Роман Абраш
г. Новочеркасск
E-mail: **arv@radioliga.com**

Книга по работе с WinAVR и AVR Studio

Продолжение. Начало в №1-5/2010

Редактирование исходных текстов

AVR Studio для исходных текстов программ реализует достаточно широкий набор функций редактирования. Часть функций стандартна для всех Windows-программ, например, работа с буфером обмена (копирование, вставка и т.п.), есть ряд возможностей, специфичных для работы с исходными текстами программ.

По умолчанию некоторые команды редактора доступны только через кнопки панели редактирования, некоторые – через команды меню и(или) горячие клавиши. Если некоторые команды вам удобнее выполнять при помощи горячих клавиш или, наоборот, при помощи мыши (через кнопки), следует перенастроить AVR Studio, как было рассказано в главе «Настройка интерфейса IDE».

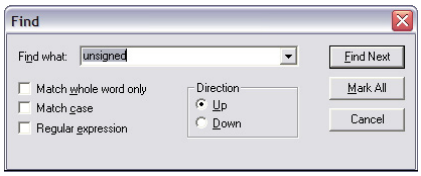
Список всех команд редактора и соответствующих горячих клавиш по умолчанию приведен в **таблице 1**.

Редактор тесно интегрирован с отладчиком, поэтому содержит ряд команд, необходимых при отладке. Хотя этими командами можно пользоваться и во время ввода и редактирования текста программы, рассматриваются они в главе «Компиляция и отладка».

В редакторе можно пользоваться всплывающим по нажатию правой кнопки мыши меню. Оно содержит достаточно много команд, часть из которых доступна только в режиме отладки (и рассматривается в соответствующей главе), другая часть команд доступна всегда, но имеет смысл только для режима отладки.

Поиск и замена текста

При нажатии **Ctrl-F** или выполнения команды меню «**Edit**» **Find** открывается окно поиска текста в текущем файле:



В поле-списке **Find what** можно ввести (или выбрать из выпадающего списка один из вариантов предыдущего поиска) текст, который следует найти. Если в момент вызова команды поиска курсор находился в каком-либо слове, оно автоматически вводится в это поле.

Далее следует задать режимы поиска:

· **Match whole words only** – искать совпадение только целых слов

· **Match case** – искать с учетом регистра символов

· **Regular expression** – введенный текст представляет собой *регулярное выражение* для поиска

Дополнительно указывается направление поиска (**Direction**) – вверх (**Up**) или вниз (**Down**).

Нажатие кнопки **Find Next** приведет к тому, что будет найдено и выделено следующее (по отношению к положению курсора) заданное вхождение искомого текста. Если в процессе поиска будет достигнут конец файла – поиск продолжится с начала, т.е. будет происходить по кругу в указанном направлении.

Если заданного текста не найдено – выводится сообщение.

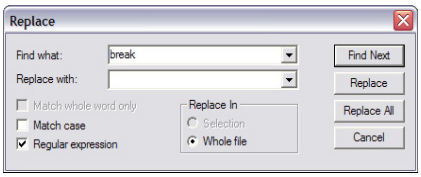
Кнопка **Mark All** приведет к тому, что будут найдены все вхождения указанного текста в файле и каждая строка с этим текстом будет помечена *закладкой*.

Поиск по регулярному выражению – это интересная опция, которая позволяет осуществлять поиск по весьма сложным алгоритмам. О формате регулярных выражений[22] можно почитать, например, в [4]. Чтобы немного пролить свет на эту возможность, приведем ряд примеров регулярных выражений, которые можно использовать в своей работе:

Нужно найти	Регулярное выражение для этого
Любой элемент массива, адресуемый целочисленной константой	**\[\s*\d*\s*\]**
Функцию, имя которой начинается с подчеркивания	**\b\s*_+\w*\s*\(**

Приведенные примеры регулярных выражений достаточно «интеллектуальны». Например, первое выражение позволит найти текст «**[1]**», «**[128]**» и «**[0]**», но проигнорирует «**[a]**» или «**[i++]**». Второе выражение обнаружит функцию «**_demo(void)**», но проигнорирует «**func_demo()**».

Поиск с заменой осуществляется при помощи команды меню «**Edit**» **Replace**. Окно для этой команды похоже на рассмотренное ранее:



Небольшое отличие заключается в том, что имеется два поля-списка: первое **Find what** для поиска текста, второе **Replace with** – для указания текста, который будет заменять найденный. Кроме того, вместо направления поиска указывается диапазон поиска (**Replace In**): только внутри выделенного текста (**Selection**) или во всем файле (**Whole file**).

[22] Регулярные выражения вполне заслуживают отдельной книги.

Кнопка **Find Next**, как и ранее, позволяет найти и выделить очередное вхождение текста, кнопка **Replace** – заменить найденный текст другим, а кнопка **Replace All** – найдет и заменит все вхождения в указанной области.

Как и ранее, для поиска можно использовать регулярные выражения.

Как было уже упомянуто, имеется возможность поиска текста в нескольких файлах, для чего служит команда меню «**Edit**» **Find in Files**.

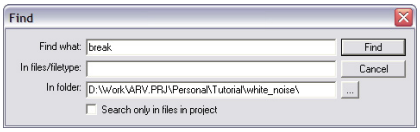


Таблица 1

Действие	Сочетание клавиш
Перемещение по тексту	
На символ влево	
На символ вправо	
На строку вверх	
На строку вниз	
Прокрутка текста на строку вверх	Ctrl –
Прокрутка текста на строку вниз	Ctrl –
В конец строки	End
В начало строки	Home
На страницу вниз	PgDn
На страницу вверх	PgUp
В конец документа	Ctrl – End
В начало документа	Ctrl – Home
К началу следующего слова	Ctrl –
Кначалу предыдущего слова	Ctrl –
Вставка табуляции	Tab
Переход к предыдущей позиции табуляции	Shift –Tab
Операции с закладками	
Включить/отключить закладку в текущей строке	Ctrl – F2
Переход к следующей закладке	F2
Переход к предыдущей закладке	Shift – F2
Удалить все закладки	Ctrl – Shift – F2
Поиск и замена	
Поиск	Ctrl – F или Alt – F9
Поиск следующего вхождения текста	F3
Поиск предыдущего вхождения текста	Shift – F3
Выделение и удаление	
Выделить все	Ctrl – A
Удалить слово правее курсора	Ctrl – Del
Удалить слово левее курсора	Ctrl – BkSp
Удалить строку	Ctrl – L
Выделить	Shift + команды перемещения по тексту
Отменить выделение	Esc
Дополнительно	
Отменить правку	Alt – BkSp
Отменить отмену правки	Ctrl – Shift – BkSp

Окно поиска в файлах более простое, содержит всего три поля для ввода:

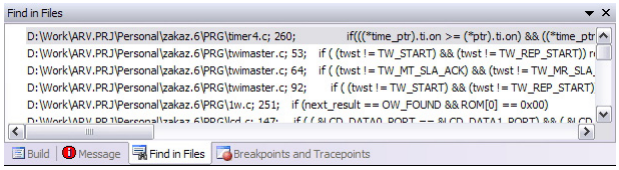
· **Find what** – текст, который надо найти

· **In files/filetype** – шаблон просматриваемых файлов (например, «*.c» - исходные тексты программ на Си)

· **In folder** – папка, в которой осуществляется просмотр указанных файлов.

Опция **Search only in files in project** ограничивает поиск файлами, составляющими текущий проект, отключая указанную папку.

При поиске в файлах нельзя использовать регулярные выражения. Введенный текст ищется с учетом регистра символов. Все найденные вхождения помещаются в окне вывода результатов поиска примерно в таком виде:



В каждой строке списка указывается полный путь к файлу, в котором найден указанный текст, затем указывается номер строки с вхождением текста и сама строка. Двойной щелчок на строке в этом окне приведет к тому, что будет открыт нужный файл, и курсор будет установлен в нужную строку.

В окне результатов поиска можно выполнить ряд других действий при помощи всплывающего по нажатию правой кнопки мыши контекстного меню:

· **Clear** – очистить окно

· **Copy** – скопировать выделенную строку в буфер обмена

· **Font** – изменить шрифт для текста в окне

· **Default Font** – установить шрифт по умолчанию

· **Help on Output** – просмотреть справку об окне

Опции **Info** (информация), **Warning** (предупреждения), **Error** (ошибки) и **Timestamp** (дата и время) позволяют указать, какого рода информация должна отображаться в этом окне.

Компиляция и отладка

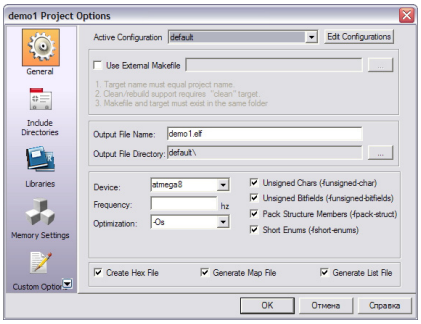
Так как проект может состоять из нескольких модулей, различают *компиляцию* модуля и *сборку* (построение) проекта. Сборка осуществляется компоновщиком из уже откомпилированных объектных модулей, а сама компиляция исходных текстов в объектные файлы выполняется компилятором.

Каждый модуль может быть откомпилирован отдельно. Сборка проекта вызывает автоматическую перекомпиляцию тех исходных файлов, которые изменились с момента предыдущей сборки. Если исходные тексты модуля не изменялись – при сборке используется ранее полученный объектный файл, что существенно ускоряет процесс. Иногда требуется процесс очистки (***Clean***), т.е. удаления всех ранее созданных объектных файлов, чтобы заново перекомпилировать их все.

Традиционно процесс компиляции и сборки проекта автоматизируется при помощи так называемого ***make***-файла, в котором на особом скриптовом языке описаны все действия, которые нужно последовательно выполнить, в результате процесс напоминает исполнение команд в bat-файле. Однако, среда AVR Studio позволяет отказаться от использования этого подхода, предоставляя альтернативу гораздо более привычную для Windows – графический интерфейс. Фактически, генерация и исполнение make-файла осуществляется, однако она скрыта от пользователя.

Параметры компиляции проекта

Ранее уже упоминались команды в главном и контекстном меню проекта, выполняющие настройку параметров компиляции. Рассмотрим диалоговое окно, которое при этом открывается.



Это окно настройки параметров компиляции и сборки проекта.

В левой части его имеется область для выбора групп параметров в виде графических символов, а остальная часть служит для настройки соответствующих опций.

Всего 5 групп параметров:

· **General** – основные

· **Include Directories** – подключаемые директории

· **Libraries** – библиотеки

· **Memory Settings** – настройки параметров памяти

· **Custom Options** – параметры пользователя

General – основные

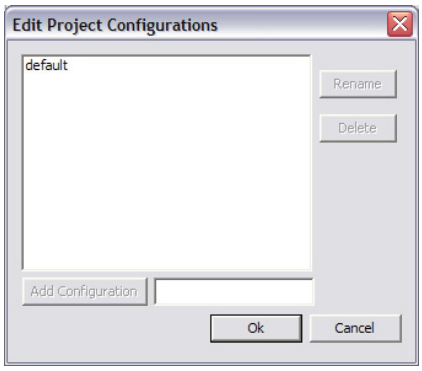
Рассмотрим опции в группе основных по порядку их размещения в окне – сверху вниз.

Первой следует выбор текущей конфигурации проекта – **Active Configuration**. Вы можете выбрать из выпадающего списка наименование нужной конфигурации или, нажав на кнопку **Edit Configurations**, добавить новую или изменить существующие.

Несколько слов о том, что такое конфигурация, которая уже упоминалась несколько раз ранее. Разрабатываемый проект может быть рассчитан на использование разных микроконтроллеров, или же на использование какой-либо разной внешней периферии, или предполагается, что проект будет портирован на другое семейство микроконтроллеров и т.п. В каждом конкретном случае наверняка будут иметься небольшие вариации исходных текстов всех модулей, параметров конфигурации и т.п. Программист стоит перед выбором: или создавать набор почти одинаковых файлов для каждого случая, или каким-то образом учесть разные варианты в одних и тех же файлах (способы для этого уже рассмотрены – см. «Директивы условной компиляции»). Но остается открытым вопрос, как упростить процесс выбора нужного варианта. Вот для этого как раз и существует так называемая конфигурация проекта, которая подразумевает наличие нескольких вариантов настроек для одного и того же проекта.

По умолчанию всегда существует одна конфигурация – **default** (по умолчанию). Если требуется создать вариант проекта, предположим для разных типов микроконтроллеров – нужно создать нужное количество конфигураций (далее будет сказано, как). Для каждой конфигурации задается свой тип микроконтроллера, свои уникальные (если нужно) подключаемые файлы, определяются константы, макросы и т.п. Все это автоматически сохраняется при сохранении проекта. Когда потребуется – следует выбрать нужную конфигурацию, т.е. сделать ее текущей, и выполнить очистку и пересборку проекта.

Вернемся к диалогу настроек. При нажатии кнопки **Edit Configurations** (редактирование конфигураций) открывается окно следующего вида:



В нем перечислены все существующие конфигурации (в данном случае – единственная **default**). Выбрав нужную из списка, вы можете ее удалить кнопкой **Delete** или переименовать кнопкой **Rename**. Ниже списка имеется поле ввода названия новой конфигурации – если вы введете там ее имя, то активируется кнопка **Add Configuration** (добавить конфигурацию). Нажатие этой кнопки добавит вашу конфигурацию к списку. Не допускается задавать имя конфигурации с пробелами или с символами кириллицы.

Все параметры проекта, рассматриваемые далее, относятся исключительно к ***выбранной конфигурации***. Если их несколько – придется настроить все параметры отдельно для каждой.

Следующей идет опция **Use External Makefile** (использовать внешний **make**-файл). Если эта опция активирована, то будет использован указанный в соответствующем поле make-файл. Использование этой возможности подразумевает наличие этого самого make-файла, правильно создать который – целая наука. В подавляющем большинстве реальных практических случаев вам это не потребуется.

Далее предлагается указать имя выходного файла проекта – **Output File Name** и директории, куда он будет помещен – **Output File Directory**. По умолчанию имя выходного файла совпадает с именем проекта (но с расширением **elf**), а имя директории – с именем выбранной конфигурации. Сама директория по умолчанию создается в папке проекта. Как правило, нужды изменять значение этих параметров не возникает.

Далее следует группа важных параметров компиляции проекта.

Device – выбор модели микроконтроллера. Из выпадающего списка вы можете выбрать любую поддерживаемую компилятором модель. Учтите, что выбранная здесь модель контроллера не обязательно должна совпадать с той, что была указана в момент создания проекта (см. «Мастер проектов»), однако во избежание недоразумений не следует допускать такого разнобоя[23].

Frequency – тактовая частота контроллера. Вы должны указать в герцах значение тактовой частоты, с которой предполагается использовать микроконтроллер. Этот параметр важен для правильной генерации кода некоторых библиотечных функций. В программе это значение всегда доступно в виде константы **F_CPU** (т.е. эту символьную константу можно использовать при написании программы). Рекомендуется всегда задавать этот параметр.

Optimization – уровень оптимизации. Можно выбрать из выпадающего списка любой доступный вариант. По умолчанию устанавливается режим наиболее сильной оптимизации по размеру кода – Os.

Unsigned Chars – опция, указывающая компилятору, что тип char должен рассматриваться как беззнаковый. В скобках приводится параметр командной строки компилятора для этой же цели. Следует с осторожностью пользоваться этой опцией!

Unsigned Bitfields – беззнаковые битовые поля. Опция влияет на структуры типа битовое поле[24].

Pack Structure Members – упаковывать поля структур. Опция указывает компилятору использовать так называемый «упакованный» формат хранения в памяти данных полей структуры. В некоторых случаях это может вызвать несовместимость с исходными текстами, написанными для других компиляторов, хотя в большинстве случаев дает экономию памяти данных.

Short Enums – короткие списки. Опция указывает компилятору, что следует использовать для хранения перечисляемых констант[24] тип short int.

Как правило, все эти 4 опции можно оставить по умолчанию (они активированы), и только в случае возникновения ошибок компиляции, связанных с ними, отключать.

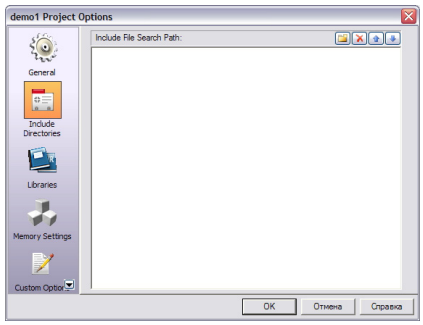
Завершают этот раздел параметров проекта три опции генерации компилятором вспомогательных файлов[25].

Create Hex File – создать Hex-файл. Если вы планируете использовать программатор для «прошивки» микроконтроллера вашей программой – включите эту опцию, т.к. формат **Hex** – де-факто стандартный формат данных для всех типов программаторов.

Generate Map File – генерировать файл «карты памяти». Карта памяти позволит детально рассмотреть результат работы компилятора: выяснить, какие переменные в каких областях памяти находятся, сколько места занимают и т.п. Бывает очень полезно при «глубинной» отладке или оптимизации. Отключение этой опции ускоряет компиляцию.

Generate List File – генерировать листинг. Листинг – это файл, в котором одновременно виден исходный текст на языке Си и его ассемблерный эквивалент, а так же машинные коды соответствующих инструкций и т.п. информация. Может быть полезен при отладке или при изучении работы компилятора. Отключение этой опции ускоряет компиляцию.

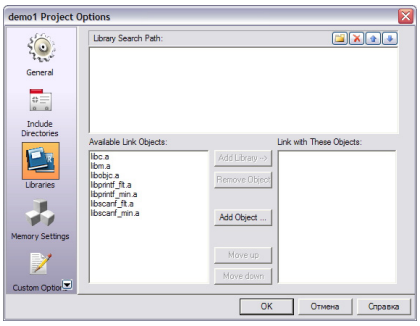
Include Directories – подключаемые директории



Эта группа опций представлена единственным (и по умолчанию пустым) списком директорий, в которых компилятор последовательно будет искать файлы, подключенные директивами **#include**. Список по умолчанию пуст неспроста – указывать дополнительные пути имеет смысл лишь в том случае, когда вы имеете пакеты библиотек или исходных файлов модулей сторонних разработчиков (или свои собственные универсальные «заготовки»).

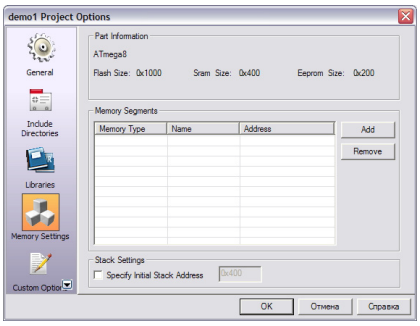
Добавить директорию можно при помощи кнопки , удалить – , а кнопками и можно изменить порядок просмотра директорий компилятором. Учтите, что компилятор использует первый найденный подходящий файл.

Libraries – библиотеки



Группа настроек библиотек служит для указания компоновщику использовать готовые библиотечные файлы (архивы) или объектные модули. Параметры по умолчанию подойдут в большинстве случаев. Однако, если необходимо, вы можете добавить пути поиска библиотек (аналогично рассмотренному способу в предыдущей главе), а так же указать конкретные варианты объектных модулей стандартных библиотек, которые следует использовать при сборке вашего проекта. Для этого в списке **Available Link Objects** (доступные для компоновки объекты) нужно выбрать требуемый файл и, нажав кнопку **Add Library** (добавить библиотеку), добавить его в список прикомпоновываемых объектов **Link with These Objects**. Кнопка **Remove Object** позволяет удалить файл из числа прикомпоновываемых, кнопки **Move Up** (переместить выше) и **Move Down** (переместить ниже) изменяют порядок файлов в списке. Наконец, кнопка **Add Object** позволит добавить файл, которого нет в левом списке, к списку справа.

Memory Settings – настройки параметров памяти



Группа параметров памяти позволяет управлять пользовательскими секциями (см. далее «Секции памяти»). В верхней части окна приводятся сведения о памяти выбранного микроконтроллера (**Part Information**):

Flash Size – объем памяти программ

Sram Size – объем ОЗУ

Eeprom Size – объем EEPROM

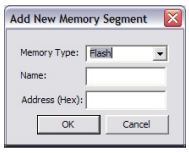
Ниже располагается список заданных пользователем секций в виде таблицы, состоящий из трех столбцов: **Memory Type** (тип памяти), **Name** (имя секции) и **Address** (стартовый адрес секции). Добавить свою

[23] Мастер проекта задает модель для отладки, в то время как компилятор может генерировать код для любой другой модели. Разумеется, ожидать адекватного поведения отладчика при разных моделях контроллера не приходится.

[24] Эта возможность языка Си не рассматривалась в кратком введении в язык. Принципиально, это никак не скажется на эффективности ваших программ.

[25] Фактически, эти файлы генерируются дополнительными утилитами, входящими в комплект WinAVR, однако с точки зрения программиста это несущественно.

секцию вы можете, нажав кнопку **Add**, при это откроется следующее окно:



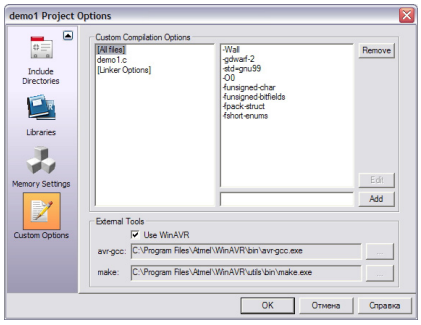
Поля ввода в этом окне совпадают с наименованиями столбцов в таблице: из выпадающего списка можно выбрать тип памяти (память программ **Flash**, ОЗУ **Sram** или **EEPROM**), затем следует ввести имя секции и, в заключение, ее стартовый адрес. Учтите, что компоновщик считает, что секция ОЗУ начинается с адреса **0x800000**, а секция EEPROM – с адреса **0x810000**, т.е. к фактическому адресу в пространстве ОЗУ или EEPROM нужно прибавить соответствующее смещение.

На добавленную секцию затем можно сослаться в программе, указав соответствующий атрибут.

Удалить секцию из таблицы можно нажатием кнопки **Remove**.

В нижней части окна имеется опция управления участком памяти, выделяемого для организации стека – **Specify Initial Stack Address**. Если активировать эту опцию, то можно задать адрес начала области ОЗУ, выделяемой под стек (дополнительно см. «Динамическое распределение памяти»).

Custom Options – параметры пользователя



Последняя группа параметров позволяет пользователю добавить опции к командной строке компилятора и компоновщика (**Custom Compilation Options**). Это дает максимальную гибкость в настройке параметров компиляции, однако требует достаточно высокой квалификации пользователя.

Слева находится окно со списком всех файлов проекта и парой «глобальных» элементов – **All files** (все файлы) и **Linker Options** (параметры компоновщика). Справа окно со списком параметров командной строки, используемых для обработки выбранного в левом списке элемента. Т.е. если вы желаете, чтобы все файлы компилировались с одним и тем же набором параметров – выберите слева вариант **All files**.

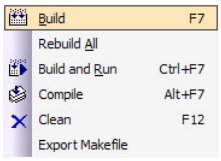
По умолчанию в списке справа перечислены все опции, которые были заданы на других страницах диалога настройки в «интерактивном режиме» – присмотритесь: вы увидите **–funsigned-char**, которая соответствует опции **Unsigned Chars** из первой группы, указание константы **F_CPU** и т.п. Удаление (кнопкой **Remove**) элемента списка приведет к соответствующему изменению поведения компилятора и соответственно отразится в других разделах настроек. Некоторые опции не имеют интерактивных аналогов, но могут серьезно влиять на компиляцию. В главе «Опции командной строки» будет рассказано о параметрах командной строки, которые вы можете дополнительно добавить к этому списку. Делается это вводом соответствующей строки в поле под правым списком и нажатию кнопки **Add** (добавить). Если требуется изменить введенную ранее строку – надо выделить ее в списке и нажать кнопку **Edit,** она скопируется в поле редактирования (не забудьте затем снова нажать **Add** для возврата строки в список).

В нижней части имеется группа опций **External Tools**, которая позволяет изменить инструментарий, необходимый для компиляции, указав другие программы компилятора **avr-gcc** и утилиты **make**. Категорически не рекомендуется изменять значения по умолчанию.

Сборка проекта

Рассмотрим меню «Build», ранее только упоминавшееся.

Меню состоит из следующих пунктов:



· **Build** (собрать) – выполняет сборку проекта, компилируя при необходимости обновленные исходные файлы модулей.

· **Rebuild All** (пересобрать все) – выполняет принудительную перекомпиляцию всех файлов проекта с последующей сборкой.

· **Build and Run** (собрать и запустить) – выполняет сборку проекта и включает режим отладки, т.е. запускает программу на исполнение.

· **Compile** (компилировать) – выполняет компиляцию только одного текущего редактируемого файла (модуля)

· **Clean** (очистить) – удаляет все результаты предыдущих компиляций всех модулей

· **Export Makefile** – сохраняет в makefайле все настройки компиляции для последующего использования компилятором в консольном режиме.

Выполнение любой команды меню, кроме последнего, заключается в автоматически (и незаметно для пользователя) выполняющихся следующих этапах:

1. Компиляция при помощи консольного компилятора **avr-gcc.exe** одного или всех модулей с «перехватом» всей выводимой компилятором информации и выводом ее в окно информации на закладке «**Build**» (окно Output – см. «Рабочее пространство»).

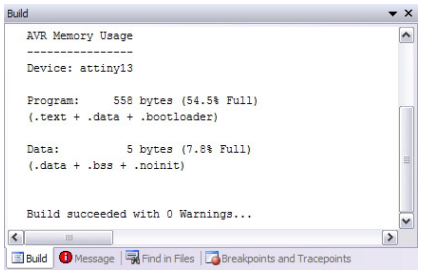
2. При отсутствии ошибок на первом этапе (и при необходимости) – запуск компоновщика с аналогичным перехватом выводимой информации.

3. При отсутствии ошибок компоновки – запуск вспомогательных утилит для генерации различных выходных файлов с аналогичным «перенаправлением» выводимых ими сообщений.

В итоге у пользователя создается впечатление, что AVR Studio сама выполняет компиляцию и т.п. действия.

Запуск компилятора сопровождается выводимой командной строкой в окне информации на закладке «**Build**», отмечаемой зеленым кружочком.

Если компиляция модуля прошла без ошибок, то в окне появляется сообщение «**Build succeeded with 0 Warnings...**» – *компиляция успешна, 0 предупреждений*. Разумеется, количество предупреждений может быть и ненулевым. В том случае, если выполнялась сборка проекта, дополнительно будет выведены сведения о полученном объектном модуле проекта, и окно будет

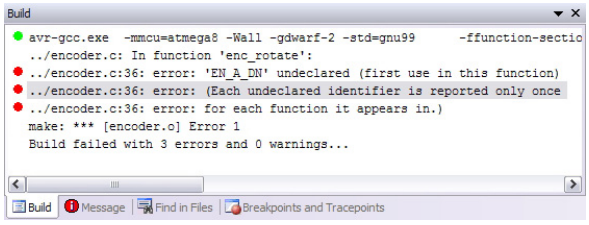


иметь следующий вид:

В приведенной на рисунке итоговой сводке указывается объем занятой памяти микроконтроллера attiny13: программа занимает 558 байт (что составляет 54,5% от всего имеющегося пространства), данные (секций «.data», «.bss» и «.noinit») занимают 5 байтов (что составляет 7,8% всей доступной памяти). Эти сведения помогут вам сделать выводы о ресурсоемкости вашего проекта.

Ошибки компиляции

Если на этапе компиляции обнаруживаются ошибки в тексте модуля, выводится соответственно иное сообщение «**Build failed with 3 errors and 0 warnings...**» – *компиляция неудачна, 3 ошибки и 0 предупреждений* (и снова – число ошибок и предупреждений может быть иным). Подобное сообщение обязательно предваряется указанным количеством сообщений об ошибках (помеченных красным кружочком) и предупреждениях (помеченных желтым кружочком), например, так:



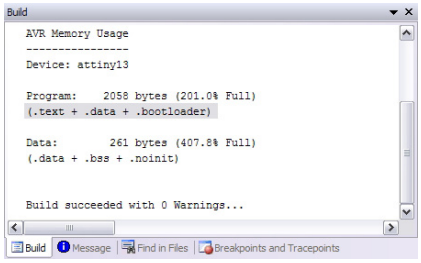
На рисунке показано, что при компиляции модуля «**encoder.c**» в функции этого модуля «**enc_rotate**» обнаружены следующие три ошибки (как правило, одна ошибка может порождать несколько следующих за ней): в строке 36 найден ранее не описанный символ EN_A_DN (остальные две ошибки – это собственно не ошибки, а

продолжение описания первой ошибки[26]). После сообщений об ошибках следует строка-уведомление о том, что сборка проекта невозможна, т.к. есть ошибка.

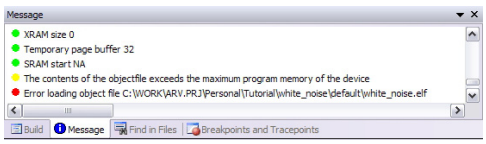
К сожалению, привести список всех сообщений об ошибках, распознаваемых компилятором, нет никакой возможности – он слишком обширен. Чтобы более-менее ориентироваться в них, следует хоть немного знать английский язык – фразы весьма примитивны, перевод их не вызывает сложностей буквально спустя две-три попытки.

Процесс работы над проектом, как правило, обязательно сопровождается неоднократной компиляцией модуля с последующим исправлением ошибок. Чтобы этот процесс был более удобен – AVR Studio реализует удобный механизм: двойной щелчок мышкой на строке с сообщением об ошибке приводит к тому, что в редакторе откроется нужный файл (если он не был еще открыт), а курсор будет установлен в строку с ошибкой, причем сама строка будет дополнительно помечена синей стрелочкой – вам останется лишь исправить ошибку.

Отдельно следует оговорить случай, когда после компиляции программы выясняется, что размер объектного модуля слишком велик, чтобы уместиться в доступной памяти микроконтроллера. Окно результатов компиляции при этом внешне выглядит, как при верной компиляции, что не должно вводить в заблуждение:



Прекрасно видно, что программа «требует» наличия **201%** памяти, что, разумеется, невозможно. В том, что это действительно ошибка, вы убедитесь в момент, когда попытаетесь приступить к отладке программы (см. «Средства виртуальной отладки»): вместо начала отладки произойдет переключение информационного окна на закладку «**Message**»:



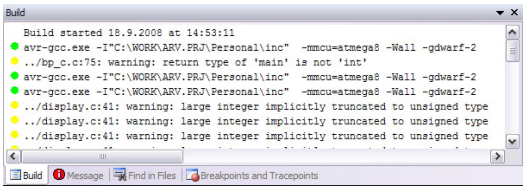
В отмеченной желтым кружком строке сказано, что «*Содержимое объектного файла превышает размер программной памяти выбранного микроконтроллера*», а следующая строка, отмеченная красным, указывает на ошибку загрузки этого объектного файла.

Борьба с размером программы – сложный процесс. Иногда слишком большой объем является следствием неудачно выбранного режима оптимизации, реже – следствием крайне неаккуратного программирования. Следует помнить, что очень требовательными к памяти программ являются любые алгоритмы с использованием чисел с плавающей точкой – только добавление в программу единственной операции деления с дробным результатом может увеличить объем программы более чем на 1,5 килобайта! Наконец, ошибка может быть и в выборе микроконтроллера с заведомо недостаточным объемом памяти.

Предупреждения компилятора

Предупреждения компилятора – это сообщения о найденных в тексте программы «подозрительных» или потенциально опасных мест, неоднозначностей, допускаемых синтаксисом языка и т.п. В некоторых случаях предупреждение – это завуалированная ошибка, но чаще это просто «очистка совести» компилятора за слишком широкие вольности, допускаемые языком Си.

В окне информации предупреждения компилятора выводятся на закладке «**Build**», и отмечаются желтыми значками:



Как и при ошибках, двойной щелчок на строке с предупреждением перенесет курсор в строку, где кроется причина этого сообщения.

Опасны ли предупреждения? Скорее следует признать, что они таят в себе некоторую опасность именно за счет своей «сомнительности», т.е. компилятор «чувствует», что тут что-то не так, но назвать это ошибкой – «язык» у него не поворачивается (извините за столь одушевленную образность). Например, на рисунке самое первое предупреждение гласит «**../bp_c.c:75: warning: return type of "main" is not "int"**» – *тип функции main – не int*. Опасно ли это? 100% нет – для микроконтроллеров совершенно без разницы, какое значение возвращает функция **main**. А несколько следующих предупреждений уже иного рода: «**../display.c:41: warning: large integer implicitly truncated to unsigned type**» – *большое число укорочено до размера unsigned int*. Это уже должно насторожить: согласитесь, что если число **2L** будет укорочено до **int** – это не страшно, но вот если **24765134** укоротить до **int** – это вряд ли может устроить. Поэтому надо внимательно проанализировать строки в программе, приведшие к предупреждениям. В частности, в случае на рисунке, предупреждения были вызваны строками типа **char var = ~2**. Почему так вышло? Все просто: по умолчанию результат любой операции в Си, если иное не оговорено, трактуется как **int**, а константа **2** по умолчанию трактуется компилятором, как **char** (см. «Константы»). Таким образом, операция инверсии **~2** из **char** получит **int**, т.е. число со знаком. Переменная **var** имеет тип **char**, т.е. меньший, чем **int** – чтобы значение «влезло» в переменную, оно укорачивается путем отбрасывания старшего байта – об этом компилятор и предупреждает. В этом конкретном случае никакой опасности нет, потому что поведение компилятора совпадает с намерением программиста получить просто инверсное представление байта 2, однако поручиться, что такое же предупреждение во всех случаях определенно безопасно, нельзя.

По возможности следует стремиться к тому, чтобы предупреждений при компиляции не возникало.

Средства виртуальной отладки

Допустим, компиляция проекта прошла без ошибок и предупреждений. Означает ли это, что программа заработает в реальной схеме? Вовсе нет! Записанная без ошибок неверная инструкция к действию не может привести к успешному результату. То есть отсутствие синтаксических ошибок, проверяемых компилятором, вовсе не означает отсутствия логических или алгоритмических ошибок программиста. Отладка – это как раз процесс поиска логических и алгоритмических ошибок.

AVR Studio предлагает программисту удобный способ проследить за тем, как его программа исполняется микроконтроллером, при этом программист может не только наблюдать за процессом, но и вмешиваться в него!

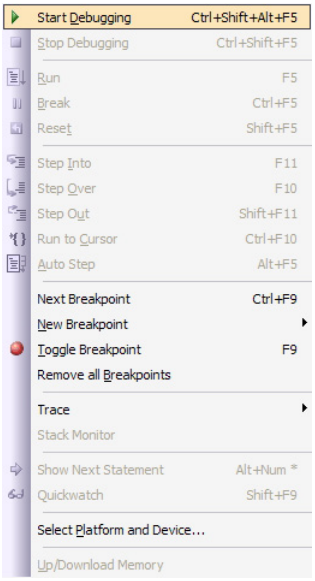
Реализуется это следующим образом: AVR Studio эмулирует[27] работу микроконтроллера, т.е. имитирует исполнение им программы. Благодаря тому, что мощность современных компьютеров очень велика, особых проблем проимитировать работу куда более слабого AVR не возникает. При этом AVR Studio заполняет «виртуальные» регистры и ячейки памяти значениями из программы, исполняя написанные программистом строки. Содержимое любой переменной в программе, и даже любого регистра или ячейки памяти, программист может в любой момент изменить, проимитировав любые варианты поступления в программу внешних данных. Такое «виртуальное» исполнение программы позволяет найти почти все отклонения поведения программы от задуманного.

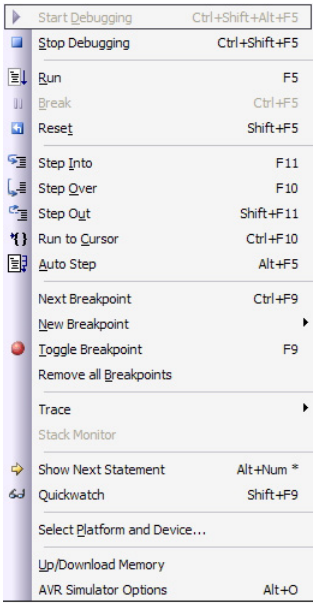
Рассмотрим меню «**Debug**», команды которого управляют всем процессом отладки.

На рисунках (*см. на следующей странице*) показаны варианты меню «**Debug**» соответственно для выключенного режима отладки и для включенного.

[26] Особенность «перехвата» сообщений – каждая строка сообщения воспринимается как новая ошибка.

[27] Наряду с термином «эмуляция» применяется и термин «симуляция». Разделение между ними скорее надуманное: первый чаще применяют в тех случаях, когда используются аппаратные средства, имитирующие действия микроконтроллера или его узлов, второе – при использовании чисто программных методов имитации. В настоящей книге разницы между терминами не делается, т.к. в русском языке это слова-синонимы.





· **Start Debugging** – запуск отладки. Команда переводит AVR Studio в режим эмуляции микроконтроллера, при этом могут автоматически появиться на дисплее новые окна и панели.

· **Stop Debugging** – остановка отладки. Команда завершает режим отладки, возвращая вид интерфейса к исходному.

· **Run** – исполнение. Команда запускает программу на исполнение. Обычно этой команде должны предшествовать команды установки точек останова, иначе возможности наблюдения за ходом работы программы практически не будет (во время исполнения содержимое отладочных окон не обновляется).

· **Break** – приостановка (пауза) исполнения. Команда активируется только во время исполнения программы по команде Run и позволяет остановить программу принудительно. После остановки исполнения обновляется содержимое окон отладки.

· **Reset** – сброс. Команда позволяет проимитировать поступление сигнала Reset на микроконтроллер, т.е. вынуждает программу начаться заново.

· **Step Into** – «шаг внутрь». Пошаговое исполнение программы с заходом в функции.

· **Step Over** – «шаг поверх». Пошаговое исполнение программы, причем обращения к функциям исполняются, как «одна команда».

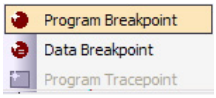
· **Step Out** – «шаг наружу». Команда вызывает исполнение всех оставшихся строк (операторов) в функции до выхода из нее, т.е. остановка происходит в момент возврата значения функции.

· **Run to Cursor** – «дойти до строки с курсором». Команда запускает программу на исполнение с текущего места и останавливает, как только исполнение дойдет до строки, в которой находился курсор в момент подачи команды.

· **Auto Step** – автопошаговое исполнение. Программа начинает исполняться в режиме «автоматического исполнения» команд Step, подаваемых через определенные интервалы времени (можно регулировать).

· **Next Breakpoint** – перейти к следующей точке останова.

· **New Breakpoint** – подменю команд установки точек останова.



Возможно три варианта точек остановки: **Program Breakpoint** – остановка по месту в программе, **Data Breakpoint** – остановка по изменению данных (переменных) в программе и **Program Tracepoint** – точка трассировки программы (не является точкой останова по своей сути).

· **Toggle Breakpoint** – включение/выключение точки останова в строке программы

· **Remove all Breakpoints** – удалить все точки останова из программы

· **Trace** – подменю управления режимами трассировки программы (рассматривается в последующих главах).

· **Stack Monitor** – команда зарезервирована

· **Show Next Statement** – показать строку, которая должна исполняться. Команда позволяет быстро переместиться к очередной исполняющейся строке из любого места программы.

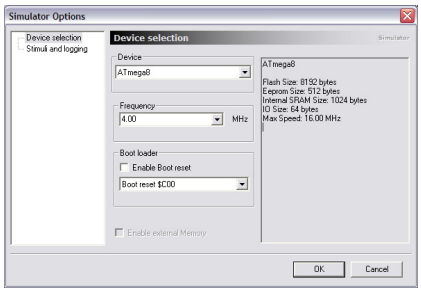
· **Quickwatch** – добавление переменной в окно просмотра. Добавляется переменная, на которой установлен текстовый курсор в тексте программы.

· **Select Platform and Device** – выбрать отладочную платформу и микроконтроллер. Позволяет изменить на время отладки настройки, сделанные при создании проекта.

· **Up/Download Memory** – сохранить/загрузить область памяти.

· **AVR Simulator Options** – настройки эмулятора AVR.

В первую очередь рассмотрим последний пункт меню – настройку параметров эмулятора (данный пункт доступен только в режиме отладки), а остальные команды будут рассмотрены более подробно далее, в контексте различных способов и режимов отладки (*см. следующую колонку*):



В левой части окна имеется древовидный список групп настроек эмулятора. Количество этих групп зависит от того, какой именно способ отладки был избран при создании проекта: мы рассматриваем программный симулятор, а для аппаратных средств отладки могут быть доступны и другие группы параметров.

В базовом варианте доступно две группы: **Device selection** – выбор устройства и **Stimuli and logging** – стимуляция и протоколирование.

В группе **Device Selection** можно выбрать тип эмулируемого микроконтроллера из раскрывающегося списка **Device**. Важно понимать, что обязательно нужно всегда указывать одинаковые типы и в настройках компилятора и здесь, иначе отладка будет некорректной или вообще невозможной. По умолчанию такое соответствие обеспечивается автоматически.

Так же можно указать значение тактовой частоты контроллера при отладке: выбрать из списка **Frequency** или ввести значение вручную. Для корректной эмуляции нужно так же обеспечить совпадение с частотой, указанной при компиляции.

Наконец, если программа содержит секцию загрузчика (**Boot loader**), можно и нужно указать адрес начала этой секции и указать режим поведения при сбросе микроконтроллера (отмеченная опция **Enable Boot reset** заставит после сброса исполняться функцию загрузчика).

Если микроконтроллер имеет возможность подключения внешнего ОЗУ, становится доступной опция **Enable external Memory**, включающая поддержку эмулятором дополнительного внешнего ОЗУ.

В правой части окна присутствует область, в которой кратко приведены основные характеристики выбранного микроконтроллера. Следует помнить, что максимальное значение тактовой частоты, указанное здесь (и даже в технической документации) не является ограничением для программной эмуляции – вы можете задать значение тактовой частоты хоть 100 МГц – процесс отладки от этого никак не изменится.

Группа **Stimuli and logging** отвечает за режим имитации и протоколирования внешних сигналов на портах микроконтроллера и подробно рассматривается в главе «Имитация входных сигналов и наблюдение выходных».

Ресурсы

4. http://www.rsdn.ru/article/alg/regular.xml

Продолжение в №7/2010