УДК 629.735.33

Н. В. Бухаренко – студент кафедры информационных систем
А. Ю. Сыщиков – научный руководитель

КОДОГЕНЕРАТОР ПАРАЛЛЕЛЬНОГО АССЕМБЛЕРНОГО КОДА ДЛЯ ЦИФРОВОГО СИГНАЛЬНОГО ПРОЦЕССОРА

Процессоры для цифровой обработки сигналов активно применяются в вычислительных системах. Для обеспечения высокой производительности в составе таких систем применяются DSP-ядра, которые имеют специализированную архитектуру, обеспечивающую высокую производительность при решении ограниченного круга ресурсоемких задач. Для современных процессоров для написания и отладки программ кроме ассемблера в основном используют такие языки программирования как Си, Си++. Для получения наиболее эффективного кода приходится писать код исключительно на низкоуровневом процессорном ассемблере. Задача генерации эффективного ассемблерного кода, активно использующего параллелизм и другие особенности DSP-ядра, является целью данной работы. Сигнальные процессоры имеют схожие архитектуры, это значит, что разрабатываемый подход можно применить для широкого класса сигнальных процессоров. В данной работе в качестве экспериментальной платформы используется микропроцессор МС-24 [1].

[http://multicore.ru/index.php?id=47].

Исходная программа представлена в виде схемы на языке VPL [2, 3], в ней описываются объекты и связи между ними (граф объектов), фактически – data-flow-подобная схема. В схеме описана вычислительная операция компонента (объект схемы), её свойства, параметры и связи с другими объектами.

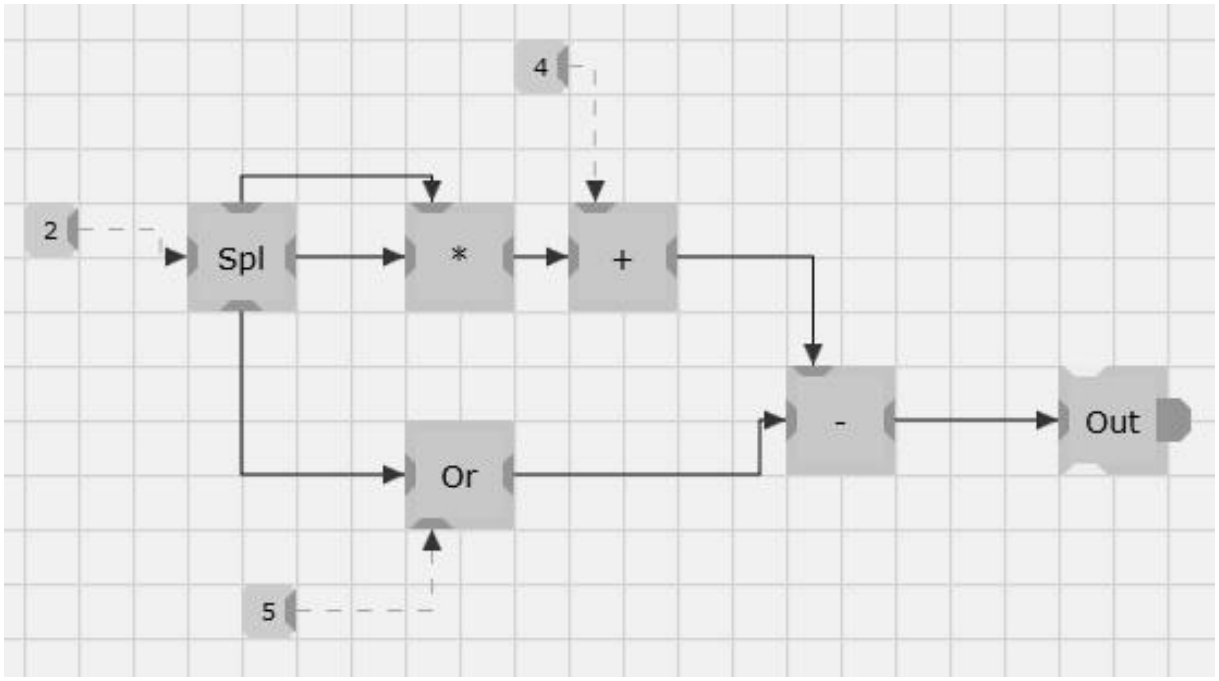

Рис. 1 Пример входной схемы программы

На приведённом примере (рис. 1) представлены следующие объекты языка:

- серые малые квадраты – константы;
- spl – «размножитель» данных;
- “*” является объектом, вычислительная операция которого умножение;

- “Or” – объект, вычислительная операция которого логическое ИЛИ;
- “+” является сумматором;
- “-” является объектом, вычислительная операция которого вычитание;
- “Out” – оператор записи в память.

Стрелками на схеме отмечены связи по данным между компонентами. На первом этапе разбирается схема и создаётся связанный граф, в котором содержится вся информация об объектах, их связях, свойствах и параметрах. Строится расписание выполнения всех объектов в соответствии со временем их выполнения и зависимостям по данным. На следующем этапе определяется возможность параллельности выполнения операций в зависимости от возможности потоков ядра и, если это необходимо, перестраивается расписание. Каждому компоненту сопоставляется соответствующий ему шаблон кода. На следующем шаге в шаблон подставляются необходимые регистры из таблицы регистров. В случае необходимости генерируется код загрузки данных из памяти в регистры или выгрузки из регистров в память. В результате, на выходе получаем рабочую программу для выбранного ядра.

Всем типам элементов схемы программы формируется представление в терминах ассемблера вычислительного ядра. Для этого используются шаблоны, содержащие вычислительный код и параметры, формируемые кодогенератором. Поскольку у каждого из ядер своя система команд, то для каждого из них необходимо разработать свой шаблон. Более того, в рассматриваемом процессоре MC-24 в DSP-ядре формат команды зависит от операционного устройства (ALU), на котором будет выполняться команда. Поэтому в кодогенераторе предусмотрена возможность задавать отдельный шаблон для различных операционных устройств внутри ядра.

Пример шаблона операции вычитания для RISC-ядра:

```
entity llsub [risc] is
        sub &regout1,&reginx1,&reginy1
entity llsub end.
```

Пример шаблона аналогичной операции для первого ALU DSP-ядра:

```
entity llsub [dsp1] is
        sub &reginx1,&reginy1,&regout1
entity llsub end.
```

Служебное слово «entity» предназначено для обозначения шаблона в текстовом файле. Последующее слово – идентификатор, обозначает операцию в схеме программы. Параллельное выполнение двух вычислительных операций DSP-ядра возможно только при условии, что они выполняются на разных операционных устройствах (ОУ). Например, не могут одновременно исполняться операции AND и OR, так как они обе должны исполняться при помощи одного и того же логического устройства LU. Вычислительные команды в зависимости от исполняющего их операционного устройства можно разделить на два типа – OP1 и OP2. Только принадлежность к различным группам дает возможность двум операциям исполняться одновременно. К первому типу (OP1) относятся операции, исполняемые при помощи арифметического или логического устройства (AU, LU), ко второму типу (OP2) – операции, исполняемые при помощи умножителя-сдвигателя MS. В шаблоне цифра «1» обозначает принадлежность к первому типу (OP1), цифра «2» соответственно ко второму типу (OP2). Слово «sub» означает команду ассемблера, для которой сделан шаблон. «&reginx1» обозначает, что параметром является входной регистр с тэгом «x» и с порядковым номером «1». «&reginy1» обозначает то же самое за исключением тэга, вместо «x» здесь «y». Тэг необходим для корректного связывания двух компонентов, двух операций. «&regout1» является параметром для выходного регистра с пустым тэгом и с порядковым номером «1». Шаблон не обязательно должен состоять только из одной команды ассемблера, шаблон может содержать код любого размера.

RISC-ядро микропроцессора не поддерживает параллельное выполнение операций в инструкциях, возможность распараллеливания обработки данных есть только у DSP-ядра. Параллелизм DSP-ядра обеспечивается за счет наличия нескольких операционных устройств (ALU). Распараллеливание возможно если две операции, не связанные по данным и имеют разные типы. Если хватает регистров общего назначения для выполнения двух операций, и они могут выполняться параллельно, происходит

генерация параллельной команды. В противном же случае, распараллеливания не происходит, и команды выполняются последовательно. В DSP-ядре процессора МС-24 задание параллельно выполняющихся команд осуществляется за счет использования VLIW-инструкций, которые могут вмещать до трех одновременно выполняющихся команд.

Пример инструкции для выполнения двух операций одновременно:

```
mpss R2,R4,R10    or R8,R6,R12
```

Пример последовательного выполнения тех же двух операций:

```
mpss R2,R4,R10
or R8,R6,R2
```

Команда «mpss» выполняет умножение двух целых чисел в формате «short» и принадлежит к типу OP2. Команда «or» предназначена для выполнения логического ИЛИ в формате «short», принадлежит к группе OP1. В первом случае регистров хватает, и флаг параллельности находится в состоянии «1», в то время как во втором случае, регистров так же хватает, но флаг параллельности опущен в состояние «0», следовательно, параллельность не реализуется. Система инструкций и гибкие адресные режимы DSP-ядра позволяют эффективно реализовать алгоритмы сигнальной обработки.

Для работы каждого ядра микропроцессора, RISC и DSP имеют набор регистров общего назначения. Исходные данные и результаты всех операций ALU хранятся в регистрах общего назначения. В кодогенераторе создаётся виртуальная таблица регистров с заданным количеством ячеек (допустим, 8), то есть в начале работы кодогенератора имеем 8 свободных регистров для обеспечения подготовки данных для команд (рис. 2). В процессе работы кодогенератора таблица постепенно заполняется и возникает необходимость в свободных регистрах. После того, как выдали компоненту все необходимые регистры и сгенерировали очередную команду, очищаем входные регистры. Когда все регистры в таблице помечены как несвободные, то заданным алгоритмом выборки получаем любой регистр, который не используется в данный момент, сохраняем его значение в память и помечаем свободным. Возможны различные алгоритмы выборки регистров из таблицы, в данном кодогенераторе реализован случайный выбор. Каждый регистр содержит в себе порядковый номер, флаг занятости в данном моменте, флаг возможности освобождения и идентификатор связи (линка) с другим компонентом.

Value:	2	3	4	5
Busy:	False	False	False	False
Link:				
Now:	False	False	False	False

Рис. 2. Пример таблицы регистров в начале работы генератора

Поле «Value» показывает порядковый номер регистра в таблице, флаг «Busy» является флагом, который показывает, можно ли освобождать регистры, если «False», то регистр можно освободить. Поле «Link» показывает, какой идентификационный номер у связующего звена между двумя компонентами. Флаг «Now» служит для отображения состояния занятости регистра для текущего компонента. Поле «Value» не изменяется на протяжении всей работы генератора. Работа программы невозможна, если во входных параметрах генератора указать количество регистров меньше чем количество регистров необходимое для выполнения каждой из операций. Другими словами, если указать во входных параметрах количество регистров 2, а какой-то операции необходимо 3, то работа генератора невозможна.

Все объекты в схеме связаны между собой посредством линков. Параметры, свойства и сам линк описаны в схеме (рис. 3).

```
<links>
  <link id="i240" comment="" tag="" subtype="read" source="i200" tar
  <link id="i241" comment="" tag="" subtype="read" source="i203" tar
  <link id="i252" comment="" tag="" subtype="read-erase" source="i23
```

Рис. 3. Пример промежуточного представления схемы программы, где определяются связи

В связанном графе, полученном после разбора схемы, содержится вся необходимая информация об обмене данными. Линку сопоставляется адрес во внутренней памяти заданного процессора (будь то RISC или DSP). В каждый регистр записывается идентификационный номер(ID) линка, чтобы знать для какого входного или выходного значения был выдан регистр из таблицы. Если на протяжении работы генератора необходимо сохранить какое-либо значение содержащиеся в регистре, то значение пишется по адресу указанному в линке. Если необходимо получить значение, то выгружаем его из памяти.

Используя описанные подходы к построению кодогенератора, можно получить работающие программы для RISC-ядра на ассемблере и для DSP-ядра как на параллельном, так и на последовательном ассемблере. В приведённой ниже таблице (таблица) показан генерируемый код и его результат выполнения для двух ядер микропроцессора по схеме объектов(рис. 1).

Таблица

Программные реализации работы схемы

<table>
<tr><th>Программа для RISC-ядра</th><th>Программа для DSP-ядра</th></tr>
<tr><td>li $2,0x2
li $3,0x5
li $4,0x4
sw $2,0x80000014
or $5,$2,$3
mul $3,$2,$2
add $2,$3,$4
sub $3,$5,$2
sw $3,0x80000990</td><td>move 0x2,R2
move 0x5,R4
move 0x4,R6
move 0x00000014,A2
move R2,(A2)
mpss R2,R2,R8 or R2,R4,R10
add R8,R6,R4
sub R10,R4,R6
move 0x00000044,A2
move R6,(A2)</td></tr>
<tr><td colspan="2">Результаты работы программ</td></tr>
<tr><td>8-7=1</td><td>8-7=1</td></tr>
<tr><td>Результат пишется в регистр $3
RISC Registers
0 zero 00000000
3 v1 00000001
6 a2 00000000
9 t1 00000000
12 t4 00000000
Затем в память по адресу 0x80000990
80000990 01 00 00
800009A1 00 00 00</td><td>Результат пишется в регистр R6
R04.L: R05 R04 0000 0008
R06.L: R07 R06 0000 0001
Затем в память по адресу 0x00000044
00000040 00000000
00000044 00000001
00000048 00000000</td></tr>
</table>

Разработанный подход позволяет упростить создание тяжёлых программных конструкций путём генерации параллельного ассемблерного кода вместо сложного ручного программирования параллельных ассемблерных команд. В качестве целевой платформы при разработке использовался процессор МС-24 на базе платформы «МУЛЬТИКОР». В виду отсутствия компилятора языка C/C++, не представляется возможным проверить, насколько генерируемый код более оптимизирован, чем получаемый код при трансляции.

Библиографический список

1. http://multicore.ru/index.php?id=47
2. *Сыщиков А.Ю.* Технология параллельного программирования неоднородных систем на кристалле. Научная сессия ГУАП: Сб. докл.: В 3 ч. Ч. I. Технические науки/ ГУАП. СПб., 2008, с.133-139.
3. *Ivanov V., Sheynin Y, Syschikov A.* Programming model for coarse-grained distributed heterogeneous architecture (Модель программирования для среднегранулярных распределенных гетерогенных архитектур).

4. Труды XI симпозиума по проблеме избыточности в информационных системах. Под редакцией проф. *Крука Е.А.* – СПб.: ГУАП, 2007, с.246-250.