотек по умолчанию, приопишите ваш типовой набор

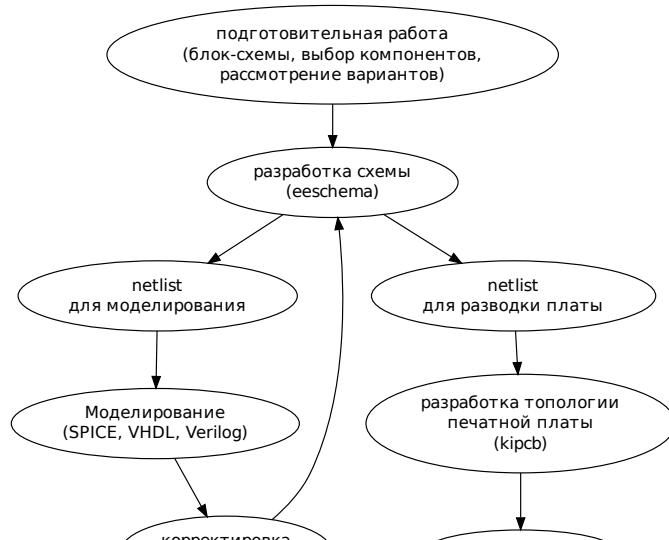
**eeschema/libraries** схемные библиотеки **eeschema**

`pcbnew/libraries` библиотеки надстеков для печатных плат `pcbnew`



# Глава 79

## Маршрут проектирования

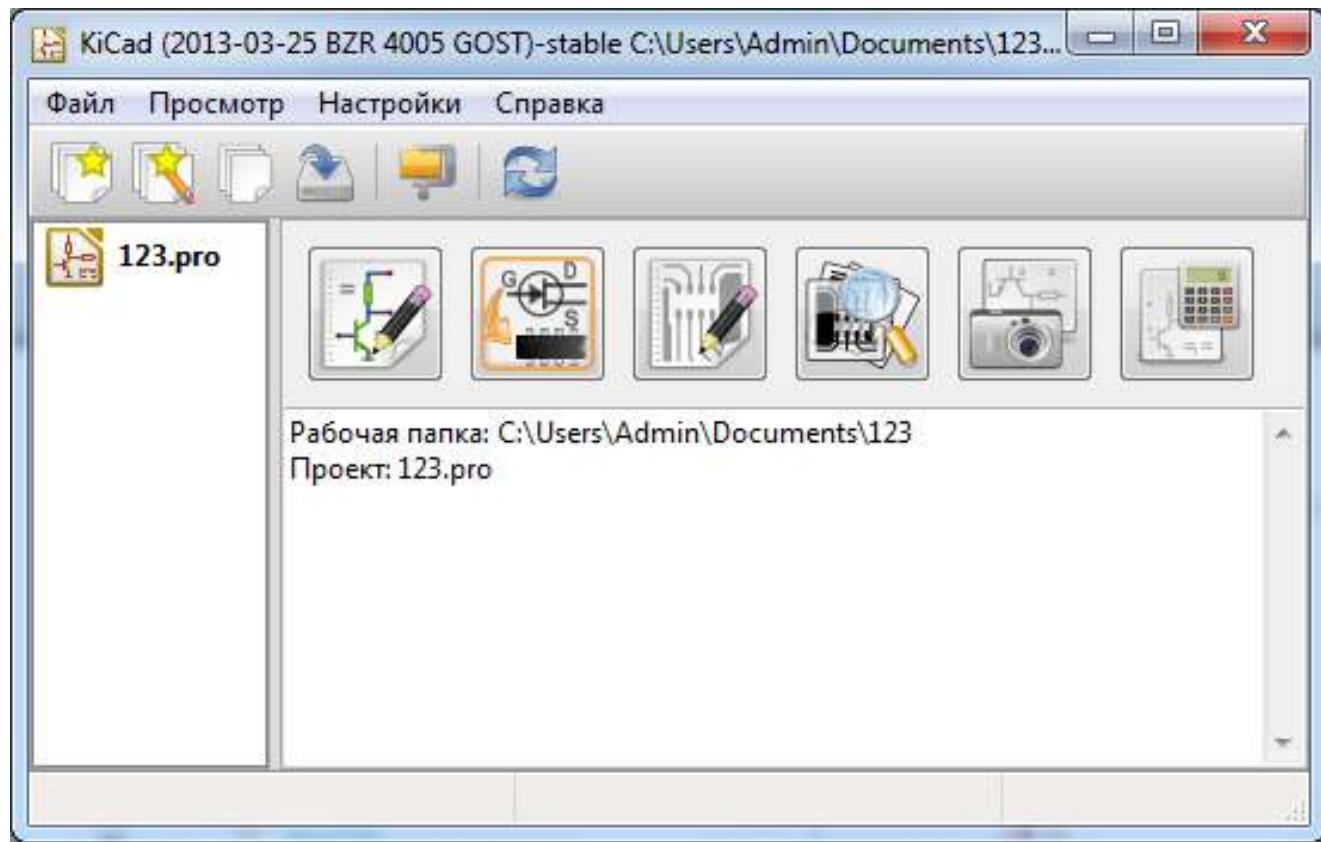


# Глава 80

## Создание проекта в менеджере проектов **kicad**

Windows:  Практически все версии Windows: Пуск > Программы > KiCAD > KiCAD

Linux: user@host\$ kicad



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

-  **eeschema** — Редактор принципиальных схем



- **pcbnew** — Редактор печатных плат



- **cvpcb** — Программа редактирования **падстеков** (отверстий и площадок)

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

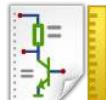


- **gerbview** — Программа просмотра фотошаблонов в формате Gerber

- **bitmap2component** — Создание компонента из черно-белого изображения (например логотипа)



- **PcbCalculator** — Калькулятор для печатных плат



- **PageLayout** — редактор формата листа схемы

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



1. Запустите программу KiCad



## 2. Создайте новый проект

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

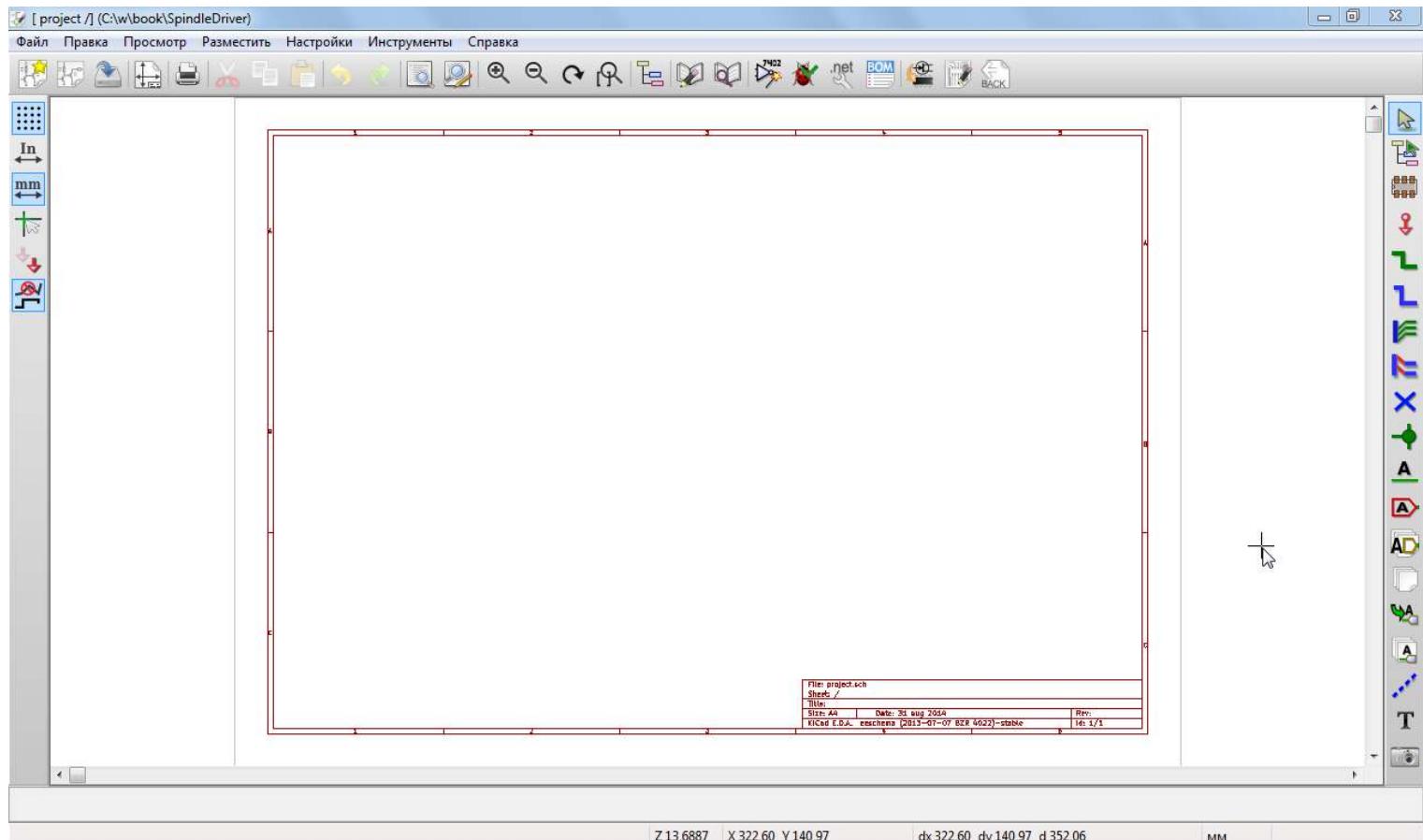
# Глава 81

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

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

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





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

# Глава 82

## **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).

# Глава 83

## Библиотеки компонентов

1

Модель компонента в САПР EDA состоит из следующих частей:

- условное графическое обозначение (УГО) для схемного редактора
- модель компонента для редактора печатных плат (ПП)
- модель для симулятора (SPICE)
- 3D модель для передачи в универсальный САПР для работы с конструкцией
- дополнительные пользовательские данные: индексы компонента для заказа у разных поставщиков, ссылки на документацию, и т.п.

---

<sup>1</sup> копипаста: <http://habrahabr.ru/post/197582/>

Части могут иметь несколько вариантов, например два варианта УГО (ГОСТ и ISO), три корпуса (DIP, PLCC и LQFP), две модели для симулятора (идеальная и с учетом паразитных эффектов), и 2 механических модели (габаритный кубик, и подробная).

Кроме того, часто в один корпус упаковывается несколько одинаковых или разных элементов. Одинаковые — 2–4 операционных усилителя (ОУ), или вентили логики. Разные — части вакуумной лампы, разнесенные на схеме по разным каскадам.

В составе KiCad поставляются библиотеки электронных компонентов (обычных и поверхностно монтируемых SMD). Для многих библиотечных компонентов есть 3D-модели, созданные в [Wings3D](#).

Но как только вы начинаете работать со свежеустановленным KiCADом, тут же обнаруживается, что библиотечные компоненты или не подходят<sup>2</sup>, или нужных компонентов попросту нет в библиотеках.

Рассмотрим последовательное создание совершенно нового элемента на примере модуля USB интерфейса HEX\_FT2232RL ??

## 83.1 Создание УГО для схем

Нам необходим встроенный редактор символов схем (библиотечных компонентов), запускаем его следующим образом:

### 1. Вначале запускаем eeschema

вверху меню и панель инструментов

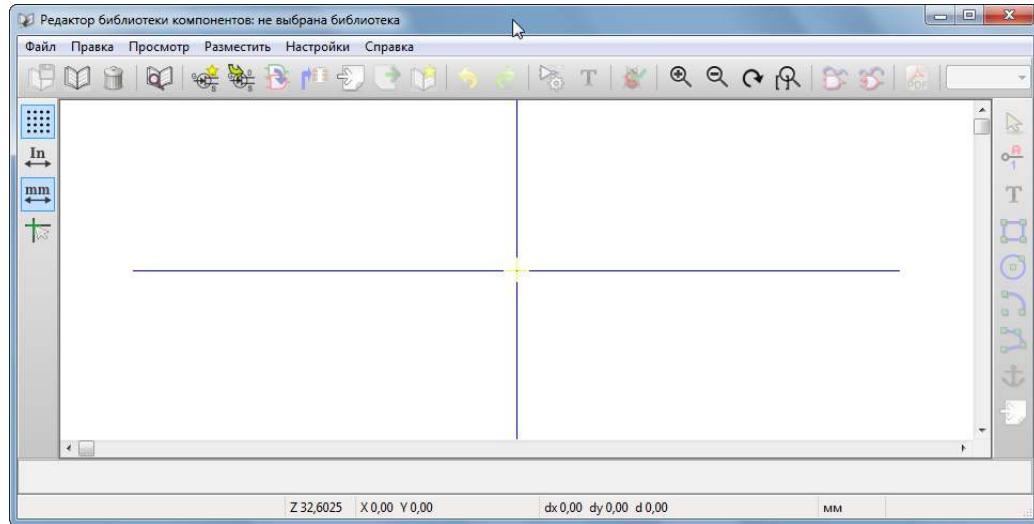
слева область размерности и шага сетки редактора (настройка рабочей области)

справа область элементов схем и перемещения по иерархии схемы

---

<sup>2</sup> например не соответствуют ГОСТ или стандартам предприятия

2. Далее запускаем встроенный Редактор библиотек



Необходимо создать новую библиотеку и первый собственный компонент:

- Создать новый компонент 
- Свойства компонента  Общие Настройки
- Имя компонента  HEX\_FT232RL
- Обозначение по умолчанию  U
- Количество элементов в корпусе  1
- OK

В верхней панели инструментов активировались несколько кнопок, выбираем

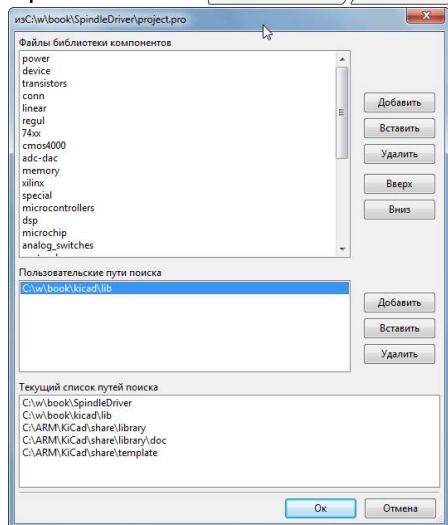


Сохранить текущий компонент в новой библиотеке

В открывшемся диалоге выберите каталог для библиотеки `/home/user/kicad/`, и укажите имя файла (новой) библиотеки **MyModules** **Сохранить**.

Теперь нужно добавить созданную библиотеку в рабочий список.

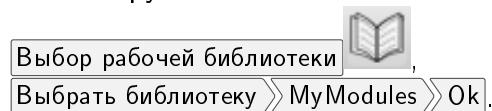
Настроим дополнительный путь, где лежат файлы библиотек. Это могут быть ваши личные библиотеки, специальная библиотека для конкретного проекта, или комплект библиотек поставляемых вместе с этой книгой: выберите в меню **Настройки** **Библиотека**, **Пользовательские пути поиска** **добавить** `/home/user/kicad/`, **Ok**.



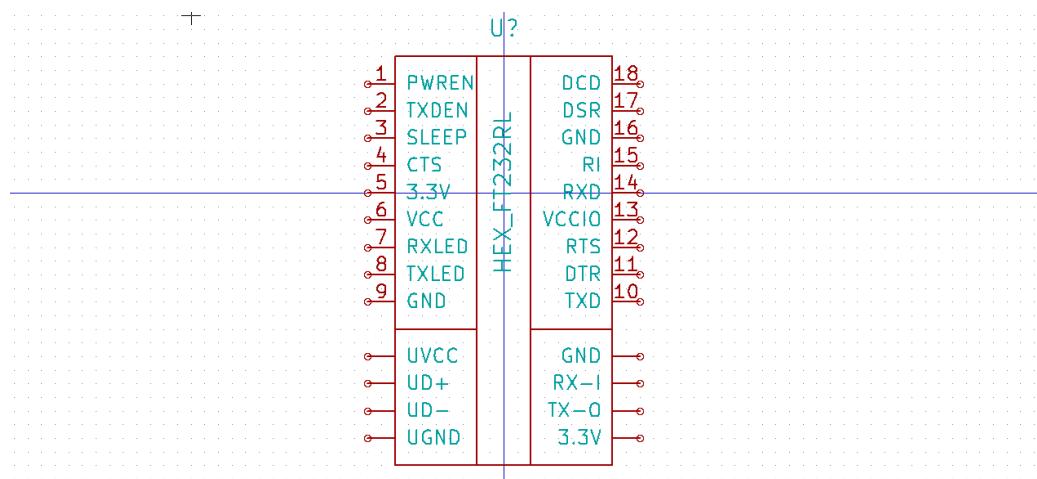
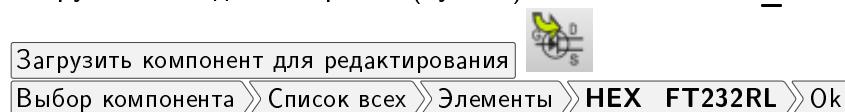
**Настройки** **Библиотека**, **Файлы библиотеки компонентов** `power` **▷ Вставить**. Выбираем только что созданную библиотеку **MyModules**. Она будет вставлена в список до выбранной **power**.

Проделанные настройки применяются только к текущему проекту. Если Вы хотите чтобы новая библиотека всегда добавлялась к новым проектам, вам нужно добавить новый путь поиска библиотек и ее название в файл шаблона, как это описано в 78.5.

Если вы хотите изменить только что созданное или уже существующее УГО, нужно выбрать рабочую библиотеку, ту библиотеку в которой мы хотим работать (создавать или редактировать компоненты). На панели инструментов нажимается кнопка



Загружаем созданный ранее (пустой) компонент **HEX\_FT232RL** (или любой другой)



Сейчас у нас элемент не имеет графических элементов, и состоит только из нескольких текстовых полей с обозначениями, слепленных в одной точке. Нужно их растянуть: САПР не может различить близкие элементы и уточняет для какого поля мы хотим контекстное меню. Выбираем любое, или кнопка Перетаскиваем элемент, и на свободном месте.

Пользуясь , отрисовываем на освободившемся месте УГО элемента, пользуясь кнопками на панели справа. Рисование выполняется по сетке, шаг выбирается Выбор сетки, набор сеток фиксированный ?. При рисовании ГОСТовских УГО округляем до ближайших дюймовых размеров<sup>3</sup>, или в меньшей сетке если нужно точно гостовские размеры.

УГО имеет **точку привязки**, относительно которой отрисовываются элементы. На практике важно то, что вокруг этой точки элемент вращается. Для перемещения этой точки можно использовать кнопку



Переместить точку привязки



Добавляем выводы компонента: Добавить вывод компонента

Например для резистора создание выводов будет выглядеть вот так:

Свойства вывода
Имя  A
Номер  1
Ориентация  Вправо
Электр.тип  Пассивный
Размер шрифта  1.27мм
Длина  2.54мм
Ok

Имя B

<sup>3</sup> 1mil=1/1000 дюйма, 100mil=2.54mm=типовoy шаг DIP микросхем

Номер >> 2

Ориентация >> Влево

Ok

R?

1

A

2

B

R

Сохраняем библиотеку: Сохранить текущую библиотеку



Подтверждение >> Включая последние изменения компонента? >> Да

Подтверждение >> Компонент существует. Изменить его? >> Да

Подтверждение >> Изменить файл библиотеки ? >> Да

## 83.2 Модель печатной платы

Модель печатной платы в программе **pcbnew** состоит из нескольких **слоев** с разными функциями, для типичной двухслойной ПП:

- верх

**Front** медь верхнего слоя

**Adhes\_Front** карта нанесения клея для компонентов

**SoldP\_** карта нанесения паяльной пасты

**SilkS\_** шелкография: маркировка элементов, надписи, и т.п.

**Mask\_** паяльная маска

- низ

**Back** медь нижнего слоя

**Adhes\_Back** карта нанесения клея для компонентов

- контуры

**Контур ПП** слой определяющий физ.геометрию платы, используется при фрезеровке контуров, сверлении монтажных отверстий, разделке сверловкой и т.п.

**Пояснения** вспомогательная текстовая информация

**Чертеж** вспомогательная графическая информация: габаритные размеры, зоны монтажа,..

- В зависимости от технологии производства и пожеланий разработчика могут добавляться любые дополнительные слои.

### 83.3 Создание падстека

Падстек — модель контактной площадки компонента, в виде набора геометрий отдельно для каждого слоя печатной платы. Также включает информацию о диаметре сверления.

## Глава 84

### **gerbview**: просмотр фотошаблонов

позволяет просматривать Gerber-файлы перед передачей печатных плат в производство.

# Глава 85

## Программа Wings3D для создания 3D моделей

Эта программа может вам пригодиться если вы планируете создавать 3D модели для PCB элементов. Архив и файлы документации (Linux и Windows) в папке kicad/wings3d.

Взгляните на домашнюю страницу Wings3D чтобы узнать подробнее о программе.

pcbnew использует файлы в формате wrml (.wrl) экспортируемые из Wings3D (родной формат Wings3D - это .wings).

### 85.1 Установка Wings3D под Windows



Location > **C:/Program Files/wings3d\_n.n.n** > Next > Install > Close

## Часть IV

Библиотека универсальных электронных  
модулей «Одурин»

Для упрощения разработки и уменьшения затрат времени на нее была разработана библиотека универсальных электронных модулей, которые могут быть использованы для сборки различных систем управления, учебных измерительных комплексов, контроллеров “Умного дома” и решения других типовых задач.

В библиотеку входят:

- библиотека компонентов для **KiCAD**
- схемы модулей
- печатные платы, разведенные с учетом технологических ограничений домашних технологий

При создании собственных устройств вы можете использовать схемы из библиотеки, подключая их как вложенные листы, при этом все элементы и связи между ними будут объединены в один нетлист для разводки платы вашего устройства.

# Глава 86

## 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»<sup>1</sup> 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.

---

<sup>1</sup> предоставить информацию о себе и своих возможностях

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 устройства<sup>2</sup>, оставляя нам самое интересное — управление функциями устройства.

Перенос кода управления устройством в `userspace` не только упрощает отладку<sup>3</sup>, но и позволяют писать процедуры управления устройством на самых разных языках программирования, а не только на Си, как это делают те, кто пишет драйверы уровня ядра. Более того, пользовательская часть драйвера, не взаимодействующая 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?

## 86.1 Стек протоколов USB

Протокол USB похож на стек сетевых протоколов, основанных на TCP/IP, точнее за основу принималась сетевая модель стека сетевых протоколов OSI/ISO<sup>4</sup>:

---

<sup>2</sup> весьма нетривиальная задача с учетом того, что сам компьютер может переключаться между несколькими энергосберегающими режимами

<sup>3</sup> при падении приложения, скорее всего, не придется перезагружать машину

<sup>4</sup> ISO/IEC 7498-1, ГОСТ Р ИСО/МЭК 7498-1-99

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

На самом нижнем логическом уровне<sup>5</sup> устройства обмениваются пакетами данных<sup>6</sup>. 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

<sup>5</sup> спецификации физического уровня мы пока не рассматриваем

<sup>6</sup> со встроенными механизмами коррекции ошибок, подтверждения получения и т.д

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.

## 86.2 libUSB

### 86.2.1 драйвер для устройства USB

7

---

<sup>7</sup> копипаста: <http://symmetrica.net/usb/usb1.htm>

## 86.3 Поддержка USB в Linux

### 86.3.1 Опции ядра

режимы host/client/otg и хост-контроллеры xHCI

data storage: носители данных

hid: клавиатура, мышь, джойстик

USB-периферия: сеть, звук,...

### 86.3.2 Настройка hotplug и автомонтирования USB носителей

### 86.3.3 Сборка и настройка libusb

### 86.3.4 Примеры программ низкоуровневого ввода/вывода

## Часть V

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 видо схемотехнических расчетов, самые важные из которых:

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

---

<sup>8</sup> копипаста: [http://www.mithatkonar.com/wiki/doku.php/kicad/kicad\\_spice\\_quick\\_guide](http://www.mithatkonar.com/wiki/doku.php/kicad/kicad_spice_quick_guide)

<sup>9</sup> копипаста: <http://physics.gmu.edu/~rubinp/courses/407/ngspice.pdf>

# Глава 87

## 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**<sup>1</sup>: Не совсем SPICE, но пытается быть синтаксически совместной.
- **SpiceOpus**: Коммерческая, но удобная, особенно в плане вывода графиков.

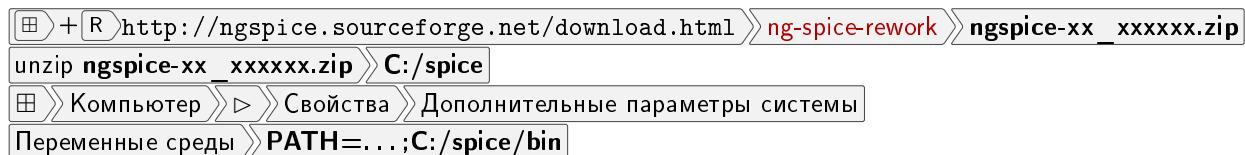
---

<sup>1</sup> уже включен в виндозную сборку KiCAD

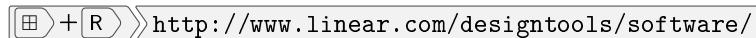
- PSpice: Windows-only, дорогой коммерческий пакет, стандарт de-facto для профессионального применения в USA в составе тяжелых EDA-продуктов. Они традиционно предоставляют обрезанную *gratis*-версию для студентов. Также имеет в комплекте GUI.
- MultiSim, OrCAD, DesignLab,.. коммерческие EDA-пакеты, содержит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

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

## 87.1 Установка ngSPICE под Windows



## 87.2 Установка LT-SPICE (только Windows)



## 87.3 Установка ngSPICE под Linux

```
root# aptitude install ngspice
```

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

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

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

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

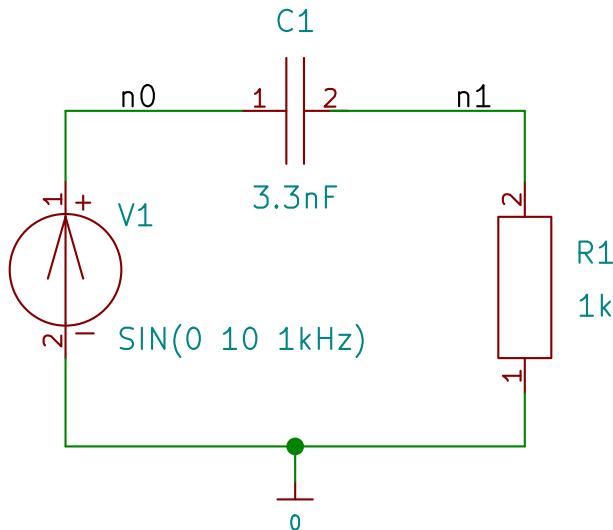
# Глава 88

## 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.

### 88.1 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 (82). Запустите [eeschema](#) и нарисуйте схему:



Простой RC-фильтр

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

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

Рисуйте вашу схему, соблюдая несколько рекомендаций:

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

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

## 88.2 Создание списка цепей

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

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

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

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

В результате экспорта будет создан файл

## RCfilter.cir

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

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

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

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

## 88.3 Запуск симуляции

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

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

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

Запускаем `ngSPICE`:

```
$ ngspice
```

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

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

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

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

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

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

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

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

```
Initial Transient Solution
```

```
-----
```

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

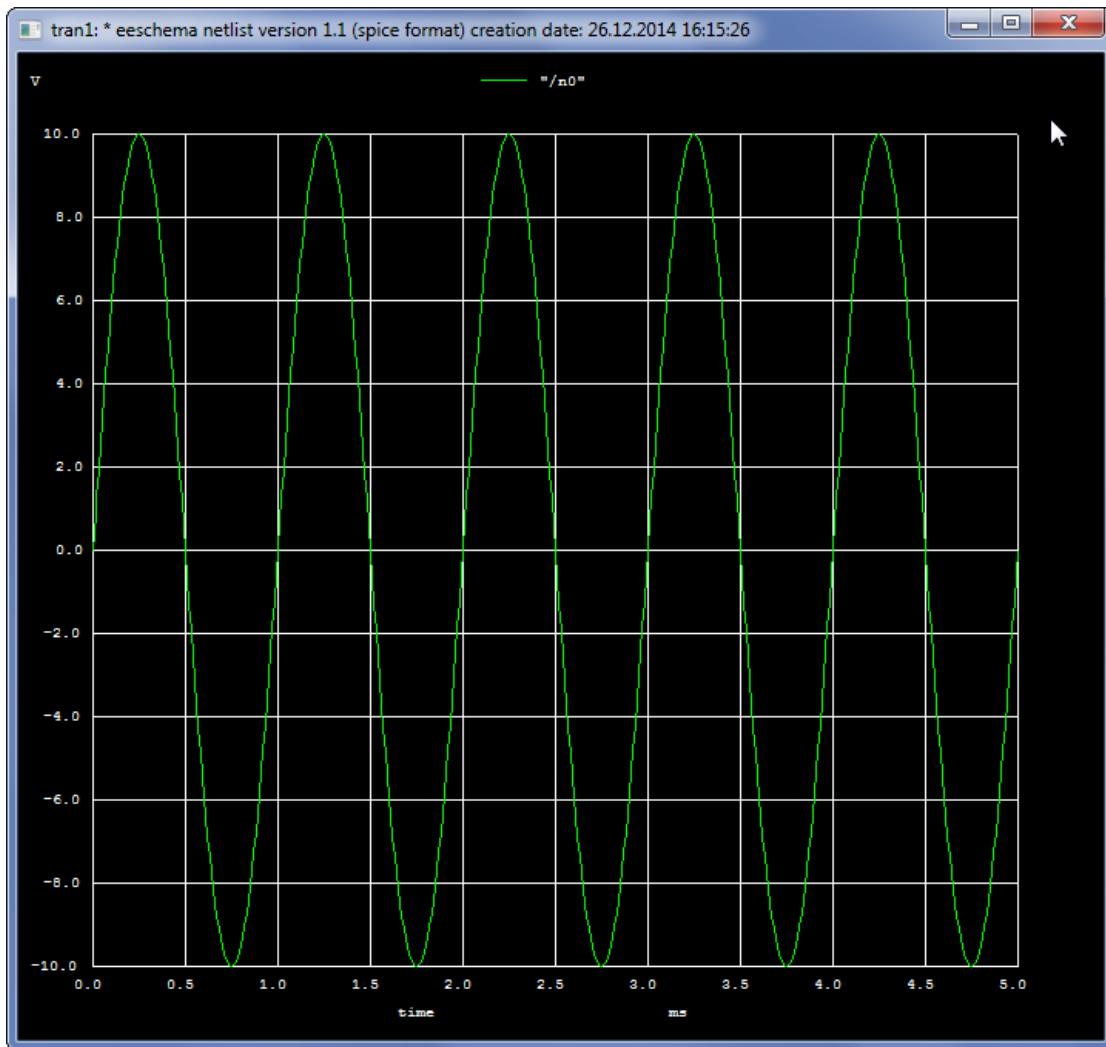
```
No. of Data Rows : 512
```

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

## 88.4 Просмотр результата расчета

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

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



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

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

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

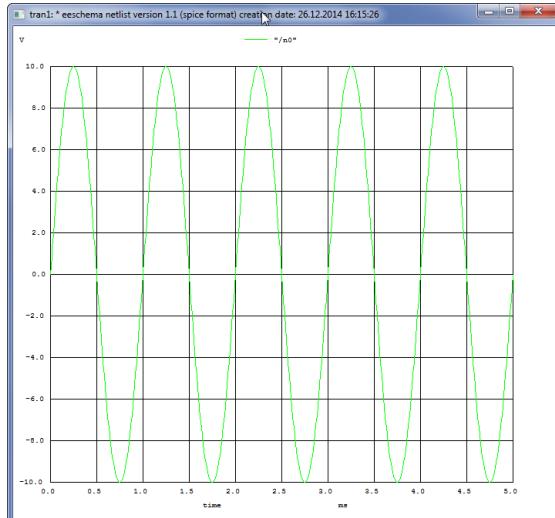
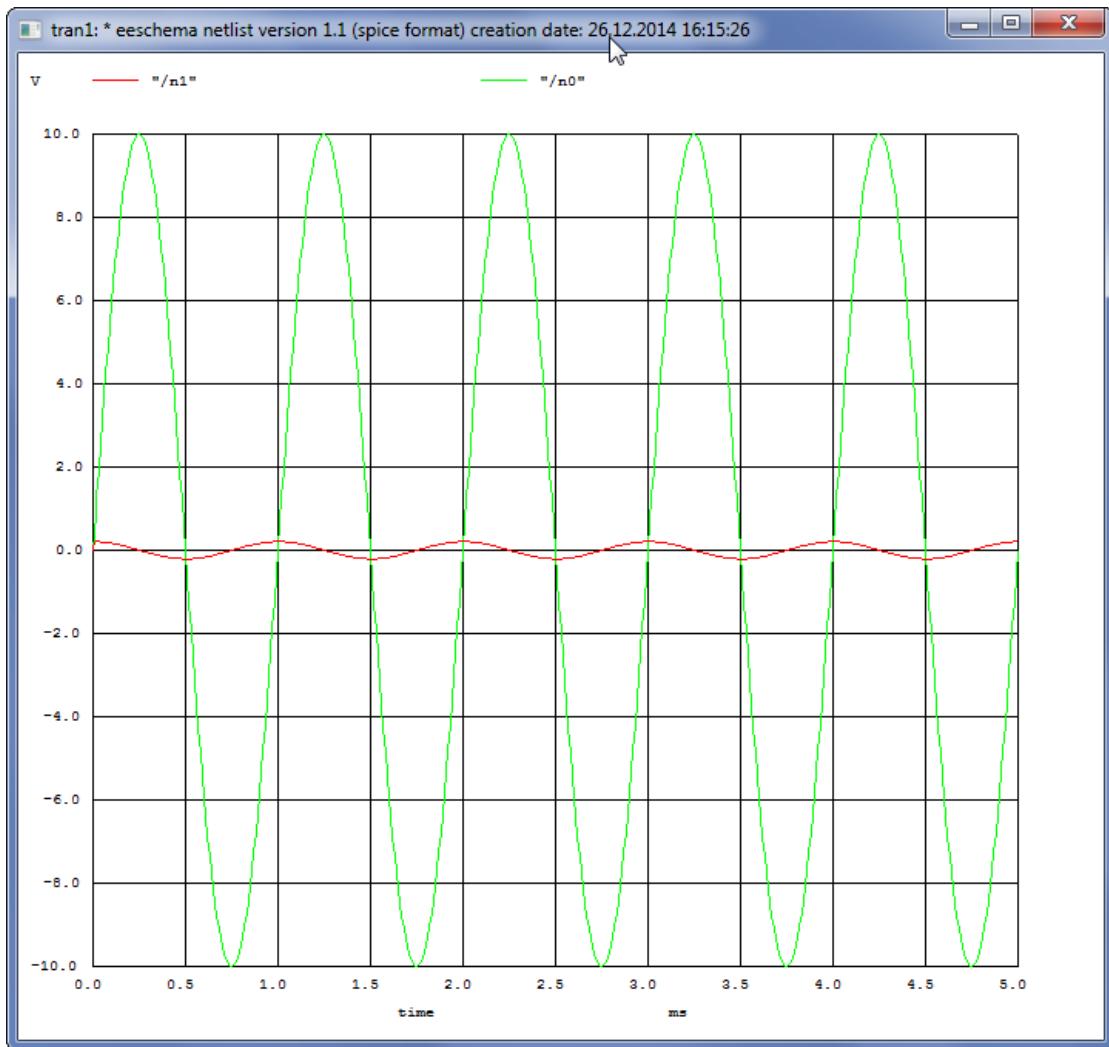


Диаграмма показывает форму входного сигнала, как мы и ожидали, но она нас мало интересует, так как мы ее и задали. Нам интереснее например *напряжение на резисторе*, кроме того мы попробуем *сравнить два сигнала*. Это легко сделать указав два имени цепи:

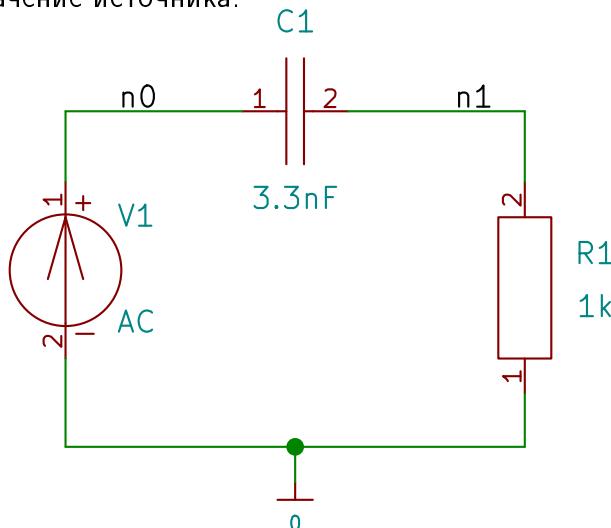
```
ngspice 31 -> plot "/n0" "/n1"
```



## 88.5 Расчет АЧХ по переменному току (AC симуляция)

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

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



Создаем нетлист, загружаем его и запускаем команду **AC анализа**:

```
$ ngspice ACanaliz.cir
```

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

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

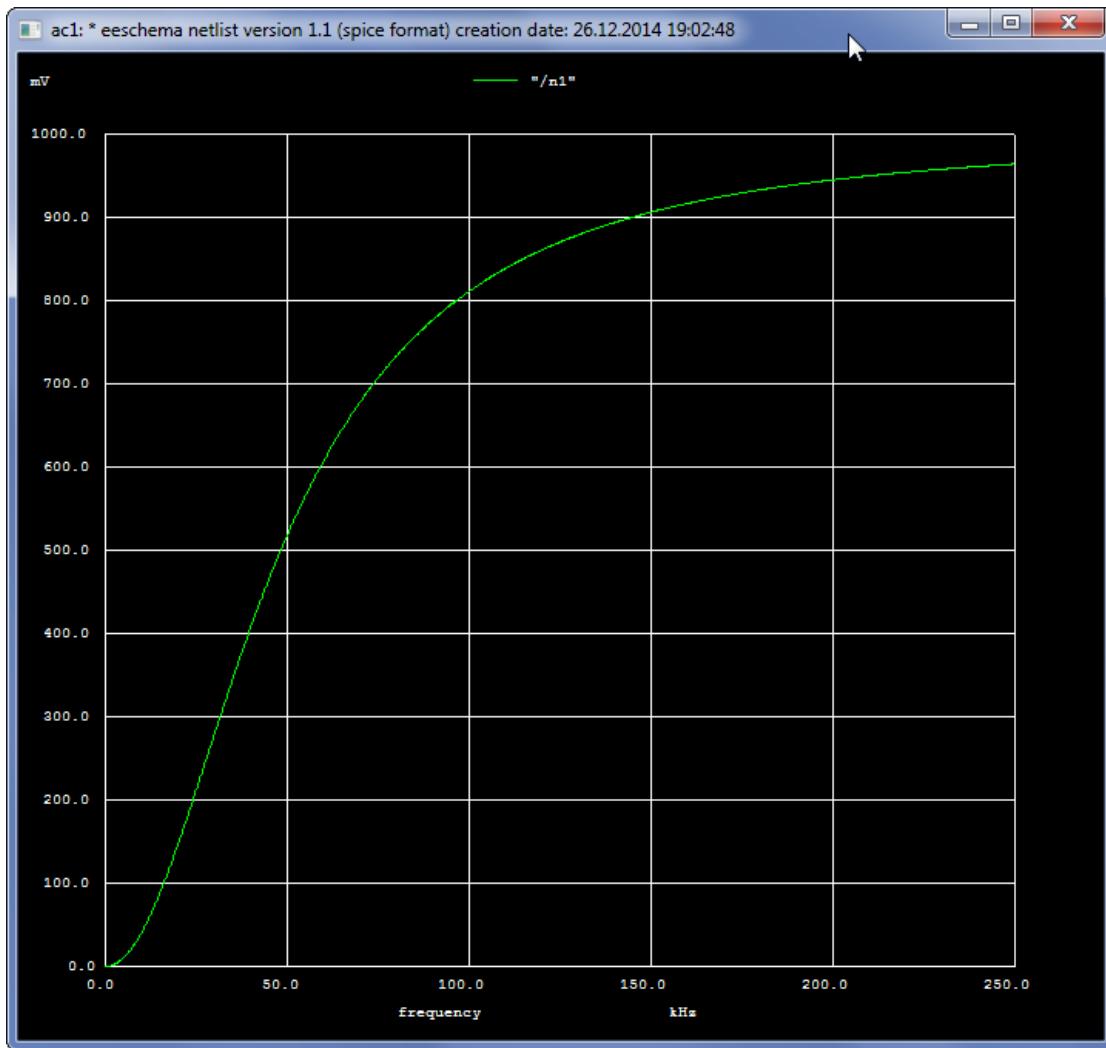
```
Warning: v1: has no value, DC 0 assumed
```

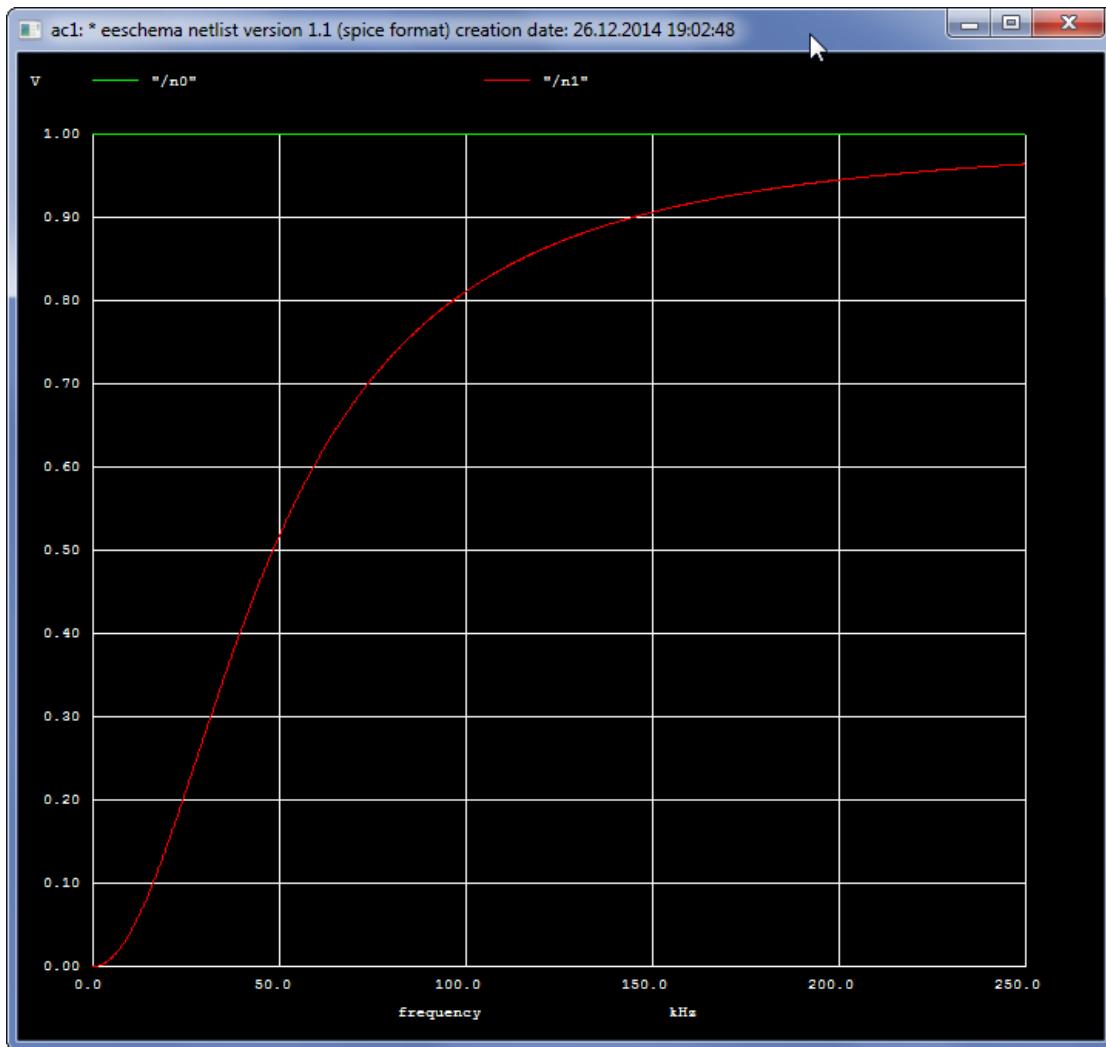
```
No. of Data Rows : 1000
```

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

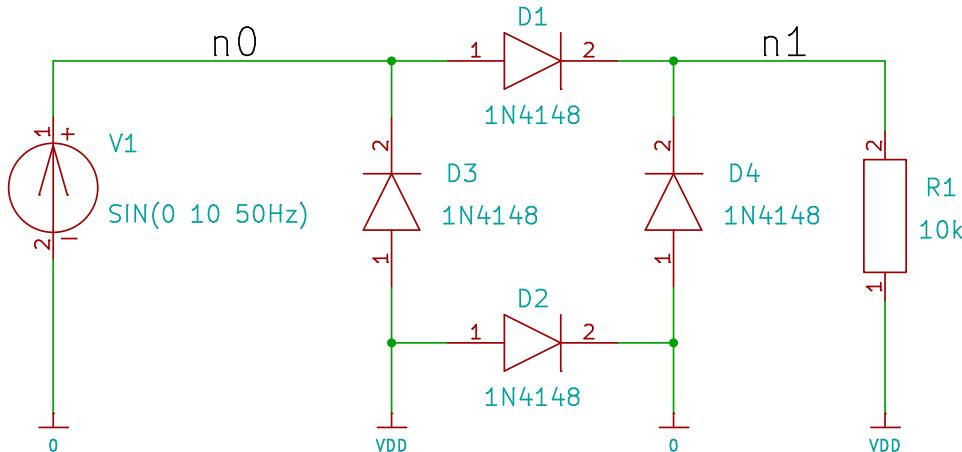
```
ngspice 3 -> plot "/n0" "/n1"
```

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





## 88.6 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
```

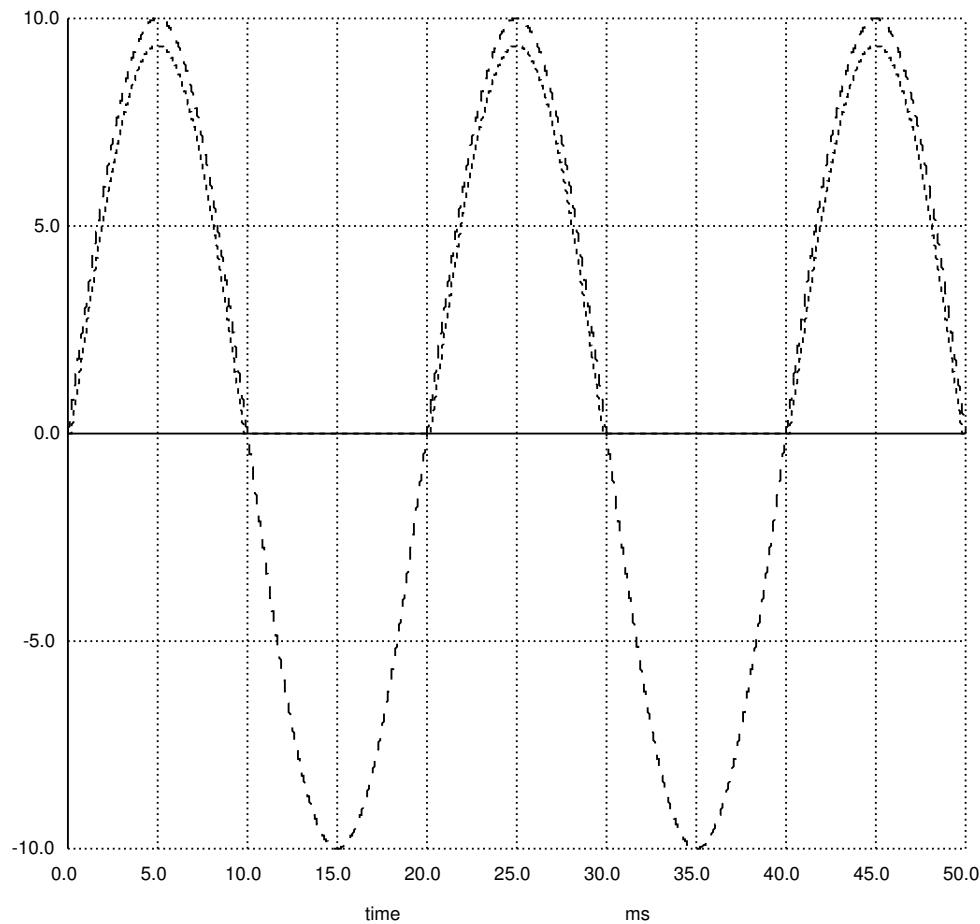
Подробнее использованные здесь приемы рисования схемы описаны в 89.

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 89

## Настройка KiCAD для SPICE-моделирования

### 89.1 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`<sup>1</sup>
- Mithat Konar <[webs@mithatkonar.com](mailto:webs@mithatkonar.com)> разрабатывает (очень медленно) собственную библиотеку с некоторыми модификациями.
- В комплекте с этой книгой поставляются библиотеки, адаптированные для SPICE.

---

<sup>1</sup> PSpice — популярная коммерческая версия SPICE

## 89.2 Настройка проекта

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

**eeshema** > Меню > Инструменты > Сформировать список цепей > Spice

Формат по умолчанию

Префикс обозначений

Использовать имена цепей

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

Список цепей

## 89.3 Как это работает

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

1 +PSPICE

2 .control

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

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

## 2. Запустите симуляцию:

eeschema > Меню > Инструменты > Сформировать список цепей > Spice  
Запустить симулятор

## Часть VI

Разработка конструкции в САПР FreeCAD



<sup>2</sup> В среде специалистов ряда отраслей известна проблема создания полноценной САПР в рамках OpenSource, и хотя FreeCAD ещё не является кандидатом на такую «полноту», этот продукт может рассматриваться как одна из попыток создания базы для решения этой проблемы. Разработчик FreeCAD Юрген Ригель, работающий в корпорации DaimlerChrysler, позиционирует свою программу как первый бесплатный инструмент проектирования механики (сравнивая свой продукт с такими развитыми проприетарными системами как CATIA версий 4 и 5, SolidWorks), созданный на основе библиотеки **Open CASCADE**. Цель программы — предоставить базовый инструментарий этой библиотеки в интерактивном режиме.

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

---

<sup>2</sup> копипаста: [https://ru.wikipedia.org/wiki/FreeCAD\\_\(Juergen\\_Riegel%27s\)](https://ru.wikipedia.org/wiki/FreeCAD_(Juergen_Riegel%27s))

<sup>3</sup> FreeCAD — CAD/CAE приложение трёхмерного параметрического моделирования. Оно в основном сделано для механического проектирования, но также может быть использовано для любых других случаев, в которых вам нужно точно моделировать трёхмерные объекты с контролем над историей моделирования.

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

Как и все проекты с открытым исходным кодом, проект FreeCAD не единственный способ работы обеспеченный Вам его разработчиками. Это во многом зависит от роста его сообществу пользователей и разработчиком, доработки функций и стабилизации кода (да здравствует исправление ошибок!). Так что не забывайте об этом, когда начинаете использовать FreeCAD, если вам он нравится, вы можете непосредственно влиять и помочь проекту!

---

<sup>3</sup> копипаста: [http://www.freecadweb.org/wiki/index.php?title=Getting\\_started](http://www.freecadweb.org/wiki/index.php?title=Getting_started)

# Глава 90

## Установка

### 90.1 Windows





[FreeCAD © Juergen Riegel, Werner Mayer, Yorik van Havre 2001-2011](#)

Версия	<b>0.14</b>
Редакция	<b>3700 (Git)</b>
Дата выпуска	<b>2014/07/13 11:34:36</b>
Операционная система	<b>Windows 7</b>
Word size	<b>32-bit</b>
Branch	<b>releases/FreeCAD-0-14</b>
Hash	<b>32f5aae0a64333ec8d5d160dbc46e690510c8fe1</b>

[Лицензия ...](#)

[Скопировать в буфер обмена](#)

OK

## 90.2 Linux

## Часть VII

Инструменты и электронное оборудование

# Глава 91

## Радиомонтажный инструмент

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

### 91.1 Pro'sKit

Отдельного обзора заслуживает инструмент и наборы Pro'sKit



PK-5308VM универсальный набор инструментов



1PK-616B Набор инструментов для электроники профессиональный



1PK-813B Набор базовых инструментов для электроники

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

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

## 91.2 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

## 91.3 Хранение



103-132D Кассетница для деталей и компонентов



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

## 91.4 Радиомонтаж



8PK-30D Кусачки миниатюрные



1PK-709 Длинногубцы-кусачки



1PK-055S Длинногубцы изогнутые



1PK-29 Круглогубцы



1PK-101Т Пинцет прямой



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



PD-374 Тиски на струбцине

## 91.5 Прочие

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



# Глава 92

## Паяльное оборудование

### 92.1 Паяльник

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

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



Паяльник ЭПЧН-25/220



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



Паяльник 220В 25Вт ZD-721N 175 р.



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

## 92.2 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 93

## Измерительное оборудование

### 93.1 Мультиметр

Мультиметр — обязателен, без него работать невозможно<sup>1</sup>. Для совсем начинающего больше всего подойдет M32093.1.3 с автодиапазоном, когда освоитесь вторым прибором что-то из крупных серий M89x/MY6x с измерением температуры<sup>2</sup> или “рыльцеметр”93.5 (RLC).

---

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

<sup>2</sup> иногда нужно для измерения температуры корпусов элементов, радиаторов, растворов если возитесь с электрохимией

### 93.1.1 Mastech M838



Простой, компактный, дешевый, с измерением температуры

### 93.1.2 Mastech M300



Простой, очень компактный, дешевый, в чехле очень удачно умещается в набор инструментов.

### 93.1.3 Mastech M320



То же что и M30093.1.2, 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.

93.2 Осциллограф

93.3 Логический анализатор

93.4 Генератор сигналов

93.5 Рыльцеметр RLC

Глава 94

Электроинструмент

## 94.1 Дрель



Дрель ударная сетевая

Praktyl-R PID13D01 400 Вт (!)395 р.



Дрель безударная сетевая

Интерскол Д-11/530ЭР (с БЗП) 1120 р.

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

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

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

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

Имея некоторое количество поделочного материала, кривые руки и особенно доступ к станочному оборудованию, можно сколкозить некоторое подобие настольных станочков 94.1 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.



## 94.2 Лобзик



Praktyl 350 Вт 356 р.



Makita 4329 2260 р.

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

## 94.3 Двигатель

Если у вас возникло желание механизировать изготовление механических деталей, а свободного доступа к настоящему станочному оборудованию нет, есть смысл рассмотреть изготовление самодельной механизированной оснастки типа 94.1, 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 тоцилке<sup>1</sup>.

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

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

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

Для самодельных серлилок и микроинструмента хороши китайские воздушные шпинNELи постоянного тока с цанговыми патронами ER11. Требуют источник питания постоянного тока 9÷48 В. В магазинах не попадались, необходима прямая покупка с AliExpress<sup>2</sup> по почте.

---

<sup>1</sup> БЗП удобен, патрон с ключем дает лучший зажим и возможно точнее

<sup>2</sup> пользуйтесь английской версией — переводная жуткое УГ



Двигатель Вятка-Автомат 19?? г.



Двигатель печки Камаза



АИРЕ 56 В2, 0.2 кВт



Воздушный шпиндель с цангой ER11

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



KRESS 530/800/1050 FM(E)  
5600+ р.



Интерскол ФМ-30/750  
/снят с производства/



Интерскол ФМ-55/1000 Э  
5050 р.

# Часть VIII

## Станочное оборудование

Самые распространенные станки — **сверлильные**, т.к. имеют самую простую конструкцию, и минимальную стоимость. Предназначены для самой частой операции: изготовления перпендикулярных круглых дырок в различных материалах, топовые модели имеют также функцию нарезания резьбы. Для монтажа электроники и кустарного 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 95

## Настольные станки

На основе<sup>1</sup>

© 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

---

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

Глава 96

Самодельная оснастка

# Глава 97

## Промышленные станки

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

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

## 97.1 1A616: станок токарно-винторезный



AvizInfo.com.ua

- 97.1.1 Назначение и области применения**
- 97.1.2 Распаковка и транспортировка**
- 97.1.3 Фундамент станка, монтаж и установка**
- 97.1.4 Подготовка станка к первоначальному пуску**
- 97.1.5 Паспортные данные**
- 97.1.6 Описание основных узлов**
- 97.1.7 Смазка**
- 97.1.8 Первоначальный пуск**
- 97.1.9 Указания по технике безопасности**
- 97.1.10 Настройка**
- 97.1.11 Регулирование**
- 97.1.12 Ведомость комплектации**

## Часть IX

Разработка ПО для встраиваемых систем

# Глава 98

## IDE

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

- средства управления проектом,
- отслеживание зависимостей между файлами (в т.ч. с анализом исходного текста программ на конструкции типа `#include`, `module`, `uses`),
- автозапуском компиляторов для изменившихся файлов,
- GUI для отладчиков (`gdb`),
- специализированный редактор `plain text`<sup>1</sup> файлов с

---

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

- цветовой и шрифтовой подсветкой синтаксиса,
  - автодополнением: дописываются имена объектов программ, синтаксические конструкции и параметры функций,
  - автоформатированием: фрагмент текста переформатируется в соответствии с синтаксисом языка редактируемого файла, проставляются отступы в зависимости от вложенности синтаксических конструкций типа циклов и условных блоков)
  - выделением строк, на которые указывают сообщения об ошибках компиляторов,
  - маркеры точек останова отладчика
- отображение структуры программ, например дерева классов и структур данных
  - контекстные справочники по используемым языкам программирования, автоматический вывод списка параметров при вводе имени функции
  - отображение дизассемблерных листингов для компилируемых языков
  - отображение браузера как вкладки или MDI окна
  - отображение вывода статических анализаторов программ с кликабельными ссылками
  - вывод компиляторов и трансляторов с цветовым выделением и переход на ошибочную строку в редакторе при щелчке на ошибке
  - ...

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

1.  **ECLIPSE** 98.1: самая навороченная и ресурсоемкая IDE, написана на Java, имеет десятки дополнительных модулей на все случаи, умеет работать со всеми распространенными языками программирования, жрет память, и требует современного компьютера минимум с 2+ Гб ОЗУ. Последний релиз  Luna работает заметно быстрее (особенно при запуске).
2. **Code::Blocks** 98.2: легкая среда для разработки на C/C<sub>+</sub><sup>+</sup>, для других языков может потребоваться написать свои модули или файлы описания синтаксиса
3. **Vim** 98.3: самый легкий и портабельный универсальный текстовый редактор с расширенными функциями, работает на всех существующих платформах (кроме совсем уж embedded), использует минимум ресурсов, но требует некоторого обучения даже чтобы выйти из vim 😊

## 98.1 eclipse



### 98.1.1 Установка eclipse под Windows

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

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

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

### 98.1.2 Установка eclipse под Linux

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

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

Или сразу сборку CDT  ECLIPSE: [Eclipse IDE for C/C++ Developers](#) » Linux 32/64 Bit

Пока качается, параллельно устанавливаем в систему Java-рантайм:

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

Распаковываем полученный архив **eclipse-java-luna-SR1-linux-gtk-x86\_64.tar.gz** в **\$HOME**:

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

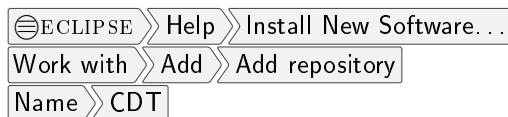
Прописываем запуск  ECLIPSE в ваш оконный менеджер или **.blackboxmenu** с параметром **-noSplash** для лечения глюка с запуском на x64-битных системах:

**.blackbox.menu**

```
1 [exec] (eclipse) {xterm -e "rm -f ~/.metadata/.lock ; eclipse/eclipse -noSplash"}
```

### 98.1.3 Установка CDT

**CDT** — расширение  ECLIPSE для разработки на *Cи/C<sub>+</sub>*, редактирования make-файлов. Это расширение критически важно для вашей работы, поэтому ставить его обязательно, или сразу качать сборку CDT  ECLIPSE.



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

## 98.1.4 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

## 98.1.5 Установка TeXlipse

Если планируете работать с документацией в формате L<sup>A</sup>T<sub>E</sub>X, 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.

## 98.1.6 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

## 98.1.7 Проверка орфографии

2

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

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

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

---

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

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

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

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

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

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

Установка состоит из 4 шагов:

1. качаем aSpell и словари для нужных языков

 + R ➤ <http://aspell.net/win32/>

Binaries ➤ Full installer

Precompiled dictionaries ➤ English

Precompiled dictionaries ➤ Russian

2. устанавливаем сначала aSpell, потом отдельно каждый словарь

**Aspell-0-50-3-3-Setup.exe** ➤ Setup GNU Aspell ➤ Next ➤ License ➤ Next

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

Additional ➤ Next ➤ Install ➤ Next ➤  View manual ➤ Finish

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

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

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

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

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

[田] + R cmd

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

4. настраиваем *spell-checker* ☰ECLIPSE

☰ECLIPSE ➤ Window ➤ Preferences ➤ Editors ➤ Text editors ➤ Spelling

User defined dictionary ➤ C:/GnuWin32/Aspell/enru.dict

Encoding ➤ UTF-8

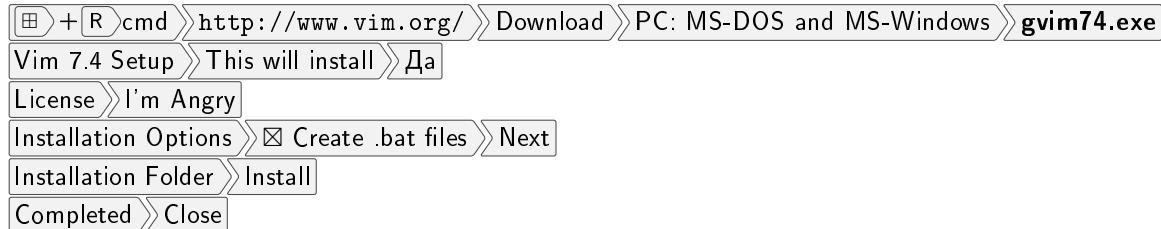
Apply ➤ OK

## 98.2 Code::Blocks

## 98.3 Vim



### 98.3.1 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 Vim

Esc: q ! Enter

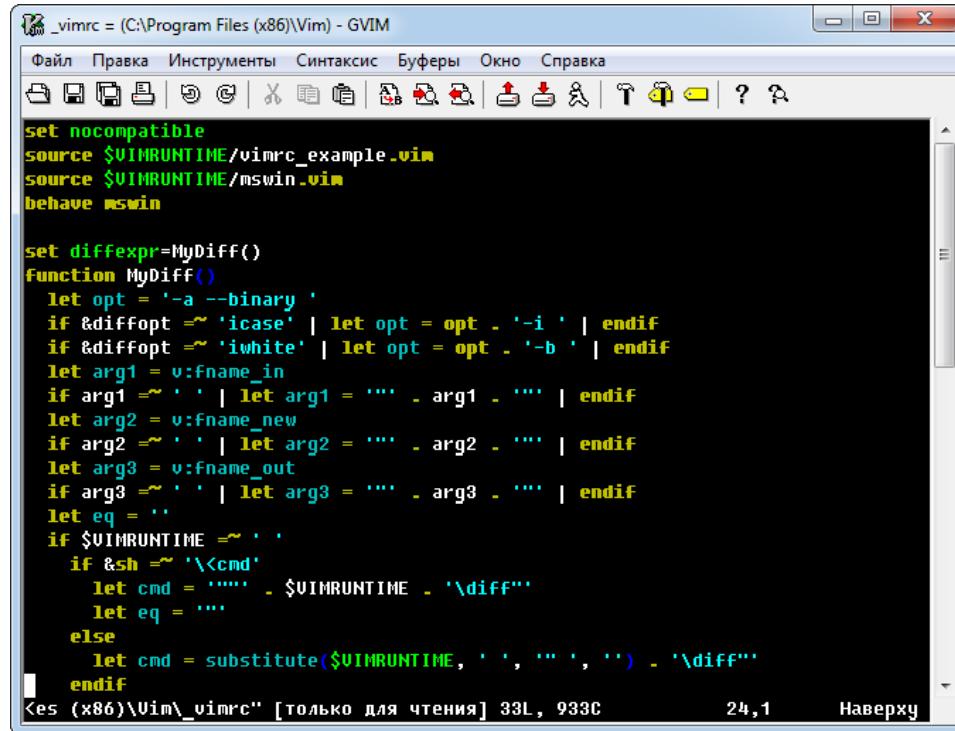
Если не получилось (под Windows 7):

Win + R cmd » /Program Files (x86)/Vim/

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

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

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



The screenshot shows the GVIM interface with a file named '\_vimrc' open. The window title is '\_vimrc = (C:\Program Files (x86)\Vim) - GVIM'. The menu bar includes 'Файл', 'Правка', 'Инструменты', 'Синтаксис', 'Буферы', 'Окно', and 'Справка'. Below the menu is a toolbar with various icons. The main text area contains Vim script code with syntax highlighting. The code includes commands like 'set', 'source', 'function', and 'let'. The script defines a custom 'MyDiff()' function that handles file arguments and runs a 'diff' command. The status bar at the bottom shows the path 'C:\Program Files (x86)\Vim\\_vimrc' and the file statistics '33L, 933C'. The bottom right corner of the status bar says 'Наверху'.

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

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

### 98.3.2 Выход из Vim

Esc : ! q Enter

### 98.3.3 Выход с автосохранением

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

### 98.3.4 Переход в режим редактирования

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*

### 98.3.5 Переход в режим команд

[Esc]

### 98.3.6 Запись редактируемого файла

[Esc] : [w] [Enter]

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

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

### **98.3.7 Перезагрузка файла**

Для перезагрузки возможно изменененного извне файла или отмены всех несохраненных изменений

**Esc** : **e** **Enter**

### **98.3.8 Отмена последних изменений (undo)**

**Esc** **u** **u**... .

# Глава 99

Make: управление сборкой проектов

# Глава 100

## VCS: системы контроля версий

100.1 CVS

100.2 Subversion

100.3 Git

100.3.1 GitHub

# Глава 101

## Вспомогательные скрипты на языке Python

# Глава 102

## Основы Си и $C_+^+$

102.1 Установка MinGW (win32)

102.2 Особенности  $C_+^+$  в embedded

102.3 Сборка кросс-компилятора GNU toolchain

# Глава 103

## 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 GPS<sup>1</sup>, очень популярный мета-формат XML со всеми его частными случаями типа HTML, XLIFF??, OpenDocument, тексты программ для станков с ЧПУ,...

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

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

Все эти техники с использованием стандартных утилит `flex` и `bison` будут кратко описаны в этой главе.

---

<sup>1</sup> протокол NMEA 0183

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

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

Habr: Компиляция. 1: лексер

Habr: Компиляция. 2: грамматики

Habr: Компиляция. 3: бизон

Habr: Компиляция. 4: игрушечный ЯП

Habr: Компиляция. 5: нисходящий разбор

Habr: Компиляция. 5 $\frac{1}{2}$ : llvm как back-end

Habr: Компиляция. 6: промежуточный код

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

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

[http://www.caps1.udel.edu/courses/cpeg421/2012/slides/Tutorial-Flex\\_Bison.pdf](http://www.caps1.udel.edu/courses/cpeg421/2012/slides/Tutorial-Flex_Bison.pdf)

## 103.1 Лексер и лексический анализ, утилита **flex**

Лéксер/сканер — программа или ее часть, которая

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

Цель лексера — подготовить последовательность лексем для входа другой программы.

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

**Лексический анализ** — процесс программного разбора входной последовательности символов<sup>2</sup> с целью получения на выходе последовательности групп символов — **токенов**, имеющих собственное смысловое значение<sup>3</sup>. 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 (лексема), которая согласно грамматике не может быть идентифицирована как токен языка, обычно рассматривается как специальный **токен-ошибка**.

Каждый выделенный токен можно представить в виде парной структуры, содержащей

1. идентификатор токена и
2. саму последовательность символов лексемы, выделенной из входного потока<sup>4</sup>.

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  $\text{\LaTeX}$ -совместимое форматирование.

<sup>2</sup> например, такой как исходный код на одном из языков программирования

<sup>3</sup> подобно группировке букв в слово

<sup>4</sup> запись строки, числа и т. д.

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

определения, опции, декларации

%%

правила выделения токенов через регулярки

%%

сишный код

**Комментарии** поддерживаются сишные /\* \*/ комментарии.

**Опции %option**

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

main            включение типовой функции main() вместо заданной пользователем

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

yylineno        в глобальной си-переменной yylineno доступен номер текущей строки

**Формат определения имя определение:**

digit        [0-9]

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

## Makefile

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

flex лексер

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

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

## 103.2 Генератор синтаксических анализаторов **bison**

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

## bison парсер

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

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

# Глава 104

## LLVM и разработка собственных компиляторов

### 104.1 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.

### 104.2 Создание компилятора с помощью инфраструктуры LLVM

<sup>1</sup> © IBM

---

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

Инфраструктура LLVM<sup>3</sup> 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<sup>4</sup> 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.

### 104.3 Инструменты **lcc** 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

---

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

<sup>3</sup> [L]ow [L]evel [V]irtual [M]achine, низкоуровневая виртуальная машина

<sup>4</sup> intermediate representation, IR

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

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

## 104.4 Семантический анализ

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

## 104.5 Оптимизация

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

## 104.6 Кодогенерация

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

## 104.7 Транслятор Паскаля

## Часть X

# Микроконтроллеры Cortex-Mx

# Глава 105

## Производители

105.1 ST Microelectronix STM32

105.2 LPC

105.3 Миландр

# Глава 106

## Отладочные платы

106.1 LeafLabs Maple Mini: STM32F103 /Cortex-M3/

106.2 Серия STM32 STM32DISCOVERY

106.2.1 STM32DISCOVERY: STM32F103 /Cortex-M3/

106.2.2 STM32F4DISCOVERY: STM32F406 /Cortex-M4F/

106.2.3 STM32F0DISCOVERY: STM32F040 /Cortex-M0/

Часть XI

ПЛИС

---

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

Часть XII

USB

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

Любое устройство, предназначенное для подключения к компьютеру через USB, должно поддерживать хотя бы небольшую часть спецификации протокола USB: уметь «представиться»<sup>2</sup> 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 устройства<sup>3</sup>, оставляя нам самое интересное — управление функциями устройства.

Перенос кода управления устройством в `userspace` не только упрощает отладку<sup>4</sup>, но и позволяют писать процедуры управления устройством на самых разных языках программирования, а не только на Си, как это делают те, кто пишет драйверы уровня ядра. Более того, пользовательская часть драйвера, не взаимодействуя

---

<sup>2</sup> предоставить информацию о себе и своих возможностях

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

<sup>4</sup> 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 107

## Стек протоколов USB

Протокол USB похож на стек сетевых протоколов, основанных на TCP/IP, точнее за основу принималась сетевая модель стека сетевых протоколов OSI/ISO<sup>1</sup>:

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

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

На самом нижнем логическом уровне<sup>2</sup> устройства обмениваются пакетами данных<sup>3</sup>. 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.

---

<sup>2</sup> спецификации физического уровня мы пока не рассматриваем

<sup>3</sup> со встроенными механизмами коррекции ошибок, подтверждения получения и т.д

# Глава 108

## libUSB

### 108.1 драйвер для устройства USB

1

---

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



# Глава 109

## Поддержка USB в Linux

### 109.1 Опции ядра

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

109.1.2 data storage: носители данных

109.1.3 hid: клавиатура, мышь, джойстик

109.1.4 USB-периферия: сеть, звук,...

### 109.2 Настройка hotplug и автомонтирования USB носителей

### 109.3 Сборка и настройка libusb

### 109.4 Примеры программ низкоуровневого ввода/вывода

# Часть XIII

## Встраиваемый emLinux

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

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

Из недостатков можно отметить:

- Отсутствие полноценной поддержки режима жесткого реального времени;
- Тяжелое ядро;
  - Поддерживаются только мощные семейства процессоров<sup>2</sup>;
  - Значительные требования по объему ОЗУ и общей производительности;
- Дремучесть техспециалистов, контуженных ТурбоПаскалем и Windowsом;

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

emLinux очень широко применяется на рынке мобильных устройств<sup>3</sup>, и устройств интенсивно использующих сетевые протоколы (роутеры, медиацентры).

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

---

<sup>1</sup> будем называть его emLinux

<sup>2</sup> 32-бит, необходим блок MMU

<sup>3</sup> в т.ч. является основой Android

# Глава 110

## 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/>

### 110.1 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>

## 110.2 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*

/bjos/com32/libutil/libutil.c32

/bios/com32/lib/libcom32.c32

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

/syslinux/syslinuxusb.bat

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

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

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

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

## 110.3 **syslinux.cfg**

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

/syslinux/syslinux.cfg

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

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

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

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

# Глава 111

## 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, защищой 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*<sup>1</sup> для выполнения различных полезных функций: сетевые серверы, символьная математика, обработка данных,...
- Еще одна ключевая фича — способность Linux полностью загружаться в ОЗУ с любых носителей, в т.ч. заблокированных на запись. Это важно для случаев, когда возможны внезапные выключения питания: вся система работает в ОЗУ-диске, а корректность записи данных на изменяемые носители можно гибко контролировать программно. При запуске после аварийного выключения никаких проверок файловых систем не требуется, ОС стартует сразу, а проверку/починку разделов данных возможно выполнять в фоновом режиме.
- Время запуска системы — на x86 удалось экспериментально получить время запуска 0.2 сек от загрузки ядра до начала выполнения пользовательского кода. Используя модульное ядро, возможно выполнить критический к времени запуска пользовательский код до инициализации USB, сети, внешних носителей данных и тяжелых сервисов.

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

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

---

<sup>1</sup> для использования как есть, изучения принципов работы и модификации под собственные нужды

Расширять функционал, добавляя libQt, X Window, Apache, MySQL, …, Gnome/KDE и т.д. не планируется в принципе — это система для решения узких прикладных задач на аппаратуре с минимальными ресурсами<sup>2</sup>. Интерактивная работа с пользователем также не предполагается, доступна только командная консоль<sup>3</sup>, 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](#)??.

## 111.1 Требования к системе сборки (**BUILD**-хост)

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

добавить в **/etc/fstab**

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

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

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 2Гб ОЗУ мне для сборки [gcc0](#) пришлось временно размонтировать [cross/src](#), сделать [make gcc ramclean](#), а потом примонтировать [tmpfs](#) опять на [src](#).

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

<sup>3</sup> ее можно считать сервисным режимом

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

## 111.2 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 ; 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` подгружает части из каталога `mk/`, поэтому *если вы добавляете новый скрипт, не забудьте прописать его в Makefile*.

### Makefile

```
1 include mk/head.mk
2 include mk/opt.mk
3 include mk/dirs.mk
4 include mk/versions.mk
5 include mk/packages.mk
6 include mk/commands.mk
7 include mk/clean.mk
8 include mk/gz.mk
9 include mk/src.mk
10 include mk/cfg.mk
11 include mk/cross.mk
12 include mk/core.mk
13 include mk/kernel.mk
14 include mk/ulibc.mk
15 include mk/busybox.mk
16
17 include mk/sdk/canadian.mk
18 include mk/sdk/python.mk
19 include mk/sdk/pascal.mk
20 include mk/sdk/lisp.mk
21
22 include mk/math/libs.mk
23 include mk/math/octave.mk
24 include mk/math/maxima.mk
```

```
25 include mk/math/mpi.mk
26
27 include mk/user.mk
28
29 include mk/libs.mk
30 include mk/lib/zlib.mk
31 include mk/lib/png.mk
32 include mk/lib/sdl.mk
33 include mk/lib/freetype.mk
34 include mk/lib/multiprecision.mk
35 include mk/lib/cpp.mk
36 include mk/lib/pcre.mk
37
38 include mk/apps.mk
39 include mk/emu.mk
40 include mk/root.mk
41 include mk/boot.mk
42 include mk/info.mk
```

### 111.3 Клонирование проекта azLinux

При необходимости вносить правки<sup>4</sup> работайте с вашим собственным форком на GitHub.

azLinux разрабатывается по нескольким веткам:

---

<sup>4</sup> что естественно — вам потребуется добавлять свои пакеты и поддержку железа

master	основная универсальная ветка, включает все скрипты сборки и пакеты в книгу включается именно эта ветка, набор скриптов в других ветках будет отличаться
minimal	минимальная система для нескольких платформ
clock	Linux-powered электронные часы/контроллер умного дома
math	математический софт и библиотеки для вычислительных кластеров beowulf
desk	десктопная сборка с функционалом ~Windows95
cnc	система управления для станков с ЧПУ

Сделайте свой форк с репозитория <https://github.com/ponyatov/azlin>.

Получите клон системы сборки из репозитория выбрав нужную ветку:

```
cd ~  
git clone --depth=1 -o gh -b minimal git@github.com:user/azlin.git myazlin
```

При необходимости обновитесь:

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

## 111.4 Общий порядок сборки

Каждый пакет собирается командой:

```
make [HW=rpi] [APP=clock] [CCACHE=] [RAMCLEAN#echo] <package>
```

- **HW** выбор аппаратной платформы **hw/**
- **APP** выбор приложения **app/**

- **CCACHE=** пустая переменная блокирует использование `ccache`
- **RAMCLEAN** переменная задает команду, которой передается набор каталогов, которые нужно зачистить; по умолчанию это `rm -rf`, заданием `echo` зачистка отключается (нужно если вы собираете систему в каталогах `src/`, `tmp/` не в ОЗУ, а на диске)

1. `dirs` создание дерева каталогов 111.6
2. `gz` закачка архивов исходников 111.7
3. `cross0` сборка кросс-компилятора 111.14.6
  - (a) `binutils0` ассемблер, линкер и утилиты ??
  - (b) `cclibs0` библиотеки для сборки `gcc` ??
  - (c) `gcc0` сборка минимального кросс-компилятора Си 111.14.9
4. `core` сборка основной системы 111.14.11
  - (a) `coref` сборка основной системы + GNU Fortran ??
  - (b) `kernel` ядро Linux 111.14.12
  - (c) `ulibc` библиотека `uClibc` 111.14.13
  - (c) `busybox` набор утилит `busybox` 111.14.15
  - (d) `gcc` пересборка полного кросс-компилятора Си/ $C^+$  111.14.14  
`gccc` кросс-компилятор Си/ $C^+$ /Fortran ??
5. `libs` сборка библиотек **`LIBS`** 111.14.16
6. `apps` сборка прикладных пакетов **`APPS`** 111.14.17
7. `user` сборка пользовательского кода 111.14.18
8. `root` формирование корневой файловой системы 111.14.19
9. `boot` сборка загрузчика 111.14.20 `syslinux/grub/uboot`
10. `emu` запуск собранной системы в эмуляторе 111.14.24
11. `netboot` сетевая загрузка 111.15
12. `firmware` прошивка на устройство 111.16

## 111.5 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.

```
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.

## 111.6 dirs: Создание дерева каталогов

После загрузки или обновления запустите пакет **dirs**:

```
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 71
-rw-r--r-- 1 user user 2087 Дек 8 13:26 Makefile
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 cput
```

```
drwxr-xr-x 2 user user 4096 Дек 8 11:43 arch
drwxr-xr-x 2 user user 4096 Дек 8 11:43 app
drwxr-xr-x 2 user user 4096 Дек 8 13:26 gz
drwxr-xr-x 2 user user 4096 Дек 8 13:26 src
drwxr-xr-x 2 user user 4096 Дек 8 13:26 tmp
drwxr-xr-x 3 user user 4096 Дек 8 13:26 x86micro
drwxr-xr-x 2 user user 4096 Дек 8 13:26 x86micro.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
```

Пакет dirs прописан в файле

### mk/dirs.mk

```
1 # directories processing
2
3 # sw sources mirror archive in .tar.[GZ]
4
5 GZ = $(PWD)/gz
6
7 # [S]ource [RC]ode unpacked
8
9 SRC = $(PWD)/src
10
11 # [T]e[MP] build dirs
12
13 TMP = $(PWD)/tmp
14
15 # build/target triplets
```

```
16 BUILD = $( shell gcc --dumpmachine )
17
18 # target root filesystem
19
20 ROOT = $(PWD)/$(HW)$(APP)
21 # cross-compiler [T]ool [C]hain
22 TC = $(ROOT).cross
23 BOOT = $(ROOT)/boot
24 ETC = $(ROOT)/etc
25 USR = $(ROOT)/usr
26 USRBIN = $(USR)/bin
27 USRLIB = $(USR)/lib
28 PACK = $(ROOT)/pack
29 ISO = $(TMP)/iso
30 LIB = $(ROOT)/lib
31
32 DIRS = $(GZ) $(GZ)/patch $(GZ)/patch/python $(SRC) $(TMP) $(TC) $(ROOT) \
33 $(LIB) $(BOOT) $(USR) $(USRBIN) $(USRLIB) $(PACK)
34
35 .PHONY: dirs
36 dirs:
37     mkdir -p $(DIRS)
```

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

Каталог зеркала архивов исходных текстов программ

GZ

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

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

### SRC

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

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

### TMP

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

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

### BUILD

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

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

### TC

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

Каталог целевой **rootfs**. Отдельно прописан загрузочный каталог, в который будет записываться собранное  
ядро, образ `initrd`, бинарники и конфиги загрузчика. Имя **ROOT** создается из двух переменных, описанных  
далее: имя аппаратной платформы `HW111.9` и имени приложения `APP111.8`.

### ROOT

```
1 ROOT = $(PWD)/$(HW)$(APP)
```

## BOOT

```
1 BOOT = $(ROOT)/boot
```

Список всех рабочих каталогов в одной переменной:

### DIRS

```
1 DIRS = $(GZ) $(GZ)/patch $(GZ)/patch/python $(SRC) $(TMP) $(TC) $(ROOT) \
2 $(LIB) $(BOOT) $(USR) $(USRBIN) $(USRLIB) $(PACK)
```

## 111.7 **gz**: Загрузка архивов исходников

### mk/gz.mk

```
1 # download sw packages sources to gz/
2
3 .PHONY: gz
4 gz:
5     make gz_$(HW)
6     make gz_$(ARCH)
7     make gz_$(APP)
8     make gz_cross
9     make gz_core
10    make gz_libs
11    # make gz_sdk
12    make gz_math
13    make gz_apps
14
```

```
15 .PHONY: gz_cross
16 gz_cross:
17     $(WGET) http://ftp.gnu.org/gnu/binutils/$(BINUTILS).tar.bz2
18     $(WGET) http://gcc.skazkaforyou.com/releases/$(GCC)/$(GCC).tar.bz2
19     $(WGET) ftp://ftp.gmplib.org/pub/gmp/$(GMP).tar.bz2
20     $(WGET) http://www.mpfr.org/mpfr-current/$(MPFR).tar.bz2
21     $(WGET) http://www.multiprecision.org/mpc/download/$(MPC).tar.gz
22     $(WGET) ftp://gcc.gnu.org/pub/gcc/infrastructure/$(ISL).tar.bz2
23     $(WGET) ftp://gcc.gnu.org/pub/gcc/infrastructure/$(CLOOG).tar.gz
24
25 .PHONY: gz_core
26 gz_core:
27     $(WGET) https://www.kernel.org/pub/linux/kernel/v3.x/$(KERNEL).tar.xz
28     $(WGET) http://www.uclibc.org/downloads/$(ULIBC).tar.xz
29     $(WGET) http://busybox.net/downloads/$(BUSYBOX).tar.bz2
30
31 .PHONY: gz_libs
32 gz_libs:
33     $(WGET) https://www.libsdl.org/release/$(SDL).tar.gz
34     $(WGET) https://www.libsdl.org/projects/SDL_image/release/$(SDL_IMAGE).tar.gz
35     $(WGET) http://www.libsdl.org/projects/SDL_ttf/release/$(SDL_TTF).tar.gz
36     $(WGET) http://download.sourceforge.net/libpng/$(PNG).tar.xz
37     $(WGET) http://download.savannah.gnu.org/releases/freetype/$(FREETYPE).tar.bz2
38     $(WGET) http://zlib.net/$(ZLIB).tar.xz
39     $(WGET) http://cxx.uclibc.org/src/$(LIBCPP).tar.xz
40
41 #.PHONY: gz_sdk
42 #gz_sdk: gz_python
```

```
43 #__$(WGET) ftp://ftp.hu.freepascal.org/pub/fpc/dist/$(FPC_VER)/source/$(FPC).source.tar.g
44 #__$(WGET) ftp://ftp.hu.freepascal.org/pub/fpc/dist/$(FPC_VER)/source/fpcbuild-$(FPC_VER)
45
46 #.PHONY: gz_python
47 #gz_python:
48 ## exit -1
49 #__$(WGET) https://www.python.org/ftp/python/$(PYTHON_VER)/$(PYTHON).tar.xz
50 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/001-r
51 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/002-f
52 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/004-s
53 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/005-p
54 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/006-c
55 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/007-d
56 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/008-c
57 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/009-n
58 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/010-f
59 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/011-r
60 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/013-d
61 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/014-a
62 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/015-f
63 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/016-s
64 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/017-d
65 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/100-o
66 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/101-o
67 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/102-o
68 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/103-o
69 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/104-o
70 #__$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/105-o
```

```
71 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/106 -o
72 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/107 -o
73 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/108 -o
74 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/109 -o
75 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/110 -o
76 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/111 -o
77 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/112 -o
78 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/113 -o
79 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/114 -r
80 #____$(WGET) -P patch/python http://git.buildroot.net/buildroot/plain/package/python/115 -c
81
82 .PHONY: gz_i386
83 gz_i386:
84 ____$(WGET) https://www.kernel.org/pub/linux/utils/boot/syslinux/$(SYSLINUX).zip
85
86 .PHONY: gz_arm
87 gz_arm:
88 ____$(WGET) ftp://ftp.denx.de/pub/u-boot/$(UBOOT).tar.bz2
89
90 .PHONY: gz_apps
91 gz_apps:
92 ____$(WGET) http://elinks.or.cz/download/$(ELINKS).tar.bz2
93
94 .PHONY: gz_math
95 gz_math:
96 #____exit -1
97 ____$(WGET) ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/$(PCRE).tar.bz2
98 ____$(WGET) http://www.netlib.org/blas/blas.tgz
```

```

99 ____$(WGET) http://www.netlib.org/lapack/$(LAPACK).tgz
00 ____$(WGET) ftp://ftp.gnu.org/gnu/octave/$(OCTAVE).tar.bz2
01 ____$(WGET) http://www.mpich.org/static/downloads/$(MPICH_VER)/$(MPICH).tar.gz
02 ____$(WGET) http://www.mpich.org/static/downloads/$(MPICH_VER)/$(HYDRA).tar.gz
03 ##____$(WGET) http://mirror.tochlab.net/pub/gnu/gsl/$(GSL).tar.gz
04 ##____$(WGET) http://downloads.sourceforge.net/project/ecls/ecls/$(ECL_VER_A)/$(ECL).tgz
05 ##____$(WGET) http://downloads.sourceforge.net/project/maxima/Maxima-source/$(MAXIMA_VER)-s
06
07 .PHONY: gz_desk
08 gz_desk:
09 ____$(WGET) http://www.x.org/releases/$(X_RELEASE)/src/xserver/$(X_SERVER).tar.bz2
10
11 .PHONY: gz_pc686
12 gz_pc686:

```

## 111.8 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]lication: micro clock math desk ...
2 APP = math
```

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

```
make APP=macro distclean dirs tc core libs apps
```

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

- **LIBS**: набор используемых библиотек
- **PACKS**: набор используемых программных пакетов 111.14

app/micro.mk

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

app/clock.mk

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

## 111.9 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 111.10.2	CeleronM	i386	512M+	<input type="checkbox"/> SSD	<input type="checkbox"/> SD	<input checked="" type="checkbox"/>	A??	<input type="checkbox"/> AR2425	
gac1037 ??	Celeron1037U	x86_64	1G+	<input type="checkbox"/> SATA		<input checked="" type="checkbox"/>	2×RTL8111		
qemuARM		arm	32M+						
cubie1 111.11.2	AllWinnerA10	armhf	1G		<input type="checkbox"/> μSD	<input checked="" type="checkbox"/>			
rpi 111.11.3	BCM2835	armel	512M		<input type="checkbox"/> SD	<input checked="" type="checkbox"/>			
tion 111.11.4	PXA270	arm??							
mb77 111.11.5	K1879ХБ1Я	arm??							
qemuMIPS		mips	32M+						
mr3020 ??	AR7240	mips	32M	4M		<input checked="" type="checkbox"/>		<input type="checkbox"/> AR9331	
vocore 111.12.3	RT5350	mips	32M	8M		<input type="checkbox"/>		<input type="checkbox"/> SoC	
bswift		AR9331	mips	64M	16M	<input type="checkbox"/>			20+

### mk/head.mk

```
1 # [H]ard [W]are: qemu386 qemuARM qemuMIPS cubie1 rpi ...
2 HW = pc686
3 # [APP]lication: micro clock math desk ...
4 APP = math
5 # load extra hw definitions: ARCH CPU ...
6 include hw/$(HW).mk
7 # load extra defs for CPU setted in $(HW).mk
8 include cpu/$(CPU).mk
9 # load extra defs for ARCHitecture setted in $(CPU).mk
```

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

## 111.10 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.

### 111.10.1 qemu386: эмулятор QEMU

hw/qemu386.mk

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

```
3 QEMU_CFG = -m 64M -net none -localtime  
4 #vga=ask  
5 #0x312 640x480x24  
6 #0x315 800x600x24
```

## 111.10.2 eeepc701: ASUS Eee PC 701



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

hw/eeeepc701.mk

1 # ASUS Eee PC 701

2 CPU = CeleronM

## 111.10.3 gac1037: Gigabyte GA-C1037UN-EU rev.2



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

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

1. x86\_64/x64: нативный CPU=Celeron1037u
2. *i686/x32*: режим совместимости со старыми процессорами CPU=i686

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

материнская плата	GA-C1037UN-EU rev.2
CPU	Celeron 1037U (впаян)
охлаждение	отвратительное, обязательны кулеры 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
2CPU = Celeron1037U
```

## 111.11 ARM

### 111.11.1 qemuARM: эмулятор QEMU

111.11.2 cubie1: Cubie Board v.1

111.11.3 rpi: Raspberry Pi model B

## 111.11.4 tion: ТионПро270

ЗАО “Завод Электрооборудования”

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

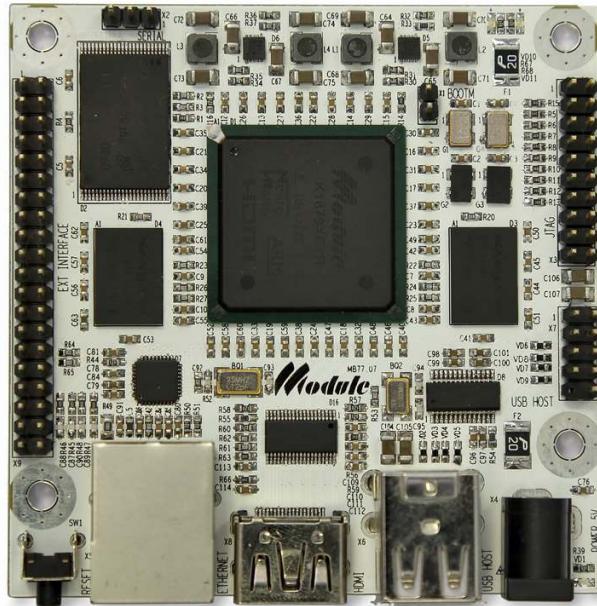


CPU	Marvell XScale PXA270 @ 416/520 МГц
OЗУ	64
Flash	32
видео	<input checked="" type="checkbox"/> VGA DSUB
Ethernet	<input checked="" type="checkbox"/> 10/100M
USB	<input checked="" type="checkbox"/> USB1.1 host
$\mu$ SD	<input checked="" type="checkbox"/>

## 111.11.5 mb77: Микрокомпьютер MB77.07 на базе СБИС K1879ХБ1Я

ЗАО НТЦ "Модуль"

[http://www.module.ru/catalog/micro/micro\\_pc/](http://www.module.ru/catalog/micro/micro_pc/)



CPU

K1879ХБ1Я

ARM1176JZF-S @ 324 МГц

GPU

NeuroMatrix NMC3 @ 324 МГц

OЗУ

Flash

video

HDMI

Ethernet

10/100M

USB

USB2 host

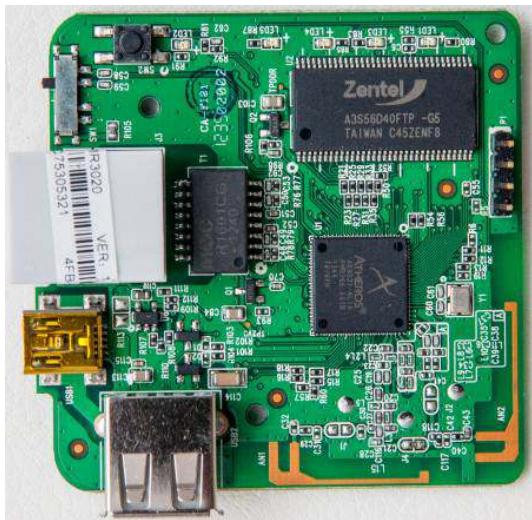
## 111.12 MIPS

### 111.12.1 qemuMIPS: эмулятор QEMU

hw/qemuMIPS.mk

## 111.12.2 mr3020: роутер MR3020

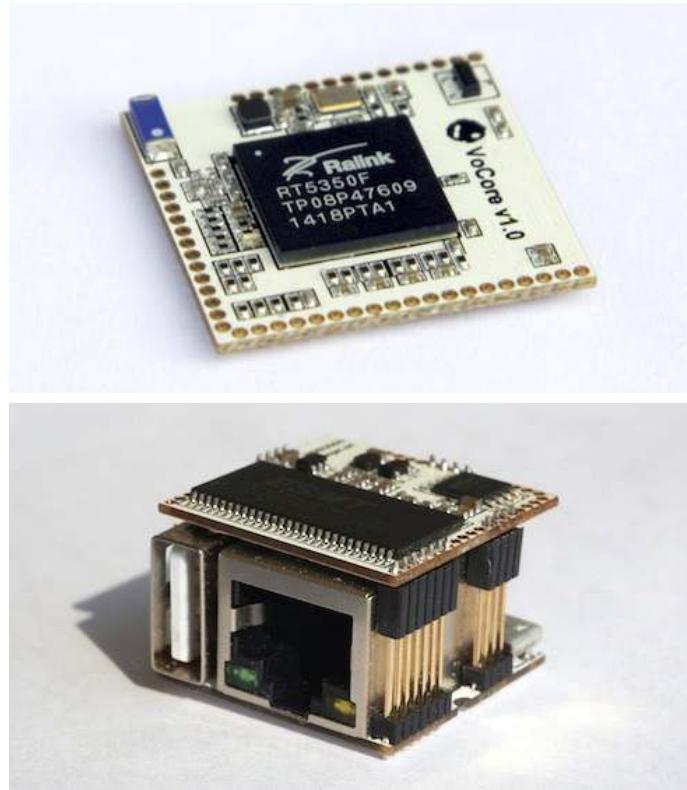
<http://wiki.openwrt.org/ru/toh/tp-link/tl-mr3020>



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

### 111.12.3 vocore: VoCore

vocore



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

### 111.12.4 bswift: BlackSwift

bswift

## 111.13 CPU: Конфигурации процессоров

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

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

### 111.13.1 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-linum-uclibc
6 # target CPU configure params
7 CFG_CPU = --with-cpu=i486 --with-tune=i486 --disable-mmx --disable-3dnow --disable-sse
8 CFLAGS_CPU = -march=i486 -mtune=i486
```

`cpu/CeleronM.mk`

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

cpu/Celeron1037U.mk

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

## 111.13.2 ARM

## 111.13.3 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
```

## 111.14 Пакеты

### 111.14.1 mk/versions.mk: Версии пакетов

### 111.14.2 cross0: кросс-компилятор

### mk/versions.mk

```
1 # cross compiler
2 BINUTILS_VER = 2.24
3 # 2.25 build error
4 GMP_VER = 5.1.3
5 MPFR_VER = 3.1.2
6 MPC_VER = 1.0.2
7 ISL_VER = 0.12.2
8 # 0.14 not works with gcc 4.9.x
9 CLOOG_VER = 0.18.1
10 GCC_VER = 4.9.2
11 # 4.9.2 used: bug arm/62098 fixed
```

### 111.14.3 core: ядро

### mk/versions.mk

```
1 # core
2 KERNEL_VER = 3.19.6
3 # 2.6.x not used: bug eth/rtl8139
4 ULIBC_VER = 0.9.33.2
5 BUSYBOX_VER = 1.23.2
6 #1.22.1
```

### 111.14.4 boot: загрузчики

mk/versions.mk

```
1 # boot loaders
2 SYSLINUX_VER = 6.03
3 UBOOT_VER = 2015.01
```

### 111.14.5 libs: библиотеки

mk/versions.mk

```
1 # libs
2 ## SDL2 not works with FB
3 SDL_VER = 1.2.15
4 SDL_IMAGE_VER = 1.2.12
5 SDL_TTF_VER = 2.0.11
6 FREETYPE_VER = 2.5.5
7 ## extra
8 ZLIB_VER = 1.2.8
9 PNG_VER = 1.6.15
10 PCRE_VER = 8.37
11 ## math libs
12 BLAS_VER = 3.5.0
13 LAPACK_VER = 3.5.0
```

### 111.14.6 cross0: сборка кросс-компилятора

```
1
2 CFG_BINUTILS0 = --target=$(TARGET) $(CFG_ARCH) $(CFG_CPU) \
3     --with-sysroot=$(ROOT) \
4     --with-native-system-header-dir=/include \
5     --enable-lto \
6     CFLAGS="$(BOPT)" CXXFLAGS="$(BOPT)"
7 #__CFLAGS_FOR_BUILD="-g0 -Ofast -march=native -mtune=native"
8
9 CFG_WITHCCLIBS = --with-gmp=$(TC) --with-mpfr=$(TC) --with-mpc=$(TC) \
10    --with-isl=$(TC) --with-cloog=$(TC)
11
12 CFG_CCLIBS00 = --disable-shared CFLAGS="$(BOPT)"
13 CFG_CCLIBS0 = $(CFG_WITHCCLIBS) $(CFG_CCLIBS00)
14
15 CFG_GMP0 = $(CFG_CCLIBS0)
16 CFG_MPFR0 = $(CFG_CCLIBS0)
17 CFG_MPC0 = $(CFG_CCLIBS0)
18 CFG_ISL0 = --with-gmp-prefix=$(TC) $(CFG_CCLIBS00)
19 CFG_CLOOG0 = --with-gmp-prefix=$(TC) $(CFG_CCLIBS00)
20
21 CFG_GCC0 = $(CFG_BINUTILS0) $(CFG_WITHCCLIBS) --disable-bootstrap \
22    --disable-shared --disable-threads \
23    --without-headers --with-newlib
24
25 CFG_GCC = $(CFG_BINUTILS0) $(CFG_WITHCCLIBS) --disable-bootstrap \
26    --enable-shared --enable-threads --enable-libgomp \
27    --enable-libstdcxx-time \
28    --enable-libstdcxx-threads \
```

```
29 ---enable-libstdc++-pch
30
31 .PHONY: cross0
32 cross0: binutils0 cclibs0 ramclean gcc0 ramclean
33
34 .PHONY: binutils0
35 binutils0: $(SRC)/$(BINUTILS)/README
36     rm -rf $(TMP)/$(BINUTILS) && mkdir $(TMP)/$(BINUTILS) && \
37     cd $(TMP)/$(BINUTILS) && \
38     $(SRC)/$(BINUTILS)/$(BCFG) $(CFG_BINUTILS0) && \
39     $(MAKE) && $(INSTALL)-strip
40
41 .PHONY: cclibs0
42 cclibs0: gmp0 mpfr0 mpc0 cloog0 isl0
43
44 .PHONY: gmp0
45 gmp0: $(SRC)/$(GMP)/README
46     rm -rf $(TMP)/$(GMP) && mkdir $(TMP)/$(GMP) && \
47     cd $(TMP)/$(GMP) && \
48     $(SRC)/$(GMP)/$(BCFG) $(CFG_GMP0) && \
49     $(MAKE) && $(INSTALL)-strip
50
51 .PHONY: mpfr0
52 mpfr0: $(SRC)/$(MPFR)/README
53     rm -rf $(TMP)/$(MPFR) && mkdir $(TMP)/$(MPFR) && \
54     cd $(TMP)/$(MPFR) && \
55     $(SRC)/$(MPFR)/$(BCFG) $(CFG_MPFR0) && \
56     $(MAKE) && $(INSTALL)-strip
```

```
57  
58 .PHONY: mpc0  
59 mpc0: $(SRC)/$(MPC)/README  
60     rm -rf $(TMP)/$(MPC) && mkdir $(TMP)/$(MPC) &&\  
61     cd $(TMP)/$(MPC) &&\  
62     $(SRC)/$(MPC)/$(BCFG) $(CFG_MPC0) &&\  
63     $(MAKE) && $(INSTALL)-strip  
64  
65 .PHONY: cloog0  
66 cloog0: $(SRC)/$(CLOOG)/README  
67     rm -rf $(TMP)/$(CLOOG) && mkdir $(TMP)/$(CLOOG) &&\  
68     cd $(TMP)/$(CLOOG) &&\  
69     $(SRC)/$(CLOOG)/$(BCFG) $(CFG_CLOOG0) &&\  
70     $(MAKE) && $(INSTALL)-strip  
71  
72 .PHONY: isl0  
73 isl0: $(SRC)/$(ISL)/README  
74     rm -rf $(TMP)/$(ISL) && mkdir $(TMP)/$(ISL) &&\  
75     cd $(TMP)/$(ISL) &&\  
76     $(SRC)/$(ISL)/$(BCFG) $(CFG_ISL0) &&\  
77     $(MAKE) && $(INSTALL)-strip  
78  
79 .PHONY: gcc0  
80 gcc0: $(SRC)/$(GCC)/README  
81     rm -rf $(TMP)/$(GCC) && mkdir $(TMP)/$(GCC) &&\  
82     cd $(TMP)/$(GCC) &&\  
83     $(SRC)/$(GCC)/$(BCFG) $(CFG_GCC0) --enable-languages="c"  
84     make gccall
```

```
85  
86 .PHONY: gccall  
87 gccall:  
88     cd $(TMP)/$(GCC) && $(MAKE) all-gcc  
89     cd $(TMP)/$(GCC) && $(MAKE) install-gcc  
90     cd $(TMP)/$(GCC) && $(MAKE) all-target-libgcc  
91     cd $(TMP)/$(GCC) && $(MAKE) install-target-libgcc  
92  
93 ,PHONY: gcpp  
94 gcpp:  
95     make gccall  
96     cd $(TMP)/$(GCC) && $(MAKE) all-target-libstdc++-v3  
97     cd $(TMP)/$(GCC) && $(MAKE) install-target-libstdc++-v3  
98     cp -a $(TC)/$(TARGET)/lib/libgcc_s* $(LIB)/  
99  
00 .PHONY: gcc  
01 gcc: $(SRC)/$(GCC)/README  
02     rm -rf $(TMP)/$(GCC) && mkdir $(TMP)/$(GCC) && \  
03     cd $(TMP)/$(GCC) && \  
04     $(SRC)/$(GCC)/$(BCFG) $(CFG_GCC) --enable-languages="c,c++"  
05     make gcpp  
06  
07 .PHONY: gccf  
08 gccf: $(SRC)/$(GCC)/README  
09     rm -rf $(TMP)/$(GCC) && mkdir $(TMP)/$(GCC) && \  
10    cd $(TMP)/$(GCC) && \  
11    $(SRC)/$(GCC)/$(BCFG) $(CFG_GCC) --enable-languages="c,c++,fortran"  
12    make gcpp
```

```
13 ____cd $(TMP)/$(GCC) && $(MAKE) all-target-libgfortran  
14 ____cd $(TMP)/$(GCC) && $(MAKE) install-target-libgfortran
```

## 111.14.7 **binutils**: ассемблер, линкер и утилиты

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

### arch/i386.mk

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

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

### arch/arm.mk

```
1 # arch : ARCH_ARM arm  
2 TARGET = arm-linux-uclibcabi  
3 CFG_ARCH = --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 = --with-cpu=i486 --with-tune=i486 --disable-mmx --disable-3dnow --disable-sse  
8 CFLAGS_CPU = -march=i486 -mtune=i486
```

## 111.14.8 cclibs: библиотеки для сборки gcc

gmp mpfr mpc isl cloog

## 111.14.9 **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:

```
$ make CCACHE= gcc0
```

## 111.14.10 **gcc**: пересборка полного кросс-компилятора Си/*C<sub>+</sub>*

Пакет собирается *после сборки core*.

## 111.14.11 **core**: сборка основной системы

mk/core.mk

```
1 # normal
2 .PHONY: core
3 core: kernel ulibc ramclean gcc busybox ramclean
4
5 # with fortran (gpcf)
6 .PHONY: coref
7 coref: kernel ulibc ramclean gpcf busybox ramclean
```

## 111.14.12 **kernel**: ядро Linux

mk/kernel.mk

```
1 CFG_KERNEL = ARCH=$(ARCH) \
```

```
2 ____INSTALL_HDR_PATH=$(ROOT) INSTALL_MOD_PATH=$(ROOT)
3
4 .PHONY: kernel
5 kernel: $(SRC)/$(KERNEL)/README
6 # 1
7 ____cd $(SRC)/$(KERNEL) && make $(CFG_KERNEL) distclean
8 ____cd $(SRC)/$(KERNEL) && make $(CFG_KERNEL) allnoconfig
9 # 2
10 ____cat kernel/all >> $(SRC)/$(KERNEL)/.config
11 ____cat kernel/arch/$(ARCH) >> $(SRC)/$(KERNEL)/.config
12 ____cat kernel/cpu/$(CPU) >> $(SRC)/$(KERNEL)/.config
13 ____cat kernel/hw/$(HW) >> $(SRC)/$(KERNEL)/.config
14 ____cat kernel/app/$(APP) >> $(SRC)/$(KERNEL)/.config
15 # 3
16 ____echo "CONFIG_CROSS_COMPILE=\"$(TARGET)-\" >> $(SRC)/$(KERNEL)/.config
17 ____echo "CONFIG_LOCALVERSION=\"-$(HW)$(APP)\" >> $(SRC)/$(KERNEL)/.config
18 ____echo "CONFIG_DEFAULT_HOSTNAME=\"$(HW)$(APP)\" >> $(SRC)/$(KERNEL)/.config
19 # 4
20 ____cd $(SRC)/$(KERNEL) && make $(CFG_KERNEL) menuconfig
21 # 5
22 ____cd $(SRC)/$(KERNEL) && $(MAKE) $(CFG_KERNEL)
23 ____cd $(SRC)/$(KERNEL) && $(MAKE) $(CFG_KERNEL) modules_install
24 # 6
25 ____make kernel-$(ARCH)-fix
26 ____cp $(SRC)/$(KERNEL)/arch/$(ARCH)/boot/zImage $(BOOT)/$(HW)$(APP).kernel
27 # 7
28 ____cd $(SRC)/$(KERNEL) && make $(CFG_KERNEL) headers_install
29
```

```
30 .PHONY: kernel-i386-fix
31 kernel-i386-fix:
32     cp \
33     $(SRC)/$(KERNEL)/arch/$(ARCH)/boot/bzImage \
34     $(SRC)/$(KERNEL)/arch/$(ARCH)/boot/zImage
35
36 .PHONY: kernel-arm-fix
37 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 <pxf>-(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 ≤ 1Gb ОЗУ
  - KERNEL\_GZIP=y ядро сжимать алгоритмом gzip<sup>5</sup>
  - 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 времени<sup>6</sup> ??
    - \* PREEMPT=y вытесняющая многозадачность в режиме ядра
    - \* HZ\_1000=y системный таймер 1KHz
  - 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
    - \* TTY=y последовательные порты и командная консоль
    - \* UNIX98\_PTYS=n псевдотерминалы UNIX98
    - \* LEGACY\_PTYS=n псевдотерминалы UNIX/BSD

<sup>5</sup> важно для загрузчиков на не-i386 системах

<sup>6</sup> повышенная или гарантированная отзывчивость системы на внешние события

- **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\_CLUT24=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** последоваельные порты
  - \* **SERIAL\_8250=y** UART 8250/16550
  - \* **SERIAL\_8250\_CONSOLE=y** командандная консоль на COM-портах
- **VGA\_CONSOLE=y** команданная 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
- **cput 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 поддерку FrameBuffer,..
- **INPUT\_MOUSE=n** мыши

## 111.14.13 ulibc: библиотека uClibc

mk/ulibc.mk

```

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

```

```

23 # 7
24 cd $(SRC)/$(ULIBC) && $(MAKE) $(CFG_ULIBC) install_utils
25 # 8
26 cd $(SRC)/$(ULIBC) && $(MAKE) $(CFG_ULIBC) hostutils
27 cp $(SRC)/$(ULIBC)/utils/ldd.host $(TC)/bin/$(TARGET)-ldd
28 cp $(SRC)/$(ULIBC)/utils/ldconfig.host $(TC)/bin/$(TARGET)-ldconfig
29 cp $(SRC)/$(ULIBC)/utils/getconf.host $(TC)/bin/$(TARGET)-getconf
30 # 9 (in root package)
31 # ldconfig in root.mk

```

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

- `UCLIBC_HAS_FPU=y` всегда включено: используется эмулятор в ядре
  - `UCLIBC_HAS_FLOATS=y` функции плавающей точки
  - `DO_C99_MATH=n` функции float math C99
  - `UCLIBCCTOR_DTOR=y` конструктор/деструктор
  - `UCLIBC_HAS_LFS=y` большие файлы
- *разделяемые shared библиотеки*
    - `HAVE_SHARED=y`
    - `LDSO_CACHE_SUPPORT=y /etc/ld.so.conf`
    - `LDSO_PRELOAD_ENV_SUPPORT=n` переменная `LD_PRELOAD` содержит список библиотек, загружаемых первыми
    - `LDSO_LD_LIBRARY_PATH=n` отключить переменную `LD_LIBRARY_PATH` со списком каталогов поиска shared библиотек
  - *многопоточные threads программы*
    - `LINUXTHREADS_OLD=y` старый вариант потоков, новый вызывает segfault при использовании ключей gcc -lpthread и использовании shared ulibc
  - *параметры установки*
    - `DOSTRIP=y`
    - `RUNTIME_PREFIX=`
    - `DEVEL_PREFIX=`
  - *опции необходимые busybox*
    - `UCLIBC_HAS_CTYPE_TABLES=y`
    - `UCLIBC_HAS_NETWORK_SUPPORT=y`
    - `UCLIBC_HAS_FNMATCH=y`
    - `UCLIBC_HAS_GNU_GETOPT=y`
    - `UCLIBC_HAS_REGEX=y`
    - `UCLIBC_SUSV3_LEGACY=y`

```
1 # CPU
2
3 ARCH_HAS_MMU=y
4 ARCH_USE_MMU=y
5
6 # FPU/floats
7
8 UCLIBC_HAS_FPU=y
9 UCLIBC_HAS_FLOATS=y
10 DO_C99_MATH=n
11 UCLIBC_HAS_LONG_DOUBLE_MATH=n
12
13 # shared libs and c++ stuff
14
15 HAVE_SHARED=y
16 LDSO_CACHE_SUPPORT=y
17 LDSO_PRELOAD_ENV_SUPPORT=n
18 LDSO_LD_LIBRARY_PATH=n
19
20 UCLIBCCTOR_DTOR=y
21 LINUXTHREADS_OLD=y
22
23 UCLIBC_LINUX_SPECIFIC=y
24 UCLIBC_LINUX_MODULE_26=n
25 UCLIBC_HAS_LFS=y
26
27 UCLIBC_HAS_REALTIME=y
```

```
28 UCLIBC_HAS_STRING_GENERIC_OPT=y
29 UCLIBC_HAS_STRING_ARCH_OPT=y
30
31 DOSTRIP=y
32 RUNTIME_PREFIX=""
33 DEVEL_PREFIX=""
34
35 # modular kernel
36 UCLIBC_LINUX_MODULE_26=y
37
38 # net
39 UCLIBC_HAS_NETWORK_SUPPORT=y
40 UCLIBC_HAS_RESOLVER_SUPPORT=y
41 UCLIBC_HAS_IPV4=y
42 UCLIBC_HAS_IPV6=n
43
44 # i18n disabled
45 UCLIBC_HAS_WCHAR=n
46
47 # BB needs
48 UCLIBC_HAS_CTYPE_TABLES=y
49 UCLIBC_HAS_FNMATCH=y
50 UCLIBC_HAS_GNU_GETOPT=y
51 UCLIBC_HAS_REGEX=y
52 UCLIBC_SUSV3_LEGACY=y
53
54 # Python needs
55 DO_C99_MATH=y
```

```
56 #UCLIBC_HAS_LONG_DOUBLE_MATH=y
57 UCLIBC_HAS_WCHAR=y
58 UCLIBC_SUSV4_LEGACY=y
59 #UCLIBC_HAS_PTY=y
60 #UNIX98PTY_ONLY=y
61 #UCLIBC_HAS_LIBUTIL=y
62
63 #DOMULTI=y
64 #UCLIBC_HAS_COMPAT_RES_STATE=n
65 #UCLIBC_HAS_SYSLOG=y
66 ## ALSA needs
67 #UCLIBC_HAS_BSD_ERR=y
68
69 # canadian cross needs
70 #UCLIBC_HAS_WCHAR=y
```

ulibc/arch/i386

```
1 TARGET_i386=y
```

ulibc/cpu/i486sx

```
1 CONFIG_486=y
```

## 111.14.14 gcc: пересборка полного gcc

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

## 111.14.15 busybox: набор утилит busybox

mk/busybox.mk

```
1 CFG_BUSYBOX = \
2     __CONFIG_PREFIX=$(ROOT) \
3     __CROSS_COMPILE=$(TARGET)- \
4     __SYSROOT=$(ROOT) \
5     __CONFIG_EXTRA_CFLAGS="$(TOPT)"
6
7 .PHONY: busybox
8 busybox: $(SRC)/$(BUSYBOX)/README
9     # 1
10    cd $(SRC)/$(BUSYBOX) && $(MAKE) $(CFG_BUSYBOX) distclean
11    cd $(SRC)/$(BUSYBOX) && $(MAKE) $(CFG_BUSYBOX) allnoconfig
12     # 2
13    cp app/${APP}.bb $(SRC)/$(BUSYBOX)/.config
14    cd $(SRC)/$(BUSYBOX) && $(MAKE) $(CFG_BUSYBOX) menuconfig
15    cp $(SRC)/$(BUSYBOX)/.config app/${APP}.bb
16     # 3
17    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*.

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

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

111.14.18 **user**: сборка пользовательского кода

111.14.19 **root**: формирование корневой файловой системы

mk/root.mk

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

```
20 # cat $(PACK)/allfiles > $(PACK)/rootfiles
21 # 7
22 cd $(ROOT) && cat $(PACK)/rootfiles | cpio -o -H newc > $(BOOT)/$(HW)$(APP).cpio
23 cat $(BOOT)/$(HW)$(APP).cpio | gzip -9 > $(BOOT)/$(HW)$(APP).rootfs
```

1. пересоздание `/etc`
2. генерация кеша динамических библиотек `/etc/ld.so.cache`
3. `/sbin/init` → `/init`
4. `/share/`
5. создание `initrd` используя фильтрацию файлов регулярным выражением в переменной `ROOTREX`

#### `/etc/inittab`

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

#### `/etc/init.d/rcS`

```
1#!/bin/sh
2# system dirs
3mkdir /tmp
4mkdir /proc ; mount -t proc proc /proc
5mkdir /sys ; mount -t sysfs sysfs /sys
6# hotplug device manager (automount, firmware boot, config)
```

```
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**, обрабатывающего **devfs**: создание/удаление устройств в **/dev**, автомонтирование, загрузка прошивок,..
3. вывод **README**

/etc/init.d/rcD

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

## 111.14.20 boot: сборка загрузчика syslinux/grub/u-boot

mk/boot.mk

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

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

### 111.14.21 syslinux

### 111.14.22 grub

### 111.14.23 uboot

### 111.14.24 emu: запуск собранной системы в эмуляторе

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

mk/emu.mk

```
1 include app/$(APP).qemu
2
```

```
3 QEMU_ALL = -m 64M -net nic -net user -localtime -append "$(QEMU_VGA)"  
4 QEMU_SERIAL_LOG = -serial file:ttyS0.log -append "console=ttyS0,115200"  
5  
6 .PHONY: emu  
7 emu: $(BOOT)/$(HW)$(APP).kernel $(BOOT)/$(HW)$(APP).rootfs  
8     qemu-system-$(ARCH) $(QEMU_CFG) $(QEMU_ALL) \  
9     --kernel $(BOOT)/$(HW)$(APP).kernel \  
10    --initrd $(BOOT)/$(HW)$(APP).rootfs  
11  
12 .PHONY: emucluster  
13 emucluster: $(BOOT)/$(HW)$(APP).kernel $(BOOT)/$(HW)$(APP).rootfs  
14     make emu &  
15     make emu &  
16 #    make emu &  
17 #    make emu &  
18  
19 .PHONY: emuk  
20 emuk: $(BOOT)/$(HW)$(APP).kernel  
21     qemu-system-$(ARCH) $(QEMU_CFG) $(QEMU_ALL) \  
22     --kernel $(BOOT)/$(HW)$(APP).kernel \  
23     $(QEMU_SERIAL_LOG)  
24  
25 .PHONY: emuiso  
26 emuiso: $(BOOT)/$(HW)$(APP).iso  
27     qemu-system-$(ARCH) $(QEMU_CFG) \  
28     --boot d --cdrom $<  
29  
30 .PHONY: bochs
```

```
31 bochs: $(BOOT)/$(HW)$(APP).iso  
32 bochs -f syslinux/bochs.rc
```

## hw/qemu386.mk

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

Переменная **QEMU\_CFG** задает параметры запуска **QEMU**:

-m	объем ОЗУ на эмулируемой системе
-net none	отключить сеть и iPXE boot
-kernel	прямая загрузка ядра Linux из файла
-initrd	образ корневой файловой системы ( <b>initrd</b> )
-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
```

111.15 **netboot**: Сетевая загрузка

111.16 Прошивка на устройство

111.17 RT-патч

111.18 **math**: компьютерная математика

111.18.1 **blas**: библиотека **BLAS**

111.18.2 **lapack**: библиотека **LAPACK**

111.18.3 **octave**: система численных методов **Octave**

111.18.4 **maxima**: система аналитической математики **Maxima**

111.19 SDK: расширения для **on-board** разработки

Иногда на целевой системе требуется набор средств программирования, для выполнения

- сложных скриптов (Python 111.19.6),

- математических вычислений<sup>7</sup> (GCL 111.19.7/Maxima 111.18.4),
  - 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 SDK<sup>8</sup>.

### 111.19.1 canadian: сборка binutils канадским крестом

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

- build платформа, на которой выполняется сборка, т.е. ваш билд-сервер: x86\_64 или i686
- host платформа, на которой будет выполняться собранный кросс-компилятор, т.е. TARGET нашей встраиваемой системы: i386-linux-uclibc, arm-linux-gnueabihf или что-то подобное
- target платформа или микропроцессор, для которого кросс-компилятор будет генерировать код, например Cortex-M3.

mk/sdk/canadian.mk

```
1 # canadian cross
2
3 CFG_CAN_BIN = \
4     --with-native-system-header-dir=/include \
5     --enable-lto --disable-multilib
6 CFG_CAN_GCC = $(CFG_CAN_BIN) \
7     --enable-threads --enable-libgomp \
```

<sup>7</sup> в т.ч. и символьных (аналитических, не численных)

<sup>8</sup> [S]oftware [D]evelopment [K]it

```
8 ____--enable-languages="c,c++"
9 #--program-prefix=$(P)
10
11 .PHONY: canadian
12 canadian: $(SRC)/$(BINUTILS)/README $(SRC)/$(GCC)/README
13 #__# binutils
14 #__rm -rf $(TMP)/$(BINUTILS) && mkdir $(TMP)/$(BINUTILS) && \
15 #__cd $(TMP)/$(BINUTILS) && \
16 #__$(XPATH) $(SRC)/$(BINUTILS)/$(CACFG) \
17 #__$(CFG_CAN_BIN) --target=$(T) $(O) --prefix=$(PFX) --with-sysroot=$(SR) && \
18 #__$(MAKE) && $(PINSTALL)-strip && \
19 #__grep $(ROOT) $(PACK)/.strace > $(PACK)/binutils.$(PK).strace && rm $(PACK)/.strace
20 #__# gcc
21 __rm -rf $(TMP)/$(GCC) && mkdir $(TMP)/$(GCC) && \
22 __cd $(TMP)/$(GCC) && \
23 __$(XPATH) $(SRC)/$(GCC)/$(CACFG) \
24 __$(CFG_CAN_GCC) --target=$(T) $(O) --prefix=$(PFX) --with-sysroot=$(SR)
25 __cd $(TMP)/$(GCC) && $(MAKE) all-gcc
26 #__cd $(TMP)/$(GCC) && $(MAKE) install-gcc
27 #__cd $(TMP)/$(GCC) && $(MAKE) all-target-libgcc
28 #__cd $(TMP)/$(GCC) && $(MAKE) install-target-libgcc
29
30 .PHONY: binhost
31 binhost:
32 __make canadian T=$(TARGET) PFX=$(USR) SR=/ PK=host O=
33 #__ O=$(CFG_ARCH) $(CFG_CPU)"
34
35 .PHONY: bin486
```

```
36 bin486 :  
37     ____make canadian T=i486-elf O=  
38 #____ O="--with-cpu=i486"  
39  
40 .PHONY: binavr  
41 binavr:  
42     ____make canadian T=avr O=  
43 #____ O=  
44  
45 .PHONY: bincmx  
46 bincmx:  
47     ____make canadian T=arm-none-eabi O=  
48 #____ P=cmx- \  
49 #____ O="--enable-interwork --disable-multilib"
```

## 111.19.2 **binhost**: binutils для хост-процессора

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

## 111.19.3 **binavr8**: binutils для Atmel ATmega (AVR8)

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

## 111.19.4 bincmx: binutils для ARM Cortex-Mx

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

## 111.19.5 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: $(SRC)/fpccore-$(FPC_VER)/Makefile
12 #____cd $(SRC)/fpccore-$(FPC_VER)/fpccore && \
13 #____make distclean && \
14 #____make all $(FPC_CFG) && \
15 #____make install $(FPC_CFG)
16 ##____ CPU_TARGET=i386 FPC=ppc386
17 #
18 #$(SRC)/fpccore-$(FPC_VER)/Makefile:
19 #____cd $(SRC) ; tar zx < $(GZ)/fpccore-$(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
```

111.19.6 **python**: интерпретатор Python

111.19.7 **gcl**: GNU Common Lisp

## Глава 112

# Применение Linux для новой “железки”

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

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

В 111 приведено полное описание системы сборки для создания пакета кросс-компиляции. Но этот вариант (полнейшей пересборки прошивки) подходит вам только в случае, если вы полностью определяете состав ПО.

Очень часто требуется сохранить прошивку и сборку Linux-системы от поставщика или производителя оборудования, т.к. в нее могут входить части, поставляемые в бинарном виде, vendor-specific патчи на ядро, и подобное закрытое ПО в виде так называемых “бинарных блобов”. Чаще всего этим страдают мобильные аппаратные платформы, в т.ч. Raspberry Pi: производитель SoC<sup>1</sup> предоставляет часть драйверов в виде за-

---

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

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

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

В качестве примера разберем добавление конфигурации для двух железок:

1. VoCore 111.12.3
2. ТионПро270 111.11.4

# Глава 113

## 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 eLinux, 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 111. Сейчас активно развивается ветка **SDL2**, ее особенности кратко рассмотрены в разделе 113.4.

## 113.1 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

## 113.2 LazyFoo tutorial

<sup>1</sup>

### 113.2.1 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.

---

<sup>1</sup> копипаста: [http://lazyfoo.net/SDL\\_tutorials/](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](#).

## 113.2.2 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](#).

### 113.2.3 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

### 113.2.4 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.



**113.2.5 Color Keying**

**113.2.6 Clip Blitting and Sprite Sheets**

**113.2.7 True Type Fonts**

**113.2.8 Key Presses**

**113.2.9 Mouse Events**

**113.2.10 Key States**

**113.2.11 Playing Sounds**

**113.2.12 Timing**

**113.2.13 Advanced Timers**

**113.2.14 Regulating Frame Rate**

**113.2.15 Calculating Frame Rate**

**113.2.16 Motion**

**113.2.17 Collision Detection**

**113.2.18 Per-pixel Collision Detection**

**113.2.19 Circular Collision Detection**

**113.2.20 Animation**

### 113.3.1 Графический Hello World

user/sdl\_hello.c

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



### 113.3.2 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 }
```

## 113.4 Версия SDL2

Глава 114

BuildRoot

Глава 115

Особенности OpenWrt

## Часть XIV

x86os: простая операционная система  
для компьютера на i386 процессоре

# Глава 116

## Ресурсы

[http://www.jamesmolloy.co.uk/tutorial\\_html/index.html](http://www.jamesmolloy.co.uk/tutorial_html/index.html)

<http://wiki.osdev.org>

Библиотека системного программиста © Александр Фролов, Григорий Фролов:

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

Графика:

- Уилтон Р.

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

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

# Глава 117

## Структура

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

# Глава 118

## 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 119

## Сборка кросс-компилиатора

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

make cross
```

cfg.mk

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

mk/versions.mk

```
1 # cross compiler
```

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

### mk/packages.mk

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

### mk/dirs.mk

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

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

### mk/src.mk

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

### mk/cross.mk

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

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

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

### mk/cclibs.mk

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

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

Как вы можете заметить, структура файлов и каталогов похожа на [azlin /111/](#).

# Глава 120

## Формат ELF32

Подробнее см. формат ELF 70 и скрипты линкера 72.4

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

```
make kernel
```

формат файла elf32-i386

архитектура i386

HAS\_SYMS в файл включена отладочная информация  
об идентификаторах (“символах”)

start address 0x00100000 стартовый адрес загрузки ядра 1 Мб

Бинарный код делится на секции, или сегменты:

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

CONTENTS

ALLOC

LOAD

READONLY

CODE

DATA

kernel.objdump

# Глава 121

## multiboot

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

загрузчик SysLinux

Спецификация Multiboot 0.6.96

Перевод Multiboot Specification

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

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

`kernel/x86os.ld`

1 ENTRY(\_multiboot)

2

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

### kernel/multiboot.S

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

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

# Глава 122

## Микроядро

```
make kernel  
make emu
```

kernel/kernel.mk

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

kernel/kernel.c

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

# Глава 123

## Драйвера

### 123.1 **vga**: текстовая консоль VGA 80×25

include/driver/vga.h

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

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

driver/vga.c

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

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

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

123.2 **kbd**: клавиатура

123.3 **ide**: жесткий диск IDE

123.3.1 **fatfs**: файловая система FAT16

## Часть XV

Символьная и численная математика

В практике любого инженера математика играет важнейшую 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 технической литературы, особенно если вам нужно разобраться в какой-либо прикладной области. Очень часто приходится реализовывать некоторые численные методы вычислений, выполняющиеся в вашем устройстве в реальном времени, для управления процессами, обработки сигналов с датчиков, принятия решений о включении исполнительных устройств и т.п. Ну и конечно вы не сможете создать само устройство, не понимая принципы его работы 😊. Это конечно не относится к различным простейшим устройствам типа таймеров или простой автоматики, но стоимость заказов такого типа → 0.

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

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

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

# Глава 124

## Общие сведения о компьютерной математике

1 2

Для начала пару слов о том, что из себя представляют эти самые **символьные** или, как их еще называют, **аналитические вычисления**, и их отличие от численных расчетов. Компьютеры, как известно, оперируют с числами, целыми и с плавающей запятой<sup>3</sup>. 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.

---

<sup>1</sup> Тихон Тарнавский. Maxima — максимум свободы символьных вычислений

<sup>2</sup> <http://maxima.sourceforge.net/ru/maxima-tarnavsky-1.html>

<sup>3</sup> на самом деле настоящую “плавучку” поддерживают только достаточно мощные процессоры, не хуже i486dx, встраиваемые не-DSP CPU/MCU аппаратно работают только с целыми числами:  $\pm 127$ ,  $\pm(2^{16} - 1)$  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 **CAS**: [C]omputer [A]lgebra [S]ystem.

Среди математического ПО для аналитических (символьных) вычислений наиболее 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 и терминами предметной области, т.е. почти математическими формулами.

Такие программы можно назвать *средой математического 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 125

## Пакет Octave

**Octave** – пакет CAS для численных вычислений.

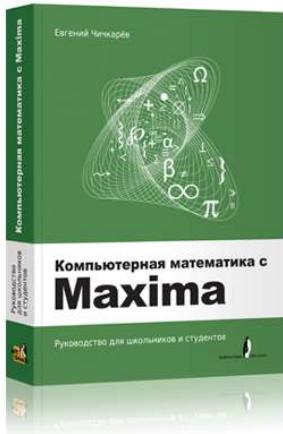


Е.Р. Алексеев, О.В. Чеснокова  
**Введение в Octave для инженеров и математиков**  
<http://www.altlinux.org/Books:Octave>

# Глава 126

## Пакет Maxima

Maxima – пакет CAS символьной математики.



Евгений Чичкаров

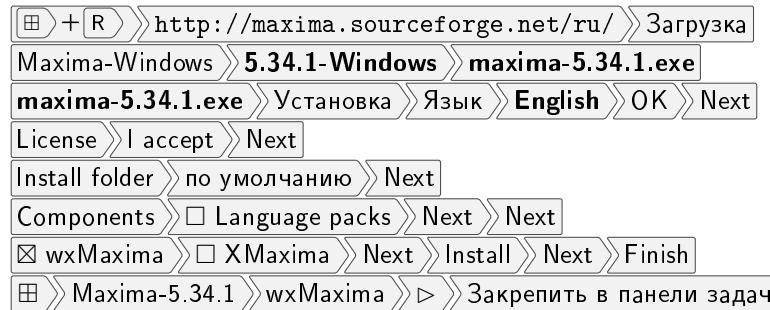
Компьютерная математика с Maxima. Руководство для школьников и студентов

<http://www.altlinux.org/Books:Maxima>

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>.

## 126.1 Установка Maxima под Windows



# Глава 127

## Параллельные вычисления

127.1 Многоядерные архитектуры с разделяемой памятью

127.2 Кластер архитектуры Beowulf

127.3 Вычисления на GPU

127.4 Средства параллелизации  $C_+$

127.5 BLAS/LAPACK/MPI/Scalapack

127.6 Средства измерения производительности

## Часть XVI

Подготовка технической документации

## Глава 128

### Верстка в LATEX

Глава 129

Оформление листингов

# Глава 130

## Подготовка иллюстраций

130.1 GIMP

130.2 Inkscape

130.3 Graphviz

# Глава 131

Замечания для соавторов “Абзуки  
АРМатурщика”

## Часть XVII

Примерные учебные планы

Глава 132

Блондинко

# Глава 133

## Школотрон

1. ??; ??

постараться понять разделы

??

??

??

??

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

2. ??; ??

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

# Глава 134

## Студень

1. ??

# Глава 135

## Технический специалист

1. ?? прочитать по диагонали

Часть XVIII

Куча

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

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

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

## Глава 136

Автоматное программирование  
/фреймворк QuantumLeaps/

# Глава 137

## Запуск Linux на android устройстве в режиме паразита

<sup>1</sup> © 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.

---

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

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

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

## Features

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

## Requirements (click on each topic for more info)

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

## Steps

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

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

## Partition the SDcard

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

At your desktop computer

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

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

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

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

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

Create new file systems with mkfs.

At your desktop computer

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

## Creating the initramfs

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

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

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

From your desktop shell to your Android device

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

/init in your new initramfs

Precompiled busybox from busybox.net

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

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

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

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

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

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

## Creating the Debian root file system

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

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

At your desktop computer

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

## On your Android device

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

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

- Ready made root file system (128MB)

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

## Creating the new Android root file system

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

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

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

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

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

## Some finishing scripts to tie all together

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

/etc/init of SDcard ext4 filesystem

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

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

The secondary script init.stage2

/etc/init.stage2 of the SDcard ext4 filesystem

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

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

My /etc/rc.local looks like below.

/etc/rc.local of the SDcard ext4 filesystem

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

## Install it

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

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

On you desktop computer

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

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

## Additional tinkering

/etc/group

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

On an android device as user shell

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

/etc/mtab

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

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

## locales

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

## Setting the system default time zone

# Using the GNU/Linux Debian environment

## Connectivity

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

## Connect using SSH to localhost

## Connect using local connection

## X11 Window

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

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

## On your Android device in the Debian system via SSH

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

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

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

## File access

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

In Gnome (Nautilus) at my workstation

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

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

Search Google Play for Media Scanners

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

## Modifying the system

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

Before

After

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

Rooted

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

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

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

Happy hacking!

Caveats

Environment variables and file descriptors

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

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

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

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

Some warnings

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

Or to conclude, always keep a backup.

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

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

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

#### Common mistakes

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

Make sure all mount points are there.

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

#### About Performance

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

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

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

#### pstree

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

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

#### Devices verified

Implemented successfully on the following Android devices.

- LG Optimus P700

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

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

/By Mikael Q Kuisma

# Предметный указатель

программы

memtest86+, 296

syslinux, 296

термины

), 278

аналитические вычисления, 424

бэкенд компилятора, 278

численные методы, 425

фронтенд компилятора, 278

грамматика, 271

канадский крест, 361

кросс-компиляция, 295

кросс-тулчейн, 295

лексема, 270

лексер, 270

лексический анализ, 271

лексический анализатор, 272

падстек, 157, 170

пакет, 304

приближенное решение, 425

промежуточное представление, 278

регулярное выражение, 270

сборка, 295

сборка пакета, 304

сегмент, 125, 411

секция ELF, 125, 411

семантический анализ, 279

символьная математика, 424

синтаксический анализ, 274

сканер, 270

слой печатной платы, 169

токен, 270

учебный план, 2

вендорная сборка, 367

CAS, 425

Data Definition Language, 269

DDL, 269

ELF, 411

emLinux, 295  
multiboot, 413  
plain text, 269  
USB  
    хост-контроллер, 178, 290  
    On-The-Go, 178, 290  
    OTG, 178, 290  
    URB, 178, 290  
VCS, 118

## azLinux

настройка  
    приложение, 318  
пакет  
    dirs, 311  
переменная  
    APP, 318  
    BOOT, 313  
    BUILD, 312  
    DIRS, 313  
    GZ, 312  
    ROOT, 312  
    SRC, 312  
    TC, 312  
    TMP, 312  
приложение  
    clock, 318

micro, 318  
скрипт  
    Makefile, 306  
железо, 319  
    AR7240, 332  
    ASUS Eee PC 701, 322  
    BlackSwift, 330  
    Celeron1037U, 332  
    CeleronM, 331  
    Gigabyte GA-C1037UN-EU, 324  
    i386, 320  
    i486sx, 331  
    QEMU, 321, 325, 328  
    RT5350, 332  
    VoCore, 330