

## Навигация

### Глава I. Микроконтроллеры серии 1986BE9х

<b>и среда программирования Keil <math>\mu</math>Vision .....</b>	<b>2</b>
1.1. Сведения о микроконтроллерах серии 1986BE9х .....	2
1.2. Сведения об отладочной плате .....	5
1.3. Подготовка отладочной платы к работе .....	9
1.4. Среда программирования Keil $\mu$ Vision .....	9
1.5. Программный проект .....	11
1.5.1. Создание проекта .....	11
1.5.2. Настройка проекта.....	13
1.5.3. Разработка модуля.....	20
1.5.4. Построение проекта .....	27
1.5.5. Загрузка программы в память микроконтроллера.....	28
1.5.6. Внутрисхемная отладка .....	30
Задачи для самостоятельной работы.....	34
Контрольные вопросы .....	34

## Глава I

### Микроконтроллеры серии 1986BE9х и среда программирования Keil $\mu$ Vision

#### Цель работы:

- знакомство с микроконтроллерами серии 1986BE9х;
- определение состава отладочных комплектов;
- получение навыков работы в среде Keil  $\mu$ Vision;
- получение представлений о структуре программного проекта на языке Си.

#### Оборудование:

- отладочный комплект для микроконтроллера 1986BE92У;
- программатор-отладчик J-LINK (или аналог);
- персональный компьютер.

#### Программное обеспечение:

- операционная система Windows 7 / 8 / 10;
- среда программирования Keil  $\mu$ Vision MDK-ARM 5.24;
- драйвер программатора J-LINK;
- примеры кода программ.

#### 1.1. Сведения о микроконтроллерах серии 1986BE9х

Микроконтроллеры серии 1986BE9х являются мощными управляющими микросхемами, построенными на базе высокопроизводительного процессорного RISC-ядра с архитектурой **ARM Cortex-M3** и содержащими развитый набор периферийных блоков. Широкий функционал данных микроконтроллеров делает их универсальными элементами широкого спектра применения: от бытовых приборов до специальной техники и систем повышенной надежности.

Микроконтроллеры серии 1986BE9х разрабатываются и производятся отечественной компанией «Миландр». В 2008 году компания «Миландр» **первой в России** получила лицензию на использование микропроцессорного ядра ARM в микроконтроллерах собственной разработки. Серийный выпуск микросхем был начат в 2010 году.

К серии 1986BE9х относятся микроконтроллеры:

- 1986BE91Т (1986BE94Т);
- 1986BE92У (К1986BE92QI);
- 1986BE93У.

Различие между перечисленными микроконтроллерами заключается только в количестве линий ввода-вывода и периферийных блоков [x]. Так, например, 1986BE91Т имеет 132 вывода, а 1986BE92У – 64. Все микросхемы этой серии, за исключением К1986BE92QI, изготавливаются в металлокерамических корпусах; микросхемы К1986BE92QI помещают в пластмассовый корпус.

На рисунке 1.1 приведена структурная схема микроконтроллеров данной серии.

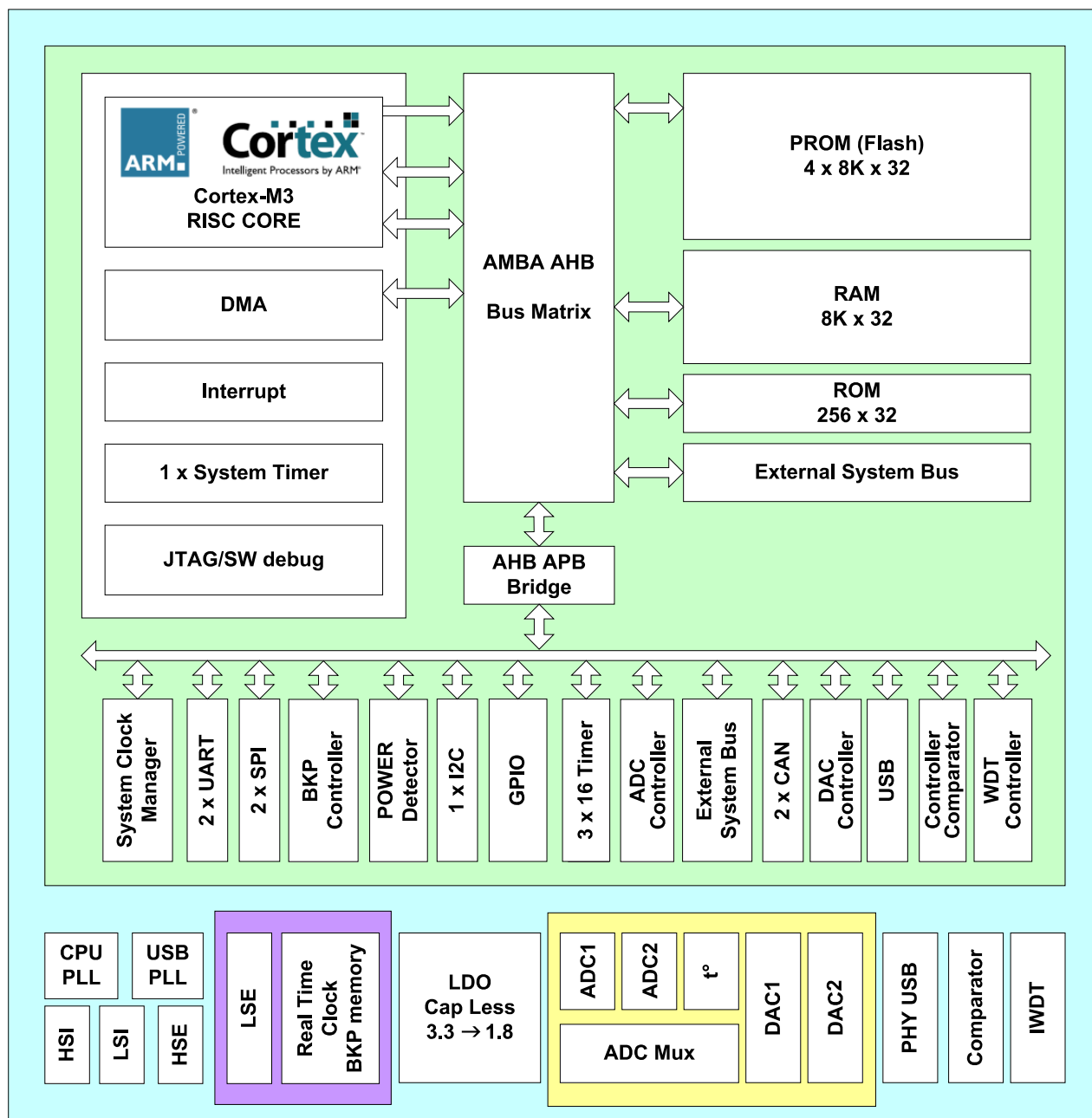


Рисунок 1.1 – Структурная схема микроконтроллеров серии 1986BE9х

Буквенно-цифровое наименование микроконтроллеров состоит из 5 элементов [x]. В таблице 1.1 приведена интерпретация названия на примере микросхемы 1986BE92У.

Таблица 1.1 – Интерпретация кодового обозначения микросхемы 1986BE92У

№ п/п	Элемент	Назначение	Расшифровка
1	–	Назначение микросхемы	Отсутствие элемента указывает на то, что микросхема является специализированной; К – микросхема широкого применения
2	1	Группа микросхемы по конструктивно-технологическому признаку	Полупроводниковая микросхема
3	986	Порядковый номер разработки серии	–
4	BE	Функциональное назначение микросхемы	Вычислительное устройство: микро-ЭВМ
5	92У	Условный порядковый номер микросхемы среди однотипных	–

Ниже приведены основные характеристика микроконтроллеров серии 1986BE9х:

- разрядность шины данных – 32 бита;
- процессорное ядро – ARM Cortex-M3;
- объем флеш-памяти программ – 128 Кбайт;
- объем статической оперативной памяти данных – 32 Кбайт;
- тактовая частота процессорного ядра – до 80 МГц;
- ток потребления в активном режиме (80 МГц) – 120 мА;
- напряжение питания – 2.2...3.6 В;

В рамках данного практикума будет использован микроконтроллер 1986BE92У.

В его состав входят следующие периферийные устройства:

- 43 линии ввода-вывода общего назначения, объединенные в 6 портов;
- 2 аналого-цифровых преобразователя (12 бит, 8 каналов);
- 1 цифро-аналоговый преобразователь (12 бит);
- контроллер прямого доступа к памяти;
- 3 аппаратных таймера общего назначения (по 16 бит каждый);
- часы реального времени;
- 1 контроллер интерфейса USB 2.0;
- 2 контроллера интерфейса UART;
- 2 контроллера интерфейса SPI;
- 1 контроллер интерфейса I<sup>2</sup>C;
- 2 контроллера интерфейса CAN.

Подробное описание микроконтроллера 1986BE92У приведено в фирменной документации АО «ПКК Миландр» [x].

Ближайшими функциональными аналогами целевых микроконтроллеров являются широко известные микросхемы серии STM32F103х зарубежной компании STMicroelectronics; поэтому программистам, знакомым с данными микросхемами, будет значительно легче освоить микроконтроллеры серии 1986BE9х.

## 1.2. Сведения об отладочной плате

Отладочная плата для микроконтроллера 1986BE92У, выпускаемая компанией «Миландр», предназначена для ознакомления с возможностями микроконтроллера и отладки программного обеспечения для него. Внешний вид отладочной платы показан на рисунке 1.2.



Рисунок 1.2 – Внешний вид отладочной платы для микроконтроллера 1986BE92У

Питание платы, как правило, осуществляется от внешнего блока питания, входящего в состав отладочного комплекта. Для этого на плате предусмотрен соответствующий разъем. Блок питания подключается к сети переменного тока ~220 В, 50 Гц и выдает постоянное напряжение +5 В при токе до 0.5 А.

Также возможно питание платы от USB-интерфейса. В этом случае плата должна быть соединена с персональным компьютером с помощью USB-кабеля, подключенного к USB-разъему отладочной платы, а переключатель *POWER\_SEL* должен находиться в положении *USB*.

На рисунке 1.3 показано размещение основных элементов на плате, а в таблице 1.2 приведено их описание.



Таблица 1.3 – Описание элементов отладочной платы

Поз.	Наименование элемента
1	Разъем CAN
2	Разъем для установки конфигурационной перемычки
3	Разъем RS-232
4	Приемопередатчик RS-232
5	Разъем для установки конфигурационной перемычки
6	Фильтр питания
7	Разъем питания 5 В
8	Кнопка RESET
9	Кнопки UP, DOWN, LEFT, RIGHT, SELECT
10	Кнопка WAKEUP
11	Жидкокристаллический дисплей с разрешением 128х64
12	Батарейный отсек (CR2032, 3 В)
13	Биполярный транзистор
14	Разъем TS 3.5 мм выхода ЦАП через звуковой усилитель
15	Разъем BNC выхода ЦАП
16	Разъем для установки конфигурационной перемычки
17	Разъем для установки конфигурационной перемычки
18	Разъем BNC входа компаратора
19	Разъем для установки конфигурационной перемычки
20	Разъем BNC канала 7 АЦП
21	Разъем для установки конфигурационной перемычки
22	Подстроечный резистор канала 7 АЦП
23	Разъем USB
24	Переключатели выбора режима загрузки
25	Разъем карты памяти microSD
26	Разъем портов В, С, D микроконтроллера
27	Контактное устройство для микроконтроллера
28	Разъем портов А, Е, F микроконтроллера
29	Разъем программирования и отладки JTAG-B
30	Разъем программирования и отладки JTAG-A
31	Разъемы для установки конфигурационных перемычек
32	Два красных светодиода (порт С)
33	Приемопередатчик CAN
34	Разъем для установки конфигурационной перемычки

Для загрузки программ, написанных с помощью персонального компьютера, во флеш-память микроконтроллера, а также для их отладки, необходим программатор-отладчик. В работах будет использован программатор J-LINK компании SEGGER, изображенный на рисунке 1.4. В целом, можно использовать любой аналог данного программатора, способный работать с ядром ARM Cortex-M3. Программатор подключается к компьютеру с помощью USB-кабеля и использует интерфейсы для внутрисхемной отладки SWD (англ. Serial Wire Debug) или JTAG (аббревиатура названия группы разработки стандарта – Joint Test Action Group). На плате предусмотрено два равнозначных разъема для подключения программатора (JTAG-A и JTAG-B).



Рисунок 1.4 – Программатор-отладчик J-LINK

Почти все линии ввода-вывода микроконтроллера заведены на штыревые разъемы X26 и X27, что позволяет подключить к ним необходимые внешние устройства. Назначение каждого контакта приведено в приложении, в таблице П.1. Также на эти разъемы заведены цепи земли GND (контакты 1, 2, 29, 30) и питания +3.3 В (контакты 3, 4) и +5 В (контакты 27, 28), от которых можно запитать подключаемые к плате внешние схемы.

Для отображения буквенно-цифровой и графической информации на плате предусмотрен монохромный жидкокристаллический индикатор размером 128x64 пикселя [x]. При отладке приложений удобно пользоваться двумя светодиодами красного цвета, подключенными к выводам микроконтроллера.

Для ввода информации можно использовать пять механических кнопок общего назначения: *LEFT*, *RIGHT*, *UP*, *DOWN* и *SEL*. Для сброса и перезапуска микроконтроллера предназначена кнопка *RESET*.

На плате также предусмотрен целый ряд иных компонентов, которые будут постепенно изучаться в последующих главах. Подробную информацию о плате можно получить из фирменного описания [x] и принципиальной схемы [x].



### 1.3. Подготовка отладочной платы к работе

Для подготовки отладочной платы к работе необходимо выполнить следующие действия:

1. Открыть коробку с отладочным комплектом и извлечь из нее плату.
  2. Отсоединить от платы дополнительные провода и детали, если таковые подключены к разъемам *X26* и *X27*.
  3. Извлечь из коробки программатор J-LINK и USB-кабель, соединить их между собой.
  4. Подключить шлейф программатора к разъему *JTAG-B*, расположенному на плате (обратите внимание на специальную направляющую, исключающую неправильное подключение).
  5. Убедиться, что переключатели выбора режима загрузки (рисунок 1.3, позиция 24) установлены в положение «0-0-0», что соответствует использованию интерфейса *JTAG-B* при загрузке и отладке.
  6. Включить компьютер, и после загрузки системы подключить USB-кабель программатора к свободному USB-порту компьютера. На программаторе должен загореться светодиод.
- Если светодиод все время мигает, то это означает некорректную связь между компьютером и программатором. Чаще всего такая проблема возникает ввиду отсутствия необходимого драйвера. Инструкция по его установке приведена в разделе «Установка программного обеспечения».
7. Убедиться, что переключатель *POWER\_SEL* установлена в положение *EXT\_DC* (внешний источник питания). При необходимости переставить ее в это положение (при другом положении переключки будет использовано питание со стороны USB-интерфейса).
  8. Включить блок питания в сетевую розетку и подключить шнур питания к соответствующему разъему на плате (рисунок 1.3, позиция 7). На плате должен загореться красный светодиод *POWER 5V*. Если в микроконтроллере уже содержится программа, то она начнет выполняться. Возможно, это будет сопровождаться миганием красных светодиодов или выводом информации на дисплей.

### 1.4. Среда программирования Keil $\mu$ Vision

Интегрированная среда программирования Keil  $\mu$ Vision MDK-ARM предназначена для написания и отладки программ для микроконтроллеров с ядром ARM с помощью языков Си, C++ и ассемблера. В рамках данного курса будет использоваться лишь язык Си.

В состав среды входят все необходимые для этого средства: специализированный текстовый редактор с семантической (смысловой) подсветкой кода, компилятор, ассемблер, компоновщик, отладчик и т.д. Среда программирования поддерживает практически все выпускаемые в мире микроконтроллеры с архитектурой ARM. Keil  $\mu$ Vision посредством драйверов может работать с различными внутрисхемными программаторами-отладчиками, в том числе с J-LINK.

Для начала работы со средой Keil  $\mu$ Vision запустите ее через ярлык на рабочем столе или, в случае его отсутствия, запустите файл *UV4.exe*, расположенный по адресу *C:\Keil\UV4*. Если на вашем компьютере еще не установлена среда программирования Keil  $\mu$ Vision, то следует обратиться к разделу «Установка программного обеспечения».

Внешним видом среда Keil  $\mu$ Vision напоминает среду Microsoft Visual Studio, поэтому программистам, знакомым с Visual Studio, нетрудно будет освоиться и с данной средой (рисунок 1.5). К сожалению, русифицированной версии Keil  $\mu$ Vision пока не существует. Весь интерфейс организован на английском языке.

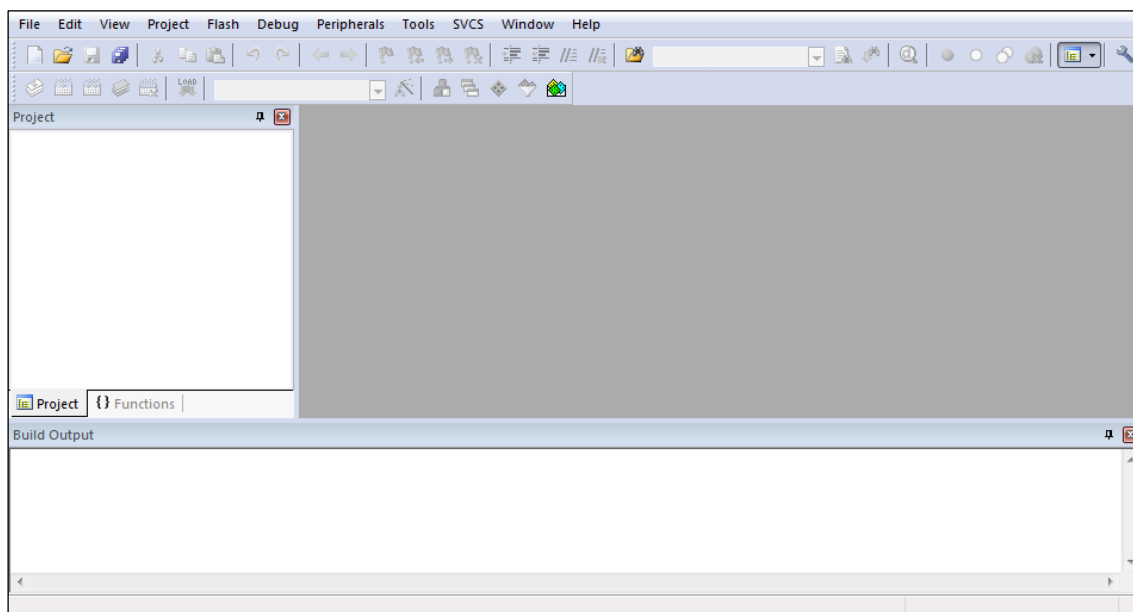


Рисунок 1.5 – Внешний вид среды Keil  $\mu$ Vision

В начале работы со средой Keil  $\mu$ Vision рекомендуется выполнить несколько действий по ее конфигурации, выбрав *Edit* → *Configuration...*

Окно конфигурации открывается на вкладке *Editor*, которая содержит практически все значимые параметры. В целом, конфигурация позволяет сделать процесс программирования удобным и привычным для пользователя. Здесь есть такие параметры, как автоматические отступы, синтаксическая подсветка для различных языков, нумерация строк, моноширинность шрифта, периодическое автосохранение проекта, правая вертикальная граница для компоновки кода и т.д. Наиболее важным здесь является параметр *Encoding*, которому необходимо присвоить значение *Russian Windows-1251*. Это позволит писать комментарии на русском языке и обеспечит верную интерпретацию кириллических символов библиотекой жидкокристаллического дисплея (раздел **x**). Остальные настройки могут быть выполнены исходя из личных предпочтений, либо, в случае отсутствия таковых, в соответствии с рисунком 1.6.

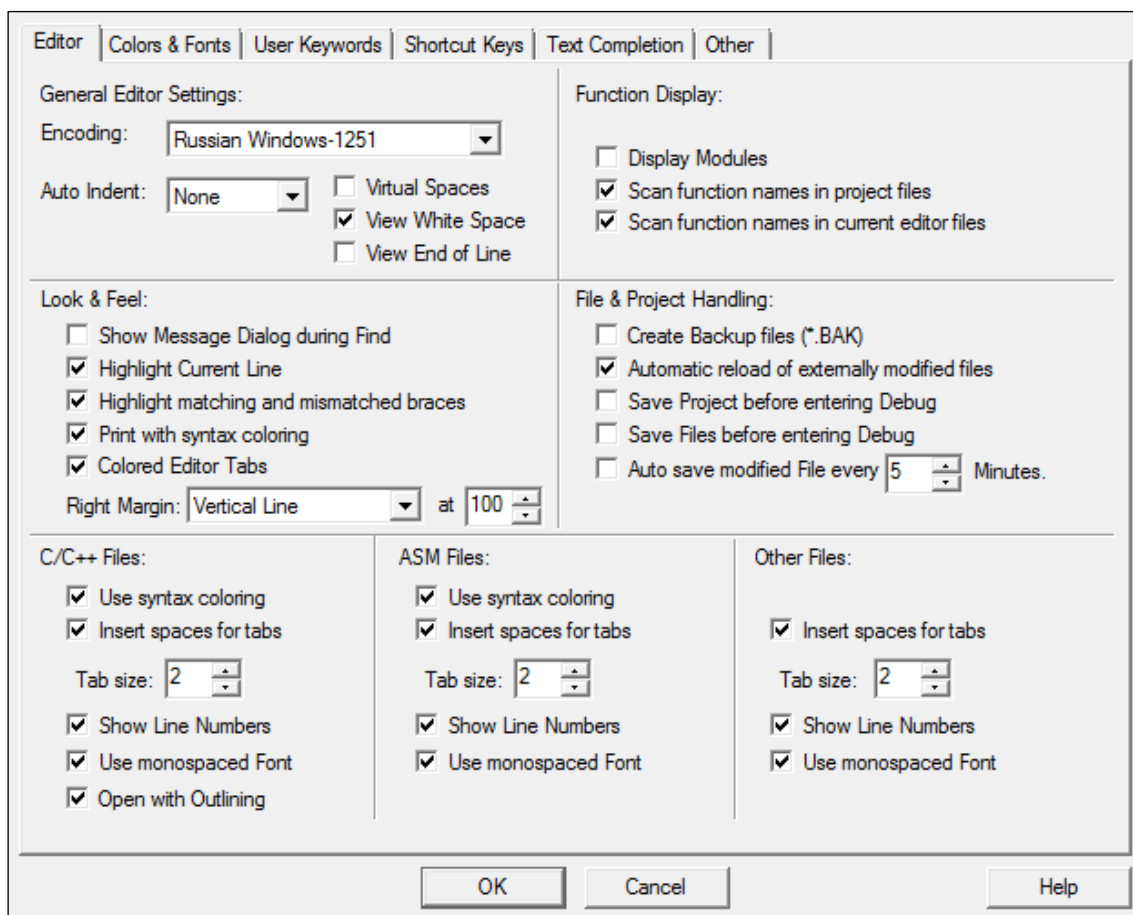


Рисунок 1.6 – Пример конфигурации среды программирования Keil μVision

## 1.5. Программный проект

В качестве первой учебной задачи предлагается написать программу, которая бы управляла миганием двух светодиодов, расположенных на отладочной плате (позиция 32 на рисунке 1.3). Для этого потребуется создать программный проект.

Программный проект для микроконтроллера в среде Keil μVision представляет собой достаточно сложную совокупность файлов, каталогов и настроек. Прежде всего, проект необходимо создать и настроить, и только после этого можно переходить непосредственно к написанию программы и ее отладке.

### 1.5.1. Создание проекта

Для создания проекта следует выбрать *Project* → *New μVision Project...*, указать в открывшемся окне директорию, в которой будет храниться проект, и его имя. Рекомендуется выбрать адрес *C:\Milandr\Samples\Sample 1.1\\_Project*, попутно создав недостающие папки. Обратите внимание, что **адрес проекта не должен содержать кириллических символов**, т.к. это может привести к техническим проблемам в ходе дальнейшей работы. Имя проекта можно задать стандартным – *Project*.

В появившемся после этого окне (рисунок 1.7) нужно выбрать целевое устройство – *MDR1986BE92*. Если в перечне устройств отсутствует секция «*Milandr*», то необходимо обратиться к разделу «Установка программного обеспечения» для импортирования стандартных периферийных библиотек.

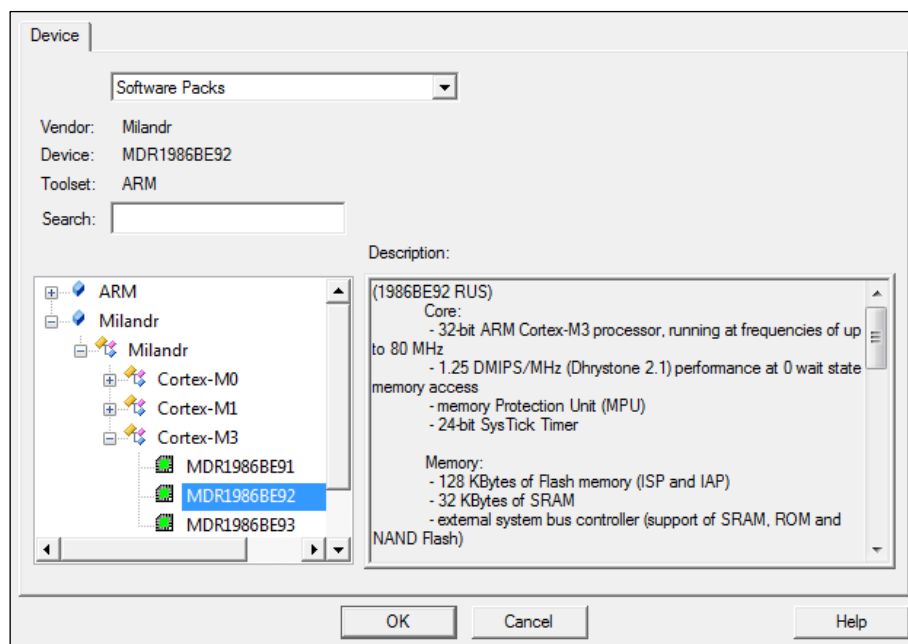


Рисунок 1.7 – Выбор целевого устройства

Далее откроется окно *Manage Run-Time Environment* (рисунок 1.8), которое позволит подключить к проекту требуемые библиотечные файлы.

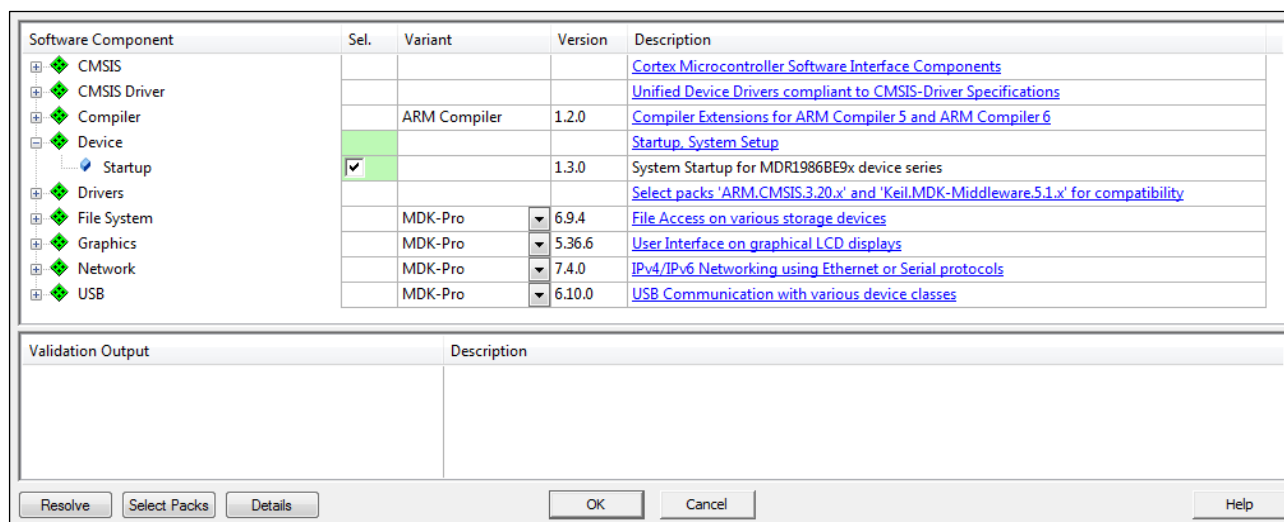



Рисунок 1.8 – Окно Manage Run-Time-Environment

Все периферийные библиотеки для микроконтроллеров серии 1986BE9x находятся в секции *Drivers*. Подключение библиотек осуществляется путем установки галочек напротив требуемых пунктов. Однако поскольку периферийные библиотеки в рамках данного цикла работ использоваться не будут, здесь потребуется подключить единственный модуль *Device* → *Startup*, содержащий данные, требуемые для запуска микроконтроллера.

В процессе работы всегда можно вернуться в это меню для подключения дополнительных библиотек или отключения ненужных с помощью кнопки  на панели инструментов.

В результате выполненных действий должен сформироваться проект со структурой, показанной на рисунке 1.9.

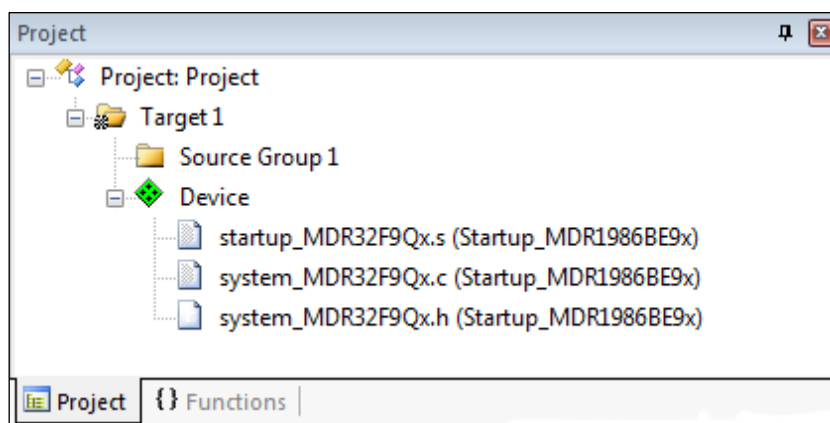


Рисунок 1.9 – Структура созданного проекта

### 1.5.2. Настройка проекта

Созданный проект необходимо настроить. Вход в меню настроек осуществляется с помощью кнопки . Меню состоит из десяти вкладок. Рассмотрим их по порядку.

#### Device

Во вкладке *Device* производится выбор целевого устройства. В данном случае нет потребности изменять эту вкладку, поскольку выбор был осуществлен на этапе создания проекта.

#### Target

Во вкладке *Target* находятся настройки параметров устройства и инструментов проекта (рисунок 1.10).

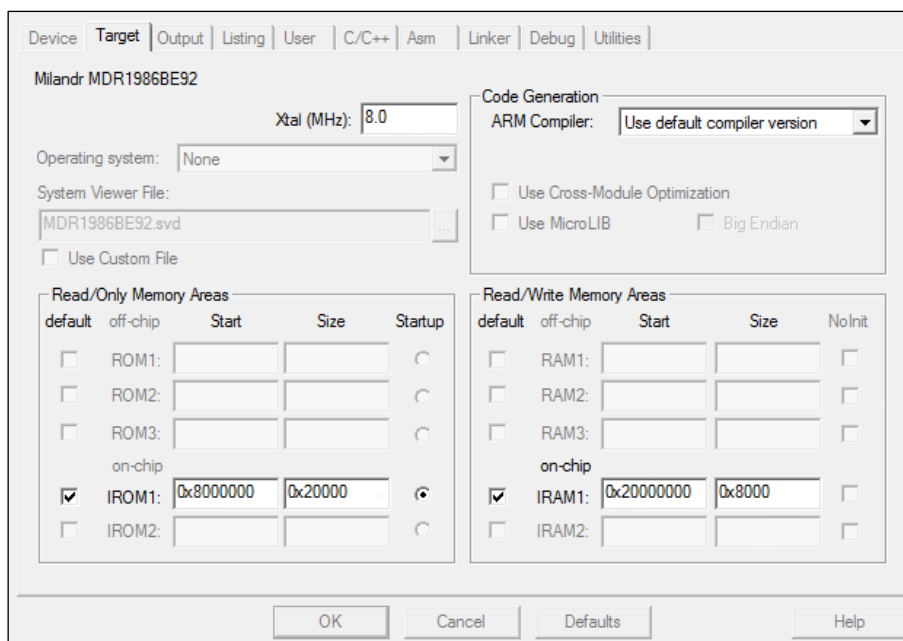


Рисунок 1.10 – Настройка параметров устройства и инструментов проекта

Здесь следует задать основную тактовую частоту ядра *Xtal (MHz)* равной 8.0: такую частоту имеет базовый генератор микроконтроллера.

При необходимости можно выбрать другой компилятор в графе *ARM Compiler*.

В секциях *Read/Only Memory Areas* и *Read/Write Memory Areas* задаются адреса и объемы флеш-памяти и оперативной памяти соответственно. Объем флеш-памяти микроконтроллеров серии 1986VE9х составляет 128 Кбайт = 131072 (0x20000) байт и начинается с адреса 0x8000000; объем оперативной памяти составляет 32 Кбайт = 32768 (0x8000) байт и начинается с адреса 0x20000000.

## Output

Вкладка *Output* позволяет указать каталог, в котором будут размещены выходные файлы проекта, задать имя исполняемого файла, а также разрешить создание специального файла с расширением \*.hex. (рисунок 1.11).

Рекомендуется задать адрес каталога ... \\_Project\Objects, чтобы все созданные объекты хранились в отдельной папке.

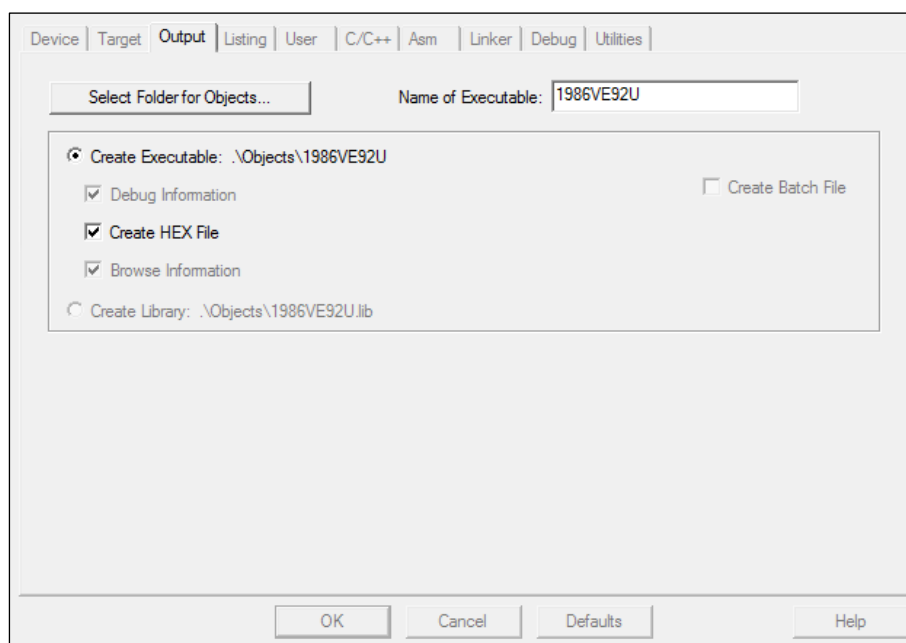


Рисунок 1.11 – Настройка выходных файлов проекта

Файл с расширением \*.hex (англ. Hexadecimal – шестнадцатеричный) содержит готовую программу для микроконтроллера в виде ряда шестнадцатеричных значений. Имея такой файл, можно запрограммировать микроконтроллер с помощью иных программно-аппаратных средств; например, по протоколу UART.

## Listings

Вкладка *Listings* содержит параметры формируемого листинга и карты проекта. Эта информация может быть полезна продвинутым пользователям при отладке программы.

Рекомендуется хранить эти файлы в отдельном каталоге ... \\_Project\Listings.

## User

Во вкладке *User* можно указать программы, которые должны быть выполнены до и после построения проекта. На данном этапе нет потребности здесь что-либо менять.

## C/C++

Во вкладке *C/C++* определяется параметры компилятора. Наибольший интерес здесь представляют три параметра: уровень оптимизации, стандарт компиляции и пути к заголовочным файлам проекта (рисунок 1.12).

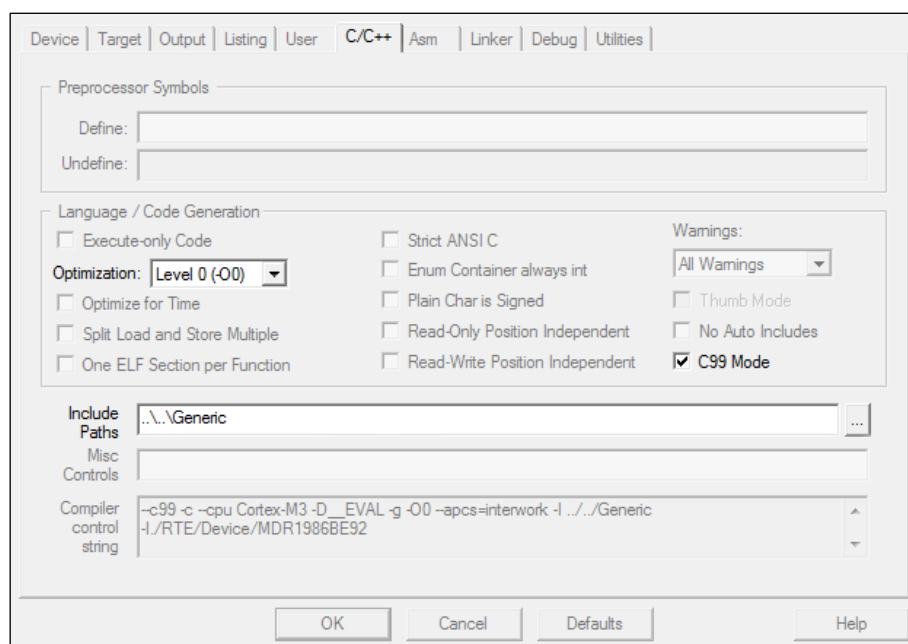


Рисунок 1.12 – Настройка параметров компилятора

Оптимизация, с одной стороны, позволяет создать более компактную и быструю программу. С другой стороны, она может привести к труднодиагностируемым ошибкам и усложняет процесс отладки. Поэтому если нет явной необходимости в снижении объема программы или времени ее выполнения, то рекомендуется выбрать низкий уровень оптимизации *Level 0 (-O0)*.

Активация параметра *C99 Mode* позволяет компилятору использовать стандарт языка Си 1999 года, который является более гибким и удобным.

Заголовочные файлы, подключаемые директивой `#include`, компилятор ищет в директории с активным модулем. Однако есть файлы, которые используются в неизменном виде в различных проектах (например, модуль с настройкой тактирования процессорного ядра). Эти файлы обычно хранят в отдельном каталоге. Для того чтобы компилятор смог их обнаружить, необходимо в поле *Include Paths* указать их адрес (рисунок 1.13).

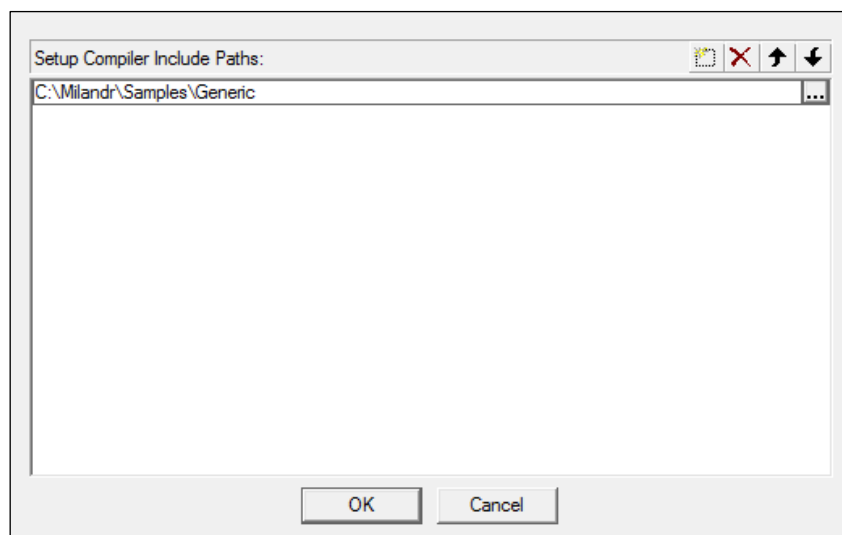


Рисунок 1.13 – Выбор директории файлов, используемых в проекте

### Asm

Вкладка Asm определяет конфигурацию ассемблера. Поскольку написание программы будет производиться на языке Си, нет нужды здесь что-то настраивать.

### Linker

Вкладке Linker содержит параметры компоновщика. Здесь можно задать целевые адреса памяти с помощью т.н. scatter-файла. На начальном этапе работы с микроконтроллером рекомендуется оставить параметры данной вкладки стандартными.

### Debug

Вкладка Debug предназначена для настройки процесса отладки (рисунок 1.14).

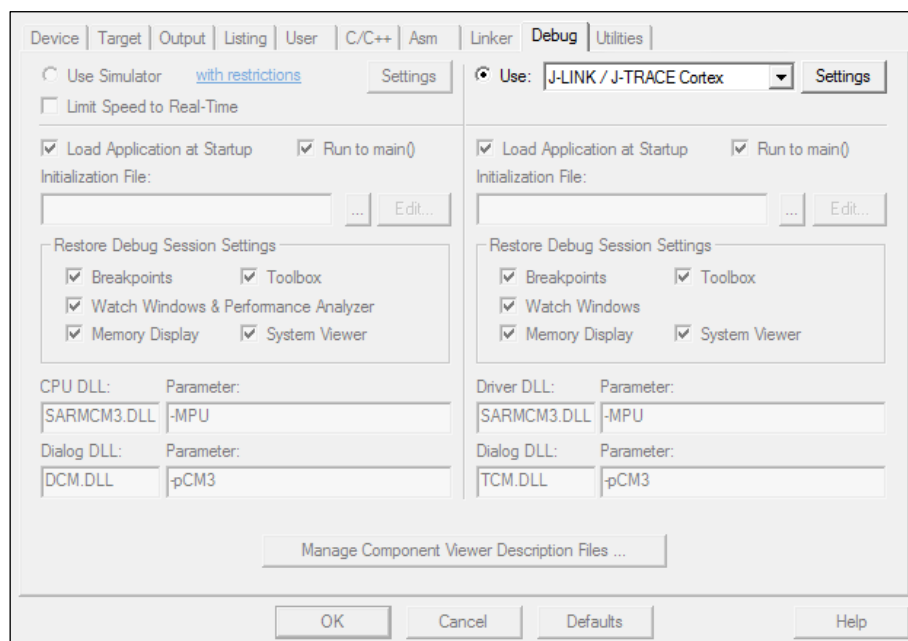


Рисунок 1.14 – Выбор устройства для отладки



Есть два варианта отладки: симуляция (левая половина окна) и использование аппаратных средств (правая половина окна). Симуляция выбирается в случае, когда нет возможности работать с аппаратурой, и вместо нее задействуется виртуальная модель устройства. Для использования программатора J-LINK необходимо выбрать в поле *Use* значение *J-LINK / J-TRACE Cortex*, а затем сконфигурировать его, нажав кнопку *Settings*.

При первом входе в настройки программатора появится сообщение о неопознанном устройстве (рисунок 1.15). После нажатия кнопки *OK* будет выведено окно с перечнем известных устройств, в котором понадобится выбрать пункт *Unspecified Cortex-M3* (рисунок 1.16).

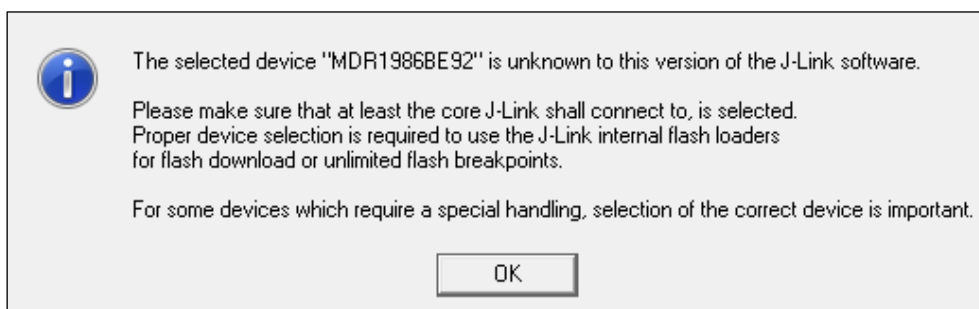


Рисунок 1.15 – Сообщение о неопознанном устройстве

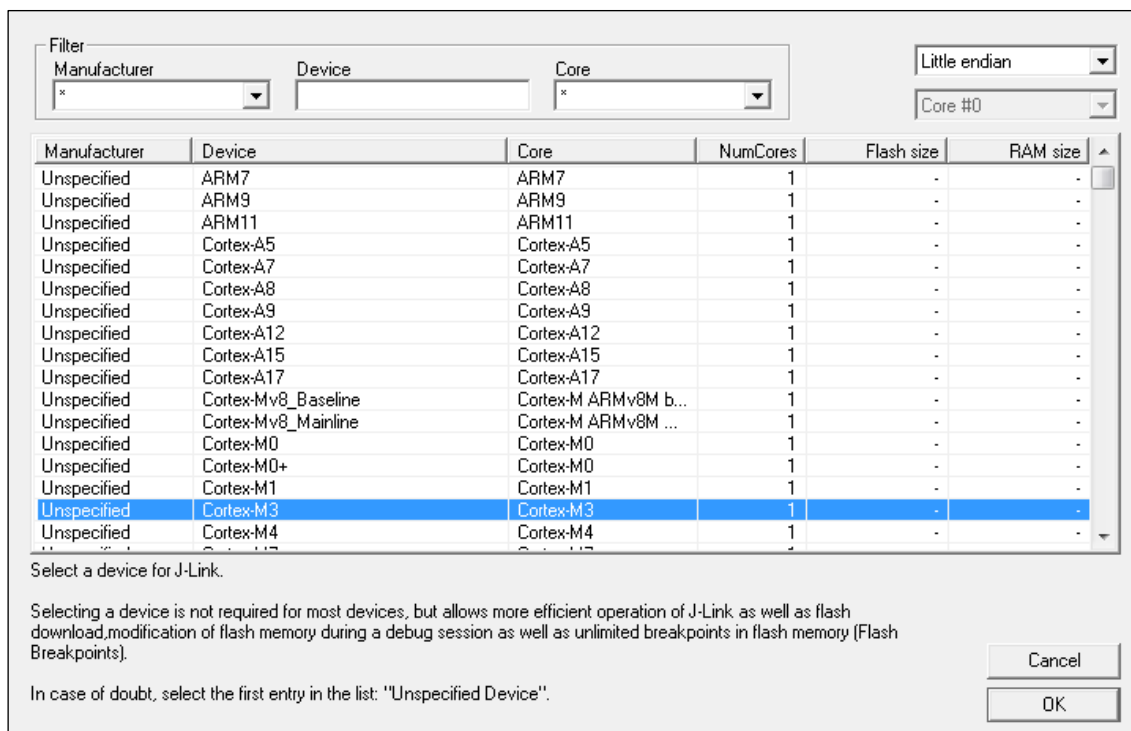


Рисунок 1.16 – Выбор устройства из перечня

Наконец, появится окно с настройками программатора, состоящее из трех вкладок. Первая вкладка, именуемая **Debug**, проиллюстрирована на рисунке 1.17.

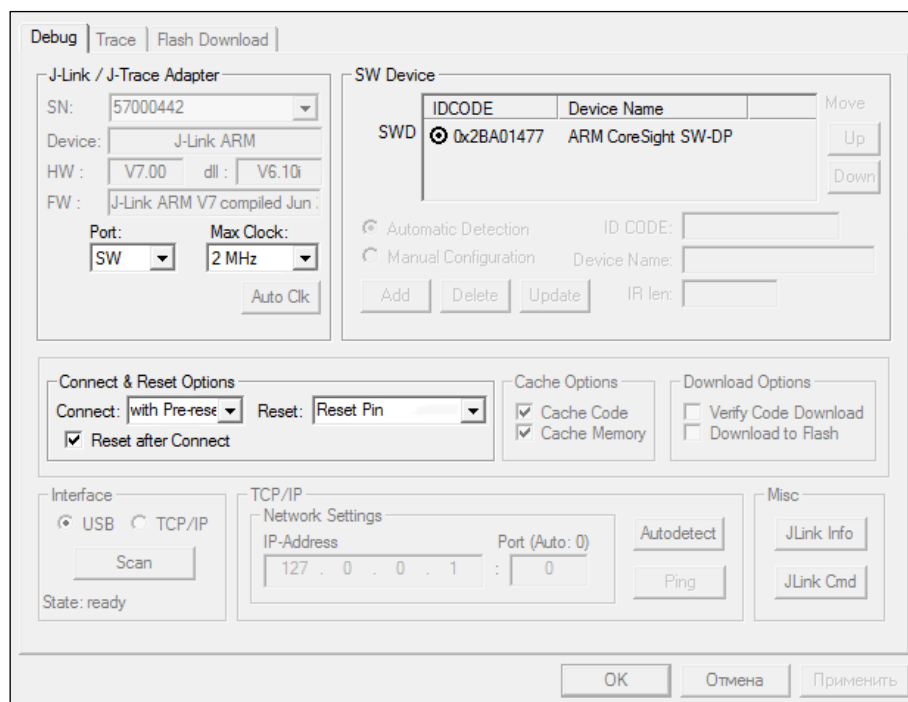


Рисунок 1.17 – Настройка процесса отладки

В секции *J-Link / J-Trace Adapter* можно выбрать протокол программирования и рабочую частоту. Протокол выбирается в поле *Port*; там следует выбрать значение *SW* (Serial Wire, последовательный порт). Частота определяется в поле *Max Clock*; ее рекомендуется задать равной 2 МГц.

Секция *Connect & Reset Options* определяет поведение системы при загрузке программы. Здесь рекомендуется задать значения *Connect: with Pre-reset* и *Reset: Reset Pin*. Это позволит избежать ошибок, связанных с программной блокировкой выводов, используемых интерфейсом SWD.

Если к моменту конфигурации проекта отладочная плата подготовлена в соответствии с разделом 1.3, то в секции *SW Device* будет отображено активное устройство *ARM CoreSight SW-DP*.

Вкладка **Trace** предназначена для настройки трассировки данных. Этот механизм по умолчанию отключен и не будет рассматриваться в рамках данного практикума.

Во вкладке **Flash Download** находятся настройки процесса загрузки программы (рисунок 1.18). В секции *Download Function* выбираются стираемые перед загрузкой области памяти и параметры загрузки:

- *Erase Full Chip* – стирание всей флеш-памяти;
- *Erase Sectors* – стирание только секторов, используемых программой;
- *Do not Erase* – не стирать память.
- *Program* – загрузка программы в микроконтроллер;
- *Verify* – верификация программы;
- *Reset and Run* – автоматический запуск программы по окончании загрузки.

В секции *Programming Algorithm* задается алгоритм программирования.

В случае отсутствия активного алгоритма, его необходимо добавить, нажав кнопку *Add* и выбрав нужный алгоритм из списка.

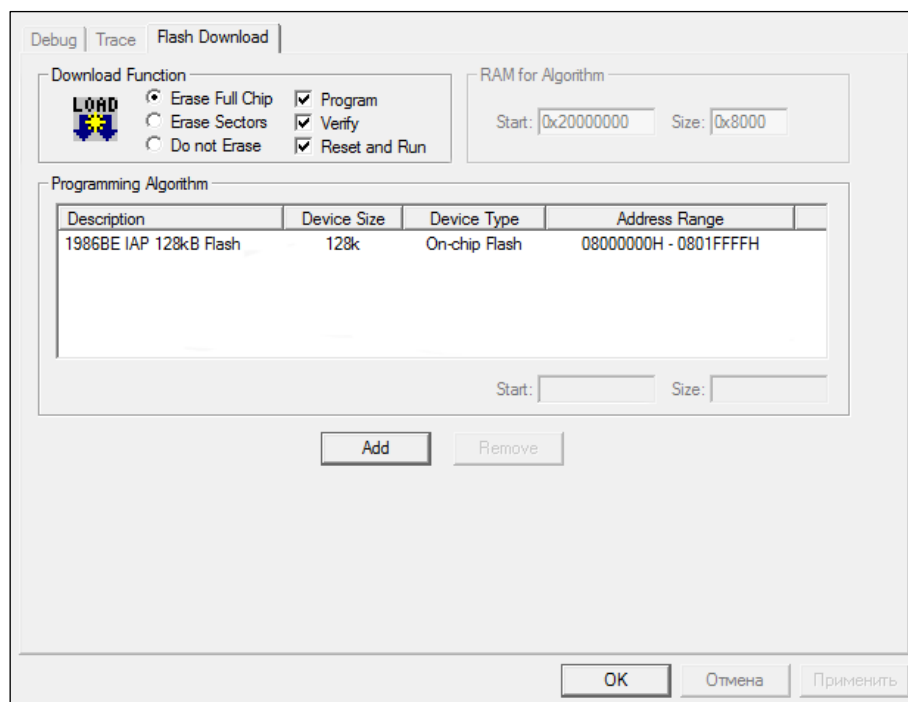


Рисунок 1.18 – Настройки процесса загрузки программы

Далее следует вернуться к основному меню настройки проекта, нажав кнопку *OK*.

## Utilities

Вкладка *Utilities* содержит дополнительные настройки процесса загрузки (рисунок 1.19). Здесь можно указать, какой программатор будет использован для загрузки программы: тот же, что для отладки, или какой-то дополнительный. Рационально, конечно, ограничиться одним программатором.

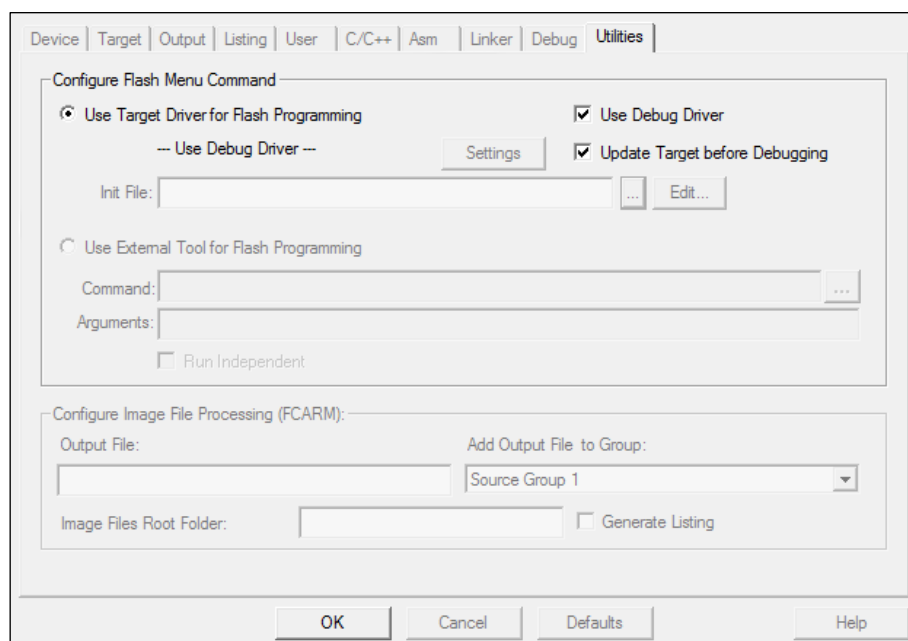


Рисунок 1.19 – Выбор устройства для загрузки программы

Настройка проекта на этом завершена. Можно покинуть меню, нажав кнопку *OK*.

### 1.5.3. Разработка модуля

Основными элементами проекта являются модули. Модуль на языке Си, как правило, состоит из двух файлов: собственно модуля – файла с расширением *\*.c*, и заголовочного файла с расширением *\*.h*; имена же файлов обычно одинаковые. В модуле содержатся исходные коды функций и объявления глобальных переменных, а в заголовке – их прототипы (предварительные описания). В отдельном модуле обычно располагают функции, родственные по смыслу, например, относящиеся к работе с определенным периферийным блоком или к определенной задаче. Наличие заголовочного файла в целом необязательно, и в простейших проектах возможно оформить всю программу в файле *\*.c*. Рекомендуется, однако, всегда ориентироваться на полноценную архитектуру проекта и не пренебрегать заголовками.

#### Создание модуля

Для создания нового модуля необходимо в окне *Project* кликнуть правой кнопкой мыши по разделу *Source Group 1* и выбрать из выпадающего списка пункт *Add New Item to Group 'Source Group 1'...* В открывшемся окне (рисунок 1.20) следует выбрать расширение файла *C File (.c)*, задать ему имя *main*, указать директорию его расположения *C:\Milandr\Samples\Sample 1.1\User* и нажать кнопку *Add*. В структуре проекта появится новый элемент *main.c*, в котором можно начать написание программы. Заголовочный файл создается таким же образом, только в качестве расширения выбирается *Header File (.h)*.

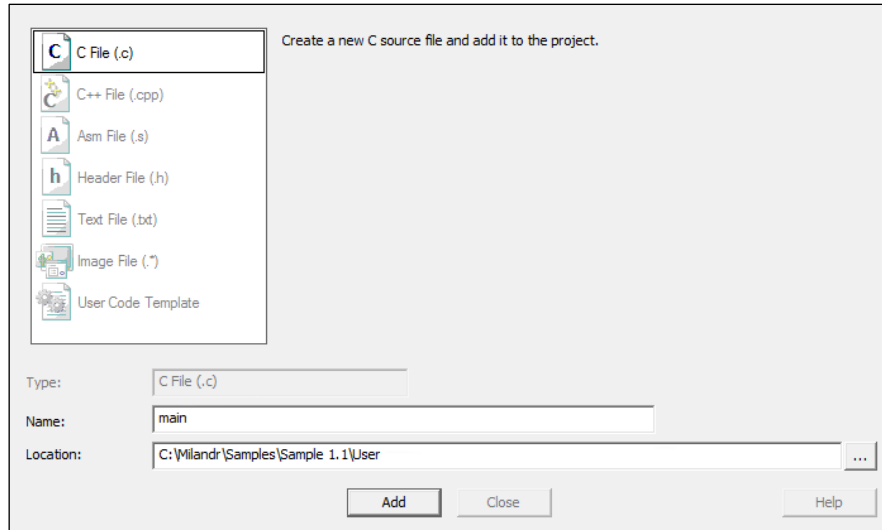


Рисунок 1.20 – Создание модуля main.c

Формирование новых модулей следует начинать с заголовочных файлов.

#### Написание заголовочного файла

**Заголовочный файл всегда должен содержать директивы условной компиляции.** Под условной компиляцией понимается проверка существования некоторой константы. Идея состоит в том, что если константа не существует, т.е. не объявлена, то это означает, что компилятор впервые достиг этого заголовка,

и его следует обработать. Если же константа существует, то повторной компиляции заголовка происходить не должно. Программно это реализуется так:

```
#ifndef MAIN_H // Проверка существования константы (if NOT defined)
#define MAIN_H // Объявление константы

// Содержание заголовочного файла
...

#endif // Завершение условной компиляции
```

Имя константы следует выбирать по определенному правилу, чтобы исключить возможность его совпадения с каким-либо иным идентификатором.

Данный метод предотвращает рекурсивное подключение заголовочных файлов и многократное определение одних и тех же функций и переменных.

Далее к заголовку следует подключить центральный библиотечный файл, содержащий разметку адресного пространства памяти микроконтроллера *1986VE9x.h*. Это позволит обращаться к именам регистров, указанных в спецификации, а не к числовым значениям адресов. Подключение файлов осуществляется с использованием директивы **#include**:

```
// Подключение центрального библиотечного файла
#include <1986VE9x.h>
```

Директива **#include** подставляет все содержимое файла в место ее вызова, позволяя, таким образом, в одном модуле **задействовать функции, переменные и константы другого**. Обратите внимание, что **организация модулей не предполагает подключение файлов с расширением \*.c**.

Работа с заголовочным файлом на этом временно завершена. Однако по мере разработки проекта к нему не раз потребуется вернуться.

### Написание модуля

В первую очередь требуется подключить к модулю прежде сформированный заголовок:

```
// Подключение заголовочного файла
#include "main.h"
```

Далее необходимо объявить основную функцию `main()`. Работа программы начинается **строго** с вызова этой функции. Кроме того, необходимо создать в этой функции **бесконечный цикл**, исключив возможность выхода из нее. Такова специфика работы с микроконтроллерами: при наличии питания на них всегда должны выполняться какие-либо операции (управление, вычисление, опрос). Выход из основной функции можно интерпретировать, как зависание системы. До входа в бесконечный цикл обычно производят требуемые однократные процедуры настройки и инициализации:

```

// Основная функция
int main(void)
{
    // Настройка и инициализация
    ...

    // Основной цикл
    while (1)
    {
        // Периодические операции
        ...
    }
}

```

Под инициализацией понимается подготовка к работе каких-либо модулей, блоков или устройств. Инициализацию рекомендуется выносить в отдельную процедуру. В частности, для выполнения поставленной задачи – управление светодиодами – потребуется выполнить инициализацию порта ввода-вывода для работы с ними; такая процедура приведена ниже (без подробного описания, т.к. конфигурация портов ввода-вывода будет детально рассмотрена в следующей главе):

```

// Инициализация порта для работы со светодиодами
void LED_Init(void)
{
    // Включение тактирования порта C
    MDR_RST_CLK->PER_CLOCK |= (1 << RST_CLK_PCLK_PORTC_Pos);

    // Конфигурация линий PC0 и PC1
    MDR_PORTC->OE      |= ((1 << 0) | (1 << 1)); // Направление данных (вывод)
    MDR_PORTC->PULL     &= ~((1 << 0) | (1 << 1)); // Подтяжка к земле (отключена)
    MDR_PORTC->PULL     &= ~((1 << 16) | (1 << 17)); // Подтяжка к питанию (отключена)
    MDR_PORTC->ANALOG   |= ((1 << 0) | (1 << 1)); // Режим работы линий (цифровой)
    MDR_PORTC->FUNC     &= ~((3 << 0) | (3 << 2)); // Функции линий (ввод-вывод)
    MDR_PORTC->PD       &= ~((1 << 0) | (1 << 1)); // Управление линиями (драйвер)
    MDR_PORTC->PD       &= ~((1 << 16) | (1 << 17)); // Триггеры Шмитта (отключены)
    MDR_PORTC->PWR      &= ~((3 << 0) | (3 << 2)); // Сброс битов регистра PWR
    MDR_PORTC->PWR      |= ((1 << 0) | (1 << 2)); // Крутизна фронтов (низкая)
    MDR_PORTC->GFEN     &= ~((1 << 0) | (1 << 1)); // Цифровые фильтры (отключены)
}

```

После описания процедуры следует сразу **добавить ее прототип в заголовочный файл**. Прототипом функции называется объявление функции, не содержащее ее тело, но указывающее ее арность и типы аргументов и возвращаемых данных:

```

// Прототип процедуры инициализации порта для работы со светодиодами
void LED_Init(void);

```

Мигание светодиодов подразумевает их периодическое включение и выключение. Однако если выполнять эти операции на частоте, с которой работает процессорное ядро (порядка 8 МГц), то визуально различить мигание будет невозможно: светодиоды будут светиться монотонно, хотя и с меньшей яркостью. Поэтому после включения и выключения необходимо организовать задержки, которые будут определять частоту мигания.

Есть несколько способов создания задержек, как, например, использование аппаратных таймеров и сервисов операционных систем реального времени. Но эти способы требуют дополнительных программных инструментов, и будут рассмотрены в дальнейшем. В этот раз для создания задержек будет использован **пустой цикл счета** с целью максимального упрощения. Идея заключается в том, что процессорное ядро не перейдет к выполнению следующей операции до тех пор, пока не выполнит счет до заданного значения. Важно отметить, что такой способ является **крайне нежелательным** в рабочих проектах, т.к. во время задержки ядро выполняет бесполезные вычисления и не способно управлять системой.

Такую задержку можно создать, используя цикл `for` или `while`:

```

for (uint32_t i = 100000; i != 0; i--);
uint32_t i = 100000;
while (i != 0) i--;

```

Каждая итерация такого цикла занимает **4 такта** процессорного ядра: чтение, сравнение, декремент, запись. Соответственно, фактическое время задержки может быть приблизительно определено по такой формуле:

$$\tau = \frac{4}{SystemCoreClock} \times i, \quad (1.1)$$

где *SystemCoreClock* – тактовая частота ядра,

*i* – количество итераций в цикле.

Таким образом, при тактовой частоте ядра равной 8 МГц (значение по умолчанию), для формирования задержки в 100 миллисекунд требуется выполнить 200 000 итераций цикла:

$$i = \frac{SystemCoreClock}{4} \times \tau = \frac{8 \times 10^6}{4} \times 0.1 = 200\,000.$$

Цикл счета и формулу 1.1 можно объединить в виде **макроподстановки** (макроса) так, чтобы в дальнейшем задавать время задержки, например, в миллисекундах:

```

// Макрос задержки (в миллисекундах)
#define DELAY(VALUE) for (uint32_t i = (VALUE) * SystemCoreClock / (4 * 1000); \
                          i != 0; i--)

```

Этот макрос следует поместить в заголовочный файл. Тогда для создания задержки в 100 миллисекунд достаточно написать:

```

// Задержка (~100 мс)
DELAY(100);

```

**Обратите внимание на тип переменной в цикле** – `uint32_t`. При работе с микроконтроллерами для объявления переменных не рекомендуется использовать отдельно стандартный тип `int`, т.к. он, во-первых, может быть интерпретирован

различными компиляторами по-разному, что приводит к трудно диагностируемым программным ошибкам; а во-вторых, занимает целое машинное слово в памяти микроконтроллера даже в тех случаях, когда в этом нет потребности. Поэтому переменные лучше объявлять с уточняющими приставками согласно таблице 1.3:

Таблица 1.3 – Примеры объявления переменных

Полное объявление	Псевдоним	Описание
<code>unsigned char</code>	<code>uint8_t</code>	Беззнаковая 8-битная
<code>unsigned short int</code>	<code>uint16_t</code>	Беззнаковая 16-битная
<code>unsigned int</code>	<code>uint32_t</code>	Беззнаковая 32-битная
<code>signed char</code>	<code>int8_t</code>	Знаковая 8-битная
<code>signed short int</code>	<code>int16_t</code>	Знаковая 16-битная
<code>signed int</code>	<code>int32_t</code>	Знаковая 32-битная

### Работа с регистрами

Светодиоды подключены к линиям PC0, PC1 (таблица П.1). Чтобы светодиоды загорелись, требуется создать на этих линиях некоторое напряжение, т.е. установить на них **высокий логический уровень**. Чтобы светодиоды погасли, требуется, соответственно, убрать с линий напряжение, т.е. установить **низкий логический уровень**. Управление этими линиями осуществляется через регистр `MDR_PORTC->RXTX`. Регистр состоит из 16 функциональных битов, каждый из которых определяет состояние соответствующей линии ввода-вывода: бит 0 соответствует линии PC0, бит 1 – линии PC1 и т.д. Для включения светодиодов требуется задать единичное значение битам 0 и 1 регистра `MDR_PORTC->RXTX` (таблица 1.4).

Таблица 1.4 – Требуемое для включения светодиодов значение регистра `MDR_PORTC->RXTX`

№ бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Значение	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1

Если пойти самым очевидным путем и напрямую записать в регистр значение  $11_2 = 3_{10}$  таким образом:

```
// Установка битов 0 и 1
MDR_PORTC->RXTX = 3;
```

то битам 2...15 при этом будут присвоены нулевые значения. Т.е. если прежде проводилась работа с данным регистром, то **после этой операции состояние регистра будет сброшено!** Поэтому данный подход может быть использован только в случае, если работа с регистром **гарантированно** производится впервые согласно алгоритму программы. **Это очень важный момент при работе с регистрами.**



Чтобы сохранить данные других битов регистра требуется выполнять действие в три операции:

- **чтение**, чтобы получить информацию о состоянии регистра;
- **модификация**, чтобы изменить значения требуемых битов;
- **запись**, чтобы поместить модифицированное значение в регистр.

Модификация значения обычно производится с помощью операций **логического «ИЛИ»** (дизъюнкция) и **логического «И»** (конъюнкция), таблицы истинности которых приведены ниже и обозначены номерами 1.5 и 1.6.

Таблица 1.5 – Истинность операции логического «ИЛИ» (OR)

x	y	x   y
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 1.6 – Истинность операции логического «И» (AND)

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

Для **установки битов** (присвоения единиц) используют операцию логического «ИЛИ»:

```
uint8_t object = 0xA8;           // Исходный объект (101010002)
uint8_t data   = 0x03;           // Целевые биты (000000112)

uint8_t result = object | data;  // Результат (101010112)
```

Для **сброса битов** (присвоения нулей) используют операцию логического «И»:

```
uint8_t object = 0xAB;           // Исходный объект (101010112)
uint8_t data   = 0x03;           // Целевые биты (000000112)

uint8_t result = object & ~data; // Результат (101010002)
```

Обратите внимание, что операция сброса битов выполняется с использованием побитового отрицания (~).

На языке Си прием «чтение – модификация – запись» можно описать одной строкой:

```
// Установка битов 0 и 1
MDR_PORTC->RXTX = MDR_PORTC->RXTX | 3;

// Сброс битов 0 и 1
MDR_PORTC->RXTX = MDR_PORTC->RXTX & ~3;
```

Или еще лаконичнее:

```
// Установка битов 0 и 1
MDR_PORTC->RXTX |= 3;

// Сброс битов 0 и 1
MDR_PORTC->RXTX &= ~3;
```

Кроме того, для работы с отдельными битами удобно использовать операцию битовых сдвигов вправо на величину, равную позиции бита в регистре, например:

```
// Установка битов 0 и 1
MDR_PORTC->RXTX |= (1 << 0) | (1 << 1);

// Сброс битов 0 и 1
MDR_PORTC->RXTX &= ~(1 << 0) | (1 << 1));
```

Нетрудно посчитать, что результат операции  $(1 \ll 0) \mid (1 \ll 1)$  равен 3. Вместо числовых позиционных значений могут использоваться их псевдонимы из заголовочного файла *1986VE9x.h*, которые дополнительно дают представление о функциональном назначении бита. Такая форма записи значительно увеличивает удобочитаемость кода.

Если же вернуться к миганию светодиодами, то для выполнения последовательной периодической установки и сброса битов можно использовать операцию **исключающего «ИЛИ»** (таблица 1.7).

```
// Инверсия битов 0 и 1
MDR_PORTC->RXTX ^= (1 << 0) | (1 << 1);
```

Таблица 1.7 – Истинность операции  
исключающего «ИЛИ» (XOR)

x	y	x   y
0	0	0
0	1	1
1	0	1
1	1	0

### Завершение написания модуля


Все требуемые для выполнения задачи элементы теперь готовы. Остается вызвать их в нужном порядке. Функция `main()` в итоге должна принять такой вид:

```
// Основная функция
int main(void)
{
    // Инициализация светодиодов
    LED_Init();

    // Основной цикл
    while (1)
    {
        // Инверсия состояния линий PC0 и PC1
        MDR_PORTC->RXTX ^= (1 << 0) | (1 << 1);

        // Задержка перед началом следующей итерации (~100 мс)
        DELAY(100);
    }
}
```

### 1.5.4. Построение проекта

Под построением проекта понимается компиляция всех модулей, входящих в его состав, а также их компоновка. Построение запускается командой *Project → Build Target* (кнопка  на панели инструментов; кнопка *F7* на клавиатуре).

Процесс и результаты построения проекта отображаются в нижнем окне среды *Build Output*. При построении проекта оно должно отобразить примерно следующую информацию:

```
*** Using Compiler 'V5.06 update 4 (build 422)', folder: 'C:\Keil\ARM\ARMCC\Bin'
Build target 'Target 1'
compiling main.c...
assembling startup_MDR32F9Qx.s...
compiling system_MDR32F9Qx.c...
linking...
Program Size: Code=704 RO-data=224 RW-data=4 ZI-data=1636
".\Objects\1986VE92U.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:01
```

Как видно, в первую очередь производится компиляция файлов с расширением \*.c и ассемблирование файлов с расширением \*.s, а затем их компоновка (англ. Linking).

Если в процессе построения проекта произошли какие-либо неполадки, то информация о них (суть ошибки, номер строки) будет отображена в указанном окне. Есть два вида неполадок:

1. **Ошибки** (англ. **Errors**) – это синтаксические или смысловые нарушения. Они **должны быть исправлены** для успешного завершения компиляции.

2. **Предупреждения** (англ. **Warnings**) – это сообщения о нерациональных или потенциально неверных областях программного кода. Например, объявление неиспользуемых переменных или выражений, отсутствие пустой строки в конце файла или пустой цикл. **Предупреждения необязательно устранять** для успешной компиляции, однако **настоятельно рекомендуется обращать на них внимание**, т.к. в них часто содержится информация о том, почему программа работает некорректно.

Если программа не содержит ошибок (*0 Error(s), 0 Warning(s)*), то в строке *Program Size* будет отображен ее информационный объем. Микроконтроллер для хранения информации использует два устройства памяти:

- постоянное запоминающее устройство (ПЗУ; англ. Read-Only Memory, ROM);
- оперативное запоминающее устройство (ОЗУ; англ. Random Access Memory, RAM).

Приведенные данные показывают распределение информации между этими устройствами (в байтах):

- *Code=704* – объем флеш-памяти, занимаемый программным кодом;
- *RO-data=224* – объем флеш-памяти, занимаемый константами, то есть данными, доступными только для чтения (Read Only);
- *RW-data=4* – объем оперативной памяти, занимаемый переменными, т.е. данными, доступными и для чтения, и для записи (Read / Write); для хранения значений переменных необходимо выделить такое же количество флеш-памяти;
- *ZI-data=1636* – объем оперативной памяти, занимаемый переменными, которые инициализированы нулевыми значениями (Zero Initialized).


Таким образом, требуемые объемы флеш-памяти и оперативной памяти составляют:

$$\text{ROM} = \text{Code} + \text{RO} + \text{RW} = 704 + 224 + 4 = 932 \text{ (байта)};$$
$$\text{RAM} = \text{RW} + \text{ZI} = 4 + 1636 = 1700 \text{ (байт)}.$$

При разработке проекта всегда нужно помнить, что микроконтроллер имеет ограниченный объем памяти. В частности, объем флеш-памяти микроконтроллера 1986BE92У составляет 128 Кбайт; объем оперативной памяти – 32 Кбайта. Кроме того, в используемой триальной версии среды Keil  $\mu$ Vision установлено ограничение в 32 Кбайта флеш-памяти. Вплотную к этим границам приближаться не следует, сохраняя запас хотя бы в 10%.

Если памяти не хватает, то можно порекомендовать следующие приемы:

- включить оптимизацию программного кода компилятором;
- использовать менее ресурсоемкие алгоритмы;
- использовать микроконтроллер с бóльшим объемом памяти.

Следует отметить, что иногда бывает полезно полностью **перестроить проект** с использованием команды *Project*  $\rightarrow$  *Rebuild all target files* (кнопка  на панели инструментов). При перестроении производится компиляция всех модулей проекта, в то время как построение касается только модулей, подвергшихся изменению с момента последней компиляции. При этом очевидно, что при первичной сборке проекта разницы между построением и перестроением нет. В целом, перестроение проекта занимает больше времени, но работает несколько надежней.


Результатом построения проекта являются файлы с расширениями *\*.axf* и *\*.hex* (если его создание включено в настройках). Файл *\*.axf* содержит исполняемый код и отладочную информацию, требуемые для работы по интерфейсам JTAG и SWD. Файл *\*.hex* также содержит исполняемый код и может быть использован для программирования микроконтроллера, однако при работе через отладочные интерфейсы не используется.

После успешного построения проекта можно перейти к его загрузке в память микроконтроллера.

### 1.5.5. Загрузка программы в память микроконтроллера

Для того чтобы созданный проект начал работать, его необходимо загрузить в память микроконтроллера. Программисты для этой процедуры нередко употребляют термины «прошить контроллер» или «залить программу».

Перед загрузкой программы **следует убедиться, что отладочная плата готова к работе** согласно разделу 1.3.

Загрузка осуществляется командой *Flash*  $\rightarrow$  *Download* (кнопка  на панели инструментов; кнопка F8 на клавиатуре). Процесс загрузки отображается в левом нижнем углу окна Keil  $\mu$ Vision, а информация о завершении этапов загрузки описывается в окне *Build Output*:

```
Full Chip Erase Done.
Programming Done.
Verify OK.
Application running ...
```

После этого программа начнет выполняться внутри микроконтроллера: два светодиода, расположенные близ разъема *JTAG-A* отладочной платы, начнут мигать. Если в настройках процесса загрузки не была активирована опция *Reset and Run*, то для запуска программы потребуется нажать кнопку *RESET* на отладочной плате.

### Возможные неполадки

Если при попытке загрузки программы появится окно с ошибкой, изображенное на рисунке 1.21, то это свидетельствует о том, что файл *\*.axf*, содержащий исполняемый код, не был создан. Необходимо провести повторное построение проекта и убедиться в отсутствии ошибок при компиляции.

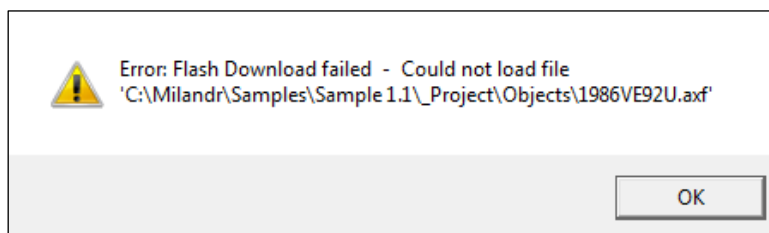


Рисунок 1.21 – Ошибка создания файла *\*.axf*

Окно с ошибкой, показанное на рисунке 1.22, наиболее вероятно, означает, что программатор не подключен к компьютеру: следует проверить соединительный кабель между ними. Иногда может потребоваться подключить кабель к другому USB-разъему компьютера.



Рисунок 1.22 – Ошибка распознавания программатора

Окно с ошибкой, изображенное на рисунке 1.23, сообщает о том, что программатору не удалось распознать микроконтроллер.

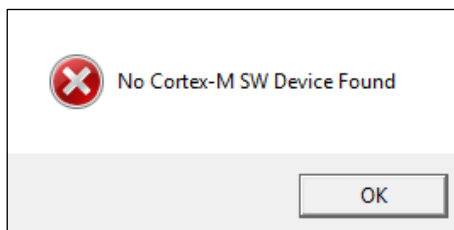


Рисунок 1.23 – Ошибка распознавания микроконтроллера

Причины этой ошибки могут быть разные. Для ее устранения требуется проверить следующие области:

- убедиться в наличии питания отладочной платы;
- убедиться, что программатор подключен к разъему *JTAG-B*;
- убедиться, что переключатели выбора режима загрузки (рисунок 1.3, позиция 24)

находятся в нужном положении («0-0-0»);

- убедиться в наличии драйвера для программатора:

*Панель управления → Диспетчер устройств → Контроллеры USB / Другие устройства* (в случае его отсутствия обратитесь к разделу «Установка программного обеспечения»);

– убедиться, что в настройках проекта установлен предварительный сброс линий, используемых интерфейсом программирования (рисунок 1.17):

*Option for Target “...” → Debug → Settings → Debug → Connect & Reset Options*


– убедиться, что микроконтроллер содержится в контактном устройстве (рисунок 1.3, позиция 27).

Если все вышеперечисленное выполнено, но ошибка все равно имеет место, то можно попробовать выполнить программирование через разъем *JTAG-A*; для этого требуется отключить питание отладочной платы, подключить программатор к разъему *JTAG-A*, выставить переключатели в положение «1-0-0», и вновь подключить питание.

### 1.5.6. Внутрисхемная отладка

При разработке проектов в программы нередко попадают ошибки различного рода. Их идентификация может быть очень непростой и трудоемкой задачей, особенно в комплексных проектах.

Для поиска и устранения ошибок реализована система **внутрисхемной отладки**, позволяющая выполнять программный код на микроконтроллере в пошаговом режиме, отслеживая состояние данных и регистров. Система состоит из уже используемого программатора-отладчика J-LINK, и программной составляющей, интегрированной в среду Keil  $\mu$ Vision.

Перед началом процесса отладки программу следует загрузить в микроконтроллер. Переход в режим отладки осуществляется командой *Debug → Start/Stop Debug Session* (кнопка  на панели инструментов; кнопки Ctrl + F5 на клавиатуре). При этом появится окно с предупреждением, что разрешенный объем памяти для используемой версии Keil  $\mu$ Vision составляет 32 Кбайта (рисунок 1.24), а основное окно среды несколько изменится (рисунок 1.25).

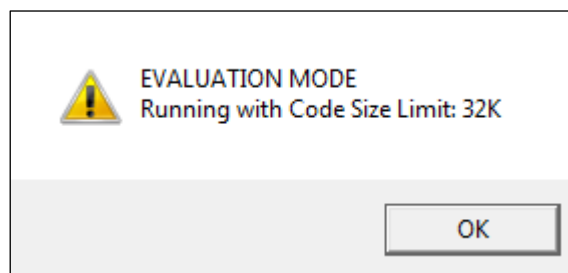


Рисунок 1.24 – Предупреждение об ограничении по размеру кода

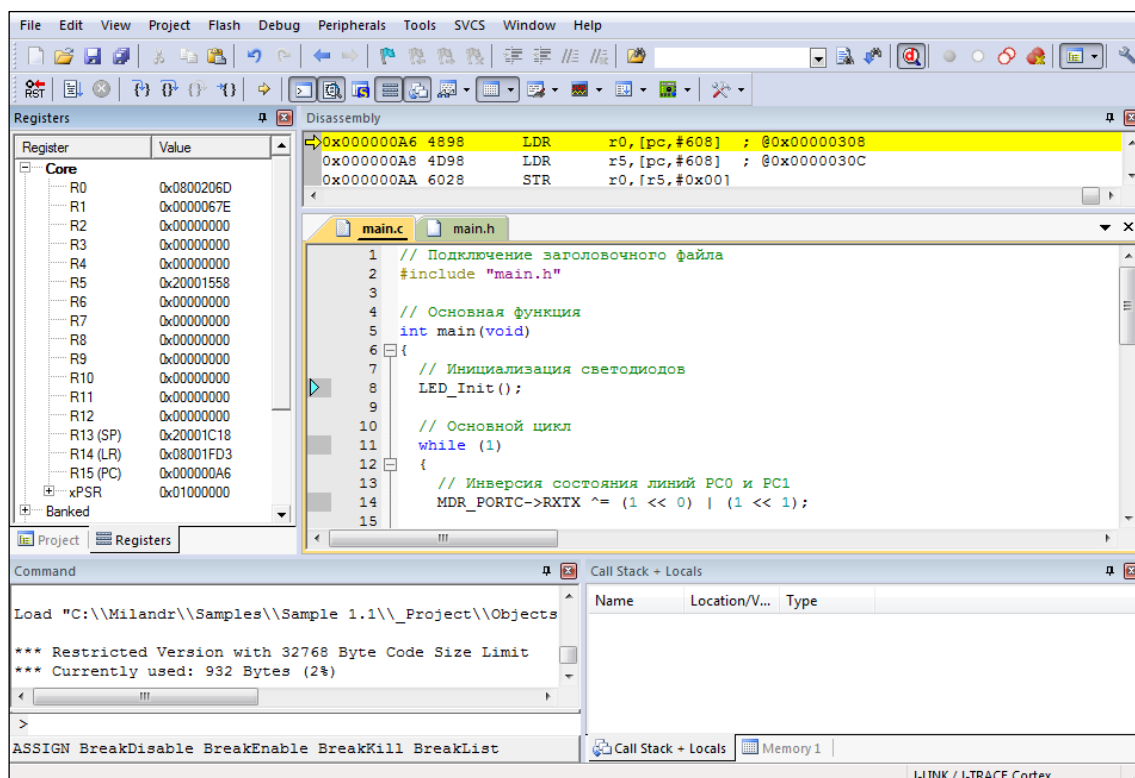






Рисунок 1.25 – Внешний вид среды Keil μVision в режиме отладки

На текущий момент программа еще не начала свое выполнение. Запуск программы производится командой *Debug* → *Run* (кнопка  на панели инструментов; кнопка *F5* на клавиатуре).

Для остановки выполнения программы используется команда *Debug* → *Stop* (кнопка  на панели инструментов). Остановка производится в случайном месте: оно будет отмечено маркером . Вероятнее всего, это будет цикл задержки.

В любом месте программы можно создать **точку останова** (англ. Breakpoint, BP), т.е. точку, по достижении которой ход выполнения программы будет приостановлен. В этот момент можно проанализировать состояние системы и убедиться, что значения целевых параметров верны. Для создания точки останова следует установить текстовый курсор на требуемый оператор и выполнить команду *Debug* → *Insert/Remove Breakpoint* (кнопка  на панели инструментов; кнопка *F9* на клавиатуре). В поле слева от номера строки появится красный круг (рисунок 1.26). Удобно также ставить точки останова, кликая по этому полю. Повторное выполнение описанных операций удаляет точку останова.

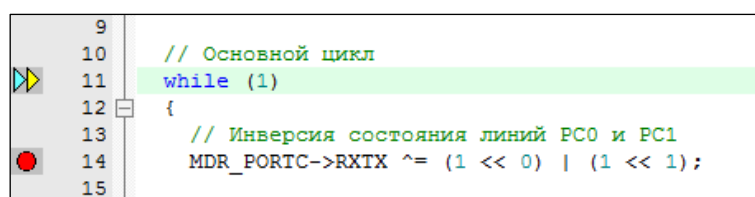



Рисунок 1.26 – Отображение точки останова

Количество точек останова ограничено: их должно быть **не более 5**, иначе программатор сообщит об ошибке при попытке запустить программу.

Все точки останова могут быть одновременно удалены командой *Debug → Kill All Breakpoints* (кнопка  на панели инструментов; кнопки *Ctrl + Shift + F9* на клавиатуре).

Для пошагового выполнения программы предусмотрено четыре функции, описанные в таблице 1.8.

Таблица 1.8 – Функции пошагового выполнения программы в среде Keil  $\mu$ Vision

№ п/п	Полная команда	Панель инструментов	Клавиатура	Описание
1	<i>Debug → Step</i>		<i>F11</i>	Выполнение оператора с заходом в тело функции
2	<i>Debug → Step Over</i>		<i>F10</i>	Выполнение оператора без захода в тело функции
3	<i>Debug → Step Out</i>		<i>Ctrl + F11</i>	Выполнение программы до выхода из функции
4	<i>Debug → Run To Cursor Line</i>		<i>Ctrl + F10</i>	Выполнение программы до текстового курсора

В режиме отладки можно посмотреть состояние практически всех регистров микроконтроллера. Для отображения информации о конкретном периферийном блоке следует выполнить команду *Peripherals → System Viewer → MDR\_xxx*. Справа появится окно с состоянием регистров данного блока (рисунок 1.27).

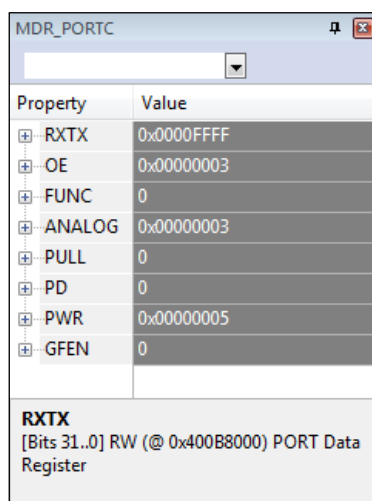


Рисунок 1.27 – Отображение состояния регистров порта C в режиме отладки



Кроме того, режим отладки позволяет посмотреть значение любой переменной. Для этого ее нужно выделить в тексте программы, нажать по ней правой кнопкой мыши и выполнить команду *Add 'variable' to... → Watch 1*. В правом нижнем углу программы откроется окно *Watch 1*, которое должно содержать значение переменной (рисунок 1.28). Надо отметить, что для вычисления значения переменной программа должна быть остановлена в окрестностях работы с ней; в противном случае переменная будет значиться, как *<cannot evaluate>* или *<not in scope>*.

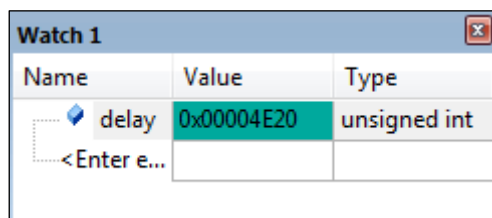


Рисунок 1.28 – Отображение значения задерживающего счетчика

Режим отладки, таким образом, позволяет на любом шаге просматривать значения переменных, находящихся в области видимости, регистров и участков памяти. Данный сервис является мощнейшим инструментом для диагностики системы и идентификации ошибок.

### **Задачи для самостоятельной работы**

1. Разработайте программный проект, осуществляющий мигание светодиодами, расположенными на отладочной плате, согласно инструкциям из главы I.
2. Модифицируйте проект так, чтобы светодиоды мигали в противофазе: пока первый светодиод горит, второй – нет; и наоборот.
3. Выполните пошаговое управление программой в режиме отладки:
  - инвертируйте состояние светодиодов по нажатию кнопки на клавиатуре;
  - отобразите значение переменной, используемой в цикле задержки, и покажите ее декремент;
  - продемонстрируйте динамику регистра MDR\_PORTC -> RXTX в реальном времени.

### **Контрольные вопросы**

1. Что такое микроконтроллер?
2. Какой микроконтроллер используется в рамках данного практикума?
3. Какая архитектура процессорного ядра используется в микроконтроллерах серии 1986BE9х?
4. Какие основные элементы входят в состав отладочной платы?
5. Как подключить отладочную плату к компьютеру?
6. Значение каких параметров определяются при настройке проекта?
7. Как создать программный модуль?
8. Что такое условная компиляция заголовочного файла?
9. Как выполнить загрузку программы в микроконтроллер?
10. Что такое внутрисхемная отладка?