Проектирование с использованием процессоров Analog Devices. Первый проект

В этой статье мы перейдем от теории к практике и создадим первый простой проект, реализующий скалярное произведение двух векторов, который, хоть и не будет иметь большой практической ценности, позволит освоить базовые принципы работы над проектами в среде VisualDSP++ и познакомиться с основами написания программ на языке ассемблера. Для определенности в этом и во всех последующих проектах мы будем использовать процессор ADSP-BF527, являющийся типичным представителем наиболее популярного семейства процессоров компании Analog Devices — ADSP-BF5xx Blackfin.

Для первого проекта нам потребуется установленная на компьютере последняя версия среды VisualDSP++ с тестовой лицензией. Для запуска среды выберите пункт Analog Devices → VisualDSP++ 5.0 → VisualDSP++ Environment меню Пуск ОС Windows. Если никаких проектов в VisualDSP++ вы пока не создавали, то в основном окне программы откроются только два окна — окно проектов (Project Window) и окно вывода информации (Output Window).

Александр СОТНИКОВ,
к. т. н.
Alexander.sotnikov@analog.com.ru

Создание проекта

Чтобы создать новый проект, выберите в меню **File VisualDSP++** пункт **New → Project…** При этом откроется окно «мастера» создания проектов (**Project Wizard**). Первое, что предлагает сделать «мастер» создания проектов, — это выбрать тип приложения, задать его название и каталог, в котором он будет располагаться. VisualDSP++ позволяет создавать приложения четырех типов: стандартные приложения, приложения с поддержкой ядра операционной системы реального времени VDK, приложения с поддержкой стека LwIP для сети Ethernet и пользовательские библиотеки. Наш первый проект будет представлять собой стандартное приложение (**Standard Application**) и иметь название **First**. При нажатии кнопки **Next** «мастер» создания проектов попросит указать семейство и тип процессора, для которого будет разрабатываться предложение, а также ревизию кристалла процессора. При выборе конкретной ревизии кристалла на этапе компоновки проекта будут использованы библиотеки, содержащие программные решения для «обхода» аномалий (аппаратных ошибок) этой ревизии. Как уже отмечалось, проект будет создаваться под процессор ADSP-BF527 семейства Blackfin, а отладка программы будет вестись с использованием симулятора. Настройки для ревизии кристалла в этом проекте мы изменять не будем.

На следующем шаге предлагается добавить в создаваемый проект шаблон исходной программы на языке C/C++ или ассемблера и выбрать тип выходного файла, который будет сгенерирован в процессе сборки проекта: исполняемый файл (.dxe) или файл загружаемого образа (.ldr). На этой вкладке измените предлагаемый по умолчанию язык шаблона программы C на ASM и оставьте все остальные настройки без изменения.

Заключительный этап работы «мастера» создания проектов состоит в настройке startup-кода и файла описания линкера. Эти два вспомогательных файла играют очень важную роль в проекте. Startup-код — это процедура, которая выполняется после сброса процессора или включения питания и вызывает функцию **_main**. Она скрывает от разработчика нюансы инициализации процессора и настройки программного окружения, позволяя сосредоточиться непосредственно на алгоритмической составляющей программы. Файл описания линкера содержит информацию об архитектуре памяти системы, необходимую для компоновки исполняемого файла. Соответствующая вкладка «мастера» создания проектов содержит три варианта выбора:

- Добавить в проект оба вспомогательных файла.
- Добавить в проект только startup-код.
- Не добавлять вспомогательные файлы в проект.

При выборе одного из двух первых вариантов появятся дополнительные опции, позволяющие модифицировать стандартные настройки файлов с учетом требований конкретного приложения. Если startup-код и/или файл описания линкера в проект не добавляются, то на этапах компиляции и компоновки будут использованы стандартные файлы для выбранного типа процессора. На данном этапе мы выберем третий пункт (не добавлять startup-код и файл описания линкера в проект).

На последней вкладке «мастер» создания проектов отображает информацию об опциях, выбранных на каждом из предыдущих шагов (рис. 1). При желании вы можете внести изменения в настройки проекта, нажав кнопку **Back** или выбрав соответствующий пункт древовидного списка в левой части окна «мастера» создания проектов. Для принятия настроек и завершения работы «мастера» нажмите кнопку **Finish**.

В результате работы «мастера» будет создан файл проекта **First.dpj**, в окне проектов

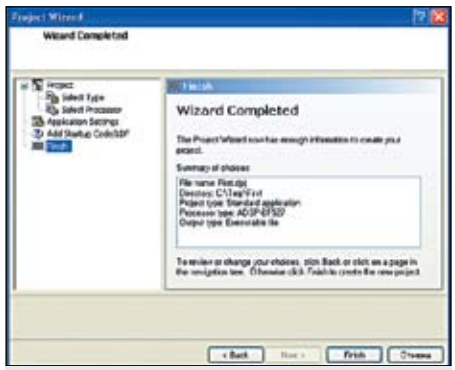

Рис. 1. Диалоговое окно «мастера» создания проектов

появится элемент с именем проекта и откроется окно редактора с шаблоном исходного файла **First.asm**. Закроем на некоторое время окно редактора и познакомимся более подробно со структурой окна проектов.

Окно проектов

Общие принципы работы с окном проектов покажутся хорошо знакомыми тем, кто имеет опыт работы в среде разработки Microsoft VisualStudio. Оно может иметь две вкладки: **Project**, которая доступна всегда, и **Kernel**, которая появляется только в проектах с поддержкой VDK. Во вкладке **Project** (рис. 2) отображается дерево иерархии проектов, на верхнем уровне которого находится группа проектов (**Project Group**). Группа может состоять из нескольких проектов, однако в отдельно взятый момент времени активным может быть только один. Проект включает в себя несколько папок. Три папки присутствуют в проекте всегда:

- **Source Files** (исходные файлы);
- **Linker Files** (файлы линкера);
- **Header Files** (заголовочные файлы).

Также в списке папок может присутствовать папка **Generated Files**, куда помещается автоматически генерируемый «мастером» создания проекта startup-код, и папки, создаваемые пользователем. Эти папки служат только для упорядочивания проекта. Они никак не связаны с реальной файловой системой компьютера, а их структура не оказывает влияния на процесс сборки проекта. Вы можете задавать соответствия между папками и расширениями файлов, помещаемых в них. После добавления к проекту нового файла он будет автоматически помещен в первую папку, поддерживающую его расширение. После того как файл автоматически помещен в папку, вы можете переместить его в другую папку, перетащив мышью. Если проект зависит от других проектов, то под списком папок проекта отображаются иконки с их именами. При сборке проекта среда VisualDSP++ также выполняет сборку всех проектов, от которых он зависит.

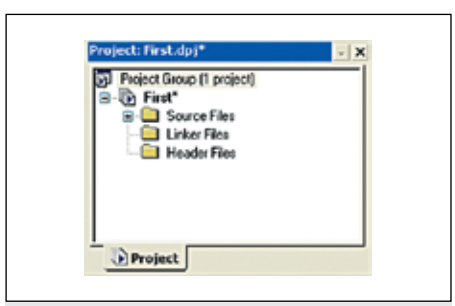

Рис. 2. Окно проектов

Щелчок правой кнопкой мыши по элементу дерева иерархии проектов вызывает контекстное меню, которое меняется в зависимости от типа элемента. При вызове контекстного меню для группы проектов будут доступны команды добавления/удаления проектов, а также команды, распространяющиеся глобально на все проекты группы (сборка/очистка всех проектов). Если вызвать контекстное меню для отдельного проекта, то оно будет содержать команды сборки/очистки данного проекта, закрытия проекта, создания папок и добавления в проект новых файлов. Еще один очень важный пункт контекстного меню проекта вызывает окно **Project Options** («Опции проекта»), через которое пользователь может управлять параметрами утилит проектирования (ассемблера, компилятора, линкера и т. д.). Контекстное меню, вызываемое при нажатии правой кнопки мыши на папке, содержит команды создания и удаления папок, добавления файлов в папку, а также команду **Properties**, при помощи которой можно назначать папке типы расширений. Через контекстное меню, вызываемое при выборе файла, можно открыть файл в текстовом редакторе, запустить компоновку файла, удалить файл из проекта. Также через это контекстное меню можно вызвать окно **File Options** («Опции файла»), при помощи которого для файла можно установить опции ассемблирования/компоновки, отличные от глобальных настроек проекта, выбранных в окне **Project Options**.

При работе с файлами и папками проекта в окне **Project** действуют следующие правила:

- В проект могут добавляться любые типы файлов, однако файлы нераспознанных типов будут игнорированы при сборке проекта. Автоматически средой VisualDSP++ распознаются файлы компилятора (.c, .cpp и .cxx), файлы ассемблера (.asm, .s и .dsp) и файлы линкера (.ldf, .dlb и .doj).
- В проекте разрешается иметь только один файл описания линкера. Если файл описания линкера в проект не добавлен, то используется стандартный файл описания линкера для выбранного процессора из папки **Путь установки\Blackfin\ldf**.
- Один и тот же файл может быть добавлен в проект только один раз.
- При добавлении файла в проект он помещается в первую папку, поддерживающую данный тип расширения. Если такой папки в проекте нет, то файл добавляется на верхний уровень иерархии проекта.

Окно редактора

При двойном щелчке по иконке или имени исходного файла в дереве иерархии проектов или выборе пункта **Open File** соответствующего контекстного меню выбранный файл открывается в окне редактора. Редактор VisualDSP++ поддерживает множество функций, упрощающих работу с файлами исходного кода, такие как:

- цветовое выделение синтаксических конструкций;
- быстрый переход между строками с синтаксическими ошибками;
- автоматическое отображение значений выражений и переменных в режиме отладки при наведении указателя мыши и т. д.

В левой части окна редактора находится вертикальная полоса серого цвета, которая используется для добавления закладок в текст программы, позволяющих быстро перемещаться между интересующими пользователя частями большого файла, задания точек останова при отладке программы и отображения текущего положения программного автомата процессора.

Исходные тексты программ на языке С могут отображаться в окне редактора в одном из двух режимов. В стандартном режиме (**Source**) на экран выводится только код на С, а в смешанном режиме (**Mixed**) после строк на языке С отображается также соответствующий им код на языке ассемблера. Чтобы код можно было просматривать в смешанном режиме, исходный файл должен быть скомпилирован с поддержкой отладочной информации. Кроме того, после компиляции исходных файлов на языке С/С++ в окне редактора могут отображаться аннотации компилятора, содержащие информацию о том, насколько отдельные фрагменты кода близки к «оптимальному» стилю кодирования и что можно сделать для повышения эффективности кода. На наличие аннотаций указывает значок в виде буквы “i” в кружке слева от строчки исходного кода.

Щелчок правой кнопкой мыши в рабочей области окна редактора вызывает контекстное меню, содержащее набор стандартных команд редактирования, команд установки/снятия закладок и точек остановки. Кроме того, при помощи контекстного меню выполняется переключение режимов отображения исходного кода (**Source/Mixed**), управление отображением номеров строк исходного кода, а также формата, в котором будут выводиться результаты выражений и значения переменных при наведении курсора в режиме отладки.

Редактирование исходного файла

Откроем любым из перечисленных выше способов исходный файл first.asm, автоматически сгенерированный «мастером» создания проекта, но перед тем как приступить к его редактированию, посвятим некоторое время

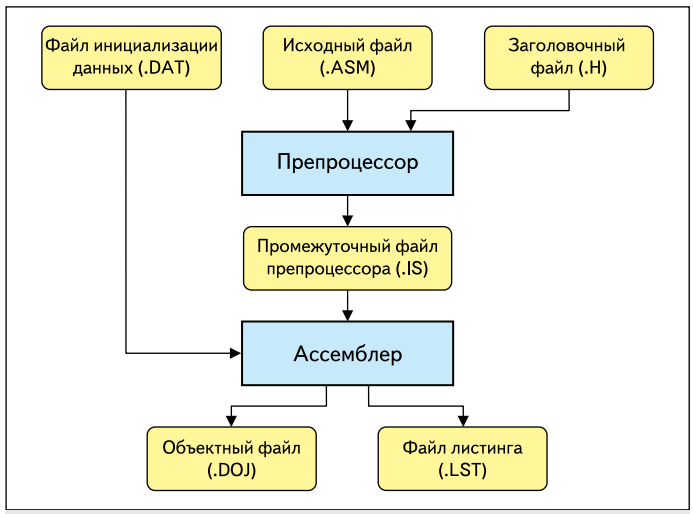

Рис. 3. Файлы и утилиты, участвующие в процессе ассемблирования

рассмотрению основ синтаксиса и общей структуры исходных файлов на языке ассемблера процессоров Blackfin. Полную информацию о синтаксисе ассемблера и наборе команд процессоров Blackfin можно найти в [1, 2].

Утилита ассемблер преобразовывает файлы, написанные на языке ассемблера пользователем или преобразованные в язык ассемблера компилятором языка C/C++, в объектные файлы. Как показано на рис. 3, ассемблер обрабатывает исходные файлы (с расширением .asm), файлы данных (.dat) и подключаемые заголовочные файлы (.h), формируя на выходе объектный файл промышленного формата двоичных файлов ELF (executable and linkable format) и необязательный файл листинга (.lst), содержащий справочную информацию о результатах ассемблирования. Исходные файлы могут содержать команды препроцессора, поэтому перед непосредственно ассемблированием вызывается утилита, которая называется препроцессор. Единственным результатом работы препроцессора является промежуточный препроцессированный исходный файл (.is).

Исходный файл на языке ассемблера состоит из набора команд ассемблера, директив ассемблера и команд препроцессора.

Команды ассемблера соответствуют синтаксису команд семейства и всегда заканчиваются точкой с запятой (“;”). Отдельные команды, входящие в состав составной многофункциональной команды, разделяются запятой (“,”) или двумя вертикальными чертами (“||”). Первый вариант соответствует параллельному выполнению двух общих команд, а второй — одновременному выполнению общей команды и одной или двух команд доступа к памяти. Команды ассемблера, равно как и другие зарезервированные слова (имена регистров, опции команд и т. д.) не чувствительны к регистру, однако для улучшения восприятия кода удобно записывать их в верхнем регистре. Команды ассемблера могут сопровождаться адресными метками, которые используются в тексте программы вместо абсолютного адреса ячейки памяти, где хранится соответствующая команда. Метка помещается в начале строки с командой ассемблера или в конце предыдущей строки и заканчивается двоеточием (“:”). Метки могут содержать любое количество символов и чувствительны к регистру. Так, “label:” и “Label:” интерпретируются ассемблером как две разные метки.

Директивы ассемблера управляют процессом ассемблирования исходного файла. В отличие от команд ассемблера при ассемблировании из директив не формируются машинные коды. Любая директива ассемблера начинается символом точка “.” и заканчивается точкой с запятой “;”. Некоторые директивы требуют использования квалификаторов и аргументов. Квалификатор следует непосредственно за директивой и отделяется от нее обратной косой чертой (“/”), аргумент записывается после квалификатора или после директивы (если квалификатор отсутствует) и отделяется от них пробелом. Таким образом, директивы ассемблера имеют обобщенный синтаксис вида:

```
.директива  [/квалификаторы |аргументы];
```

Директивы ассемблера не чувствительны к регистру, однако запись директив в верхнем регистре облегчает визуальное восприятие кода.

К наиболее часто используемым директивам относятся:

- **.GLOBAL** (изменяет область видимости идентификатора с локальной на глобальную);
- **.EXTERN** (позволяет использовать в исходном файле глобальные идентификаторы, определенные в других исходных файлах);
- **.SECTION** (указывает на начало секции);
- директивы определения и инициализации переменных (**.VAR**, **.BYTE**, **.BYTE2** и **.BYTE4**).

Команды препроцессора используются для задания макросов и символьных констант (#define), подключения заголовочных файлов (#include), проверок на ошибки и управления условным ассемблированием (#ifdef и др.) и обрабатываются препроцессором перед началом ассемблирования исходного файла.

Команды препроцессора начинаются с символа “#” и заканчиваются символом возврата каретки. Символ “#” должен быть первым символом в строке, содержащей команду препроцессора, не считая пробелов. Если команда не помещается на одной строке, то для ее переноса на следующую строку используется комбинация обратной косой черты “\” и символа возврата каретки. Команды препроцессора чувствительны к регистру и должны записываться строчными буквами.

Для добавления комментариев в языке ассемблера процессоров Blackfin используются конструкции, аналогичные языкам C/C++: комментарии, распространяющиеся на одну строку, начинаются с символов “//”, а комментарии, распространяющиеся на несколько строк, выделяются с двух сторон комбинацией символов “/*” и “*/”.

И, наконец, для записи чисел в языке ассемблера процессоров Blackfin используются обозначения, указанные в таблице.

Таблица. Форматы чисел

Форма записи	Пример	Описание
0xчисло	0x0123ABCD	Число в шестнадцатеричном формате
B#число или b#число	b#0000000000000001	Число в двоичном формате
Число	55	Число, записанное без префикса, соответствует десятичному формату
Числог	0.5r	Число с суффиксом r соответствует дробному формату 1.15 (используется по умолчанию) или 1.31 с диапазоном значений чисел [–1...1)

Команды ассемблера и данные, используемые в исходном файле, должны быть определенным образом сгруппированы, чтобы линкер смог сопоставить их с информацией в файле описания линкера (.ldf) при формировании исполняемого файла. Такое структурирование обеспечивает сам разработчик программы, помещая различные части кода или данных в именованные секции. Объектные файлы, создаваемые ассемблером или компилятором (с использованием ассемблера), состоят из входных секций, каждая из которых содержит определенный тип компилированного/ассемблированного исходного кода. Например, одна входная секция может содержать машинные коды программы, а другая — данные. Некоторые входные секции могут содержать отладочную информацию.

Для размещения кода или данных в определенных секциях в исходных файлах на языке ассемблера используется директива **.SECTION**. Эта директива указывает, что группа команд (данных), записанных (объявленных) после нее, должна занимать смежную непрерывную область адресов в карте памяти процессора. Входные секции должны иметь уникальные имена, совпадающие с именами входных секций в LDF-файле.

Итак, вернемся к нашему исходному файлу (рис. 4) и рассмотрим его более внимательно. Пока что он содержит лишь общий каркас

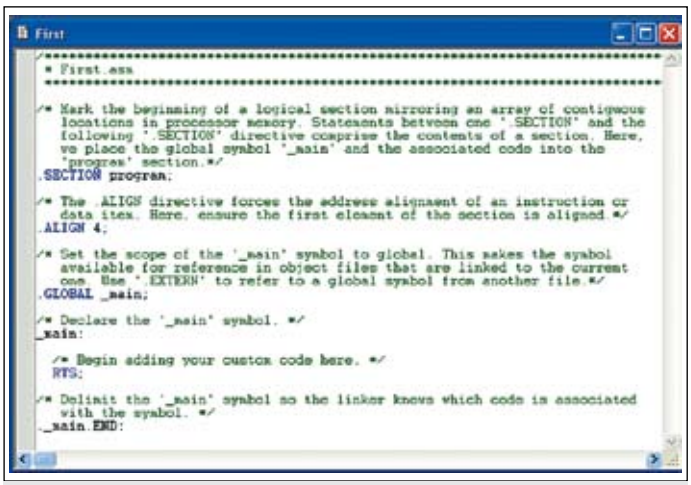

Рис. 4. Окно редактора с шаблоном исходного кода

программы, который включает в себя три директивы, две метки и одну команду ассемблера.

Первая директива указывает на то, что код нашей программы будет размещаться во входной секции под названием program. Это имя является стандартным именем, рекомендуемым для секций с кодом программы процессоров Blackfin. Директива **.ALIGN 4;** указывает на то, что первый элемент секции program должен располагаться по адресу, кратному четырем. И, наконец, директива **.GLOBAL** определяет область видимости функции main. Эта директива позволяет вызывать функцию **_main** из startup-кода, который содержит парную директиву **.EXTERN_main;**.

Единственная команда, которая пока содержится в функции _main, — это команда возврата из подпрограммы (RTS;).

Модифицируем наш исходный код следующим образом. Между строками комментария и первой директивой программы добавим объявление и инициализацию двух массивов длиной пять элементов, которые будут размещаться в секции со стандартным именем data1:

```
.SECTION data1;
.ALIGN 4;
.VAR buf1[5] = 0x1,0x2,0x3,0x4,0x5;
.VAR buf2[5] = 0x2,0x4,0x6,0x8,0xa;
В тело функции main перед командой RTS; мы добавим следующие строки:
P0 = buf1.h;
P0 = buf1.l;
P1 = buf2.h;
P1 = buf2.l;
P2 = 4;
R0 = 0;
R1 = [P0++];
R2 = [P1++];
LSETUP (begin_loop, end_loop) LC0 = P2;
begin_loop:   R1 *= R2;
              R2 = [P1++];
end_loop: R0 = R0 + R1 || R1 = [P0++] || NOP;
R1 *= R2;
R0 = R0 + R1;
```

В этом коде регистры P0 и P1 будут использоваться для индексирования массивов buf1 и buf2. Регистры R1 и R2 будут играть роль источников входных операндов АЛУ, а в регистре R0 будет накапливаться результат скалярного произведения векторов, элементы которых хранятся в массивах buf1 и buf2.

По причине ограниченного объема статьи мы лишь вкратце обсудим назначение команд в этой программе. Подробную информацию о командах процессоров Blackfin можно найти в [2].

Первые четыре строчки кода инициализируют регистры указателей. Как и в языках C/C++, в языке ассемблера процессоров Blackfin имя массива является указателем на его первый элемент. Поскольку адреса буферов являются 32-битными, а ассемблер допускает непосредственную загрузку в регистры только 7- или 16-битных констант, инициализация регистров P0 и P1 осуществляется в два этапа: сначала загружается старшая часть адреса, а затем — младшая.

Следующие две строчки инициализируют регистр P2, который используется для промежуточного хранения количества итераций цикла, и регистр R0.

Команды Rx = [Px++]; осуществляют загрузку в регистр Rx содержимого ячейки памяти, на которую указывает регистр Px, с последующим инкрементом Px.

Команда LSETUP используется для настройки аппаратного цикла. Метки begin_loop и end_loop указывают границы тела цикла (адреса первой и последней команды, которые помещаются в регистры LT0 и LB0 процессора), а количество итераций цикла задается при помощи регистра счетчика циклов LC0.

В теле цикла производится перемножение элементов массивов, сложение полученного результата с накопленной в регистре R0 суммой результатов предыдущих умножений и загрузка новых элементов массивов. Последняя команда в теле цикла является многофункциональной. Обратите внимание на то, что количество итераций взято на единицу меньше, чем количество элементов буфера. Так происходит потому, что первая загрузка регистров R1 и R2 осуществляется до начала цикла, а последние операции умножения и сложения — по завершении цикла.

Запуск отладочной сессии и отладка программы

После того как редактирование исходного файла завершено, необходимо подключиться к целевому объекту отладки (в нашем случае это симулятор процессора ADSP-BF527), создав и запустив отладочную сессию. Сделать это можно, нажав кнопку **Connect to Target** на панели управления или выбрав пункт меню **Session → New Session**. При этом откроется окно «мастера» настройки сессий (**Session Wizard**), показанное на рис. 5.

Процесс настройки сессии симулятора ADSP-BF527 достаточно очевиден. Некоторого пояснения требует только третий этап (**Select Platform**), на котором пользователю будет предложено сделать выбор между стандартным симулятором (ADSP-BF5xx Single Processor Simulator) и компилятивным симулятором (Blackfin Family Compiled Simulator). Стандартный потактовый симулятор обеспечивает максимально подробное моделирование, но работает довольно медленно. В свою очередь, компилятивный симулятор позволяет значительно повысить скорость за счет того, что моделирование обеспечивается на функциональном уровне. Для нашего проекта мы выберем первый вариант. После закрытия окна «мастера» в строке заголовка

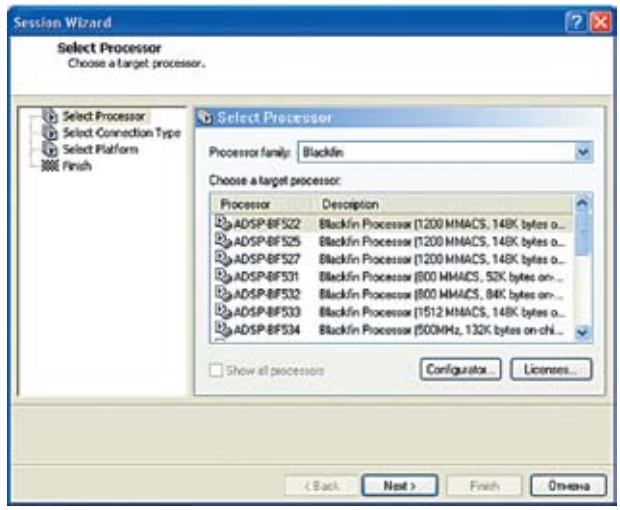

Рис. 5. Диалоговое окно «мастера» настройки сессий (Session Wizard)

главного окна VisualDSP++ появится надпись, сигнализирующая об удачном открытии сессии с целевым объектом отладки ([Target: ADSP-BF527 ADSP-BF5xx Single Processor Simulator]).

При повторных запусках среды VisualDSP++ последняя используемая сессия будет открываться автоматически, если установлен флажок **Restore last session on startup** на вкладке **General** диалогового окна **Preferences**. Это диалоговое окно вызывается через пункт **Settings → Preferences** меню **VisualDSP++** и используется для настройки различных параметров среды VisualDSP++.

Запустить сборку проекта можно разными способами: через кнопку панели инструментов, меню **Project → Build Project**, соответствующий пункт контекстного меню или нажатием кнопки **F7**. Выберите любой удобный для вас способ и попробуйте запустить сборку проекта. Информация о ходе процесса сборки проекта отображается на вкладке **Build** окна вывода информации (**Output Window**). В нашем случае в этом окне появится информация об ошибке: [Error ea5006] ".\First.asm":30 Illegal multi issue construct. При двойном щелчке по этой надписи курсор в окне редактора переместится на соответствующую строку исходного файла. Эта ошибка вызвана тем, что при использовании в составе многофункциональной команды команда сложения должна обязательно сопровождаться опцией, определяющей поведение АЛУ при переполнении результата, (S), если будет производиться насыщение, или (NS), если насыщение выполняться не будет.

Исправим строку с ошибкой так, чтобы она приняла вид:

```
end_loop: R0 = R0 + R1 (NS) || R1 = [P0++] || NOP;
```

и снова запустим сборку проекта. Теперь все должно пройти удачно. Полученный в результате исполняемый файл с расширением .dxe автоматически будет загружен в программную модель, если на вкладке **General** диалогового окна **Preferences** установлен флажок **Load executable after build**. Если в том же окне установлен флажок **Run to main after load**, то симулятор после загрузки выполнит в фоновом режиме все команды инициализации, содержащиеся в startup-коде, и остановится в начале функции **_main**. При этом на вкладке **Console** окна **Output Window** будет отображена информация о том, что исполняемый файл успешно загружен и программа достигла точки останова по адресу ffa108b0. Кроме того, справа от окна редактора откроется окно с дизассемблированным текстом (Disassembly).

Обратите внимание на то, что в серой полосе в левой части окна редактора напротив первой команды функции **_main** появятся два символа — желтая стрелка, указывающая текущее положение программного автомата процессора (команду, которая будет исполняться процессором на следующем такте), и красный кружок, символизирующий точку останова. Аналогичные символы появляются и в окне дизассемблированного текста. Точка останова в начале функции **_main** устанавливается средой VisualDSP++ автоматически. Вы можете добавить собственные точки останова двойным щелчком мыши слева от интересующей команды в окне редактора или окне дизассемблированного текста. Установить или снять точку останова можно также при помощи кнопок панели инструментов или контекстного меню окна редактора. В нашем примере имеет смысл остановить точку останова напротив закрывающей метки функции **_main**.

Для запуска программы на исполнение нужно нажать кнопку Run панели инструментов, выбрать пункт **Run** меню **Debug** или нажать клавишу F5. Однако больший интерес с точки зрения ознакомления с процессором и средой разработки представляет режим пошагового исполнения программы, который управляется клавишей F11.

Чтобы проследить, как изменяются регистры процессора в процессе выполнения программы, нужно открыть соответствующие окна отладки, вызываемые через пунк-ты меню **Register → Core → Data Register File** (регистры R0, R1, R2), **Register → Core → Loop Counters** (регистры LC0, LT0 и LB0) и **Register → Core → P Registers** (регистры P0, P1 и P2). Вы также можете создать свое собственное окно отладки, которое будет содержать только интересующие вас регистры, используя диалоговое окно **Manage Custom Register Windows**, вызываемое через пункт меню **Register → Custom → Manage**. Чтобы изменить режим отображения содержимого регистров в выбранных окнах отладки, можно вызвать контекстное меню, щелкнув правой кнопкой мыши внутри них. В нашем примере для окон **P Registers** и **Loop Counters** можно оставить выбранной по умолчанию формат **Hex** (шестнадцатеричные числа), а для окна **Data Register File** удобно задать формат **Unsigned Integer** (беззнаковые целые числа).

Если в какой-то момент вам понадобится перезапустить программу сначала, воспользуйтесь функцией перезагрузки исполняемого файла, которая может быть вызвана при помощи соответствующей кнопки панели инструментов, пункта меню или комбинации клавиш Ctrl+R.

Поэкспериментировав с пошаговым режимом и точками останова, вы сможете получить первые представления о процессе отладки проектов. В то же время рассмотренные в этой статье возможности отладки составляют лишь малую часть того, что позволяет разработчику среда VisualDSP++. В последую-щих статьях цикла мы познакомимся с более продвинутыми возможностями отладки в режиме симулятора, включая графическое отображение данных, моделирование работы периферийных устройств и рядом других.

Литература

1. VisualDSP++ 5.0 Assembler and Preprocessor Manual, Revision 3.3, September 2009, Analog Devices Inc.
2. Blackfin Processor Programming Reference, Revision 1.3, September 2008, Analog Devices Inc.