

# Азбука халтурщика-ARMатурщика

## лабораторные работы

учебный курс по микроконтроллерам Cortex-Mx:  
Миландр 1986BE, STM32F, LPC21xx

(сорупаста) Понятов Д.А. <[dponyatov@gmail.com](mailto:dponyatov@gmail.com)>, ИКП СГАУ

20 июля 2014 г.

# Лабораторные работы

<b>Установка ПО</b>	<b>1</b>
LP1: Установка Debian GNU/Linux	2
LP2: Установка Git	3
LP3: Установка GNU toolchain	5
LP4: Установка утилит GnuWin32	8
LP5: Установка MinGW	9
LP6: Редактирование системной переменной Windows <b>\$PATH</b>	9
LP7: Установка Java	12
LP8: Установка IDE $\ominus$ ECLIPSE	13
LP9: Установка симулятора QEMU	18
LP10: Установка системы верстки документации L <sup>A</sup> T <sub>E</sub> X	19
LP11: Создание нового проекта в $\ominus$ ECLIPSE	21
 <b>Первые шаги</b>	 <b>21</b>
LP12: Создание Makefile	22
LP13: Hello World	29


# ЛР1: Установка Debian GNU/Linux

## ЛР2: Установка Git

Создадим рабочий каталог, установим систему контроля версий Git?? и получим локальную копию проекта этой книги, содержащий кроме текста для издательской системы L<sup>A</sup>T<sub>E</sub>X еще и исходные коды библиотек, примеры кода и т.п., которые вы захотите использовать в своих проектах.

 +  <http://git-scm.com/download/win>

Запуститься закачка установочного пакета scm-git (**Git-1.9.4-preview20140611.exe**), после его загрузки запустите установщик,


Welcome  Next

GNU GPL  Next

Select components  Windows Explorer Integration  Simple Context Menu  Git GUI here  Next

Use Git and optional Unix tools from the Command Prompt  Next

Use OpenSSH  Next

Checkout Windows-style  Next

Extracting files...

Completing Setup  ☐ View ReleaseNotes  Finish

Проверим что Git правильно установился:

 +  cmd

```
1 C:\Documents and Settings\pda>git --version
2 git version 1.9.4.msysgit.0
```

Первое, что вам следует сделать после установки Gita —указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Gite содержит эту информацию, и она включена в коммиты, передаваемые вами:

```
1 C:\Documents and Settings\pda>git config --global user.name "Vasya Pupkin"
2 C:\Documents and Settings\pda>git config --global user.email no@mail.com
3 C:\Documents and Settings\pda>git config --global push.default simple
```

Эти настройки достаточно сделать только один раз, поскольку в этом случае Git будет использовать эти данные для всего, что вы делаете. Если для каких-то отдельных проектов вы хотите указать другое имя или электронную почту, можно выполнить эту же команду без параметра `--global` в каталоге с нужным проектом.


Создаем каталог **D:/ARM** и выгружаем текущую копию этой книги из репозитория <https://github.com/ponyatov/CortexMx>, создавая свой собственный локальный [репозиторий проекта](#).

 +  cmd





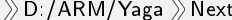




```
1 C:\Documents and Settings\pda>D:
2 D:\>mkdir \ARM
3 D:\>cd \ARM
4 D:\ARM>git clone --depth=1 https://github.com/ponyatov/CortexMx.git book
```

## ЛР3: Установка GNU toolchain

Самая важная часть — ставим GCC toolchain (набор инструментов) для процессоров ARM, собранный для **\$TARGET = arm-none-eabi**. Вариантов сборок для разработки для ARM под Windows много, есть и такие дистрибутивы как **CooCox IDE**, включающие полный комплект ПО одним пакетом. Ограничимся установкой варианта сборки под названием Yagarto:

  <http://sourceforge.net/projects/yagarto/> 

Запускаем скачанный инсталлятор.

Welcome   
License    
Choose Components  Add YAGARTO to PATH   
Destination folder    
Start Menu Folder    
Installation Complete  

Ygà поставилась, теперь можно проверить что доступны базовые утилиты:

Ассемблер

```
1 C:\Documents and Settings\pda>arm-none-eabi-as --version
2 GNU assembler (GNU Binutils) 2.23.1
3 Copyright 2012 Free Software Foundation, Inc.
4 This program is free software; you may redistribute it under the terms of
5 the GNU General Public License version 3 or later.
6 This program has absolutely no warranty.
7 This assembler was configured for a target of 'arm-none-eabi'.
```

## Линкер

```
1 C:\Documents and Settings\pda>arm-none-eabi-ld --version
2 GNU ld (GNU Binutils) 2.23.1
```

## Утилиты для работы с объектными файлами в формате ELF

```
1 C:\Documents and Settings\pda>arm-none-eabi-objdump --version
2 GNU objdump (GNU Binutils) 2.23.1
```

```
1 C:\Documents and Settings\pda>arm-none-eabi-objcopy --version
2 GNU objcopy (GNU Binutils) 2.23.1
```

## Препроцессор (не компилятор $C^{++}$ )

```
1 C:\Documents and Settings\pda>arm-none-eabi-cpp --version
2 arm-none-eabi-cpp (GCC) 4.7.2
3 Copyright (C) 2012 Free Software Foundation, Inc.
4 This is free software; see the source for copying conditions. There is NO
5 warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

## Компилятор Си

```
1 C:\Documents and Settings\pda>arm-none-eabi-gcc --version
2 arm-none-eabi-gcc (GCC) 4.7.2
```

## Компилятор $C^{++}$

```
1 C:\Documents and Settings\pda>arm-none-eabi-g++ --version
2 arm-none-eabi-g++ (GCC) 4.7.2
```

### Утилита Make



```
1 C:\Documents and Settings\pda>make --version
2 "make" is not internal or external command.
3
4 C:\Documents and Settings\pda>arm-none-eabi-make --version
5 "make" is not internal or external command.
6
7 C:\Documents and Settings\pda>dir D:\ARM\Yaga\bin\*make*
8 Volume D has no label.
9 Serial #: 6588-9778
10
11 Direcorey contents D:\ARM\Yaga\bin
12
13 File not found
```



Упс, а **make** почему-то в комплект не включили ☹. Придется его ставить отдельно в ЛР4.





## ЛР4: Установка утилит GnuWin32

Для совместимости скриптов придется поставить несколько пакетов из **GnuWin32**:

 +  <http://gnuwin32.sourceforge.net/packages.html>

 +  >> coreutils >>  + > Setup

 +  >> wget >>  + > Setup

 +  >> gnu make >>  + > Setup

**coreutils-5.3.0.exe** основные UNIX-утилиты типа rm ls , собранные под win32

Welcome >> Next

License >> Accept >> Next

Folder >> D:/ARM/GnuWin32 >> Next

Components >> Next

Start Menu >> GnuWin32/CoreUtils >> Next

Select Additional >> Next

Ready to Install >> Next

Completing >> Finish

Аналогично ставим:

**make-3.81.exe** утилита make

**wget-1.11.4.exe** консольная утилита загрузки файлов по HTTP/FTP

**grep-2.5.4.exe** утилита поиска строк в файлах и stdin/stdout потоке

## ЛР5: Установка MinGW

Под Windows x64 обнаружилась проблема — если в **Makefile** используется перенаправление вывода команды в файл типа `objdump -xd program.o > program.o.dump` или маркеры сцепления, при выполнении **make** (из пакета **GnuWin32**) завершается с ошибкой

```
make: Interrupt/Exception caught (code = 0xc00000fd, addr = 0x4227d3).
```

Для обхода этой проблемы был найден способ — поставить пакет **MinGW**: это GNU тулчейн для нативной компиляции программ под win32. За счет небольшой потери объема диска получаем решение проблемы с **make** на Windows x64, и заодно получаем возможность компилировать простые вспомогательные утилиты на Си/C<sup>+</sup>.

Кроме того, нужно стараться писать [портابلельный](#) код максимально независимый от платформы<sup>1</sup>, а тестировать платформо-независимость можно, компилируя один и тот же код одновременно и для микроконтроллера, и для выполнения под Windows.

Для совместимости поставим 32-битную версию **MinGW**.



## ЛР6: Редактирование системной переменной Windows \$PATH

Чтобы утилиты **GnuWin32** были доступны, нужно прописать переменную пользователя **\$PATH** в системном окружении.

---

<sup>1</sup>чтобы можно было по необходимости легко и быстро перенести ваш проект на любой другой микроконтроллер, или использовать одни и те же куски кода как на МК, так и на ПК, например процедуры кодирования/декодирования данных или реализации протоколов обмена данными

Пуск > Настройка > Панель управления > Система > Дополнительно > Переменные среды

Переменные среды > переменные пользователя > Создать/Изменить

Имя переменной > PATH

Значение переменной > добавить в начало D:/ARM/GnuWin32/bin;D:/MinGW/bin;D:/ARM/Yaga/bin;..

Ok > Ok > Ok


Проверяем:

```
1 C:\Documents and Settings\pda>ls -la
2 total 3111
3 drwxr-xr-x   29 pda      user          0 Jul  4 14:03 .
4 drwxr-xr-x    9 pda      user          0 Oct  8 2013 ..
5 -rw-r--r--    1 pda      user       5242 May 22 14:29 .bash_history
6 drwxr-xr-x    2 pda      user          0 May 23 2013 .borland
7 drwxr-xr-x   18 pda      user          0 Sep  4 2013 .ccache
8 drwxr-xr-x    3 pda      user          0 Mar 26 2013 .eclipse
```

```
1 C:\Documents and Settings\pda>wget --version
2 GNU Wget 1.7
3
4 Copyright (C) 1995, 1996, 1997, 1998, 2000, 2001 Free Software Foundation, Inc.
5 This program is distributed in the hope that it will be useful,
6 but WITHOUT ANY WARRANTY; without even the implied warranty of
7 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
8 GNU General Public License for more details.
9
10 Originally written by Hrvoje Niksic <hniksic@arsdigita.com>.
```

```
1 C:\Documents and Settings\pda>mingw32-make --version
2 GNU Make 3.82.90
3 Built for i686-pc-mingw32
4 Copyright (C) 1988-2012 Free Software Foundation, Inc.
5 License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
6 This is free software: you are free to change and redistribute it.
7 There is NO WARRANTY, to the extent permitted by law.
```

## ЛР7: Установка Java

Для работы IDE  ECLIPSE требуется установленная Java:



- Минимальный вариант — ставим только Java Runtime:

Java Platform, Standard Edition >> JRE >> Download >> Accept License >> **jre-8u5-windows-i586.exe**  
**jre-8u5-windows-i586.exe** >> Welcome >> ☒ Change destination folder >> Install  
Destination folder >> **D:/Java/jre8** >> Next >> Installing >> Close

- Если вы планируете параллельно еще и осваивать язык Java — ставим Java SE JDK:

Java Platform, Standard Edition >> JDK >> Download >> Accept License >> **jdk-8u5-windows-i586.exe**  
**jdk-8u5-windows-i586.exe** >> Welcome >> Next  
Install to: **D:/Java/jdk8** >> Next  
JRE Destination folder >> Install to: **D:/Java/jre8** >> Next  
Java SE Development Kit 8 Update 5 Successfully Installed >> Close

## ЛР8: Установка IDE ☰ECLIPSE

Для работы IDE ☰ECLIPSE требуется установленная Java ЛР7.

Для установки доступны два варианта:


1. **Eclipse Standard** базовый вариант среды, в ЛР рассмотрен именно он для иллюстрации ручной установки расширений
2. **Eclipse IDE for C/C++ Developers** вариант сборки уже включает расширение CDT, поэтому в следующий раз рекомендуем сразу качать его, это упростит и сэкономит немного времени на установку рабочей среды

☰ + R <http://www.eclipse.org/downloads/>

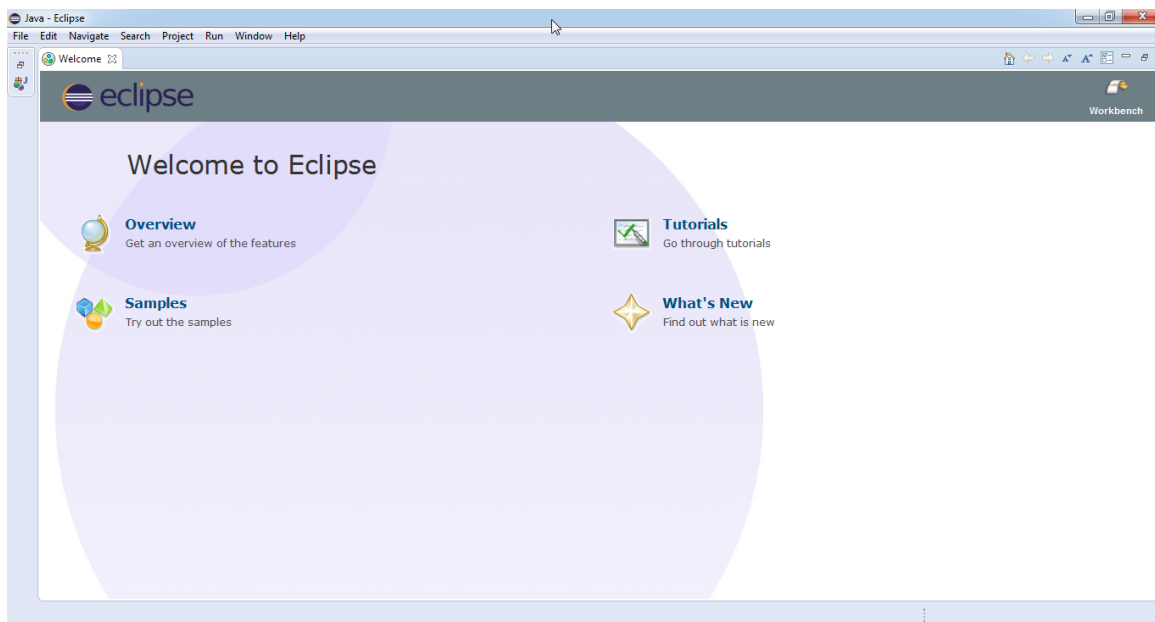
Eclipse Standard >> Windows 32 Bit >> Download >> **eclipse-standard-luna-R-win32.zip**

Перетащите каталог **eclipse** из архива в **D:/ARM** и создайте удобным для вас способом ссылку на **D:/ARM/eclipse/eclipse.exe**.



Workspace — рабочий каталог, в котором создаются каталоги отдельных проектов, типа **D:/WORK**. Eclipse создаст в нем служебный каталог **.metadata**, и поместит в него служебную информацию, относящуюся сразу ко всем проектам. Как побочный эффект, если в workspace уже есть какой-то каталог, можно создать новый проект (например **book**), и в левой части рабочей области  ECLIPSE в окне **Project Explorer** появится дерево файлов **book/\***.

D:/ARM/eclipse/eclipse.exe >> Workspace >> D:/ARM >> Use as default >> OK



Проверяем наличие обновлений

Help >> Check for Updates >> Details >> No updates found >> OK

В базовом варианте Eclipse поддерживает только Java, поэтому нужно установить расширение для работы с C/C++: **CDT**.

Проект **CDT** предоставляет полнофункциональную интегрированную среду для разработки на Си и C<sup>++</sup>. Поддерживаются: управление проектами и компиляцией для различных тулчейнов, стандартная сборка через **make**, навигация по исходным текстам, различные инструменты для работы с исходным текстом, такие как иерархия типов, граф вызовов, браузер подключаемых файлов, браузер макроопределений, редактор кода с подсветкой синтаксиса, сворачивание синтаксических структур (фолдинг) и гипертекстовая навигация, рефакторинг и генерация кода, средства визуальной отладки, включающие просмотр памяти, регистров и дизассемблер.

 +  <http://www.eclipse.org/cdt/downloads.php>

Выделить и скопировать в буфер обмена ссылку

**p2 software repository:** <http://download.eclipse.org/tools/cdt/releases/8.4>.

Добавляем сетевое хранилище пакетов для  ECLIPSE:

 ECLIPSE >> Help >> Install New Software >> Work with >> Add

Name >> CDT

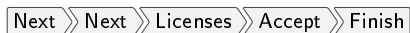
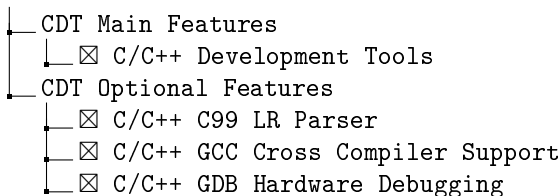
Location >> <http://download.eclipse.org/tools/cdt/releases/8.4>

OK

Выбрать (если оно не выbralось само) хранилище  Work with: >> CDT, и в дереве выбора пакетов выбрать:

CDT



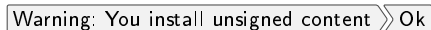
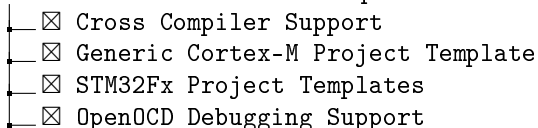



После установки пакетов появится окно с запросом перезапуска  ECLIPSE.

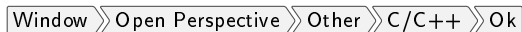
Аналогично ставим плагин GNU ARM Eclipse:



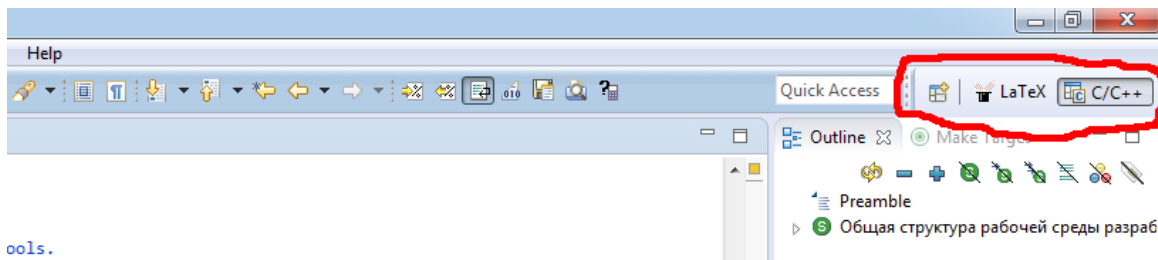
GNU ARM C/C++ Cross Development Tools



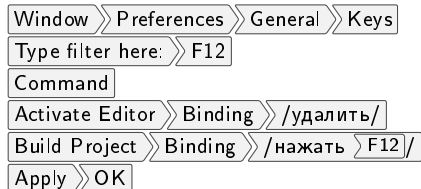
В  ECLIPSE есть так называемые **перспективы** (perspective) — это переключаемые режимы отображения рабочего набора окон, настроенные под тип работы. По умолчанию запускается перспектива **Java**. Нас интересует перспектива **C/C++**:



Также перспективу можно переключить кнопкой на панели в правом верхнем углу:



Для настройки привычных вам клавиш можно сразу зайти в глобальные настройки среды и поменять привязку клавиш:



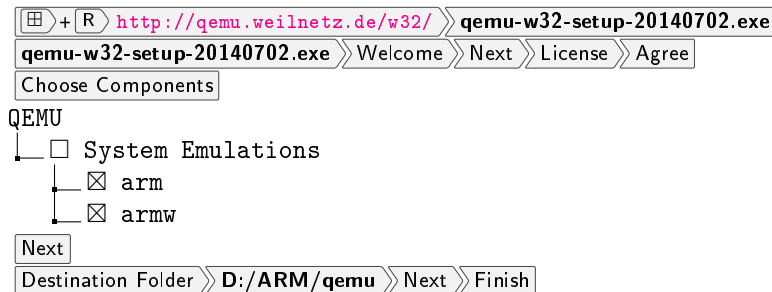
## ЛР9: Установка симулятора QEMU

Нередко в практике разработчика возникают ситуации, когда программное обеспечение (ПО) для микроконтроллера приходится писать в отсутствии под рукой аппаратной платформы.

Например, печатная плата устройства отдана на подготовку к производству, а времени ждать готовое устройство для тестирования на нем программного обеспечения нет.

В таких случаях для оценки работоспособности ПО можно воспользоваться программным симулятором целевого микроконтроллера.

Для интегрированной среды разработки  $\oplus$ ECLIPSE CDT в качестве программного симулятора микроконтроллеров ARM можно использовать симулятор (или виртуальную машину,если быть точным) **qemu-arm** с интерфейсом командной строки:



Добавьте **D:/ARM/qemu** в системную переменную **\$PATH** (ЛР6).

```
1 C:\Documents and Settings\pda>qemu-system-arm -version
2 C:\Documents and Settings\pda>cat D:\ARM\qemu\stdout.txt
3 QEMU emulator version 2.0.90, Copyright (c) 2003-2008 Fabrice Bellard
```

# ЛР10: Установка системы верстки документации $\text{\LaTeX}$

Если вы планируете писать полноценную документацию на программы и оборудование, или участвовать в доделке этой книги, вы можете установить систему верстки  $\text{\LaTeX}$ .

Для работы с  $\text{\TeX}$  требуется довольно приличное по усилиям (само)обучение [12], но оно оправдывается если вы часто пишете документацию, особенно если в ней больше 10 формул. Готовить документацию в  $\text{\M\$ Word}$  — (само)убийство мозга и времени, идеология подстановочных макросов  $\text{\TeX}$ , богатый набор доп.пакетов и командный ввод формул очень доставляют.



Скачайте и установите пакет  $\text{\MiKTeX}$ :

 +  <http://miktex.org/download> > Other Downloads > Net Installer  
Save as: > D:/ARM/soft/MikTeX/miktex-netsetup-2.9.4503

Загрузка дистрибутивных файлов

**miktex-netsetup-2.9.4503** > License > Accept > Далее  
Task > Download > Далее

Если у вас постоянное internet-соединение: Package Set > Basic  $\text{\MiKTeX}$  > Далее

Для offline работы<sup>2</sup> Package Set > Complete  $\text{\MiKTeX}$  > Далее

Download Source > Russian Federation (ctan.uni-altai.ru) > Далее

Distribution Directory > D:/ARM/soft/MikTeX > Далее > Start > Executing > Далее > Close

Установка из ранее загруженного дистрибутива

D:/ARM/soft/MikTeX/miktex-netsetup-2.9.4503 > License > Accept > Далее  
Task > Install > Далее > Basic  $\text{\MiKTeX}$  > Далее

---

<sup>2</sup> когда неизвестно какие пакеты понадобятся —  $\text{\MiKTeX}$  умеет их докачивать по необходимости

Install for >> Anyone/Only for user >> Далее

Install from: >> D:/ARM/soft/MikTeX >> Далее

Install to: >> D:/LaTeX/MiKTeX >> Далее

Settings

Preferred paper >> A4

Важная опция: автоматическая докачка отсутствующих пакетов Install missing packages >> Yes

Далее >> Start >> Executing >> Close

Двухступенчатая установка позволяет сначала скачать полный дистрибутив MiKTeX, а затем установить его на другой компьютер, не подключенный к Internet, или с медленным/платным каналом не дающим взять и качнуть 200 Мб.

Для удобной работы с **.tex** файлами в ECLIPSE нужно поставить дополнение **TeXlipse**:

ECLIPSE >> Help >> Install >> Work with >> Add

Name >> TeXlipse

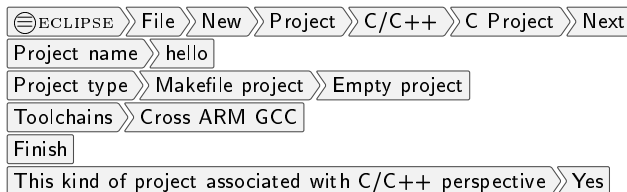
Location >> <http://texlipse.sourceforge.net>

TeXlipse

└─ ☒ TeXlipse

## ЛР11: Создание нового проекта в ECLIPSE

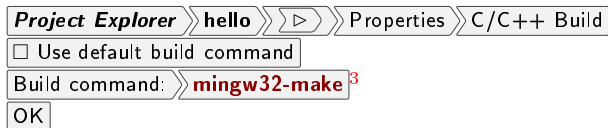
Создадим новый проект, напомним простую программу, и запустим ее в отладчике.



В окне *Project Explorer* появится пустая закладка проекта **hello**. Если вдруг там будут какие-то файлы, значит кто-то до вас уже создал проект, и что-то туда наляпал. В этом случае повторите создание, задав имя типа **hello<номер группы><FIO>** или типа того, для полной уверенности можно сначала посмотреть что в **D:/ARM** нет папки с таким именем.

Нужно сразу настроить несколько свойств проекта.

Команда-билдер для проекта — задаем явно **make**:



---

<sup>3</sup> почему не просто **make** см. ЛР<sup>5</sup>

## LP12: Создание Makefile

Стоит объяснить, почему при создании проекта мы выбрали тип **Makefile project**, хотя были доступны более логичные варианты типа **ARM C Project**.

Утилита **make** ведет свою историю с 70х гг. Компьютеры тогда были большими, тяжелыми, а главное медленными и с очень маленькой памятью (десятки÷сотни Кб). Компиляторам зачастую не хватало памяти, чтобы скомпилировать большую программу. Кроме того, скорость их запуска и работы была тоже черепашье. Поэтому исходный код программы делили на модули, компилировали или ассемблировали каждый модуль по-отдельности в [объектный код](#), а затем уже на конечном этапе с помощью [линкера](#) собирали несколько файлов объектного кода в один исполняемый файл.

Для ускорения и упрощения этого процесса и была создана утилита **make**. Чтобы не вызывать лишний раз компилятор или какой-нибудь транслятор, в файле **Makefile** прописываются зависимости между файлами. Затем запускается **make** с указанием какой файл нам нужно получить, и выполняется цепочка вызовов нужных программ.

Следует отметить, что утилита **make** используется до сих пор для сборки самых современных программных пакетов<sup>4</sup>, правда в комплексе с другими средствами, обеспечивающими переносимость программ между разными ОС и автогенерацией зависимостей из исходного кода.

Для наших целей **make** используется как самое простое средство управления компиляцией проекта. В средах разработки, особенно в коммерческих, используются служебные файлы проектов, иногда бинарные, чаще текстовые, но всегда запутанные и весьма развесистые.

Если вам вдруг понадобится откомпилировать ваш проект на другом компьютере, с другой архитектурой, возможно вообще без графического интерфейса<sup>5</sup>, или вы вдруг решите попробовать работать в другой IDE — вы тут же вляпаетесь в ситуацию, когда нечем открыть файл проекта с заботливо прописанными опциями компиляции.

---

<sup>4</sup> типа GCC 4.9.x, ядра Linux или KDE под FreeBSD

<sup>5</sup> например какой-нибудь удаленный сервер на процессоре 1995BM666 под раскрытым Solaris 7α4, на котором лежит криптобиблиотека, использующая при компиляции трофейный электро-механический энкодер, существующий в единственном экземпляре ☹

```

1 # пример мейкфайла для проекта Азбука ARMатурщика
2 # лабораторная работа ЛР{labmkmake}
3 # символ # в начале -- комментариев
4
5 # пример использования переменных
6
7 # простое присваивание значения переменной
8 # обнуление переменной
9 SOMEVAR =
10 # маски временных файлов
11 TMPFILES = *.o *.elf *.hex *.objdump
12
13 # целевая платформа $TARGET, часто называют "префикс целевой платформы"
14 TARGET = arm-none-eabi
15 # целевой процессор
16 CPU = STM32F100RB
17 # переопределение переменной
18 CPU = LM3S811
19
20 # присваивание переменной с подстановкой значений другой переменной
21
22 # опции целевого процессора для gnu as и gnu cc
23 CPUOPT = -mcpu=cortex-m3 -mthumb -DCPU=$(CPU)
24
25 # стандартные переменные, задающие команды ассемблера, компилятора и линкера

```



```
26 AS = $(TARGET)-as
27 CC = $(TARGET)-gcc
28 LD = $(TARGET)-ld
29 OBJDUMP = $(TARGET)-objdump
30 OBJCOPY = $(TARGET)-objcopy
31 SIZE = $(TARGET)-size
32 GDB = $(TARGET)-gdb
33 MAKE = mingw32-make
34 EMU = qemu-system-arm -M lm3s811evb -S -s -kernel
35
36 # нестандартная (?) переменная - опции оптимизации
37 OPTFLAGS = -O1
38 # опции генерации отладочной информации, для отладки _выключить оптимизацию_
39 DEBFLAGS = -g -O0
40 # стандартная переменная - флаги компилятора Си
41 CFLAGS = $(CPUOPT) $(OPTFLAGS) $(DEBFLAGS)
42 # флаги ассемблера
43 ASFLAGS = $(CPUOPT) $(DEBFLAGS)
44
45 # указание что цели all и clean являются фиктивными целями, а не файлами
46 .PHONY: all clean deb emu
47
48 # первая цель, заданная в Makefile, является целью по умолчанию
49 # и обрабатывается при вызове $(MAKE) без параметров
50
51 # стандартная цель, предусматривающая сборку всего проекта
52 all: elf.elf bin.bin hex.hex
53
```

```
54 elf.elf: $(CPU).ld generic.ld startup.o init.o main.o
55 ____$(LD) -T ../hello/$(CPU).ld -T ../hello/generic.ld -o $@ *.o && \
56 ____$(OBJDUMP) -xd $@ > $@.objdump && \
57 ____$(SIZE) $@
58
59 hex.hex: elf.elf
60 ____$(OBJCOPY) -O ihex $< $@
61 bin.bin: elf.elf
62 ____$(OBJCOPY) -O binary $< $@
63
64 # стандартная цель, удаление всех временных и конечных бинарных файлов
65 clean:
66 ____rm -f $(TMPFILES)
67
68 # макро-правило: как компилировать сишные файлы в объектный код
69 # вместо % в других правилах могут подставляться любые символы, см. цель all
70 # тэг $@ заменяется на цель правила, т.е. %.o
71 # тэг $< заменяется на первый источник, т.е. %.c
72 %.o: %.c Makefile
73 ____$(CC) $(CFLAGS) -c -o $@ $<
74 ____$(OBJDUMP) -dx $@ > $@.objdump
75
76 # макро-правило: как компилировать ассемблерные файлы
77 %.o: %.S Makefile
78 ____$(AS) $(ASFLAGS) -o $@ $< && $(OBJDUMP) -dx $@ > $@.objdump
79
80 # запуск симулятора
81 emu: elf.elf
```

```
82 ____$(EMU) $<
83
84 # запуск отладчика
85 deb: elf.elf
86 ____$(GDB) $<
```

Обратите внимание, особенно если не используете  ECLIPSE — текстовый редактор должен быть настроен так, чтобы символ табуляции <TAB> не заменялся на пробелы, и отображался как 4 <пробел>а. В листинге табуляции специально выделены, т.к. *имеют синтаксическое значение*.

Этот пример **Makefile** достаточно универсален и самодостаточен для большинства проектов в этой книге. Кажущийся большой объем получился за счет использования комментариев и переменных. И те, и другие служат для документирования проекта, и повышают читаемость кода. В принципе никто не мешает<sup>6</sup> написать несколько строк в **.bat**нике с явным указанием опций компиляторам, или вообще откомпилировать все исходники сразу одним вызовом **gcc** с кучей опций и списком исходных файлов. Но если вам потребуется что-то изменить, куда проще и быстрее сделать это в аккуратно оформленном самодокументированном **Makefile**.

Компилятор преобразует программу на языке программирования высокого уровня<sup>7</sup> в объектный код (смесь кусочков машинного кода со служебной информацией) или в текст на языке ассемблера.

Кросс-компилятор (**gcc**) отличается от обычного компилятора тем, что генерирует код не для компьютера на котором он выполняется (хост-система, **\$HOST**), а для компьютера другой архитектуры — целевой системы, **\$TARGET**.

Ассемблер (**as**) преобразует человекочитаемый машинный код программы в объектный код.

---

<sup>6</sup>особенно для микроскопических объемов исходных текстов программ для контроллеров — в самом худшем случае какие-то жалкие сотни Кб

<sup>7</sup>для микроконтроллерных встраиваемых систем используются Си и C<sup>++</sup>, на более тяжелых процессорах типа Cortex-Ax свободно применяются Java, Fortran, Python, и еще стоицот языков, созданных за последние 50 лет истории IT

Линкер (**ld**) объединяет несколько файлов объектного кода в один, и корректирует машинный код с учетом его конечного размещения в памяти целевой системы (адреса переменных, адреса переходов, размещение сегментов кода и данных в физической памяти целевой системы).

Дампер (**objdump**) позволяет получить информацию о содержимом объектных файлов, в частности значения различных служебных полей, и дизассемблированный машинный код.

Копир (**objcopy**) преобразует сегменты кода/данных из файла, полученного линкером, в формат, необходимый для ПО программатора: бинарные файлы, Intel HEX, ELF,.. загружаемые в масочное ПЗУ, FlashPROM (и EEPROM данных на МК ATmega).

Так как часто разработчики встраиваемых систем работают с разными аппаратными платформами, для команд тулчайна принято использовать префиксы типа **arm-none-eabi-**, чтобы явно отличать, какой именно (кросс-)компилятор вызывается.

Главная синтаксическая конструкция **Makefile** — блок правила, задающий зависимость между файлами и набор команд, которые нужно выполнить, если **дата модификации файла-цели старше, чем дата модификации одного из файлов-источников**. То есть если вы измените какой-то из файлов проекта, начнут срабатывать правила, которые обновляют зависимые от него файлы.

Синтаксис:

```
<файл-цель>: [<файл-источник1> ...]  
[<tab><команда1>]  
[<tab><команда2>]  
[...]
```

Количество файлов-источников и команд может быть любое, в том числе и нулевое. Каждая команда правила отбивается слева одной табуляцией (один символ с кодом 0x09, **не пачка пробелов**). Будьте аккуратны, редактируя **Makefile** во всяких блокнотах, вордпадах и прочей ереси, любящей “оптимизировать” пробелы: истинный ТАВ и 4 пробела на экране, как завещал Великий Столлман.

Использование переменных особых комментариев не требует, обычная подстановка. Есть переменная `$@`, имеющая значение текущего файла-цели. Есть похожая переменная `$<` — имя первого файла-источника.

Если кто вдруг не знает — символ `>` в командной строке применяется для перенаправления текстового вывода любой команды в файл. Если нужно в одной строке выполнить последовательно несколько команд, используются маркеры сцепления `;` `&&` и `|`. Описание их применения см. любую книжку по UNIX детсадовского уровня. В **Makefile** для простого последовательного выполнения команд<sup>8</sup> рекомендуется использовать цепку `&&`<sup>9</sup>.

Команды выполняются с синтаксисом: `<[путь]команда[.exe]> [параметры через пробел]`.

Команда — имя выполняемого файла, может указываться с полным путем (диск, цепочка каталогов) или без. Если путь не указан, поиск выполняемого файла проводится в списке каталогов, заданном в системной переменной `$PATH`. Под DOS и Windows исполняемые файлы имеют суффикс `.exe`, `.bat` и `.com`, который в командной строке обычно не указывается. Под UNIX флаг выполнимости можно поставить вообще на любой файл.

В параметрах указываются имена файлов и опции: текстовые одно- и многобуквенные имена, начинающиеся с одинарного или двойного минуса. Параметры разделяются одним или несколькими пробелами. Порядок и значение параметров зависит от команды. Параметры для команд GNU toolchain и ПО программаторов подробно описаны далее.

---

<sup>8</sup>без передачи данных через потоки ввода/вывода

<sup>9</sup>следующие команды выполняются только если предыдущая завершилась без ошибок — если компиляция завершится ошибкой, незачем вызывать программатор

# ЛР13: Hello World

Для начала нужно рассмотреть набор файлов минимального проекта:

- **README.txt**

Краткая информация о проекте — название, авторы, обязательно ссылки на Git-репозиторий, сайт, форум, и т.п.

- **Makefile**

Файл с описанием зависимостей между файлами, настройками проекта (в переменных) и правилами вызова компиляторов.

- **startup.S**

Стартовый код процессора, включает инициализацию системы тактирования, мапинга памяти, контроллера прерываний и минимальную инициализацию периферии. Пишется на ассемблере, т.к. на Си получается слишком сложно, синтаксически запутанно, или очень специфично для компилятора.

- **startup.c**

Сишный вариант стартового кода, использование синтаксических ухищрений GNU CC позволяет обойтись без ассемблера даже для описания таблицы векторов, но для начинающего сложновато разобраться.

- **init.c**

Сишный код инициализации железа (синтаксически легче описать блоки кода, зависящие от целевого процессора).

- **main.c**

Основной код, решающий поставленную задачу.

- **\$CPU.ld**

Скрипт линкера, настраивающий генерацию выходного бинарного файла в зависимости от целевого процессора — прежде всего организация памяти, и размещение сегментов кода/данных по фактическим адресам памяти. Поэтому здесь имя файла задано через переменную, описанную в **Makefile**.

- **generic.ld**

Скрипт линкера, содержащий общую часть для всех микроконтроллеров

Создаем эти файлы аналогично **Makefile** в ЛР12:

 **Project Explorer** > **hello** >  > **New** > **File** > File name: **НужныйФайл.xxx**

README.txt

```
1 Азбука ARМатурщика: лабораторная работа HelloWorld
2 (copy/pasta) Dmitry Ponyatov <dponyatov@gmail.com>
3 https://github.com/ponyatov/CortexMx
```

startup.S

```
1 // универсальный стартовый код для любых Cortex-Mx микроконтроллеров
2 // этот стартовый код должен быть слинкован на начало ПЗУ,
3 // которое не обязательно начинается с нулевого адреса
4
5 .thumb // Cortex-M умеет только Thumb режим
6
7 .section ".vectors"
8 .func _vectors
9 .global _vectors
```

```
10 _vectors:    // таблица векторов прерываний/исключений Cortex-M
11 // используется команда относительного перехода,
12 // т.к. она корректно работает при стартовом ремапинге памяти
13     .word _stack_top    // Initial SP значение указателя стека после сброса
14     .word _reset        // Reset // Сброс
15     .word _dummy        // NMI
16     .word _dummy        // Hard fault
17     .word _dummy        // Memory management fault
18     .word _dummy        // Bus fault
19     .word _dummy        // Usage fault
20     .word _dummy        // r
21     .word _dummy        // r
22     .word _dummy        // r
23     .word _dummy        // r
24     .word _dummy        // SVCall
25     .word _dummy        // Reserved for Debug
26     .word _dummy        // r
27     .word _dummy        // PendSV
28     .word _dummy        // SysTick
29     .word _dummy        // IRQ0
30     .word _dummy        // IRQ1
31     .word _dummy        // IRQ2
32     .word _dummy        // IRQ3
33     .word _dummy        // IRQ4
34     .word _dummy        // IRQ5
35     .word _dummy        // IRQ6
36     .word _dummy        // IRQ7
37 .endfunc
```



```
38
39 // строка копирайта, конец дополняется до границы машинного слова
40 .section ".copyright"
41 .func _C_startup
42 .global _C_startup
43 _C_startup:
44 .string "startup.asm (c) Azbuka ARMaturschika"
45 .align 4
46 .endfunc
47
48 .text // сегмент кода
49
50 // сюда передается управление при сбросе
51 .thumb_func
52 _reset:
53 // LDR SP,=_stack_top
54 B .
55
56 // обработчик-заглушка
57 .thumb_func
58 _dummy:
59 B .
60
61 // .data
62 // .word 0x12345678,1234
63
64 // .bss
65 // .comm buf,0x10,10
```

```
66
67 .end
```

# init.c

```
1
2 // пример как преобразовать макроопределение в строку
3 #define QUOTEQUOTE(Y) #Y
4 #define QUOTE(Y) QUOTEQUOTE(Y)
5
6 // копи направо
7 __attribute__((section(".copyright")))
8 char _C_init[] = QUOTE(CPU) "_init_(c)_Azbuka_ARMaturschika";
9
10 // пример условной компиляции по значению макроса, заданного
11 // в командной строке запуска компилятора -DCPU=$(CPU)
12 #if CPU==LM3S81122
13     #include "LM3S811.h"
14 #else
15     // пример как аварийно остановить компиляцию
16     #pragma message QUOTE(CPU)
17     #error CPU not supported
18 #endif
19
20 // #pragma message QUOTE(CPU)
21 // #error not supported
22
23 // точка входа
24 init () {}
```

## main.c

```
1 main () {}
```

## LM3S811.ld

```
1 /* скрипт линкера для Luminary Micro lm3s811 (эмулятор в qemu-arm) */
2
3 /* конфигурация памяти целевой системы, сильно зависит от микроконтроллера */
4
5 MEMORY
6 {
7     FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 64K /* 0x00010000 */
8     SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 8K /* 0x00002000 */
9 }
10 _stack_size = 1K; /* размер стека */
11
12 /* INCLUDE generic.ld */
```

## generic.ld

```
1 /* часть скриптов линкера, общая для всех микроконтроллеров */
2
3 /* описание упаковки секций объектных файлов
4    в целевой файл и размещение в памяти */
5
6 SECTIONS
```

```

7 {
8     /* секция кода, обеспечиваем нужный порядок сегментов */
9     .text : {
10         _text = .;
11         startup.o(.vectors) /* в начало кладем таблицу векторов */
12         *(.copyright)      /* потом копирайты (попадут в начало прошивки) */
13         *(.text)           /* а потом все остальное */
14     } >FLASH
15
16     /* стек по рекомендации MISRA располагаем НИЖЕ данных т.к. растет вниз
17        (предотвращение затирания данных при переполнении стека) */
18     .stack : {
19         . = ALIGN(8);
20         _stack = .;
21         . = . + _stack_size; /* резервируем память под стек */
22         _stack_top = .;      /* создаем указатель на вершину стека */
23     } >SRAM
24
25     /* секция инициализированных данных (константы) */
26     .data : {
27         _data = .;
28         *(.data)
29     } >SRAM AT>FLASH
30
31     /* секция пустых данных и кучи
32        (переменные и массивы без заданных значений, динамическая память) */
33     .bss : {
34         _bss = .;

```

```
35         * ( . b s s )
36     } >SRAM
37 }
```

# Литература

- [1] <https://github.com/ponyatov/CortexMx> Азбука халтурщика-ARMатурщика
- [2] Getting started with CMSIS [http://www.doulos.com/knowhow/arm/CMSIS/CMSIS\\_Doulos\\_Tutorial.pdf](http://www.doulos.com/knowhow/arm/CMSIS/CMSIS_Doulos_Tutorial.pdf)
- [3] Ю.С. Магда Программирование и отладка C/C++ приложений для микроконтроллеров ARM. — М.: ДМК Пресс, 2012. — 168 с.: ил.
- [4] © Quantum<sup>®</sup>*LeaPs*
- [5] [http://www.state-machine.com/arm/Building\\_bare-metal\\_ARM\\_with\\_GNU.pdf](http://www.state-machine.com/arm/Building_bare-metal_ARM_with_GNU.pdf) Quantum<sup>®</sup>*LeaPs*  
Building Bare-Metal ARM Systems with GNU
- [6] <http://milandr.ru/> ЗАО «ПМК Миландр»
- [7] <http://git-scm.com/book/ru> перевод: Scott Chacon **Pro Git**
- [8] <http://habrahabr.ru/post/114239/> хабра: Quantum<sup>®</sup>*LeaPs* QP и диаграммы состояний в UML
- [9] <http://www.state-machine.com/> Quantum<sup>®</sup>*LeaPs* State Machines & Tools

- [10] <http://makesystem.net/?p=988> Изучаем ARM. Собираем свою IDE для ARM
- [11] <http://makesystem.net/?p=2146> Изучаем ARM. Отладка ARM приложений в Eclipse IDE
- [12] Львовский С.М. Набор и вёрстка в пакете L<sup>A</sup>T<sub>E</sub>X