Книга по работе с WinAVR и AVR Studio

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

Продолжение. Начало в №1-8/2010

МИГРАЦИЯ ПРОГРАММ

В этой главе рассматриваются основные проблемы, связанные с миграцией проектов программ для микроконтроллеров, даются общие рекомендации по их преодолению.

Под миграцией понимаются два процесса:

1. Имеется программа для одного компилятора, требуется адаптировать ее для другого.

2. Имеется программа для одного микроконтроллера, требуется адаптировать ее для другого.

Первая ситуация имеет место в случае перехода программиста с одной системы программирования на другую – например, с платного компилятора CVAVR на бесплатный WinAVR. Несмотря на постоянно муссируемую тему о кроссплатформенности языка Си, о стандартах этого языка и т.п. еще никому не удавалось просто взять программу, написанную для одной версии компилятора, и перекомпилировать ее другим компилятором – всегда находятся какие-то препоны, требующие вмешательства.

Связано это отчасти с тем, что каждый производитель программных средств самостоятельно решает, какому стандарту Си соответствовать, а в вопросах, не затрагиваемых стандартами, вообще царит полная анархия.

Однако, влияние языка высокого уровня все-таки велико: усилия для миграции между различными компиляторами относительно мало́. Обычно бывает достаточно изменить имена некоторых функций и/или макросов, а так же имена каталогов и подключаемых файлов. Например, для упомянутого CVAVR константа в памяти программ определяется при помощи ключевого слова flash:

```
flash unsigned int var;
```

в то время как в WinAVR для этой цели служит макрос PROGMEM:

```
PROGMEM unsigned int var;
```

Несколько больше проблем возникает с теми местами программы, которые реализуют работу с такими «переменными». В WinAVR для чтения константы, описанной в памяти программ, используется специальный макрос, в то время как многие другие компиляторы выполняют необходимые операции автоматически. Сравните:

```
// участок программы CVAVR
flash unsigned int var = 12;
for (int i = 0; i < var; i++)

// аналогичный участок для WinAVR
PROGMEM unsigned int var = 12;
for (int i = 0; i < pgm_read_word(&var); i++)
```

Разница, как видите, заметная, хотя принципиально переделка несложная. Аналогичная ситуация возникает и при работе с данными, сохраняемые в EEPROM: WinAVR для этих целей так же использует макросы-функции, а коммерческие компиляторы позволяют просто использовать значение «переменной», описанной в соответствующей памяти.

Ситуация осложняется и другими «нюансами», связанными, например, с тем, как по умолчанию компилятор трактует тип char – как число со знаком или без. Этот, малозначительный на первый взгляд, факт может принести немало хлопот при отладке программы. Ответьте, какое значение получит переменная var после выполнения этого цикла:

```
int var = 0;
for (char i=250; i > 0; i++) var++;
```

Увы, дать правильный ответ, не вспоминая об «истинном» типе char, невозможно: если компилятор считает его числом без знака, то var = 6, а если со знаком, то var будет равно 0.

«Мелочи», вроде способа описания функций-обработчиков прерываний можно даже и не упоминать:

```
// Обработка прерывания по переполнению таймера 0 в CVAVR
interrupt [TIM0_OVF] void timer0_ovf_isr(void){
       TCNT0=0x10;
}

// Обработка прерывания по переполнению таймера 0 в WinAVR
ISR (TIMER0_OVF_vect){
       TCNT0=0x10;
}
```

Вывод, к сожалению, неутешительный: чтобы выполнить миграцию проекта с одного компилятора на другой, необходимо выполнить следующие шаги, проверяя результат пробной компиляцией проекта:

1. Проверить, имеются ли в наличии использованные в исходном тексте подключаемый файлы, доступны ли указанные для них каталоги. При необходимости заменить или изменить их на те, которые являются соответствующими аналогами для нового компилятора. Например, для WinAVR характерно использование #include <avr/io.h>, в то время как для многих других компиляторов используется #include <io.h>.

2. Найти и изменить имена всех функций исходного проекта, для которых существуют соответствующие аналоги в новом компиляторе. Например, функция задержки на несколько микросекунд для CVAVR delay_us() имеет аналог в WinAVR с идентификатором _delay_us().

3. Найти и модифицировать в соответствии с особенностями нового компилятора все описания констант, переменных, макросов и функций, специфичных для старого компилятора. Например, ранее рассмотренные особенности, связанные с прерываниями, EEPROM и т.п.

4. Проверить (возможно, с использованием отладчика) корректность алгоритма программы, полученной после всех предыдущих этапов. В случае неверной работы обратить внимание на переменные типа char, режимы оптимизатора и т.п.

Увы, для выполнения всех этих рекомендаций во многих случаях недостаточно знания особенностей только одного компилятора…

Миграция программы между разными типами микроконтроллеров может потребоваться как по желанию разработчика, так и без, например, в случае снятия с производства устаревшей модели микроконтроллера.

Когда фирма Atmel снимает модель контроллера с производства, то всегда предлагает ей адекватную (лучшую) модель. В этом случае публикуется специальный файл-рекомендация по миграции, в котором изложены все основные отличия между старой и новой моделями, и даются советы по адаптации программ. Если же разработчик принимает решение о замене микроконтроллера другим, это может быть связано с различными причинами, и проблем с миграцией возникает больше.

Если исходный и новый микроконтроллеры имеют аналогичные наборы периферийных устройств, или новый имеет более развитую периферию – миграция практически не вызывает проблем. Максимум, что может потребоваться – это изменить идентификаторы, соответствующие регистрам управления периферией или битам этих регистров. Например, в некоторых моделях микроконтроллеров регистр управления АЦП имеет идентификатор ADCSRA, но в некоторых он видоизменен на ADCSR0A. Такие несоответствия устраняются элементарно – на каждое неверное описание компилятор отреагирует сообщением об ошибке.

Больше проблем может принести ситуация, когда новый микроконтроллер имеет существенные отличия в периферии. В этом случае часто помогает лишь полная переделка всей программы. Например, если исходная программа была написана для микроконтроллера с тремя аппаратными таймерами, а требуется перенести ее на контроллер всего с двумя – это может быть очень серьезной

проблемой! Не меньшей проблемой может быть разница в объеме памяти программ и/или ОЗУ.

Дать осмысленные рекомендации для всех возможных вариантов не представляется возможным, но, как правило, без проблем переносятся программы между контроллерами одного семейства со сходными параметрами: с atmega8 на atmega88 или atmega48, с attiny13 на attiny25; почти всегда возможен перенос с «младшего» микроконтроллера на «старший»: с attiny25 на atmega48, с atmega8 на atmega16 и т.д. Перенос программ со «старшего» семейства контроллеров на «младшее» почти всегда невозможен или сильно затруднен: маловероятно, что программа для atmega16 будет адаптирована для attiny2313. Невозможен принципиально перенос программ, использующих уникальные особенности архитектуры отдельных микроконтроллеров, например программа, написанная для attiny15 (достаточно «слабого» микроконтроллера) и использующая его аппаратный синтезатор частоты для формирования ШИМ, никогда не сможет быть перенесена на гораздо более мощный микроконтроллер atmega16, в котором отсутствует такой синтезатор.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Информация для этого раздела собрана из различных источников, в том числе получена при переводе различной документации и сообщений на форумах, структурирована и снабжена авторскими комментариями и пояснениями. Большая часть информации о компиляторе WinAVR GCC и стандартных библиотечных функциях, являющихся его неотъемлемой частью, получена из оригинальной документации, входящей в его комплект.

По материалам различных Интернет-форумов собраны наиболее часто задаваемые вопросы и приведены ответы на них.

AVR-GCC

Аббревиатура GCC расшифровывается как GNU Compiler Collection, что можно перевести как Коллекция Компиляторов по лицензии GNU, т.е. свободно распространяемое бесплатное программное обеспечение. Иначе говоря, GCC – это семейство бесплатных компиляторов.

GCC поддерживает множество платформ, в том числе микроконтроллерных, практически все языки программирования (от ассемблера до Java), есть версии, работающие во всех известных операционных системах.

В этой книге под GCC подразумевается только компилятор C для платформы AVR, запускаемый в операционной системе семейства Win32 – версия, получившая название avr-gcc. Особенности GCC для других платформ, операционных систем и языков могут упоминаться, но подробно не рассматриваются.

Общие сведения о GCC для AVR

GCC – это компилятор консольного типа, т.е. основная работа с ним осуществляется через ввод большого количества параметров командной строки. Далее будет рассмотрена часть параметров, специфичных для платформы AVR.

GCC – универсальный компилятор, поддерживающий не только язык Си, но и ассемблер и расширения C++. Характерно, что компилятор самостоятельно определяет тип языка исходного текста (по расширению файла) и осуществляет «невидимый» запуск соответствующего модуля, являющегося дополнением к нему. Такими дополнениями может быть C или C++ препроцессор, компилятор ассемблера и т.п.

Компилятор генерирует в результате своей работы либо ассемблерный файл, либо завершенный объектный файл. Каждый отдельный модуль компилируется в отдельный объектный файл. Для объединения объектных файлов в загружаемый образ или конечный объектный файл проекта служит компоновщик. Этот подход характерен для любых компиляторов, GCC не исключение.

Традиционно для упрощения и автоматизации процесса компиляции и компоновки проекта, а так же для ряда вспомогательных задач, используется утилита make, для которой на специальном скриптовом языке описывается «программа» действий по сборке проекта. В комплект WinAVR она включена, однако здесь не рассматривается, т.к. большинство выполняемых ею функций реализуется в интерактивном виде при помощи графической среды разработки AVR Studio.

Особенности языка Си

GCC полностью реализует требования стандартов, с которыми заявлена совместимость. Кроме этого в реализации C поддерживаются некоторые возможности, определенные стандартом только для C++. То есть программист может использовать некоторые возможности, не свойственные классическому C, достигая своих целей с меньшими затратами. Однако при этом следует помнить, что получаемые таким образом исходные тексты могут оказаться несовместимы с другими компиляторами, реализующими только стандартные возможности.

Объявление и инициализация переменных

Допускается использовать символ доллара «$» в именах переменных и функций. Однако это может сделать программу не совместимой с другими компиляторами.

При описании переменных можно использовать присваивание им начальных значений не только в виде констант, но и вычисляемых на основе значений других переменных. То есть допускается такое определение:

```
foo (float f, float g)
     {
       float beat_freqs[2] = { f-g, f+g };
       /* ... */
     }
```

Интересной возможностью является объявление массивов с вычисляемым в процессе выполнения программы размером:

```
FILE *
     concat_fopen (char *s1, char *s2, char *mode)
     {
       char str[strlen (s1) + strlen (s2) + 1];
       strcpy (str, s1);
       strcat (str, s2);
       return fopen (str, mode);
     }
```

Предусмотрена возможность выборочной инициализации элементов массива. Например, описывается массив из 10 элементов, в котором только 5-й и 10-й элементы должны быть проинициализированы определенными значениями, а остальные могут иметь значение по умолчанию (т.е. 0). Чтобы не писать лишнего, допускается использовать такую форму инициализации:

```
int arr[10] = {[4]=15, [9]=25};
```

Это будет эквивалентно следующей традиционной записи:

```
int arr[10] = {0,0,0,0,15,0,0,0,0,25};
```

Отдельно стоит отметить, что порядок выборочной инициализации может быть произвольным, т.е. сначала можно проинициализировать 7-й элемент массива, а затем первый:

```
int arr[10] = {[6]=15, [0]=25};
```

Предусмотрен упрощенный вариант инициализации диапазона элементов массива одинаковыми значениями:

```
int arr[10] = {[0 ... 4]=15}; // первые 5 элементов имеют значение 15
```

При инициализации структур так же можно осуществить выборочную инициализацию их полей:

```
struct point { int x, y; };
point pt = {.y = 10 }; // поле .x остается инициализированным по умолчанию
```

Точечную нотацию для обращения к полям инициализируемой структуры можно использовать и в том случае, если инициализируется массив структур, причем делать это можно выборочно:

```
struct point { int x, y; };
point ptarray[10] = {[3].y = 12, [4].x = 2};
```

Допускается определять структуры, не содержащие полей:

```
struct empty {
};
```

Разрешено определение объединения (**union**) без ключевого слова **typedef**:

```
union foo { int i; double d; };
int x;
double y;
union foo u;
     /* ... */
     u = (union foo) x; // равносильно u.i = x
     u = (union foo) y; // равносильно u.d = y
```

Расширения операторов

Синтаксис некоторых операторов расширен по сравнению со стандартом Си. Например, в операторе switch можно использовать для case диапазон значений:

```
switch(x){
       case 0 :         // обработка одного значения break;
       case 1 ... 5:    // обработка диапазона значений
}
```

Программист должен выделять троеточие пробелами, иначе это может быть интерпретировано неверно, если значения диапазона представляют собой числа.

Допускается сокращенная запись оператора **?**:

```
x ? : y
```

Результатом этого выражения будет x, если x не равно 0, и y в противном случае, т.е. сокращенная запись эквивалентна следующей:

```
x ? x : y
```

Пользоваться расширениями следует с осторожностью. Т.к. получаемый текст программы может быть несовместим с другими компиляторами.

Атрибуты

Атрибут – это особый символ, связанный с определенным объектом (переменной, функцией или типом) и определяющий особенности работы компилятора с этим объектом. Атрибут указывается при помощи специального ключевого слова **__attribute__**, за которым следует собственно список атрибутов, заключенный в двойные круглые скобки:

__attribute__ ((**список_атрибутов**))

Список атрибутов может состоять из одного или нескольких идентификаторов атрибутов, часто в качестве таких идентификаторов используются макросы. В некоторых случаях используется макрос-подобная форма идентификатора, т.е. идентификатор с круглыми скобками, внутри которых указан какой-либо параметр. Несколько идентификаторов разделяются запятыми. Допустим и пустой список идентификаторов, такое назначение атрибута попросту игнорируется.

Для различных объектов определены различные наборы допустимых атрибутов, хотя некоторые атрибуты могут назначаться и разным объектам, например, атрибуты для типов и переменных в своем большинстве не разделяются.

packed

Атрибут для переменной (или типа), определяющий, что переменной должно быть выбрано минимально возможное пространство памяти. Пример:

```
int arr[10] __attribute__ ((packed));
```

unused

Атрибут для переменной, указывающий, что переменная может быть неиспользована в тексте программы. Для определенных с таким атрибутом переменных GCC не выводит предупреждений. Пример:

```
int some __attribute__ ((unused));
```

progmem

Атрибут для переменной, указывающий на то, что переменная на самом деле есть константа, размещенная в памяти программ. Пример:

```
char str[] __attribute__ ((progmem)) = "Строка в памяти программ"
```

deprecated

Атрибут для переменной типа или функции для указания о том, что соответствующий объект не должен использоваться в текущем файле. Это может быть необходимо, если, например, переменная введена лишь для совместимости с другими версиями программы, или же в будущем должна быть удалена. Компилятор выводит предупреждение с указанием на строку программы в которой происходит обращение к переменной с этим атрибутом. Например, компилятор выведет предупреждение на вторую строку следующего кода:

```
extern int old_var __attribute__ ((deprecated));
int new_fn () { return old_var; }
```

section

Формат атрибута следующий:

section("**имя секции**"), где **имя секции** – имя любой определенной секции памяти.

Атрибут указывает, что переменная должна размещаться в указанной секции. Обычно переменные размещаются в секциях **.data** или **.bss**. Программист может разместить переменную в любой иной секции, главное, чтобы эта секция была определена заранее. Пример:

```
struct duart a __attribute__ ((section («DUART_A»))) = { 0 };
struct duart b __attribute__ ((section («DUART_B»))) = { 0 };
char stack[10000] __attribute__ ((section («STACK»))) = { 0 };
int init_data __attribute__ ((section («INITDATA»))) = 0;
```

Данный атрибут применим к переменным и функциям.

alias

Формат атрибута следующий:

alias("**имя функции**"), где **имя функции** – любое ранее определенное имя функции.

Атрибут позволяет назначить новое имя (псевдоним) для функции. Следующий пример показывает, как при обращении к функции **demo()** будет происходить вызов функции **__foo()**:

```
void __foo () { /* Do something. */; }
void demo () __attribute__ ((weak, alias («__foo»)));
```

weak

Атрибут, ограничивающий видимость функции или ее псевдонима текущим модулем.

always_inline

Атрибут, указывающий, что функция всегда должна быть встраиваемой (**inline**). Обычно, пока не включена оптимизация, помеченные ключевым словом **inline** функции не будут таковыми. Более того, они будут включены в код лишь в том случае, если оптимизатор сочтет это необходимым в конкретном контексте программы. Указание же атрибута **always_inline** гарантирует, что функция всегда будет встроена в код.

nacked

Атрибут, указывающий компилятору, что для функции не требуется генерировать код пролога и эпилога, программист обязан самостоятельно выполнить необходимые действия.

noreturn

Атрибут, указывающий, что функция никогда не возвращает управление.

Определено несколько функций, не возвращающих управление, например **abort()** или **exit()**. Эти функции «знакомы» компилятору. Но программист может определить и свои функции аналогичного поведения при помощи атрибута **noreturn**. В этом случае ком-

пилятор генерирует максимально оптимальный код, в котором не производится формирование эпилога, т.е. восстановления стека, освобождения локально выделенной функцией памяти и собственно, инструкция RET, завершающей код любой функции.

Примечание: функции, не возвращающие управление, не должны возвращать какое-либо значение, т.е. должны иметь тип **void**.

Типы поддерживаемых GCC файлов

Как было сказано ранее, GCC автоматически распознает тип содержимого файла по его расширению и выполняет соответствующие этому типу действия. Всего поддерживается более 40 типов файлов. В **таблице 2** перечислены большинство из них с кратким описанием, жирным шрифтом выделены актуальные для программ на Си для AVR.

Таблица 2

Расширение файла	Описание
.c	**Исходный текст на C, подлежащий обработке препроцессором**
.i	**Исходный текст на C, не подлежащий обработке препроцессором**
.ii	**Исходный текст на C++, не подлежащий обработке препроцессором**
.m	Исходный текст на Objective-C, подлежащий обработке препроцессором
.mi	Исходный текст на Objective-C, не подлежащий обработке препроцессором
.M, .mm	Исходный текст на Objective-C++, подлежащий обработке препроцессором
.mii	Исходный текст на Objective-C++, не подлежащий обработке препроцессором
.h	**Заголовочный файл для C, C++, Objective-C или Objective-C++**
.cc, .cp, .cxx, .cpp, .CPP, c++, .C	Исходный текст на C++, подлежащий обработке препроцессором
.H, .hh, .hp, .hxx, hpp, .HPP, .h++, .tcc	Заголовочный файл для C++
.F, .for, .FPP	Исходный текст на фортране, не подлежащий обработке препроцессором
.f90, .f95	Исходный текст на фортране-90/95, не подлежащий обработке препроцессором
.F90, .F95	Исходный текст на фортране-90/95, подлежащий обработке препроцессором
.ads, .adb	Исходный текст модуля на языке Ада
.s	**Исходный текст на ассемблере**
.S, .sx	**Исходный текст на ассемблере, подлежащий обработке препроцессором**

Опции командной строки

Количество параметров командной строки для GCC вообще огромно и составляет значительно больше сотни! К счастью, многие из этих параметров не актуальны для платформы AVR. Часть опций подставляется автоматически при использовании AVR Studio (например, подключение библиотек, установка входного и выходного имени файла, указание путей поиска файлов и т.п.), поэтому здесь будут рассмотрены лишь те опции, которые программист должен при необходимости задавать самостоятельно. Опции, специфичные для расширения C++, не рассматриваются подробно, лишь упоминаются при необходимости.

Важно, что параметры командной строки для GCC *регистрозависимы*, т.е. верхний и нижний регистр символов различается.

Большое количество опций, начинающихся с **–f** или **–W**, имеют 2 формы: *положительную* и *отрицательную*, т.е. первая включает какой-то режим, а вторая – отключает его. Например, для «положительной» опции **–ffoo** ее «антипод» записывается с приставкой «**no-**», т.е. **–fno-foo**. Далее упоминается лишь одна из этих форм: та, которая не принята компилятором по умолчанию.

Далее рассматриваются все опции, удовлетворяющие вышеперечисленным условиям и оговоркам, по функциональным группам.

Выходной формат

Группа опций компилятора, определяющих его поведение.

-c

Компилировать файлы исходных текстов, но не выполнять их компоновку в объектный модуль. При этом генерируются объектные файлы для каждого файла с исходным текстом, результирующие файлы имеют расширение **.o**. Файлы, тип содержимого которых не определен по их расширению, игнорируются.

-S

Прекратить компиляцию на этапе получения ассемблерного текста. При этом для каждого файла с исходным текстом генерируется ассемблерный файл с расширением **.s**, генерация объектного файла не осуществляется. Файлы, тип содержимого которых не определен по их расширению, игнорируются.

-E

Прекратить компиляцию после завершения работы препроцессора. Генерация выходных файлов не происходит – результат направляется в поток стандартного вывода. Файлы, тип содержимого которых не определен по их расширению, игнорируются.

Диалект C

Группа опций, определяющих степень совместимости компилятора с различными версиями стандарта Си.

-ansi

Опция включает поддержку стандарта ANSI-C, для программ на Си это означает поддержку стандарта ISO C90. Опция отключает некоторые расширения GCC, например, поддержку ключевых слов asm или typeof, однако __asm__ и __typeof__ и другие аналогичные ключевые слова-синонимы продолжают функционировать.

-std

Опция позволяет выбрать поддерживаемый стандарт языка. Полный формат опции следующий:

-std=<**ключ**>, где ключ – это один из указанных в **таблице 3**, выделенная жирным опция принимается по умолчанию, если опция –std не указана.

Таблица 3

Ключ	Значение
c89, iso9899:1990	ISO C90 (то же самое, что и опция –ansi
iso9899:199409	ISO C90 в редакции с поправкой №1
c99, c9x, iso9899:1999, iso9899:199x	ISO C99 (пока поддерживается не в полном объеме)
gnu89	ISO C90 с расширениями (включая некоторые расширения C99)
gnu99, gnu9x	ISO C99 плюс расширения GNU. Эта опция станет умалчиваемой, когда C99 будет полностью поддерживаться GCC

Предупреждения компилятора

Группа опций компилятора, позволяющих управлять выводом предупреждений компилятора. Предупреждение обычно выводится в том случае, если в тексте имеется программная конструкция, не являющаяся ошибочной по синтаксису языка, но могущая привести к ошибке во время исполнения программы при определенном стечении обстоятельств.

Многие особенности генерации предупреждений управляются при помощи опций-переключателей (например, начинающихся с –W). Для этих опций во многих случаях придется использовать «негативную» форму, чтобы отключить определенный вид предупреждений.

GCC поддерживает более 100 опций управления предупреждениями компилятора. Формат книги не позволяет рассмотреть их все, поэтому рассмотрена только часть, по мнению автора, наиболее ходовых.

-fsyntax-only

Опция вынуждает компилятор контролировать только синтаксис, ничего более.

-pedantic

Опция включает проверку на соответствие всем расширениям выбранного стандарта. Не следует ожидать, что будет проведена проверка на соответствие всем требованиям стандарта – это неверно. Проверка происходит лишь на соответствие тем возможностям, которые выбранный стандарт требует проверять.

-w

Опция запрещает вывод всех сообщений об ошибках.
Не рекомендуется использовать ее без необходимости.

-Wno-import

Опция блокирует вывод предупреждений о директиве #import

-Wchar-subscripts

Опция включает вывод предупреждения, если элемент массива имеет тип **char**. Это делается для того, чтобы предупредить распространенную ошибку программистов, связанную с тем, что они забывают о том, что во многих случаях тип **char** – это число со знаком.

-Wcomment

Опция включает вывод предупреждения всякий раз, когда внутри многострочного комментария, начинающегося с «/*» встречается снова символ начала такого комментария (т.е. снова «/*»); или когда в однострочном комментарии «//» встречается символ слияния строк «\»

-Wformat

Опция, включающая проверку соответствия строки-формата типам передаваемых переменных для вывода в функциях семейства printf() и scanf().

Опция имеет много дополнительных вариантов, которые могут использоваться для выключения некоторых предупреждений, если глобально включена опция –Wformat.

-Wno-format-extra-args – выключает предупреждение, если в списке выводимых переменных больше значений, чем указано в строке формата. Стандарт Си подразумевает, что лишние значения будут отброшены.

-Wno-format-zero-length – выключает предупреждение, если строка формата имеет нулевую длину. Стандарт Си допускает пустую строку формата.

-Winit-self

Опция включает вывод предупреждения, если переменная в момент описания инициализируется собственным значением, т.е. так:

```
int some = some;
```

Опция действует только в том случае, если включена опция –Wuninitialized, которая в свою очередь включается только при включенной оптимизации.

-Wimplicit-int

Опция вызывает вывод предупреждения, если для переменной не определен тип, т.е. происходит присвоение типа int по умолчанию.

-Wimplicit-function-declaration

Опция вызывает вывод предупреждения, если функция используется до того, как определена.

Имеется сходная опция -Werror-implicit-function-declaration, которая вместо предупреждения в этом случае выводит сообщение об ошибке. Негативный вариант опции -Werror-implicit-function-declaration не поддерживается.

-Wimplicit

Опция, объединяющая действие двух опций сразу: -Werror-implicit-function-declaration и -Wimplicit-int.

-Wmain

Опция вызывает вывод предупреждения, если определение функции main() вызывает подозрения. Считается, что функция main() должна возвращать значение типа int и не иметь аргументов.

Примечание: для платформы AVR данное требование к функции main() бессмысленно.

-Wmissing-braces

Опция вызывает вывод предупреждения, если инициализация многомерного массива осуществляется без должного количества фигурных скобок, например, в подобном случае:

```
int a[2][2] = { 0, 1, 2, 3 };           // есть предупреждение
int b[2][2] = { { 0, 1 }, { 2, 3 } };   // нет предупреждения
```

-Wparentheses

Опция вызывает вывод предупреждения, если есть подозрение на отсутствие скобок или на неверное вложение операторов. Например, следующие операторы могут вызывать такое предупреждение:

```
x<=y<=z
if (a)
        if (b)
                foo ();
        else
                bar ();
```

Первый пример эквивалентен **(x<=y ? 1 : 0) <= z**, однако есть подозрение, что программист надеялся получить иной результат. Во втором случае может оказаться, что программист надеялся, что **else** относится к верхнему оператору **if(a)**, а не к **if(b)**, как действительно считает компилятор.

-Wsequence-point

Опция вызывает вывод предупреждения, если встречается конструкция, допустимая синтаксически, но есть подозрение о том, что последовательность вычислений нарушена. Как известно, синтаксис Си допускает весьма витиеватые конструкции, чрезмерное увлечение которыми может создать неоднозначности, например:

```
a = a++;
a[n] = b[n++];
a[i++] = i;
```

Примечание: всегда есть возможность реализовать тот же алгоритм без неоднозначных или запутанных конструкций. Наличие предупреждения – верный признак плохого стиля программирования. Ошибочно считать, что уровень программиста определяется его способностью писать код, с трудом разбираемый даже компилятором.

-Wall

Опция, активирующая все ранее рассмотренные опции генерации предупреждений (исключая те опции, которые отменяют определенные предупреждения).

По умолчанию всегда используется при компиляции из AVR Studio, поэтому программист может добавлять лишь негативные опции для отключения части предупреждений.

Оптимизация

Опции управления оптимизацией позволяют выбрать те или иные возможности компилятора по уменьшению размера генерируемого кода и(или) уменьшению времени его исполнения. Как правило, требуется получить максимально быстрый код минимального размера. Гибкость GCC в плане оптимизации очень высока: он поддерживает более сотни различных опций. Как правило, необходимости в столь тонкой и тщательной настройки оптимизатора нет, тем более что есть ряд опций, оптимально объединяющих другие опции для достижения определенных целей.

В эту группу включены так же опции, позволяющие осуществить дополнительную оптимизацию кода, но уже не за счет оптимизатора, а за счет осознанного желания программиста отказаться от каких-либо возможностей для этого.

-O или –O1

Первый уровень оптимизации. Компилятор пытается предпринять действия, которые уменьшают объем кода и время его исполнения, но не требуют при этом длительного компилирования.

-O2

Второй уровень оптимизации. Компилятор использует почти все средства оптимизации, кроме «развертывания» циклов и встраивания inline-функций. По сравнению с –O1 процесс компиляции длится существенно дольше, но генерируемый код обладает большей производительностью.

-O3

Третий уровень оптимизации. Компилятор применяет все средства оптимизации для получения максимально быстрого кода, в том числе использует развертывание циклов и встраивание в код inline-функций. Размер генерируемого кода при этом увеличивается.

-Os

Оптимизация по размеру кода. Используются все средства оптимизации кроме тех, что приводят к увеличению размера кода. В большинстве случаев использование этой опции генерирует оптимальный код, т.е. компромиссно быстрый и компактный одновременно. Нередко этот код оказывается и самым быстрым.

-O0

Оптимизация отключена. Компилятор не предпринимает никаких мер по уменьшению объема кода и по увеличению его производительности.

-mno-interrupts

Опция для генерации кода, не совместимого с системой обработки прерываний. Исключает из генерируемого кода запрещение прерываний при изменении указателя стека, уменьшая тем самым объем кода. В итоге в некоторых случаях возможно получение неработоспособного кода в программах с прерываниями.

-mcall-prologues

Опция, указывающая компилятору оформлять код пролога и эпилога функций в виде отдельных подпрограмм. Итоговый объем кода может при этом уменьшиться, но может возрасти время исполнения программы.

-mno-tablejump

Опция, запрещающая компилятору генерировать таблицы переходов. В некоторых случаях это позволяет уменьшить объем результирующего кода.

-mint8

Опция уменьшения в 2 раза размера всех целых чисел. Таким образом, тип **int** и **char** будут однобайтными, **long** – 16-битным, а **long long** – 32-битным. Эта опция противоречит стандартам Си, однако позволяет получить меньший по объему код с лучшим быстродействием.

Препроцессор Си

Группа опций, управляющих работой препроцессора.

-D

Опция, позволяющая определить «внешний», т.е. не определенный внутри текста программы, макрос. Имеется две разновидности формата этой опции:

-D <**имя**> - определяет <**имя**> как макрос со значением 1

-D <**имя**>=<**значение**> - определяет <**имя**> как макрос определенным значением.

Например:

-D debug

-D clock=1000

Эти строки эквивалентны тому, как если бы в первых строках компилируемого текста программы находились бы следующие директивы:

```
#define debug 1
#define clock 1000
```

Примечание: имя макроса отделяется от опции минимум одним пробелом.

-U

Опция, отменяющая определение любого макроса: встроенного в GCC или определенного директивой –D. Формат опции следующий:

-U <**имя**> - макрос <**имя**> перестает существовать для препроцессора.

Примечание: имя макроса отделяется от опции минимум одним пробелом.

Ассемблер

Группа опций, которые позволяют указать параметры ассемблера, который вызывается GCC автоматически.

-Wa

Опция имеет следующий формат:

-Wa,<**параметр**>

Назначение опции – передать ассемблеру указанный после запятой **параметр**. Если этот параметр содержит запятые, происходит разбиение его на несколько параметров. То есть при помощи этой опции можно передать сразу несколько параметров, разделив их запятыми.

-Xassembler

Опция имеет следующий формат:

-Xassembler <**параметр**>

Назначение опции аналогично –Wa: передать параметр компилятору ассемблера. В отличие от –Wa эта опция не может передать несколько параметров, для каждой части составного параметра следует использовать отдельные опции –Xassembler.

Примечание: параметр отделяется от опции минимум одним пробелом.

Эта опция не анализируется GCC, что позволяет передавать ассемблеру нестандартные команды.

Компоновщик

Группа опций для управления процессом компоновки.

-Wl

Опция, позволяющая передать компоновщику список параметров точно так же, как –Wa.

-Wl,-gc-sections

Вариант использования опции –Wl для передачи компоновщику параметра **-gc-sections**. Этот параметр означает, что компоновщик должен удалить все секции памяти, к которым нет обращения из других секций. Это позволяет в некоторых случаях уменьшить объем результирующего кода.

Платформа

Группа опций, позволяющих настроить компилятор на поддержку определенной платформы. В настоящей книге рассматриваются только микроконтроллеры AVR в качестве платформы.

-mcu

Опция указания системы команд микроконтроллера. По умолчанию автоматически добавляется средой AVRStudio. Формат опции следующий:

-mcu=<**тип**>, где **тип** – это обозначение системы команд. Определено 5 вариантов систем команд:

avr1 – система команд микроконтроллеров «минимальной» конфигурации, не имеющих ОЗУ и потому поддерживаемых только ассемблером. К их числу относятся, например, AT90S1200, attiny12 и другие.

avr2 – система команд «классических» AVR с объемом памяти программ до 8К. Этот набор используется по умолчанию, т.е. если опция –mcu отсутствует. В эту группу микроконтроллеров входят, например, at90s2313, at90s2323, attiny22 и др.

avr3 –система команд классического ядра AVR с объемом памяти программ до 128K.

avr4 – система команд «улучшенного» ядра AVR с объемом памяти программ до 8K. К этой группе относятся, например, контроллеры семейства mega: atmega8 и т.п.

avr5 – система команд «улучшенного» ядра AVR с объемом памяти программ до 128K, например, atmega16, atmega128 и т.п.

-mtiny-stack

Опция указывает компилятору изменять только младший байт указателя стека, ограничивая тем самым его глубину.

-minit-stack

Опция позволяет назначить начальный адрес указателя стека. Формат опции следующий:

-minit-stack=N, где **N** – значение указателя стека. N может быть числом или символом, по умолчанию используется символ **__stack**.

Глобальные параметры компилятора

Группа опций, оказывающих влияние на различные параметры компилятора, которые сложно отнести к ранее рассмотренным группам.

-gdwarf-2

Для реализации отладки в AVR Studio необходимо использование опции генерации отладочной информации -gdwarf-2. GCC поддерживает кроме этой еще более 6 десятков опций управления отладочной информацией, однако их актуальность для платформы AVR и AVR Studio сомнительна, поэтому они не рассматриваются.

-fshort-enums

Оправляет способом организации перечисляемых типов. При использовании этой опции под переменную перечисляемого типа (enum) будет выделено минимально необходимое для хранения типа целое число. В некоторых случаях это позволяет уменьшить объем кода и требования к ОЗУ.

Примечание: код, генерируемый в случае использования этой опции, не совместим с кодом, генерируемым по умолчанию. Это может вызвать проблемы при компоновке модулей, оттранслированных с этой опцией и без нее.

-fshort-double

Опция указывает, что тип double будет равнозначен типу float. Это может уменьшить объем кода и увеличить быстродействие программ, использующих сложные математические вычисления, однако в ущерб точности расчетов.

-ffunction-sections

Опция указывает компилятору выделять код каждой функции в отдельную ***секцию***. Имя секции совпадает с именем функции. Самостоятельное использование этой опции не дает никакого эффекта, однако в совокупности с другими может дать существенный выигрыш в объеме кода.

-fdata-sections

Опция указывает компилятору выделять каждую переменную в отдельную ***секцию***). Имя секции совпадает с именем переменной. Самостоятельное использование этой опции не дает никакого эффекта, однако в совокупности с другими может дать существенный выигрыш в объеме требуемой памяти (ОЗУ).

Продолжение в №10/2010