

Из документа «Начинаем работу с μ Vision2»

Глава 6. Отладочные функции среды μ Vision2

В этой главе рассматривается мощный аспект среды μ Vision2: использование отладочных функций. Вы можете использовать эти функции, чтобы расширить возможности отладчика среды μ Vision2. Вы можете создать функции, которые генерируют внешние прерывания, сохраняют дампы памяти в файл, периодически обновляют сигналы на аналоговых входах и последовательно подают данные на вход последовательного порта процессора.

Примечание

*Не путайте отладочные функции среды μ Vision2 с функциями в своей программе. Отладочные функции среды μ Vision2 помогают вам при отладке вашей программы и вводятся при помощи **Редактора Функций** или в командной строке отладчика μ Vision2.*

Отладочные функции среды μ Vision2 используют подмножество языка программирования Си. Это подмножество обладает следующими возможностями и ограничениями:

- В отладочных функциях могут использоваться операторы управления ходом выполнения программы: **if**, **else**, **while**, **do**, **switch**, **case**, **break**, **continue** и **goto**. В отладочных функциях среды μ Vision2 все эти операторы работают так, как это определено в стандарте ANSI C.
- В отладочных функциях можно объявлять локальные скалярные переменные, аналогично тому, как это определено в стандарте ANSI C. Массивы не поддерживаются.

Полное описание отличий вы найдете в параграфе «Различия между отладочными функциями и Си-функциями» на странице 147.

Создание функций

Среда μ Vision2 содержит встроенный редактор отладочных функций, который открывается из меню **Debug – Function Editor**. Когда вы запускаете редактор функций, редактор запрашивает имя файла или открывает файл, указанный в опциях проекта, которые можно настроить через меню **Options for Target – Debug**, в поле **Initialization File**. Редактор отладочных функций работает аналогично редактору текста программы среды μ Vision2 и дает вам возможность ввести текст отладочной функции и откомпилировать его.

```

/*-----*/
/* MyStatus shows analog and other values ... */
/*-----*/

FUNC void MyStatus (void) {
    printf ("=====\n");
    printf (" Analog-Input-0: %f\n", ain0);
    printf (" Analog-Input-1: %f\n", ain1);
    printf (" Analog-Input-2: %f\n", ain2);
    printf (" Analog-Input-3: %f\n", ain3);
    printf (" Registers (CP): %04X\n", CP);
    printf (" Program Counter: %06lXH\n", $);
    printf ("=====\n");
}

```

Управляющие элементы	Описание
Кнопка Open	Открывает существующий файл, содержащий отладочные функции или команды среды μ Vision2
Кнопка New	Создает новый файл
Кнопка Save	Сохраняет текст в окне редактора в файл
Кнопка Save as	Позволяет указать имя файла для сохранения отладочных функций
Кнопка Compile	Посылает текущее содержимое окна редактора в командный интерпретатор среды μ Vision2. При этом компилируются все отладочные функции
Поле Compile Error	Показывает список всех ошибок, обнаруженных в процессе компиляции. Выберите ошибку из списка – курсор в окне редактора будет установлен на строку с ошибкой

После создания файла с отладочными функциями среды μ Vision2 вы можете использовать команду **INCLUDE** для чтения текстового файла с диска и запуска в интерпретаторе. Например, если вы введете следующую команду в командном окне (Command) содержимое файла **MYFUNCS.INI** будет загружено и интерпретировано средой μ Vision2.

```
>INCLUDE MYFUNCS.INI
```

Файл **MYFUNCS.INI** может содержать отладочные команды и определения отладочных функций. Вы также можете указать этот файл в настройках проекта **Options for Target – Debug - Initialization File**. Тогда каждый раз при запуске отладчика среды μ Vision2 будет обрабатываться содержимое файла **MYFUNCS.INI**.

Функции, которые более не нужны в текущем сеансе отладки, могут быть удалены с использованием команды **KILL**.

Вызов функций

Для вызова или запуска отладочных функций вы должны набрать в командном окне имя функции и необходимые параметры функции. Например, для выполнения встроенной функции **printf** для вывода строки «Hello World» введите следующий текст в командном окне:

```
>printf ("Hello World\n")
```

Отладчик μ Vision2 в ответ напечатает текст «Hello World» в поле вывода страницы Command, расположенной в окне Output Window отладчика.

Классификация функций

Среда μ Vision2 поддерживает следующие 3 класса функций: предопределенные функции, пользовательские функции и сигнальные функции.

- **Предопределенные функции** выполняют полезные задачи, такие как ожидание некоего периода времени или вывода сообщений. Предопределенные функции не могут быть удалены или переопределены.
- **Функции пользователя** расширяют возможности отладчика μ Vision2 и могут обрабатывать те же выражения, которые допускаются на командном уровне. Вы можете использовать предопределенную функцию **exec** для выполнения отладочных команд внутри пользовательских или сигнальных функций.
- **Сигнальные функции** моделируют поведение сложных генераторов сигналов и позволяют вам создавать различные входные сигналы для вашей программы. Например, некие сигналы могут быть поданы на входные линии процессора во время отладки в симуляторе. Сигнальные функции выполняются в фоновом режиме во время выполнения отлаживаемой программы. Выполнение сигнальных функций синхронизировано со счетчиком тактов процессора, который имеет разрешение эквивалентное длительности одной инструкции. Одновременно может быть запущено не более 64 сигнальных функций.

Как только функции будут определены, они заносятся во внутреннюю таблицу пользовательских или сигнальных функций. Вы можете использовать команду **DIR** для вывода списка доступных предопределенных, пользовательских и сигнальных функций.

Команда **DIR BFUNC** показывает имена всех встроенных (предопределенных) функций. Команда **DIR UFUNC** показывает имена всех пользовательских функций. Команда **DIR SIGNAL** показывает имена всех сигнальных функций. Команда **DIR FUNC** показывает имена всех функций – пользовательских, сигнальных и встроенных.

Встроенные (предопределенные) функции

μ Vision2 содержит набор предопределенных функций, которыми вы всегда можете воспользоваться. Эти функции нельзя переопределить или удалить. Предопределенные функции предоставлены для поддержки функциональности в ваших пользовательских и сигнальных функциях. В следующей таблице приведен список всех предопределенных отладочных функций среды μ Vision2.

Тип возвращаемого значения	Имя функции	Параметры	Описание
void	exec	("command_string")	Выполнить команду отладчика
double	getdbl	("prompt_string")	Запрашивает у пользователя дробное число типа double
int	getint	("prompt_string")	Запрашивает у пользователя целое число типа int
long	getlong	("prompt_string")	Запрашивает у пользователя целое число типа long
void	memset	(start_addr, len, value)	Заполняет область памяти константой
void	printf	("string", ...)	Работает аналогично функции printf из ANSI C
int	rand	(int seed)	Возвращает случайное число из интервала от -32768 до +32767
void	rwatch	(address)	Приостанавливает выполнение сигнальной функции до тех пор, пока не будет произведено чтение памяти по указанному адресу
void	wwatch	(address)	Приостанавливает выполнение сигнальной функции до тех пор, пока не будет произведена запись в память по указанному адресу
void	swatch	(float seconds)	Приостанавливает выполнение сигнальной функции на указанное количество секунд
void	twatch	(ulong states)	Приостанавливает выполнение сигнальной функции на указанное количество тактов процессора
int	_TaskRunning_	(ulong func_address)	Производит проверку того, что указанная задача является текущей выполняемой задачей. Функция доступна если используется специальная DLL для отладки ядра RTX.
uchar	_RBYTE	(address)	Читает значение типа char из памяти по указанному адресу
uint	_RWORD	(address)	Читает значение типа int из памяти по указанному адресу
ulong	_RDWORD	(address)	Читает значение типа long из памяти по указанному адресу
float	_RFLOAT	(address)	Читает значение типа float из памяти по указанному адресу
double	_RDOUBLE	(address)	Читает значение типа double из памяти по указанному адресу
void	_WBYTE	(address, uchar val)	Записывает значение типа char в память по указанному адресу
void	_WORD	(address, uint val)	Записывает значение типа int в память по указанному адресу
void	_WORD	(address, ulong val)	Записывает значение типа long в память по указанному адресу
void	_WFLOAT	(address, float val)	Записывает значение типа float в память по указанному адресу
void	_WDOUBLE	(address, double val)	Записывает значение типа double в память по указанному адресу

Ниже дано описание предопределенных функций.

void exec (“command_string”)

Функция **exec** позволяет вам вызвать команды отладчика μ Vision2 из ваших пользовательских или сигнальных функций. Командная строка (*command_string*) может содержать несколько команд, разделенных точкой с запятой.

Командная строка передается командному интерпретатору и потому должна быть корректной командой отладчика.

Example

```
>exec ("DIR PUBLIC; EVAL R7")
>exec ("BS timer0")
>exec ("BK *")
```

double getdbl (“prompt_string”), int getint (“prompt_string”), long getlong (“prompt_string”)

Эти функции выводят пользователю приглашение для ввода числа, и после окончания ввода возвращают введенное значение. Если пользователь ничего не ввел, возвращается 0.

Example

```
>age = getint ("Enter Your Age")
```

void memset (start address, ulong length, uchar value)

Функция **memset** заполняет участок памяти, начиная с указанного адреса **address**, указанным значением **value**. Длина заполняемого участка памяти задается в **length**.

Example

```
>MEMSET (0x20000, 0x1000, 'a') /* Заполнит память с 0x20000 до 0x20FFF значением "a" */
```

void printf (“format_string”, ...)

Функция **printf** работает аналогично библиотечной функции ANSI C. Первый аргумент – строка форматирования. Следующие аргументы могут быть выражениями или строками. Все определенные стандартом ANSI C спецификации форматирования полностью применимы к **printf**.

Example

```
>printf ("random number = %04XH\n", rand(0))
random number = 1014H

>printf ("random number = %04XH\n", rand(0))
random number = 64D6H

>printf ("%s for %d\n", "uVision2", 8051)
uVision2 for 8051

>printf ("%lu\n", (ulong) -1)
4294967295
```

int rand (int seed)

Функция **rand** возвращает случайное число из интервала от -32768 до +32767. Генератор случайных чисел инициализируется каждый раз, когда в функцию передается аргумент *seed* отличный от нуля. Вы можете использовать функцию **rand** для формирования задержки на случайное количество тактов процессора или для генерации случайных данных, подаваемых на вход какого-либо алгоритма или процедуры.

Example

```
>rand (0x1234)      /* Генератор случайных чисел инициализируется значением 0x1234 */
0x3B98

>rand (0)          /* Без инициализации */
0x64BD
```

void twatch (long states)

Функция **twatch** может использоваться в сигнальных функциях для задержки выполнения функции на указанное количество *тактов* процессора. Отладчик μ Vision2 обновляет счетчик тактов во время выполнения вашей программы.

Example

Следующая сигнальная функция периодически переключает вход INT0 (P3.2) каждую секунду.

```
signal void int0_signal (void) {
    while (1) {
        PORT3 |= 0x04; /* перевести INT0 (P3.2) в «1» */
        PORT3 &= ~0x04; /* перевести INT0 (P3.2) в «0» и сгенерировать прерывание
    */
        PORT3 |= 0x04; /* перевести INT0 (P3.2) снова в «1» */
        twatch (CLOCK); /* подождать 1 секунду */
    }
}
```

Примечание

Функция **twatch** может вызываться только внутри сигнальных функций. Вызов не из сигнальных функций не допускается и приводит к появлению сообщения об ошибке.

void swatch (float seconds)

Функция **swatch** может использоваться в сигнальных функциях для задержки выполнения функции на указанное количество *секунд*.

Example

Следующая сигнальная функция периодически переключает вход INT0 (P3.2) каждые полсекунды.

```
signal void int0_signal (void) {
    while (1) {
        PORT3 |= 0x04; /* перевести INT0 (P3.2) в «1» */
        PORT3 &= ~0x04; /* перевести INT0 (P3.2) в «0» и сгенерировать прерывание
    */
        PORT3 |= 0x04; /* перевести INT0 (P3.2) снова в «1» */
        swatch (0.5); /* подождать 0,5 секунды */
    }
}
```

Примечание

Функция **swatch** может вызываться только внутри сигнальных функций. Вызов не из сигнальных функций не допускается и приводит к появлению сообщения об ошибке.

void rwatch (address)

Функция **rwatch** может использоваться в сигнальных функциях для задержки выполнения функции до тех пор, пока указанная ячейка памяти не будет прочитана.

Example

Следующая сигнальная функция периодически переключает вывод P1.0 каждый раз при чтении из памяти XDATA по адресу 0x1234.

```
signal void my_signal (void) {
    while (1) {
        PORT1 ^= 0x01;    /* переключение P1.0 */
        rwatch (X:0x1234); /* ожидание пока ячейка X:0x1234 не будет прочитана */
    }
}
```

Примечание

Функция **rwatch** может вызываться только внутри сигнальных функций. Вызов не из сигнальных функций не допускается и приводит к появлению сообщения об ошибке.

void wwatch (address)

Функция **wwatch** может использоваться в сигнальных функциях для задержки выполнения функции до тех пор, пока не произойдет операция записи в указанную ячейку памяти.

Example

Следующая сигнальная функция периодически переключает вывод P1.0 каждый раз при записи в память XDATA по адресу 0x4000.

```
signal void my_signal (void) {
    while (1) {
        PORT1 ^= 0x01;    /* переключение P1.0 */
        wwatch (X:0x4000); /*ожидание пока в X:0x4000 не будет записано что-либо*/
    }
}
```

Примечание

Функция **wwatch** может вызываться только внутри сигнальных функций. Вызов не из сигнальных функций не допускается и приводит к появлению сообщения об ошибке.

int _TaskRunning_ (ulong *func_address*)

Эта функция производит проверку того, что указанная задача является текущей выполняемой задачей.

Функция **_TaskRunning_** доступна только тогда, когда вы выбрали опцию **Operating System** на странице настроек проекта **Options for Target – Target**. Отладчик μ Vision2 загружает дополнительную DLL библиотеку, которая позволяет следить за состоянием ядра операционной системы. За более детальной информацией обращайтесь к параграфу «Отладка ядра RTX» на странице 180.

Результат функции **_TaskRunning_** может быть присвоен системной переменной **_break_** что позволит остановить выполнение программы когда указанная задача будет активна. Пример смотри на странице 182.

Example

```
>_TaskRunning_ (command)      /* проверяет - выполняется ли задача 'command' */
0001                          /* возвращает 1 если задача выполняется */

>_break_ = _TaskRunning_ (init) /* остановить, когда задача 'init' выполняется */
```

uchar _RBYTE (address),
uint _RWORD (address),
ulong _RDWORD (address),
float _RFLOAT (address),
double _RDOUBLE (address)

Эти функции возвращают содержимое памяти по указанному адресу *address*.

Example

```
>_RBYTE (0x20000) /* возвращает символ из ячейки 0x20000 */
>_RFLOAT (0xE000) /* возвращает дробное число из ячейки 0xE000 */
>_RDWORD (0x1000) /* возвращает длинное целое из ячейки 0x1000 */
```

`_WBYTE` (*address*, *uchar value*),
`_WORD` (*address*, *uint value*),
`_DWORD` (*address*, *ulong value*),
`_WFLOAT` (*address*, *float value*,
`_WDOUBLE` (*address*, *double value*)

Эти функции записывают значение *value* в память по указанному адресу *address*.

Example

```
>_WBYTE (0x20000, 0x55)      /* записать байт 0x33 в ячейку 0x20000 */  
>_RFLOAT (0xE000, 1.5)     /* записать дробное число 1.5 в ячейку 0xE000 */  
>_RDWORD (0x1000, 12345678) /* записать длинное целое 12345678 в ячейку 0x1000*/
```

Пользовательские функции

Пользовательские функции – это функции, которые вы создаете для использования во время отладки в среде μ Vision2. Эти функции вы можете непосредственно ввести в редакторе функций, или же можно использовать команду **INCLUDE** для загрузки файла, который содержит одну или несколько пользовательских функций.

Примечание

μ Vision2 предоставляет набор системных переменных, которые вы можете использовать в пользовательских функциях.

Для получения более полной информации смотрите параграф “Системные переменные” на странице 113.

Пользовательские функции начинаются с ключевого слова **FUNC** и определяются следующим образом:

```
FUNC return_type fname (parameter_list) {
    statements
}
```

<i>return_type</i>	Тип значения, возвращаемого функцией. Может быть: bit, char, float, int, long, uchar, uint, ulong, void . Вы можете использовать тип void если функция не возвращает значения. Если тип возвращаемого значения не указан, то по умолчанию принимается тип int .
<i>fname</i>	Имя функции.
<i>parameter_list</i>	Список параметров, передаваемых в функцию. Каждый аргумент должен иметь тип и имя. Если функция не имеет параметров, то используйте void в качестве списка параметров. Параметры разделяются запятой.
<i>statements</i>	Инструкции, выполняемые функцией.
{	Открывающая фигурная скобка. Определение функции считается завершенным, если количество открывающих фигурных скобок соответствует количеству закрывающих фигурных скобок.

Пример

Следующая пользовательская функция выводит на экран содержимое некоторых регистров процессора. За более полной информацией про «Создание функций» обратитесь на страницу 131.

```
FUNC void MyRegs (void) {
printf ("----- MyRegs() ----- \n");
printf (" R4  R8  R9  R10 R11 R12 \n");
printf (" %04X %04X %04X %04X %04X %04X \n",
R4, R8, R9, R10, R11, R12);
printf ("----- \n");
}
```

Для вызова этой функции наберите в командном окне следующее

```
MyRegs ()
```

После вызова функция **MyRegs** выведет содержимое регистров, что будет выглядеть примерно так:

```
----- MyRegs () -----
R4  R8  R9  R10 R11 R12
B02C 8000 0001 0000 0000 0000
-----
```

Вы можете определить кнопку на панели инструментов (toolbox), которая позволит вызывать пользовательскую функцию, следующей командой:

```
DEFINE BUTTON "My Registers", "MyRegs ()"
```

Ограничения

- Отладчик μ Vision2 проверяет, чтобы возвращаемое значение пользовательской функции соответствовало возвращаемому типу функции. Функции, которые имеют **void** в качестве возвращаемого типа, не должны возвращать никакое значение. Функции, определенные как возвращающее значение, отличное от **void**, должны возвращать какое-то значение. Заметьте, что отладчик μ Vision2 не проверяет каждую ветку с `return` на корректность возвращаемого значения.
- Пользовательские функции не могут вызывать сигнальные функции или функцию **twatch** *.
- Значение локальных переменных не определено до тех пор, пока им не будет присвоено какое-то значение.
- Для удаления пользовательских функций из памяти отладчика используйте команду **KILL FUNC**.

* *Прим. переводчика* – пользовательские функции могут все-таки вызывать сигнальные функции, но не могут вызывать функции `twatch`, `swatch`, `gwatch`, `wwatch`. Проверено на собственном опыте.

Сигнальные функции

Сигнальные функции позволяют вам запустить в фоновом режиме повторяющиеся операции, такие как подача сигналов и импульсов на входы процессора, в то время как отладчик μ Vision2 выполняет вашу программу. Сигнальные функции позволяют вам моделировать и отлаживать последовательный ввод-вывод, аналоговый ввод-вывод, взаимодействия с портами и другие повторяющиеся *внешние* события.

Сигнальные функции выполняются в фоновом режиме, в то время как отладчик μ Vision2 моделирует поведение вашей программы. Поэтому сигнальные функции должны в некоторый момент вызывать функцию **twatch**, чтобы сделать паузу и позволить отладчику μ Vision2 выполнять вашу программу. Отладчик μ Vision2 сообщает об ошибке, если сигнальная функция никогда не вызывает **twatch***

Примечание

μ Vision2 предоставляет набор системных переменных, которые вы можете использовать в сигнальных функциях.

Для получения более полной информации смотрите параграф “Системные переменные” на странице 113.

Сигнальные функции начинаются с ключевого слова **SIGNAL** и определяются следующим образом:

```
SIGNAL void fname (parameter_list) {
    statements
}
```

<i>fname</i>	Имя функции.
<i>parameter_list</i>	Список параметров, передаваемых в функцию. Каждый аргумент должен иметь тип и имя. Если функция не имеет параметров, то используйте void в качестве списка параметров. Параметры разделяются запятой.
<i>statements</i>	Инструкции, выполняемые функцией.
{	Открывающая фигурная скобка. Определение функции считается завершенным, если количество открывающих фигурных скобок соответствует количеству закрывающих фигурных скобок.

* Прим. переводчика – здесь и далее подразумевается одна из 4х функций ожидания: twatch, swatch, rwatc, wwatch.

Пример

Следующий пример показывает сигнальную функцию, которая помещает символ 'А' во входной буфер последовательного порта каждые 1 000 000 (1 млн.) тактов процессора. За более полной информацией про «Создание функций» обратитесь на страницу 131.

```
SIGNAL void StuffS0in (void) {  
    while (1) {  
        S0IN = 'A';  
        twatch (1000000);  
    }  
}
```

Для вызова этой функции наберите в командном окне следующее:

```
StuffS0in ()
```

После запуска сигнальная функция **StuffS0in** помещает символ 'А' во входной буфер последовательного порта, ждет 1 000 000 тактов процессора, затем повторяет все сначала.

Ограничения

Сигнальные функции имеют следующие ограничения:

- Типом возвращаемого значения сигнальной функции должен быть **void**.
- Сигнальная функция может иметь максимум 8 параметров.
- Сигнальные функции могут вызывать предопределенные и пользовательские функции.
- Сигнальная функция не может вызывать другую сигнальную функцию.
- Сигнальная функция может быть вызвана из пользовательской функции.*
- Сигнальная функция должна вызывать функцию **twatch** хотя бы один раз. Сигнальные функции, которые никогда не вызывают **twatch**, не предоставляют времени для выполнения моделируемой программы. Поскольку вы не можете использовать **Ctrl+C** для того, чтобы прервать выполнение сигнальной функции, отладчик μ Vision2 может войти в бесконечный цикл.

* *Прим. переводчика* – это утверждение истинно, хотя и противоречит ограничениям, накладываемым на пользовательские функции, которые приведены на странице 142.

Управление выполнением сигнальных функций

Отладчик μ Vision2 имеет очередь активных сигнальных функций. Сигнальная функция может находиться в одном из двух состояний – в режиме простоя или в режиме выполнения. Сигнальная функция, находящаяся в режиме простоя, – это функция, выполнение которой приостановлено пока она ожидает истечение интервала, заданного при вызове функции **twatch**. Сигнальная функция, находящаяся в режиме выполнения, выполняет операторы, заданные в ее теле.

Когда вы вызываете сигнальную функцию, отладчик μ Vision2 добавляет ее в очередь и помечает как функцию в режиме выполнения. Сигнальная функция может быть активизирована один раз, если функция уже помещена в очередь, то будет выведено предупреждающее сообщение. Для просмотра состояния активных сигнальных функций используйте команду **SIGNAL STATE**. Для удаления активной сигнальной функции из очереди используйте команду **SIGNAL KILL**.

Когда сигнальная функция вызывает функцию **twatch**, то она переходит в режим простоя на количество тактов процессора, заданных в параметре вызова функции **twatch**. После того, как при выполнении программы счетчик тактов процессора изменится на заданную величину, сигнальная функция перейдет в режим выполнения. Выполнение функции продолжится с оператора, следующего за вызовом **twatch**.

Если сигнальная функция завершается естественным путем, при достижении оператора `return`, то она автоматически удаляется из очереди активных сигнальных функций.

Пример аналогового генератора

Следующий пример демонстрирует сигнальную функцию, которая изменяет сигнал на аналоговом входе 0 процессора со встроенным АЦП. Эта функция увеличивает и уменьшает входное напряжение на 0,1В от 0В и до верхнего порога, который задается параметром при вызове сигнальной функции. Сигнальная функция повторяет свою работу в вечном цикле, делая задержку на 200 000 тактов процессора после каждого шага по напряжению.

```
signal void analog0 (float limit) {
    float volts;

    printf ("Analog0 (%f) entered.\n", limit);
    while (1) {
        volts = 0;
        while (volts <= limit) {
            ain0 = volts;
            twatch (200000);
            volts += 0.1;
        }
        volts = limit;
        while (volts >= 0.0) {
            ain0 = volts;
            twatch (200000);
            volts -= 0.1;
        }
    }
}
```

Сигнальная функция **analog0** может быть запущена следующим образом:

```
>ANALOG0 (5.0)
ANALOG0 (5.000000) ENTERED
```

Команда **SIGNAL STATE** показывает текущее состояние функции **analog0**:

```
>SIGNAL STATE
1 idle      Signal = ANALOG0 (line 8)
```

μ Vision2 показывает внутренний номер функции, статус сигнальной функции: простой или выполнение (idle или running), имя функции и номер строки, которая выполняется в данный момент.

Поскольку мы видим, что сигнальная функция находится в режиме простоя, то можно сделать вывод, что **analog0** выполняет функцию **twatch** (строка 8 функции **analog0**) и ожидает истечения интервала времени, заданного в тактах процессора. Когда 200 000 тактов пройдет, функция **analog0** продолжит свое выполнение, пока не встретит следующий вызов **twatch** в строке 8 или 14.

Следующая команда удаляет сигнальную функцию **analog0** из очереди активных сигнальных функций.

```
>SIGNAL KILL ANALOG0
```

Различия между отладочными функциями и Си-функциями

Существует ряд различий между стандартом ANSI C и подмножеством языка, который поддерживается в отладчике μ Vision2 в пользовательских и сигнальных функциях.

- В подмножестве μ Vision2 не различаются прописные и строчные буквы. Имена объектов и управляющие операторы могут быть записаны, как прописными, так и строчными буквами.
- Подмножество μ Vision2 не имеет препроцессора. Такие директивы препроцессора как **#define**, **#include**, и **#ifdef** не поддерживаются.
- Подмножество μ Vision2 не поддерживает объявления глобальных переменных. Скалярные переменные должны объявляться внутри определений функций. Вы можете определить символы командой **DEFINE** и затем использовать их аналогично глобальным переменным.
- В подмножестве μ Vision2, переменные не могут быть инициализированы при объявлении. Для инициализации переменных должна использоваться инструкция явного присваивания значения.
- Функции в подмножестве μ Vision2 поддерживают только скалярные типы переменных. Структуры, массивы и указатели не допускаются. Это правило распространяется на тип значения возвращаемого функцией, а также на параметры функции.
- Функции в подмножестве μ Vision2 не могут вызываться рекурсивно. Во время выполнения функции отладчик μ Vision2 распознает рекурсивные вызовы и тотчас прерывает выполнение функции.
- В подмножестве μ Vision2 функции могут вызываться только явным образом по имени функции. Косвенный вызов функции через указатели не поддерживается.
- Подмножество μ Vision2 поддерживает только стиль ANSI для декларации списка параметров функции. Старый формат K&R не поддерживается. Например, в следующей функции использован допустимый стиль ANSI.

```
func test (int pa1, int pa2) {                               /* стиль ANSI, все корректно */
/* ... */
}
```

В следующем примере используется стиль K&R, что не допускается.

```
func test (pa1, pa2)                                       /* Старый стиль K&R */
int pa1, pa2;                                             /* не поддерживается */
{
/* ... */
}
```

Параграф не переведен, поскольку представляет очень узкий интерес для тех, кто ранее работал с dScope

Differences Between dScope and the μ Vision2 Debugger

The μ Vision2 debugger replaces the Keil dScope for Windows. dScope debug functions require the following modifications for correct execution in the μ Vision2 debugger.

- In dScope the **memset** debug function parameters are different. The μ Vision2 **memset** debug function parameters are now identical with the ANSI C **memset** function.
- The dScope debug function **bit** is no longer available and needs to be replaced with **_RBYTE** and **_WBYTE** function calls. With dScope debug functions **char**, **uchar**, **int**, **uint**, **long**, **ulong**, **float**, and **double** it is possible to read and write memory. Replace these debug functions in μ Vision2 according the following list.