

Министерство образования и науки Российской Федерации  
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ  
И РАДИОЭЛЕКТРОНИКИ(ТУСУР)

А.В.Пуговкин, И.А.Куан, Н.К. Ахметов, А.В. Бойченко

## **Методическое пособие по программированию микроконтроллеров**

Учебно-методическое пособие

Томск

2016

## Оглавление

1. Описание микроконтроллеров .....	3
1.1. Общие сведения .....	3
1.2. Описание микроконтроллера 1986BE92У .....	5
2. Демонстрационно-отладочная плата 1986EvBrd_64. Техническое описание. 8	
2.1. Общие положения .....	8
2.2. Состав платы .....	9
3. Описание среды разработки .....	15
4. Установка и настройка Keil uVision .....	16
4.1. Установка .....	16
4.2. Программатор .....	20
4.3. Настройка Keil и запуск демонстрационного проекта .....	22
4.4. Создание нового проекта в среде Keil uVision .....	28
5. Лабораторная работа №1. Повторение языка Си .....	36
6. Лабораторная работа №2. Порты ввода/вывода (General-purposeinput/output, GPIO) .....	46
7. Лабораторная работа №3. Использование таймера .....	53
8. Лабораторная работа №4. Универсальный приемопередатчик (USART) 61	
9. Список литературы .....	69

## 1. Описание микроконтроллеров

### 1.1. Общие сведения

**Микроконтро́ллер** (англ. *MicroControllerUnit*, *MCU*) — микросхема, предназначенная для управления электронными устройствами.

Типичный микроконтроллер сочетает на одном кристалле функции процессора и периферийных устройств, содержит ОЗУ и (или) ПЗУ. По сути, это однокристалльный компьютер, способный выполнять относительно простые задачи.

Отличается от микропроцессора интегрированными в микросхему устройствами ввода-вывода, таймерами и другими периферийными устройствами.

При проектировании микроконтроллеров приходится соблюдать компромисс между размерами и стоимостью с одной стороны и гибкостью, и производительностью с другой. Для разных приложений оптимальное соотношение этих и других параметров может различаться очень сильно. Поэтому существует огромное количество типов микроконтроллеров, отличающихся архитектурой процессорного модуля, размером и типом встроенной памяти, набором периферийных устройств, типом корпуса и т. д. В отличие от обычных компьютерных микропроцессоров, в микроконтроллерах часто используется гарвардская архитектура памяти, то есть раздельное хранение данных и команд в ОЗУ и ПЗУ соответственно.

Кроме ОЗУ, микроконтроллер может иметь встроенную энергонезависимую память для хранения программы и данных. Многие модели контроллеров вообще не имеют шин для подключения внешней памяти.

Наиболее дешёвые типы памяти допускают лишь однократную запись, либо хранимая программа записывается в кристалл на этапе изготовления (конфигурацией набора технологических масок). Такие устройства подходят для массового производства в тех случаях, когда программа контроллера не будет обновляться. Другие модификации контроллеров обладают

возможностью многократной перезаписи программы в энергонезависимой памяти.

Неполный список периферийных устройств, которые могут использоваться в микроконтроллерах, включает в себя:

- универсальные цифровые порты, которые можно настраивать как на ввод, так и на вывод;
- различные интерфейсы ввода-вывода, такие, как UART, I<sup>2</sup>C, SPI, CAN, USB, IEEE 1394, Ethernet;
- аналого-цифровые и цифро-аналоговые преобразователи;
- компараторы;
- широтно-импульсные модуляторы (ШИМ-контроллер);
- таймеры;
- контроллеры бесколлекторных двигателей, в том числе шаговых;
- контроллеры дисплеев и клавиатур;
- радиочастотные приемники и передатчики;
- массивы встроенной флеш-памяти;
- встроенные тактовый генератор и сторожевой таймер;

## **1.2. Описание микроконтроллера 1986BE92У**

Разработка Центра Проектирования российской компании ЗАО "ПКК Миландр" – 32-разрядный RISC микроконтроллер.

Микроконтроллеры серии 1986BE9х, K1986BE9х и K1986BE92QI, K1986BE92QC (далее 1986BE9х), построенные на базе высокопроизводительного процессорного RISC ядра ARM Cortex-M3, содержат встроенную 128 Кбайт Flash-память программ и 32 Кбайта ОЗУ. Микроконтроллеры работают на тактовой частоте до 80 МГц. Периферия микроконтроллера включает контроллер USB интерфейса со встроенным аналоговым приемопередатчиком со скоростями передачи 12 Мбит/с (FullSpeed) и 1.5 Мбит/с (LowSpeed), стандартные интерфейсы UART, SPI и I2C, контроллер внешней системной шины, что позволяет работать с внешними микросхемами статического ОЗУ и ПЗУ, NAND Flash-памятью и другими внешними устройствами. Микроконтроллеры содержат три 16-разрядных таймера с 4 каналами схем захвата и ШИМ с функциями формирования «мертвой зоны» и аппаратной блокировки, а также системный 24-х разрядный таймер и два сторожевых таймера. Кроме того, в состав микроконтроллеров входят: два 12-разрядных высокоскоростных (до 0,5М выборок в сек) АЦП с возможностью оцифровки информации от 16 внешних каналов и от встроенных датчиков температуры и опорного напряжения; два 12-разрядных ЦАП; встроенный компаратор с тремя входами и внутренней шкалой напряжений.

Таблица 1.1 - Основные характеристики микроконтроллеров серии 1986BE9х

	<u>1986BE91T</u> <u>1986BE94T</u>	<b>1986BE92Y,</b> <b>K1986BE92QI</b>	<u>1986BE93Y</u>
<b>Корпус</b>	4229.132-3, LQFP144,100 (*)	H18.64-1B, LQFP64	H16.48-1B
<b>Ядро</b>	ARM Cortex-M3		
<b>ПЗУ</b>	128 Кбайт Flash		
<b>ОЗУ</b>	32 Кбайт		
<b>Питание</b>	2.2...3.6В		
<b>Частота</b>	80 МГц		
<b>Температура</b>	минус 60°С...+125°С (**)		
<b>USER IO</b>	96	43	30
<b>USB</b>	Device и Host FS (до 12 Мбит/с), встроенный PHY		
<b>UART</b>	2	2	2
<b>CAN</b>	2	2	2
<b>SPI</b>	2	2	1
<b>I2C</b>	1	1	нет
<b>2хADC 12 разрядов 1 Мвыб/с</b>	16 каналов	8 каналов	4 канала
<b>DAC 12 разрядов</b>	2	1	1
<b>Компаратор</b>	3 входа	2 входа	2 входа
<b>Внешняя шина</b>	32 разряда	8 разрядов	нет

Таблица 1.2 - Сравнение микроконтроллеров STМи Миландр

	<b>Stm32f0</b>	<b>1986BE92Y</b>
<b>Корпус</b>	H18.64-1B, LQFP64	UFQFN32, LQFP32, LQFP48,LQFP64
<b>Ядро</b>	ARM Cortex-M0	ARM Cortex-M3
<b>ПЗУ</b>	До 64кбайт Flash	128 Кбайт Flash
<b>ОЗУ</b>	8кбайт	32 кбайт
<b>Питание</b>	2...3.6В	2.2...3.6В
<b>Частота</b>	48МГц	80 МГц
<b>Температура</b>	-40°С ....+105°С	- 60°С...+125°С
<b>Коммуникационные интерфейсы</b>	I2C, USART, SPI, I2S,HDMI	I2C, USART, SPI, CAN,HDMI

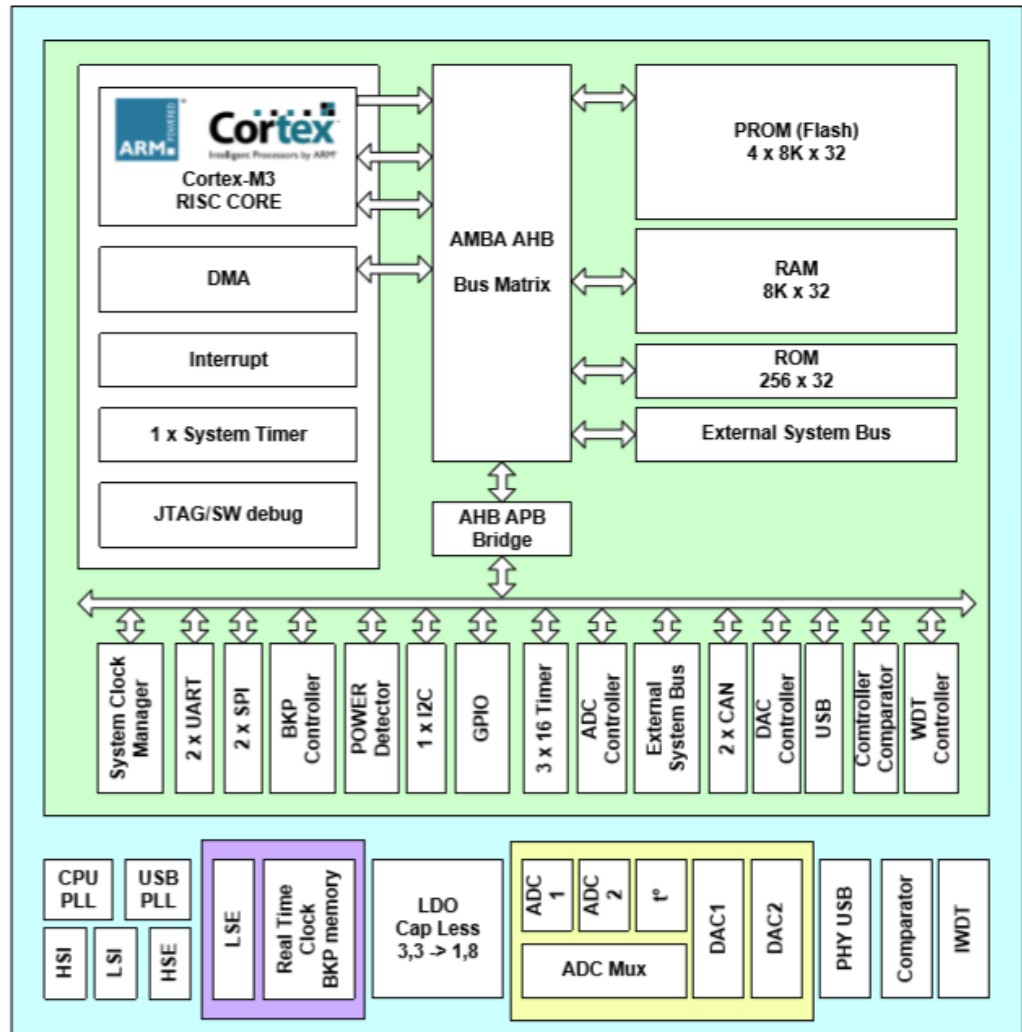


Рисунок 1.1 – Структурная блок-схема микроконтроллера 1986BE9x

## **2. Демонстрационно-отладочная плата 1986EvBrd\_64.**

### **Техническое описание.**

#### **2.1. Общие положения.**

Демонстрационно-отладочная плата 1986EvBrd\_64 (далее 1986EvBrd\_64) предназначена для:

- демонстрации функционирования и оценки производительности микроконтроллера 1986BE92У и его основных периферийных модулей;
- демонстрации функционирования интерфейсных микросхем CAN и COM (RS-232) интерфейсов;
- отладки собственных проектов с применением установленных на плате блоков;
- программирования памяти программ микроконтроллеров 1986BE92У.

Для демонстрации функционирования, 1986EvBrd\_64 подключается к:

- к COM порту персонального компьютера;
- к CAN или COM (RS-232) интерфейсу дополнительного внешнего устройства, например, аналогичной демонстрационно-отладочной плате 1986EvBrd\_64;
- к источнику питания +5В.

Для программирования памяти программ микроконтроллеров 1986BE92У применяется внешний внутрисхемный программатор ULINK2 (Keil) или JEM-ARM-V2(Phyton).

Питание 1986EvBrd\_64 осуществляется от адаптера постоянного тока напряжением +5 вольт или от шины USB.

Комплектация:

- печатная плата 1986EvBrd\_64;
- образец микроконтроллера 1986BE92У;
- нуль-модемный кабель для COM (RS-232) интерфейса;
- кабель USB-A/USB-B;



- блок питания для отладочной платы
- диск с программным обеспечением, документацией, схемотехническими файлами и исходными кодами программ.

## 2.2. Состав платы

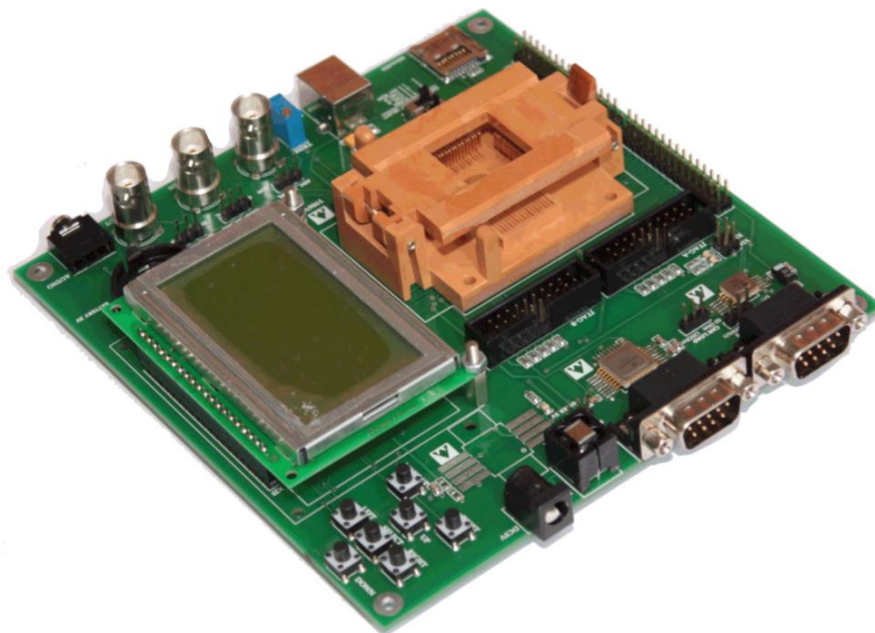


Рисунок 2.1– Внешний вид демонстрационно-отладочной платы

Установленные на плату компоненты показаны Рисунок 2.2, их описание содержится в Таблица 2.1.

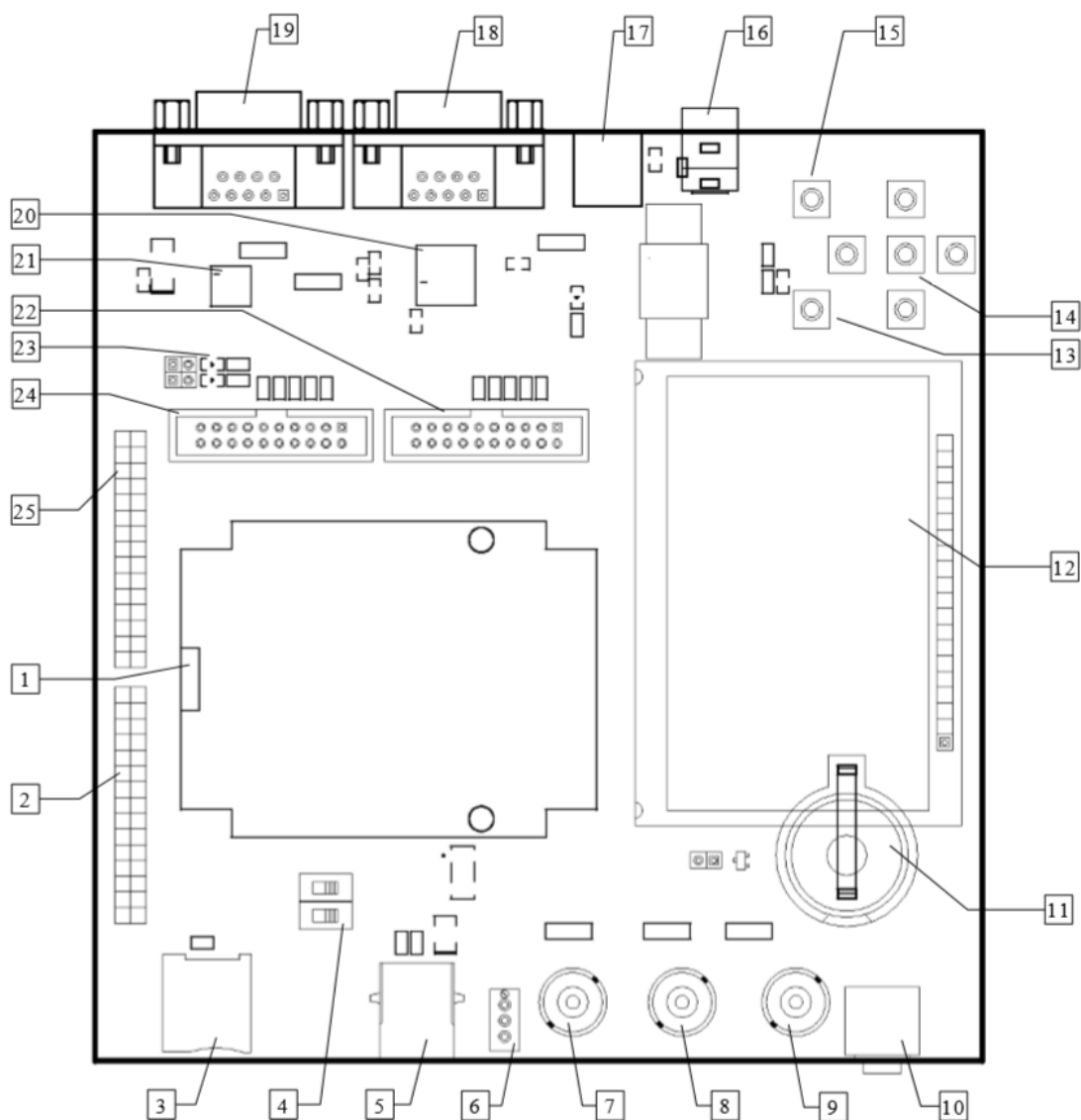


Рисунок 2.2– Компоненты платы

Таблица 2.1 - Описание компонентов платы 1986EvBrd\_64

№ на Рисунок 2.2	Описание компонентов платы 1986EvBrd_64
1	Контактирующее устройство для микроконтроллера 1986BE92Y. Микроконтроллер должен быть установлен в спутник-держатель.
2	Разъем X27 портов A,E,F микроконтроллера.
3	Разъем карты памяти microSD.
4	Переключатели выбора режима загрузки.
5	Разъем USB-B.

№ на Рисунок 2.2	Описание компонентов платы 1986EvBrd_64
6	Подстроечный резистор на 7-м канале АЦП.
7	Разъем BNC внешнего сигнала на 7-м канале АЦП.
8	Разъем BNC внешнего сигнала на 1-м входе компаратора
9	Разъем BNC выхода ЦАП1
10	Разъем Audio 3.5мм выхода ЦАП1 через звуковой усилитель
11	Батарея 3.0В
12	ЖК индикатор 128х64
13	Кнопка WAKEUP
14	Кнопки UP, DOWN, LEFT, RIGHT, SELECT
15	Кнопка RESET.
16	Разъем питания 5В.
17	Фильтр питания
18	Разъем RS-232.
19	Разъем CAN.
20	Приемо-передатчик RS-232 5559ИН4.
21	Приемо-передатчик CAN 5559ИН14.
22	Разъем отладки JTAG-B.
23	Набор светодиодов на порте С.
24	Разъем отладки JTAG-A.
25	Разъем X26 портов В,С,Д микроконтроллера.

Таблица 2.2- Подключение портов микроконтроллера к разъемам X26, X27

Контакт	Вывод МК/питание	
	X26	X27
1,2	GND	GND

Контакт	Вывод МК/питание	
	X26	X27
3,4	+3,3V	+3,3V
5	PD0	PA6
6	PD1	PA7
7	PD2	PA4
8	PD3	PA5
9	PD4	PA2
10	PD5	PA3
11	PD6	PA0
12	-	PA1
13	PB0	-
14	PB1	-
15	PB2	PE1
16	PB3	PE3
17	PB4	-
18	PB5	-
19	PB6	PF0
20	PB7	PF1
21	PB8	PF2
22	PB9	PF3
23	PB10	PF4
24	PC0	PF5
25	PC1	PF6
26	PC2	-
27,28	+5V	+5V
29,30	GND	GND

Назначение установленных на плате конфигурационных перемычек:

- POWER\_SEL – выбор источника питания для платы между разъемом USB и внешним источником питания.
- SLEW RATE – выбор скорости передачи данных интерфейса CAN.
- CAN\_LOAD – выбор нагрузки линии CAN.
- ADC\_INP\_SEL – выбор источника сигнала для 7-го канала АЦП между подстроечным резистором “TRIM” и BNC разъемом “ADC”.
- COMP\_INP\_SEL – выбор источника сигнала на 1-м входе компаратора между BNC разъемом “COMP\_INP” и выходом ЦАП1.
- DAC\_OUT\_SEL – выбор назначения сигнала с выхода ЦАП1 между BNC разъемом “DAC\_OUT” и звуковым усилителем.

Назначение установленных на плате переключателей и клавиш:

- SW1, SW2 – переключатели выбора режима работы.

Таблица 2.3– Режимы работы

SW2	SW1	Режим работы
0	0	Режим микроконтроллера, код выполняется из Flash памяти начиная с адреса 0x0800_0000.
0	1	Режим микроконтроллера, код выполняется из Flash памяти начиная с адреса 0x0800_0000, отладка через разъем JTAG_A.
1	0	Режим микропроцессора, код выполняется из внешней памяти начиная с адреса 0x1000_0000.
1	1	Режим микропроцессора, код выполняется из внешней памяти начиная с адреса 0x1000_0000, отладка через разъем JTAG_B.

- UP, DOWN, LEFT, RIGHT, SELECT – программируемые пользователем клавиши.
- RESET – сигнал аппаратного сброса МК.

- WAKEUP – сигнал внешнего выхода из режима Standby.

### **3. Описание среды разработки**

Для программирования микроконтроллеров используется среда разработки KeilVision.

Немецкая фирма Keil разрабатывает и поставляет среды разработки для платформ: ARM, 8085, 251, C166, JTAG-отладчики и отладочные платы для них. Следует отметить, что компания Keil является официальным партнером ARM, а сама Keil-MDK является совместной разработкой Keil и ARM (<http://www.arm.com>). Так ядро IDE (компилятор, линковщик, ассемблер и ряд утилит) собственная разработка ARM, от Keil-а только оболочка ( $\mu$ Vision IDE) и отладчик.

## 4. Установка и настройка KeilVision

### 4.1. Установка

В папке «Материалы для лабораторных работ» запустить файл «MDK520.EXE» (Рисунок 4.1).

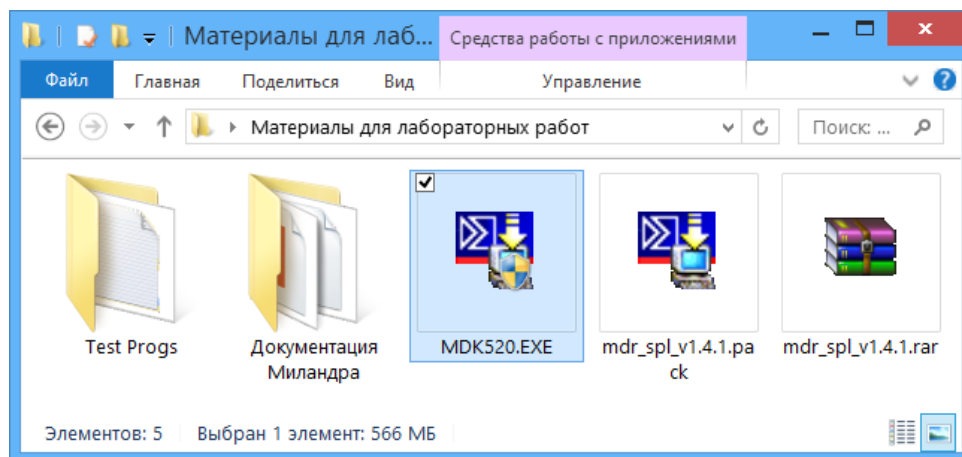


Рисунок 4.1

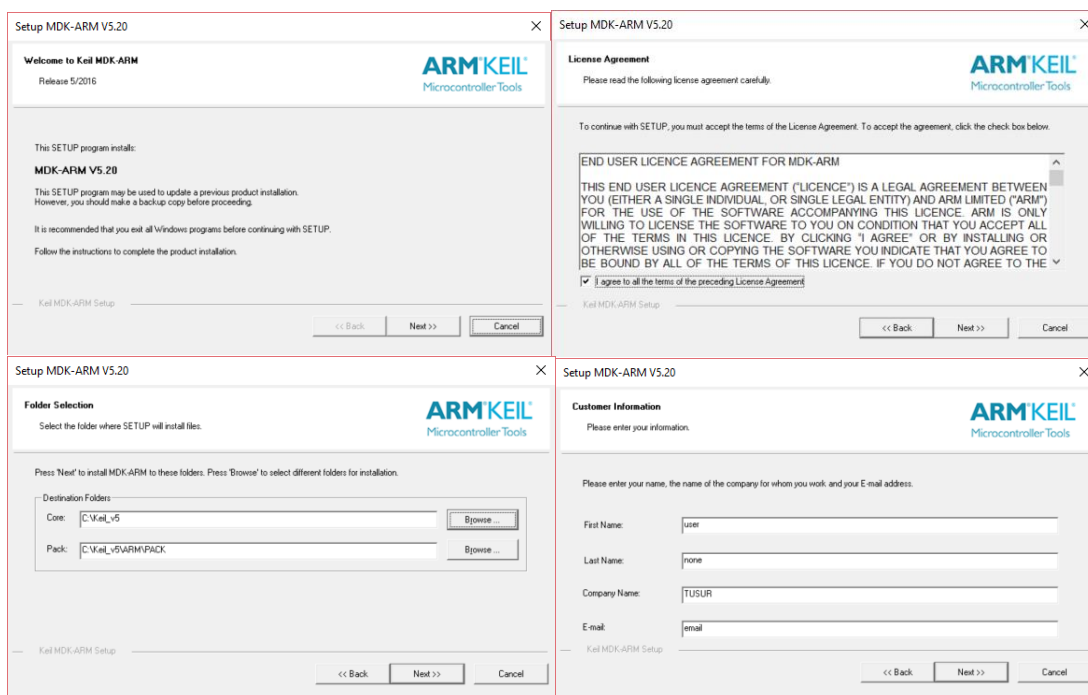


Рисунок 4.2 – Процесс установки



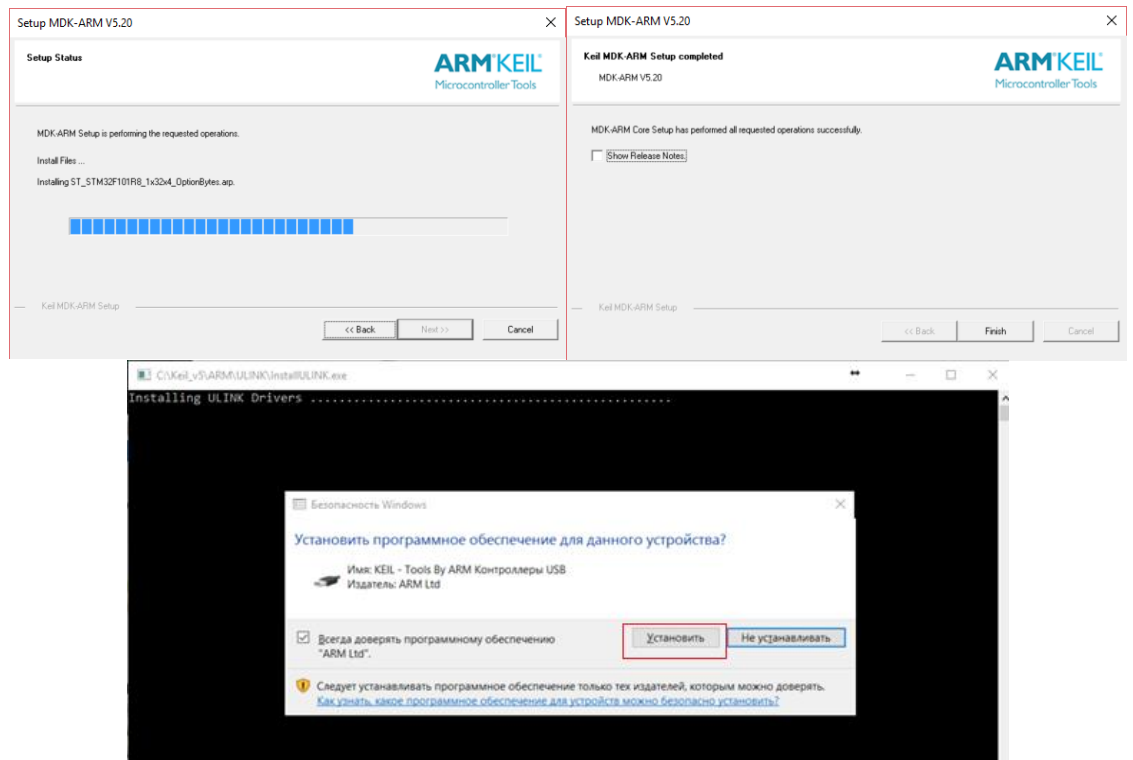


Рисунок 4.3 – Процесс установки

После установки, запускаем Keil. При первом запуске запустится PackInstaller (Рисунок 4.4).

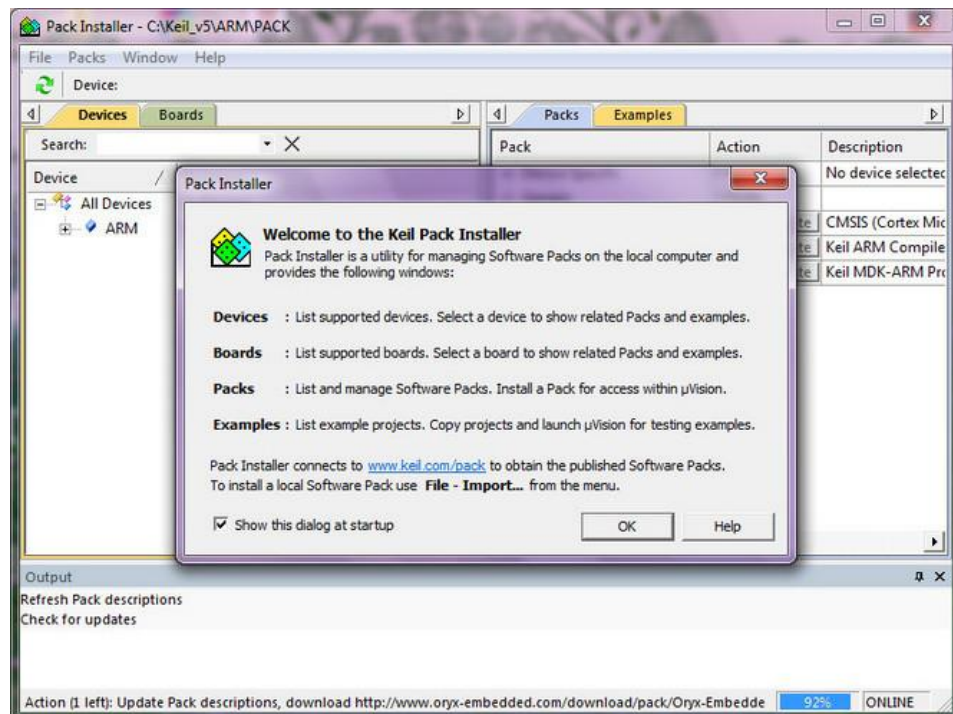
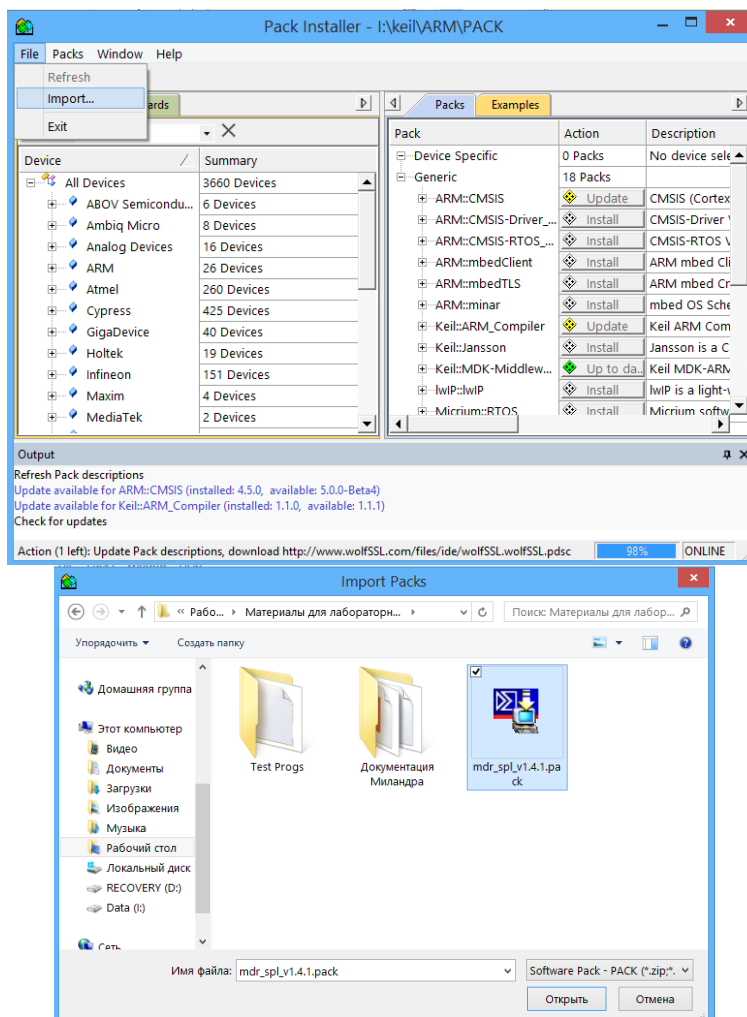


Рисунок 4.4 - PackInstaller

Теперь добавим пакет поддержки контроллеров Миландр.Пакет находится в папке «Материалы для лабораторных работ» файл «mdr\_spl\_v1.4.1.rar».

В PackInstaller жмем File ->Import и выбираем необходимый нам пакет (Рисунок 4.5).



### Рисунок 4.5 – Добавление пакета в PackInstaller

После установки пакета во вкладке Devices должен появиться раздел Milandr, а во вкладке Packs раздел Keil::MDR1986BExx.

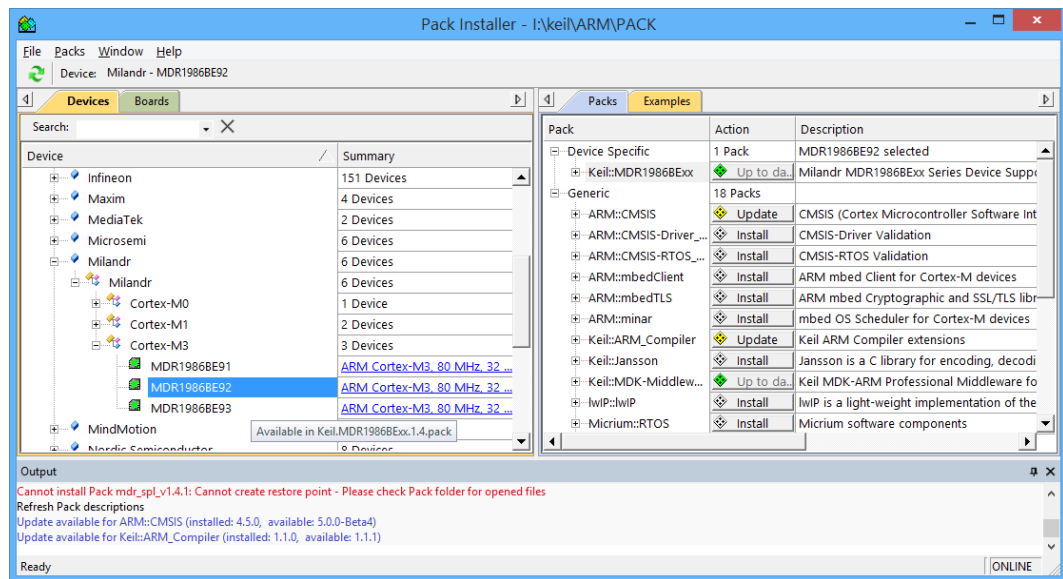


Рисунок 4.6 – Результат добавления пакета

Закрываем PackInstaller и запускаем Keil.

## 4.2. Программатор

Для отладки, тестирования и программирования внутренней памяти микроконтроллеров необходим программатор-отладчик. Нами будет использоваться программатор MT-Link. Он является аналогом известного программатора J-Link.

Программатор подключается к компьютеру с помощью USB-кабеля и использует интерфейсы для внутрисхемной отладки SWD – SerialWireDebug или JTAG. На плате предусмотрено два разъема для подключения программатора (JTAG-A и JTAG-B).



где 1 – кабель USB; 2 - шлейф программатора; 3-программатор MT-Link.

Рисунок 4.7 – Программатор MT-Link

Установим драйвера программатора J-Link(MT-Link), запустив файлы «Setup\_JLinkARM\_V468a» и «MT-Link», которые находятся в папке «Материалы для лабораторных работ».

Программатор MT-Link и USB-кабель соедините междусобой.

Подключите шлейф программатора к разъему JTAG-A, расположенному на плате.

Установите переключатели в положения:

Таблица 4.1 – Положения переключателей

SW1	SW2	SW3
1	0	0

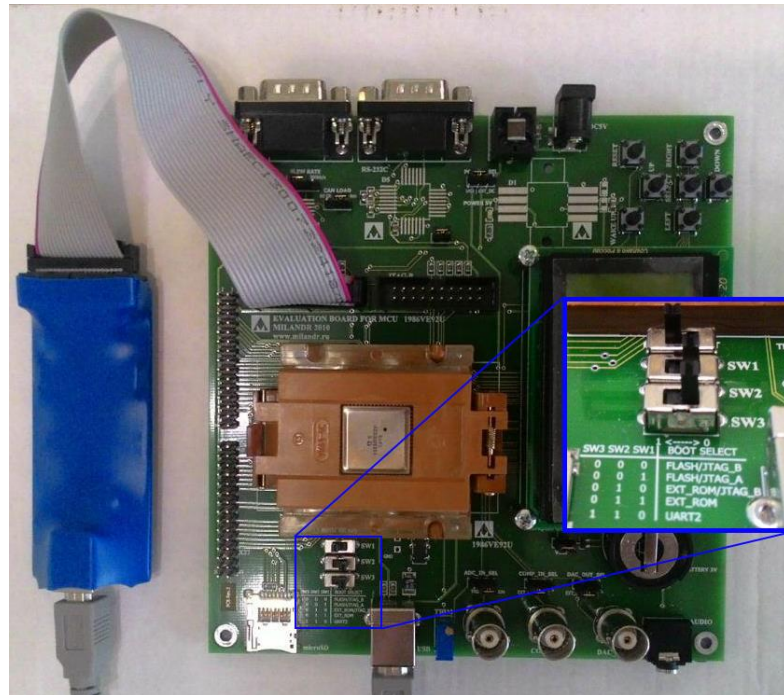


Рисунок 4.8 – Подключение платы

Подключаем MT-Link к компьютеру, в диспетчере устройств должно отобразиться устройство J-Linkdriver в разделе «Контроллеры USB» (Рисунок 4.9).

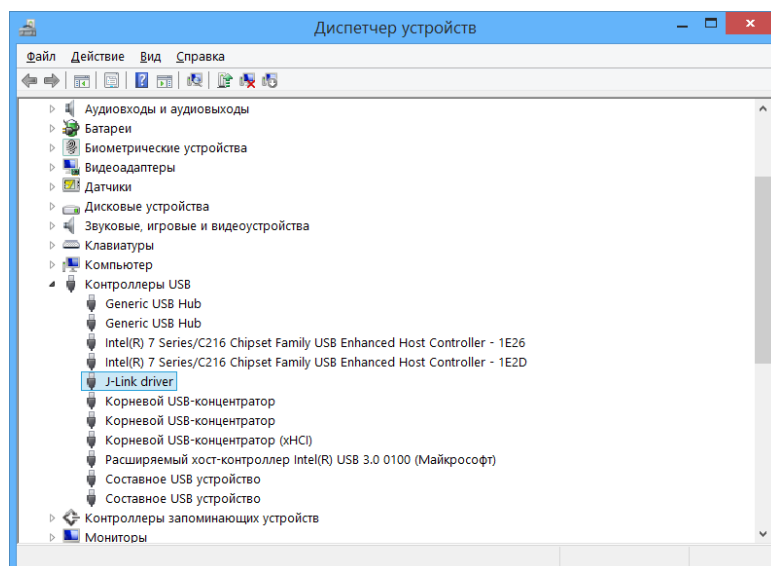


Рисунок 4.9

### 4.3. Настройка Keil и запуск демонстрационного проекта

Далее необходимо скопировать файл «MDR32F9x.FLM», который находится в папке «Материалы для лабораторных работ», в папку «Flash» где установлен Keil uVision (по умолчанию путь «C:\Keil\_v5\ARM\Flash»).

Запустим демонстрационную программу EV1986BE2Test.uvproj, которая находится в папке «Материалы для лабораторных работ->TestProgs».

Переходим в Project -> «Options for Target» (Рисунок 4.10).

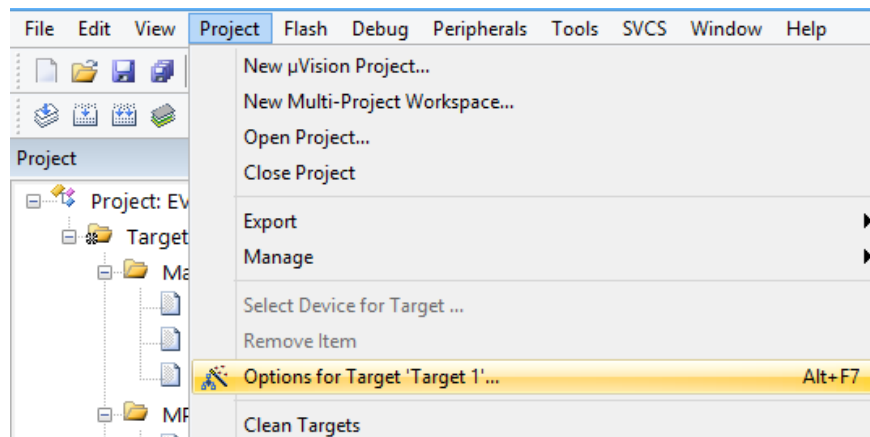


Рисунок 4.10

В вкладке Device необходимо выбрать процессор ARM Cortex-M3: Milandr->Milandr->Cortex-M3->MDR1986BE92.

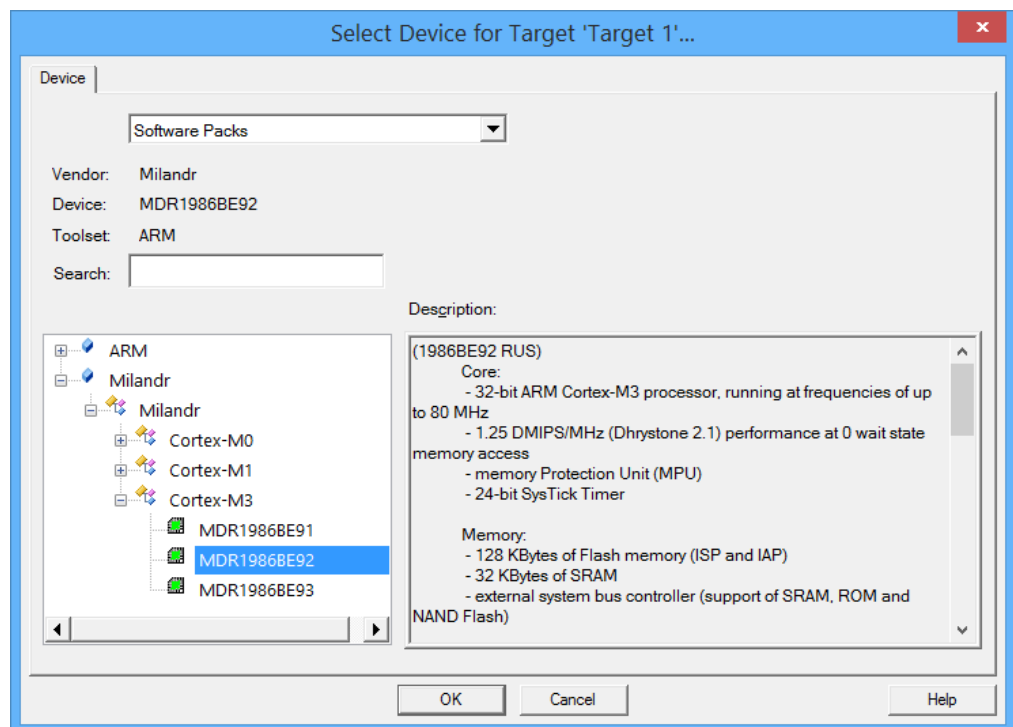


Рисунок 4.11 – Выбор процессора

Во вкладке Debug выбираем(устанавливаем) следующие параметры:

- Use: J-LINK/J-TRACE Cortex
- Load Application at Startup
- Run to main().

После чего (далее) нажмем кнопку «Settings».

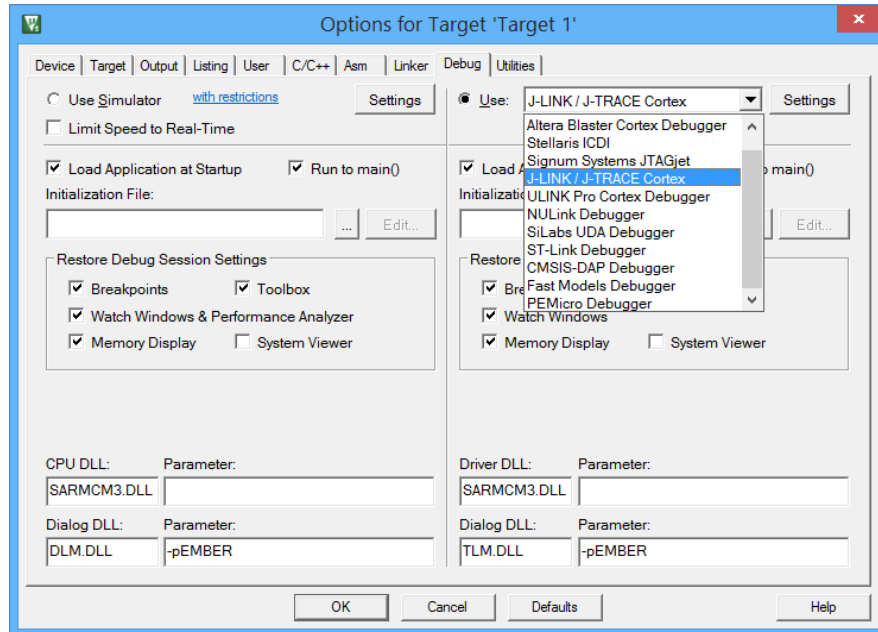


Рисунок 4.12 – Установка параметров во вкладке Debug

В списке «PORT» нужно сменить JTAG на SW и выбрать частоту в списке рядом в 1MHz.

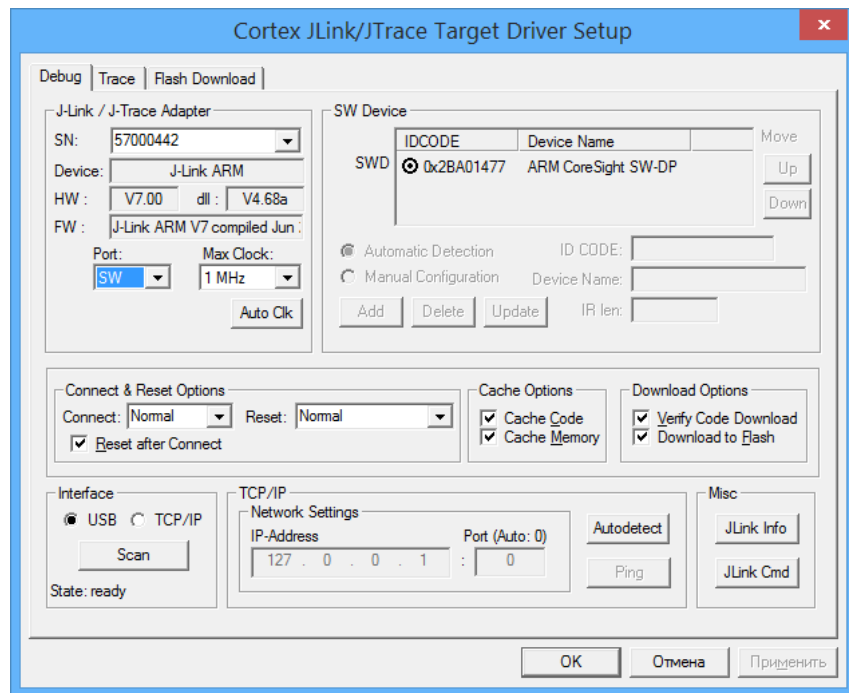


Рисунок 4.13 – Настройка J-Link

Переходим во вкладку «FlashDownload» (Рисунок 4.14). Там ставим галочку «EraseFullChip» и затем нажимаем кнопку Add.

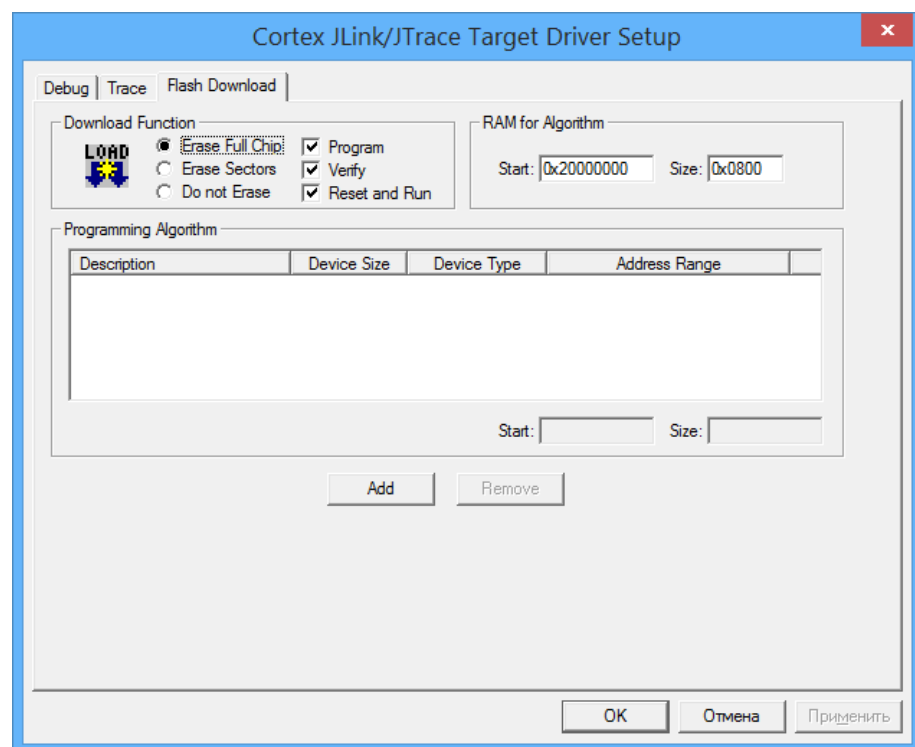


Рисунок 4.14

Из списка выбираем нужный микроконтроллер. Затем нажимаем кнопку Add (Рисунок 4.15).



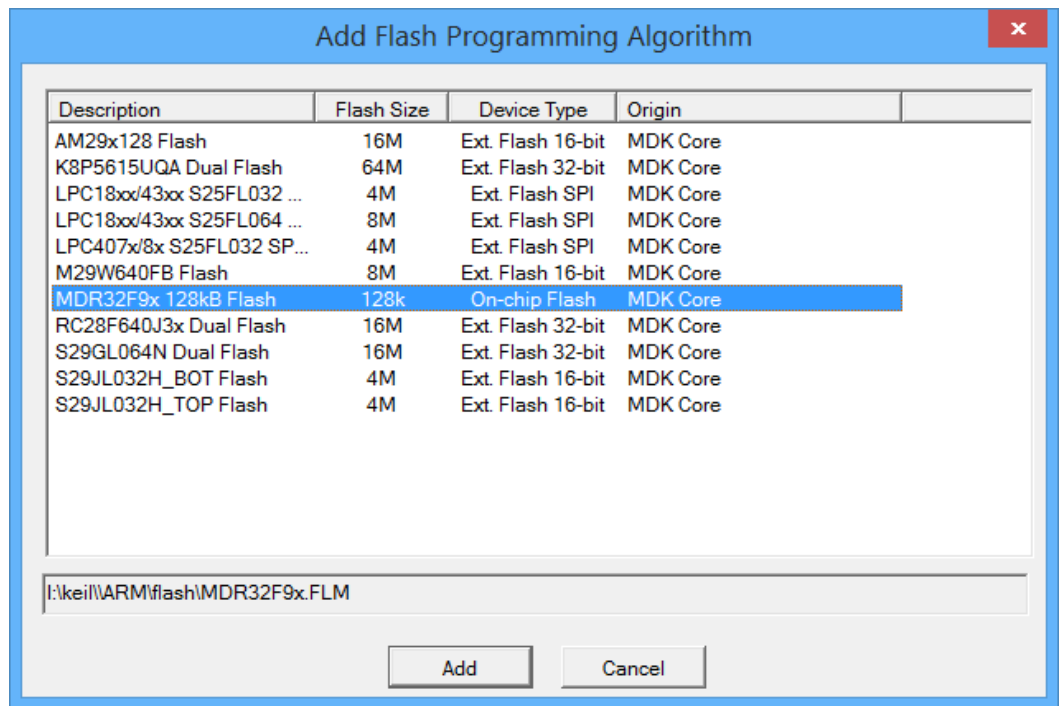


Рисунок 4.15

После добавления микроконтроллера он отражается в окне ProgrammingAlgorithm. Нажмите кнопку ОК (Рисунок 4.16).

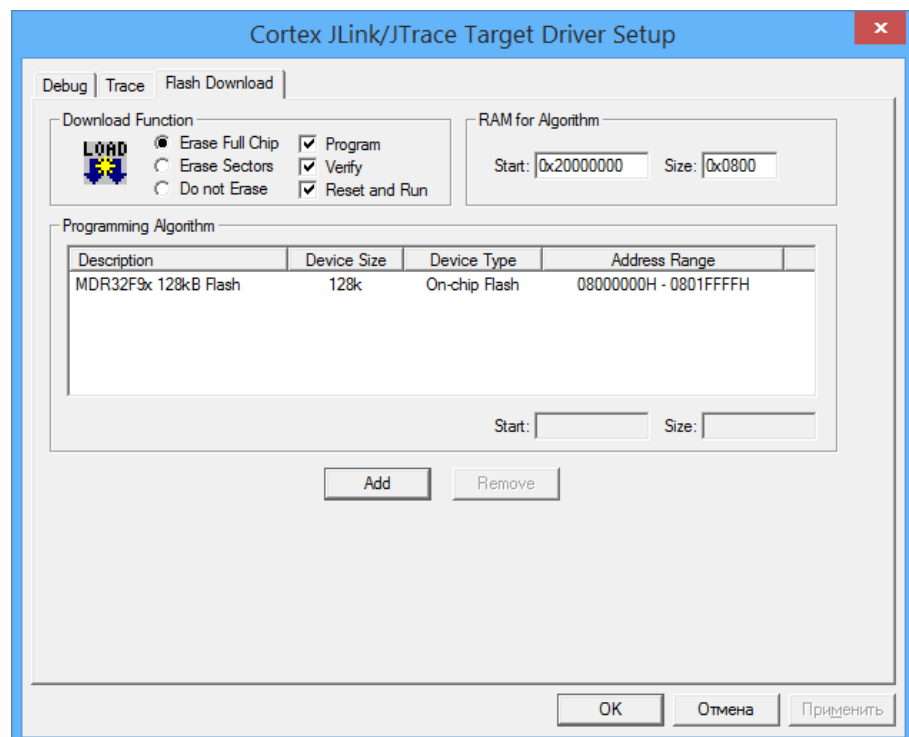


Рисунок 4.16

Теперь среда разработки KeilVision готова для разработки и отладки приложений на микроконтроллере.

Теперь мы можем запустить демонстрационный проект. Для этого в главном меню выбираем Project->BuildTarget.

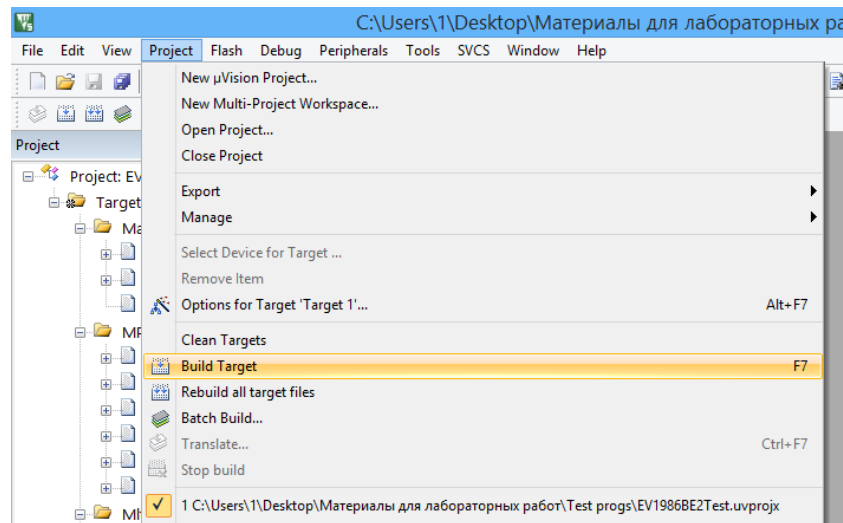


Рисунок 4.17 – Построение проекта

При успешной компиляции, в левом нижнем окне BuildOutput появится надпись «0 Error(s), 0 Warnings».

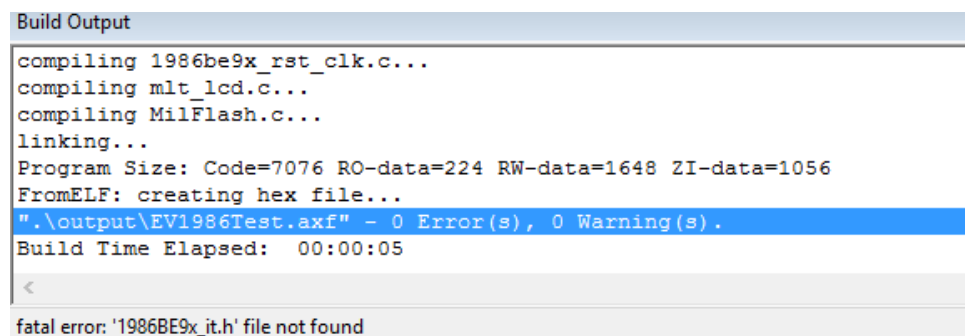


Рисунок 4.18 – Окно BuildOutput

Теперь необходимо загрузить программу в микроконтроллер выбрав Flash->Download.

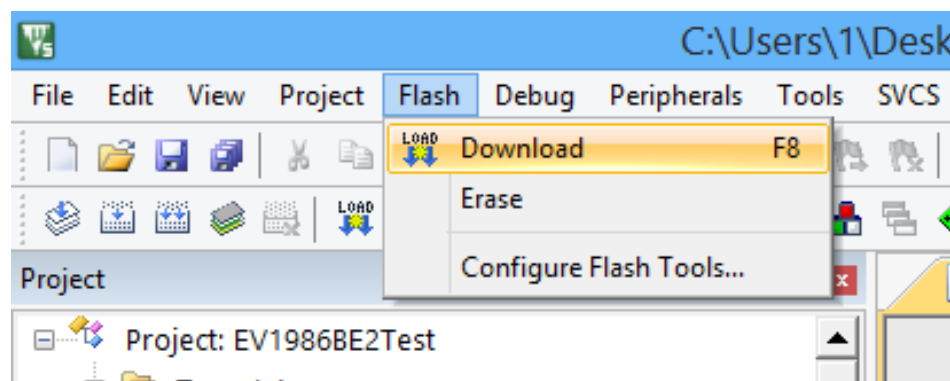


Рисунок 4.19 – Загрузка проекта

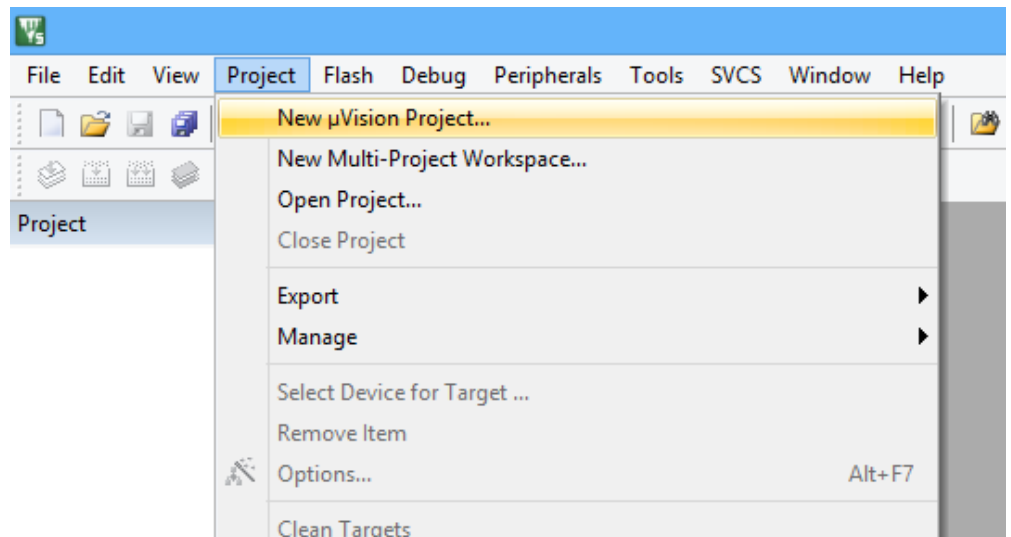
В результате успешной загрузки на LED-дисплее вы увидите следующее сообщение. Управляя клавишами, в меню можно включить различные тесты. Выберите тест светодиодов, установив курсор напротив LEDS, нажмите кнопку SELECT. В результате светодиоды на порте C загорятся.



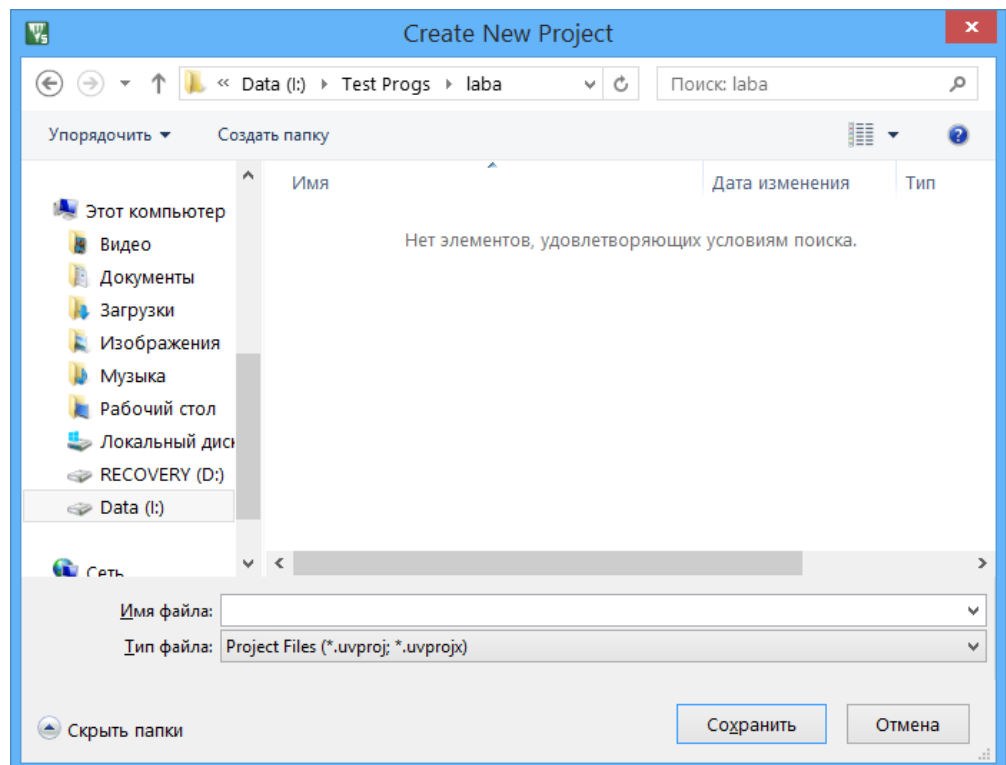
Рисунок 4.20 – Результат исполнения демонстрационного проекта

#### 4.4. Создание нового проекта в среде KeilVision

Запустите среду KeilVision5, в строке главного меню выберите:  
Project -> New uVision Project...

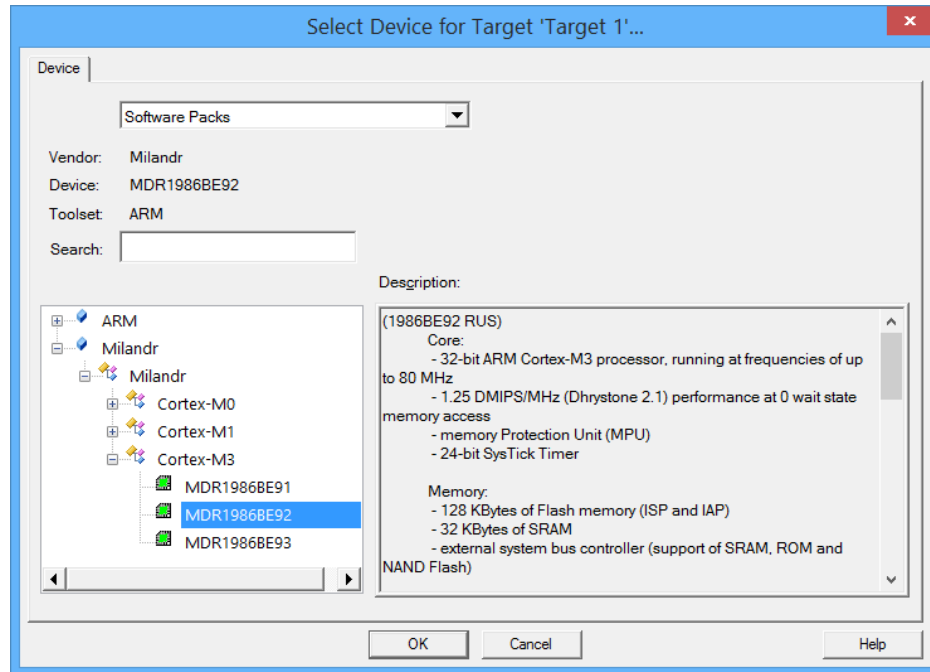


Создайте папку для проекта. Примечание: путь до папки с проектом не должен содержать кириллицы.

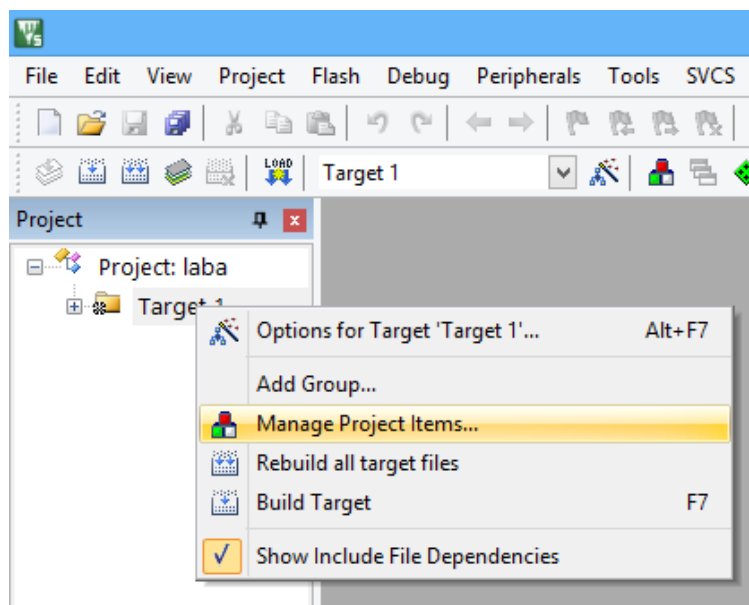


После создания нового проекта в появившемся окне выбора микроконтроллера во вкладке Device необходимо выбрать процессор ARM Cortex-M3: Milandr->Milandr->Cortex-M3->MDR1986BE92.

После выбора микроконтроллера появляется окно с выбором библиотек. Нажмите «Ок», все необходимые библиотеки добавим в ручную, взяв их из демонстрационного проекта.

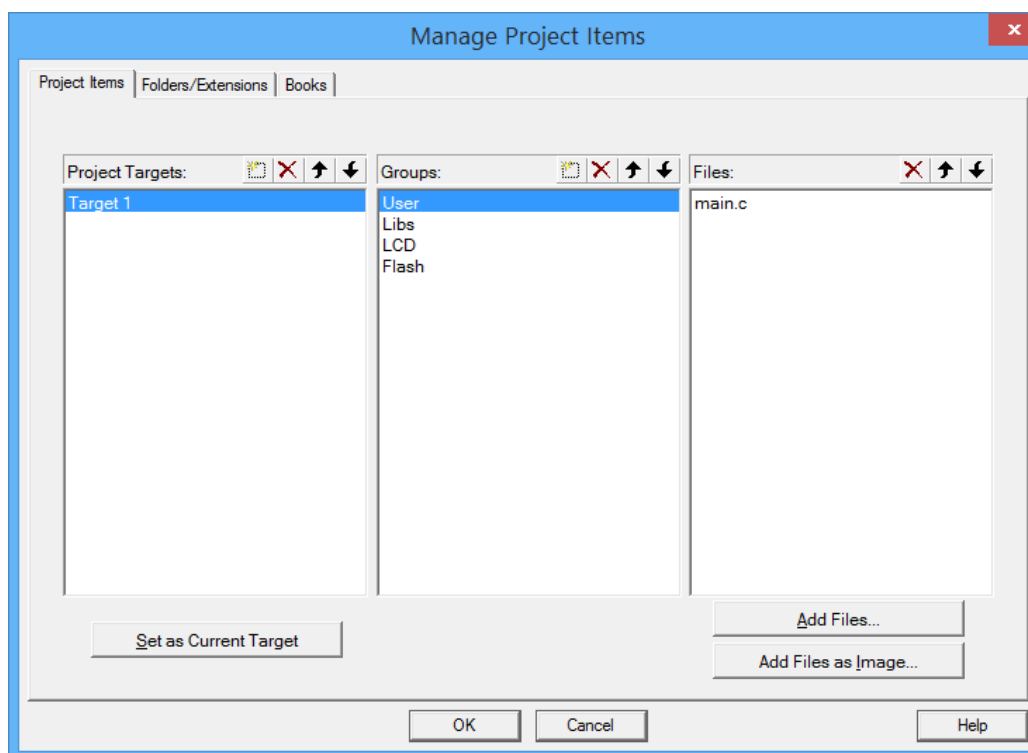


В результате у нас получилось дерево проекта. Для того чтобы создать необходимые подкатегории. Для этого по самой верхней папке жмем правой кнопкой мыши и выбираем «ManageProjectitems...»

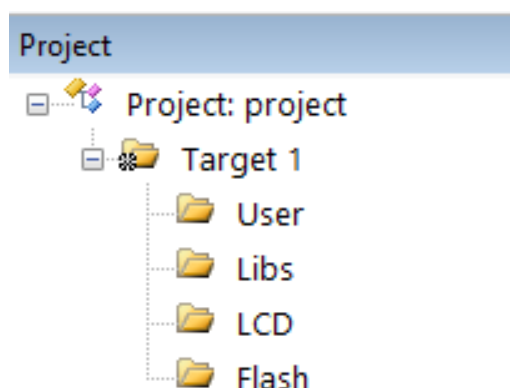


В разделе «Groups» создадим несколько папок для различных видов файлов:

- User— для пользовательских данных;
- LCD - для драйверов LCD, библиотека работы с LCD, который установлен на плате.
- Flash- для кода работы с Flash, здесь хранится библиотека для работы с Flashконтроллера.
- Libs—для библиотек CMSIS и SPL.

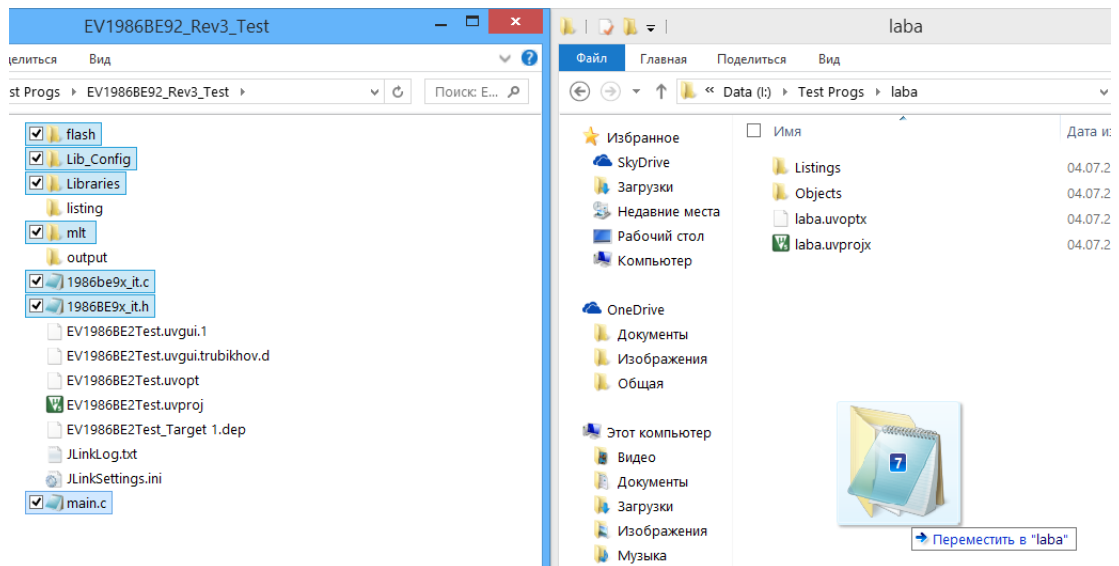


Структура проекта выглядит следующим образом:



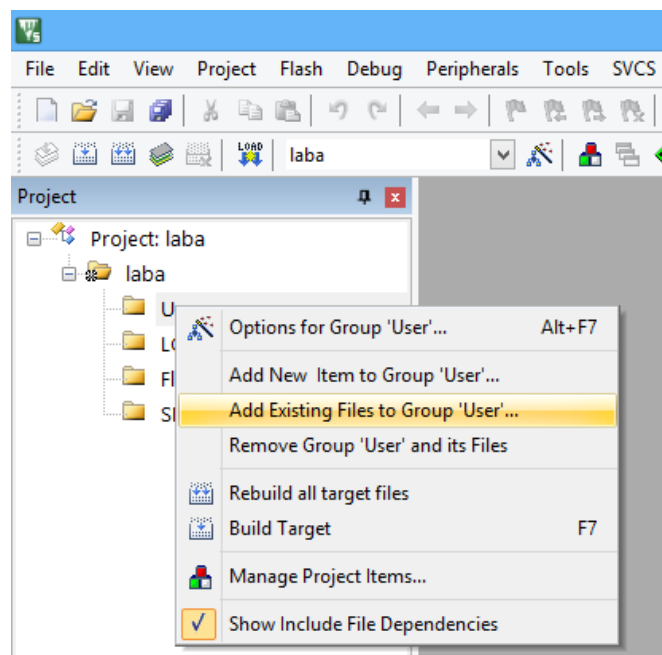
Далее необходимо добавить все необходимые файлы из демонстрационного проекта «Материалы для лабораторных работ->TestProgs».

Скопировать нужно выделенные файлы. Файл main.c добавить в папку User.



Далее необходимо добавить файлы библиотеки в проект.

Для этого нажимаем правой кнопкой по нужной папке в дереве проекта и выбираем «AddExistingFilestoGroup 'Имя группы'...».



В открывшемся окне нужно выбрать тип файлов «Allfiles (\*.\*)». После чего выбрать все необходимые файлы.

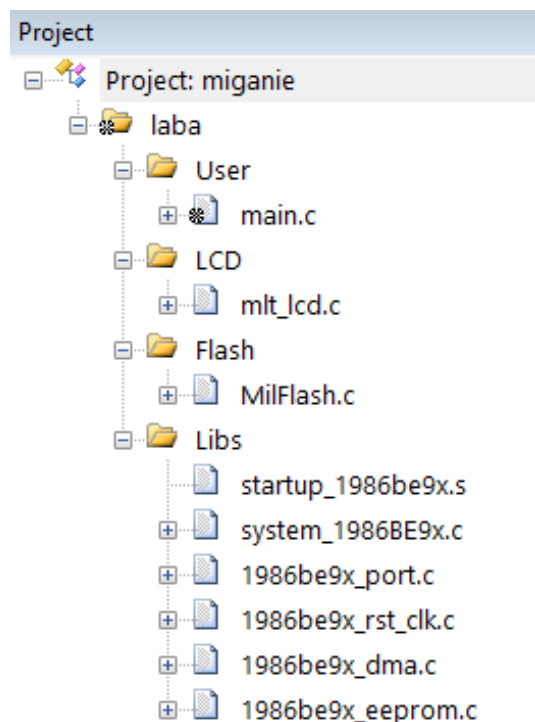
Необходимо добавить:

- В папку User -> «User ->main.c».
- В папку Flash -> «flash ->MilFlash.c».
- В папку LCD -> «mlt ->mlt\_lcd.c».

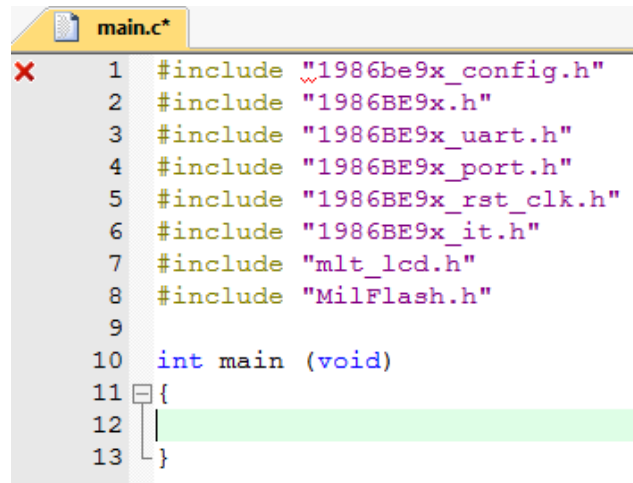


- В папку Libs -> «Libraries\1986BE9x\_StdPeriph\_Driver\src -> все файлы .c».
- В папку Libs -> «Libraries\CMSIS\CM3\DeviceSupport\1986BE9x\startup\arm -> startup\_1986be9x.s». В этом файле прописаны все «вектора переходов». Иначе говоря, по любому прерыванию (к примеру, нажатие кнопки) контроллер возвращается к этой таблице и смотрит, куда ему перейти, чтобы выполнять код дальше.
- В папку Libs -> «Libraries\CMSIS\CM3\DeviceSupport\1986BE9x\startup\arm -> system\_1986BE9x.c».

В результате проект примет следующий вид:



Далее необходимо очистить файл `main.c`. Удаляем все, кроме оболочки функции `main` и `#include` файлов. Должно остаться так:

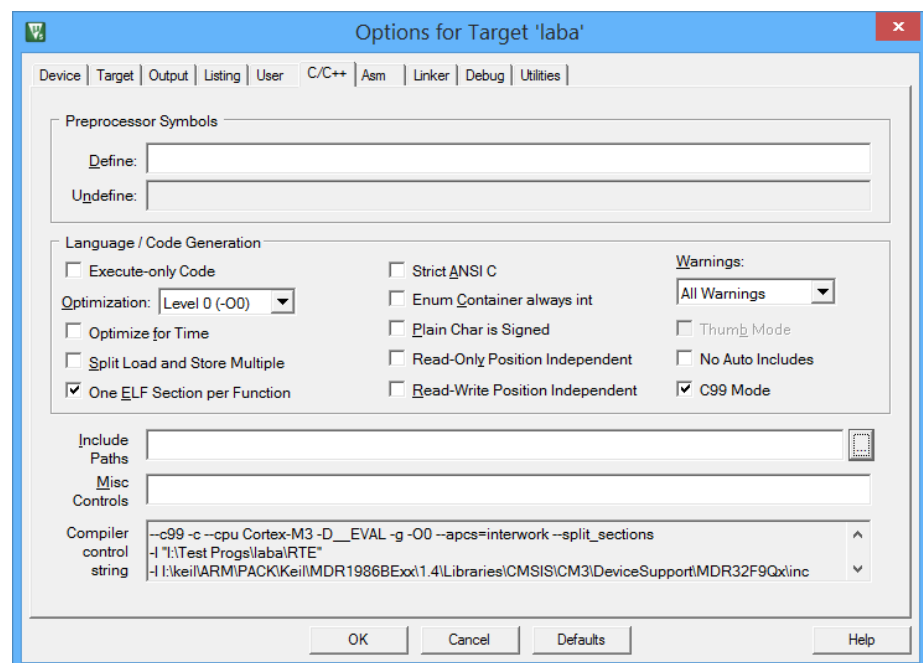


```

1  #include "1986be9x_config.h"
2  #include "1986BE9x.h"
3  #include "1986BE9x_uart.h"
4  #include "1986BE9x_port.h"
5  #include "1986BE9x_rst_clk.h"
6  #include "1986BE9x_it.h"
7  #include "mlt_lcd.h"
8  #include "MilFlash.h"
9
10 int main (void)
11 {
12
13 }

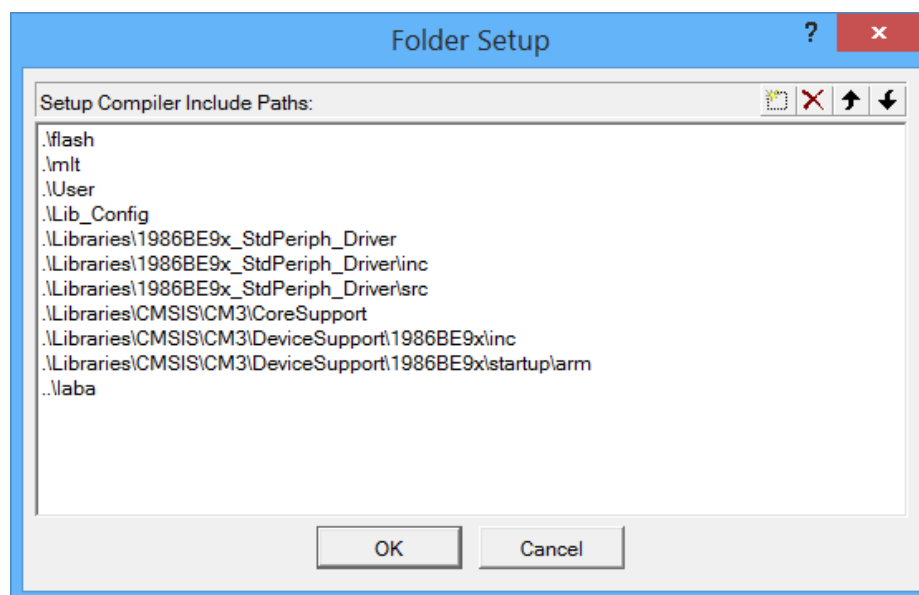
```

Около самого верхнего `#include` файла стоит крестик - keil не видит данный файл. Для того, чтобы исправить это, необходимо указать путь к этому файлу. Для этого жмем `Alt+F7`. В открывшемся окне переходим во вкладку `C/C++`. Для того, чтобы исправить это, нужно нажать галочку около надписи «C99 Mode». Это даст возможность писать на более совершенном стандарте языка Си, чем это можно было делать изначально. Далее следует нажать на прямоугольник с «...» внутри. Справа около строчки с подписью «IncludePaths».



В открывшемся окне нажимаем на иконку с прямоугольником, слева от крестика. Это создаст пустую строку. В правом углу созданной строчки жмем на «...». После чего указываем нужную папку, в которой лежат интересующие

нас файлы. После этого жмем «ОК». Папка будет добавлена. Необходимо добавить все эти пути.



Жмем «ОК» и переходим в файл main.c.

Теперь осталось лишь в настройках настроить J-LINK (см. выше).

## 5. Лабораторная работа №1. Повторение языка Си

### Цель работы:

Повторение основ программирования на языке Си. Разработка первых программ.

### Теоретические сведения

#### Компиляция

Си (англ. C) — это язык высокого уровня.

Программа, написанная на языке Си, является **текстовым** файлом с расширением «.c». Такая программа преобразуется с помощью **компиляции** в программу низкого уровня, состоящую из машинных кодов.

Компиляция состоит из следующих этапов:

1. *Препроцессирование.* Препроцессор добавляет к исходному файлу \*.c или \*.cpp заголовочные файлы \*.h, \*.hpp константы. Получается единый текстовый файл.

2. *Трансляция.* Единый текстовый файл передается *транслятору*. На выходе транслятора возникает файл с ассемблерным текстом.

3. *Ассемблирование.* Ассемблерный текст передается программе, которая преобразует его в *объектный файл* \*.obj.

4. *Линковка.* Линковщик объединяет все поданные ему объектные файлы с библиотечными файлами и формирует один большой файл.

#### Заголовочные файлы

Структура сокращенной программы на языке Си может быть такой:

```
#include "stdafx.h" //подключение заголовочных файлов
int main() //заголовок основной программы
{
    //тело основной программы
}
```

Функция *main* — главная функция, с которой начинается выполнение программы. Директива `#include` подключает библиотеки.

## Типы данных

В языке C имеется следующий набор стандартных типов данных:

- целые числа (`int`, `long`, `unsigned int`, `unsigned long`, `bool`);
- вещественные числа с плавающей точкой (`float`, `double`, `longdouble`);
- указатели;
- символьные переменные (`char`);
- тип `void`.

Диапазоны значений стандартных типов данных приведены в таблице ниже.

Таблица 5.1 – Типы переменных

Тип	Диапазон значений	Размер(байт)
Bool	true и false	1
Char	-128 .. 127	1
unsigned char	0 .. 255	1
int	-32 768 .. 32 767	2
unsigned int	0 .. 65 535	2
long	-2 147 483 648 .. 2 147 483 647	4
unsigned long	0 .. 4 294 967 295	4
float	3.4e-38 .. 3.4e+38	4
double	1.7e-308 .. 1.7e+308	8
long double	3.4e-4932 .. 3.4e+4932	10

## Операторы

Операторы предназначены для осуществления действий и для управления ходом выполнения программ.

*Условный оператор:*

`if` (условие) (оператор1).

Более сложная форма условного оператора содержит ключевое слово `else`(иначе):

`if` (условие) (оператор1); `else` (альтернативный оператор).

*Оператор выполнения цикла с предусловием (пока):*

`while` (условие) [тело цикла].

*Оператор выполнения цикла с постусловием (выполняется, пока):*

`do` [тело цикла] `while`(условие).

*Оператор выполнения цикла, содержащий слово `for`, заголовок и тело цикла:*

`for` (начальные значения; условие; оператор) [тело цикла].

Цикл `for` используется тогда, когда количество повторений цикла заранее известно или может быть вычислено.

*Оператор безусловного перехода:*

`goto` [метка].

*Оператор выхода из цикла и перехода к следующему оператору:*

`break`.

*Оператор завершения текущего шага цикла и перехода к новому шагу (не выходя из цикла):*

`continue`.

*Оператор возврата из функции:*

`return`.

## Указатели

Указатель — это переменная, содержащая адрес другой переменной.

Применение указателя определяется следующими присвоениями:

```
rx = &x; //присвоение переменной rx адреса переменной x.
```

```
y = *rx; //присвоение переменной y величины,адрес которой
содержится в переменной rx.
```

## Подпрограммы

Подпрограмма (процедура, функция) — это именованная часть программы, к которой можно обращаться из других частей программы.

Функция имеет следующее оформление: имя функции, аргументы функции и тело функции в фигурных скобках. Перед именами функции и именами аргументов ставятся их типы.

Краткий пример функции `vuxod` может быть таким:

```
floatvuxod (int n, floatvhod) { тело функции }.
```

## Массивы

Массив - это конечная совокупность данных одного типа. Способы представления массивов поясним конкретными примерами:

```
int x[10]; //массив, состоящий из 10 целых чисел
(вектор)
```

```
int a[2][5]; //двумерный массив или матрица,
состоящая из 2 строк и 5 столбцов
```

В языке Си понятие массива тесно связано с понятием указателя. Имя массива само является указателем на первый элемент массива. Доступ к членам массива можно получать как через имя массива и индекс, так и через указатель на элемент массива и индекс.

## Структура (struct)

Структура - это совокупность данных одинакового или различного типа, обозначенная одним именем. Данные еще называют элементами.

В качестве примера структуры можно взять учетную карточку одного сотрудника предприятия. Элементами такой структуры являются: табельный номер сотрудника, его имя, пол, дата рождения, адрес. Некоторые из этих элементов сами могут оказаться структурами. Например, имя, дата рождения, адрес состоят из нескольких частей – элементов другого уровня других структур.

Часто объявляют в начале шаблон структуры, который затем используют для создания структур. Для примера с учетной карточкой шаблон структуры может быть таким:

```
struct anketa
{
    int tab_nom; // табельный номер
    char fio [30]; // фамилия, имя, отчество
    char pol [3]; // пол
    char data ; // дата рождения
    char adres;
```

```
};
```

Ключевое слово `struct` сообщает компилятору об объявлении шаблона структуры с именем `anketa`. Для того, чтобы создать структуру, например, с именем `anketa_1`, следует написать `struct anketa anketa_1;`

Созданную структуру с именем `anketa_1` называют структурной переменной или просто переменной. Когда объявлена структурная переменная, компилятор выделяет необходимый участок памяти для размещения всех ее элементов. При объявлении структуры можно одновременно объявить одну или несколько переменных, например,

```
struct anketa anketa_1, anketa_2, anketa_3;
```

По-другому, следом за закрывающей правой фигурной скобкой, заканчивающей список элементов, может следовать список структурных переменных:

```
struct anketa
{...} anketa_1, anketa_2;
```



Доступ к структурной переменной осуществляется с помощью оператора «точка»:

```
имя_структуры.имя_элемента;
```

### **Логические операторы сдвига**

*Оператор сдвига влево:* «<<». Когда оператор сдвига влево (<<) выполняется над некоторым значением, все биты, составляющие это значение, сдвигаются влево. Связанное с этим оператором число показывает количество бит, на которое значение должно переместиться. Биты, которые сдвигаются со старшего разряда, считаются потерянными, а на место младших битов всегда помещаются нули.

*Оператор сдвига вправо:* «>>». Операция сдвига вправо (>>) сдвигает разряды левого операнда вправо на количество позиций, указываемое правым операндом. Выходящие за правую границу разряды теряются. Для типов данных без знака (unsigned) освобождаемые слева позиции заполняются нулями. Для знаковых типов данных результат зависит от используемой системы. Освобождаемые позиции могут заполняться нулями либо копиями знакового (первого слева) разряда.

### **Структура программы**

Так как при программировании микроконтроллера мы будем часто прибегать к структурам и библиотекам, то в качестве повторения напишем программу, в которой будем заполнять и выводить на экран элемент типа структура. При этом саму структуру опишем в отдельном файле.

В программе опишем структуру `sklad`, которая будет хранить данные о хранящихся на складе овощах.

Проект будет состоять из трёх файлов:

`Main.c` – содержит функцию `main()`.

`Sklady.h` – содержит описание структуры и заголовки функций для работы с ней.

Sklady.c – содержит описание функций для работы со структурой.

Файл «sklady.h»

Структуры описываются следующим образом:

```
structsklad
{
    int carrot;
    int potato;
    int tomato;
    char * adress;
};
```

Gdesklad – имяструктуры. В фигурных скобках перечислены члены структуры. Переменные типа int будут хранить количество овощей на складе. Указатель на char по факту является строкой и будет хранить адрес склада.

Чтобы более удобно выводить хранящиеся в структуре данные, напишем для этого специальную функцию со следующим заголовком:

```
voidprintsklad(structsklad * sample);
```

В качестве аргумента функция берет указатель на элемент объявленной нами структуры.

Файл «Sklad.c»

В начале этого файла необходимо подключить файлы «sklady.h» и «stdio.h». Последний позволяет использовать функцию printf, которая будет выводить значения переменных в консоль. Для работы вывода данных необходимо в настройках IAR указать, что вывод сообщений от printf передается в окно TerminalIO через библиотеку семихостинга (диалог Options проекта ->GeneralOptions ->LibraryConfiguration ->Librarylow-levelinterfaceimplementation ->stdout/stderr ->Viasemihosting).

В этом файле опишем объявленную ранее функцию printsklad(), которая будет выводить адрес склада и количество килограмм каждого овоща на складе:

```
voidprintsklad(structsklad * sample)
{
```

```

printf("На складе по адресу %s
содержится:\n", (*sample).adress);
printf("Морковки: %d кг\n", (*sample).carrot);
printf("Картофеля: %d кг\n", (*sample).potato);
printf("Помидоров: %d кг\n", (*sample).tomato);
}

```

Функция `printf` работает по следующему принципу: текстовые сообщения передаются в кавычках (" "), далее, через запятую, идёт имя переменной, которая будет выводиться в данном сообщении. Сообщение пишется целиком, в место, где должно быть значение переменной вписывается специальный спецификатор (для целых чисел - `%d`, для символьных строк - `%s`).

Вывести на экран значения всех полей структуры нельзя, возможно лишь получать значения определённых членов структуры с помощью оператора «точка», например `sample.tomato`, где `sample` – имя элемента структуры, а `tomato` – имя переменной-члена структуры.

Так как функция принимает в качестве аргумента адрес структуры, при обращении к ней необходимо использовать оператор `"*"`, чтобы получить значение по адресу.

Файл «main.c»

В данном файле объявим элемент описанной ранее структуры:

```
structskladsLen;
```

Заполним значения полей структуры:

```
sLen.carrot=200;
```

```
sLen.potato=100;
```

```
sLen.tomato=50;
```

```
sLen.adress="Lenina";
```

И передадим её адрес в описанную ранее функцию с помощью оператора `"&"`:

```
printsklad(&sLen);
```

## Ход работы

### Задание на лабораторную работу

1. Ознакомиться с теоретическими сведениями.
2. Написать, отладить и запустить программу.
3. Выполнить индивидуальное задание.

### Текст программы

Файл «sklady.h»:

```
structsklad
{
    int number;
    int carrot;
    int potato;
    int tomato;
    char * adress;
};

void printsklad(structsklad * sample);
```

Файл «sklady.c»:

```
#include "sklady.h"
#include "stdio.h"

void printsklad(structsklad * sample)
{
    printf("На складе по адресу %s
содержится:\n", (*sample).adress);
    printf("Морковки: %d кг\n", (*sample).carrot);
    printf("Картофеля: %d кг\n", (*sample).potato);
    printf("Помидоров: %d кг\n", (*sample).tomato);
}
```

Файл «main.c»:

```
#include "sklady.h"

int main()
```

```
{  
structskladsLen;  
sLen.carrot=200;  
sLen.potato=100;  
sLen.tomato=50;  
sLen.adress="Lenina";  
  
printsklad(&sLen);  
  
return 0;  
}
```

### **Индивидуальные задания**

1. Перевоз овощей с двух складов на третий.

Дописать к имеющейся программе функцию, которая будет возвращать элемент структуры, содержащий сумму имеющихся овощей на двух складах, передаваемых функции в качестве аргументов.

2. Количество имеющегося картофеля.

В имеющейся программе объявить массив из 5 складов и посчитать количество имеющегося на них картофеля.

3. Сортировка овощей.

Дописать к имеющейся программе функцию, которой будут передаваться адреса 3-х элементов структуры. В результате на первом складе должна быть только сумма морковки с трёх складов, на втором – картофеля, на третьем – помидоров.

## **6. Лабораторная работа №2. Порты ввода/вывода (General-purpose input/output, GPIO).**

**Цель работы:** Изучить работы портов микроконтроллера на примере программы мигания светодиодами.

### **Теоретические сведения**

1986BE92x\_StdPeriph\_Driver — стандартная библиотека ввода-вывода, созданная компанией Фитон24 на языке Си для микроконтроллеров семейства Cortex-M производства Миландр. Содержит функции, структуры и макросы для облегчения работы с периферийными блоками микроконтроллеров. Библиотека документирована, включает примеры по каждому периферийному устройству, полностью поддерживает CMSIS (Cortex Microcontroller Software Interface Standard) и предоставляется компанией Миландр бесплатно.

CMSIS — стандартная библиотека для всех микроконтроллеров семейства Cortex-M.

Для работы с портами ввода/вывода используются библиотека MDR32F9Qx\_port.h, которая описывает следующие регистры:

- MDR\_PORTA
- MDR\_PORTB
- MDR\_PORTC
- MDR\_PORTD
- MDR\_PORTE
- MDR\_PORTF

Исходя из спецификации к отладочной плате светодиоды находятся на порте C (Таблица 2.1).

Кнопка UP находится на порте B, а кнопка DOWN на порте E (Рисунок 6.1).

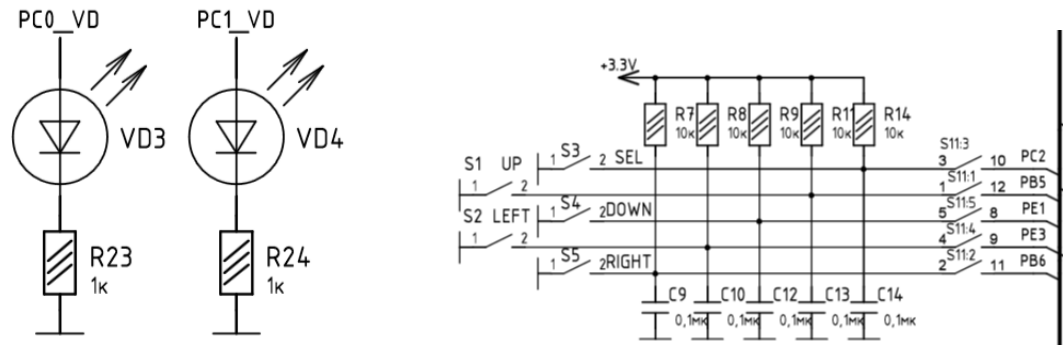


Рисунок 6.1 – Схема положения светодиодов и кнопок

В начале необходимо включить тактирование используемых портов (т.к на регистры портов должна поступить тактовая частота, иначе проект не будет работать) в данном случае В, С, Е.

```
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
```

Затем необходимо настроить порты ввода-вывода (см. спецификация 1986BE9X.pdf, стр.177, таблица 120)

Вывод	Аналоговая функция ANALOG_EN=0	Цифровая функция			
		Порт IO MODE[1:0]=00 ANALOG_EN=1	Основная MODE[1:0]=01 ANALOG_EN=1	Альтернативная MODE[1:0]=10 ANALOG_EN=1	Переопределенная MODE[1:0]=11 ANALOG_EN=1
		Порт А			
PA0	-	PA0	DATA0	1) EXT_INT1	9) -
PA1	-	PA1	DATA1	TMR1_CH1	TMR2_CH1
PA2	-	PA2	DATA2	TMR1_CH1N	TMR2_CH1N
PA3	-	PA3	DATA3	TMR1_CH2	TMR2_CH2
PA4	-	PA4	DATA4	TMR1_CH2N	TMR2_CH2N
PA5	-	PA5	DATA5	TMR1_CH3	TMR2_CH3
PA6	-	PA6	DATA6	CAN1_TX	2) UART1_RXD
PA7	-	PA7	DATA7	CAN1_RX	UART1_TXD
PA8	-	PA8	DATA8	TMR1_CH3N	TMR2_CH3N
PA9	-	PA9	DATA9	TMR1_CH4	TMR2_CH4
PA10	-	PA10	DATA10	nUART1DTR	10) TMR2_CH4N
PA11	-	PA11	DATA11	nUART1RTS	TMR2_BLK
PA12	-	PA12	DATA12	nUART1RI	TMR2_ETR
PA13	-	PA13	DATA13	nUART1DCD	TMR1_CH4N
PA14	-	PA14	DATA14	nUART1DSR	TMR1_BLK
PA15	-	PA15	DATA15	nUART1CTS	TMR1_ETR
Порт В					
PB0	-	PB0 JA_TDO	DATA16	1) TMR3_CH1	UART1_TXD
PB1	-	PB1 JA_TMS	DATA17	TMR3_CH1N	UART2_RXD
PB2	-	PB2 JA_TCK	DATA18	TMR3_CH2	CAN1_TX
PB3	-	PB3 JA_TDI	DATA19	TMR3_CH2N	CAN1_RX
PB4	-	PB4 JA_TRST	DATA20	TMR3_BLK	TMR3_ETR
PB5	-	PB5	DATA21	UART1_TXD	10) TMR3_CH3
PB6	-	PB6	DATA22	UART1_RXD	TMR3_CH3N
PB7	-	PB7	DATA23	nSIROUT1	TMR3_CH4
PB8	-	PB8	DATA24	COMP_OUT	7) TMR3_CH4N
PB9	-	PB9	DATA25	nSIRIN1	10) EXT_INT4
PB10	-	PB10	DATA26	EXT_INT2	9) nSIROUT1
PB11	-	PB11	DATA27	EXT_INT1	COMP_OUT
PB12	-	PB12	DATA28	SSP1_FSS	SSP2_FSS
PB13	-	PB13	DATA29	SSP1_CLK	SSP2_CLK

Рисунок 6.2 – Функции портов

Исходя из таблицы, мы видим, что у портов микроконтроллера есть аналоговая и цифровая функция.

Аналоговая отвечает за блоки АЦП, ЦАП.

Цифровая функция порта разделена на несколько видов. Основная, альтернативная и переопределенная отвечают за взаимодействие внутренних периферийных компонентов с выводами МК.

Для данной лабораторной работы необходима колонка таблицы, которая отвечает за использование портов как «Порт ИО»

Микроконтроллер имеет 6 портов ввода/вывода. Порты 16-разрядные и их выводы мультиплексируются между различными функциональными блоками, управление для каждого вывода отдельное. Для того, чтобы выводы порта перешли под управление того или иного периферийного блока, необходимо задать для нужных выводов выполняемую функцию и настройки.

Таблица 6.1 - Описание регистров портов ввода-вывода

Название	Описание	
MDR_PORTA	Порт A	
MDR_PORTB	Порт B	
MDR_PORTC	Порт C	
MDR_PORTD	Порт D	
MDR_PORTE	Порт E	
MDR_PORTF	Порт F	
RXTX[15:0]	MDR_PORTx->RXTX	Данные порта
OE[15:0]	MDR_PORTx->OE	Направление порта
FUNC[31:0]	MDR_PORTx->FUNC	Режим работы порта
ANALOG[15:0]	MDR_PORTx->ANALOG	Аналоговый режим работы порта
PULL[31:0]	MDR_PORTx->PULL	Подтяжка порта
PD[31:0]	MDR_PORTx->PD	Режим работы выходного драйвера



Название	Описание	
PWR[31:0]	MDR_PORTx->PWR	Режим мощности передатчика
GFEN[31:0]	MDR_PORTx->GFEN	Режим работы входного фильтра

PORT\_Pin – выбор выводов для инициализации

**Регистр OE** - определяет режим работы порта (*направление передачи данных*):

- ввод PORT\_OE\_IN (0);
- вывод PORT\_OE\_OUT (1)

**Регистр FUNC** - определяет режим работы вывода порта:

- Порт PORT\_FUNC\_PORT (0);
- Основная функция PORT\_FUNC\_MAIN (1);
- Альтернативная функция PORT\_FUNC\_ALTER (2);
- Переопределенная функция PORT\_FUNC\_OVERRID (3)

**Регистр MODE** - определяет режим работы контроллера:

- аналоговый PORT\_MODE\_ANALOG(0);
- цифровой PORT\_MODE\_DIGITAL(1)

**Регистр SPEED**- определяет скорость работы порта:

- зарезервировано (передатчик отключен) PORT\_OUTPUT\_OFF (0);
- медленный фронт (порядка 100 нс) PORT\_SPEED\_SLOW (1);
- быстрый фронт (порядка 20 нс) PORT\_SPEED\_FAST (2);
- максимально быстрый фронт (порядка 10 нс) PORT\_SPEED\_MAXFAST (3);

Для инициализации используется структура типа PORT\_InitTypeDef, поэтому необходимо объявить переменную данного типа:

```
static PORT_InitTypeDef PortInit;
//Объявляем структуру для конфигурации порта
```

Функция настройки порта будет выглядеть следующим образом:

```
voidLedPinGfg (void)
{
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE); //Включаем
тактирование порта C
    PortInit.PORT_Pin    = PORT_Pin_1; //Устанавливаем номер
вывода порта
    PortInit.PORT_OE     = PORT_OE_OUT;      //Направление передачи
данных - на выход
    PortInit.PORT_FUNC= PORT_FUNC_PORT; //Режимработы - порт
    PortInit.PORT_MODE = PORT_MODE_DIGITAL; //Режимработы -
цифровой
    PortInit.PORT_SPEED = PORT_SPEED_SLOW; //Скоростьработы -
медленныйрежим

    PORT_Init(PORTC, &PortInit); // ПередаемструктурупортуC
}
```

Функции состояния вывода:

PORT\_SetBits() - установить на передаваемом в аргументе выводе единицу.

PORT\_ResetBits() - установить на передаваемом в аргументе выводе ноль.

## Ход работы

**Структура программы:**

- Включаем тактирование периферии
- Настраиваем порты ввода-вывода
- Начинаем выполнять основной код программы

**Исходный код программы:**

```

#include "1986be9x_config.h"
#include "1986BE9x.h"
#include "1986BE9x_port.h"
#include "1986BE9x_rst_clk.h"
// Объявляем структуру для конфигурации порта
static PORT_InitTypeDef PortInit;

void LedPinGfg (void)
{
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE); //Включаем
тактирование порта C
    PortInit.PORT_Pin    = PORT_Pin_1; //Устанавливаем номер
вывода порта
    PortInit.PORT_OE      = PORT_OE_OUT; //Устанавливаем направление
передачи данных - на выход
    PortInit.PORT_FUNC= PORT_FUNC_PORT;    //Режим работы - порт
    PortInit.PORT_MODE= PORT_MODE_DIGITAL; //Режим работы -
цифровой
    PortInit.PORT_SPEED = PORT_SPEED_SLOW; //Скорость работы -
медленный режим
    PORT_Init(PORTC, &PortInit); //Передаем структуру порту C
}

int main (void)
{
    LedPinGfg (); // инициализация порта C
    while (1) {
        PORT_SetBits(PORTC, PORT_Pin_1); //включение светодиода
        for (uint32_t i=0; i<1000000; i++) {} //задержка
        PORT_ResetBits(PORTC, PORT_Pin_1); //отключение светодиода
        for (uint32_t i=0; i<1000000; i++) {}
    }
}

```

**Индивидуальные задания**

1. Реализовать работу двух светодиодов:

- В ходе работы микроконтроллера светодиоды должны загораться одновременно;
- В ходе работы микроконтроллера светодиоды должны загораться по очереди.

2. Реализовать работу кнопки:

- При нажатии кнопки LEFT загорается левый светодиод, при повторном нажатии он затухает;
- При нажатии кнопки RIGHT загорается правый светодиод, при повторном нажатии он затухает;
- При нажатии кнопки UP светодиоды загораются по очереди, при повторном нажатии они потухают;
- При нажатии кнопки SELECT светодиоды загораются одновременно, при повторном нажатии они потухают.

## **7. Лабораторная работа №3. Использование таймера**

**Цель работы:** Изучить работу системного таймера SysTick.

### **Теоретические сведения**

В предыдущей лабораторной работе в качестве задержки был использован пустой цикл, что не совсем корректно. Для реализации задержки можно использовать таймеры, встроенные в микроконтроллер.

Таймеры - один из основных типов периферии микроконтроллера. Они используются для организации временных задержек, выполнения каких-то периодических событий, генерации ШИМ и различных других применений, жестко связанных со временем.

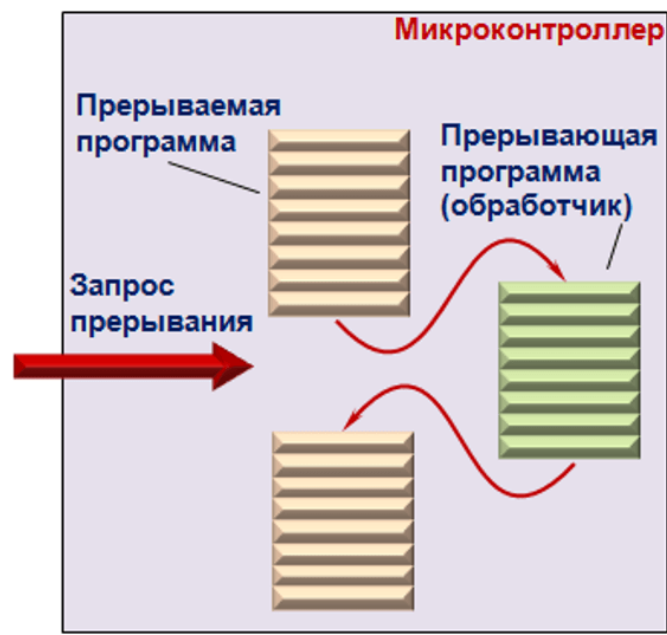
Микроконтроллер 1986BE92У содержит три 16-разрядных таймера с 4 каналами схем захвата и ШИМ с функциями формирования «мертвой зоны» и аппаратной блокировки, а также системный 24-х разрядный таймер и два сторожевых таймера.

В данной лабораторной работе необходимо освоить системный таймер SysTick.

Процессор имеет 24-х разрядный системный таймер, SysTick, который считает вниз от загруженного в него значения до нуля; перезагрузка (возврат в начало) значения в регистр LOAD происходит по следующему фронту синхросигнала, затем счёт продолжается по последующему фронту. Когда процессор остановлен для отладки, таймер не декрементируется.

При работе с различной периферией, в том числе и таймерами, часто используют прерывания.

*Прерывание* – событие, требующие немедленной реакции со стороны процессора. Реакция состоит в том, что процессор прерывает обработку текущей программы и переходит к выполнению некоторой другой программы, специально предназначенной для данного события. По завершении этой программы процессор возвращается к выполнению прерванной программы.



Настройка:

**Описание регистров системного таймера SysTick**

Таблица 61 – Описание регистров системного таймера SysTick

Адрес	Название	Тип	Доступ	Значение после сброса	Описание
0xE000E010	SysTick				Системный таймер SYSTICK
0xE000E010	CTRL	RW	привилегированный	0x00000004	SysTick->CTRL
0xE000E014	LOAD	RW	привилегированный	0x00000000	SysTick->LOAD
0xE000E018	VAL	RW	привилегированный	0x00000000	SysTick->VAL
0xE000E01C	CALIB	RO	привилегированный	0x00002904 <sup>(1)</sup>	SysTick->CAL

<sup>1)</sup> Калибровочное значение системного таймера.

Всего четыре регистра, но понадобятся нам всего два из них. Рассмотрим их поподробнее.

**SysTick->CTRL**

Регистр CTRL разрешает основные функции системного таймера.

Назначение бит:

Таблица 62 – Регистр контроля и статуса CTRL

Номер	31...17	16	15...3	2	1	0
Доступ						
Сброс						
	-	COUNTFLAG	-	CLKSOURCE	TICKINT	ENABLE

**COUNTFLAG**

Возвращает 1, если таймер досчитал до нуля с последнего момента чтения.

**CLKSOURCE**

Указывает источник синхросигнала:

0 - LSI

1 - HCLK

**TICKINT**

Разрешает запрос на прерывание от системного таймера:

0 - таймер досчитает до нуля и прерывание не возникнет;

1 - таймер досчитывает до нуля и возникает запрос на прерывание.

Программное обеспечение может использовать бит COUNTFLAG, чтобы определить, досчитал таймер до нуля или нет.

**ENABLE**

Разрешает работу таймера:

0 - работа таймера запрещена;

1 - работа таймера разрешена.

Когда ENABLE установлен в единицу, таймер загружает значение RELOAD из регистра LOAD и затем начинает декрементироваться. По достижению значения 0 таймер устанавливает бит COUNTFLAG и в зависимости от TICKINT генерирует запрос на прерывание. Затем загружается значение RELOAD и продолжается счёт.

Необходимо выбрать источник тактового сигнала, разрешить прерывания и включить счетчик. Тактировать таймер необходимо от HCLK (частота тактирования 8 МГц). Прерывания нам будут нужны для создания собственной функции точной задержки.

```
#define CLKSOURCE (1<<2) //Указывает источник синхросигнала: 0
- LSI, 1 - HCLK.
```

```
#define TICKINT (1<<1) //Разрешает запрос на прерывание от
системного таймера.
```

```
#define ENABLE (1<<0) //Разрешает работу таймера.
```

Создаем отдельную функцию настройки таймера и вписываем в нее нашу конструкцию.

```
void Init_SysTick (void)
{
```

```

SysTick->CTRL |= CLKSOURCE|TCKINT|ENABLE;
}

```

### SysTick->LOAD

Регистр LOAD устанавливает стартовое значение, загружаемое в регистр VAL.

Таблица 63 – Регистр перегружаемого значения LOAD

Номер	31...24	23...0
Доступ		
Сброс		
	-	RELOAD

### RELOAD

Значение, загружаемое в регистр VAL, когда таймер разрешён и когда достигается значение нуля.

#### Расчёт значения RELOAD

Значение RELOAD может быть любым в диапазоне 0x00000001–0x00FFFFFF. Значение 0 допустимо, но не оказывает эффекта, потому что запрос на прерывание и активизация бита COUNTFLAG происходит только при переходе таймера из состояния 1 в 0.

Расчёт значения RELOAD происходит в соответствии с использованием таймера:

- Для формирования мультикороткого таймера с периодом N процессорных циклов применяется значение RELOAD, равное N-1. Например, если требуется прерывание каждые 100 циклов, то устанавливается значение RELOAD, равное 99;
- Для формирования одиночного прерывания после задержки в N тактов процессора используется значение N. Например, если требуется прерывание после 400 тактов процессора, то устанавливается RELOAD, равное 400.

В данный регистр необходимо загрузить значение задержки. Настроим таймер на прерывание раз в одну миллисекунду. Таймер настроен на тактирование от HCLK – это частота тактирования ядра микроконтроллера. По умолчанию она равна 8 МГц = 8000000 тактов в секунду. В одной секунде тысяча миллисекунд => 8000000(количество тактов в секунду)/1000(количество миллисекунд в секунде) = количество тактов в одной миллисекунде. Необходимо отнять «1», как описано на рисунке выше.

```

SysTick->LOAD = (8000000/1000)-1;

```

Разместим данную настройку в функции Init\_SysTick перед включением таймера.

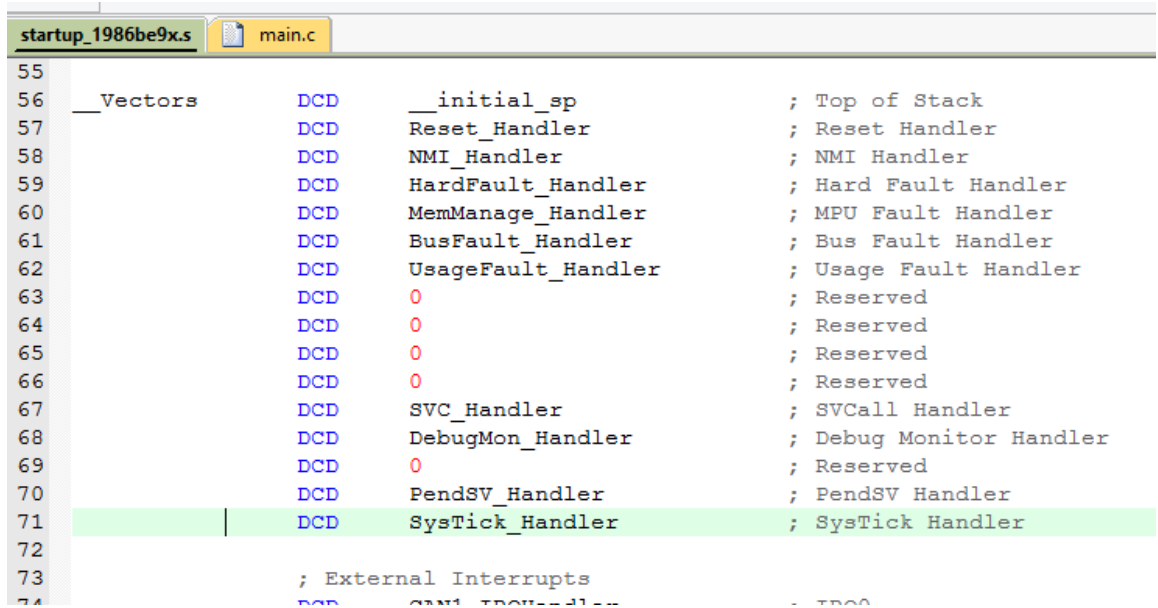
```

void Init_SysTick (void)
{
    SysTick->LOAD |= (8000000/1000)-1;
    SysTick->CTRL |= CLKSOURCE|TCKINT|ENABLE;
}

```



Мы настроили таймер на появление прерываний, но необходимо действие, которое будет происходить, когда данное прерывание появится. В стартап файле находим вектора прерываний:



```

55
56 __Vectors      DCD      __initial_sp          ; Top of Stack
57                DCD      Reset_Handler        ; Reset Handler
58                DCD      NMI_Handler          ; NMI Handler
59                DCD      HardFault_Handler    ; Hard Fault Handler
60                DCD      MemManage_Handler    ; MPU Fault Handler
61                DCD      BusFault_Handler     ; Bus Fault Handler
62                DCD      UsageFault_Handler    ; Usage Fault Handler
63                DCD      0                     ; Reserved
64                DCD      0                     ; Reserved
65                DCD      0                     ; Reserved
66                DCD      0                     ; Reserved
67                DCD      SVC_Handler          ; SVC Call Handler
68                DCD      DebugMon_Handler     ; Debug Monitor Handler
69                DCD      0                     ; Reserved
70                DCD      PendSV_Handler        ; PendSV Handler
71                DCD      SysTick_Handler       ; SysTick Handler
72
73                ; External Interrupts
74                DCD      0                     ; EXTI Interrupts

```

Когда появляется какое-то внешнее/внутреннее неотложное событие (прерывание), микроконтроллер прерывает выполнение основной программы, переходит к этой таблице и смотрит, куда ему нужно перейти дальше. Например, когда появляется прерывание от нашего таймера, он переходит к пункту с именем «SysTick\_Handler». В случае, если вектор не прописан нами в программе (нет функции с таким именем) – контроллер игнорирует его и продолжает выполнение своей программы. Но если в программе есть функция с этим именем, то он переходит к ее выполнению.

В файле main.c создадим функцию с именем SysTick\_Handler. Объявим глобальную переменную 32-х битной разрядности. Для того, чтобы компилятор не оптимизировал данную переменную, перед ней добавляем «volatile» (это необходимо во избежание ошибок). В данной функции прописываем условие: если переменная еще не равна нулю – отнять 1. Таким образом, каждую миллисекунду контроллер будет останавливать основную программу и проверять, если еще счетчик не пуст – отнять от него 1.

```

volatile uint32_t Delay_dec = 0;
void SysTick_Handler (void)

```

```

{
    if (Delay_dec) Delay_dec--;
}

```

Создадим функцию задержки, которая будет использовать данное прерывание, которая принимает длительность задержки (в миллисекундах), копирует ее в переменную, из которой по прерываниям системного таймера функция отнимает 1 каждую миллисекунду. После того, как задержка исчерпана, программа продолжает свою работу.

```

void Delay_ms (uint32_t Delay_ms_Data)
{
    Delay_dec = Delay_ms_Data;
    while (Delay_dec) {};
}

```

**Исходный код программы:**

```

#include "1986be9x_config.h"
#include "1986BE9x.h"
#include "1986BE9x_port.h"
#include "1986BE9x_rst_clk.h"

void Led_init (void)
{
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
    PortInit.PORT_Pin    = PORT_Pin_1;
    PortInit.PORT_OE     = PORT_OE_OUT;
    PortInit.PORT_FUNC= PORT_FUNC_PORT;
    PortInit.PORT_MODE= PORT_MODE_DIGITAL;
    PortInit.PORT_SPEED = PORT_SPEED_SLOW;

    PORT_Init(PORTC, &PortInit);
}

#define CLKSOURCE (1<<2) //Указывает источник синхросигнала: 0
- LSI, 1 - HCLK.

#define TCKINT (1<<1) //Разрешает запрос на прерывание от
системного таймера.

#define ENABLE (1<<0) //Разрешает работу таймера.

void Init_SysTick (void) //Прерывание раз в миллисекунду.
{
    SysTick->LOAD |= (8000000/1000)-1;
    SysTick->CTRL |= CLKSOURCE|TCKINT|ENABLE;
}

volatile uint32_t Delay_dec = 0;
void SysTick_Handler (void)
{
    if (Delay_dec) Delay_dec--;
}

void Delay_ms (uint32_t Delay_ms_Data)

```

```

{
    Delay_dec = Delay_ms_Data;
    while (Delay_dec) {};
}

int main (void)
{
    Init_SysTick(); //Инициализируем системный таймер.
    Led_init(); //Инициализируем ножку 0 порта C для
светодиода.
    while (1)
    {
        PORTC->RXTX |= 1;
        Delay_ms (5000);
        PORTC->RXTX = 0;
        Delay_ms (5000);
    }
}

```

### Индивидуальные задания

#### 1. Реализовать работу двух светодиодов:

- В ходе работы микроконтроллера светодиоды должны загораться одновременно с задержкой (номер вариант секунд);
- В ходе работы микроконтроллера светодиоды должны загораться по очереди с задержкой (номер вариант секунд).
- В ходе работы микроконтроллера светодиоды должны загораться по очереди: первый с задержкой (номер вариант секунд), второй с задержкой ( номер варианта / 2 + 1 секунд).

## 8. Лабораторная работа №4. Универсальный приемопередатчик (USART)

**Цель работы:** Научиться использовать универсальный синхронный/асинхронный приемопередатчик, организовать передачу данных данным устройством.

### Теоретические сведения

**Универсальный асинхронный приёмопередатчик** (УАПП, англ. *Universal Asynchronous Receiver-Transmitter, UART*) — узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами. Преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по цифровой линии другому аналогичному устройству. Метод преобразования хорошо стандартизован и широко применяется в компьютерной технике.

Представляет собой логическую схему, с одной стороны подключённую к шине вычислительного устройства, а с другой имеющую два или более выводов для внешнего соединения.

Для работы с модулем UART необходимо подключить нуль-модемный кабель (Рисунок 8.1) одним концом к порту RS-232 отладочной платы (Рисунок 2.2 18-й элемент), другим — к персональному компьютеру через преобразователь порта RS-232 в USB порт (Рисунок 8.2), или же напрямую к порту RS-232, если таковой имеется.



Рисунок 8.1 - Нуль-модемный кабель



Рисунок 8.2 – Кабель преобразователь RS-232 в USB

### Взаимодействие между отладочной платой и персональным компьютером

Для того что бы принимать и отправлять данные через порт RS-232 (COM порт) необходимо использовать стороннее программное обеспечение, такое как терминальные программы для интерфейса RS-232. В данной лабораторной работе рекомендуется использовать свободную программу Terminal 1.9b (Рисунок 8.3).

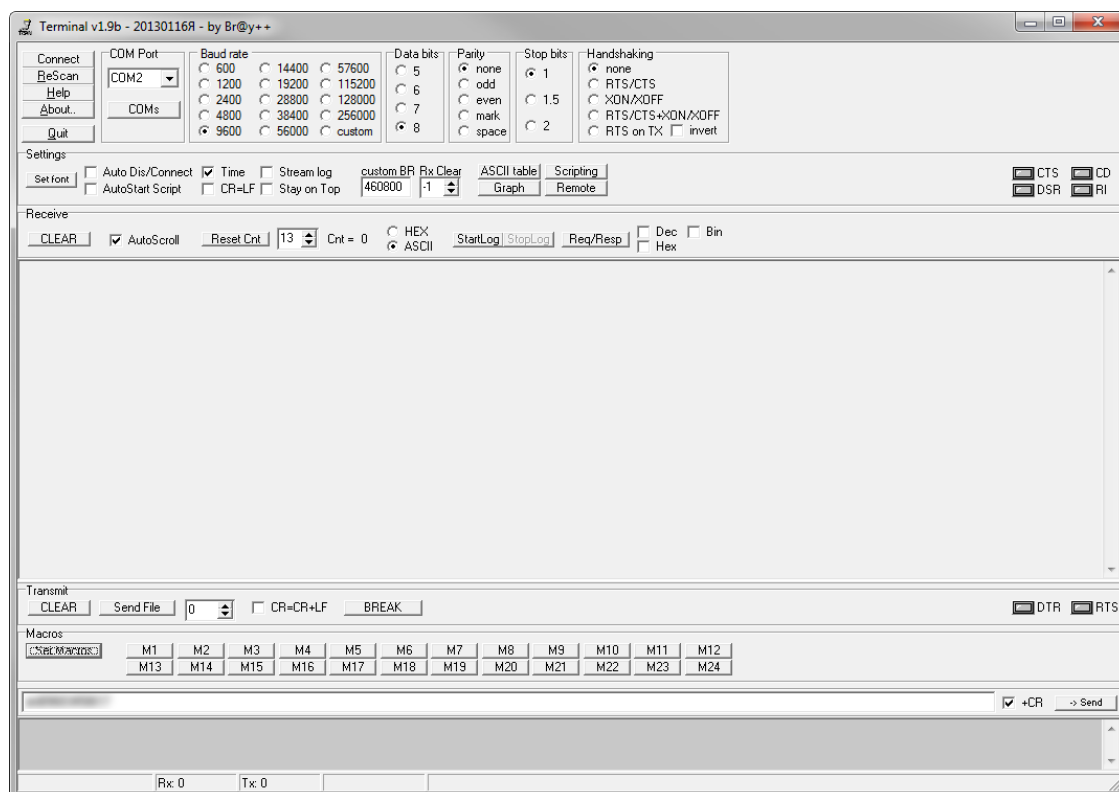


Рисунок 8.3 – Внешний вид программы «Terminal 1.9b»

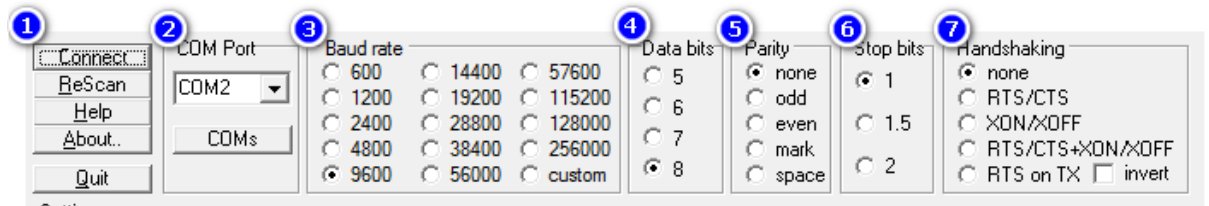


Рисунок 8.4 – Параметры подключения

Описание параметров подключения:

1. Функциональные кнопки: **Connect** — кнопка для открытия COM-порта, **Rescan** — пересканировать список COM-портов, **Help** — справка, **About..** — о программе, **Quit** — выход из программы;
2. Поле выбора номера COM-порта для подключения;
3. Выбор скорости COM-порта;
4. Выбор количества бит данных;
5. Выбор четности;
6. Выбор количества стоповых бит;
7. Выбор типа управления потоком.

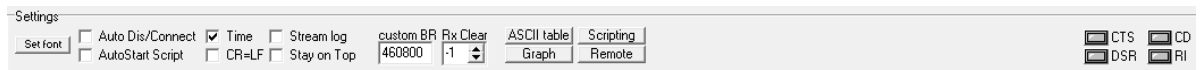


Рисунок 8.5 – Дополнительные параметры



Рисунок 8.6 – Параметры отображения сообщения от устройства

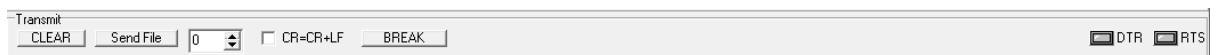


Рисунок 8.7 – Параметры передачи данных на устройство



Рисунок 8.8 – Поле для отправки сообщения

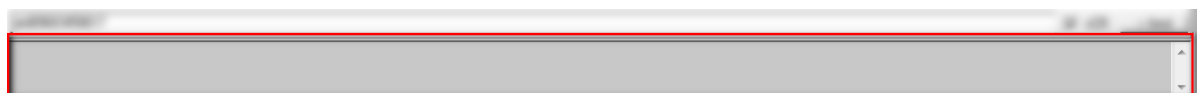


Рисунок 8.9 – Поле отображения отправленных сообщений

**Индивидуальные задания**

Реализовать программу, в которой микроконтроллер опрашивает кнопку, при нажатии которой контроллер вышлет в USART сообщение «Pressed», а если кнопка будет не нажата, то «NotPressed».



**Исходный код программы:**

```

#include "1986be9x_config.h"
#include "1986BE9x.h"
#include "1986BE9x_uart.h"
#include "1986BE9x_port.h"
#include "1986BE9x_rst_clk.h"
#include "1986BE9x_it.h"
#include "MilFlash.h"

void LedPinGfg (void)
{
    PORT_InitTypeDef PortInit;
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
    PortInit.PORT_Pin    = PORT_Pin_1;
    PortInit.PORT_OE     = PORT_OE_OUT;
    PortInit.PORT_FUNC   = PORT_FUNC_PORT;
    PortInit.PORT_MODE   = PORT_MODE_DIGITAL;
    PortInit.PORT_SPEED  = PORT_SPEED_SLOW;
    PORT_Init(PORTC, &PortInit);
}

static UART_InitTypeDef UART_InitStructure;

void Uart2PinCfg(void)
{
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTF, ENABLE);
    PORT_InitTypeDef PortInit;
    PortInit.PORT_PULL_UP = PORT_PULL_UP_OFF;
    PortInit.PORT_PULL_DOWN = PORT_PULL_DOWN_OFF;
    PortInit.PORT_PD_SHM = PORT_PD_SHM_OFF;
    PortInit.PORT_PD = PORT_PD_DRIVER;
    PortInit.PORT_GFEN = PORT_GFEN_OFF;
    PortInit.PORT_FUNC = PORT_FUNC_OVERRID;
    PortInit.PORT_SPEED = PORT_SPEED_MAXFAST;
}

```

```

PortInit.PORT_MODE = PORT_MODE_DIGITAL;

PortInit.PORT_OE = PORT_OE_IN;
PortInit.PORT_Pin = PORT_Pin_0;
PORT_Init(PORTF, &PortInit);

PortInit.PORT_OE = PORT_OE_OUT;
PortInit.PORT_Pin = PORT_Pin_1;
PORT_Init(PORTF, &PortInit);
}

void Uart2Setup(void)
{
    /* Select HSI/2 as CPU_CLK source*/
    RST_CLK_CPU_PLLconfig (RST_CLK_CPU_PLLsrcHSIdiv2,0);
    /* Enables the CPU_CLK clock on UART2 */
    RST_CLK_PCLKcmd(RST_CLK_PCLK_UART2, ENABLE);
    /* Set the HCLK division factor = 1 for UART2*/
    UART_BRGInit(UART2, UART_HCLKdiv1);

    UART_InitStructure.UART_BaudRate= 9600;
    UART_InitStructure.UART_WordLength= UART_WordLength8b;
    UART_InitStructure.UART_StopBits = UART_StopBits1;
    UART_InitStructure.UART_Parity = UART_Parity_No;
    UART_InitStructure.UART_FIFOMode = UART_FIFO_OFF;
    UART_InitStructure.UART_HardwareFlowControl =
UART_HardwareFlowControl_RXE | UART_HardwareFlowControl_TXE;

    UART_Init (UART2,&UART_InitStructure);
    UART_Cmd(UART2,ENABLE);
}

void ButtonPinGfg (void)
{
    PORT_InitTypeDefPortInit;

```

```

RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE); //
Включаем тактирование порта C - кнопка выбора
    /* Конфигурируем порт Спин 2 */
    PortInit.PORT_Pin    = (PORT_Pin_2);
    PortInit.PORT_OE     = PORT_OE_IN;
    PortInit.PORT_FUNC   = PORT_FUNC_PORT;
    PortInit.PORT_MODE   = PORT_MODE_DIGITAL;
    PortInit.PORT_SPEED  = PORT_SPEED_SLOW;

    PORT_Init(PORTC, &PortInit);
}

int main (void)
{
    LedPinGfg();
    ButtonPinGfg();
    Uart2PinCfg();
    Uart2Setup();

    uint16_t data;
    int flag = 0;

    while(1)
    {
        if (!PORT_ReadInputDataBit(PORTC, PORT_Pin_2)) //
        обработка нажатия кнопки select (выбор)
        {
            PORT_SetBits(PORTC, PORT_Pin_1); // включение светодиода
            if(data == 0) {
                data = 1;
                flag = 0;
            }
        }
        else
        {

```

```
PORT_ResetBits(PORTC, PORT_Pin_1); // отключение светодиода
if(data == 1) {
    data = 0;
    flag = 1;
        }
    }

if(flag == 1) {
    UART_SendData (UART2,data);
    flag = 0;
}
    }
}
```

## 9. Список литературы

1. А. В. Пуговкин, А. В. Бойченко, Р. В. Губарева, Е. С. Сорокина, А. М. Мукашев. Методическое пособие по программированию микроконтроллеров. Томск 2015.
2. Недяк С.П., Шаропин Ю.Б. Лабораторный практикум по микроконтроллерам семейства Cortex-M. Методическое пособие по проведению работ. - Томск: ТУСУР, 2013. - 77 с.
3. Коллективный блог Хабрахабр. Блог пользователя Vadimotorikda. [электронный ресурс]. – Режим доступа <https://habrahabr.ru/users/vadimotorikda/topics/> (дата обращения 10.08.2016)
4. Быстрый старт для K1986BE92QI.
5. Спецификация Микросхем серии 1986BE9ху, K1986BE9ху, K1986BE9хуK K1986BE92QI, K1986BE92QC, 1986BE91H4, K1986BE91H4, 1986BE94H4, K1986BE94H4. – Миландр: Версия 3.8.0 - 08.09.2015 – 518 с.
6. Демонстрационно-отладочная плата 1986EvBrd\_64. Техническое описание. – Миландр: Версия 1.0 – 25.05.2010 – 9 с.