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

Книга по работе с WinAVR и AVR Studio

Продолжение. Начало в №1-9/2010

AVR-LIBC

ВВЕДЕНИЕ

Комплект WinAVR содержит много различных средств для разработки программного обеспечения микроконтроллерных систем на основе микроконтроллеров AVR. Однако, сам по себе компилятор, каким бы хорошим и универсальным он ни был, не содержит главного: набора библиотечных функций. Существует некий набор библиотечных функций, ставший стандартным, т.е. набор, к которому привыкли все без исключения программисты Си, его так и называют стандартной библиотекой.

Очевидно, для полноценной разработки ПО для микроконтроллеров AVR так же необходима такая стандартная библиотека. И она включена в состав WinAVR: это библиотека AVR-LIBC. Она представляет собой набор объектных модулей и соответствующих заголовочных файлов, обеспечивает большинство традиционно-стандартных функций, а так же ряд платформо-ориентированных расширений, т.е. функций, специфичных только для микроконтроллеров AVR. Стандартные функции порой имеют отклонения от привычного поведения в «компьютерном Си», что связано с ограничениями, накладываемыми архитектурой микроконтроллера. Программист должен быть осведомлен о таких отклонениях.

Далее будут рассмотрены основные составляющие части библиотеки AVR-LIBC с необходимыми комментариями. Они разделены на три группы: Стандартные модули, AVR-специфичные модули и Вспомогательные модули.

СТАНДАРТНЫЕ МОДУЛИ

Стандартные модули – это модули, реализующие набор привычных функций, характерных для всех Си-программ вне зависимости от платформы. В этот раздел включен и ряд модулей, определяющих некоторые важные для GCC-компилятора константы, а так же модули, необходимые для всех прочих библиотечных модулей AVR-LIBC.

Далее будет принят следующий формат изложения материала:

- **жирным шрифтом** выделяется наименование раздела, как правило, совпадающего с соответствующим модулем библиотеки;
- **подчеркнутым шрифтом** выделяется функция или макрос, определяемый в рассматриваемом модуле;
- после слова **Определение** приводится прототип функции, как он определен в соответствущем хидере.

Далее следует подробное описание функции или макроса, иногда с пояснениями.

alloca.h – Выделение памяти в стеке

Определение:
void * alloca(size_t size);

Описание: функция выделяет **size** байт в области стека вызывающего модуля. Это временно выделяемое пространство автоматически освобождается, когда функция, вызывающая **alloca()**, возвращает управление в вызывающую ее функцию. **alloca()** реализована в виде макроса, который транслируется в inline-функцию **__builtin_alloca()**. Это означает, что нет возможности обращаться к функции по ее адресу или изменять ее поведение при связывании с другими библиотеками.

Возвращаемое значение: функция возвращает указатель на начало выделенной области. Если происходит переполнение стека, поведение программы не определено.

Предупреждение: избегайте использования функции внутри списка параметров функций.

assert.h – Диагностика

Определение:
#include <assert.h>

Описание: этот заголовочный файл определяет вспомогательные отладочные возможности.

Так как в большинстве приложений нет стандартно заданного потока вывода для ошибок, генерирование печатаемых сообщений об ошибках по умолчанию отключено. Эти сообщения будут генерироваться только в том случае, если в приложении определен макрос **__ASSERT_USE_STDERR** прежде подключения файла <**assert.h**>. По умолчанию для остановки приложения должна использоваться только функция **abort()**.

Применение:
#define assert(<выражение>)

Макрос **assert()** принимает в качестве параметра выражение, проверяет его на истинность и, если выражение ложно, вызывает функцию **abort()** для завершения приложения. При этом в стандартный поток вывода ошибок выводится соответствующее сообщение. Если выражение истинно, этот макрос не выполняет никаких действий.

Все макросы **assert()** могут быть удалены из текста программы при компиляции с параметром –**DNDEBUG**.

ctype.h – Символьные операции

Тут определены различные функции над символами.

Определение:
#include <ctype.h>

Описание: этот заголовочный файл подключает к проекту некоторые функции (см. далее детальное описание каждой) обработки символов. Функции разделены на две группы: классифицирующие символ и преобразующие символ.

Классифицирующие функции следующие:

isalnum()
isalpha()
isascii()
isblank()
iscntrl()
isdigit()
isgraph()
islower()
isprint()
ispunct()
isspace()
isupper()
isxdigit()

Эти функции получают в качестве параметра символ, и возвращают истинное или ложное значение в зависимости от того, как этот символ проклассифицирован. Если параметр не является **unsigned char** – все функции возвращают **FALSE**.

Преобразующие функции следующие:

toascii()
tolower()
toupper()

Эти функции так же получают в качестве параметра символ, а возвращают уже преобразованный символ.

Функции модуля

Обратите внимание на то, что параметр этих функций имеет тип **int**, хотя используется как *символ*, т.е. *char*.

isalnum()
Определение:
int isalnum (int c)

Описание: Проверяет, является ли аргумент алфавитно-цифровым символом. Результат эквивалентен **(isalpha(c) || isdigit(c))**

isalpha()
Определение:
int isalpha (int c)

Описание: Проверяет, является ли символ алфавитным. Результат эквивалентен **(isupper(c) || islower(c))**

isascii()
Определение:
int isascii (int c)

Описание: Проверяет, является ли параметр 7-битным ASCII-символом.

isblank()
Определение:
int isblank (int c)

Описание: Проверяет, является ли символ «пустым», т.е. является ли он пробелом или символом табуляции.

iscntrl()
Определение:
int iscntrl (int c)

Описание: Проверяет, является ли символ управляющим.

isdigit()
Определение:
int isdigit (int c)
Описание: Проверяет, является ли символ цифровым.

isgraph()
Определение:
int isgraph (int c)
Описание: Проверяет, является ли символ «псевдографическим».

islower()
Определение:
int islower (int c)
Описание: Проверяет, является ли «регистр» символа «нижним».

isprint()
Определение:
int isprint (int c)
Описание: Проверяет, является ли символ «печатаемым», т.е. любым, который можно отобразить на печатающем устройстве (включая пробел).

ispunct()
Определение:
int ispunct (int c)
Описание: Проверяет, является ли символ печатаемым, но при этом не принадлежит алфавитно-цифровым или «пустым».

isspace()
Определение:
int isspace (int c)
Описание: Проверяет, является ли символ символом-разделителем. В этой реализации AVR-LIBC в число этих символов входят:
- пробел
- пролистывание формы (form-feed, “\f”)
- перевод строки (“\n”)
- возврат каретки (“\r”)
- горизонтальная табуляция (“\t”)
- вертикальная табуляция (“\v”)

isupper()
Определение:
int isupper (int c)
Описание: Проверяет, является ли символ буквой в верхнем регистре.

isxdigit()
Определение:
int isxdigit (int c)
Описание: Проверяет, является ли символ допустимой шестнадцатеричной цифрой.

toascii()
Определение:
int toascii (int c)
Описание: Преобразует символ в 7-битный ASCII-символ путем очистки старших битов.

Предупреждение: пользователи могут быть недовольны, если вы применяете эту функцию, т.к. многие символы (например, национальных алфавитов) эта функция преобразует в непредсказуемые, делая текст нечитаемым.

tolower()
Определение:
int tolower (int c)
Описание: Преобразует, если возможно, регистр символа к нижнему.

toupper()
Определение:
int toupper (int c)
Описание: Преобразует, если возможно, регистр символа к верхнему.

errno.h – Системные ошибки

Определение:
#include <errno.h>
Описание: многие библиотечные функции при возникновении ошибок устанавливают значение встроенной глобальной переменной **errno**. Подключение заголовочного файла **errno.h** вводит этим значениям символьные эквиваленты.

Предупреждение: использование глобальной переменной **errno** в мультизадачных или многопоточных приложениях не является удачным. Проблема в том, что ее значение может быть изменено другим потоком в промежутке между ее установкой и началом анализа в исходном потоке.

Примечание. Символьные константы, описанные в этом файле, не имеют практической ценности для начинающего разработчика, поэтому не приводятся – вы можете самостоятельно изучить их по содержимому указанного файла.

inttypes.h – Целочисленные преобразования

Определение:
#include <inttypes.h>
Описание: этот заголовочный файл использует определения целых чисел **stdint.h** и расширяет их дополнительными вариантами, используемыми в работе компилятора.

В настоящее время расширения включают 2 дополнительных целочисленных типа для «дальних указателей» (т.е. для указателей в коде, способных адресовать память более чем 64K), а так же все типы, поддерживаемые функциями форматированного вывода из **stdio.h**. Так как эти функции не реализуют полного комплекта спецификаций, определяемых ISO 9899:1999, включены только реализованные возможности.

Идея подхода в том, что каждый тип, определенный в **stdint.h**, сопровождается макросами, позволяющими определить соответствующий объект форматирования для функций printf() или scanf(). Пример:

```
#include <inttypes.h>
uint8_t smallval;
int32_t longval;
...
printf(«The hexadecimal value of smallval is
«PRIx8», the decimal value of longval is
«PRId32 «.\n», smallval, longval);
```

Дальние указатели для адресации более, чем 64K памяти

typedef int32_t int_farptr_t – целое со знаком для хранения дальнего указателя.

typedef uint32_t uint_farptr_t – целое беззнаковое для хранения дальнего указателя.

Макросы для спецификатора форматов printf() и scanf()

Для C++ эти макросы подключаются только в том случае, если определен символ **__STDC_LIMIT_MACROS** до подключения файла <inttypes.h>.

Модуль содержит большое количество макросов для определения форматов ввода-вывода различных чисел. Т.к. эти макросы применяются в основном встроенными в библиотеку функциями, практическая их ценность для конечного пользователя невелика. Поэтому из рассмотрения в данном издании они опущены.

math.h – Математика

Определение:
#include <math.h>
Описание: этот заголовочный файл определяет некоторые базовые константы и математические операции.

Функции, определенные этим файлом, требуют связывания с **libm.a**. Так же следует учитывать, что математические функции не генерируют исключений и не изменяют переменную **errno**. Для улучшения качества оптимизации **GCC** они реализованы в большинстве своем с атрибутом **const**.

Определения модуля

#define M_PI 3.14159265358979323846264 3 – число p
#define M_SQRT2 1.41421356237309504880168872 – квадратный корень из двух
#define NAN __builtin_nan(«») – константа определения «не числа»
#define INFINITY __builtin_inf() – константа определения бесконечно большого значения

Функции модуля

cos()
Определение:
double cos (double x)
Описание: возвращает косинус аргумента **x** (должен быть в радианах).

fabs()
Определение:
double fabs (double x)
Описание: возвращает модуль значения аргумента **x**.

fmod()
Определение:
double fmod (double x, double y)
Описание: возвращает остаток от деления **x/y**.

modf()
Определение:
double modf (double x, double *iptr)

Описание: функция разбивает аргумент **x** на целую и дробную части, каждая из которых имеет тот же знак, что и **x**. Функция возвращает дробную часть (со знаком), а целую часть (со знаком) помещает по указателю **iptr**.

Примечание: если указатель **iptr** нулевой – возврат целой части пропускается.

sin()
Определение:
double sin (double x)
Описание: возвращает синус аргумента **x** (должен быть в радианах).

sqrt()
Определение:
double sqrt (double x)
Описание: возвращает не отрицательный квадратный корень аргумента **x**.

tan()
Определение:
double tan (double x)
Описание: возвращает тангенс аргумента **x** (должен быть в радианах).

floor()
Определение:
double floor (double x)
Описание: возвращает в виде числа с плавающей точкой ближайшее целое число, меньшее или равное аргументу **x**.

ceil()
Определение:
double ceil (double x)
Описание: возвращает в виде числа с плавающей точкой ближайшее целое число, большее или равное аргументу **x**.

frexp()
Определение:
double frexp (double x, int *pexp)
Описание: функция разбивает число **x** на два множителя: первый множитель – это число 0 или число в диапазоне (по модулю) от 0,5 до 1, – функция возвращает в качестве результата; второй множитель образуется как число 2 в степени pexp. Таким образом, если результат функции обозначить **y**, то $\mathbf{x} = \mathbf{y} * \mathbf{2}^{\mathbf{pexp}}$.

Если **x** равно нулю – обе возвращаемые части равны нулю. Если x не является конечным числом – функция возвращает x, а pexp равно нулю.

Если второй параметр функции NULL, то функция воспринимает это как указание не вычислять и не возвращать значение pexp.

ldexp()
Определение:
double ldexp (double x, int exp)
Описание: возвращает число **x**, умноженное на 2 в степени **exp**.

exp()
Определение:
double exp (double x)
Описание: возвращает e в степени **x** (экспоненциальная функция).

cosh()
Определение:
double cosh (double x)
Описание: возвращает гиперболический косинус аргумента **x**.

sinh()
Определение:
double sinh (double x)
Описание: возвращает гиперболический синус аргумента **x**.

tanh()
Определение:
double tanh (double x)
Описание: возвращает гиперболический тангенс аргумента **x**.

acos()
Определение:
double acos (double x)
Описание: возвращает арккосинус аргумента **x**. Значение аргумента должно быть в пределах допустимых значений -1…+1. Возвращается значение в пределах -p/2…+p/2 радиан.

asin()
Определение:
double asin (double x)
Описание: возвращает арксинус аргумента **x**. Значение аргумента должно быть в пределах допустимых значений -1…+1. Возвращается значение в пределах -p/2…+p/2 радиан.

atan()
Определение:
double atan (double x)
Описание: возвращает арктангенс аргумента **x**. Возвращается значение в пределах -p/2…+p/2 радиан.

atan2()
Определение:
double atan2 (double y, double x)
Описание: возвращает арктангенс частного **x**/**y**. Возвращается значение в пределах -p…+p радиан.

log()
Определение:
double log (double x)
Описание: возвращает натуральный логарифм аргумента **x**.

log10()
Определение:
double log10 (double x)
Описание: возвращает десятичный логарифм аргумента **x**.

pow()
Определение:
double pow (double x, double y)
Описание: возвращает **x**, возведенное в степень **y**.

isnan()
Определение:
int isnan (double x)
Описание: возвращает 1, если аргумент **x** бесконечно малое значение, и 0 – в противном случае.

isinf()
Определение:
int isinf (double x)
Описание: возвращает -1, если аргумент **x** – «минус-бесконечность», 1, если аргумент «плюс-бесконечность» и 0 – в противном случае.

square()
Определение:
double square (double x)
Описание: возвращает **x** в квадрате.
Примечание: это нестандартная для C функция.

copysign()
Определение:
static double copysign (double x, double y)
Описание: возвращает **x**, но со знаком от **y**. Эта функция работает даже если любой из аргументов (или оба) – бесконечность или пренебрежимо малые числа.

fdim()
Определение:
double fdim (double x, double y)
Описание: возвращает наибольшее значение из вариантов (**x**-**y**) или 0. Если хотя бы одно из чисел бесконечно мало – возвращается бесконечно малое число.

fma()
Определение:
double fma (double x, double y, double z)
Описание: функция «умножение с накоплением». Возвращает (**x***y)+**z**, однако, в процессе вычислений промежуточный результат не округляется, что позволяет порой улучшить точность вычислений.

fmax()
Определение:
double fmax (double x, double y)
Описание: возвращает наибольший из аргументов. Если оба аргумента NAN - возвращается NAN, если один аргумент NAN, возвращается другой.

fmin()
Определение:
double fmin (double x, double y)
Описание: возвращает наименьший из аргументов. Если оба аргумента NAN - возвращается NAN, если один аргумент NAN, возвращается другой.

signbit()
Определение:
int signbit (double x)
Описание: возвращает 1, если знаковый бит у аргумента **x** установлен.
Примечание: это не то же самое, что результат (x < 0.0), так как спецификация IEEE 754 допускает наличие знака у нуля! Значение (-0.0 < 0.0) ложно, но signbit(-0.0) возвратит ненулевое значение!

trunc()
Определение:
double trunc (double x)
Описание: возвращает округленное до ближайшего целого, не большего по модулю аргумента **x**, число.

isfinite()
Определение:
static int isfinite (double x)
Описание: возвращает ненулевое значение, если аргумент **x** – конечное число: не бесконечность (с любым знаком) и не NAN.

hypot()
Определение:
double hypot (double x, double y)
Описание: возвращает квадратный корень из суммы квадратов аргументов. Применение этой функции вместо ее эквивалента sqrt(**x*****x** + **y*****y**) более предпочтительно, так как обеспечивает лучшую точность. Исключается ошибка промежуточного округления, ошибки переполнения (при больших значениях аргументов) и «исчезновения» (при малых значениях) результата.

round()
Определение:
double round (double x)
Описание: возвращает округленное до ближайшего целого значение аргумента **x**, но при этом не учитывает правило «половины единицы». Переполнение не возникает.
Примечание: если аргумент – целое число или бесконечность – оно и возвращается. Если аргумент NAN – возвращается NAN.

lround()
Определение:
long lround (double x)
Описание: возвращает целое число. В остальном аналогична round(), кроме того, что ошибка переполнения возможна.

lrint()
Определение:
long lrint (double x)
Описание: возвращает округленное до целого значение аргумента **x** с учетом «правила половины», т.е. 1,5 и 2,5 будут округлены до 2.
Примечание: возвращаемое значение может быть LONG_MIN (т.е. 0x80000000), если аргумент – не конечное число или было переполнение.

setjmp.h – Нелокальные переходы goto

Подробное описание

Когда в тексте программы применяется оператор **goto**, он позволяет осуществить переход только внутри функции, где он применен (т.е. локально). Чтобы осуществить переход в произвольное место программы, определены функции setjmp() и longjmp(). Их использование позволяет, например, реализовать более эффективно обработку ошибок и исключительных ситуаций при низкоуровневом программировании.

Врезка. Пример использования функций.

```
#include <setjmp.h>
jmp_buf env; // env – переменная-метка

int main (void){
       if (setjmp (env)){
             ... здесь должна обработаться ошибка ...
}

while (1){
             ... основной цикл, в котором вызывается foo() ...
}
}
...
void foo (void){
      ... что-то делается ...
if (err){
       // а если происходит ошибка – происходит переход в главную функцию longjmp (env, 1);
}
}
```

Использование этих подходов может сделать программу сложной для понимания, а так же может привести к потере контекста глобальных регистровых переменных. Поэтому по возможности следует искать альтернативные пути.

Пример использования функций ***см. на врезке***:

Функции модуля

setjmp()
Определение:
int setjmp (jmp_buf jmpb)
Параметры:
jmp_buf **jmpb** – переменная сохранения контекста.
Описание: функция сохраняет контекст стека и «окружения» в переменной **jmpb** для последующего использования в функции longjmp(). Сохраненный контекст теряется, если происходит возврат из функции, использовавшей setjmp().
Функция возвращает 0, если возврат произошел немедленно, и не 0, если возврат произошел при вызове longjmp() с использованием сохраненного контекста.

longjmp()
Определение:
void longjmp (jmp_buf jmpb, int ret) __ATTR_NORETURN__
Параметры:
jmp_buf **jmpb** – переменная сохранения контекста.
int **ret** – значение, используемое для возврата.
Описание: функция осуществляет нелокальный переход к сохраненному ранее контексту. Возврата из функции в традиционном понимании нет, нет и возвращаемого значения.
Функция восстанавливает окружение (т.е. состояние среды) в том виде, как сохранено ранее в переменной **jmpb**, после чего осуществляет передачу управления так, как будто завершилась функция setjmp() в том месте, откуда была вызвана, но с возвратом значения **ret**.

Примечание от автора

Не смотря на некоторую гибкость возможностей, предлагаемых функциями модуля **setjmp.h**, настоятельно не рекомендуется применять их, особенно в случае, когда общая квалификация программиста невысока (т.е. начинающими).

stdint.h – Стандартные целые числа

Модуль вводит описания различных типов целых чисел, определяемых стандартом C99. Просто используйте для определения переменных разрядностью N бит (где N может быть 8, 16, 32 или 64) следующую форму записи типа: **[u]intN_t**. Например, **uint8_t** – эквивалент **unsigned char**; а **int16_t** – эквивалент **signed int**.
Кроме того, вводится большое количество макросов и констант, которые можно использовать в своих программах.

Типы целых чисел

Следующие типы вводят (определяют) обозначения-синонимы типам целых чисел по размеру в битах. Любой из этих типов может быть представлен стандартным эквивалентом, однако вводимые типы имеют более короткую запись, что, несомненно, более удобно:
int8_t – байт со знаком (8 бит)
uint8_t – байт без знака (8 бит)
int16_t – целое число со знаком (16 бит)
uint16_t – целое число без знака (16 бит)
int32_t – длинное целое число со знаком (32 бита)
uint32_t – длинное целое число без знака (32 бита)
int64_t – большое длинное число со знаком (64 бита)
uint64_t – большое длинное число без знака (64 бита)

Вводятся 2 типа для указателей:
intptr_t – то же самое, что и int16_t, используется для хранения указателя
uintptr_t – то же самое. Что и uint16_t, используется для хранения указателя

Ряд типов для чисел, содержащих числа с числом битов не менее определенного:

int_least8_t – для чисел со знаком не более чем из 8 битов

uint_least8_t – для чисел без знака не более чем из 8 битов

int_least16_t – для чисел со знаком не более чем из 16 бит

uint_least16_t – для чисел без знака не более чем из 16 бит

int_least32_t – для чисел со знаком не более чем из 32 битов

uint_least32_t – для чисел без знака не более чем из 32 битов

int_least64_t – для чисел со знаком не более чем из 64 битов

uint_least64_t – для чисел без знака не более чем из 64 битов

Определения типов для чисел заданной размерности. Обрабатывающихся с максимальной скоростью[31] (расшифровка типов не приводится ввиду явной очевидности):

int_fast8_t
uint_fast8_t
int_fast16_t
uint_fast16_t
uint_fast32_t
int_fast64_t
uint_fast32_t

Определения типов целых чисел, способных хранить максимально возможные числа среди поддерживаемых знаковых или беззнаковых:

intmax_t – наибольшее целое со знаком (аналог int64_t)

uintmax_t – наибольшее целое без знака (аналог uint64_t)

Константы

В модуле определен ряд констант (макросов), определяющих предельные значения для чисел каждого типа. Данные константы для программ на C++ подключаются только в том случае, если определено __STDC_LIMIT_MACROS до подключения модуля stdint.h.

INT8_MAX – максимальное число типа int8_t

INT8_MIN – минимальное число типа int8_t

UINT8_MAX – максимальное число типа uint8_t

INT16_MAX – максимальное число типа int16_t

INT16_MIN – минимальное число типа int16_t

UINT16_MAX – максимальное число типа uint16_t

INT32_MAX – максимальное число типа int32_t

INT32_MIN – минимальное число типа int32_t

UINT32_MAX – максимальное число типа uint32_t

INT64_MAX – максимальное число типа int64_t

INT64_MIN – минимальное число типа int64_t

UINT64_MAX – максимальное число типа uint64_t

Аналогичные константы предельных значений определены и для всех других типов чисел (приводятся без описания):

INT_LEAST8_MAX
INT_LEAST8_MIN
UINT_LEAST8_MAX
INT_LEAST16_MAX
INT_LEAST16_MIN
UINT_LEAST16_MAX
INT_LEAST32_MAX
INT_LEAST32_MIN
UINT_LEAST32_MAX
INT_LEAST64_MAX
INT_LEAST64_MIN
UINT_LEAST64_MAX
INT_FAST8_MAX
INT_FAST8_MIN
UINT_FAST8_MAX
INT_FAST16_MAX
INT_FAST16_MIN
UINT_FAST16_MAX
INT_FAST32_MAX
INT_FAST32_MIN
UINT_FAST32_MAX
INT_FAST64_MAX
INT_FAST64_MIN
UINT_FAST64_MAX
INTPTR_MAX
INTPTR_MIN
UINTPTR_MAX
INTMAX_MAX
INTMAX_MIN
UINTMAX_MAX

Имеется так же ряд констант максимальных значений для специальных типов (эти определения для C++ так же требуют наличия определения __STDC_LIMIT_MACROS до подключения модуля stdint.h):

PTRDIFF_MAX
PTRDIFF_MIN
SIG_ATOMIC_MAX
SIG_ATOMIC_MIN
SIZE_MAX

Макросы задания числовых констант

Модуль определяет ряд макросов-функций, позволяющих определять числовые константы заданного размера без использования «суффиксов» размерности:

INT8_C(value) – константа 8-битная со знаком

UINT8_C(value) – константа 8-битная без знака

INT16_C(value) – константа 16-битная со знаком

UINT16_C(value) – константа 16-битная без знака

INT32_C(value) – константа 32-битная со знаком

UINT32_C(value) – константа 32-битная без знака

INT64_C(value) – константа 64-битная со знаком

UINT64_C(value) – константа 64-битная без знака

INTMAX_C(value) – константа наибольшего размера со знаком

UINTMAX_C(value) – константа наибольшего размера без знака

Использование этих макросов вместо простых чисел позволит избежать ошибок, связанных с определением размерности результата вычислений с константами.

[31] Это не более чем условность – на самом деле вводимые «скоростные» типы, как и все предыдущие, есть всего лишь синонимы ранее определенным, т.е. обрабатываются компилятором абсолютно с той же скоростью. Очевидно, эти типы введены для совместимости с GCC для других платформ.

Продолжение в №11/2010