Интегрированная среда описания системы команд встраиваемых процессоров

В.В. Рубанов, А.С. Михеев

Аннотация. В статье рассматривается интегрированная среда MetaDSP для описания системы команд встраиваемых процессоров. Такое описание включает спецификацию синтаксиса и поведения команд процессора и позволяет автоматически настроить набор кросс-инструментария разработки (ассемблер, дисассемблер, симулятор, отладчик), а также сгенерировать документацию прикладного программиста для целевого процессора. Эти возможности позволяют использовать MetaDSP и соответствующий настраиваемый кросс-инструментарий на этапе дизайна аппаратуры для прототипирования встраиваемых процессоров.

1. Введение

В последнее время за счет развития технологий и удешевления производства, появляется огромное количество так называемых встраиваемых процессоров и их модификаций. К таким процессорам, в частности, относят цифровые процессоры обработки сигнала (DSP) и микроконтроллеры. Встраиваемые процессоры характеризуются ориентированностью на оптимальное (с точки зрения стоимости, размера кристалла, производительности и энергопотребления) решение узкого класса конкретных задач. В процессе создания таких решений важным этапом является прототипирование возможных альтернатив дизайна (Design Space Exploration – DSE). Для этого важно уметь получать оценки эффективности реализации конкретных алгоритмов и программ для возможных модификаций предполагаемого процессора. Одним из методов решения этой задачи является использование настраиваемых кросс-средств разработки, которые позволяют получать искомые оценки с помощью симуляции и профилирования тестовых программ для целевого прототипа с использованием инструментальной машины [11]-[12].

Для настройки инструментальных средств (прежде всего, ассемблера и симулятора/профилировщика) на конкретный целевой процессор необходимо уметь эффективно и формально задавать спецификацию этого процессора. Заметим, что для построения ассемблера и симулятора нет необходимости задавать детали реализации процессора на низком уровне, как это делается в синтезируемых описаниях на Verilog или VHDL. Для целей проверки

функциональной корректности тестовой программы и ее временных характеристик достаточно описать систему команд процессора (синтаксис, поведение и свойства команд, а также регистры и подсистему памяти).

В данной работе рассматривается интегрированная среда MetaDSP, которая позволяет визуально описывать систему команд встраиваемых процессоров. Необходимость разработки интегрированной среды с графическим интерфейсом обусловлена требованиями возможности быстрого внесения согласованных изменений и автоматической верификации описаний «на лету». Ввиду наличия во встраиваемых процессорах, как правило, нерегулярной системы команд с множеством различных форматов, внесение изменений напрямую в текст спецификации чревато большим количеством ошибок. Среда MetaDSP позволяет упростить внесение изменений, обеспечивая при этом средства контроля над согласованностью изменений в ассемблере, дисассемблере, симуляторе и документации пользователя.

Статья состоит из введения и пяти разделов. В первом разделе представляется общая архитектура системы MetaDSP. Второй и третий разделы характеризуют возможности системы MetaDSP по описанию синтаксиса/бинарного кодирования и поведения команд соответственно. В четвертом разделе дается описание концепции иерархического описания системы команд и наследования кода операции, операндов и поведения. Пятый раздел содержит описание основных элементов интерфейса пользователя системы MetaDSP.

2. Архитектура системы MetaDSP

Система MetaDSP предназначена для автоматизации описания системы команд встраиваемых процессоров с целью обеспечения последующей автоматической настройки ассемблера, дисассемблера, симулятора, отладчика и генерации документации прикладного программиста. При таком подходе вся необходимая информация содержится в одном месте, а именно в файле описания MetaDSP, что позволяет поддерживать целостность и согласованность этого описания для использования в процессе настройки соответствующих кросс-инструментов. При этом уменьшается количество ошибок, ускоряется процесс внесения изменений и повышается наглядность описания. Эти характеристики очень важны для использования рассматриваемой системы в процессе прототипирования встраиваемых процессоров.

Общая схема форматов описания MetaDSP и их использование для настройки соответствующих кросс-инструментов показаны на Рис. 1.

Из файла описания MetaDSP генерируется файл спецификации синтаксиса команд и бинарного кодирования на языке ISE [14], который используется для настройки ассемблера, дисассемблера и отладчика, а также декодера симулятора. Основная часть симулятора процессора генерируется из файлов на языке ISE-Exec, в которых определяется поведение каждой инструкции.

Документация прикладного программиста для целевой системы команд генерируется из описания MetaDSP в виде документа MS Word.

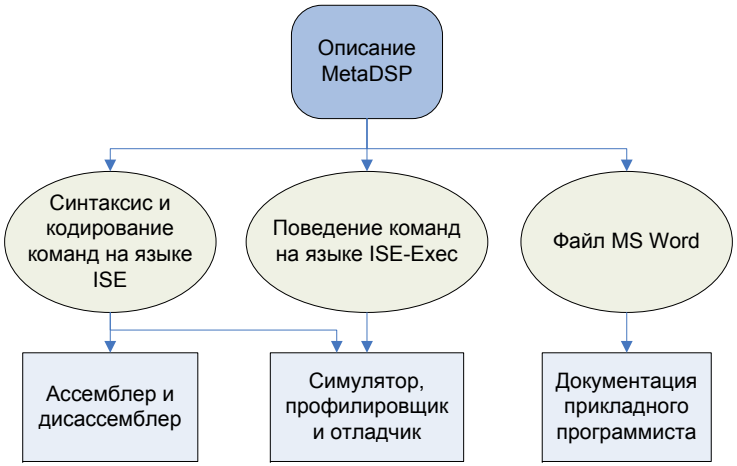

Рис. 1. Форматы описаний в системе MetaDSP и их использование

Таким образом, для различных целей (настройки различных инструментов и генерации документации) требуется свое подмножество общего описания системы команд; при этом эти подмножества имеют различные пересечения. Например, информация о бинарном кодировании команд используется для настройки всех инструментов и для генерации документации. Информация о поведении команд используется только для настройки симулятора и профилировщика, а информация об ассемблерном синтаксисе инструкций – для настройки ассемблера, дисассемблера, отладчика и генерации документации. Использование среды MetaDSP позволяет согласованно вносить изменения в различные части описания системы команд и корректно учитывать эти изменения при настройке всех инструментов.

3. Описание синтаксиса и бинарного кодирования команд

В этом разделе описываются возможности системы MetaDSP для спецификации бинарного кодирования инструкции, ее ассемблерного синтаксиса, статических свойств и ограничений, а также правил определения межкомандных конфликтов.

3.1. Бинарное кодирование инструкции

Бинарное кодирование команды (формат) задается в виде строки вида:

```
00YY-0000-SSAA-XXXX
```


Дефисы в этой строке не являются значимыми и используются для косметического разграничения групп битов. Символы 0 и 1 ставятся на местах, которые вместе образуют так называемый код операции (КОП). Символ X означает, что данная битовая позиция не используется (может быть как 0, так и 1). Другие символы обозначают, что на данных местах располагаются коды операндов (в данном примере заданы операнды YY, SS и AA).

3.2. Ассемблерный синтаксис инструкции

Синтаксис команды задается строковым шаблоном, в котором можно использовать комбинацию статических символов и ссылок на операнды, заключенных в фигурные скобки. Ссылки на операнды должны быть разделены статическим текстом, пробелы не являются значимыми:

```
IF {cond} XOR {GRs}, {GRt}
```

Здесь IF, XOR и запятые задают статическую часть синтаксиса команды, а cond, GRs и GRt являются ссылками на операнды. Синтаксис операндов определяется их типом, который задается отдельно для каждого операнда.

3.3. Операнды инструкции

Для каждой команды описывается ноль или более операндов. При этом для каждого операнда задаются следующие параметры:

- имя операнда, определяющее ссылку на операнд в строке синтаксиса команды (например, GRs);
- тип операнда (например, регистр общего назначения, адресный регистр, 8-битная константа и так далее). Типы операндов задаются отдельно и определяют семантику в виде ссылки на объект в состоянии симулятора, синтаксис, разрядность и кодирование в виде перечисления пар значений – {синтаксис=код}. Например, {{GR0=00}, {GR1=01}}. Заметим, что тип операнда не определяет место кодирования операнда в машинном слове;
- место кодирования операнда в бинарном коде инструкции в виде строки, в которой буквы, отличные от X, определяют положение кода операнда (наиболее значимый бит слева). Например, XXXX-XXXX-XXXX-XXAA определяет положение двухбитового операнда AA в двух младших битах машинного кода команды. Коды различных операндов в принципе могут перекрываться, если это не вызывает конфликта, кроме того, код операнда может иметь разрывы, например, четырехбитный операнд AAAA может кодироваться, как XXXX-XXXX-XAXA-XXAA – такие возможности очень важны для описания нерегулярных систем команд.

3.4. Ограничения на значения и связи операндов

В ряде случаев для данной инструкции допустимыми являются не все возможные значения операнда данного типа. Например, если в данной инструкции в качестве операнда `GRt` могут использоваться только регистры общего назначения `GR0`, `GR1`, `GR2`, `GR3`, а не все 16 регистров, имеющиеся в процессоре, то необходимо отдельно описать ограничения на такой операнд. Ограничения можно задать в виде предикатов. Если предикат ложный, то ограничение не выполнено, и ассемблер выдаст сообщение об ошибке. Например, для операнда `GRt` ограничение можно задать так:

```
Predicate : GRt <= 4
Message   : Only GR0 - GR3 can be used for this operand
```

В предикатах можно использовать логические и арифметические операции над одним или несколькими операндами. Например, `(GRt/4+1)*2-GRt >=GRt*2` представляет собой пример допустимого предиката. Использование предикатов над несколькими операндами позволяет задать ограничения на связи операндов, например, `GRs=GRt` обозначает тождественность операндов в синтаксисе.

3.5. Межкомандные конфликты

Каждой команде можно назначить некоторый *набор свойств* и задать для них значения и *области активации*. В качестве значения свойства может выступать либо константа, либо значение одного из операндов команды. Область активации задает диапазон соседних команд, на котором данное свойство активно. По умолчанию [1;1] область активации затрагивает только текущую команду.

Пример:

```
MAC {acr},{grs},{grt}
  [read_grn:grs, read_grn:grt, write_acr:acr:2;2]
```

Данная команда обладает следующими свойствами:

- `read_grn` – двойное свойство со значениями, равными значениям операндов `grs` и `grt`. Область активации по умолчанию затрагивает только текущую команду (здесь это означает, что значения регистров, заданных операндами `grs` и `grt`, читаются на первом такте);
- `write_acr` – значение свойства равно значению операнда `acr`. Область активации [2;2] затрагивает следующую команду (здесь это означает, что значение `acr` будет записано на втором такте).

На механизме описания свойств базируется способ задания ограничений на использование ресурсов. Задается список *предикатов совместимости свойств*. В предикате совместимости свойств в квадратных скобках

указывается набор пар свойств (пары разделяются запятыми, свойства в паре – знаком «=»). Предикат истинен для пары команд, когда выполняются следующие условия: первая команда обладает всеми свойствами из левых частей пар, вторая обладает всеми свойствами из правых частей пар; при этом для каждой пары значения свойств совпадают на пересечении их областей активации. Предикаты совместимости оцениваются ассемблером для всех пар команд при ассемблировании программы - так обнаруживаются конфликтующие команды. Заметим, что оценки работают гарантированно корректно только на линейных участках.

Пример:

```
[write_acr=read_acr] % warning: "WAR conflict for
ACRs"
```

Данный предикат будет верен, если у пары команд значение свойства `write_acr` первой команды совпадет со значением свойства `read_acr` второй команды на пересечении областей активации этих свойств. В данном примере это отражает конфликт по данным (по аккумуляторным регистрам) типа WRITE AFTER READ.

Зарезервировано специальное свойство «any», которым по умолчанию обладает любая инструкция. `[any=X]` дает истинный предикат, если вторая инструкция обладает свойством X (независимо от его значения).

3.6. Дополнительная информация об инструкции

Дополнительно для каждой команды можно указать:

- идентификатор команды;
- текстовое описание на естественном языке;
- указание на то, что инструкция состоит из нескольких параллельно выполняющихся частей;
- различные метрики инструкции, такие как количество тактов, которое занимает выполнение данной инструкции, энергия, потребляемая процессором при выполнении данной инструкции, и т.п. Эти данные используются профилировщиком.

4. Описание поведения команд

В этом разделе рассматриваются средства системы MetaDSP для описания поведения инструкций. Для этой цели используется язык ISE-Exec, являющийся расширением C++. Описание на этом языке проходит через генератор, который преобразует его в чистый C++. Цель разработки этого языка состоит в том, чтобы повысить уровень абстракции для учета специфических потребностей и более удобного описания поведения команд процессора.

4.1. Обращение к операндам и битовым полям

При описании поведения инструкции необходимо знать конкретные значения операндов инструкции. В рассматриваемом языке эти значения можно получить двумя путями.

- Путем использования просто имени операнда, как обычной переменной. Такая ссылка автоматически преобразуется в необходимый код, который извлечет нужное значение из машинного кода команды в процессе симуляции.
- Путем непосредственного обращения к именованным битовым полям машинного слова. Это осуществляется с помощью использования синтаксиса:

  `#[+]LLLL`

Здесь символ + является не обязательным; если он есть, то закодированное число нужно трактовать как знаковое. На месте LLLL может быть любое количество подряд идущих букв в бинарном коде инструкции. При задании этих букв нужно следить, чтобы их положение в бинарном шаблоне команды определялось однозначно (см. 3.1).

Вот простой пример обращения к битовым полям инструкции:

```
MOVE {XM0}({YAv} + {offset}), {GRs}
```

с кодом:

```
00DX-0100-01AA-RRRR-MMMM-MMMM
```

Данная инструкция осуществляет пересылку в память с именем XM0 (это может быть память данных DM0 или TM0) регистра общего назначения GRs (в качестве GRs может выступать один из регистров общего назначения GR0,GR1,…, GRF). Адрес является суммой явно заданной константы offset и значения адресного регистра YAv (это может быть один из адресных регистров DA0 или TA0).

Здесь кодирование соответствует операндам следующим образом:

```
D        : Destination memory field (XM0)
AA       : Destination address pointer field (YAv):
RRRR     : General register field (GRs):
MMMMMMMM : offset field constant [-128;127]
```

При описании поведения возможны следующие выражения:

```
GRs, #RRRR, YAv + #+MMMMMMMM
```

4.2. Обращение к состоянию процессора и системы

Состояние процессора и системы задается значением регистров и всех памятей системы. Обращение к этим ресурсам происходит следующим образом.

- Обращения к памяти осуществляются с помощью макроса:

  `MEM(memory_name, address)`

  Здесь memory_name обозначает индекс памяти, а address – адрес, по которому идет обращение.
- Обращение к регистрам производится путем использования макросов вида:

  `<register_type>(register_name)`

  Здесь register_type – тип регистра, а register_name – имя или код регистра этого типа.

Например, для описанной выше инструкции `MOVE XM0(YAv + offset), GRs` поведение можно задать так:

```
MEM(XM0, ARN(YAv)+offset) := GRN(GRs);
```

Здесь `YAv` – адресный регистр типа `ARN`, а `GRs` – регистр общего назначения типа `GRN`.

Стоит обратить внимание на операцию «`:=`», использованную в данном примере. Эта операция необходима, если в левой или правой части присваивания используется ресурс системы (регистр или память). При этом обращение транслируется в соответствующие функции доступа (set/get), которые кроме собственно чтения/записи значения обеспечивают сбор необходимых статистик для профилировки.

Допустимые типы памятей и регистров, их разрядность и размер задаются разработчиком общей части симулятора.

4.3. Микрооперации в описании поведения

При описании поведения инструкции, помимо стандартных операций C++, можно использовать описываемые дополнительно микрооперации (например REVERSEBITS(REG)). Стандартные микрооперации входят в библиотеку поддержки; разработчик общей части симулятора может определить свои микрооперации.

4.4. Временные переменные в описании поведения

При описании поведения инструкции можно использовать любое число временных переменных. При этом для удобства эти переменные можно не объявлять, а можно просто написать имя переменной в левой части операции ‘=’. После обработки генератором все определения переменных будут сгенерированы автоматически.

5. Иерархичность описания команд в системе MetaDSP

В среде MetaDSP система команд описывается иерархично в виде дерева. Листьями дерева являются команды, промежуточные узлы соответствуют группам команд (группы могут быть вложенными). Например, инструкции пересылки можно разместить в одной группе дерева, арифметические в другой, инструкции управления в третьей. При этом инструкции пересылки можно

далее дробить на более мелкие группы: пересылки память-память, регистр-память и т. д.

Иерархичность описания системы команд позволяет использовать наследование определенных свойств от родительских групп. Наследование возможно для следующих элементов: код операции (КОП), отдельные операнды и поведение. Наследование этих свойств может выполняться независимо и позволяет автоматизировать изменение групп инструкций, так как изменения у родителя вызывают аналогичные изменения у всех его потомков.

5.1. Наследование кода операции

Под наследованием кода операции (КОП) подразумевается наследование битовых полей, установленных в бинарном коде инструкции в фиксированные значения – ‘0’ или ‘1’. Например, пусть есть две инструкции:

```
MOVE GRs, GRt  с кодом 0111-0011-GGGG-RRRR
```

и

```
MOVE ARs, GRt  с кодом 0011-0011-AAAA-RRRR.
```

Видно, что в этих двух инструкциях общим КОПом является:

```
0X11-0011-XXXX-XXXX
```

В таких случаях в системе MetaDSP обычной практикой является вынесение описания общего КОП в родительский узел. При этом для каждого потомка этого узла можно указать, наследует ли он КОП родителя или нет (по умолчанию – да). Для узлов, наследующих КОП родителя, изменение значения зафиксированных в родителе битовых позиций запрещено, то есть в потомке можно определять только те позиции шаблона, в которых в родителе стоит X. Наследование КОП может быть многоуровневым в соответствии с организацией дерева системы команд.

При добавлении новых потомков к родителю унаследованные биты КОП автоматически проставляются в потомке. Это ускоряет добавление новых инструкций.

5.2. Наследование операндов

Под наследованием операндов подразумевается использование в инструкции родительских операндов. При этом изменение таких операндов возможно только на уровне родителя. В качестве примера возьмем те же инструкции, рассмотренные при описании наследования кода операции:

```
MOVE GRs, GRt  с кодом 0111-0011-GGGG-RRRR,
```

где операнды `GRs, GRt` кодируются битами, помеченными в бинарном коде инструкции буквами `GGGG` и `RRRR` соответственно, и

```
MOVE ARs, GRt  с кодом 0011-0011-AAAA-RRRR,
```

где операнды `ARs, GRt` кодируются битами, помеченными в бинарном коде инструкции соответственно буквами `AAAA` и `RRRR`.

Видно, что вторые операнды этих двух инструкций `GRt` имеют одинаковый тип и одинаковое положение в бинарном коде. Таким образом, если в дереве инструкций добавить родительский узел с определенным в нем операндом:

```
Имя:          GRt
Тип:          General Purpose Register
Кодирование:      XXXX-XXXX-XXXX-RRRR,
```

а затем добавить к этому родительскому узлу двух потомков:

`MOVE GRs, GRt` с операндами:

```
Имя:          GRs
      Унаследован:нет
Тип:          General Purpose Register
Кодирование:      XXXX-XXXX-GGGG-XXXX,

Имя:          GRt
      Унаследован:да
```

и

`MOVE ARs, GRt` с операндами:

```
Имя:          ARs
      Унаследован:нет
Тип:          Address Register
Кодирование:      XXXX-XXXX-AAAA-XXXX,
Имя:          GRt
      Унаследован:да
```

то эти потомки будут содержать унаследованный операнд GRt. При изменении каких либо свойств этого операнда в родительском узле соответствующие изменения автоматически применятся к потомкам. Опыт авторов показывает, что в условиях должной организации дерева команд при добавлении новой инструкции большая часть операндов наследуется от родительского узла.

Наследовать можно любое количество операндов, по-разному расположенных в бинарном коде родителя, в том числе и с перекрытием. При удалении операнда у родителя соответствующий унаследованный операнд удаляется у всех потомков.

5.3. Наследование поведения инструкций

Поведение инструкции задается в виде кода на языке ISE-Exec, описанном в разделе 4. Под наследованием поведения подразумевается использование в описании поведения команды определенных частей родительского описания. Важность этого механизма обусловлена тем, что при описании поведения часто возникают ситуации, когда у нескольких дочерних инструкций описания очень похожи между собой. Чаще всего похожи начала или концы этих описаний (например, загрузка и выгрузка данных), а в середине они различны (обработка данных, специфичная для каждой команды). Поэтому в системе

MetaDSP существует возможность определить общие части поведения в родительском узле и наследовать их в потомках.

Описание поведения у каждого узла дерева системы команд разделяется на две части: начало и конец. При этом в описании, принадлежащем данному узлу, можно использовать операнды, видимые на уровне этого узла. Дочерние узлы могут наследовать начало и конец описания родителя, то есть начало описания дочернего узла является конкатенацией начала описания родителя и своего начала, а конец описания дочернего узла является конкатенацией своего конца и конца описания родителя (матрешка).

При таком подходе общий код в потомках большей частью может быть вынесен на уровень родителя, что обеспечивает более удобное и быстрое внесение изменений, а также уменьшает количество ошибок.

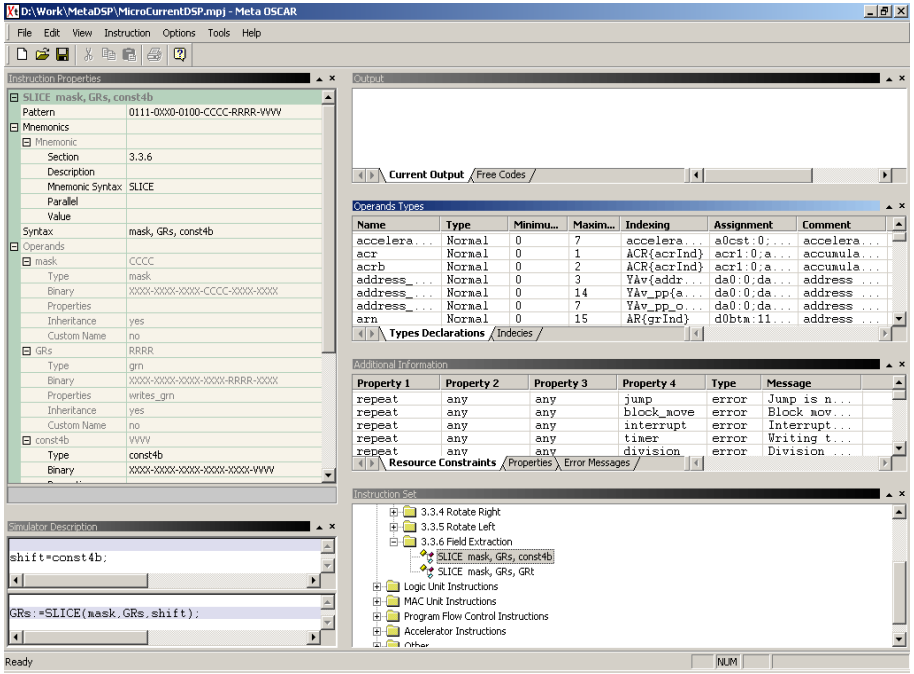

Рис. 2. Графический интерфейс системы MetaDSP.

6. Интерфейс системы MetaDSP

Визуальная часть среды MetaDSP реализована на C++ с использованием графического интерфейса Windows и элементов управления библиотеки Codejock Xtreme Toolkit. Программа использует многооконный интерфейс с плавающими окнами (Рис. 2).

Основные окна программы:

Instruction Set – это окно (Рис. 3) отображает дерево инструкций и является основным элементом навигации. При выделении определенного узла дерева информация об этом узле автоматически отображается в других окнах. Узлы дерева имеют контекстное меню, позволяющее удалять, копировать и добавлять новые узлы:

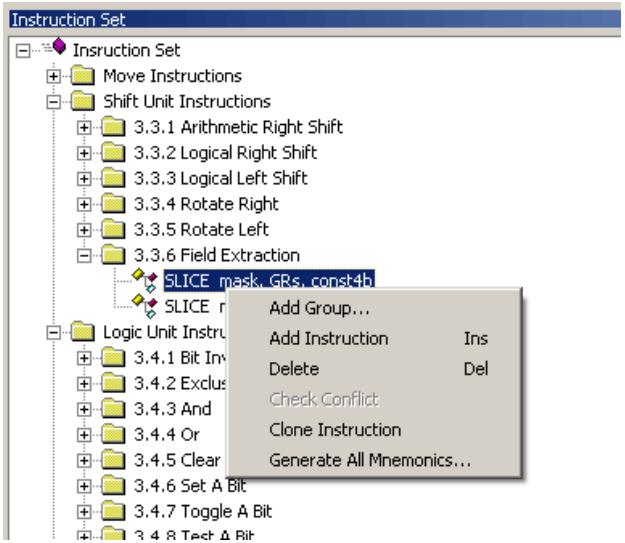

Рис. 3. Окно Instruction Set.

Instruction Properties – это окно (Рис. 4, 5) отображает и позволяет редактировать свойства узлов, в частности бинарное кодирование, синтаксис, операнды и ограничения:

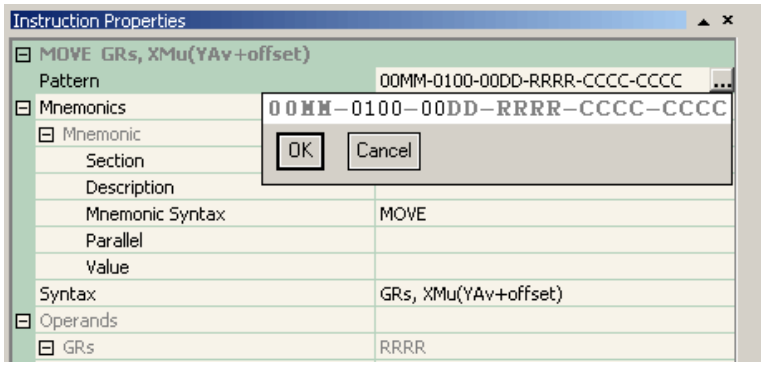

Рис. 4. Задание шаблона бинарного кодирования в окне Instruction Properties.

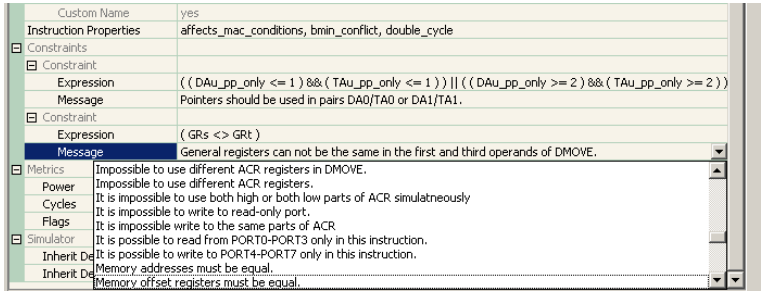

Рис. 5. Задание ограничений в окне Instruction Properties.

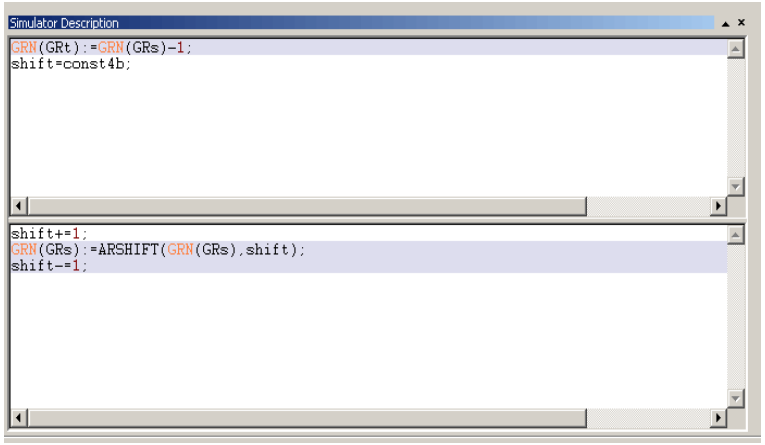

Рис. 6. Окно поведения команды.

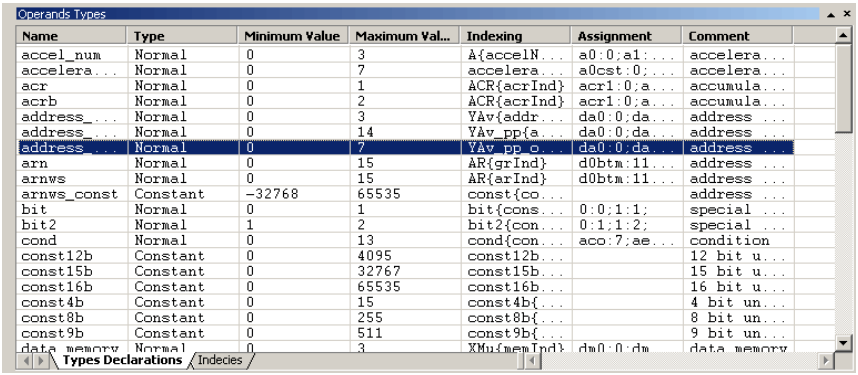

Рис. 7. Окно типов операндов.

Simulator Description – в этом окне (Рис. 6) описывается поведение команды. Окно разделено на две части, каждая из которых разделяется на собственную

зону и зону, унаследованную от родителя (темный фон). Текст в унаследованных зонах заблокирован для редактирования (он задается в родителе - см. 5.3).

Operand Types – в этом окне (Рис. 7) описываются глобальные типы операндов, которые потом используются при задании операндов в командах:

Добавление, удаление и изменение типов осуществляется с помощью контекстного меню (Рис. 8).

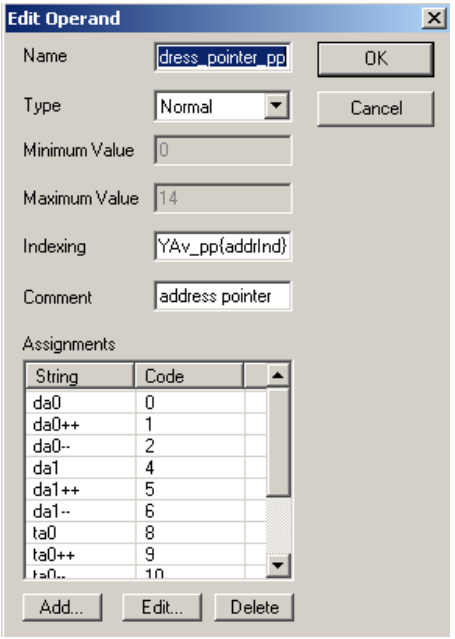

Рис. 8. Диалог редактирования типов операндов.

Additional Information – в этом окне задается дополнительная информация для системы команд в целом: описание межкомандных конфликтов, список свойств команд, список сообщений об ошибках. Свойства команд и сообщения об ошибках используются при задании ограничений (как внутрикомандных, так и межкомандных).

Output – в это окно выводятся сообщения об ошибках или предупреждениях при автоматической верификации описаний, а также выводятся результаты аналитических запросов (в частности, статистика об использовании бинарного пространства и свободных кодах).

7. Заключение

В данной работе представлены возможности интегрированной среды MetaDSP по описанию системы команд для встраиваемых процессоров. Используя MetaDSP, можно определить синтаксис, бинарное кодирование и поведение команд в степени, достаточной для автоматической настройки основных кросс-инструментов разработки: ассемблера, дисассемблера, симулятора с профилировщиком и отладчика. Кроме того, использование автоматической верификации описаний, графического интерфейса с встроенными контекстными редакторами и иерархичность описания команд позволяют значительно повысить эффективность редактирования процессорных описаний и внесения согласованных изменений, тем самым сокращая цикл внесения типовых изменений и их отладки для десятка команд до минут в сравнении с часами и даже днями без использования интегрированной среды. Это позволяет использовать эту систему для проведения этапа проектирования (DSE) в процессе разработки встраиваемых решений. При этом получаемые кросс-инструменты обладают качеством, достаточным для производственного применения при разработке прикладных программ для целевой системы.

Система MetaDSP была успешно применена в коммерческих проектах с компаниями Freehand и VIA Technologies, в которых были получены наборы инструментальных средств кросс-разработки для пяти различных встраиваемых процессоров (включая процессоры цифровой обработки сигналов и RISC-контроллер) и десятков их модификаций.

Литература

1. *Hiroyuki Tomiyama, Ashok Halambi, Peter Grun*. Architecture Description Languages for Systems-on-Chip Design. Center for Embedded Computer Systems, Univertsity of California. 2000.
2. *Wei Qin, Sharad Malik*. Architecture Description Languages for Retargetable Compilation. The Compiler Design Handbook, CRC Press, 2003.
3. *Clifford Liem, Pierre G. Paulin, Ahmed A.Jerraya*. Retargetable Compilers for Embedded Core Processors. Kluwer Academic Publishers, 1997.
4. *Rainer Leupers*. Retargetable Code Generation for Digital Signal Processors. Kluwer Academic Publishers, 1997.
5. *Lin Yung-Chia*. Hardware/Software Co-design with Architecture Description Language. Programming Language Lab. NTHU. 2003.
6. *A. Fauth, J. Van Praet, M. Freericks*. Describing Instruction Set Processors Using nML. Proc European Design and Test Conf., Paris, March 1995.
7. *Mark R. Hartoog, James A. Rowson, Prakash D. Reddy*. Generation of Software Tools from Processor Descriptions for Hardware/Software Codesign. Alta Group of Cadence Design Systems, Inc. DAC 1997.
8. Sim-nML Homepage. http://www.cse.iitk.ac.in/sim-nml/
9. ISDL Project Homepage. http://caa.lcs.mit.edu/caa/home.html
10. *George Hadjiyannis, Silvina Hanono*. ISDL: An Instruction Set Description Language for Retargetability. Srinivas Devadas. Department of EECS, MIT. DAC 1997.
11. EXPRESSION Homepage. http://www.cecs.uci.edu/~aces/index.html

12. *Ashok Halambi, Peter Grun, Vijay Ganesh, Asheesh Khare, Nikil Dutt and Alex Nicolau*. EXPRESSION: A Language for Architecture Exploration through Compiler/Simulator Retargetability, DATE 99.
13. *Prabhat Mishra, Frederic Rousseau, Nikil Dutt, Alex Nicolau*. Architecture Description Language Driven Design Space Exploration in the Presence of Coprocessors. SASIMI 2001.
14. *В.В. Рубанов, Д.А. Марковцев, А.И. Гриневич.* Динамическая поддержка расширений процессора в кросс системе. Труды ИСП РАН, том 5, 2004.