

Контроллеры ARMatura

© Dmitry Ponyatov <dponyatov@gmail.com>, SSAU ASCL

18 марта 2013 г.

Оглавление

Оглавление	1
Listings	5
I Введение	6
II Железо	8
1 STM32VLDISCOVERY /STM32F100RBT6/	9
2 STM32F0DISCOVERY /STM32F051R8T/	11
3 STM32F4DISCOVERY /SRM32F407VGT6/	13
4 PION /STM32F103C8/	15
5 ARMatura /STM32F407IGT/	16
6 робоплатформа ROBOtron	17
III Первые шаги	18
7 Установка ПО	19
8 Первый проект: blink	20
8.1 Структура файлов	33
8.2 blink.cpp	34
9 Hell Of World	36

IV Средства разработки	37
10 IDE	38
10.1 Keil	38
10.2 Настройки среды	44
10.3 Настройки проекта	45
10.4 Eclipse	58
10.5 Code::Blocks	59
10.6 gVim	59
10.7 IAR	59
11 Компиляторы	60
11.1 GCC	60
11.2 KeilCC	60
11.3 IAR	60
12 Адаптеры	61
12.1 STlink	61
12.2 JTAG	68
V Отладка	69
13 Отладка в Keil	70
14 STM32 SWD	71
15 GDB	72
15.1 STlink gdbserver	72
16 OpenOCD	73
17 JTAG	74
VI Основы языка C⁺⁺	75
18 Синтаксис	76
19 Типы данных	77
20 Стандартная библиотека libc	78
VI Wiring Framework	79
VII CMSIS STM32	81
21 Установка	83
22 Startup код	84
22.1 startup_stm32f10x_ld_vl.s	84

22.2 startup_stm32f40xx.s	84
22.3 system_stm32f10x.c	84
22.4 system_stm32f4xx.c	84
23 CMSIS	85
23.1 stm32f10x.h	85
23.2 core_cm3.h	85
23.3 core_cm4.h	85
23.4 core_cmInstr.h	85
23.5 core_cmFunc.h	85
24 BSP	86
24.1 STM32VLDISCOVERY1	86
24.2 STM32F4DISCOVERY3	86
25 Cortex-M4 расширение SIMD/DSP	87
25.1 core_cm4_simd.h	87
IX Ядро Cortex-Mx	88
26 Режимы ARM и Thumb	89
27 DMA	90
28 DSP /Cortex-M3/	91
29 FPU /Cortex-M4F/	92
X Семейство STM32F10x	93
30 Об этом документе	95
30.1 Typographical conventions	95
30.2 List of abbreviations for registers	95
30.3 About the STM32 Cortex-M3 processor and core peripherals	95
31 Процессор Cortex-M3	97
31.1 Programmers model	97
31.2 Memory model	97
31.3 Exception model	97
31.4 Fault handling	97
31.5 Power management	97
32 Набор инструкций Cortex-M3	99
32.1 Instruction set summary	99
32.2 Intrinsic functions	99
32.3 About the instruction descriptions	99
32.4 Memory access instructions	99
32.5 General data processing instructions	99
32.6 Multiply and divide instructions	99
32.7 Saturating instructions	99

32.8 Bitfield instructions	99
32.9 Miscellaneous instructions	99
33 Периферия ядра	101
33.1 About the STM32 core peripherals	101
33.2 Memory protection unit (MPU)	101
33.3 Nested vectored interrupt controller (NVIC)	101
33.4 System control block (SCB)	101
33.5 SysTick timer (STK)	101
34 История изменений	102
XII Интерфейсы	103
35 USB	104
36 UART	105
37 SPI	106
38 I2C	107
39 CAN	108
XIII Операционные системы ОСРВ	109
40 Keil RTX	110
41 FreeRTOS	111
42 eCos	112
43 Linux	113
XIV Протоколы TCP/IP	114
44 Ethernet	115
45 PPP	116
XV Типовые применения	117
46 GPS	118
46.1 Протокол NMEA 0183	118
46.2 \$GPRMC — рекомендуемый минимум навигационных данных	119
46.3 Системы координат (датум)	120
46.4 Методы преобразования систем координат	121
46.5 Геоид и эллипсоиды	122
46.6 Tistar15	123

46.7 WISMO228	123
47 GSM	124
47.1 WISMO228	124
XWCS: управление версиями	125
48 Git	129
48.1 Установка	129
48.2 Создание репозитория из проекта	129
48.3 Работа с удалёнными репозиториями	129
48.4 Репа на флешке	130
48.5 Работа с проектом github://ARMatura	131
49 SubVersion	138
50 CVS	139
51 Mercurial	140
52 Bazaar	141
53 шина Dallas 1Wire	142
53.1 RTC	142
53.2 Датчики температуры DS18x20	142
XЧИстраиваемый Linux	143
XЧQA	144
XЧПриложения	146
54 Сводная таблица процессоров	147

Listings

8.1 blink.c	35
versioncontrol/gitmkrepo.txt	129
versioncontrol/gitremote.txt	130
versioncontrol/gitflash.txt	130

Часть I

Введение

Эта книга – набор методичек по разработке ПО для встраиваемых систем, написанных для Института космического приборостроения СГАУ.

Для применения в реальных проектах научной аппаратуры была разработана линейка унифицированных модулей:

1. ARMatura — модуль на мощном микропроцессоре STM32F417IGT: 1M Flash, 192K SRAM, TQFP176, DSP, FPU,.. [54](#)

предназначен для использования в качестве центрального процессора цифровой системы: обработка данных, сложные алгоритмы управления, ЦОС, вычисления, реализация протоколов передачи данных по интерфейсам USB, Ethernet, RS232/UART, SPI, I2C, CAN,..

2. PION [4](#) — модуль на самом простом и дешевом STM32F100C4T6B: 128K Flash, 8K SRAM, UART, SPI [54](#)

периферийный модуль для стыковки с аналоговыми датчиками и исполнительными устройствами, предварительная ЦОС обработка, передача данных на ARMatura-модули для дальнейшей обработки данных.

также модуль применим в качестве самостоятельного простого интерфейса при замене на чип STM32F103 с портом USB или установки внешних интерфейсных микросхем FT232RL (USB Serial), CP1202, MC1551 (CAN).

3. BACKPLANE — коммутационная плата межмодульного интерфейса
4. POWER — модуль импульсного источника питания
5. STEPPER — модуль управления двухфазным шаговым двигателем
6. WISMO — несущая плата для GPS/GSM модуля WISMO 228
7. QVGA — несущая плата для TFT touch-панели

В качестве базового микроконтроллера были выбраны чипы семейства STM32Fxxx с ядрами Cortex-M3, Cortex-M4F (ARM) как самые дешевые, и имеющие хорошую поддержку в виде отладочных плат линейки Discovery.

В общем, линейка модулей ARMatura может рассматриваться в качестве замены устаревшей линейки периферийных контроллеров Arduino на базе МК AVR8.

Проект размещен в репозитории <https://github.com/ponyatov/ARMatura.git> и предоставляется на условиях OpenHardware licence (за исключением прошивок и схем по тематике ИКП СГАУ).

Контакты разработчиков:

- ИКП СГАУ <semkin@ssau.ru>
- Дмитрий Понятов <dponyatov@gmail.com>

Часть II

Железо

Глава 1

STM32VLDISCOVERY /STM32F100RBT6/



Первая модель дешевых демоплат STM32, которые заметно дернули рынок дешевых ARMов в среде embedded разработчиков, а особенно хоббитов, давно ждавших дешевый аналог плат Arduino, по пока до сих пор не дождавшихся.

- Микроконтроллер STM32F051R8T~~54~~, 128 KB Flash, 8 KB RAM in 64-pin LQFP
- Встроенный STlink~~12.1~~ с возможностью использования в режиме внешнего программатора (только с SWD коннектором)
- Разработана для питания как от USB, так и от внешнего источника 3.3 или 5 вольт

- Может обеспечить питание 3 и 5 вольт для внешних устройств
- Два пользовательских светодиода (зеленый и синий)
- Пользовательская кнопка USER
- Кнопка сброса RESET
- Контактные гребенки для всех выводов QFP64 для быстрого подключения и монтажа прототипа

Цена в розницу: 750 р.

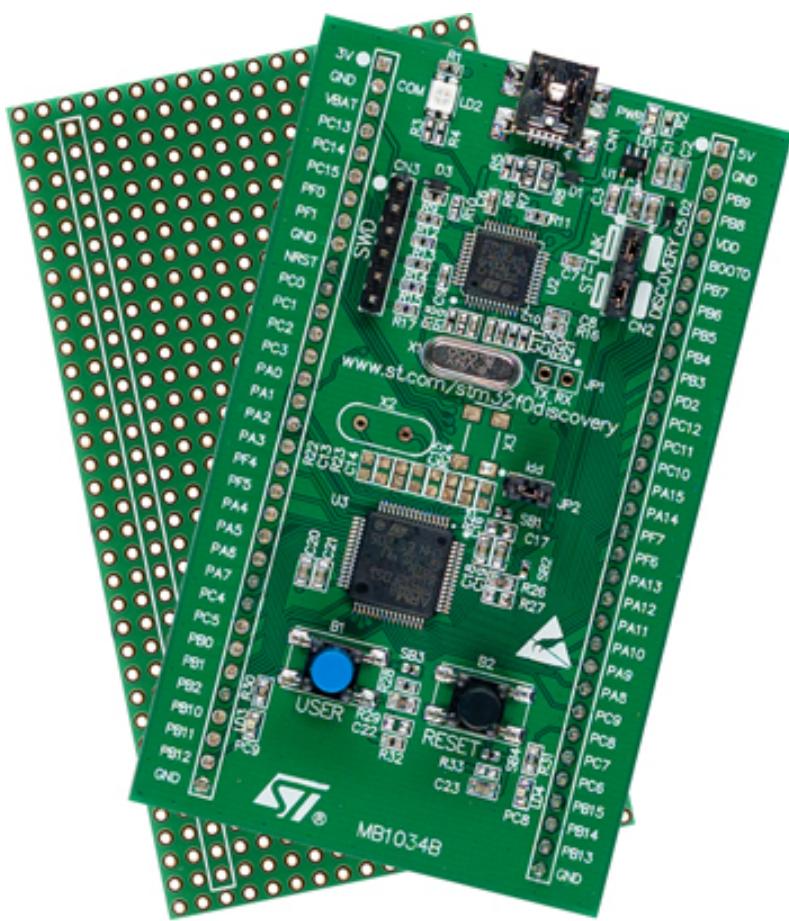
<http://www.voltmaster.ru/cgi-bin/qwery.pl?id=127000573571&group=7000046>

<http://we.easyelectronics.ru/STM32/stm32vldiscovery-the-grabli-ili-dlya-samyh-nach.html>

- Контроллер STM32F100RBT6 (128KB Flash, 8KB RAM, up to 24 MHz)
- Кварц 8MHz, установлен на цангах, так что можно ставить свой
- Кварц 32.768KHz X3
- Встроенный программатор-отладчик STlink^{12.1}, только с выходом SWD, можно его использовать и для прошивки своих девайсов. Что характерно, отладчик выполнен на более серьёзном STM32F103CBT6⁵⁴.
- Две кнопки (правда одна из них — RESET).
- Два светодиода PC8,PC9

Глава 2

STM32F0DISCOVERY /STM32F051R8T/



Плата отличается тем, что установлен самый толстый процессор линейки STM32F0. Если вам не удается впихнуть в него текущую прошивку, сразу можно сказать что придется уходить на процессор более толстой линейки.

- Микроконтроллер STM32F051R8T⁵⁴, 64K Flash, 8K SRAM, LQFP64
- Встроенный STlink^{12.1} с возможностью использования в режиме внешнего программатора (только с SWD коннектором)
- Разработана для питания как от USB, так и от внешнего источника 3.3 или 5 вольт

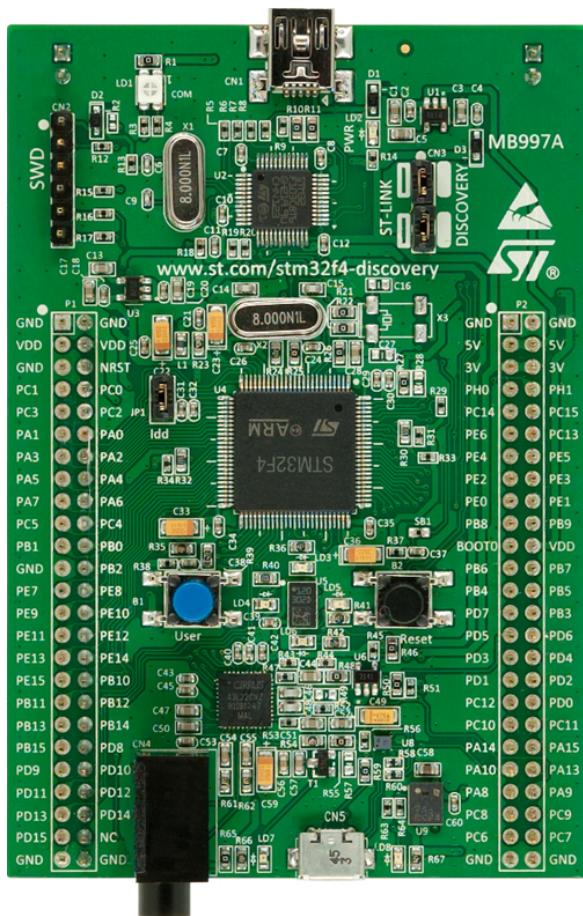
- Может обеспечить питание 3 и 5 вольт для внешних устройств
- Два пользовательских светодиода (зеленый и синий)
- Пользовательская кнопка USER
- Кнопка сброса RESET
- Контактные гребенки для всех выводов QFP64 для быстрого подключения и монтажа прототипа
- В комплект включена макетная плата под DIP

Цена в розницу: 650 р.

<http://www.voltmaster.ru/cgi-bin/qwery.pl?id=127001024065&group=7000046>
<http://www.st.com/web/en/catalog/tools/FM116/SC959/SS1532/PF253215>

Глава 3

STM32F4DISCOVERY /SRM32F407VGT6/



- микроконтроллер STM32F407VGT6 [54](#) на базе 32-битного ядра Cortex-M4F, 1 MB Flash, 192 KB RAM в корпусе LQFP100
- встроенный ST-LINK/V2 с возможностью использования в режиме внешнего программатора (только с SWD коннектором)
- Разработана для питания как от USB, так и от внешнего источника 5 вольт
- Может обеспечить питание 3 и 5 вольт для внешних устройств
- LIS302DL, ST MEMS датчик движения, 3-осевой цифровой гироскоп
- MP45DT02, ST MEMS аудиодатчик, всенаправленный цифровой микрофон

- CS43L22, аудиоЖАП со встроенным аудиоусилителем класса D
- Восемь LEDs: LD1 (red/green) for USB communication LD2 (red) for 3.3 V power on Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue) 2 USB OTG LEDs LD7 (green) VBus and LD8 (red) over-current
- Пользовательская кнопка USER
- Кнопка сброса RESET
- USB OTG FS with micro-AB connector
- Контактные гребенки для всех выводов LQFP100 ля быстрого подключения и мон-тажа прототипа

Цена в розницу: 1140 р.

<http://www.voltmaster.ru/cgi-bin/qwery.pl?id=127000854271&group=7000046>

Глава 4

PION /STM32F103C8/

Модуль PION предназначен для мелких задач управления, первичной обработки данных, стыковки с устройствами измерения и исполнительными устройствами, т.е. для тех задач, для которых ранее использовались микроконтроллеры Atmel AVR8.

процессор	STM32F103C8	??
тактовая	72Mhz	
Flash	64K	
SRAM	20K	
шина	AVTObus	
интерфейсы	UART	3
	USB	1
	CAN	1
	SPI	2
	АЦП	10x12b
	ЦАП	2x12b
буфер	Parallel Flash	64K

Выбор процессора определялся ценой, наличием CAN интерфейса (требование для шины AVTObus), USB (возможность использования в виде изолированного контроллера для ПК), объемом SRAM для хранения рабочих данных, и количеством ног ввода/вывода.

Глава 5

ARMatura /STM32F407IGT/

Большая полнофункциональная плата с самым мощным из доступных MCU STM32F407IGT⁵⁴, набором наплатной периферии, которая чаще всего нужна для автономного контроллера, и коннектором шины AVTObus для подключения модулей расширения.

- MCU Cortex-M4F: 1M Flash, 192K SRAM, LQFP144, 123 GPIO
- FPGA
- 2×USB
- 2×CAN
- SDIO
- AVTObus
- 4x16 LCD
- joystick
- 4×LED
- Audio in (jack 3.5) с поддержкой микрофона
- Audio out (jack 3.5) со стереоДАС и усилителем
- коннектор Dallas 1wire
- 6-осевой акселерометр
- поддержка ROBOtron⁶

Глава 6

робоплатформа ROBOtron

Колесная робоплатформа для покатушек за зайцами и прочими домашними животными.

Оборудована аккумулятором, ходовым электродвигателем, поворотной сервой, набором сенсоров, светотехникой, матюгальником и модулем беспроводной связи.

Конструктивно выполнена в виде механизированного AVTObus carrier, электронная часть в виде отдельных AVTObus-модулей с возможностью модификации и расширения.

Часть III

Первые шаги

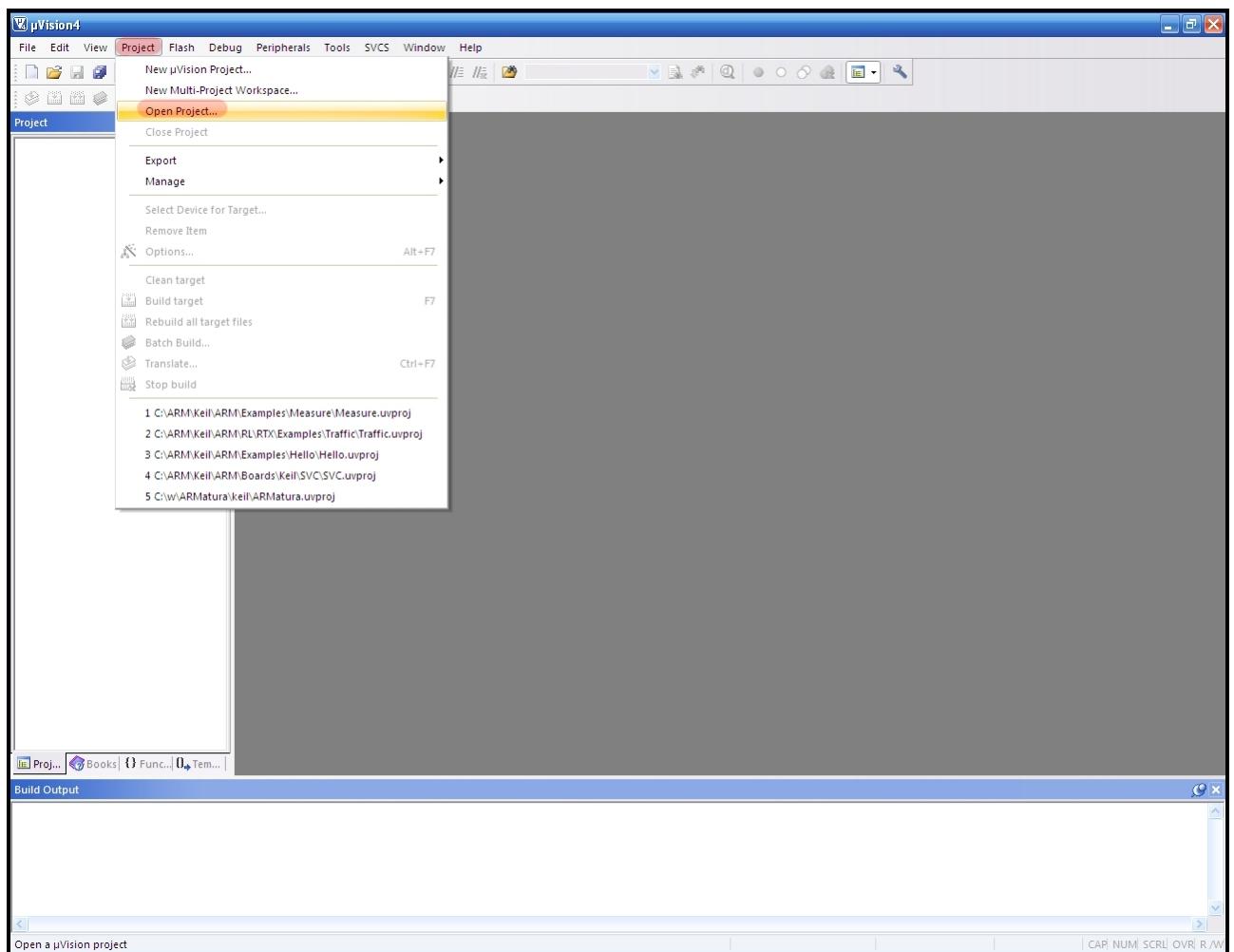
Глава 7

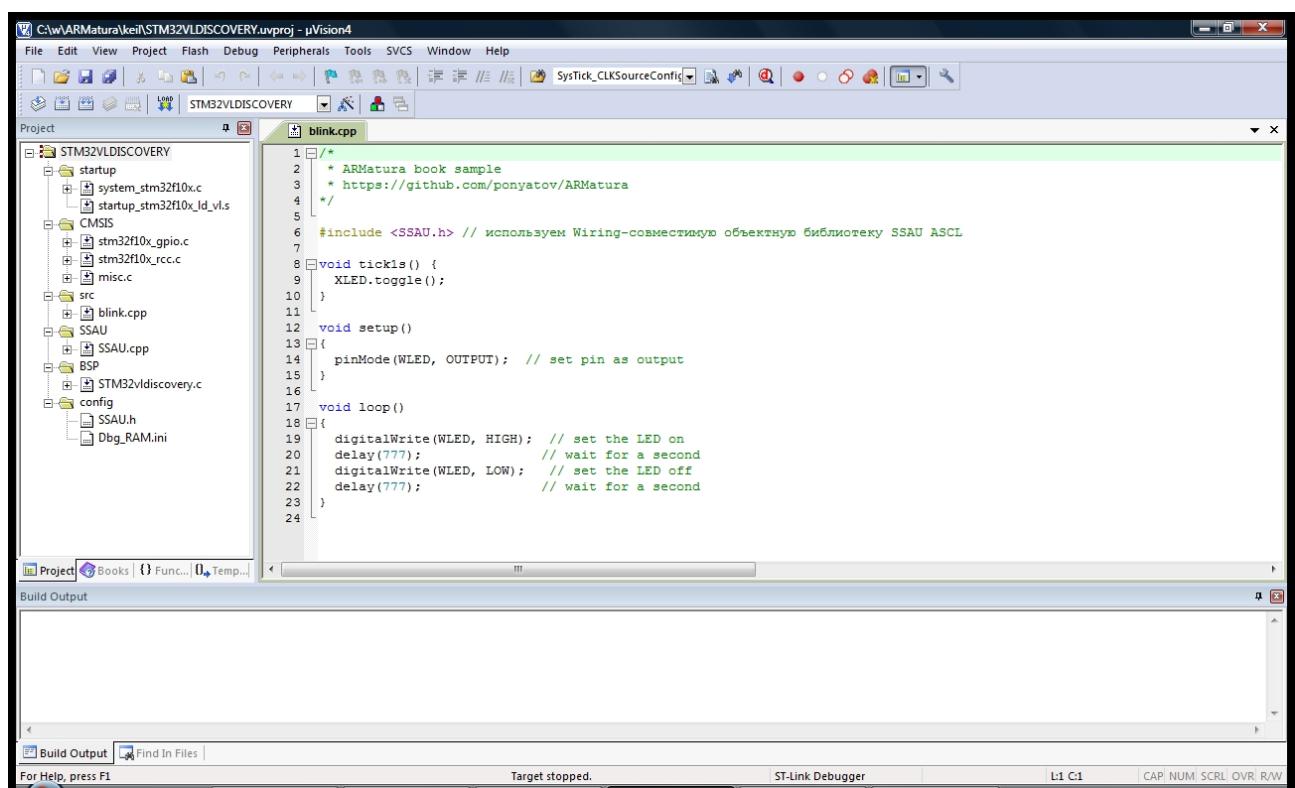
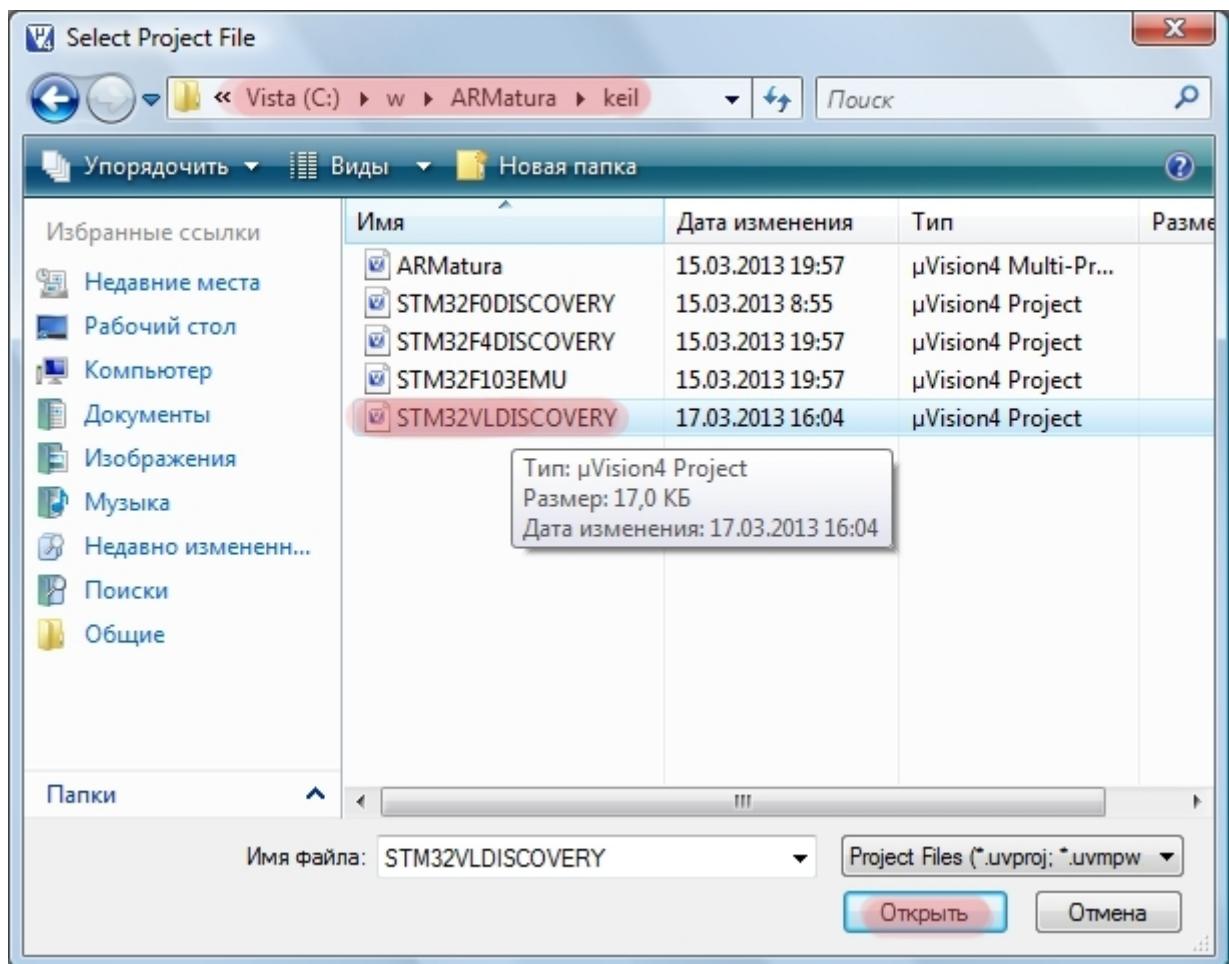
Установка ПО

1. установка пакета ПО STlink [12.1](#)
2. установка Keil MDK-ARM [10.1](#)

Глава 8

Первый проект: blink

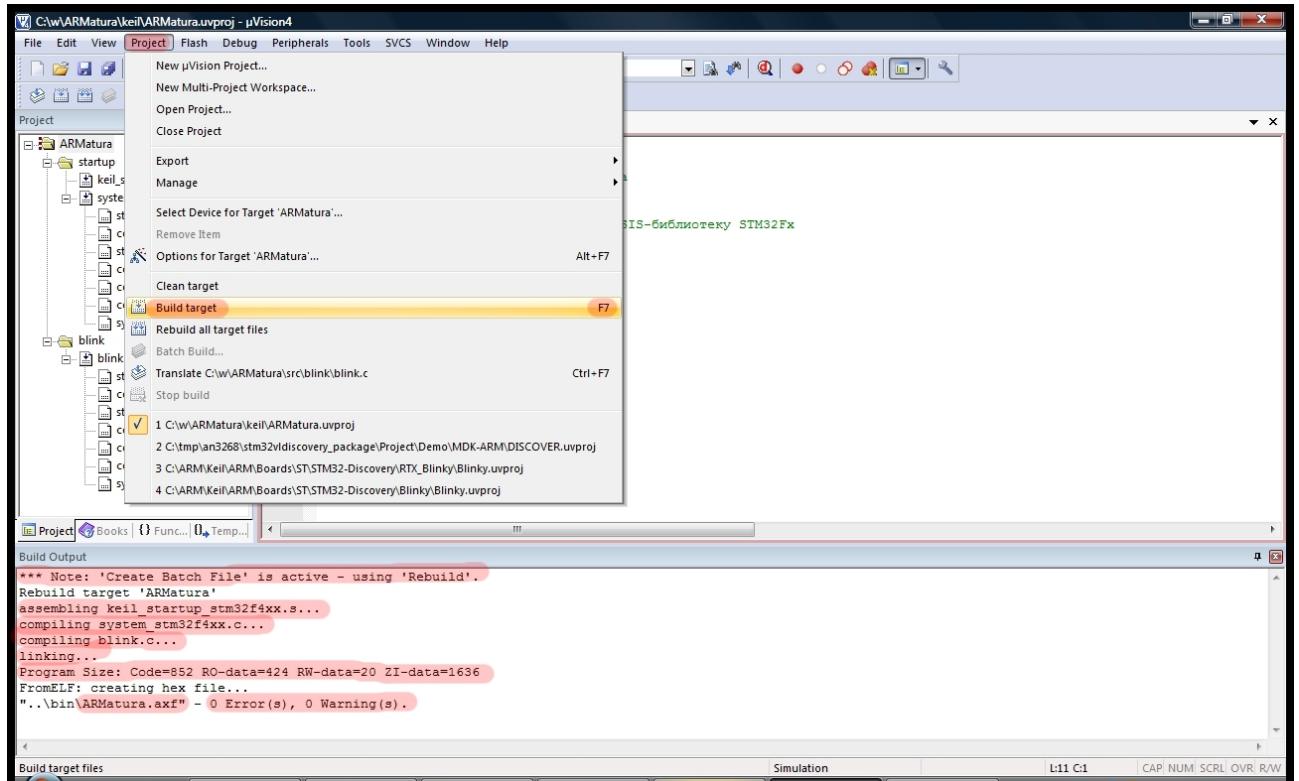




При необходимости можно изменить настройки проекта см. 10.3. Перенастройка может понадобиться если вы пытаетесь работать не с STM32VLDISCOVERY¹, а с другой платой

или самопальной железкой. Также перенастройка может понадобится если почему-то не срабатывает подключение к адаптеру STlink^{12.1}.

Собираем проект, нажав клавишу **F7**:



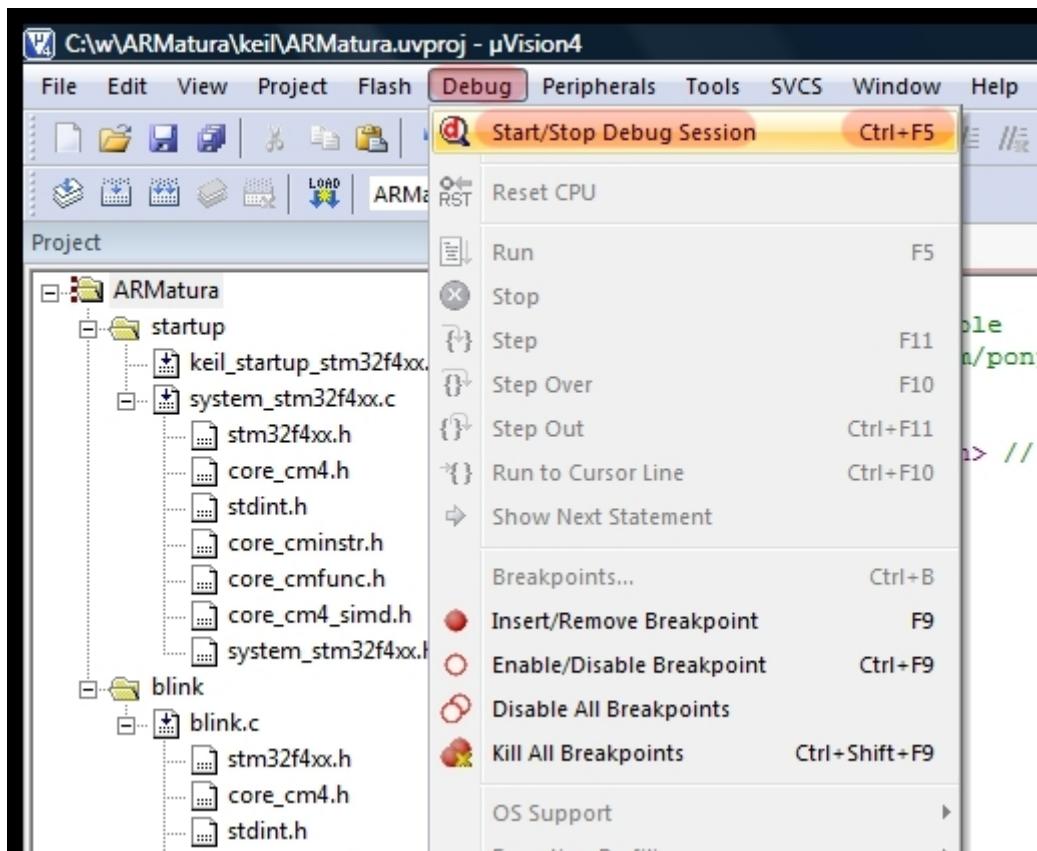
Проект успешно собрался, теперь можно проставить на `loop()` точку останова (breakpoint) переместив курсор на заголовок функции и нажав кнопку **F9**. При работе программы в отладочном режиме работа программы останавливается на точках останова, позволяя посмотреть состояние регистров процессора, переменных и т.п.

```

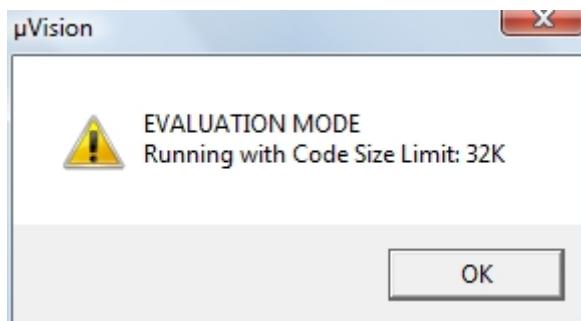
17 void loop()
18 {
19     digitalWrite(WLED, HIGH); // set the LED on
20     delay(777); // wait for a second
21     digitalWrite(WLED, LOW); // set the LED off
22     delay(777); // wait for a second
23 }
24

```

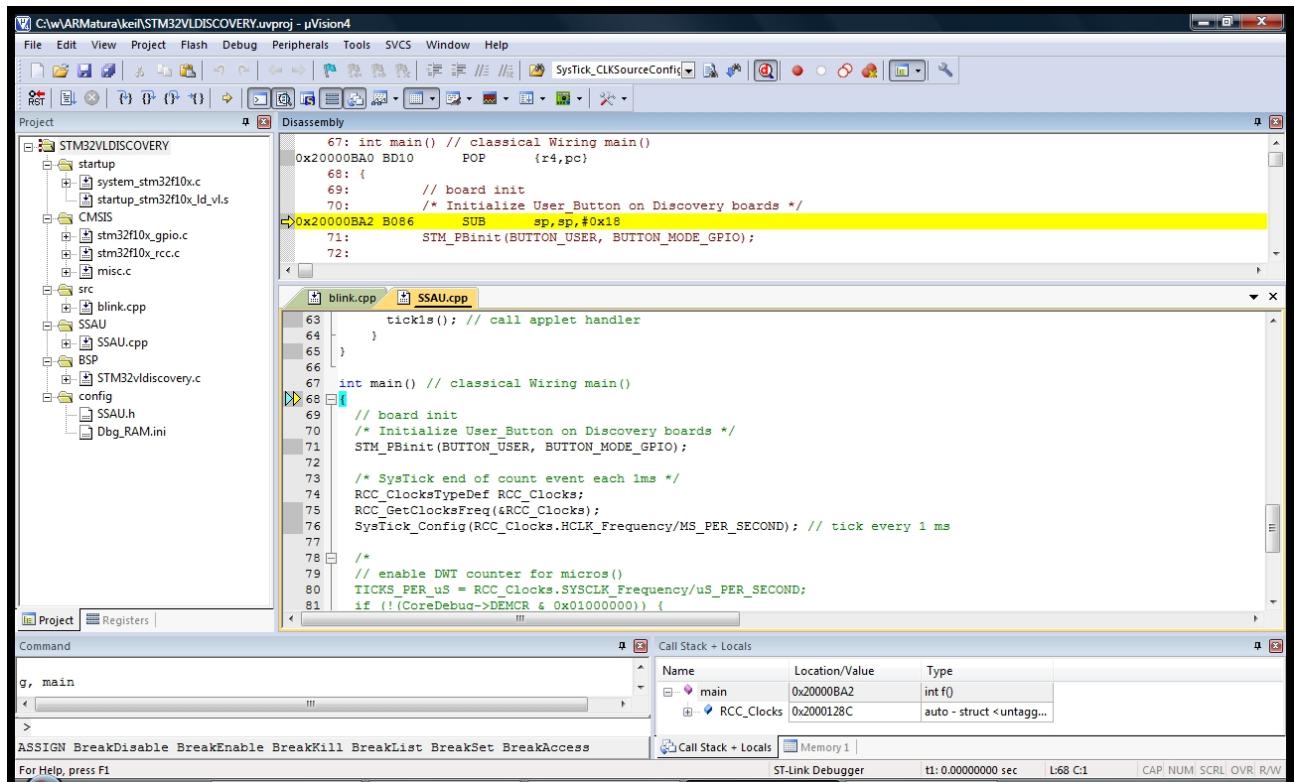
Подключаем отладочную плату, а затем запускаем отладку нажав **Ctrl + F5**.



Вывалится диалог с сообщением об использовании оценочной версии Keil10.1.



После дрыганья окнами откроется отладочный режим:



В окне редактора автоматически откроется код `SSAU.cpp`

В окне `Registers` показывается состояние регистров процессора. В верхней части расположено окно дизассемблированного кода, снизу слева командная консоль отладчика, снизу справа — просмотр стека и памяти.

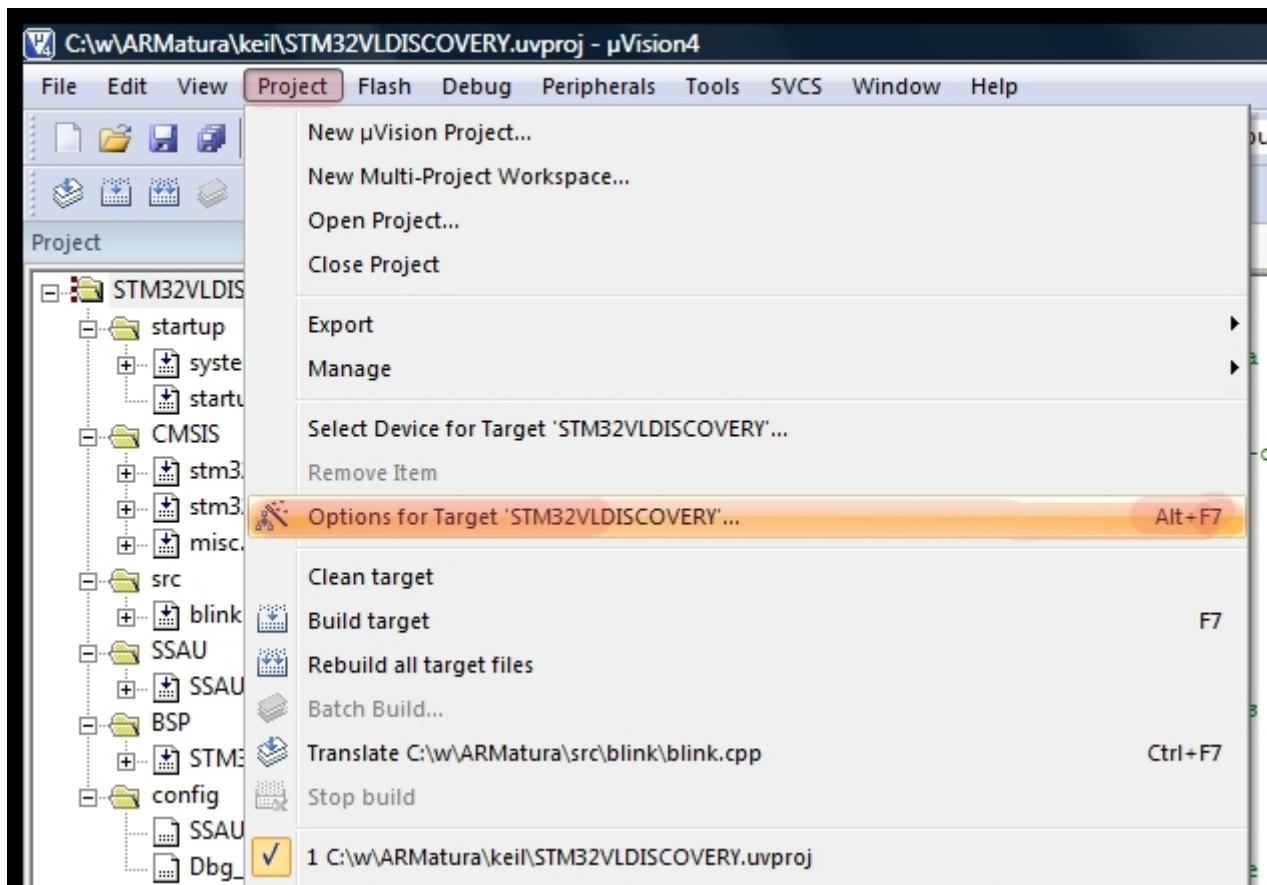
Подробно работа с отладчиком описана в разделе 13, при желании вы можете пройтись в пошаговом режиме (нажимая кнопку `F11`) по коду, и глядя в даташит на ваш микроконтроллер, попытаться разобраться что именно он делает.

Пока же разборки с работой в отладчике можно отложить, лучше вместо этого запустим программу на целевой плате, нажав `F5`. Картинка не изменится, но плата должна запуститься и мигать светодиодами: LD3 (зеленый) будет мигать с частотой $0.777ms \times 2 = 1.554$ Гц, а LD4 (синий) будет переключаться с частотой 2 Гц асинхронно функцией `tick1s()`, которую вызывает обработчик системного таймера.

Проект по умолчанию настроен на запуск программы в SRAM¹, не используя Flash — эта фича Cortex-ов позволяет экономить циклы перезаписи флеша, но требует использования отладчика Keil 10.1, и позволяет запускать только простые программы, т.к. ОЗУ у МК младших линеек очень мало, для платы STM32VLDiscovery всего 8К.

Если вам не хватит SRAM для программы, или захочется получить автономное устройство, нужно будет перенастроить проект под использование Flash:

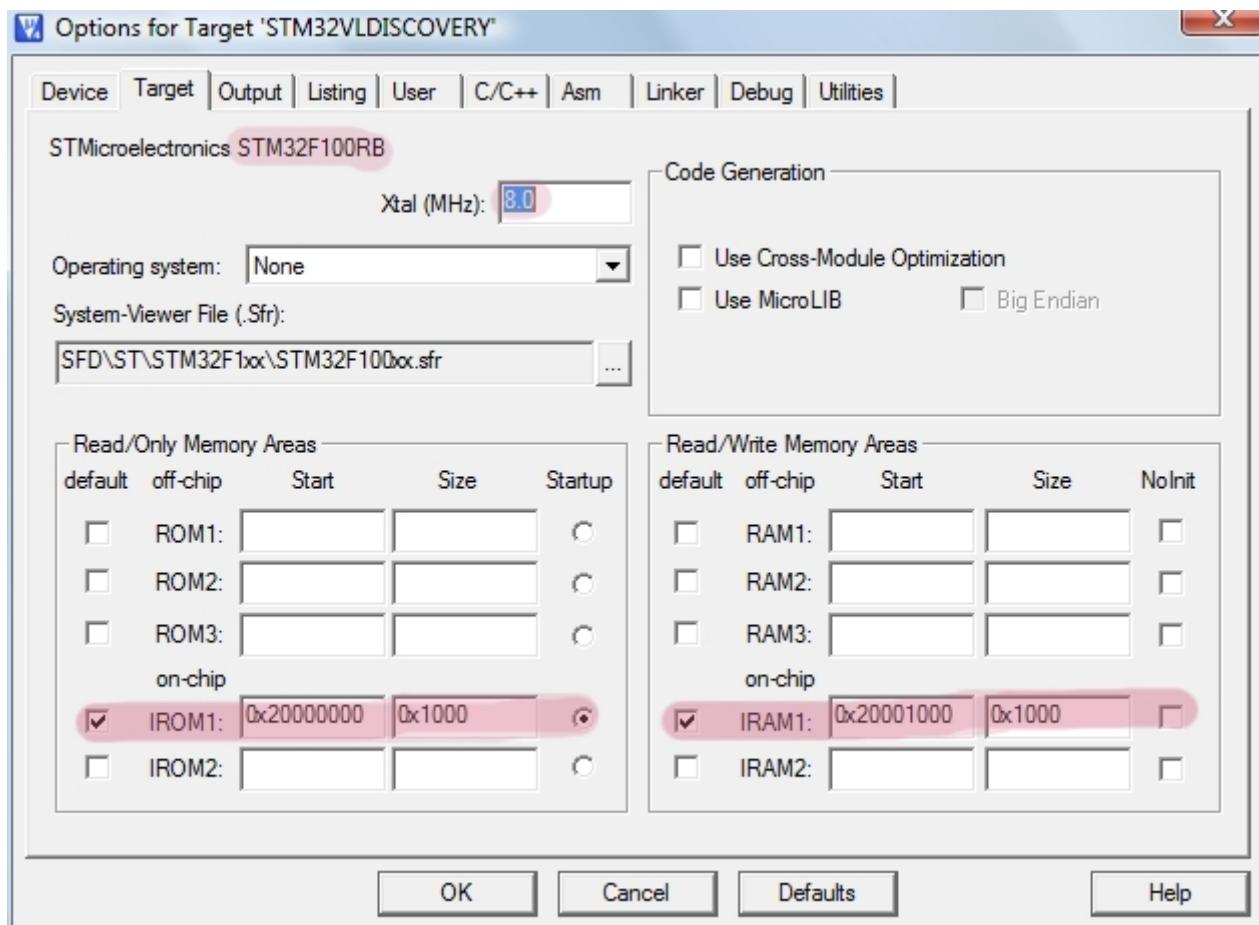
¹статическое ОЗУ



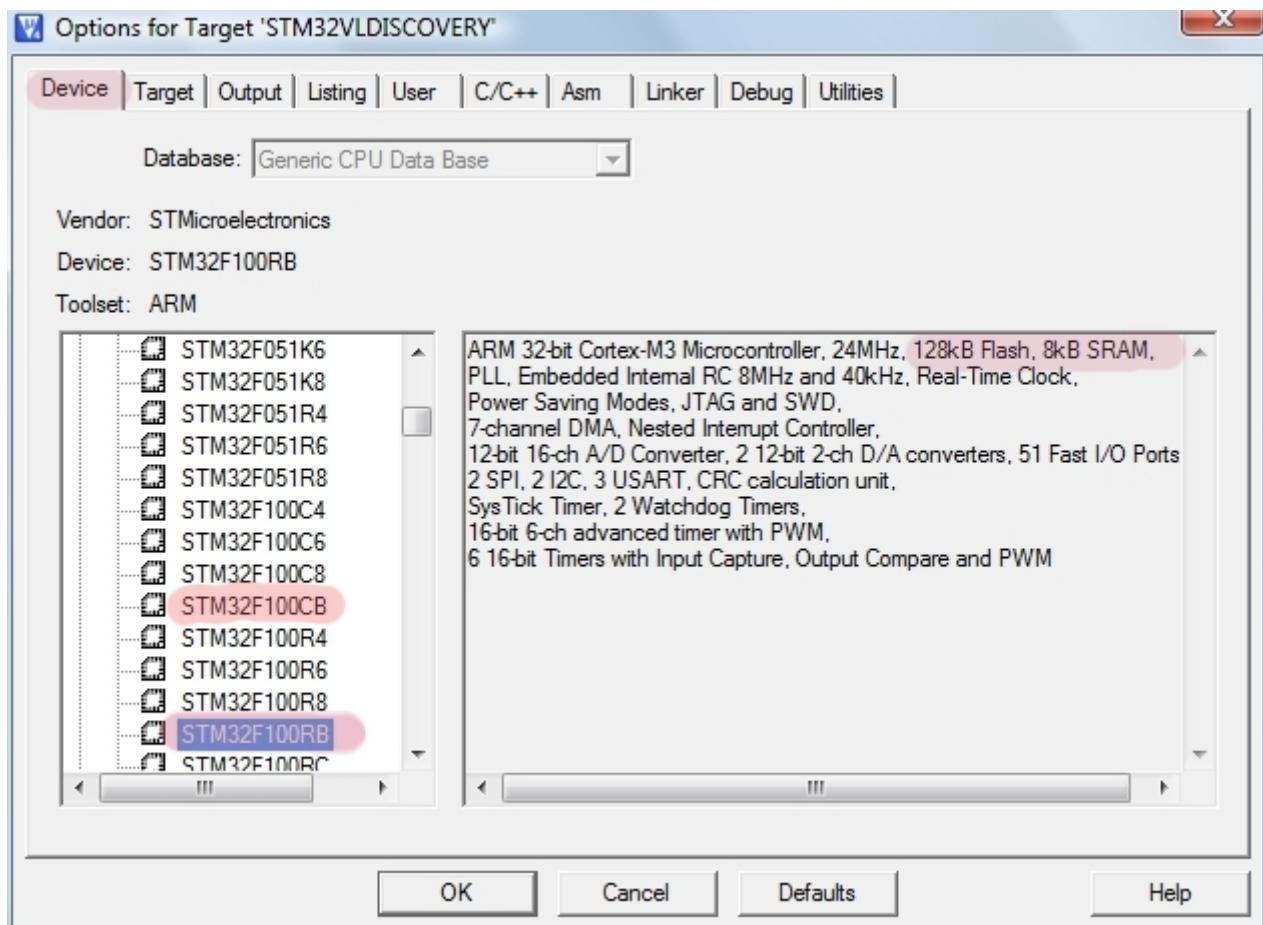
Выделен текущий процессор STM32F100RBТ6, использующий внешний квадр 8 МГц.

Обратите внимание на настройки областей IROM1 и IRAM1: обе области прописаны в область SRAM контроллера, но на начало SRAM привязан IROM1, а на вторую половину ОЗУ — IRAM1. В результате при компиляции линкер генерирует код для исполнения из SRAM, а отладчик при своем запуске проложит код, и выставит регистры, считав значения из таблицы векторов в начале стартового кода:

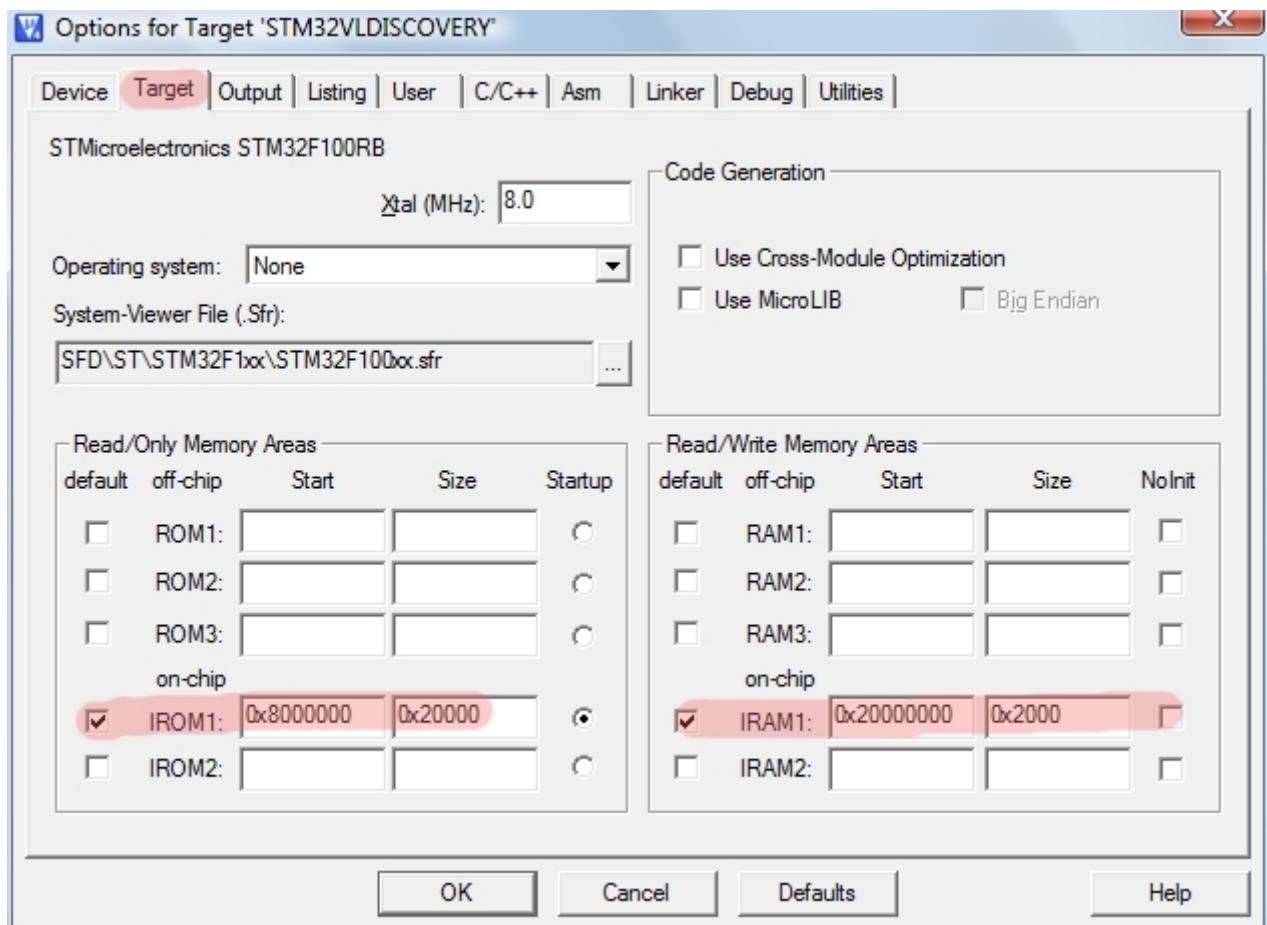
- PC указатель выполняемой инструкции на начало кода инициализации в SRAM
- SP указатель стека на значение из таблицы векторов
- VTOR перенастроит на начало таблицы векторов в ОЗУ (вместо 0го адреса по умолчанию)



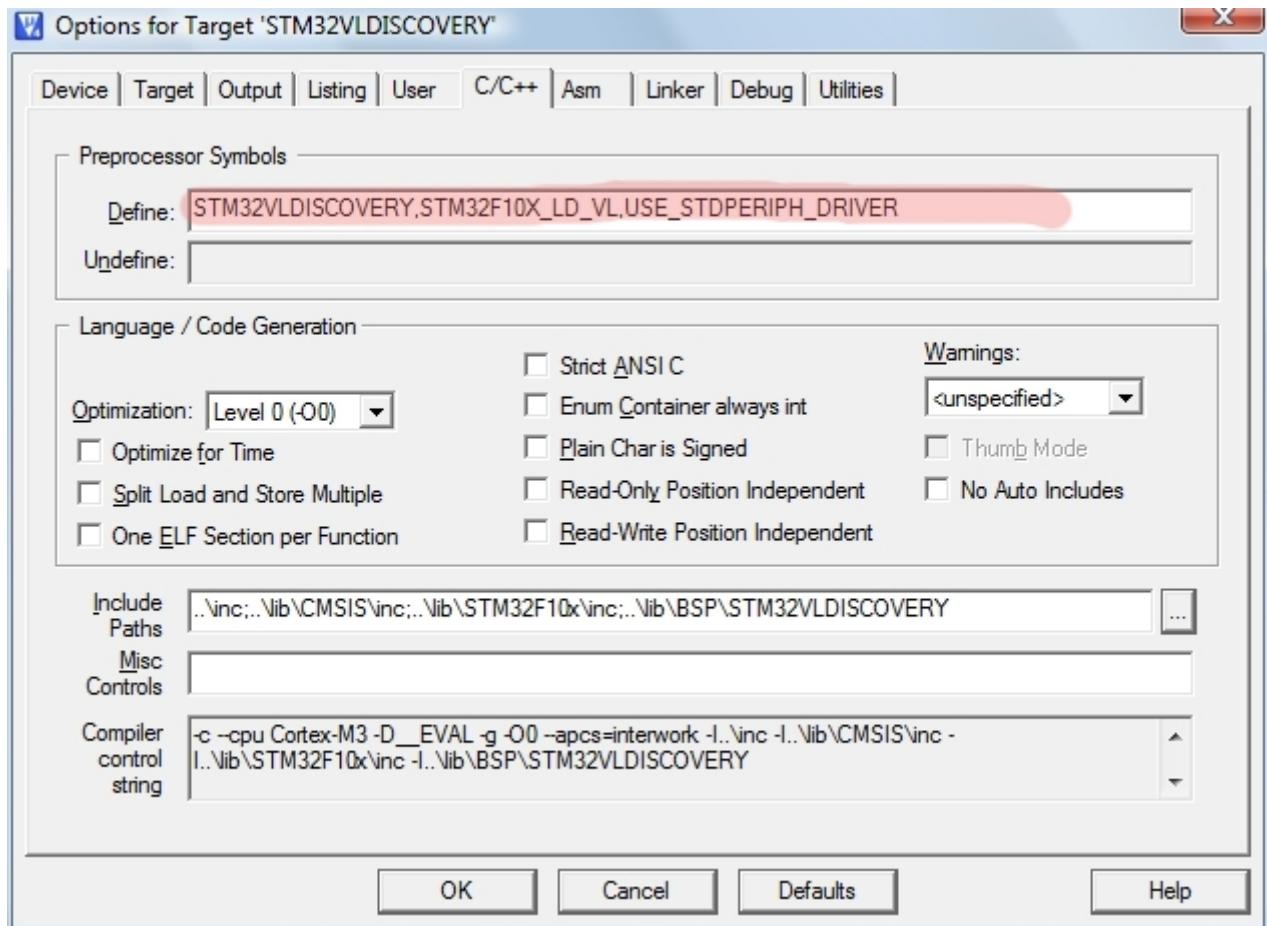
Чтобы не возиться с ручным изменением адресов, нужно перейти на вкладку **Device**, запомнить текущий МК, выбрать любой соседний контроллер, а затем еще раз выбрать нужный. Цветом выделены размеры памяти — 128K флеша и 8K ОЗУ.



Теперь если еще раз выбрать вкладку **Target**, в настройки областей памяти пропишутся нужные значения для загрузки во флеш, а объем доступного ОЗУ увеличится:

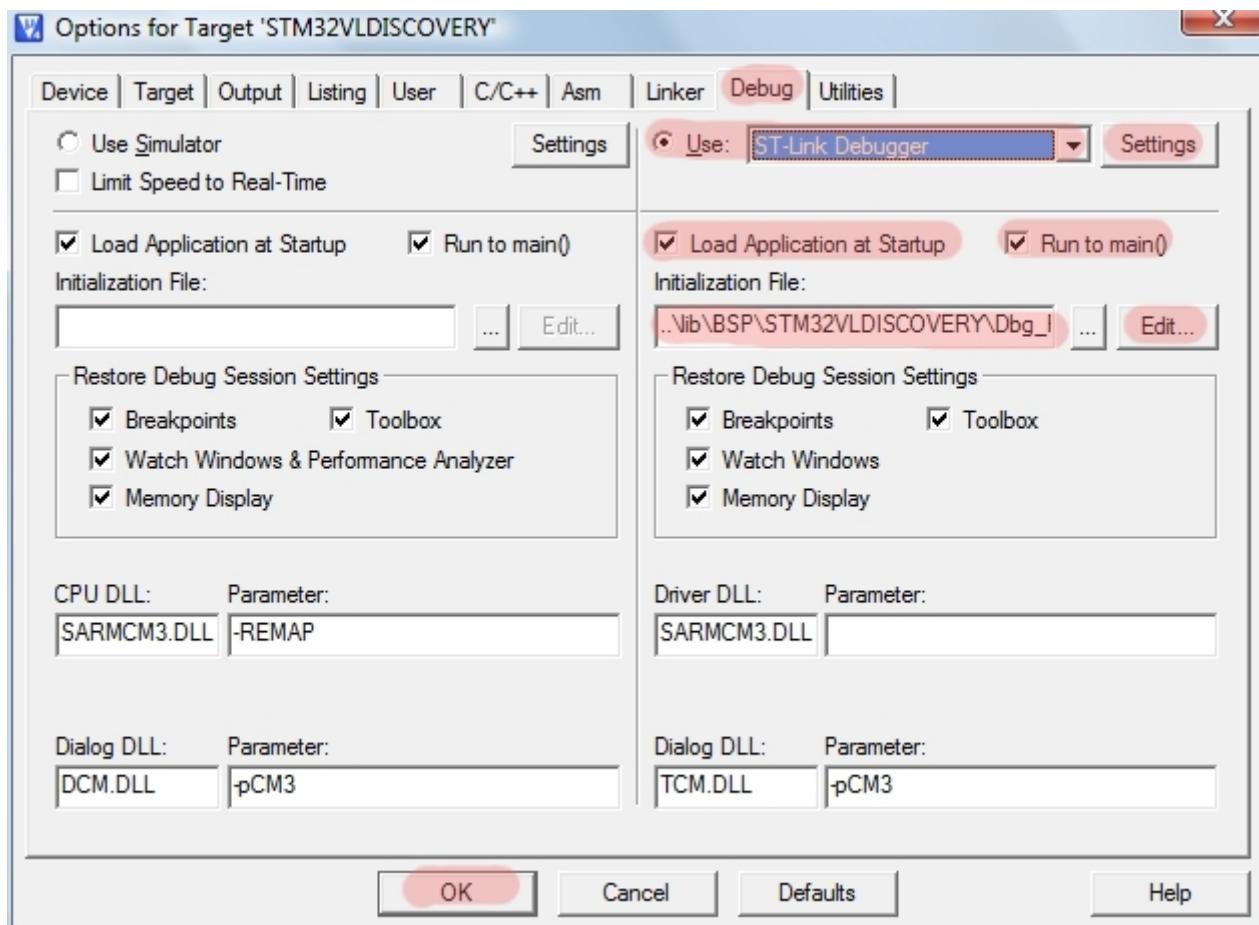


На вкладке **C/C⁺** и **Asm** нужно убрать дефайн **VECT_TAB_SRAM**, включающий в старто-вом коде **system_stm32fxyz.(c|h)** использование таблицы векторов в SRAM:

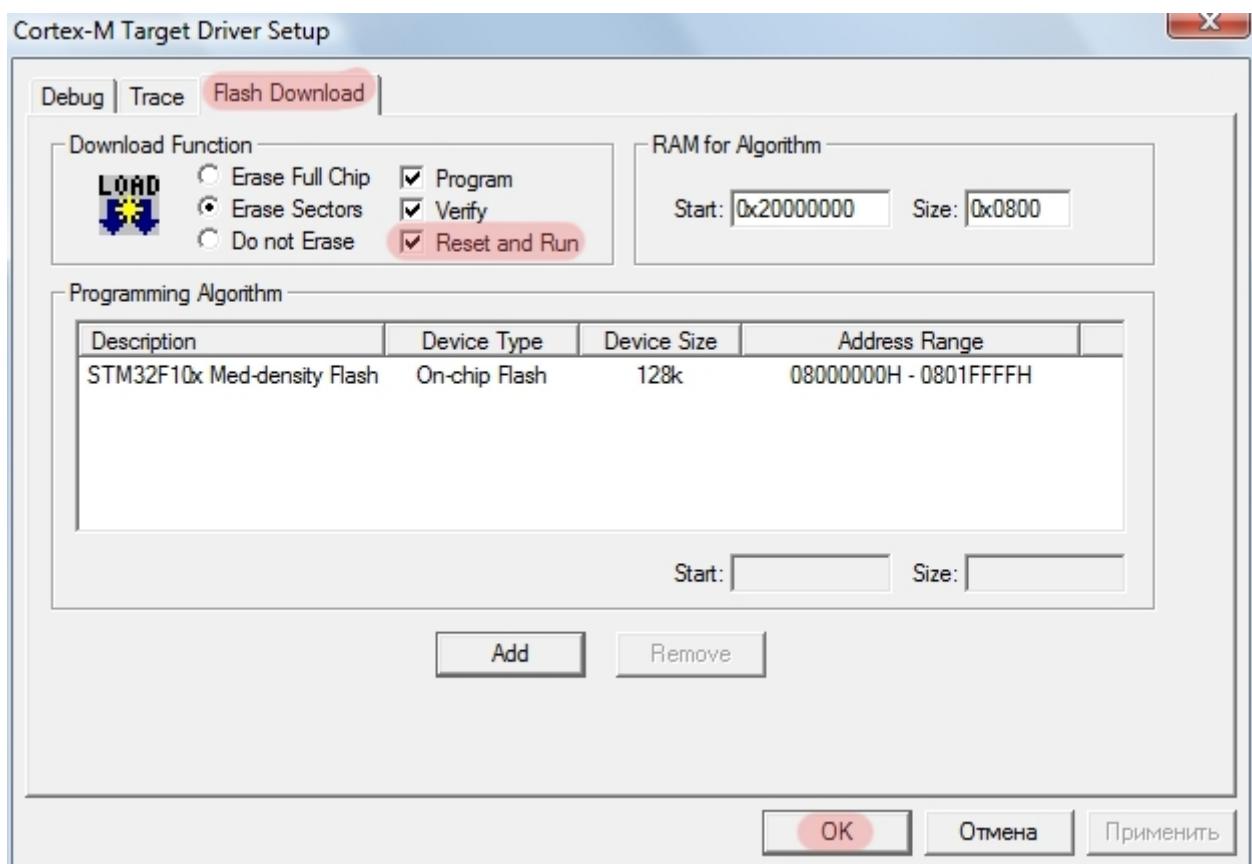
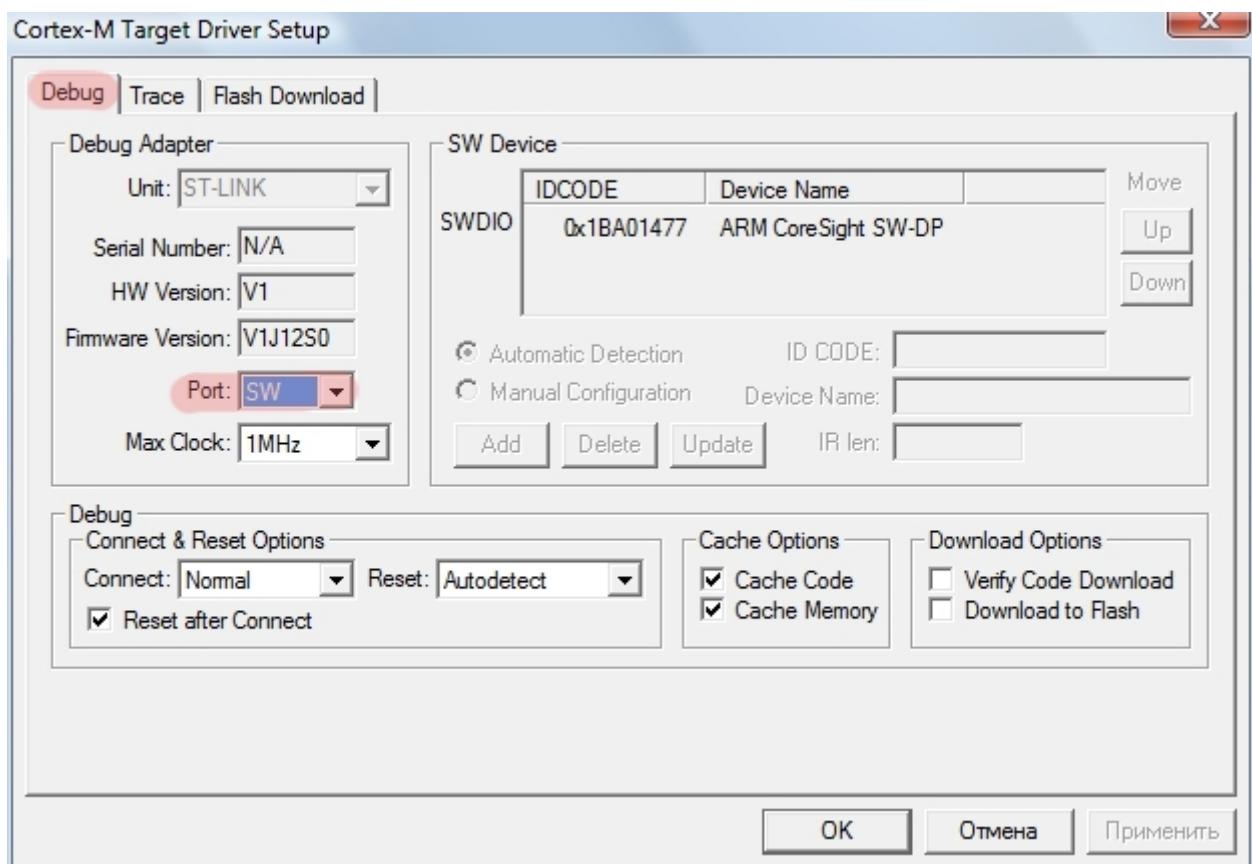


Теперь из-за глюка в Keil 10.1 придется пройтись по настройкам отладчика и прошивальщика — они почти всегда слетают.

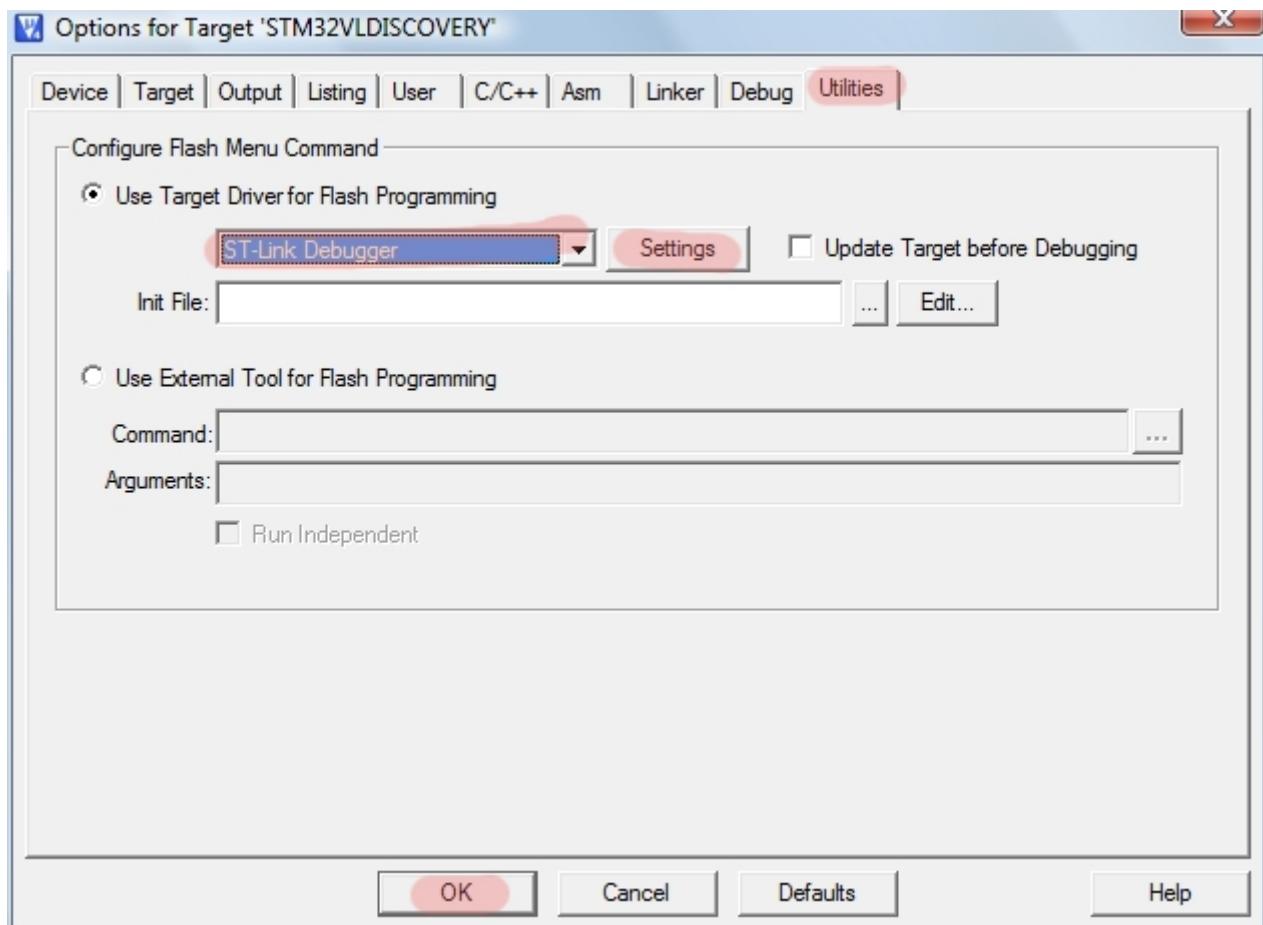
На вкладке **Debug** выберите **ST-Link Debugger**, поставьте галку на загрузке приложения (во флешу) при старте отладки, галку выполнять стартовый код до функции **main()**, щелкните кнопку **Edit** для открытия .ini файла с процедурой запуска отладчика (далее ее нужно будет изменить), и не забудьте щелкнуть кнопку **Settings** около программатора чтобы поправить слетевшие настройки:



Выставьте **Port: SW** для использования интерфейса SWD?? вместо JTAG??, которого нет в наплатном STlink12.1, а на вкладке **Flash Download** поставьте галку на **Reset and Run** и проверьте значения в других полях:



На вкладке Utilities еще раз проставьте слетевший STlink12.1:



Теперь осталось всего лишь ☺ отредактировать файл `Dbg_RAM.ini`, закомментировав загрузку регистров, и нажать **Ctrl**+**S** для сохранения.

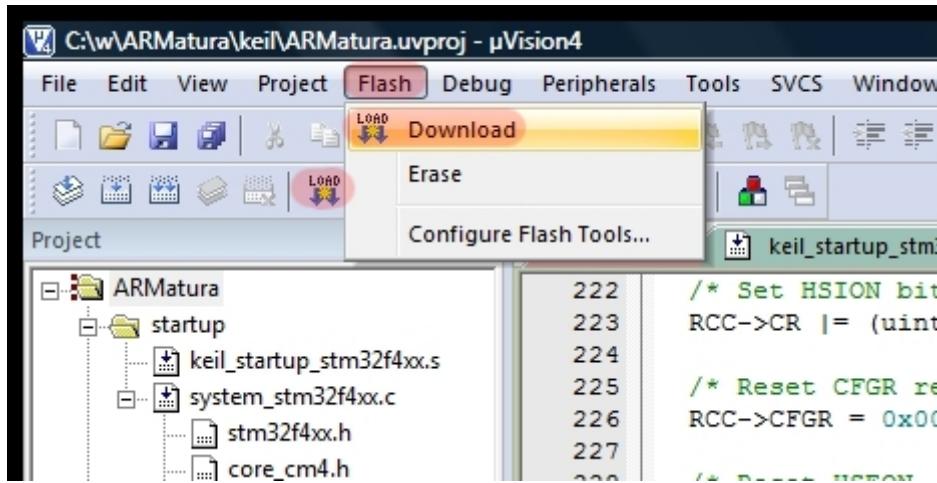
```

6 * This file is part of the uVision/ARM
7 * This software may only be used under
8 * end user licence from KEIL for a co-
9 * development tools. Nothing else giv-
10 *
11 * This software is supplied "AS IS" w-
12 *
13 * Copyright (c) 2011 Keil - An ARM Co-
14 *-----
15
16 /**
17 *-----*
18 *-----*
19 FUNC void Setup (void) {
20 // SP = _RDWORD(0x20000000);
21 // PC = _RDWORD(0x20000004);
22 // _WDWORD(0xE000ED08, 0x20000000);
23 }
24

```

Так подобно перенастройка на использование флеш приведена здесь для того, чтобы при необходимости вы сами смогли перенастроить проект в обратную сторону, если создадите его с нуля. Особое внимание при этом обратите на поле дефайнов на вкладках **C** и **Asm** — без них ваш код не будет компилироваться.

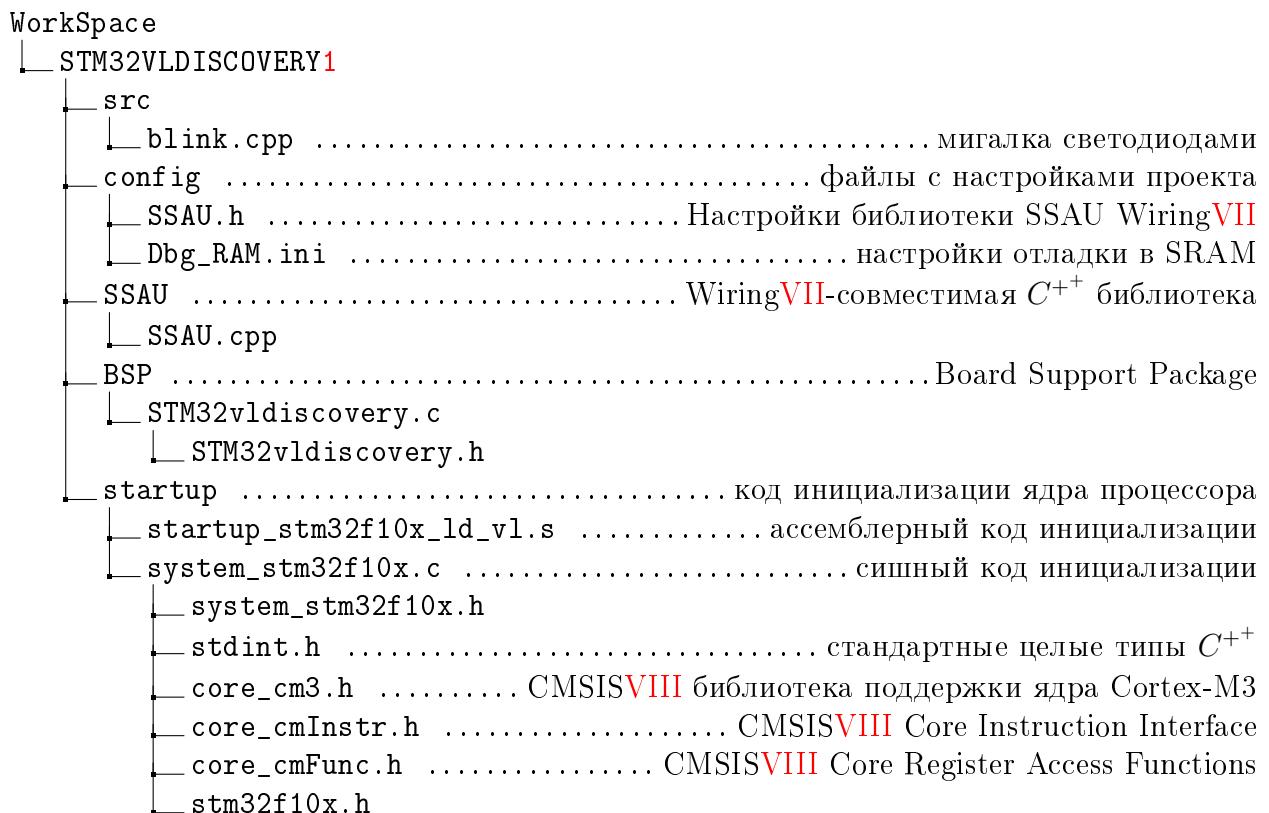
После сохранения настроек проекта осталось запустить прошивку программы:

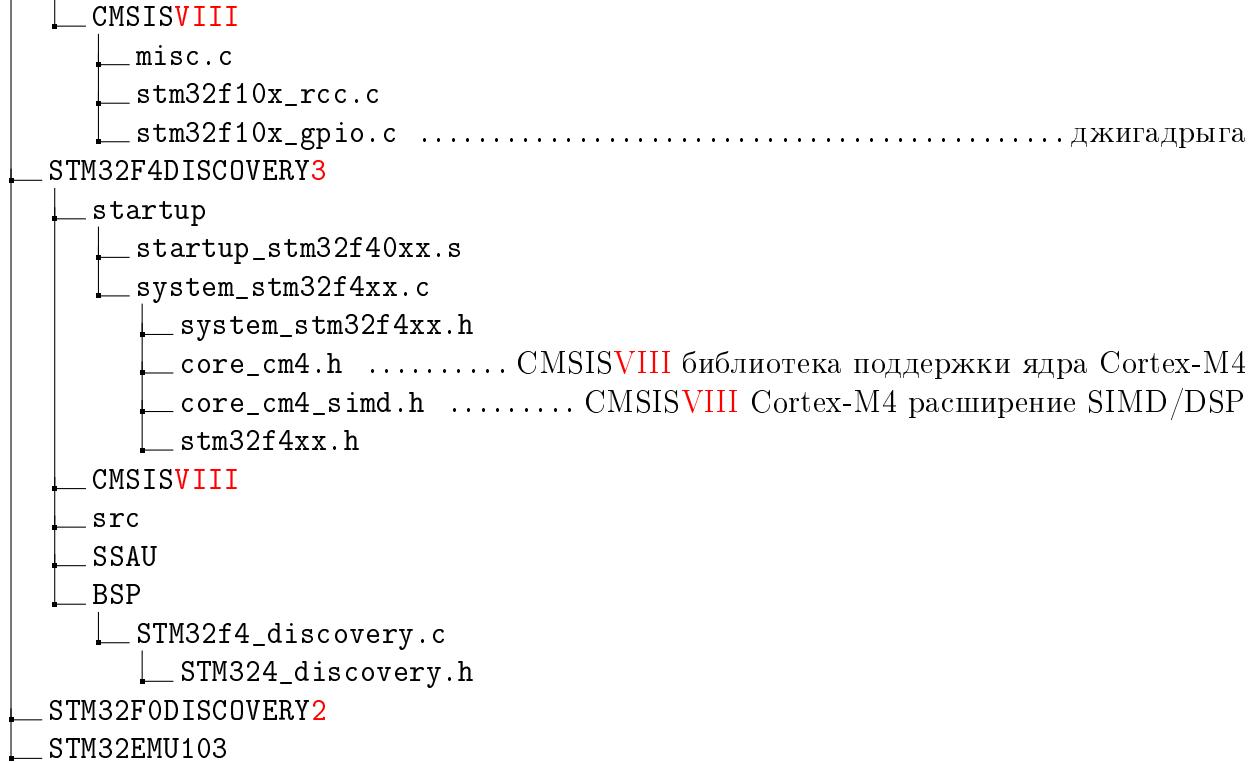


Запускается прошивка флешпамяти на отладочной плате, на программаторе мигает светодиод передачи данных, и после успешной прошивки наша плата начинает мигать светодиодами.

В качестве первой практической работы вы можете открыть новое окно Keil^{10.1}, и создать в нем новый проект, сохранив его куда-нибудь в `..\keil\>UserProject01.uvproj`, а затем попытаться повторить структуру файлов (описана в следующем разделе) и настройки проекта так, чтобы он у вас собрался и заработал в режиме отладки (с компиляцией в SRAM) и в конечном варианте (с прошивкой во флешу). Замените файл `blink.cpp` на свой `src\user\game.cpp` чтобы можно было экспериментировать со своим кодом не потеряв примеры проектов.

8.1 Структура файлов





В отличие от кристаллов типа Atmel AVR, при программировании для архитектуры Cortex-Mx принято использовать готовую библиотеку CMSISVIII, для которой проводится активная стандартизация среди производителей процессоров.

Cortex > Microcontroller > Software > Interface > Standard

<http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php>

Кроме основной части CMSISVIII, общей для всех производителей МК Cortex-Mx, каждый подставщик предоставляет аппаратно-зависимую часть библиотеки:

- стартовый код инициализации ядра процессора
- библиотеку периферии например STM32F Standard Peripherial Library, включающую код обеспечивающий доступ к устройствам, встроенным в кристалл в т.ч.
- USB OTG Host & Device
- DSP /SIMD Extension (для Cortex-M4F) системо-зависимую часть расширения CMSISVIII DSP

Список ссылок для загрузки библиотек см. ??.

8.2 *blink.cpp*

Сначала посмотрим код основной программы:

Listing 8.1: /src/blink/blink.cpp

```

/*
 * ARMatura book sample
 * https://github.com/ponyatov/ARMatura
 */

#include <SSAU.h> // используем Wiring - совместимую
                   // объектную библиотеку SSAU ASCL

void tick1s() {
    XLED.toggle();
}

void setup() {
    pinMode(WLED, OUTPUT); // set pin LD3 as output
    pinMode(XLED, OUTPUT); // set pin LD4 as output
}

void loop() {
    digitalWrite(WLED, HIGH); // set the LED on
    delay(777);             // wait for a second
    digitalWrite(WLED, LOW); // set the LED off
    delay(777);             // wait for a second
}

```

Как видим — в ней нет абсолютно ничего сложного: только подключение библиотеки Wiring^{VII}, две типовые функции `setup()` и `loop()`, и новая функция `tick1s()`, срабатывающая каждую секунду от таймера асинхронно основной программе.

При активном использовании `tick1s()` нужно быть аккуратнее, чем с обычной аудиоритной — код должен быть thread-safe, т.е. нужно учитывать что ваша программа может быть прервана в любой точке, и запущена функция `tick1s()`. Если вы выполняете какие-то действия с периферией или со структурами данных и в основном коде, и в функции `tick1s()`, возможны всякие детские неожиданности, характерные для многозадачного программирования.

Файлы разделов CMSIS, BSP и SSAU подробно рассмотрены далее в отдельных главах. На текущем слое знаний не стоит в них углубляться: для начала лучше заставить вашу плату работать, и выполнять какие-то простые бесполезные функции, а подробности библиотек изучим позднее.

Глава 9

Hell Of World

Часть IV

Средства разработки

Глава 10

IDE

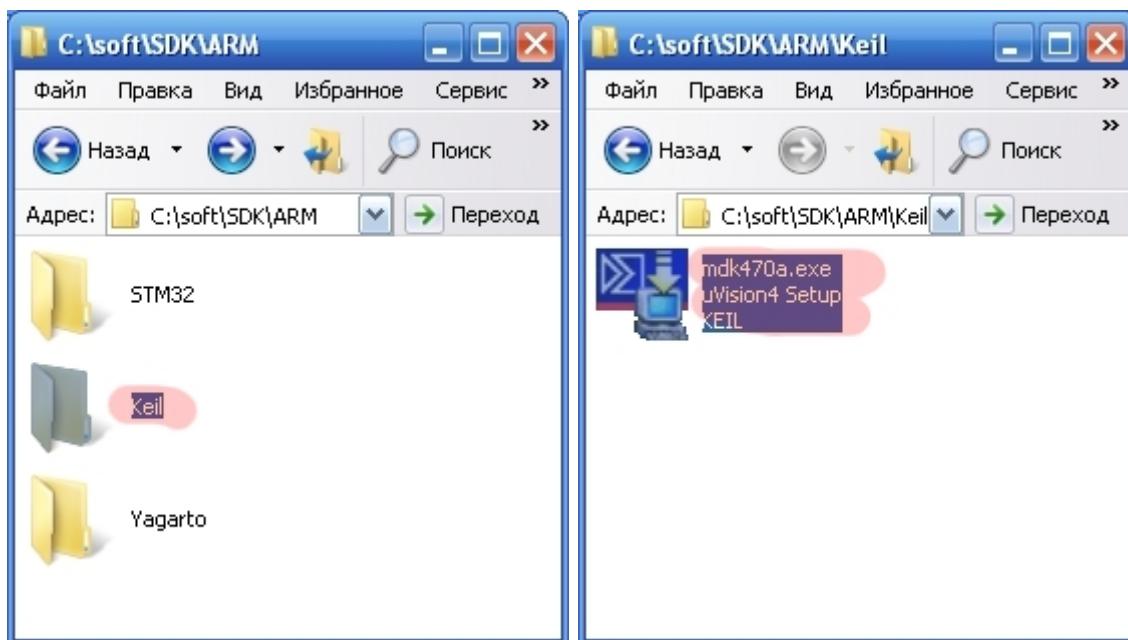
10.1 Keil

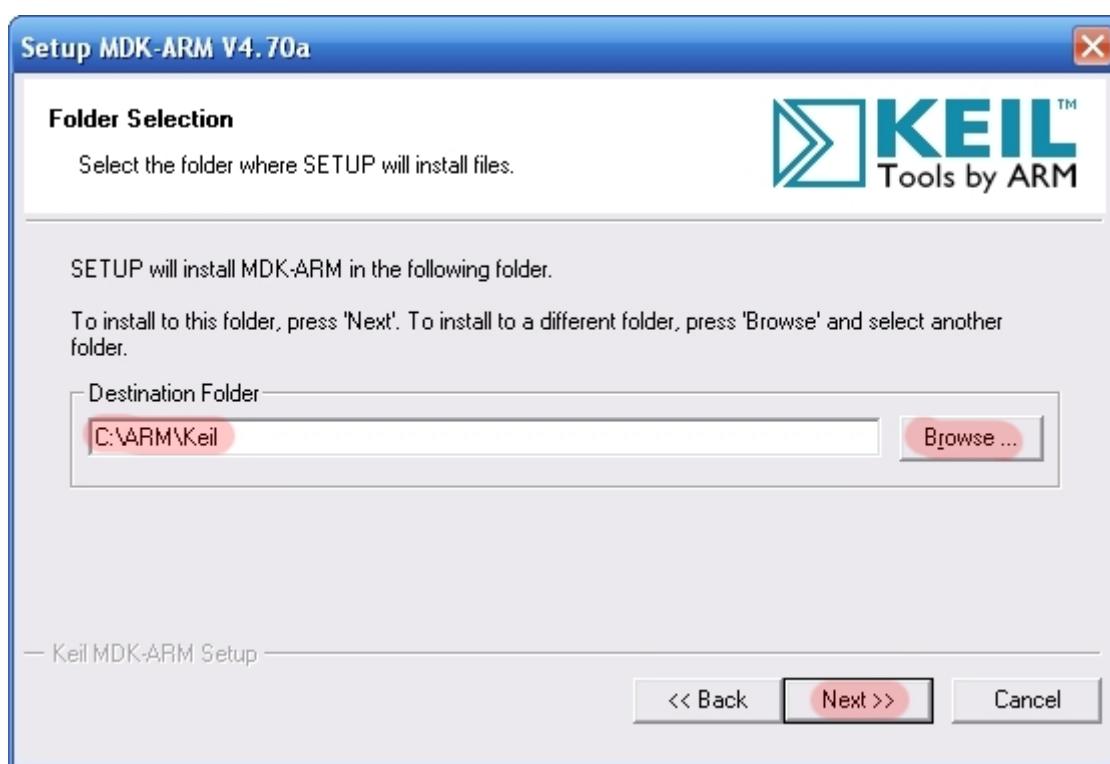
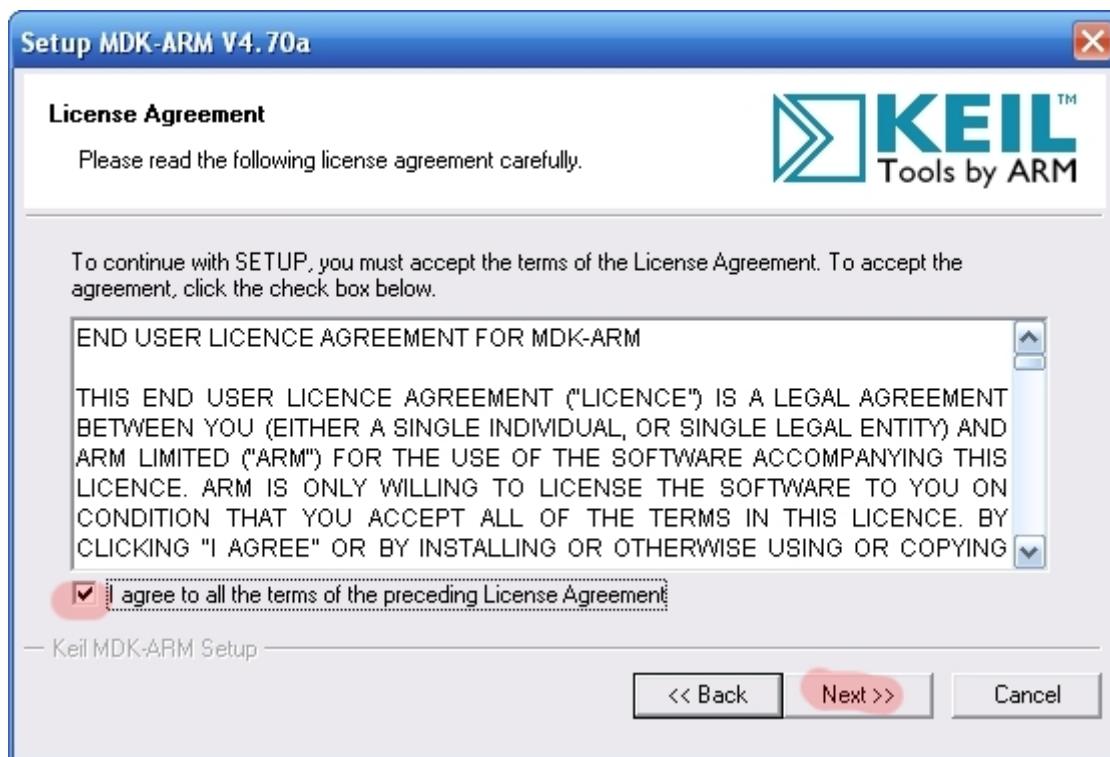


Для начала освоения программирования для ARM рекомендуем использовать бесплатный пакет от Keil: <http://www.keil.com/arm/mdk.asp> — ограничения бесплатной версии в 32К кода вполне достаточно для начального освоения программирования под процессоры семейства Cortex-Mx, а затем уже можно переползать на открытое ПО: GNU toolchain 11.1, Eclipse 10.4 и Linux XVI.

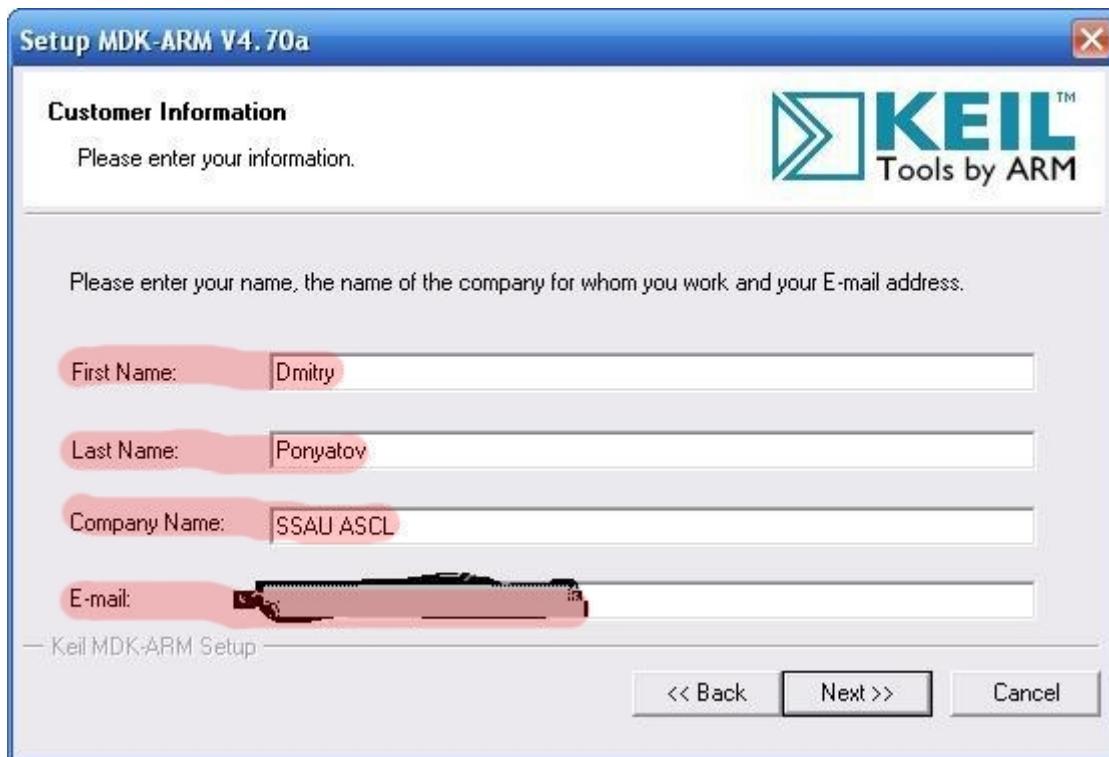
Установка

Качаем пакет с официального сайта, заполнив анкету: <https://www.keil.com/demo/eval/arm.htm>.

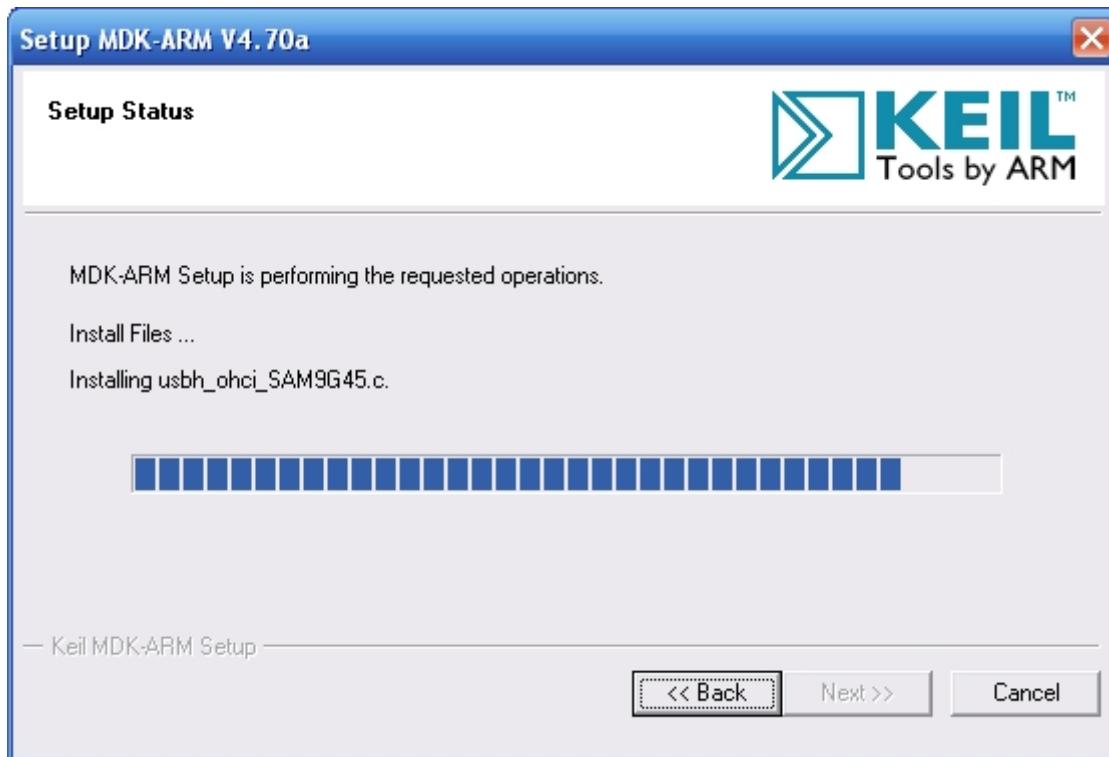


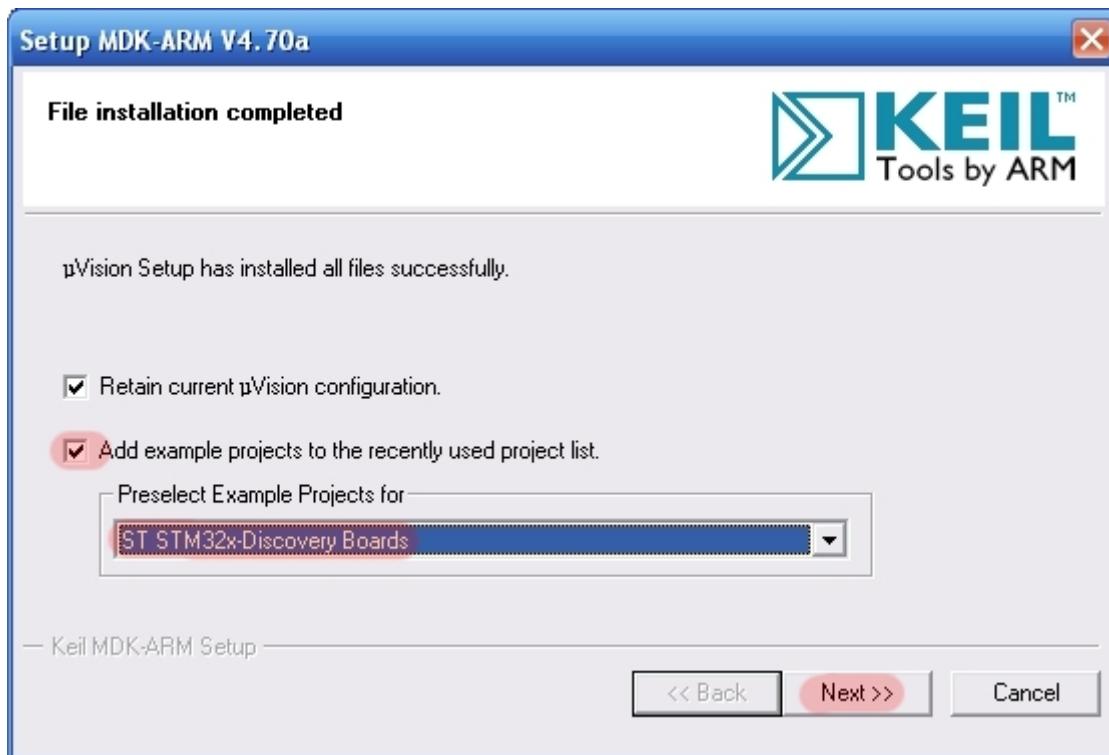


Путь установки пакета

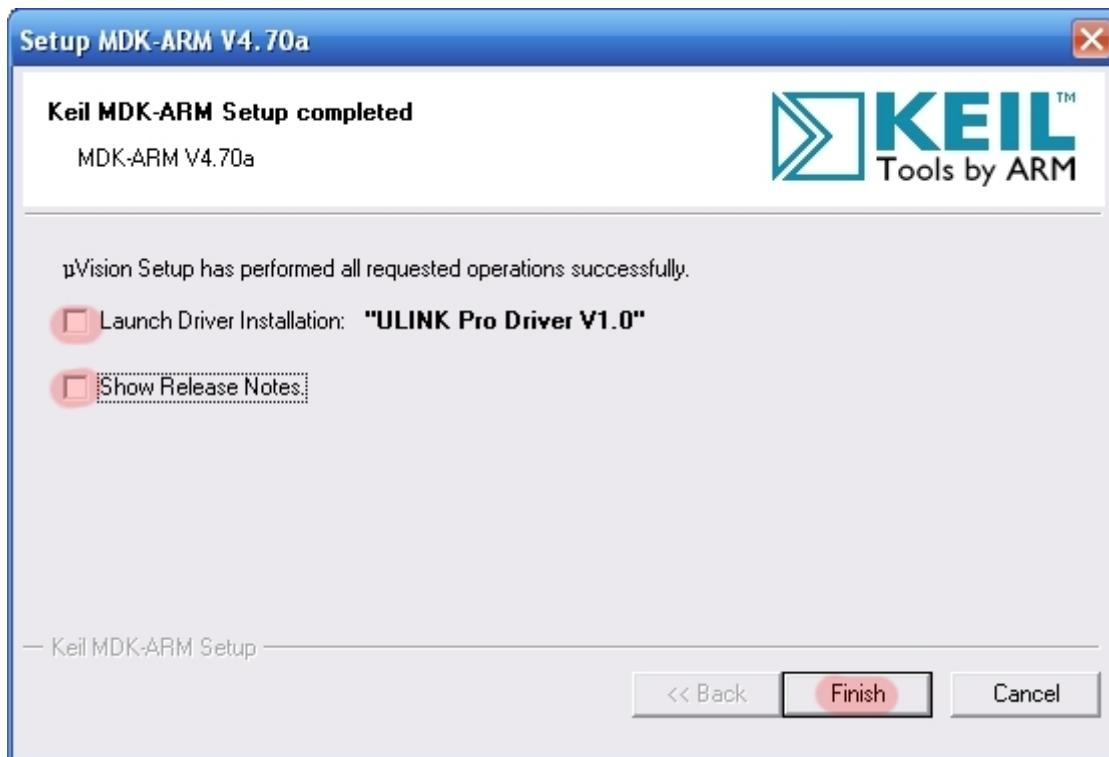


Личные данные: имя, название компании или hobbit, адрес электронной почты.





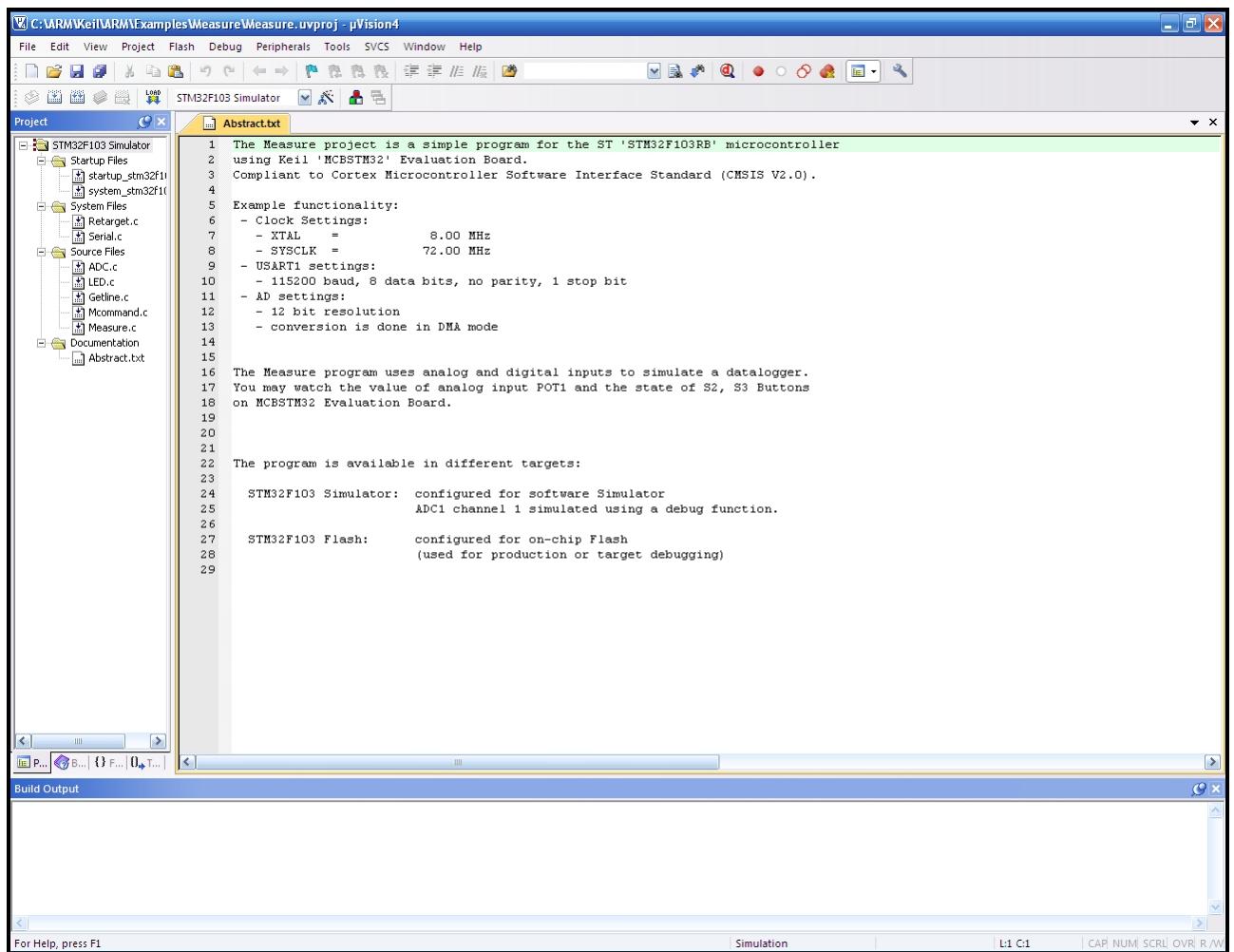
Укажите какие примеры кода добавить в список recently used project list: для работы с другими типами микропроцессоров выберете соответствующий раздел, или оставьте Simulation Hardware по умолчанию.



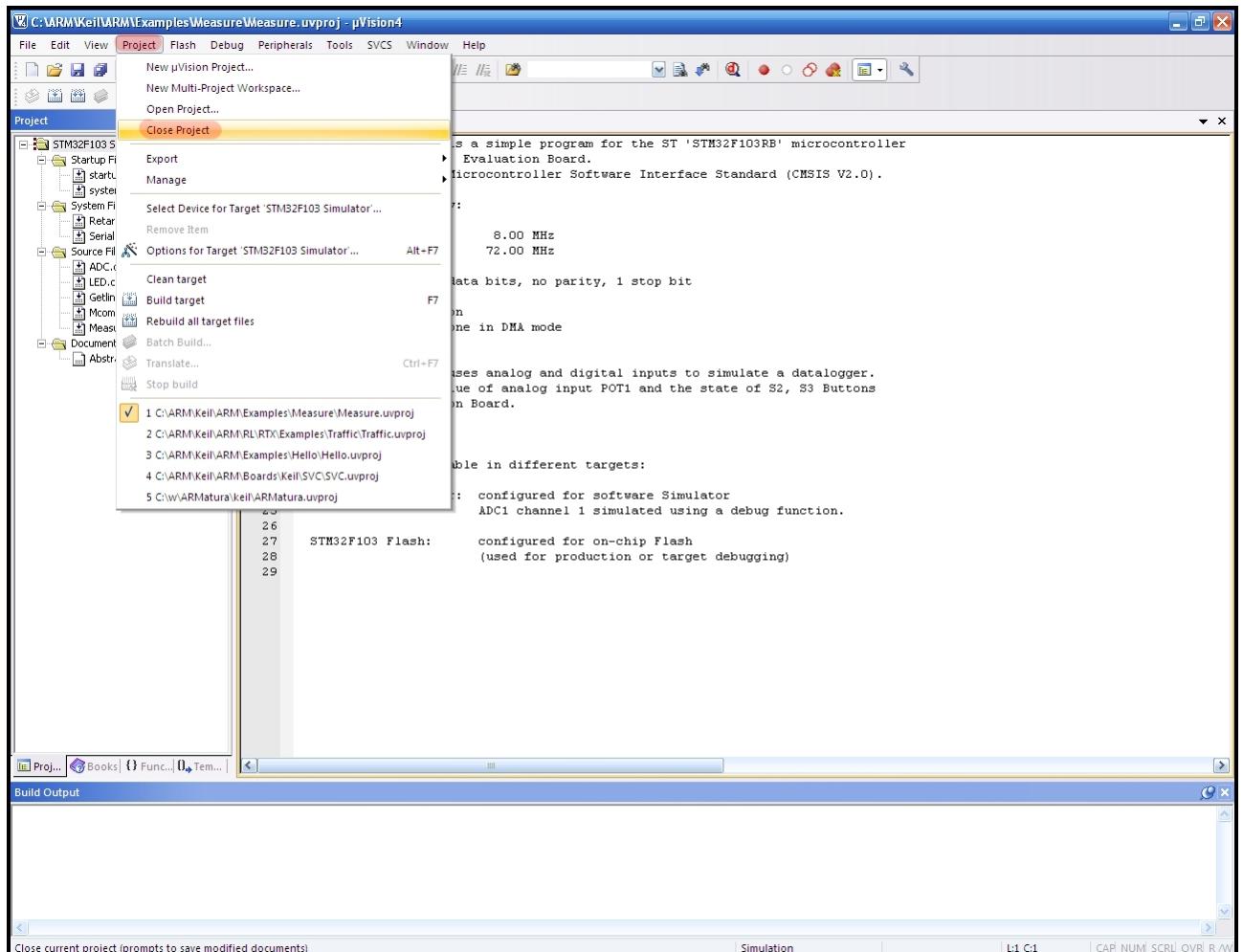
Снять установку драйвера программатора ULINK (если у вас его нет) и вывод текстового файла с последними изменениями Keil.



После запуска открывается проект по умолчанию, настроенный для программного симулятора STM32F103.



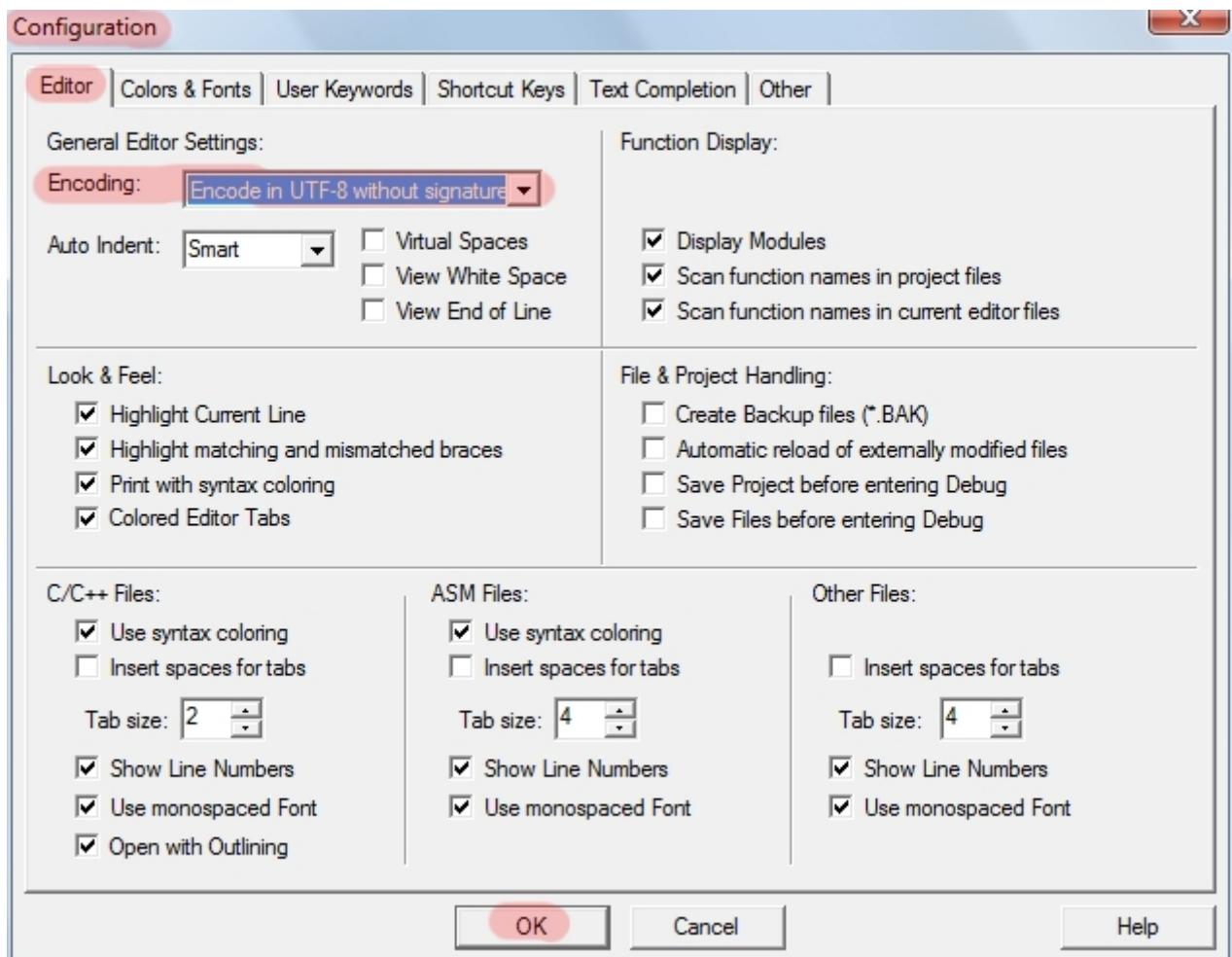
Нужно его закрыть,



и вернуться в раздел III и открыть первый проект 8.

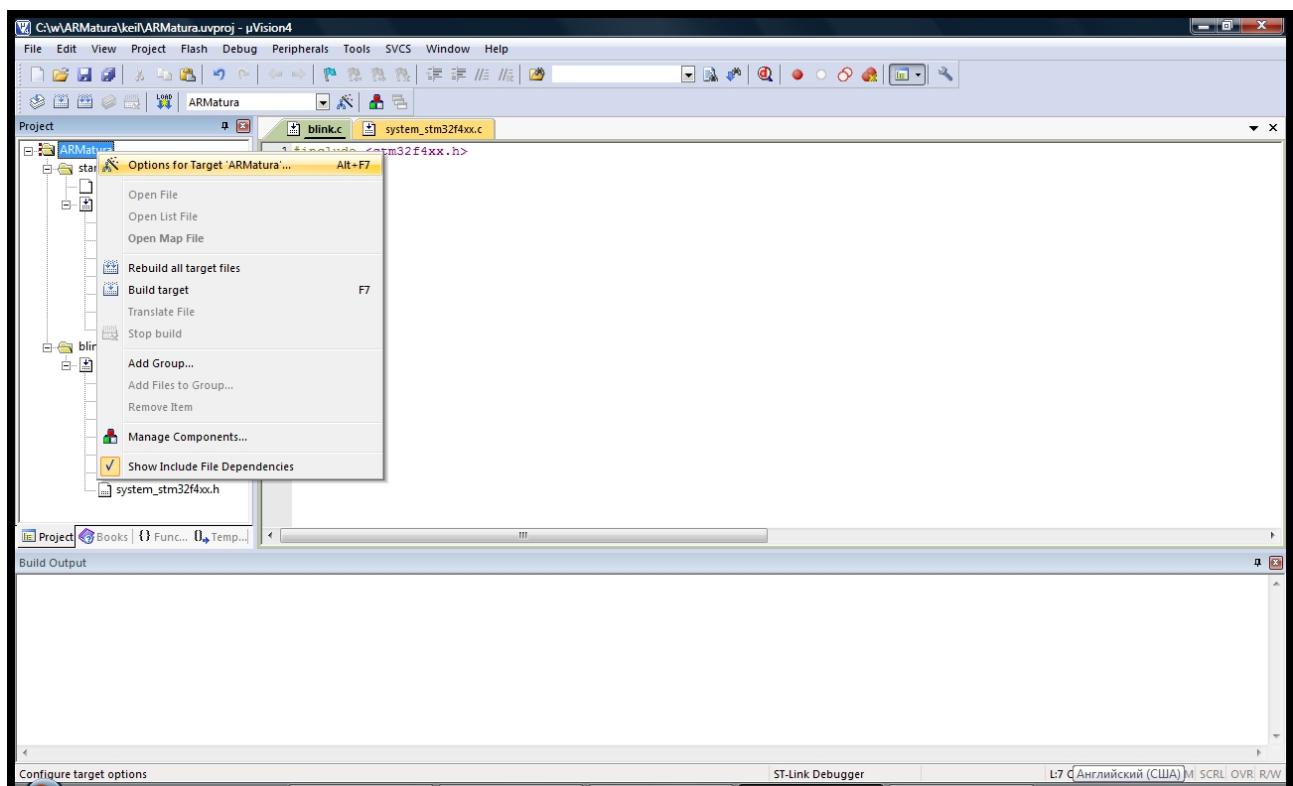
10.2 Настройки среды

По умолчанию Keil 10.1 косячит с кириллицей, поэтому идем в меню **Edit > Configuration > Editor** и устанавливаем кодировку UTF8

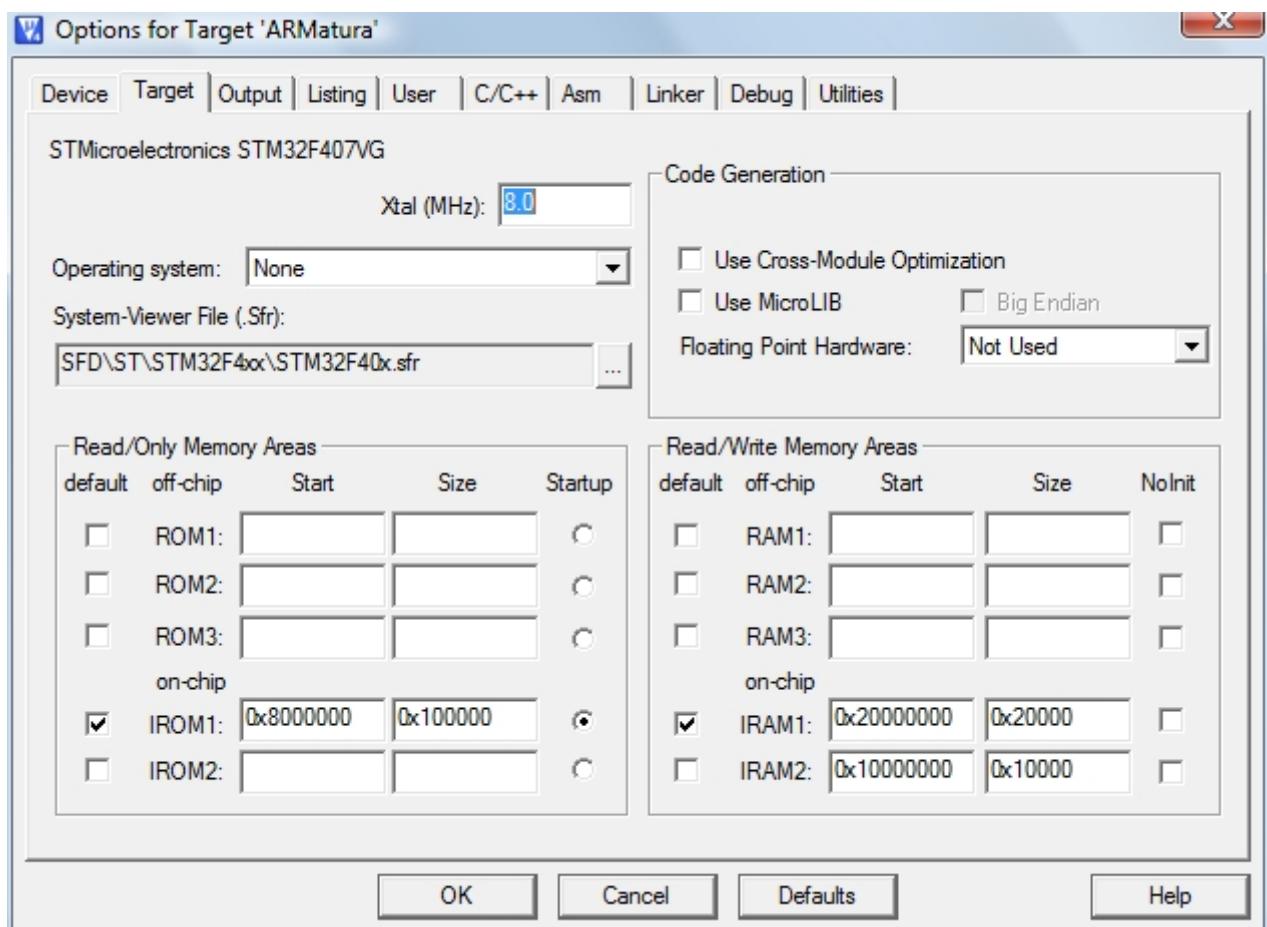


10.3 Настройки проекта

Настройки проекта вызываются из меню **Project** > **Options for Target 'ARMatura'...**, комбинацией клавиш **Alt** + **F7**, или выбором аналогичной опции из контекстного меню, вызываемого щелчком правой кнопкой мыши на корне дерева проекта **ARMatura** в левом окне **Project**.

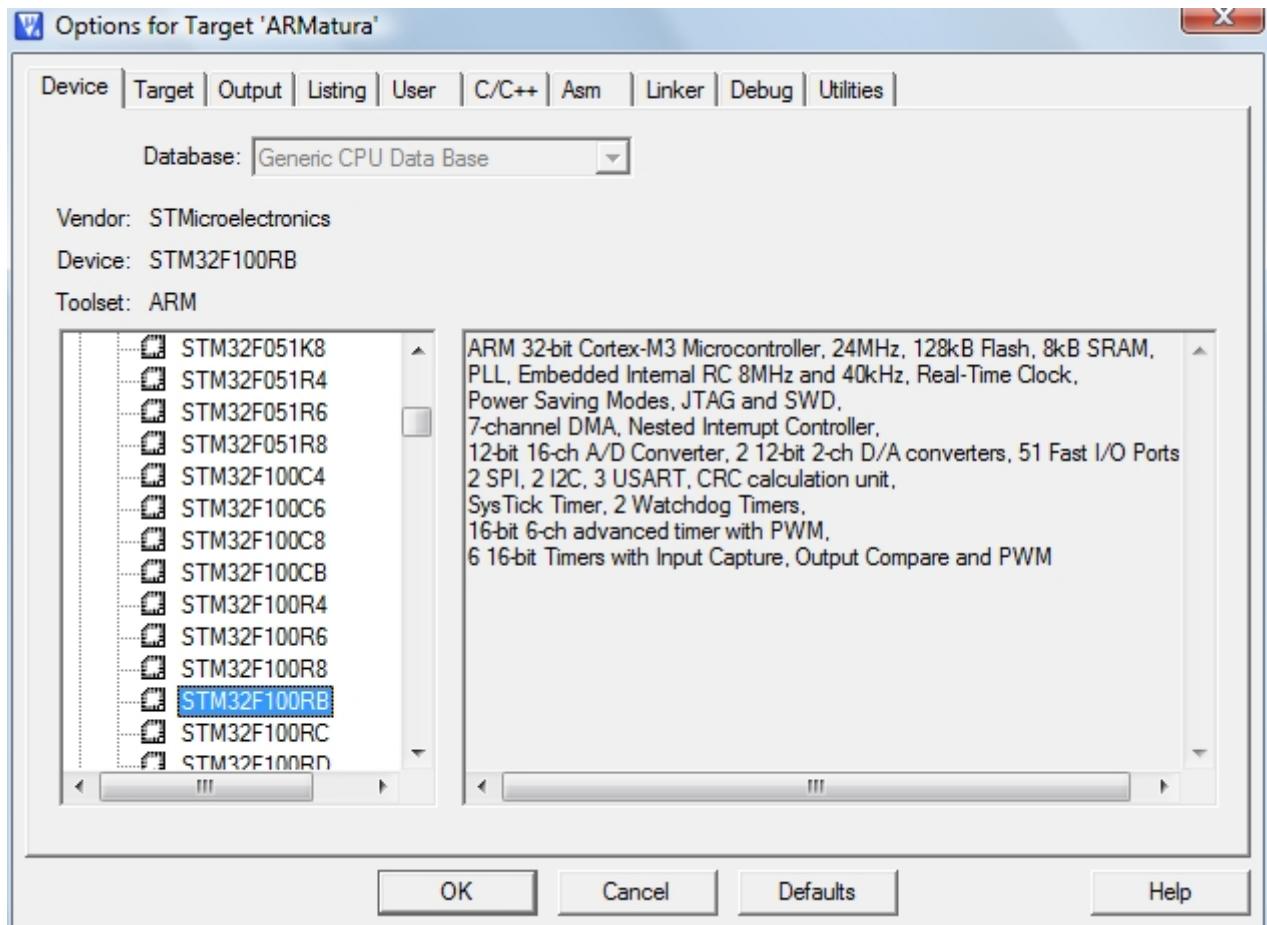


По умолчанию открывается вкладка **Target**:

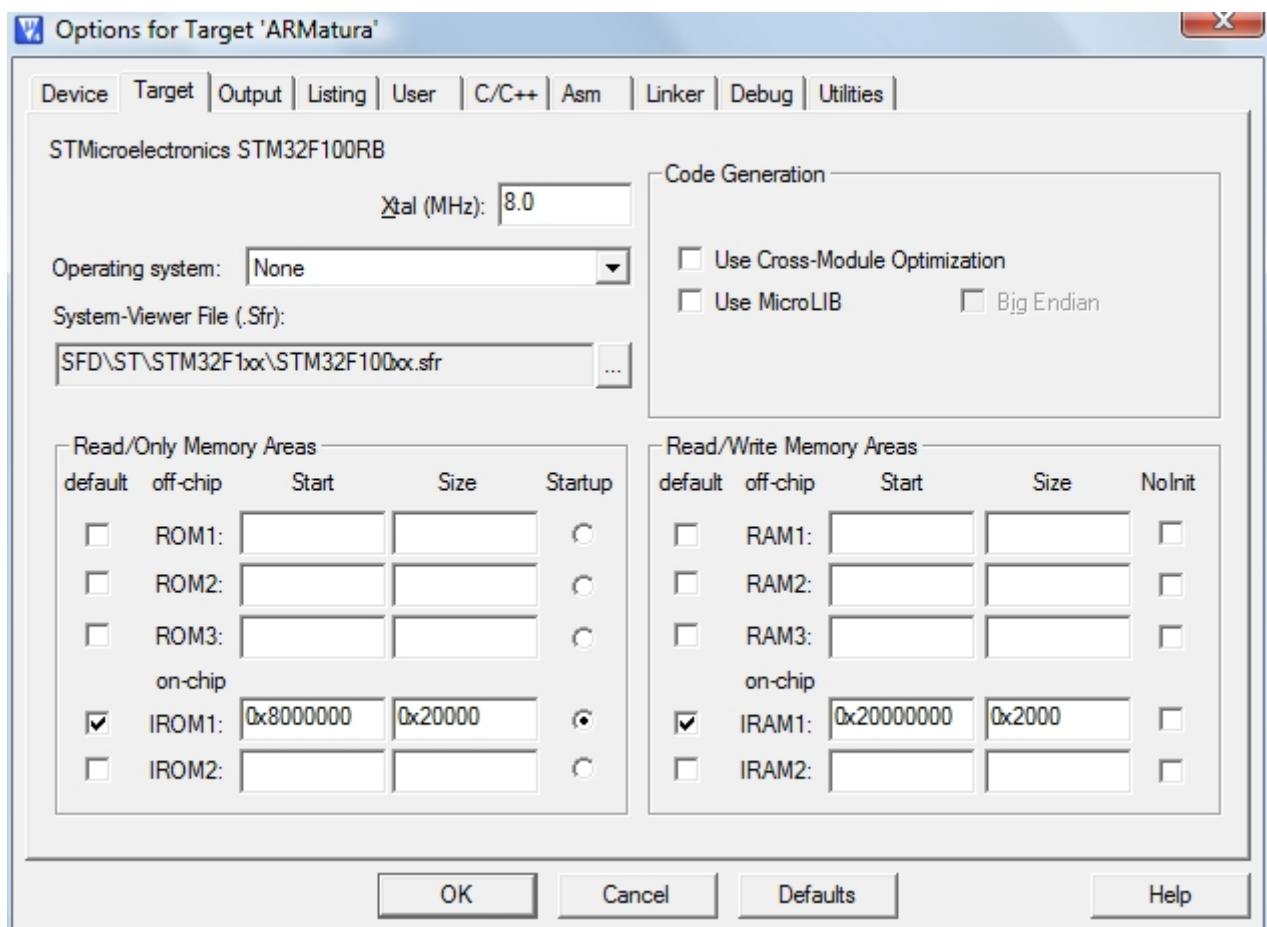


Xtal (MHz)	8.0	текущий процессор частота внешнего кварца на платах Discovery обычно стоит 8
Operating system:	None Use MicroLIB	или OCPB Keil RTX использовать libc от Keil
Floating Point Hardware:	Not Used	для Cortex-M4 доступен аппаратный FPU
	На этой вкладке также прописывается карта встроенной и внешней памяти Flash/SRAM.	

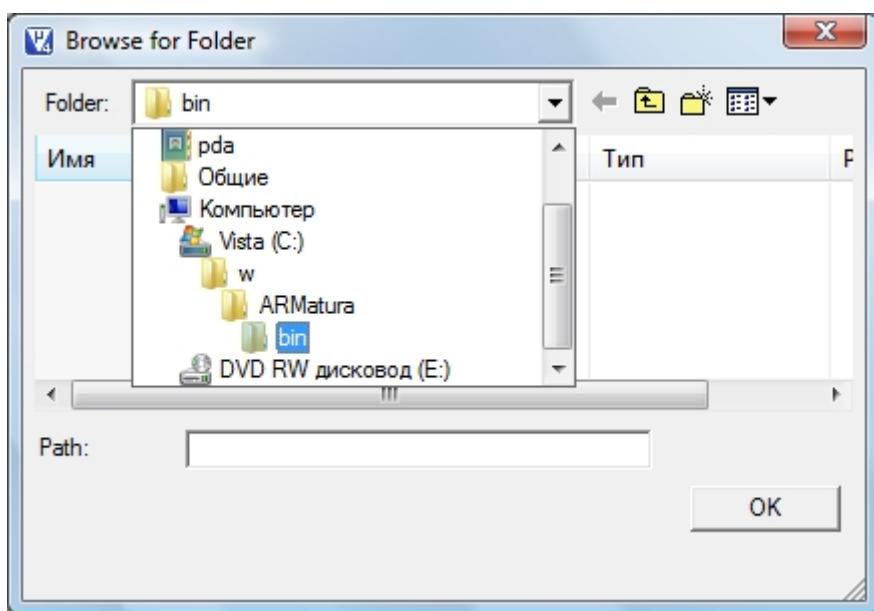
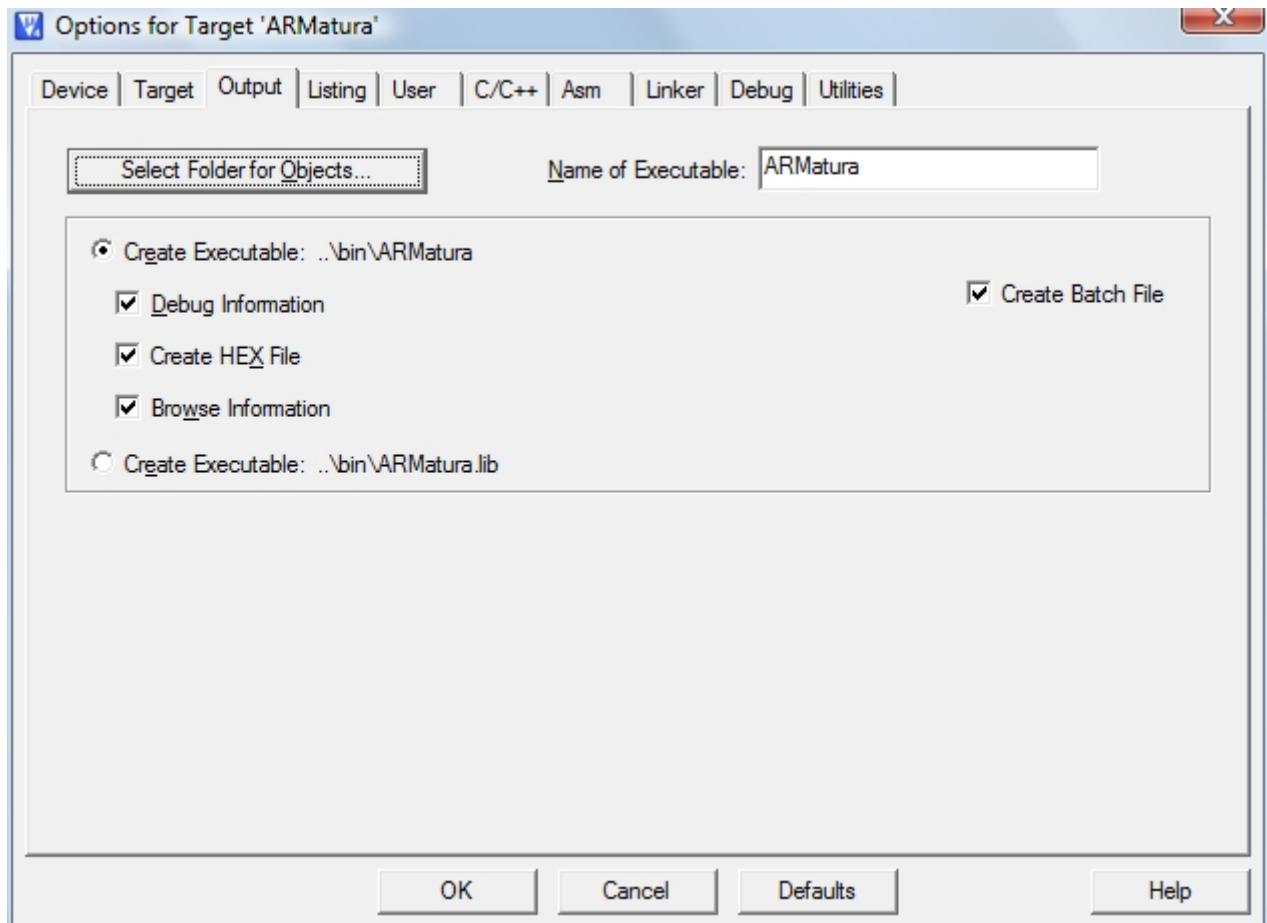
При необходимости собрать проект под другой процессор открываем вкладку **Device**, переконфигурируем проект под другую плату – STM32VLDISCOVERY 1:



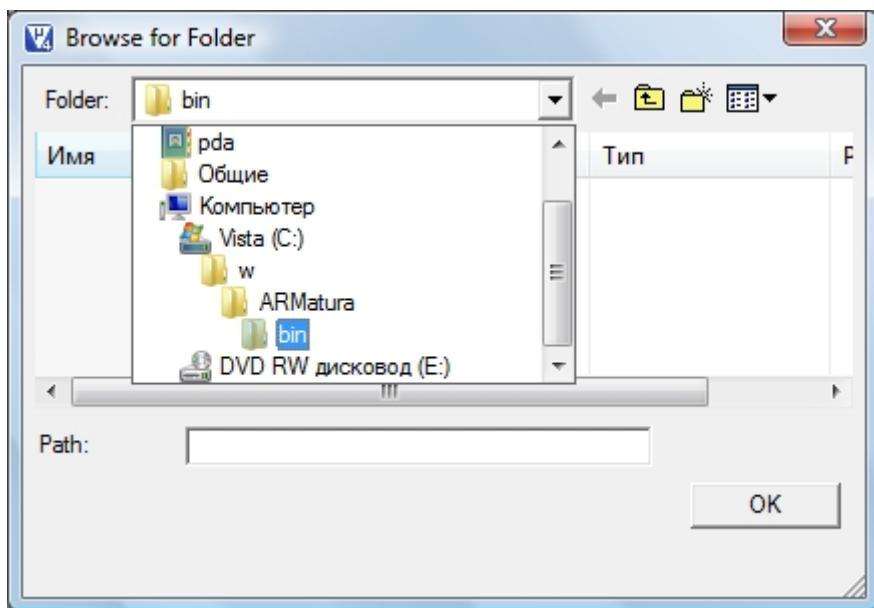
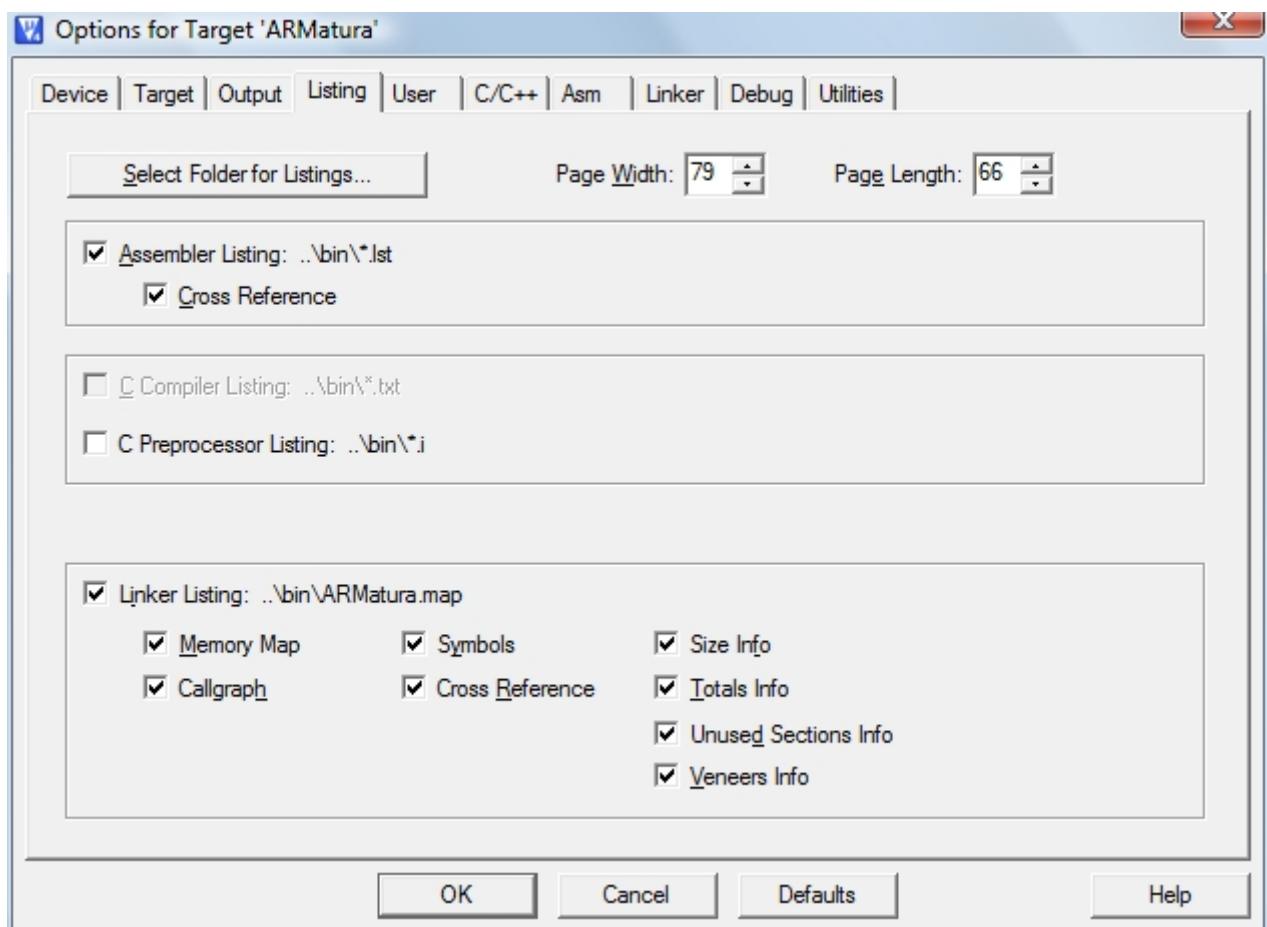
Обратите внимание что на вкладке **Target** изменился чип и настройки памяти:



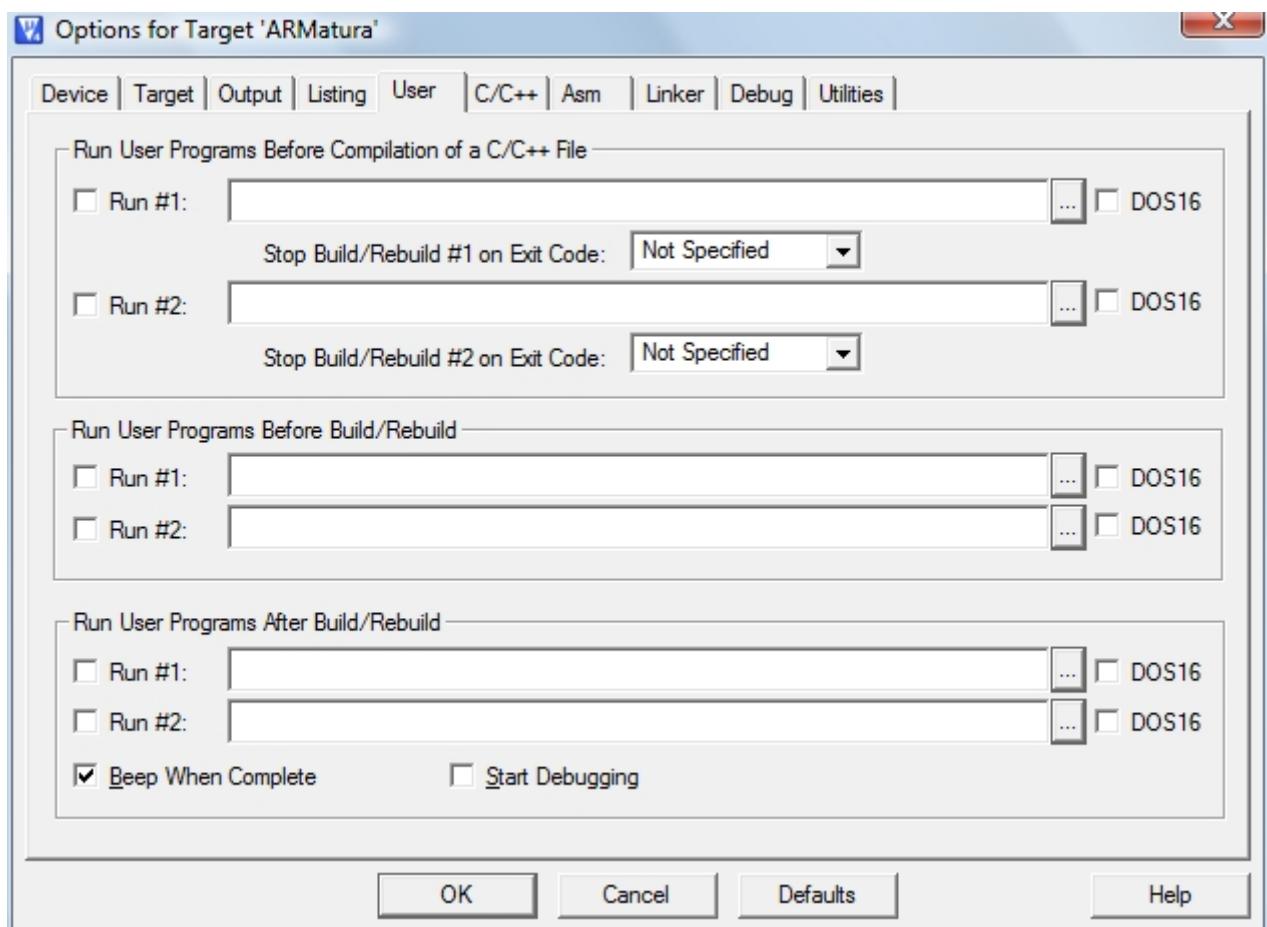
Output Настройки выходных файлов: каталог для бинарных файлов прошивки = firmware, при необходимости можно использовать .hex файл прошивки, имя файла прошивки



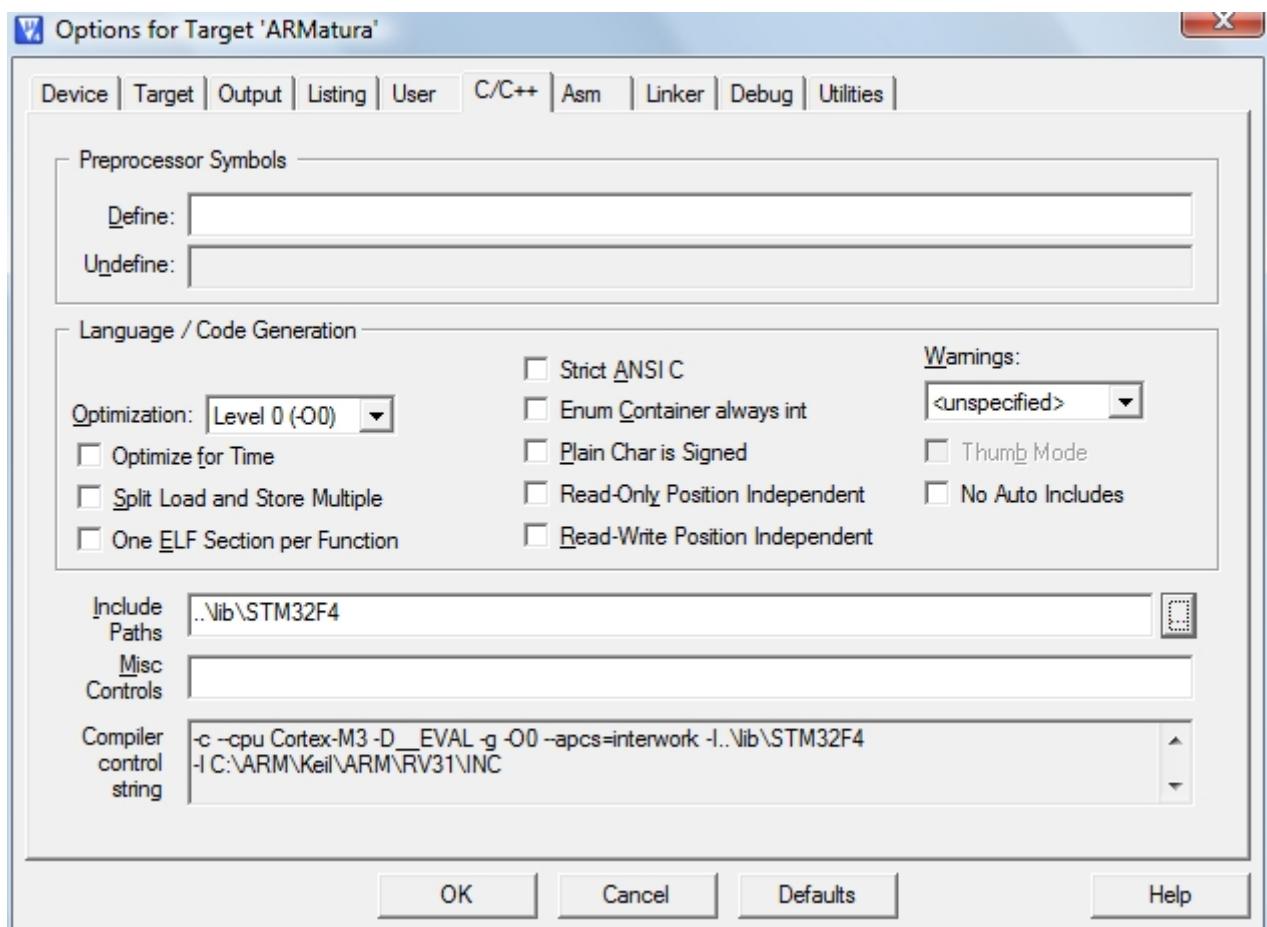
[Listing] Настройки генерации ассемблерных листингов в тот же каталог с прошивкой



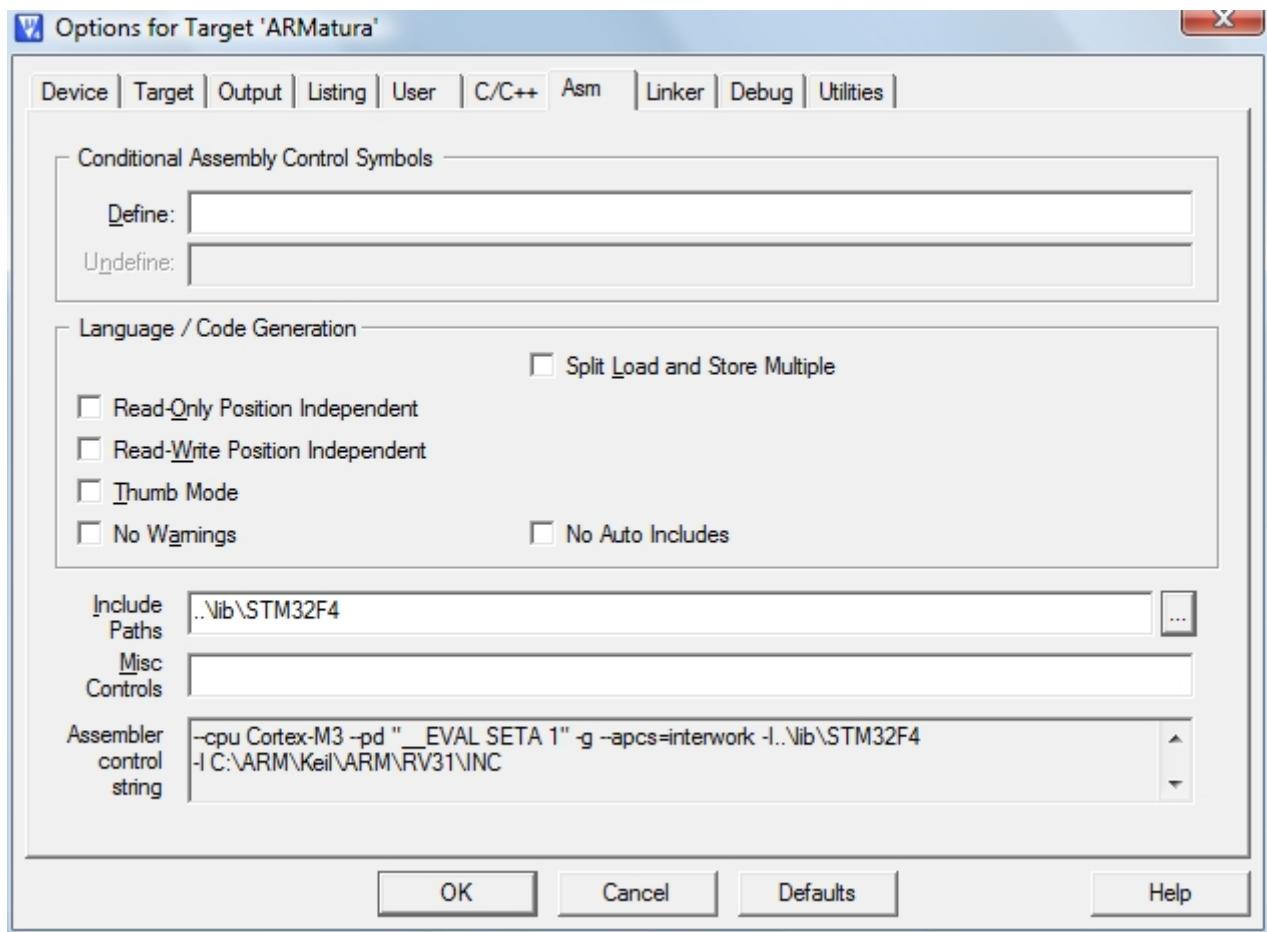
User Запуск пользовательских программ во время сборки проекта: пока не используем



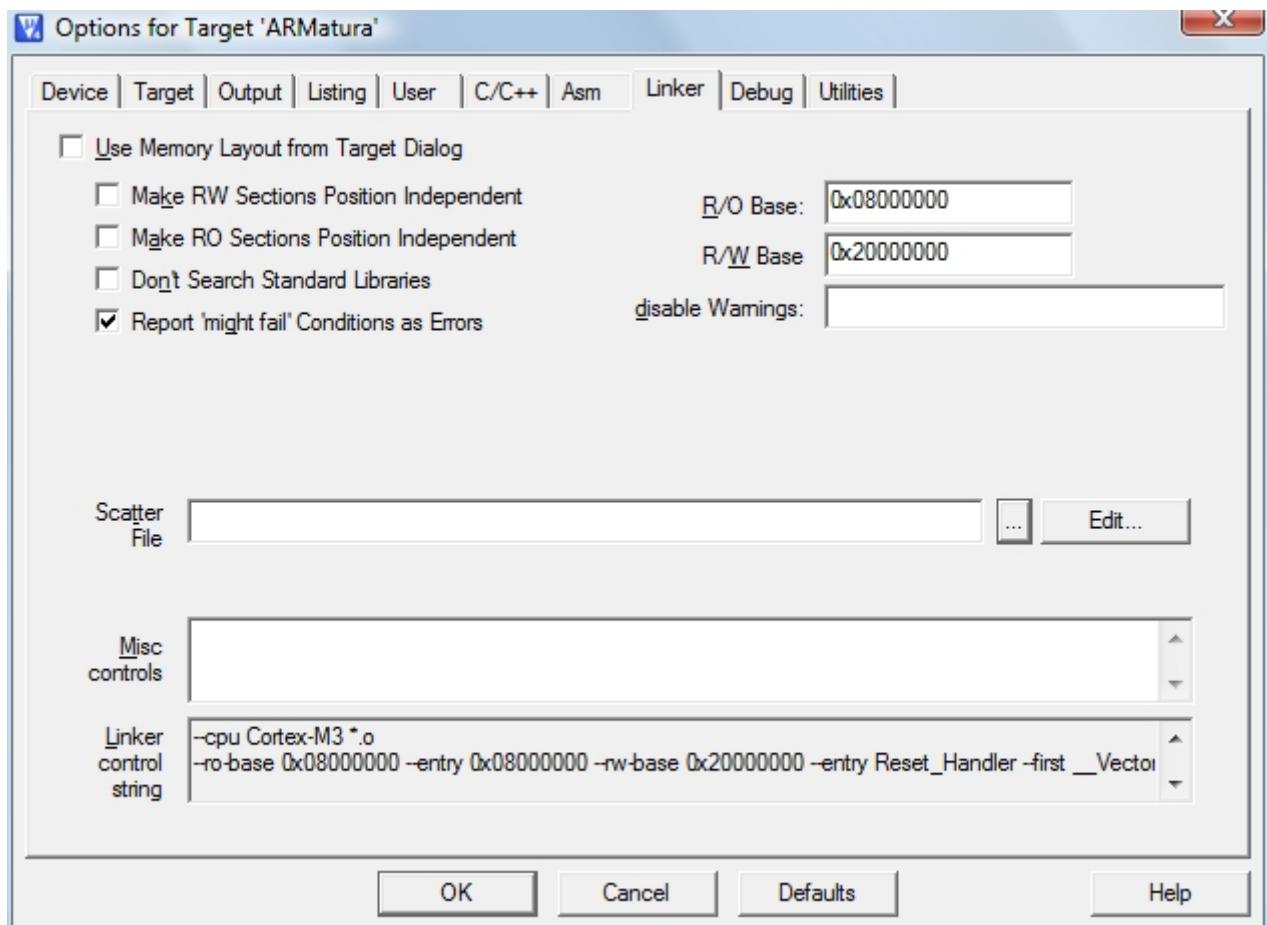
C/C++ Настройки компилятора C^{++} : опции оптимизации, каталоги библиотек и включаемых .h файлов, в нижнем поле прописывается полная командная строка вызова компилятора, которая может вам в дальнейшем понадобится, если потребуется компилировать без использования Keil IDE.



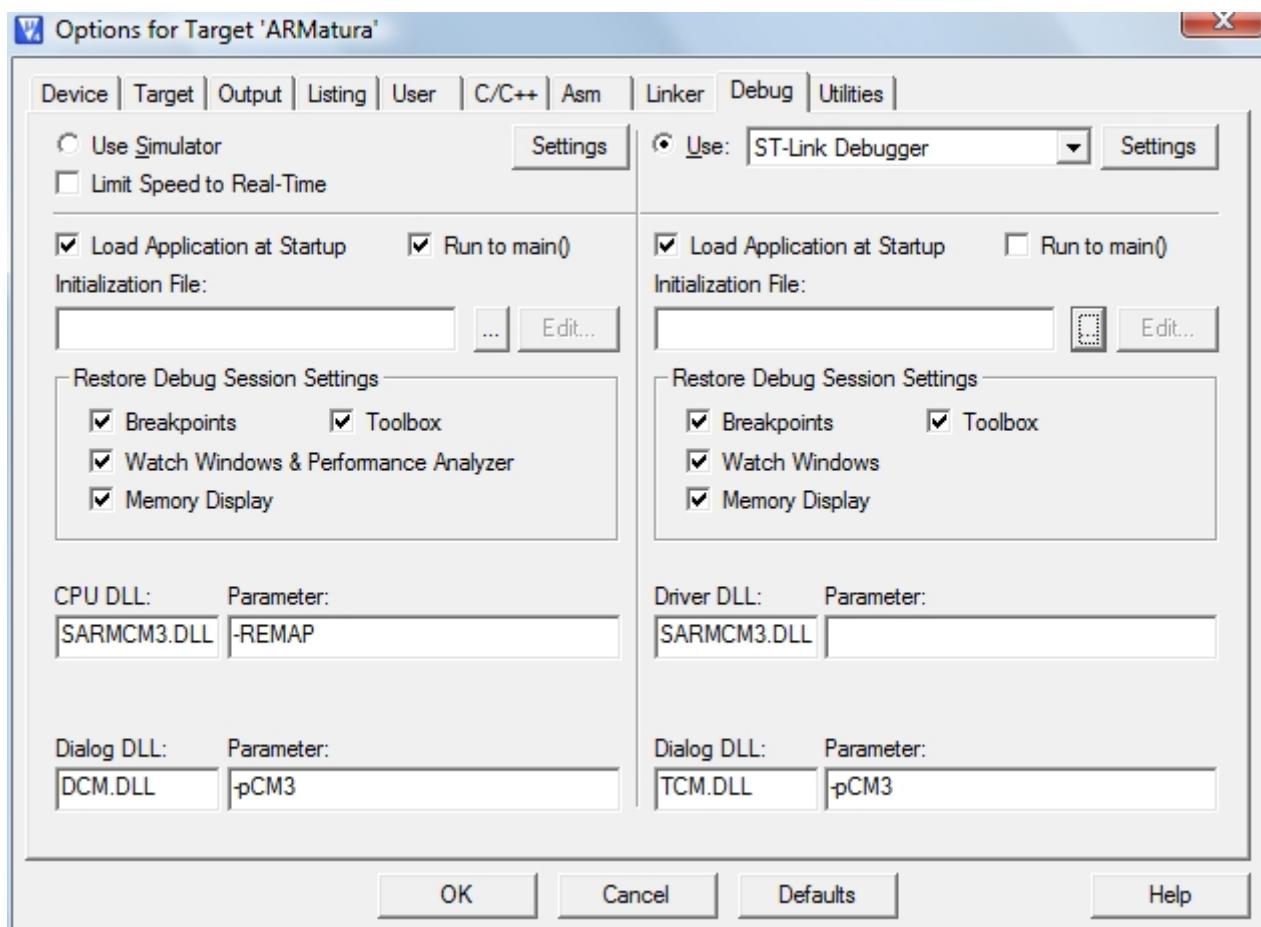
Asm Настройки ассемблера, приблизительно те же что и для C^{++}



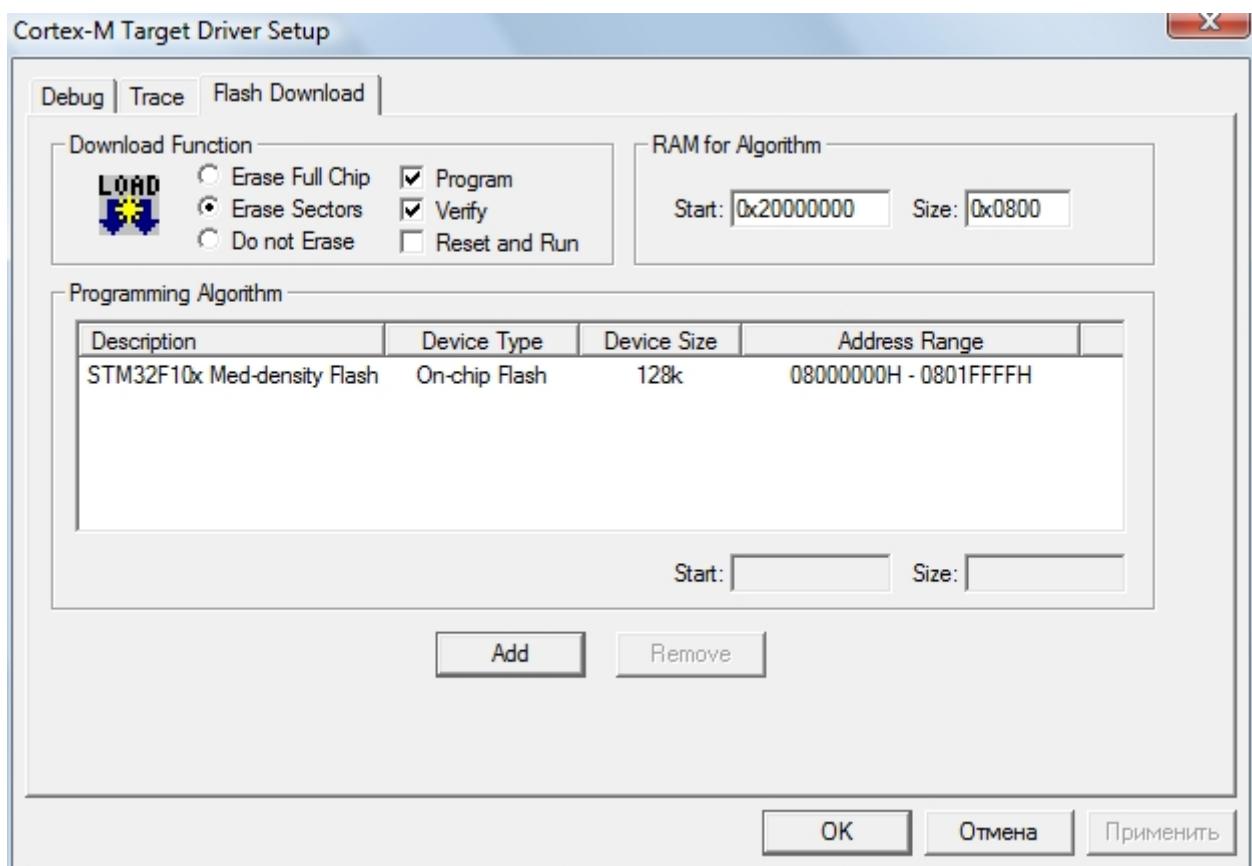
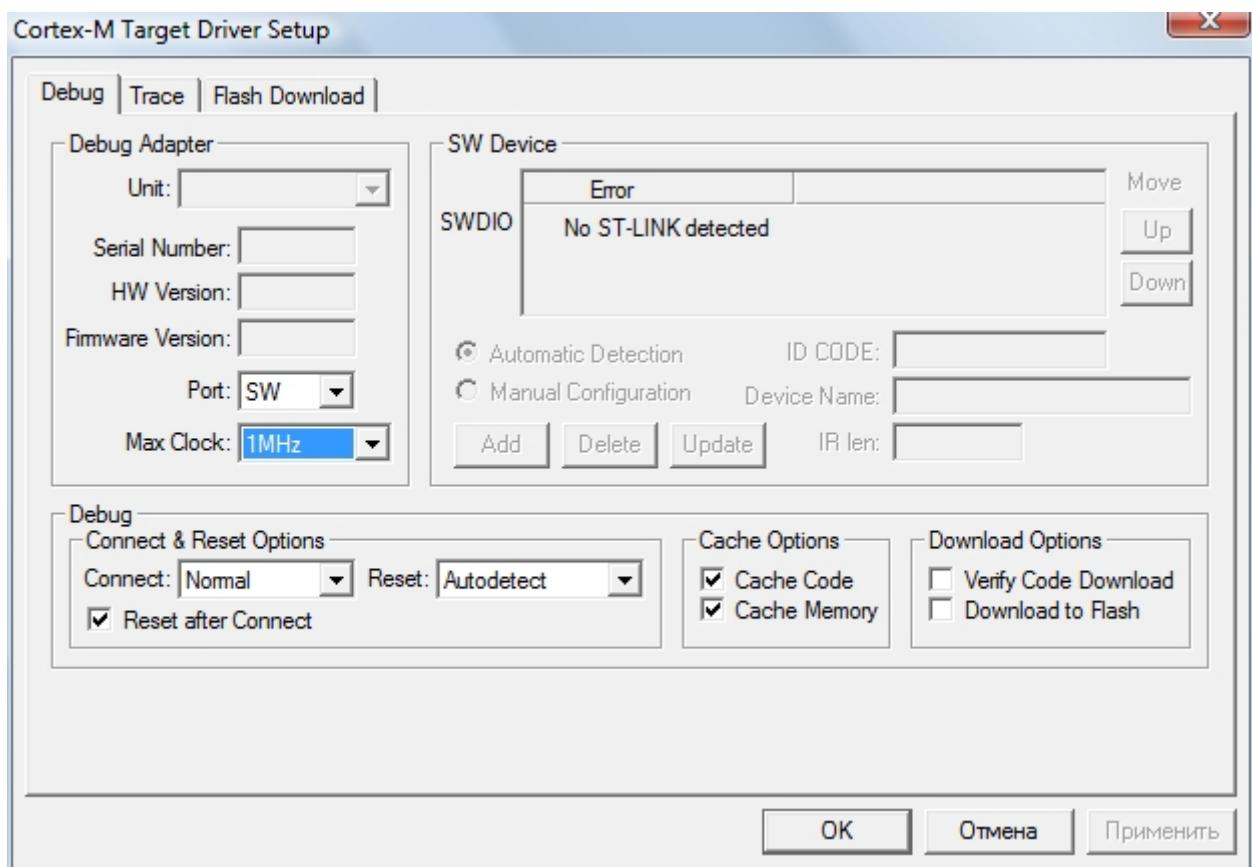
Linker Настройки линкера, который собирает объектные файлы .o, в которые компилируются каждый программный файл (модуль) по отдельности, в один готовый файл прошивки (настройки карты памяти контроллера, адреса точки входа программы и т.п.)



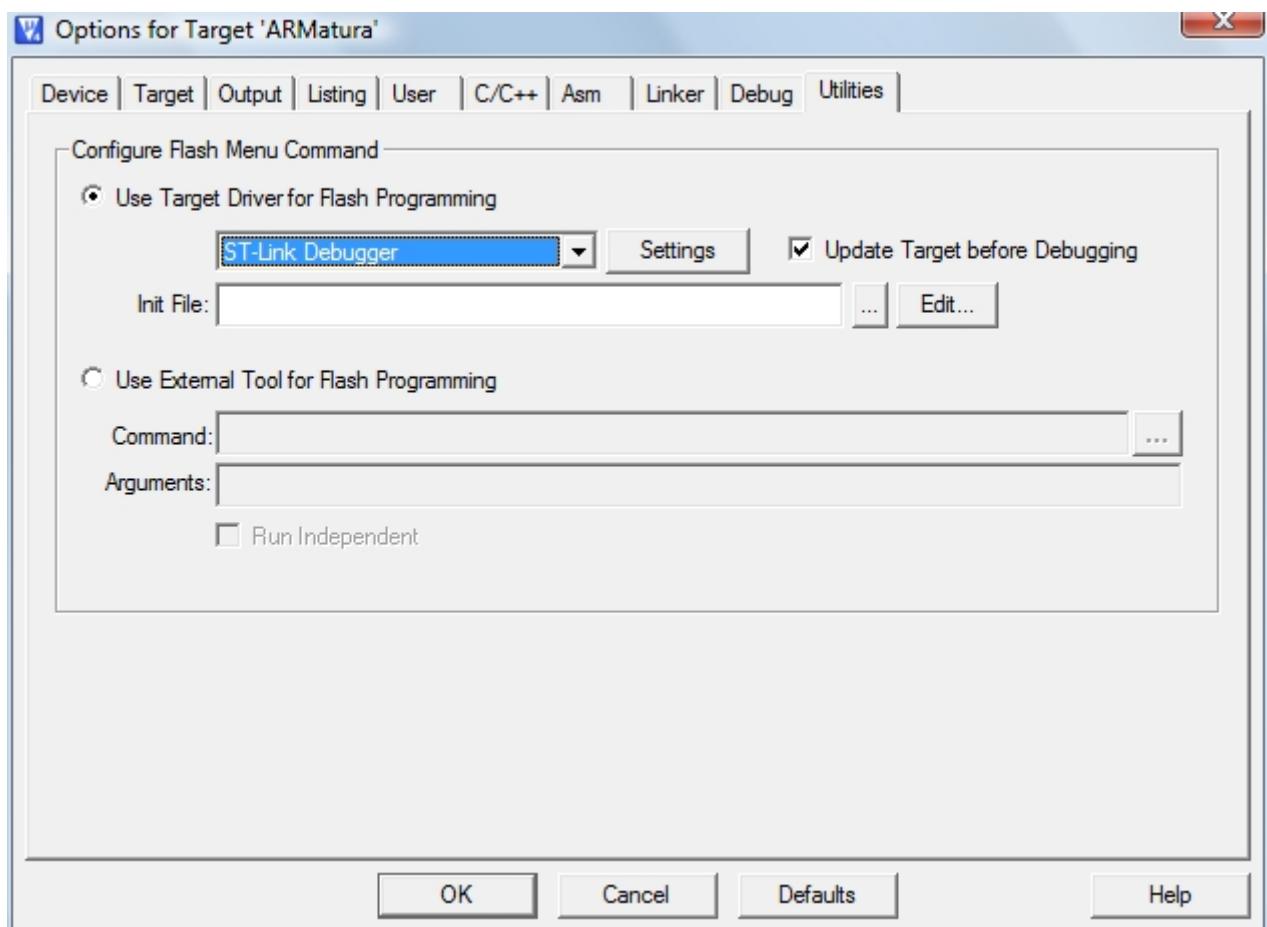
Debug Настройки отладки с помощью адаптера – STlink (внешний или встроенный в оценочную плату), JTAG.



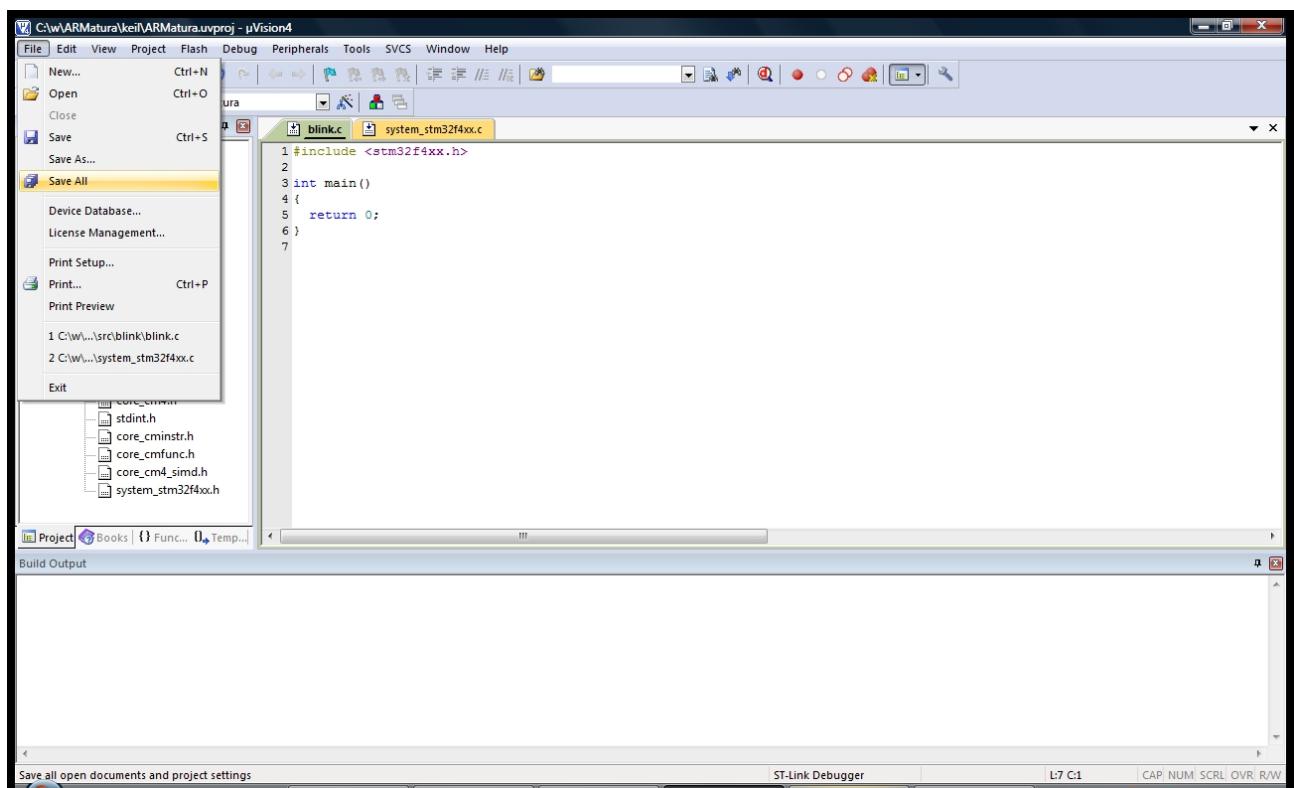
Настройки адаптера STlink – включаем режим SWD для варианта встроенного на оценочную плату, и обнаруживаем что не установили пакет поддержки STlink и драйвера.



Utilities Настраиваем в качестве программатора для прошивки тот же STlink



Сохраняем настроенный проект



10.4 Eclipse

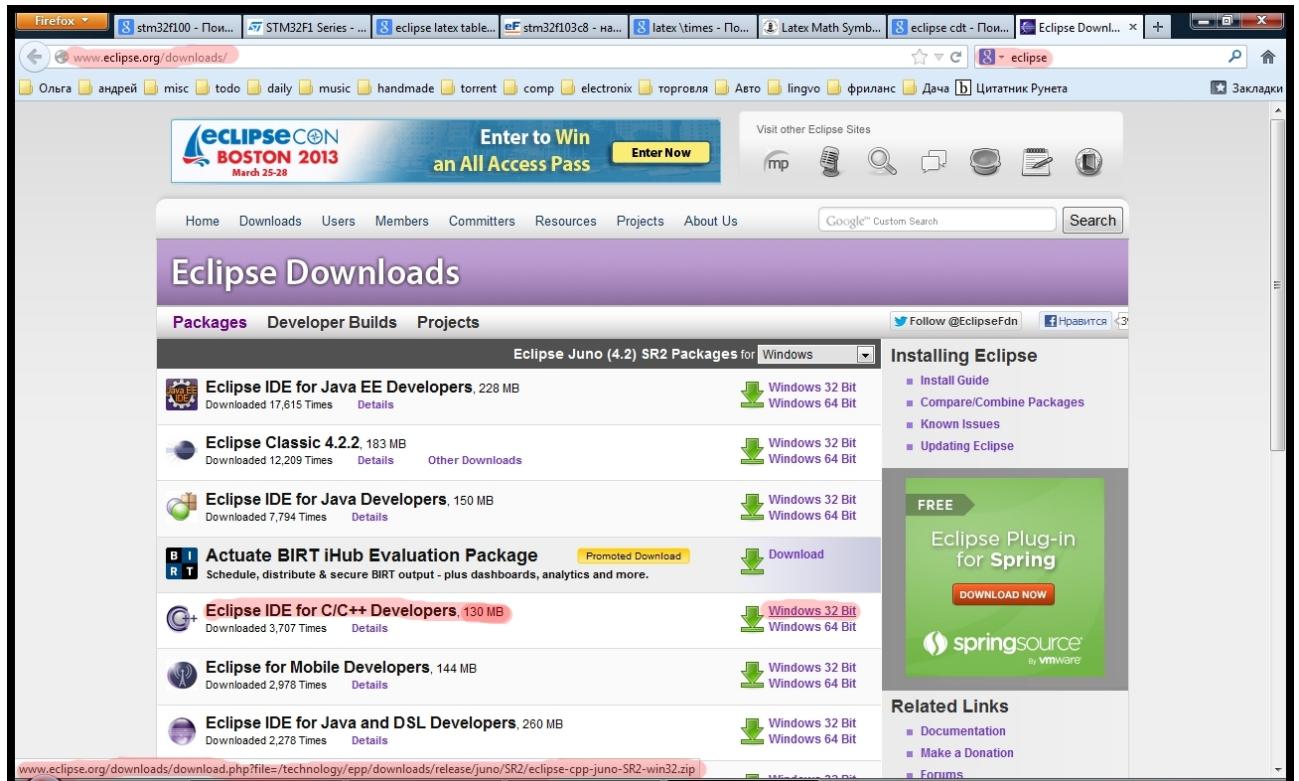
Установка

Для работы нам потребуется сборка Eclipse 10.4 в варианте

```
http://www.eclipse.org/downloads/
http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/
release/juno/SR2/eclipse-cpp-juno-SR2-win32.zip
```

Пакет поставляется в виде zip-архива, который достаточно распаковать в любое место, например в C:\Eclipse\.

После запуска нужно будет указать каталог для хранения проектов: C:\w\.

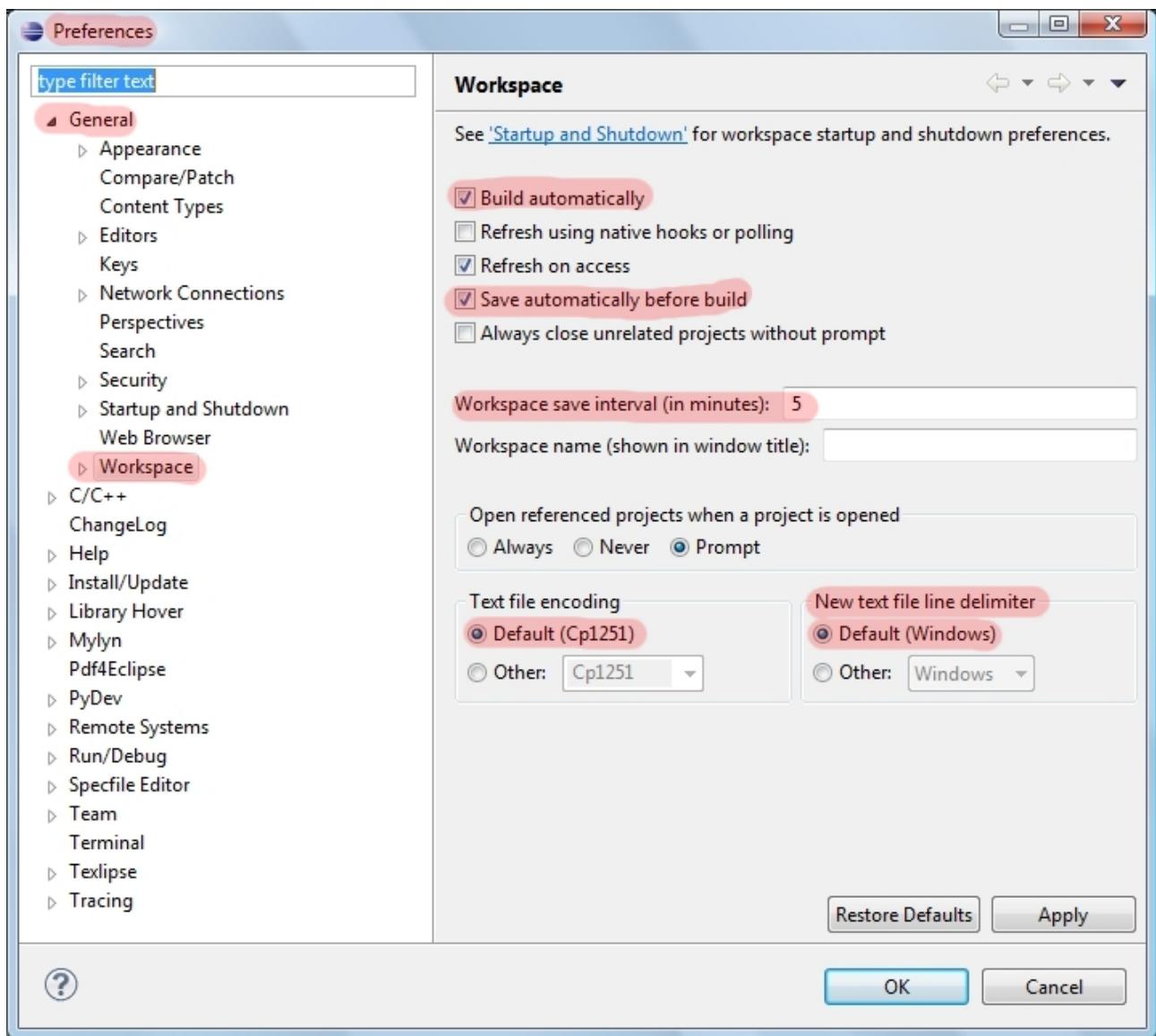


Настройка глобальных свойств

Глобальные свойства настраиваются в меню **Windows > Preferences**.

Прежде всего настроим автосохранение при запуске Build **Ctrl + B**:

- запуск сборки при сохранении файла
- автоматическое сохранение всех измененных файлов перед сборкой
- автосохранение рабочих файлов
- кодировка файлов с кириллицей (может потребоваться корректировка при импорте файлов в кодировках KOI8, UTF8)
- конец строки в стиле dos/windows: <CR><LF> (в UNIX используется только <LF>)



10.5 Code::Blocks

10.6 gVim

10.7 IAR

Глава 11

Компиляторы

11.1 GCC

11.2 KeilCC

11.3 IAR

Глава 12

Адаптеры

Адаптер = программатор = внутрисхемный отладчик = ... — аппаратное устройство, подключаемое одним концом в компьютер (обычно по USB), а другим в отлаживаемое устройство с микроконтроллером.

По подключению к устройству разделяют JTAG-адAPTERЫ, адAPTERЫ, впариваемые производителями конкретных чипов (Atmel, STmicroelectronics), и коммерческих IDE (Keil Ulink), и разнообразный самопал.

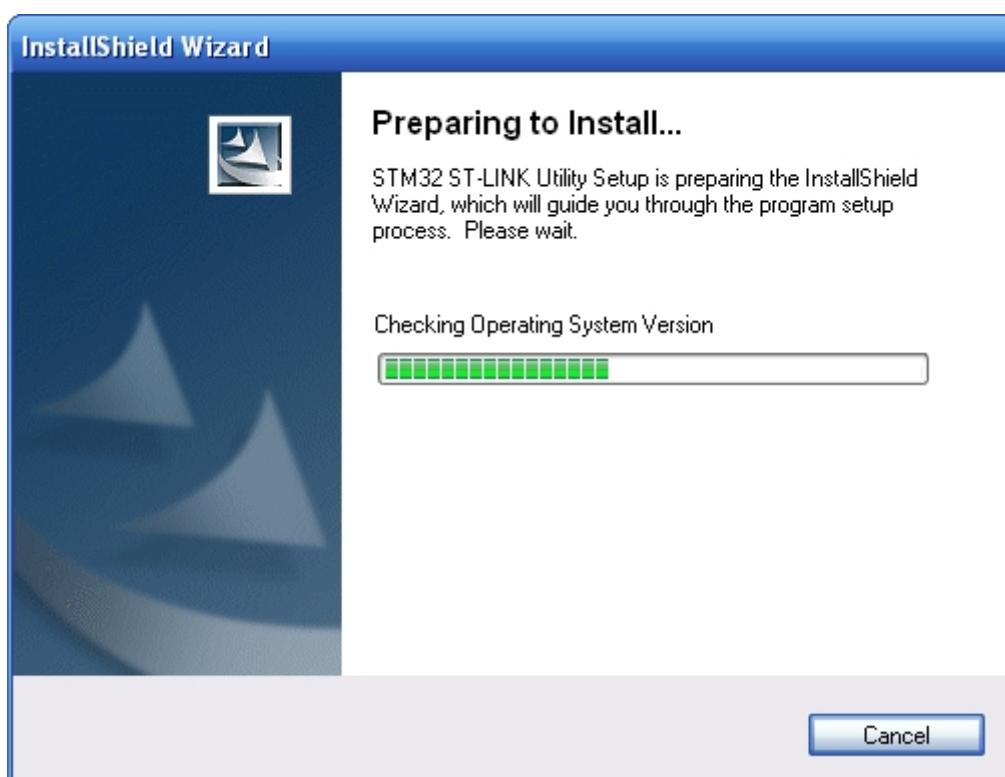
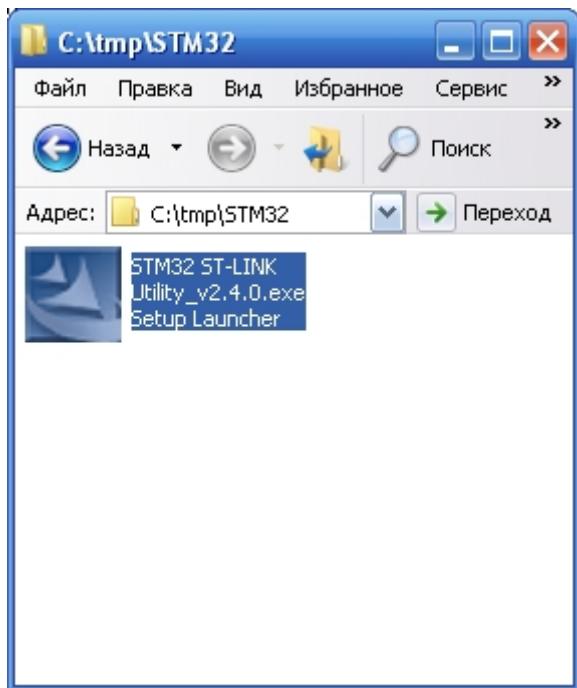
АдAPTER обеспечивает

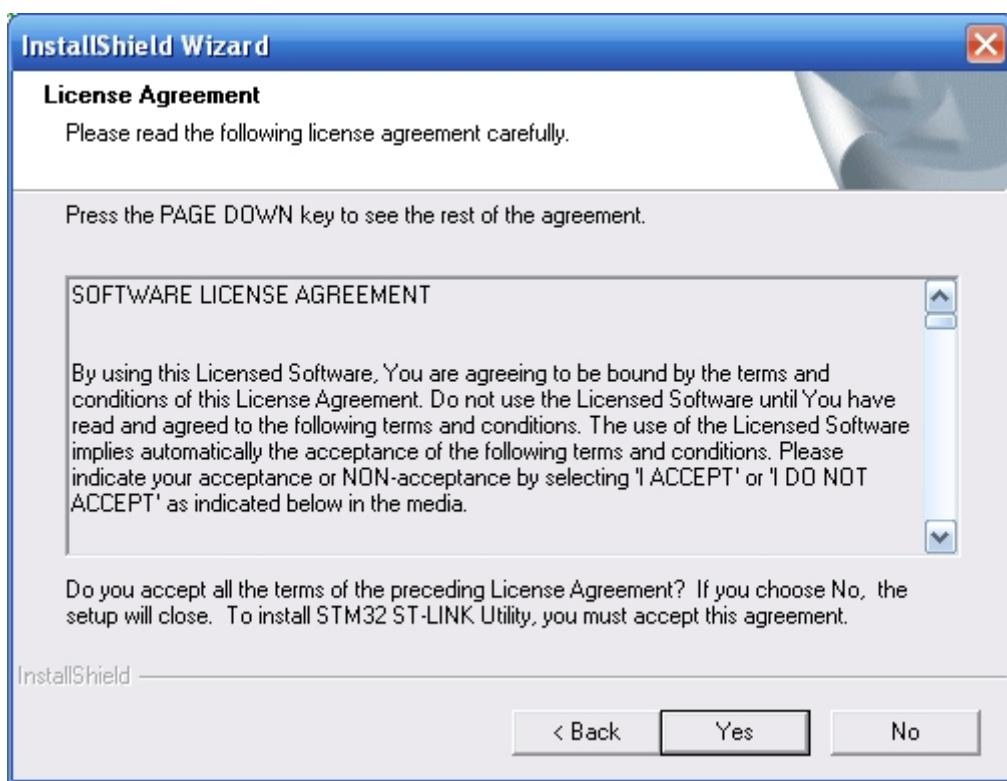
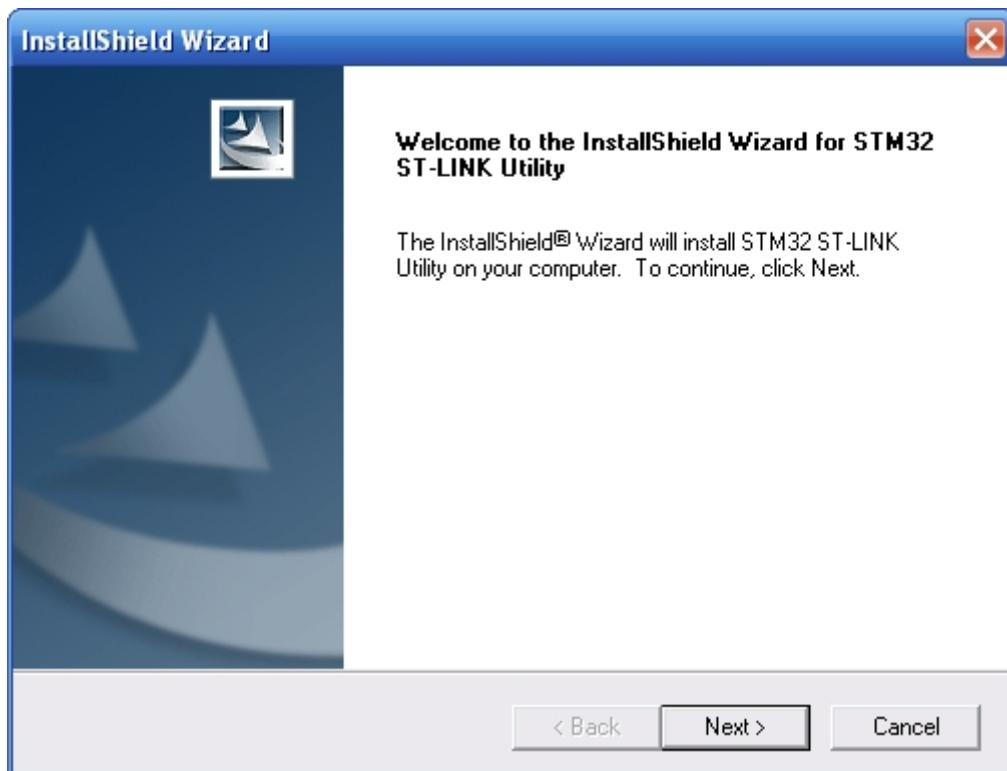
- загрузка прошивки во встроенную Flash (обеспечивается всеми адAPTERами)
- чтение/запись областей памяти
- интерактивную отладку в процессе работы программы ??

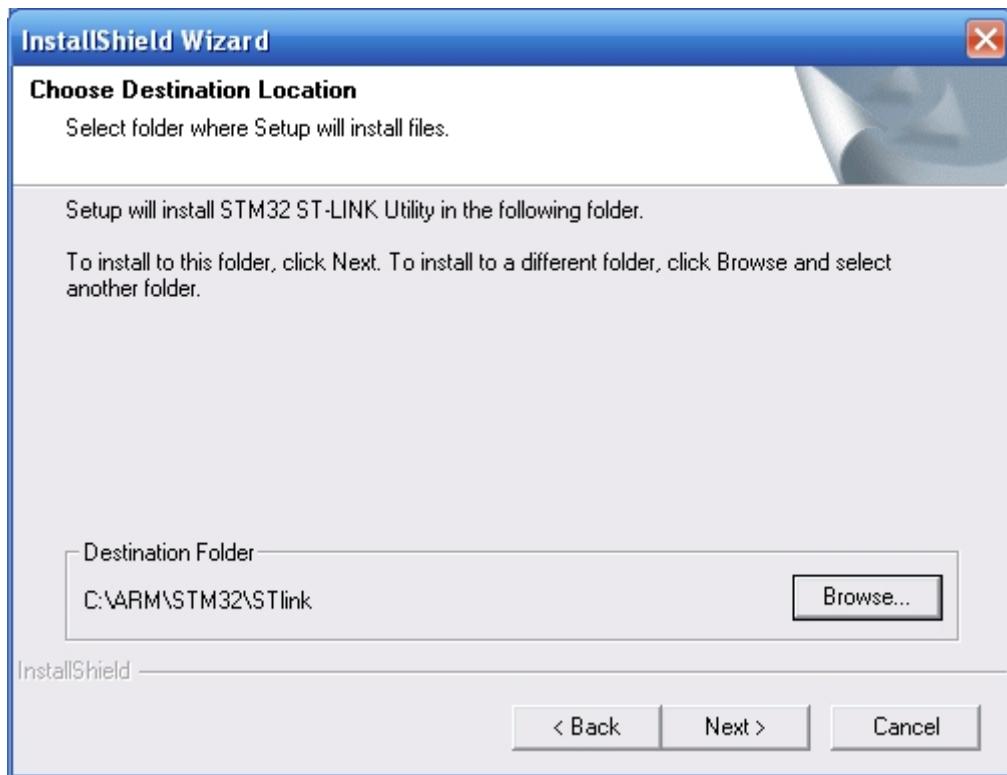
12.1 STlink

Установка ПО

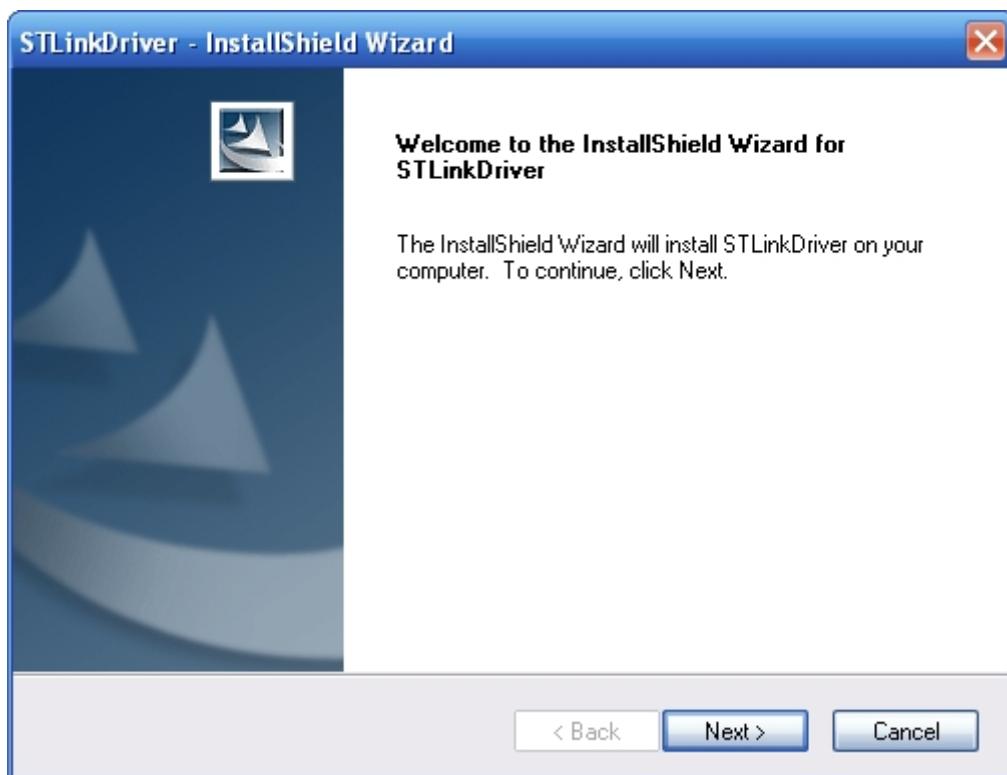
Для работы с чипами STM32 используя программаторы STlink, встроенные на демо-платы Discovery необходимо установить пакет STSW-LINK004 (STM32 ST-link Utility и драйвер), скачав его с <http://www.st.com/web/en/catalog/tools/PF258168>

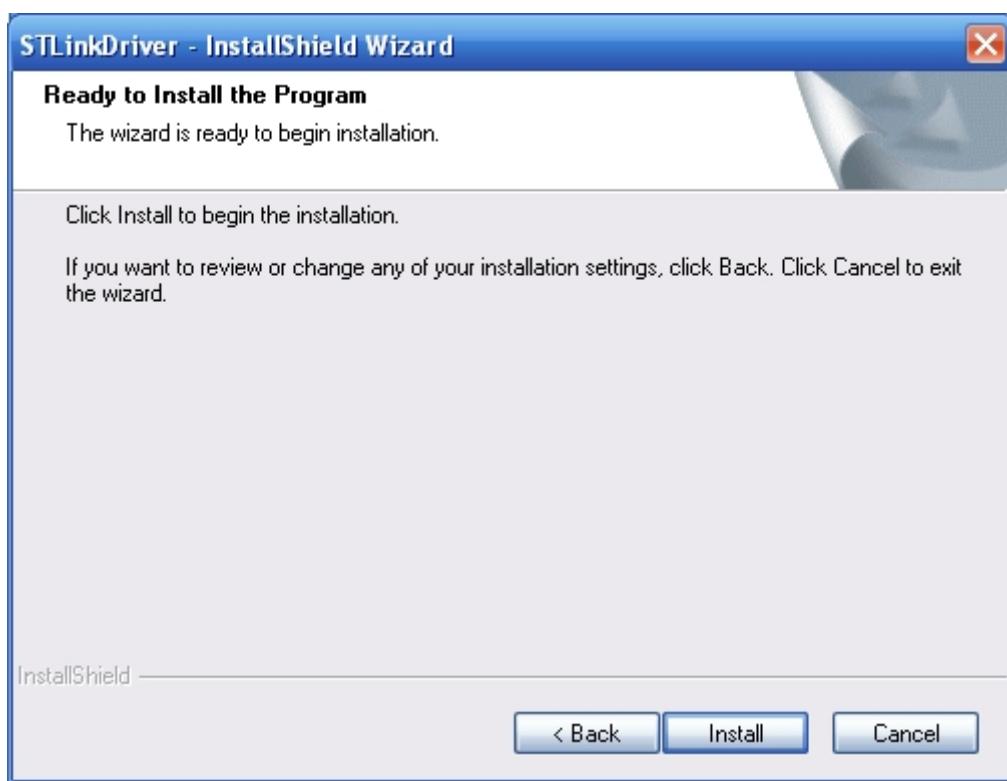
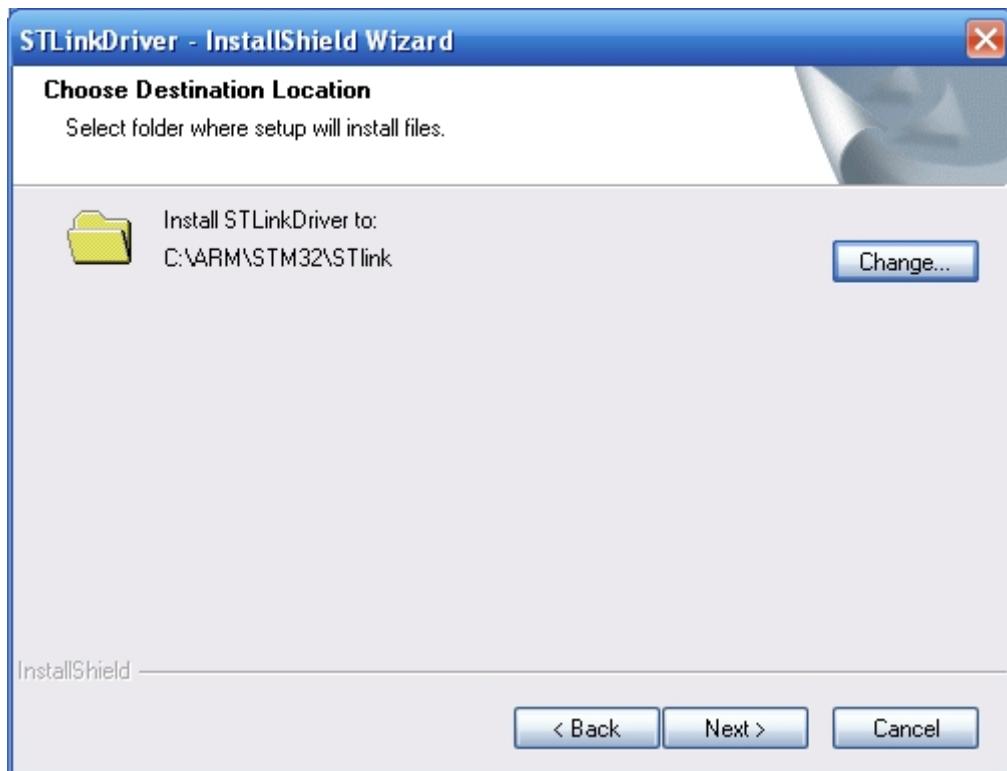


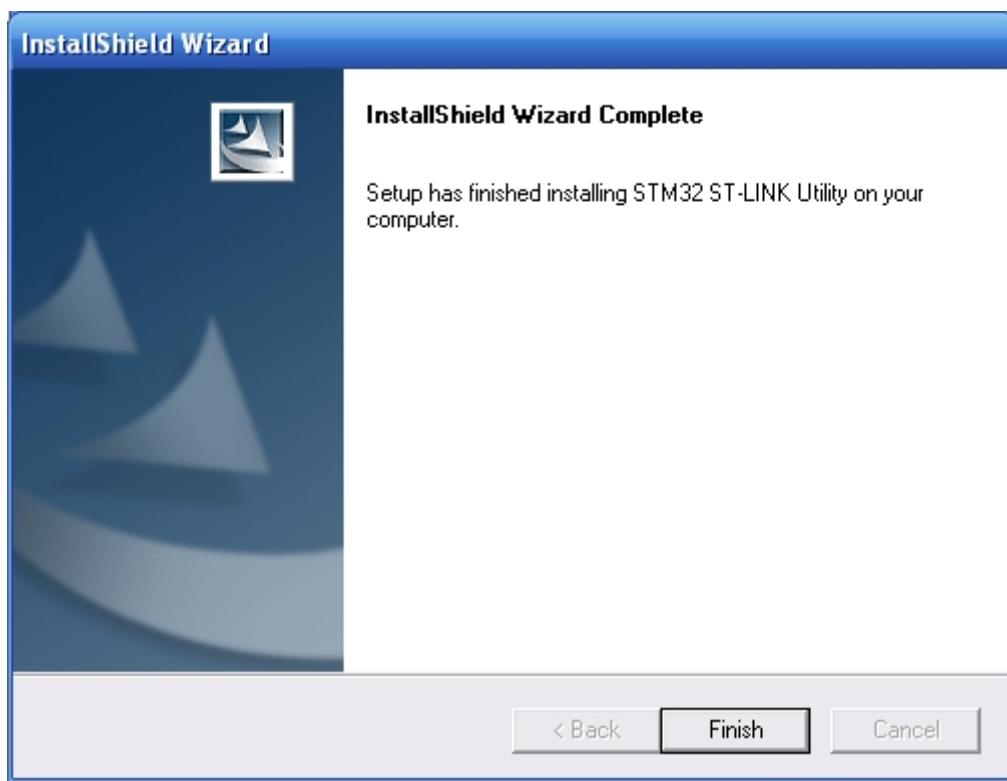
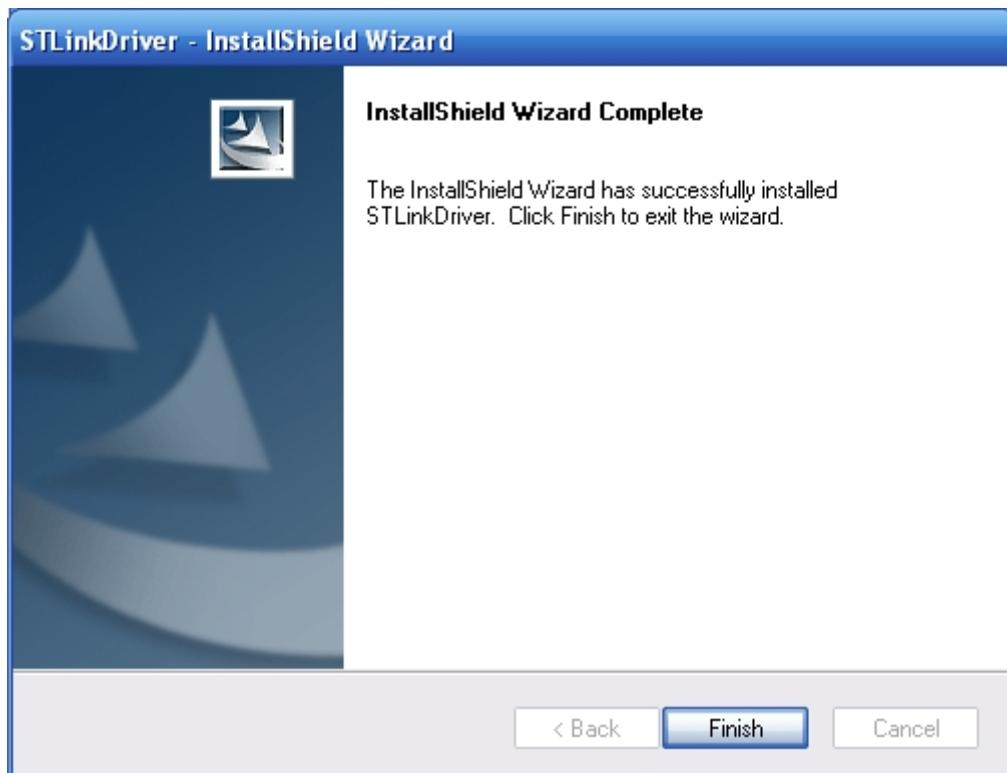




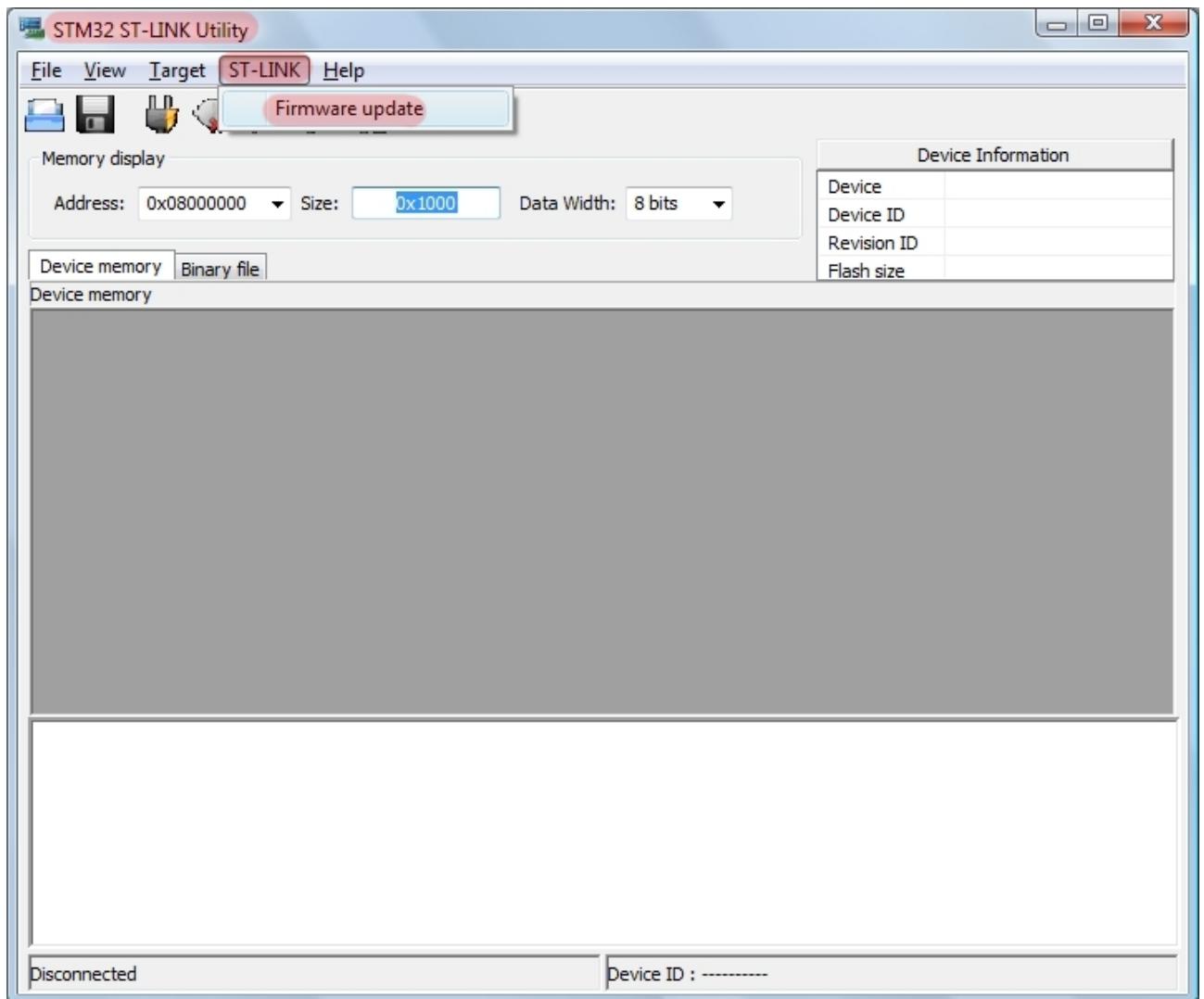
Пакет драйвера входит в пакет STSW-LINK004, и запускается автоматически



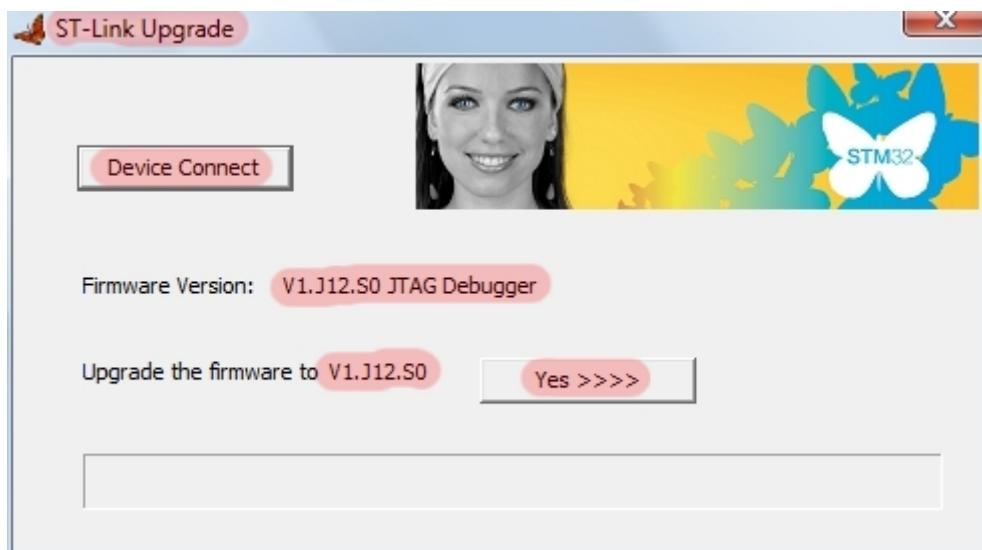




Подключаете отладочную плату, запускаете STlink12.1 Utility [Пуск > Программы > STMicroelectronics > STM32 ST-LINK Utility](#)



Запускаем **ST-LINK** > **Firmware update** > **Device Connect** и проверяем что прошивка STlink **12.1** не устарела (ПК должен быть подключен к Сети), при необходимости нажимаем **Yes** и обновляемся.



STlink gdbserver

Необходим для отладки через GDB [15](#).

12.2 JTAG

Часть V

Отладка

Глава 13

Отладка в Keil

Глава 14

STM32 SWD

Глава 15

GDB

15.1 STlink gdbserver

Глава 16

OpenOCD

Глава 17

JTAG

Часть VI

Основы языка C^{+^+}

Глава 18

Синтаксис

Глава 19

Типы данных

Глава 20

Стандартная библиотека libc

Часть VII

Wiring Framework

<http://wiring.org.co/reference/>
<https://github.com/WiringProject/Wiring>

Wiring^{VII}— OpenSource фреймворк для программирования микроконтроллеров.

Wiring^{VII} позволяет писать кроссплатформенный код для управления устройствами, подключенными к широкому диапазону различных микроконтроллерных плат, для создания разнообразных интерактивных объектов, креативных пространств или физических экспериментов. Этот фреймворк thoughtfully дизайнераами и художниками предполагая создание сообщества где начинающие с помощью экспертов со всего мира обмениваются идеями, знаниями и коллективным опытом. Существуют тысячи студентов, художников, дизайнеров, исследователей и хоббитов, которые используют Wiring^{VII} (или его частную реализацию Arduino) для исследований, прототипирования, и производства законченных устройств.

- Roadmap включает поддержку различных аппаратных ядер¹
- The current AVR8 Core supports the Wiring hardware and any hardware based on the AVR atmega processors. AVR Xmega, AVR Tiny, TI MSP430, Microchip PIC24/32 Series and STM M3 ARM Cores will be available soon².

На текущий момент существует одна единственная мощная реализация: Arduino, на базе 8-битных микроконтроллеров Atmel AVR8 (ATmega). Попытки реализации Wiring^{VII} для других МК выливаются в замкнутые на себя микросообщества, собирающиеся вокруг единственного производителя железа: например LeafLabs Maple (пара плат на базе 54), дорогой Arduino Due, ардуиноны на MSP430,..

Для ARMatura мы пойдем тем же путем — сделаем минимальную реализацию Wiring^{VII} и наплодим кучку несовместимых библиотек. Желающие могут присоединяться к разработке.

¹на самом деле только AVR8

²судя по активности визуалдизайнеров — году к 2048

Часть VIII

CMSIS STM32

В этом разделе будут описаны только *некоторые* части библиотеки CMSIS^{VIII} для STM32F, которые используются в коде ARMatura. Объяснить это очень просто — попытавшись включить в книгу полный листинг `stm32f4xx.h` получил 205 страниц листинга 😊

Поэтому подробнейше описывать все файлы библиотеки, а уж тем более еще и переводить комментарии не вижу смысла см.^{XVII}.

CMSIS^{VIII} — Cortex Microcontroller Software Interface Standard

The ARM Cortex Microcontroller Software Interface Standard — независимый от производителя HAL (hardware abstraction layer) для серии процессоров Cortex-M. CMSIS^{VIII} обеспечивает единообразный и простой программный интерфейс к процессору и его периферии, упрощает повторное использование готового кода, уменьшает усилия по изучению новых микроконтроллеров и уменьшает время запуска на рынок для новых устройств.

Создание ПО является основным фактором затрат в индустрии встраиваемых устройств. Стандартизация программных интерфейсов среди всех поставщиков чипов с ядром Cortex-M, особенно при создании новых проектов или миграции существующего ПО на новый процессор, приводит к значительному уменьшению стоимости.

CMSIS^{VIII} состоит из следующих компонент:

- CMSIS-CORE: предоставляет интерфейс к ядрам процессоров Cortex-M0, Cortex-M3, Cortex-M4, SC000, and SC300 и регистрам периферии
- CMSIS-DSP: библиотека функций ЦОС содержит более 60 функций реализованных для чисел с фиксированной точкой (дробные q7, q15, q31) и с плавающей точкой одинарной точности (32 бита)
- CMSIS-RTOS API: стандартизованный программный интерфейс для ОСРВ для управления нитями, ресурсами, и управления временем
- CMSIS-SVD: файлы System View Description XML, содержащие полное представление с точки зрения программиста всей микроконтроллерной системы включая периферию

Стандарт полностью масштабируем, и применим для всех микропроцессоров серии Cortex-M, в диапазоне от самых маленьких с 8К флеша до кристаллов с богатой коммуникационной периферией типа Ethernet и USB. (Требования к памяти со стороны CMSIS-CORE: менее 1К кода, менее 10 байт ОЗУ).

Глава 21

Установка

Для работы необходимо скачать библиотеки поддержки от ST:

BSP	STSW-STM32049	STM32F0 Discovery kit firmware package
SPL	STSW-STM32048	STM32F0xx standard peripherals library
BSP	STSW-STM32078	STM32VLDISCOVERY1 firmware package (AN3268)
SPL	STSW-STM32054	STM32F10x standard peripheral library
BSP	STSW-STM32068	STM32F4DISCOVERY3 board firmware package
SPL	STSW-STM32065	STM32F4 DSP and standard peripherals library
USB	STSW-STM32046	STM32F105/7, STM32F2 and STM32F4 USB on-the-go Host and device library

BSP **B**oard **S**upport **P**ackage

пакет поддержки конкретной демоплаты, содержит библиотеки ядра + библиотеки работы с наплатным железом, макросы описания выводов, библиотеки работы с внешними чипами

SPL **S**tandard **P**eripheral **L**ibrary

пакет содержит только библиотеки ядра и встроенной в кристалл периферии (таймеры, ЦАП,...)

Глава 22

Startup код

22.1 `startup_stm32f10x_ld_v1.s`

22.2 `startup_stm32f40xx.s`

22.3 `system_stm32f10x.c`

22.4 `system_stm32f4xx.c`

Глава 23

CMSIS

23.1 `stm32f10x.h`

23.2 `core_cm3.h`

CMSIS**VIII** библиотека поддержки ядра Cortex-M3.

23.3 `core_cm4.h`

CMSIS**VIII** библиотека поддержки ядра Cortex-M4.

23.4 `core_cmInstr.h`

CMSIS**VIII** Core Instruction Interface.

23.5 `core_cmFunc.h`

CMSIS**VIII** Core Register Access Functions.

Глава 24

BSP

24.1 STM32VLDISCOVERY1

STM32vldiscovery.c

24.2 STM32F4DISCOVERY3

STM32f4_discovery.c

Глава 25

Cortex-M4 расширение SIMD/DSP

25.1 core_cm4_simd.h

Часть IX

Ядро Cortex-Mx

Глава 26

Режимы ARM и Thumb

Глава 27

DMA

Глава 28

DSP /Cortex-M3/

Глава 29

FPU /Cortex-M4F/

Часть X

Семейство STM32F10x

Черновой перевод руководства по программированию
© ST PM0056 STM32F10x/20x/21x/L1x Programming manual
http://www.st.com/web/en/resource/technical/document/programming_manual/CD00228163.pdf

Глава 30

Об этом документе

30.1 Typographical conventions

30.2 List of abbreviations for registers

30.3 About the STM32 Cortex-M3 processor and core peripherals

System level interface

Integrated configurable debug

Cortex-M3 processor features and benefits summary

Cortex-M3 core peripherals

Глава 31

Процессор Cortex-M3

31.1 Programmers model

Processor mode and privilege levels for software execution

Stacks

Core registers

Exceptions and interrupts

Data types

The Cortex microcontroller software interface standard (CMSIS)

31.2 Memory model

Memory regions, types and attributes

Memory system ordering of memory accesses

Behavior of memory accesses

Software ordering of memory accesses

Bit-banding

Memory endianness

Synchronization primitives

Programming hints for the synchronization primitives

31.3 Exception model

Exception states

Exception types

Exception handlers

Vector table

Exception priorities

Interrupt priority grouping

Глава 32

Набор инструкций Cortex-M3

32.1 Instruction set summary

32.2 Intrinsic functions

32.3 About the instruction descriptions

Operands

Restrictions when using PC or SP

Flexible second operand

Shift operations

Address alignment

PC-relative expressions

Conditional execution

Instruction width selection

32.4 Memory access instructions

ADR

LDR and STR, immediate offset

LDR and STR, register offset

LDR and STR, unprivileged

LDR, PC-relative

LDM and STM

PUSH and POP

LDREX and STREX

CLREX

32.5 General data processing instructions

Глава 33

Периферия ядра

33.1 About the STM32 core peripherals

33.2 Memory protection unit (MPU)

MPU access permission attributes

MPU mismatch

Updating an MPU region

MPU design hints and tips

MPU type register (MPU_TYPER)

MPU control register (MPU_CR)

MPU region number register (MPU_RNR)

MPU region base address register (MPU_RBAR)

MPU region attribute and size register (MPU_RASR)

33.3 Nested vectored interrupt controller (NVIC)

The CMSIS mapping of the Cortex-M3 NVIC registers

Interrupt set-enable registers (NVIC_IERx)

Interrupt clear-enable registers (NVIC_ICERx)

Interrupt set-pending registers (NVIC_ISPRx)

Interrupt clear-pending registers (NVIC_ICPRx)

Interrupt active bit registers (NVIC_IABRx)

Interrupt priority registers (NVIC_IPRx)

Software trigger interrupt register (NVIC_STIR)

Level-sensitive and pulse interrupts

NVIC design hints and tips

NVIC register map

Глава 34

История изменений

Часть XI

Интерфейсы

Глава 35

USB

Глава 36

UART

Глава 37

SPI

Глава 38

I2C

Глава 39

CAN

Часть XII

Операционные системы ОСРВ

Глава 40

Keil RTX

Глава 41

FreeRTOS

Глава 42

eCos

Глава 43

Linux

подробно рассмотрен в отдельном разделе XVI

Часть XIII

Стек TCP/IP

Глава 44

Ethernet

Глава 45

PPP

Часть XIV

Типовые применения

Глава 46

GPS

46.1 Протокол NMEA 0183

NMEA 0183— текстовый протокол связи морского и навигационного оборудования.

http://www8.garmin.com/support/pdf/NMEA_0183.pdf
http://www8.garmin.com/support/pdf/NMEA_0183.pdf
http://ru.wikipedia.org/wiki/NMEA_0183
<http://www.robosoft.info/ru/technologies/knowledgebase/nmea0183>

Датум *WGS⁸⁴*

http://ru.wikipedia.org/wiki/WGS_84

Большинство GPS приемников отдают данные по NMEA 0183, но вместо режима последовательного порта 4800 8N1 прописанного в протоколе, могут использовать другие режимы, характерные для портов RS232 (9600, 115200). Координаты передаются в датуме *WGS⁸⁴*.

Формат сообщения NMEA¹:

\\$[A-Z]5(,<data>)+*[0-9A-F]2<CR><LF>

- символ \$
- 5-буквенный идентификатор сообщения. Первые две буквы — идентификатор источника сообщения, следующие три буквы — идентификатор формата сообщения
 - GPGGA — данные о последнем определении местоположения
 - GPGLL — координаты, широта/долгота
 - GPGSA — DOP (GPS) и активные спутники
 - GPGSV — наблюдаемые спутники
 - GPWPL — параметры заданной точки
 - GPBOD — азимут одной точки относительно другой
 - GPRMB — рекомендуемый минимум навигационных данных для достижения заданной точки

¹регулярные выражения http://en.wikipedia.org/wiki/Regular_expression

- GPRMC — рекомендуемый минимум навигационных данных: информацию о времени, местоположении (*WGS⁸⁴*), курсе и скорости, передаваемые навигационным GPS приёмником. Контрольная сумма обязательна для этого сообщения, интервалы передачи не должны превышать 2 секунды.
- GPRTE — маршруты
- HCHDG — данные от компаса
- блоки данных, разделённых запятыми. Пустые данные, не заданные в середине строки, оставляют запятыю. Пустые данные в конце строки могут отбрасываться вместе с запятой.
- символ *
- двузначное hex число — контрольная XOR-сумма всех байт в строке между \$ и *.
- конец строки <CR><LF> = 0x0D 0x0A = \r\n

46.2 \$GPRMC — рекомендуемый минимум навигационных данных

\$GPRMC, hhmmss.ss, [AV], GGMM.MM, [NS], gggmm.mm, [EW], v.v, b.b, ddmmyy, x.x, n, m*hh<CR><LF>

- GP – приём сигналов GPS, GN – ГЛОНАСС
- RMC – Recommended Minimum sentence C
- hhmmss.ss – время фиксации местоположения по времени UTC
- A – данные достоверны, V – данные недостоверны
- GGMM.MM – широта, 2 цифры градусов, 2 цифры целых минут, точка и дробная часть минут;
- N/S – северная/южная широта
- gggmm.mm – долгота, 3 цифры градусов, 2 цифры целых минут, точка и дробная часть минут;
- E/W – восточная/западная долгота
- v.v – горизонтальная составляющая скорости в узлах
- b.b – путевой угол (направление скорости) в градусах: 0° север, 90° восток, 180° юг, 270° запад
- ddmmyy – дата
- x.x – магнитное склонение в градусах (часто отсутствует)
- n – направление магнитного склонения: для получения магнитного курса магнитное склонение необходимо вычесть (E) или прибавить (W) к истинному курсу
- m – индикатор режима: A – автономный, D – дифференциальный, E – аппроксимация, N – недостоверные данные (часто отсутствует, данное поле включая запятую отсутствует в старых версиях NMEA)

46.3 Системы координат (датум)

<http://ne-grusti.narod.ru/Glossary/datums.html>

Системы координат (datums) можно разделить на геоцентрические и топоцентрические.

В геоцентрической системе размеры эллипсоида, ориентация и положение его центра выбираются следующим образом:

- объем эллипсоида предполагается равным объему геоида;
- большая полуось эллипсоида лежит в плоскости экватора геоида;
- малая полуось направлена по оси вращения Земли;
- среднеквадратичное отклонение поверхности эллипсоида от поверхности геоида
- минимально по всей территории земного шара.

WGS⁷² и сменившая ее *WGS⁸⁴*, а также российская *SGS⁸⁵* являются геоцентрическими системами координат на эллипсоидах *WGS72*, *GRS80* и *SGS85* соответственно. В системе GPS/NAVSTAR используется *WGS⁸⁴*, а в системе GLONASS — *SGS⁸⁵*.

Топоцентрическая (национальная) система координат появляется так: вы берете некоторый эллипсоид и располагаете его таким образом, чтобы для заданной территории среднеквадратичное отклонение поверхности эллипсоида от поверхности геоида было минимальным. При этом остальная часть мира вас не интересует: отклонения на другой стороне Земли может быть сколь угодно велико.

В России используются несколько геодезических систем координат: Пулково¹⁹⁴² Пулково 1942 г., 1963 г. и 1991 г. Система координат 1963 г. используется военными и ее параметры преобразования засекречены. Обычно мы пользуемся картами, составленными в системе Пулково¹⁹⁴². Она базируется на эллипсоиде Красовского.

Параметры преобразования для Пулково¹⁹⁴²:

Преобразование Bursa-Wolf (Position Vector Transformation)

<http://ne-grusti.narod.ru/Glossary/transformations.html#pvt>

направление	dX	dY	dZ	rX	rY	rZ	M	источник
<i>WGS84</i> → 1942	-27.0	+135.0	+84.5	0.0	0.0	0.554	-0.2263	Data+
1942 → <i>WGS84</i>	+25.0	-141.0	-78.5	0.0	0.35	0.736	0.0	Stefan A. Voser

Преобразование Молоденского

<http://ne-grusti.narod.ru/Glossary/transformations.html#molodensky>

направление	dX	dY	dZ	da	df	источник
1942 → <i>WGS84</i>	+28.0	-130.0	-95.0	-108.0	+0.00480795	Vladimir Zh. Boston-PC Forum
1942 → <i>WGS84</i>	+28.0	-130.0	-95.0	-108.0	+0.00480795	MADTRAN, Peter H. Dana
1942 → <i>WGS84</i>	+24.0	-123.0	-94.0	-108.0	+0.00480795	Stefan A. Voser

46.4 Методы преобразования систем координат

<http://ne-grusti.narod.ru/Glossary/transformations.html>

Часто требуется перейти от одной системы координат (datum) к другой. Например, вы хотите данные с GPS в системе координат WGS^{84} наложить на карту в системе координат Пулково¹⁹⁴².

Произодить преобразование проще (математически), если сначала перевести координаты из географических (широта и долгота) в прямоугольные (XYZ).

Чтобы получить точные прямоугольные координаты точки на поверхности Земли, необходимо знать не только широту и долготу, но и ее высоту над поверхностью эллипсоида. GPS показывает высоту над эллипсоидом WGS^{84} . Можно вычислить прямоугольные координаты только по широте и долготе, считая, что точка лежит на поверхности эллипса, но при этом точность понизится.

Высоту точки на другими эллипсоядами получить сложнее. Обычно мы знаем высоту в национальной системе высот. За нулевую высоту, как правило, принимается среднее значение уровня моря в определенной точке побережья по результатам многолетних наблюдений. Высоты в этой системе измеряются геодезическими методами относительно поверхности геоида. Поэтому, чтобы узнать высоту точки над эллипсоядом, нужно знать возвышение геоида над эллипсоядом в данном месте, которое трудно вычислить с хорошей точностью. Существуют различные математические модели геоида для разных территорий и для всего земного шара, которые постоянно уточняются.

На заре спутниковой геодезии, когда взаимосвязи между системами координат не были четко определены, а данные исследований были не очень точны, применяли простой сдвиг начала координат dX , dY , dZ для перехода от одной системы координат к другой. Это предполагало, что направления осей двух эллипсоядов параллельны (что во многих случаях не соответствует действительности). Для работ на небольшой территории погрешности, вносимые этим предположением, были меньше, чем точность самих данных. Однако, по мере накопления и уточнения данных и повышения точности измерений, стало очевидно, что преобразование по трем параметрам не подходит для больших территорий и глобального использования, если требуется максимальная точность и единый набор параметров преобразования.

Простейший метод — сдвиг центра координат прямоугольной системы, предполагая, что оси исходной и целевой систем координат параллельны.

$$\begin{pmatrix} X_B \\ Y_B \\ Z_B \end{pmatrix} = \begin{pmatrix} X_A \\ Y_A \\ Z_A \end{pmatrix} + \begin{pmatrix} dX \\ dY \\ dZ \end{pmatrix} \quad (46.1)$$

Молоденский разработал формулы для применения параметров сдвига к географическим координатам.

$$\Delta\varphi'' = \frac{206265}{\rho} (-dX \sin \varphi \cos \lambda - dY \sin \varphi \sin \lambda + dZ \cos \varphi + [a\Delta f + f\Delta a] \sin 2\varphi) \quad (46.2)$$

$$\Delta\lambda'' = \frac{206265}{\nabla \cos \varphi} (-dX \sin \lambda + dY \cos \lambda) \quad (46.3)$$

$$\Delta h = dX \cos \varphi \cos \lambda + dY \cos \varphi \sin \lambda + dZ \sin \varphi + (a\Delta f + f\Delta a) \sin^2 \varphi - \Delta a \quad (46.4)$$

здесь dX , dY , dZ — сдвиг по осям, м; a и f — большая полуось и сжатие исходного эллипса; da и df — разности между большой полуосью и сжатием исходного эллипса и целевого эллипса.

Повышенная точность достигается преобразованием Хелмерта с семью параметрами. Есть две его разновидности, различающиеся присвоением знака для параметров поворота.

1. Position Vector Transformation (Bursa-Wolf)

$$\begin{pmatrix} X_B \\ Y_B \\ Z_B \end{pmatrix} = M \cdot \begin{pmatrix} 1 & -Rz & +Ry \\ +Rz & 1 & -Rx \\ -Ry & +Rx & 1 \end{pmatrix} \begin{pmatrix} X_A \\ Y_A \\ Z_A \end{pmatrix} + \begin{pmatrix} dX \\ dY \\ dZ \end{pmatrix} \quad (46.5)$$

2. Coordinate Frame Transformation

$$\begin{pmatrix} X_B \\ Y_B \\ Z_B \end{pmatrix} = M \cdot \begin{pmatrix} 1 & +Rz & -Ry \\ -Rz & 1 & +Rx \\ +Ry & -Rx & 1 \end{pmatrix} \begin{pmatrix} X_A \\ Y_A \\ Z_A \end{pmatrix} + \begin{pmatrix} dX \\ dY \\ dZ \end{pmatrix} \quad (46.6)$$

Здесь M — это масштаб (scale), параметры берутся из описания системы координат.

46.5 Геоид и эллипсоиды

<http://ne-grusti.narod.ru/Glossary/ellipsoid-geoid.html>

Геоид — фигура сложной формы, образованная поверхностью уровня вод Мирового океана, продолженной под материками. Эта поверхность во всех точках перпендикулярна (нормальна) вектору силы тяжести. Отвес направлен перпендикулярно поверхности геоида, а не к центру Земли! Это связано с тем, что плотность Земли распределена неравномерно.

Эллипсоид — тело, полученное вращением эллипса вокруг его малой оси. Размеры подбирают так, чтобы среднеквадратичное отклонение от поверхности геоида было минимально либо по всей поверхности Земли, либо для заданной территории.

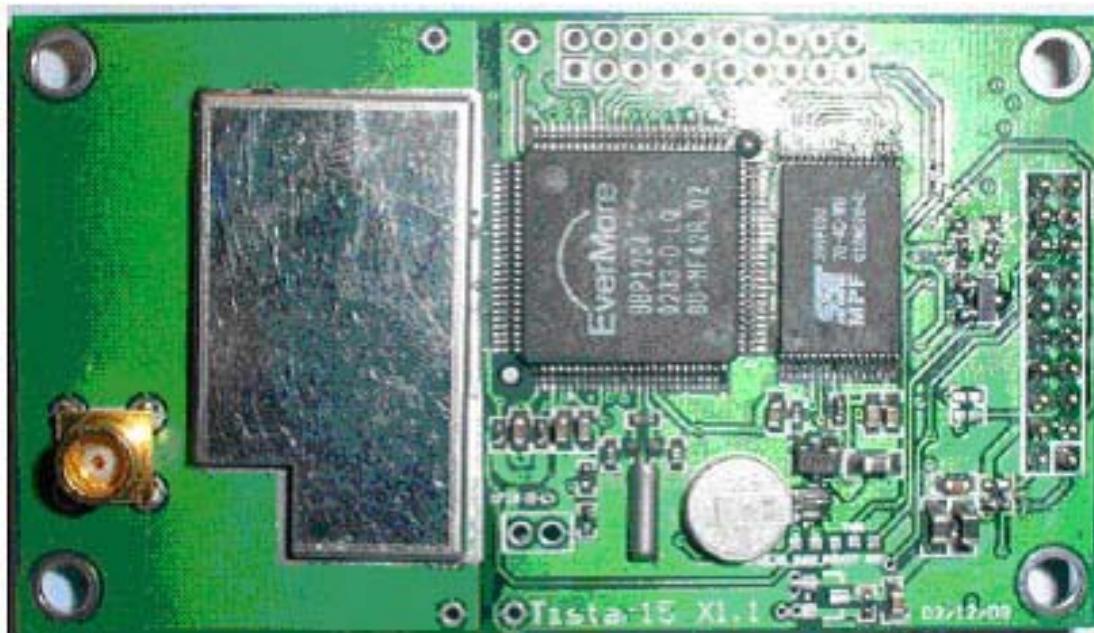
Параметры некоторых эллипсоидов

эллипсоид	использование	большая полуось a, м	малая полуось b, м	сжатие $f = (a - b)/a$
Красовского (1940)	Россия и др. Пулково ¹⁹⁴²	6378245	6356863	1/298.3
GRS80	международный <i>WGS⁸⁴</i>	6378137	6356752.31425	1/298.25722356
SGS85	ГЛОНАСС SGS85			
Бесселя	СССР до 1942 г.			

В таблицах эллипсоидов часто указывается не полярное сжатие f , а обратная величина $1/f$, например, для эллипсоида Красовского $1/f = 298.3$.

Отклонения эллипсоида Красовского от геоида на территории СНГ не превышают 150 м.

46.6 Tistar15



Цена: 250 руб. <http://www.voltmaster.ru/cgi-bin/qquery.pl?id=127000056703&group=700000>

46.7 WISMO228



Цена: 1070 руб. <http://www.voltmaster.ru/cgi-bin/qquery.pl?id=127000881980&group=700000>

Глава 47

GSM

47.1 WISMO228



Цена: 1070 руб. <http://www.voltmaster.ru/cgi-bin/qwery.pl?id=127000881980&group=700000>

Часть XV

VCS: управление версиями

Для реальных проектов, даже состоящих из пары файлов, всегда¹ используются системы контроля версий (VCS) — некая база данных, в которой хранятся исходные тексты и бинарные ресурсы (таблицы, битмапы, сторонние библиотеки в бинарниках, аудиофайлы...).

Эта база (репозиторий проекта) хранит *полную историю* всех изменений, выполненных в файлах проекта, и позволяет получить исходные тексты на любой момент разработки, вести несколько альтернативных веток разработки, выяснить кто когда и что делал, и кто что и когда сломал.

<http://ru.wikipedia.org/wiki/VCS>

Система управления версиями (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).

Ситуация, в которой электронный документ за время своего существования претерпевает ряд изменений, достаточно типична. При этом часто бывает важно иметь не только последнюю версию, но и несколько предыдущих. В простейшем случае можно просто хранить несколько вариантов документа, нумеруя их соответствующим образом. Такой способ неэффективен (приходится хранить несколько практически идентичных копий), требует повышенного внимания и дисциплины и часто ведёт к ошибкам, поэтому были разработаны средства для автоматизации этой работы.

Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создается локальная копия документа, т. н. «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время.

Часто бывает, что над одним проектом одновременно работают несколько человек. Если два человека изменяют один и тот же файл, то один из них может случайно отменить изменения, сделанные другим. Системы управления версиями отслеживают такие конфликты и предлагают средства их решения. Большинство систем может автоматически объединить (слиять) изменения, сделанные разными разработчиками. Однако такое автоматическое объединение изменений, обычно, возможно только для текстовых файлов и при условии, что изменились разные (непересекающиеся) части этого файла.

Часто выполнить слияние невозможно ни в автоматическом, ни в ручном режиме, например, если формат файла неизвестен или слишком сложен. Некоторые системы управления версиями дают возможность заблокировать файл в хранилище. Блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению ра-

¹ кроме индивидуумов, контуженных Паскалем

бочей копии файла (например, средствами файловой системы) и обеспечивает, таким образом, исключительный доступ только тому пользователю, который работает с документом.

Распределённые системы управления версиями также известны как англ. Distributed Version Control System, DVCS. Такие системы используют распределённую модель вместо традиционной клиент-серверной. Они, в общем случае, не нуждаются в централизованном хранилище: вся история изменения документов хранится на каждом компьютере, в локальном хранилище, и при необходимости отдельные фрагменты истории локального хранилища синхронизируются с аналогичным хранилищем на другом компьютере. В некоторых таких системах локальное хранилище располагается непосредственно в каталогах рабочей копии.

Когда пользователь такой системы выполняет обычные действия, такие как извлечение определённой версии документа, создание новой версии и тому подобное, он работает со своей локальной копией хранилища. По мере внесения изменений, хранилища, принадлежащие разным разработчикам, начинают различаться, и возникает необходимость в их синхронизации.

Описанная модель логически близка созданию отдельной ветки для каждого разработчика в классической системе управления версиями (в некоторых распределённых системах перед началом работы с локальным хранилищем нужно создать новую ветвь). Отличие состоит в том, что до момента синхронизации другие разработчики этой ветви не видят. Пока разработчик изменяет только свою ветвь, его работа не влияет на других участников проекта и наоборот. По завершении обособленной части работы, внесённые в ветви изменения сливают с основной (общей) ветвью. Как при слиянии ветвей, так и при синхронизации разных хранилищ возможны конфликты версий. На этот случай во всех системах предусмотрены те или иные методы обнаружения и разрешения конфликтов слияния.

С точки зрения пользователя распределённая система отличается необходимостью создавать локальный репозиторий и наличием в командном языке двух дополнительных команд: команды получения репозитория от удалённого компьютера (*pull*) и передачи своего репозитория на удалённый компьютер (*push*). Первая команда выполняет слияние изменений удалённого и локального репозиториев с помещением результата в локальный репозиторий; вторая — наоборот, выполняет слияние изменений двух репозиториев с помещением результата в удалённый репозиторий. Как правило, команды слияния в распределённых системах позволяют выбрать, какие наборы изменений будут передаваться в другой репозиторий или извлекаться из него, исправлять конфликты слияния непосредственно в ходе операции или после её неудачного завершения, повторять или возобновлять неоконченное слияние. Обычно передача своих изменений в чужой репозиторий (*push*) завершается удачно только при условии отсутствия конфликтов. Если конфликты возникают, пользователь должен сначала слить версии в своём репозитории (выполнить *pull*), и лишь затем передавать их другим.

Обычно рекомендуется организовывать работу с системой так, чтобы пользователи всегда или преимущественно выполняли слияние у себя в репозитории. То есть, в отличие от централизованных систем, где пользователи передают свои изменения на центральный сервер, когда считают нужным, в распределённых системах более естественным является порядок, когда слияние версий инициирует тот, кому нужно получить его результат (например, разработчик, управляющий сборкой дистрибутива для передачи его на конечное тестирование).

Основные преимущества распределённых систем — их гибкость и значительно большая (по сравнению с централизованными системами) автономия отдельного рабочего места. Каждый компьютер разработчика является, фактически, самостоятельным и полнофункциональным сервером. При этом каждый разработчик может вести работу независимо,

так, как ему удобно, изменяя и сохраняя промежуточные версии документов, пользуясь всеми возможностями системы (в том числе доступом к истории изменений) даже в отсутствие сетевого соединения. Связь с сервером или другими разработчиками требуется исключительно для проведения синхронизации.

К недостаткам распределённых систем можно отнести увеличение требуемого объёма дисковой памяти: на каждом компьютере приходится хранить полную историю версий, тогда как в централизованной системе на компьютере разработчика обычно хранится лишь рабочая копия.

Можно выделить следующие типичные ситуации, в которых использование распределённой системы даёт заметные преимущества:

- Периодическая синхронизация нескольких компьютеров под управлением одного разработчика (рабочего компьютера, домашнего компьютера, ноутбука и так далее). Использование распределённой системы избавляет от необходимости выделять один из компьютеров в качестве сервера, а синхронизация выполняется по необходимости, обычно при «пересадке» разработчика с одного устройства на другое.
- Совместная работа над проектом небольшой территориально распределённой группы разработчиков без выделения общих ресурсов. Как и в предыдущем случае, реализуется схема работы без главного сервера, а актуальность репозиториев поддерживается периодическими синхронизациями по схеме «каждый с каждым».
- Крупный распределённый проект, участники которого могут долгое время работать каждый над своей частью, при этом не имеют постоянного подключения к сети. Такой проект может использовать централизованный сервер, с которым синхронизируются копии всех его участников. Возможны и более сложные варианты — например, с созданием групп для работы по отдельным направлениям внутри более крупного проекта. При этом могут быть выделены отдельные «групповые» серверы для синхронизации работы групп, тогда процесс окончательного слияния изменения становится древовидным: сначала отдельные разработчики синхронизируют изменения на групповых серверах, затем обновлённые репозитории групп синхронизируются с главным сервером. Возможна работа и без «групповых» серверов, тогда разработчики одной группы синхронизируют изменения между собой, после чего любой из них (например, руководитель группы) передаёт изменения на центральный сервер.

Проект ARMatura хранится в публичном репозитории [git@github.com:ponyatov/ARMatura.git](https://github.com/ponyatov/ARMatura.git) используя систему контроля версий Git.

Глава 48

Git

```
http://git-scm.com/book/ru/
http://rogerdudler.github.com/git-guide/index.ru.html
http://githowto.com/ru
http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/ru/
http://ru.wikipedia.org/wiki/Git
http://freesource.info/wiki/RuslanHihin/gitusermanual?v=b7s&
http://freesource.info/wiki/RuslanHihin/GitTutorial1?v=3pr#ruslanhihin/gitusermanual
```

48.1 Установка

48.2 Создание репозитория из проекта

```
C:\Documents and Settings\user> cd \w\
C:\w> mkdir YourProject
C:\w> cd YourProject
C:\w\ARMatura> git init
```

48.3 Работа с удалёнными репозиториями

Чтобы иметь возможность совместной работы над каким-либо Git-проектом, необходимо знать, как управлять удалёнными репозиториями. Удалённые репозитории — это модификации проекта, которые хранятся в интернете, в локальной сети, или на локальных или съемных дисках. Их может быть несколько, каждый из которых, как правило, доступен только на чтение, либо на чтение и запись. Совместная работа включает в себя управление удалёнными репозиториями и помещение (push) и получение (pull) данных в и из них тогда, когда нужно обменяться результатами работы. Управление удалёнными репозиториями включает добавление удалённых репозиториев, удаление тех из них, которые больше не действуют, управление различными удалёнными ветками и определение их как отслеживаемых (tracked) или нет,...

Отображение удалённых репозиториев

Чтобы просмотреть, какие удалённые серверы у вас уже настроены, следует выполнить команду `git remote`. Она перечисляет список имён-сокращений для всех уже указанных удалённых дескрипторов. Если вы склонировали ваш репозиторий, у вас должен отобразиться, по крайней мере, `origin` — это имя по умолчанию, которое Git присваивает серверу, с которого вы склонировали:

```
C:\Documents and Settings\user> cd \w\ARMatura
C:\w\ARMatura> git remote -v
origin  git@github.com:ponyatov/ARMatura.git (fetch)
origin  git@github.com:ponyatov/ARMatura.git (push)
```

Добавление удалённых репозиториев

Чтобы добавить новый удалённый Git-репозиторий под именем-сокращением, к которому будет проще обращаться, выполните `git remote add [сокращение] [url]`:

48.4 Репа на флешке

Иметь репу на флешке может понадобится:

- для работы в оффлайне
- в качестве резервной копии
- для передачи коллеге по палате, чтобы совместно заниматься вашим проектом

```
C:\Documents and Settings\user> H:
H:\> mkdir ARMatura_C
H:\> cd ARMatura_C
H:\ARMatura_C> git init
Initialized empty Git repository in H:/ARMatura_C/.git/
H:\ARMatura_C>git remote -v

H:\ARMatura_C>git remote add local C:\w\ARMatura

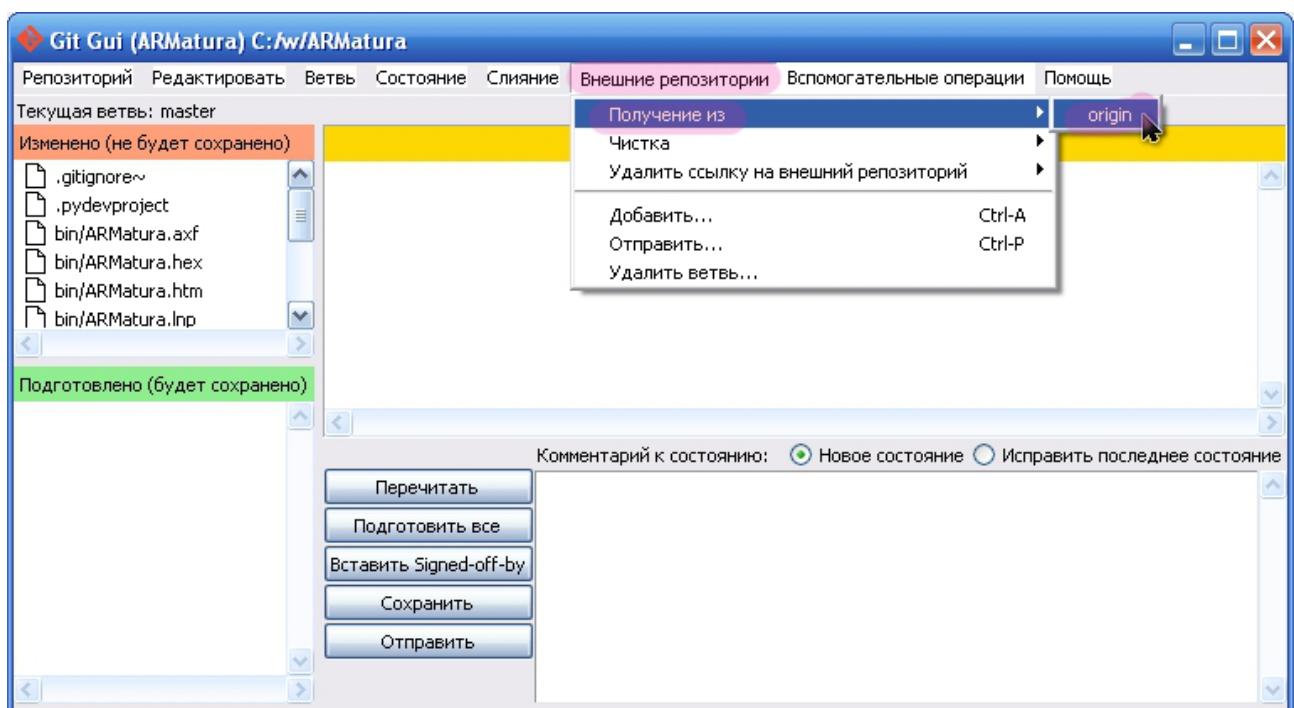
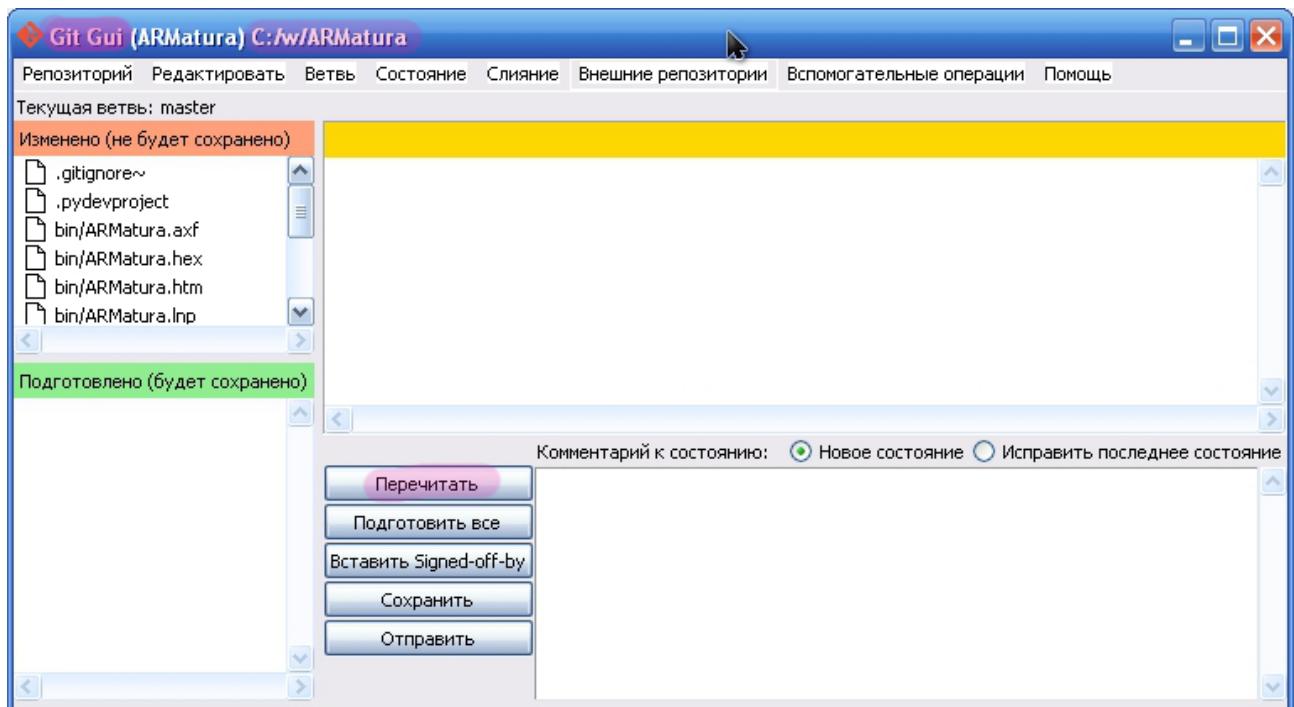
H:\ARMatura_C>git remote add github git@github.com:ponyatov/ARMatura.git

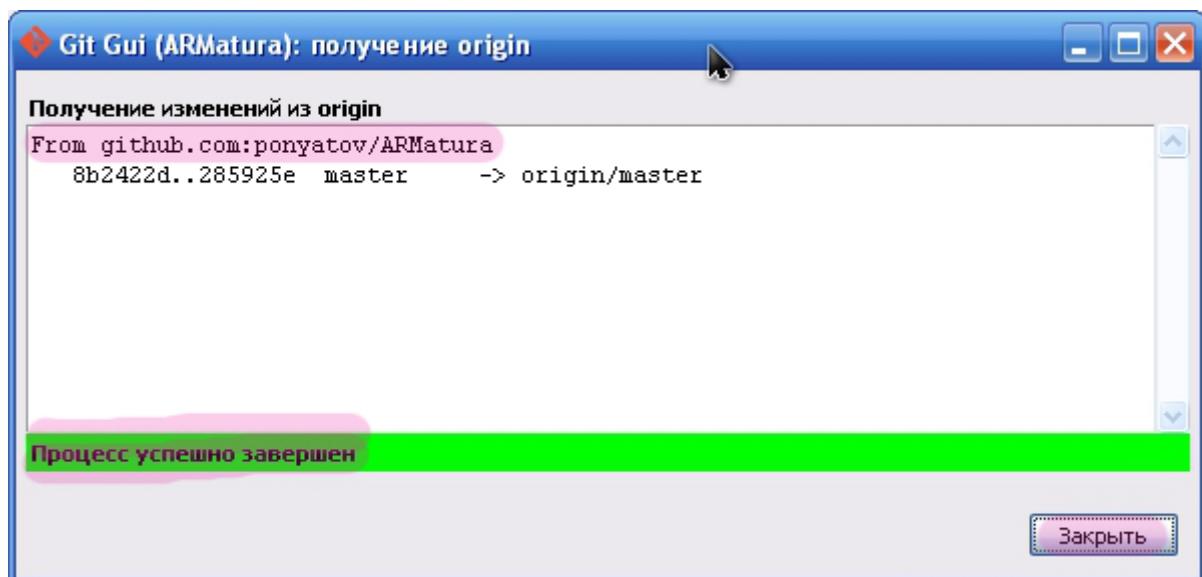
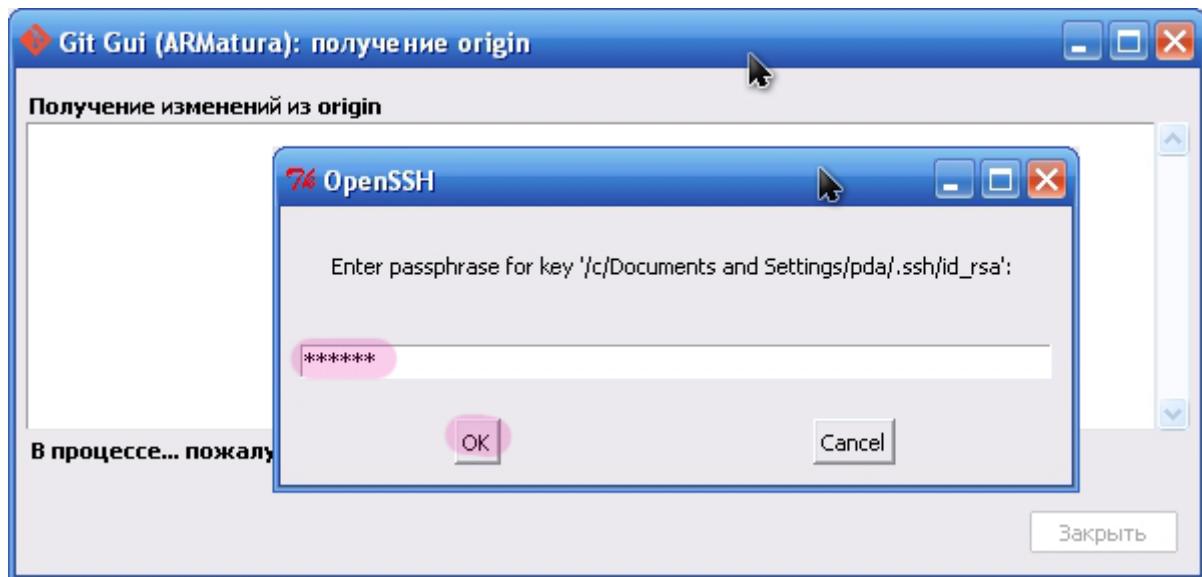
H:\ARMatura_C>git remote -v
github  git@github.com:ponyatov/ARMatura.git (fetch)
github  git@github.com:ponyatov/ARMatura.git (push)
local   C:\w\ARMatura (fetch)
local   C:\w\ARMatura (push)

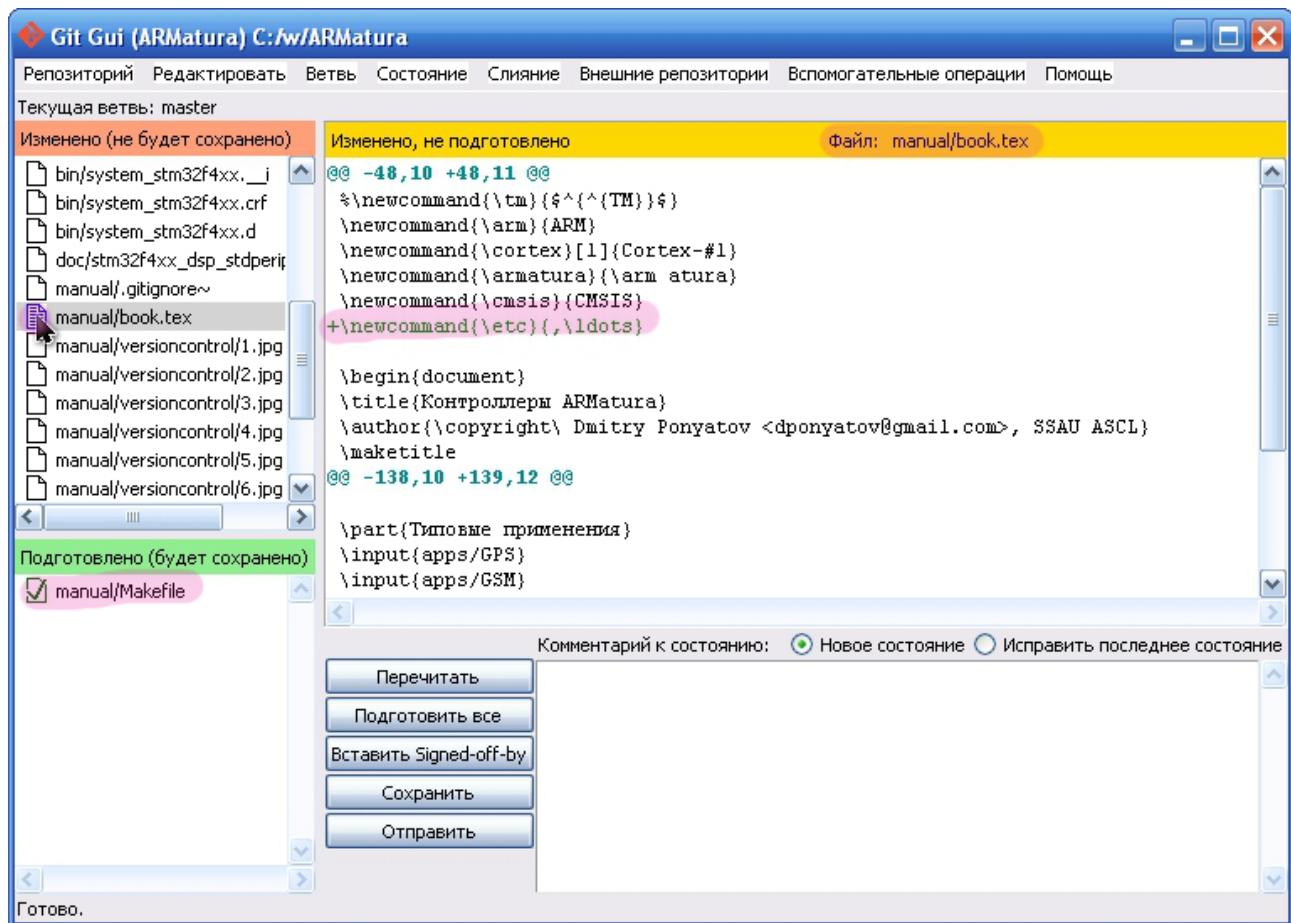
H:\ARMatura_C>git pull local master
remote: Counting objects: 612, done.
remote: Compressing objects: 100% (377/377), done.
remote: Total 612 (delta 234), reused 528 (delta 192)
Receiving objects: 100% (612/612), 46.39 MiB | 5.89 MiB/s, done.
Resolving deltas: 100% (234/234), done.
From C:\w\ARMatura
 * branch                  master      -> FETCH_HEAD
```

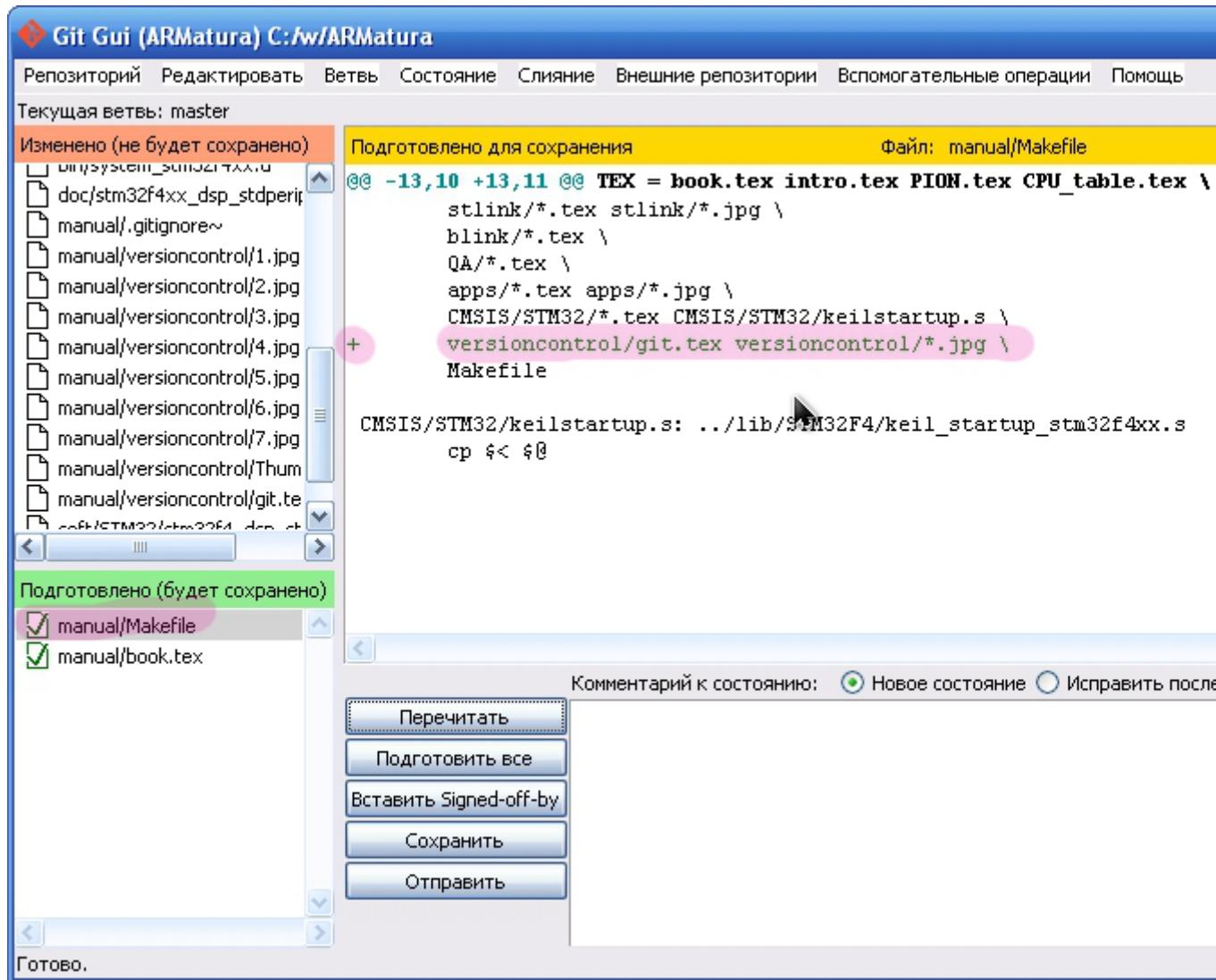
```
H:\ARMatura_C>git push -u local master
Branch master set up to track remote branch master from local.
Everything up-to-date
```

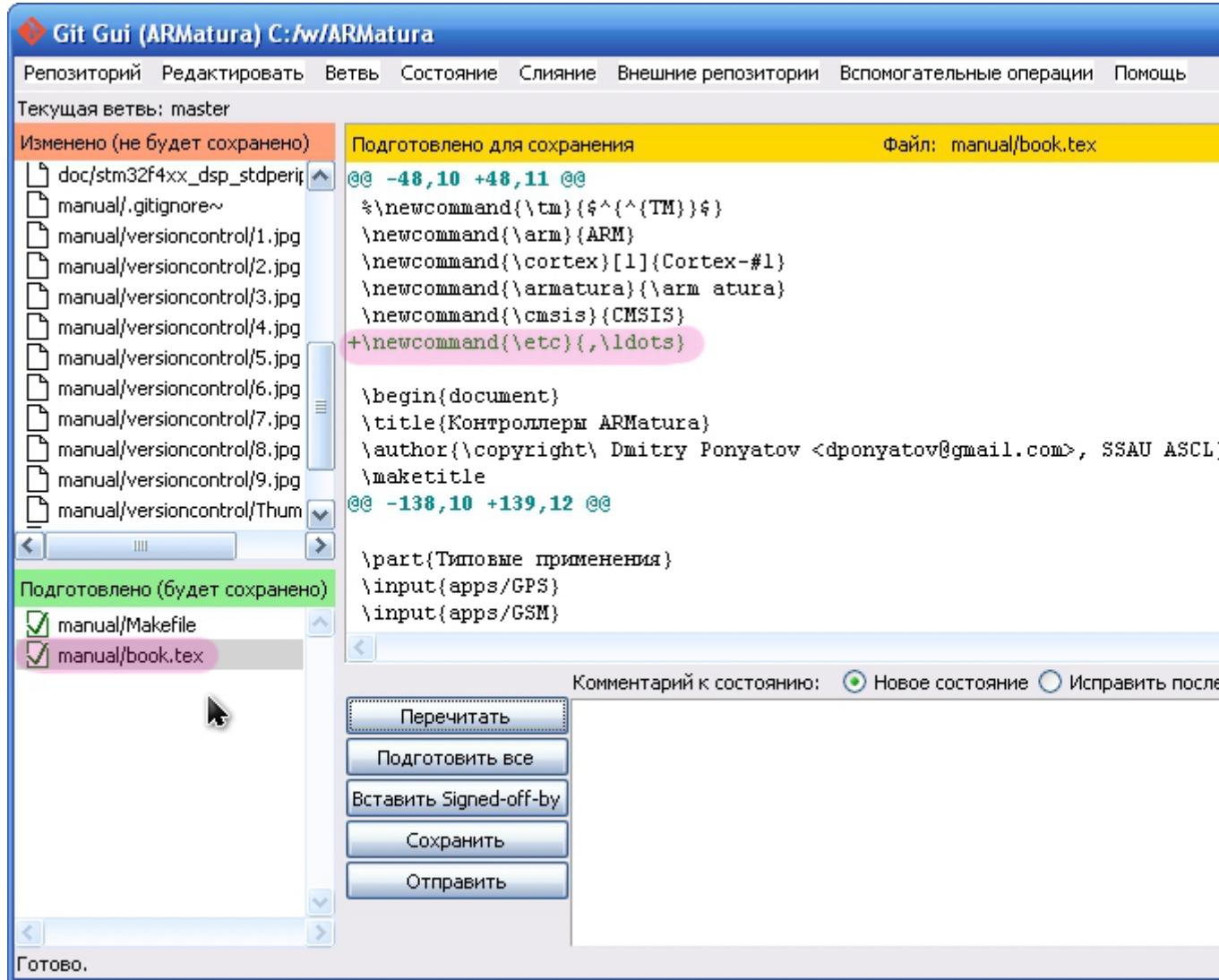
48.5 Работа с проектом github://ARMatura

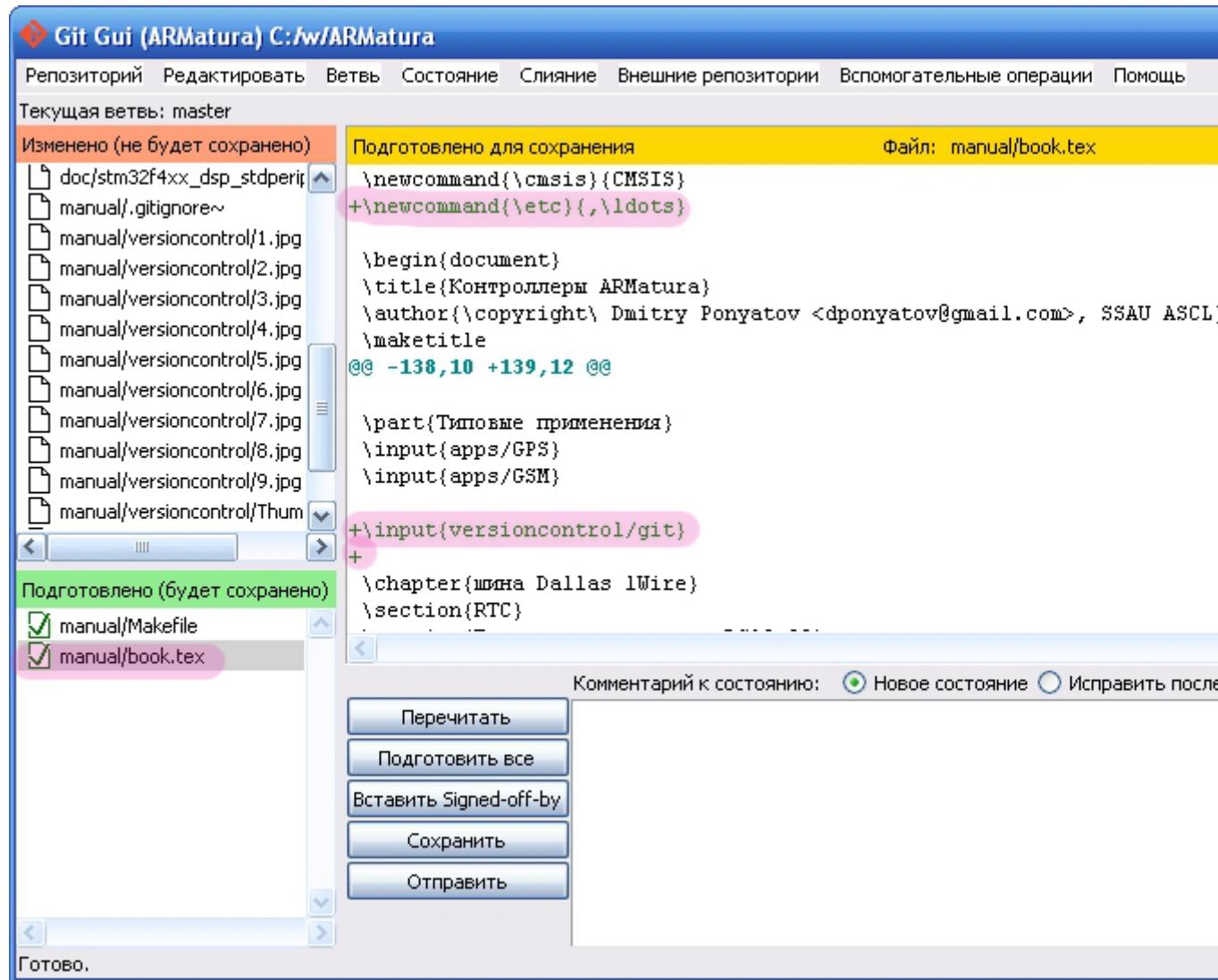


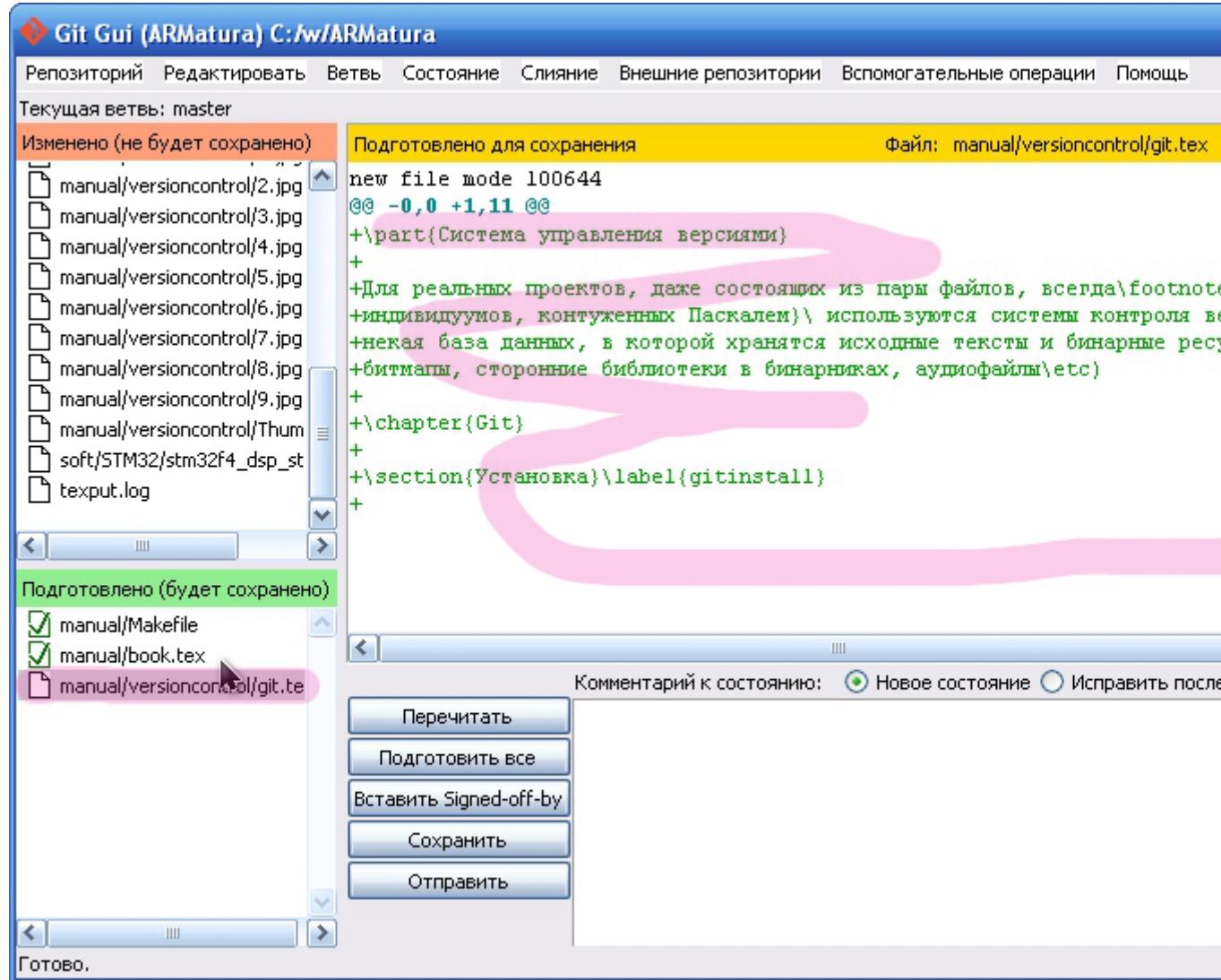












Глава 49

SubVersion

Глава 50

CVS

Глава 51

Mercurial

Глава 52

Bazaar

Глава 53

шина Dallas 1Wire

53.1 RTC

53.2 Датчики температуры DS18x20

Часть XVI

Встраиваемый Linux

Часть XVII

QA

QA00: Где перевод пунктов меню Если вы собираетесь заниматься разработкой embedded ПО, знание английского на уровне пользования online словарями и Google Translate обязательно. Кому это не нравится, могут и дальше ждать перевод datasheetов и programmer's manualов на снимаемые с производства компоненты — перевод как раз появляются к этому времени.

Смысла в переводе оригинальной документации и уж тем более комментариев в исходных кодах библиотек практически нет — объем работы возрастает в десятки раз, количество неточностей перевода на порядок, а практическая отдача наоборот стремиться к нулю.

QA01: Мне нужно подробное пошаговое руководство со скриншотами По мере возможности я добавляю в книгу скриншоты там где описывать все подробно слишком нудно.

QA02: Почему все так сумбурно и поверхностно Книга пополняется по мере того как я сам разбираюсь с разработкой под ARM. Естественно здесь не будет переводов мануалов по каждому пакету разработки, библиотеке, ядру Linux. Каждый раздел требует отдельной книги, некоторые — целой серии¹. По каждой теме существует как минимум один отдельный толстенный мануал.

В каждом разделе я привожу ссылки на те ресурсы, которыми сам пользовался при разборке с каждой конкретной темой.

Пользуйтесь Гуглом

По сумбурности очень просто — книга требует вычитки, чистки, испытания на студенте, и повторить несколько десятков раз пока не будут возникать тонны вопросов.

Если хотите можете присоединяться к наполнению книги — см. Git⁴⁸, работу с github⁴⁸, делайте форк проекта целиком или только /manual и присылайте pull requestы.

¹Ага, щаз, буду я тут 163 Мб исходников переводить и описывать каждый файл ☺

Часть XVIII

Приложения

Глава 54

Сводная таблица процессоров

STM32F >> core >> crypto >> model >> pins >> flash >> package >> n

core

0	Cortex-M0
1	Cortex-M3
3	Cortex-M4
4	Cortex-M4F с расширениями FPU (плавающая точка) и SIMD для приложений цифровой обработки сигналов

crypto

0	Cortex-M3	только CRC
1		Crypto расширение с функционалом вычислений MD5, SHA,..
2	Cortex-M4	CRC
3		Crypto
5		

model

0	
1	
3	включена поддержка CAN, USB, SDIO
7	

package

T LQFP

pins

C	48
R	64
V	100
I	176

flash

8	64K
B	128K
G	1M
I	2M

	ядро Cortex-	MHz	Flash	SRAM	корпус LQFP	GPIO	ST family
STM32F100C4T6B	M3	24	16K	4K	48		LD
STM32F100RBT	M3	24	128K	8K	100		MD
STM32F103C8T	M3	72	64K	20K	48	37	MD
STM32F103CBT	M3	72	128K	20K	48	37	MD
STM32F103RBT	M3	72	128K	20K	64	51	MD
STM32F103VBT	M3	72	128K	20K	100	80	MD
STM32F407VGT	M4F	168	1M	192K	144		
STM32F407IGT	M4F	168	1M	192K	176		
STM32F427IIT	M4F	168	2M	256K	176		
STM32F051R8T	M0	48	64K	8K	64	55	
	CAN	USB	UART	I2C	SPI	ADC	DAC
STM32F100C4T6B	○	○	2		1		
STM32F100RBT	○	○	1				
STM32F103C8T	●	●	3	2	2	2x12b (10ch)	
STM32F103C8T	●	●	3	2	2	2x12b (10ch)	
STM32F103RBT	●	●	3	2	2	2x12b (16ch)	
STM32F103VBT	●	●	3	2	2	2x12b (16ch)	
STM32F407VGT	2	2	6				
STM32F407IGT	2	2	6				
STM32F427IIT	2	2	8				
STM32F051R8T						2x12b (19ch)	
	таймеры IC/OC/PWM	PWM таймер	DMA	WDT	SysTick 24b	RTC	
STM32F100C4T6B							
STM32F100RBT							
STM32F103C8T	3x16b		1x16b	7ch	2	●	●
STM32F103C8T	3x16b		1x16b	7ch	2	●	●
STM32F103RBT	3x16b		1x16b	7ch	2	●	●
STM32F103VBT	3x16b		1x16b	7ch	2	●	●
STM32F407VGT					●	●	
STM32F407IGT					●	●	
STM32F427IIT					●	●	
STM32F051R8T	7x16b 1x32b		7x16b 1x32b		2	●	●