

# СИСТЕМА ПРОГРАММИРОВАНИЯ ОТЕЧЕСТВЕННОЙ СЕРИИ СИГНАЛЬНЫХ КОНТРОЛЛЕРОВ «МУЛЬТИКОР»

**Дмитрий Бочарников**, технический директор, ЗАО «Интерстрон»  
**Игорь Замятин**, ведущий программист, ЗАО «Интерстрон»  
**Станислав Крысенков**, программист-разработчик, ЗАО «Интерстрон»  
**Вадим Синицын**, ведущий инженер-программист, «Элвис»

**Самая совершенная архитектура процессора не позволяет воспользоваться всеми ее преимуществами, если для нее отсутствует система программирования. В некотором смысле, система программирования — это «костный мозг» вычислительной системы, позволяющий «вырабатывать» качественное программное обеспечение, делая ее привлекательной для пользователя. В статье представлена инновационная концепция разработки системы программирования для первой линейки отечественных DSP-контроллеров семейства «Мультикор», разработанной в ГУП НПЦ «Элвис» (г. Зеленоград).**

Самая совершенная архитектура процессора не позволяет воспользоваться всеми ее преимуществами, если для нее отсутствует система программирования. В некотором смысле, система программирования — это «костный мозг» вычислительной системы, позволяющий «вырабатывать» качественное программное обеспечение, делая ее привлекательной для пользователя. В статье представлена инновационная концепция разработки системы программирования для первой линейки отечественных DSP-контроллеров семейства «Мультикор», разработанной в ГУП НПЦ «Элвис» (г. Зеленоград).

Концепция является результатом объединения Инструментальной системы для разработки и отладки программ (MCStudio™, «Элвис») и пакета средств разработки программ для процессоров серии «Мультикор», созданного фирмой «Интерстрон», центральной частью которого служит ее компилятор полного стандарта для языка C/C++. Появление нового продукта на базе данной концепции — MCStudio-ECL — значительно упростит проблему переноса ранее наработанного программного обеспечения с ряда зарубежных DSP-платформ (таких, как TMS 320xx Texas Instruments или Tiger SHARC Analog Devices) на отечественную платформу «МУЛЬТИКОР».

## ВВЕДЕНИЕ

Разработки новых архитектур процессоров, если они не направлены на решение определенной задачи, отно-

сятся к одному из двух классов (или реже — к обоим сразу) и призваны:

- заменить старую архитектуру, придав вычислительному процессу большую надежность, быстродействие, детерминированность;
- освоить новое направление развития за счет возможностей архитектурных решений нового процессора. Пример — создание КПК за счет миниатюризации элементной базы и новых (стандартных) архитектурных решений.

При разработке первой отечественной линейки сигнальных контроллеров семейства «Мультикор» специалисты ГУП НПЦ «Элвис» ставили перед собой следующие задачи:

- разработать конкурентоспособную с зарубежными аналогами архитектуру процессора, наилучшим образом решающую определенный класс задач;
- разработать систему программирования, позволяющую получить исполняемый код, по своим характеристикам близкий к оптимальному (написанному на Ассемблере);
- для реализации на многопроцессорной вычислительной системе использовать ранее разработанное ПО для однопроцессорных систем;
- предоставить разработчикам возможность создания ПО на традиционных языках программирования.

## БАЗОВАЯ СЕРИЯ СИГНАЛЬНЫХ КОНТРОЛЛЕРОВ «МУЛЬТИКОР»

Отечественные сигнальные микроконтроллеры (МК) «Мультикор» — это однокристалльные программируе-

мые многопроцессорные «системы на кристалле». Они созданы на базе IP-библиотеки (IP — Intellectual property) ядер платформы «МУЛЬТИКОР», разработанной в ГУП НПЦ «Элвис». Каждый МК содержит управляющее ядро с архитектурой RISC, совместимой с MIPS32, и одно или несколько ядер цифровой обработки сигналов (DSP) серии ELcore-xxФ. Информация о микросхемах серии сведена в таблицы 1 и 2.

Для создания конкурентоспособного продукта на базе МК «Мультикор» необходимо было также создать систему разработки ПО, позволяющую получить максимальный эффект от его возможностей. В систему разработки входят:

- компиляторы языков программирования C/C++;
- Ассемблер со встроенными макросредствами;
- редактор связей (линкер) и библиотекар;
- отладчик;
- интегрированная пользовательская среда, объединяющая перечисленные компоненты и функционирующая в двух основных режимах — разработки и отладки программ.

Ниже дано краткое описание архитектуры компонентов такой системы и их основных особенностей.

## РЕАЛИЗАЦИЯ КОМПИЛЯТОРА C/C++

Развитие систем программирования от низкоуровневых средств (машинный код, ассемблеры) до высокоуровневых (C/C++, Pascal/Delphi, Ада) показало, что для эффективного использования возможностей новых процессоров целесообразно предоставлять пользователям-разработчикам как низкоуровневые, так и высокоуровневые средства.

Эти средства разработки, выполненные в виде утилит, вызываемых из командной строки, образуют минимальный базис, необходимый разработчику для создания целевого ПО. Для повы-

Таблица 1. Характеристики сигнальных контроллеров серии «МУЛЬТИКОР»

Микросхема	1892ВМ3Т (МС-12)	1892ВМ2Т (МС-24)	1892ВМ4Я (5Я) (МС-0226/Г)	«Мультифорс» MCF-0428 <sup>1</sup>
Технология изготовления, мкм	0,25			0,18
Размер кристалла, мм	10 × 10		12,3 × 12,6	12 × 12
Степень интеграции, млн. транзисторов	~ 18		~ 26	~ 65
Корпус	PQFP 240	HSBGA 292	BGA 416	Уточняется
Многоядерная MIMD-архитектура на базе платформы «МУЛЬТИКОР»	2 ядра: RISCore32 + Elcore-14™ <sup>2</sup>	2 ядра: RISCore32 + Elcore-24™	3 ядра: RISCore32 + 2 × Elcore-26™	5 ядер: RISCore32-64FP + 4 × Elcore-28™
Рабочая частота, МГц	100	120	400	
Пиковая производительность: – 8 бит, целые – 16 бит, целые – 32 бит, плав. точка, IEEE754	1800 MOPs 800 MOPs 300 MFLOPs	3600 MOPs 1600 MOPs 600 MFLOPs	8640 MOPs 3840 MOPs 1440 MFLOPs	57,6 GOPs 25,6 GOPs 9,6 GFLOPs
Разрядность шины памяти: адрес/данные	32/32	32/64		32/64 <sup>3</sup>
Серийные образцы	2004 г.	2004 г.	2005 г.	2007–2008 г.

*Примечания.*<sup>1</sup> Предварительно.<sup>2</sup> DSP-ядра серии Elcore-xx™ из библиотеки IP-ядер платформы «МУЛЬТИКОР» (в табл. 2 отражены основные характеристики – число фаз и тактовая частота реализованных к настоящему времени DSP-ядер серии Elcore-xx).<sup>3</sup> Дополнительно в СБИС «Мультифорс» планируется: порт DDR – 64 разряда, не менее 200 МГц; контроллер PCI, 32 разряда, 66 МГц; 4 канала RapidIO Serial, 1×/4×-разрядов, 1,25 Гбод, дуплексный режим на базе международного стандарта RapidIO ISO/IEC 18372:2004 для создания однородных коммутируемых сред с пакетными передачами на базе СБИС; 2 канала SpaceWire на базе международного стандарта ECSS-E-50-12A, каждый не менее 200 МГц, работающих на расстоянии до 10 м в дуплексном режиме.

шения производительности разработки необходим также ряд дополнительных средств, в частности, текстовый редактор, система диагностики ошибок и отладчик, включающий модель (эмулятор) целевого процессора (для поддержки технологии кросс-разработки). Объединение перечисленных средств в единый комплекс позволяет получить новое качество, а именно, комплекс средств разработки программ.

**Архитектура переносимых компиляторов C/C++**

Создание компилятора для новой платформы не всегда означает его написание «с нуля». В большинстве случаев необходимо только создать генератор кода для новой платформы, адаптировать под нее существующие модули оптимизатора и/или создать новые.

Традиционная архитектура переносимых и перенацеливаемых компиляторов включает две основные подсистемы: компонент, ориентированный на конкретный входной язык и практически независимый от целевой аппаратной платформы, получил название front end (компилятор переднего плана, FE). Он производит полный синтаксический и семантический разбор программ на входном языке. Второй компонент ориентирован на целевую платформу и не зависит от входного языка; он называется back end (BE) и выполняет генерацию исполняемого целевого кода. Взаимодействие между частями компилятора осуществляется посредством некоторого промежуточного представления исходного кода, создаваемого FE и используемого BE.

В компиляторах фирмы «Интерстрон» данная архитектура подверглась существенной модернизации. Наиболее радикальная модификация заключалась в изменении роли промежуточного представления (ПП). Вместо низкоуровневой структуры, ориентированной исключительно на генерацию результирующего кода, традиционно присущей ПП, было разработано универсальное семантическое представление (СП) программ, которое содержит полное знание об исходной программе. Использование СП при генерации кода позволяет более полно учесть особенности исходной программы, что способствует повышению эффективности кода. Принципиально важное обстоятельство заключается в том, что высокоуровневое СП пригодно не только для генерации машинного кода (что составляет традиционное и практически единственное применение ПП), но и для широкого спектра операций над программами, допуская, в том числе, обратный инжиниринг и рефакторинг исходных программ (в частности, на основе методик и средств моделирования UML), статический анализ и генерацию отчетов, а также многие другие практически полезные операции, вплоть до потенциальной возможности автоматической верификации.

Вторая группа модернизаций традиционной схемы касается BE, который рассматривается как подсистема, организованная по модульному принципу. Первоначальная генерация

Таблица 2. Основные характеристики DSP-ядер серии Elcore-xx™ из библиотеки IP-ядер платформы «МУЛЬТИКОР»

DSP-ядро <sup>1</sup>	Микросхема	Число фаз конвейера	Тактовая частота, МГц <sup>2</sup>
Elcore-11™	МС-11	3	100
Elcore-14™	1892ВМ3Т (МС-12)		
Elcore-24™	1892ВМ2Т (МС-24)		
Elcore-26™	1892ВМ4Я (МС-0226), 1892ВМ5Я (МС-0226Г)	4	120
Elcore-17™	МСам-01	7	400
Elcore-18™	МС-0128		340

*Примечания.*<sup>1</sup> DSP-ядра, в номере которых присутствует первая цифра «1» (к примеру, Elcore-11, -14, -17, -18), имеют организацию SISD (Single Instructions Single Data), первая цифра «2» соответствует архитектуре 2SIMD (Single Instructions, Multiple Data), к примеру, Elcore-24, -26.<sup>2</sup> Тактовая частота для нормальных условий.

кода из СП происходит не в Ассемблер целевой платформы, а в высокоуровневый абстрактный Ассемблер. Это позволяет некоторую часть оптимизаций объектного кода производить одинаковым образом для различных целевых платформ — как существующих, так и перспективных.

Составной частью подсистемы ВЕ служит инфраструктура оптимизаций. Наличие в компиляторе такой компоненты позволяет значительно ускорить и удешевить разработку оптимизаций для перспективных аппаратных платформ — как традиционных, так и специальных оптимизаций, учитывающих особенности конкретной платформы. Эта инфраструктура основана на традиционных методиках анализа потоков управления и данных. Кроме того, она включает и сравнительно новые методы, в частности, технологию представления программы в виде единичных статических присваиваний (Static Single Assignment Forms). Это позволяет применять достаточно агрессивные по отношению к получаемому результату и в то же время консервативные по отношению к семантике программы способы оптимизации. Среди них можно отметить удаление избыточного кода (Dead Code Elimination), продвижение констант (Sparse Conditional Constant Propagation), глобальное распределение регистров (Global Register Allocation) и т.д.

### Особенности реализации компилятора C/C++ для платформы «МУЛЬТИКОР»

Основной задачей при реализации компилятора языков C/C++ для платформы «МУЛЬТИКОР» было обеспечение прозрачного взаимодействия кода и данных, относящихся к разным процессорным ядрам. Данная задача осложняется тем фактом, что размер адресуемой единицы памяти для ядер различен: 8-битный байт для RISC-ядра и 32-битное слово для DSP-ядра. Один из наиболее очевидных результатов этого — необходимость одновременной поддержки нескольких таблиц типов с возможностью переключения между ними во время компиляции одного и того же модуля. Менее очевидное, но не менее значимое следствие состоит в том, что внутренние структуры, используемые компилятором для поддержки разнообразных свойств языка времени исполнения (например, таблицы виртуальных функций, указатели на функции-члены, структуры поддержки обработчиков), также должны иметь разные размеры, в

зависимости от того, с точки зрения какого процессорного ядра мы на них смотрим.

Кроме того, организация взаимодействия ядер процессора «Мультикор» требует идентификации функций ядра DSP, вызываемых из RISC-ядра, и функций, вызываемых из самого DSP-ядра.

Указанные особенности архитектуры были поддержаны в компиляторе соответствующими директивами `pragma`. Такие конструкции являются рекомендованным стандартным способом расширения языка без модификации его семантики и потери переносимости (неизвестные директивы `pragma` компилятор может проигнорировать).

В компилятор были введены две директивы `pragma` с именами `dsp` и `dsp_root`, общий вид которых таков:

```
#pragma dsp(a, b, c)
#pragma dsp_root(x, y)
```

Посредством этих директив указываются, во-первых, глобальные переменные, которые должны располагаться в памяти DSP, и во-вторых, функции, которые должны выполняться DSP-ядром. При этом предполагается, что основной управляющий код работает на процессорном ядре RISC и, соответственно, такой код никак не помечается.

Вторая директива отражает особенность организации взаимодействия ядер, которая требует различения функций для ядра DSP, вызываемых из RISC, и вызываемых из самого DSP-ядра. Единственное отличие второй формы заключается в том, что она используется для имен функций, предназначенных для выполнения DSP-ядром, но вызываемых из кода, написанного для RISC-ядра. Имена всех остальных функций DSP-ядра, вызываемых из других функций DSP-ядра, помещаются в директиву первой формы.

### Использование директив

Директивы могут задаваться в программе на уровне пространства имен (включая глобальное пространство имен). Считается, что все сущности из данного пространства имен, упомянутые в директиве, относятся к ядру DSP. Задание в директиве некоторого идентификатора должно предшествовать объявлению или описанию переменной или функции с этим именем.

Директива `#pragma dsp_root` может задавать только имена функций (включая имена глобальных функций и функций-членов классов) и имена

классов. В последнем случае считается, что все методы класса являются `dsp_root`-методами, то есть могут вызываться только из RISC-кода.

Классовые типы могут относиться только к одному ядру. Если идентификатор `A` объявлен как DSP-идентификатор и есть класс с именем `A`, то объекты этого класса не могут быть объявлены в режиме RISC, и наоборот. Конструкции `typedef` и `enum`, объявленные как относящиеся к DSP, могут также использоваться в режиме RISC.

Следующий пример иллюстрирует использование директив:

```
#pragma dsp (A, INT, CH)
struct A
{
  int x;
  int y;
};
typedef char CH;
enum E
{
  e1, e2, e3
};
void f ()
{
  A a; // ошибка: объект DSP типа в стеке
      //RISC функции
  E e; // допустимо
  CH c; // допустимо: с имеет тип char,
      //заданный для RISC-ядра
  A *ra; // допустимо: указатель на объекты
      //типа A в DSP
}
```

В режиме DSP (в теле DSP-функции и в ее заголовке, в определении DSP-класса и его методов и статических членов) не могут использоваться сущности из RISC. В режиме RISC переменные и типы из DSP доступны.

Заметим, что макроопределения, заданные с помощью директивы `#define`, действуют как в режиме RISC-ядра, так и в режиме DSP-ядра.

### DSP-типы

Если сущности из DSP встречаются в режиме RISC-ядра, считается, что они имеют DSP-тип. Если это составной тип, то все типы, из которых он составлен, также считаются DSP-типами. Однако типы, используемые при настройке шаблонов DSP, не считаются DSP-типами, за исключением классовых типов. Например:

```
#pragma dsp (df, di, dpi, dpipi)
void df (int i, char c);
// dsp-функция имеет dsp-тип, параметры
//также имеют dsp-типы
```

```
int di; // di имеет dsp-тип int
int* dpi; // dpi — dsp-указатель на dsp-тип int
int** dppi; // dppi — dsp-указатель на
//dsp-указатель на dsp-тип int
```

Если в режиме RISC по стандарту должно быть выполнено продвижение (promotion) по отношению к выражению, имеющему арифметический DSP-тип, продвижение выполняется к RISC-типу.

### Преобразования типов

В режиме RISC ядра любой арифметический DSP-тип может быть преобразован к любому арифметическому RISC-типу, и наоборот. Указатель на продвинутый арифметический DSP-тип может быть неявно преобразован к указателю на тот же RISC-тип (если такое преобразование допускается стандартом), и наоборот. То же допустимо и для указателей на члены. Для указателей на другие типы такие преобразования не разрешены. Пример:

```
#pragma dsp (df, dc, di, dpi, dppi)
void df (int i, char c); // dsp-функция
int ri, di;
char c, dc, *rpc;
int *rpi, *dpi;
int **rpri, **dppi;
```

```
void (*pf) (int, char);
void rf () // risc-функция
{
ri = di;
di = ri;
ri = dc;
rc = dc; // допустимо
rpi = dpi; // допустимо: указатели на
//продвинутый тип
dpi = &ri; // допустимо, но возможно,
//нужно запретить (!)
rpc = &dc; // ошибка: указатели на
//непродвинутый тип
rpri = dppi; // ошибка
pf = df; // ошибка
}
```

Явные преобразования типов и операция sizeof

Типы, указанные в конструкциях явного преобразования типа в режиме RISC-ядра, считаются RISC-типами, даже если преобразование применяется к выражению, имеющему DSP-тип. Результат операции sizeof, примененной к выражению DSP-типа, считается в адресуемых единицах DSP-ядра, например:

```
#pragma dsp (dc, di, dca)
char dc;
char dca[10];
int di;
```

```
void f () // RISC-функция
{
int i;
i = sizeof (di); // i = 1
i = sizeof (dc); // i = 1
i = sizeof (+di); // i = 4, т.к. произошло
//продвижение к risc-типу
*((char *)dca) = 1; // dca преобразуется к
//указателю на risc char
}
```

Созданный полный компилятор C/C++, с одной стороны, соответствует международным стандартам, а с другой — использует особенности архитектуры платформы «МУЛЬТИКОР».

Как показывают предварительные исследования, эффективность оптимизации исходной программы составляет 2—2,9, а значит, в столько же раз ускоряются вычисления и сокращается объем памяти, занимаемый исполнимым файлом.

*В работе над проектом активное участие принимали: Е.А. Зуев, М.Ю. Донорский, А.А. Чупринов, Ю.Н. Александров, В.Ф. Никольский, Т.В. Солохина, Я.Я. Петричквич, Беляев А.А., Глушков А.В.*

**Окончание следует.**