СПЕЦИАЛИЗИРОВАННЫЙ ПРОЦЕССОР – СВОИМИ РУКАМИ
ПРОЕКТИРОВАНИЕ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ПРОЦЕССОРОВ В МНОГОЯДЕРНЫХ СИСТЕМАХ С ПОМОЩЬЮ ASIP DESIGNER

И.Попов[1]

УДК 621.3.049.77
ВАК 05.27.00

Процессоры с проблемно-ориентированной системой команд или, проще, проблемно-ориентированные (специализированные) процессоры (Application-Specific Instruction-set Processor – ASIP) предназначены для реализации конкретных типов приложений. Их можно рассматривать как высокопроизводительную и энергетически эффективную альтернативу процессорам общего назначения. При этом крайне актуальна проблема разработки высокоэффективных ядер ASIP и многоядерных систем на их основе. Создание многоядерной системы осложняется задачей выбора и оптимизации архитектуры ASIP, что требует применения специальных средств и методов проектирования. Компания Synopsys предлагает эффективный набор инструментов – ASIP Designer, ориентированный на быстрый поиск архитектурного решения, синтез аппаратного и программного обеспечения, компилирование и верификацию проекта на базе ASIP.

Современные системы-на-кристалле (СнК) содержат в себе множество различных функций, каждая из которых оптимизирована с точки зрения производительности, энергопотребления и занимаемой на кристалле площади. С другой стороны, рынок диктует жесткие условия успешности и конкурентоспособности продукта: минимальная стоимость разработки и производства, а также время за которое разработка пройдет путь от идеи до готового к продаже продукта. Аппаратная реализация функций СнК возможна в рамках нескольких концепций: процессоры общего назначения, интегральные схемы специального назначения и проблемно-ориентированные процессоры, занимающих промежуточное положение между ними.

Процессоры общего назначения предназначены для решения широкого спектра задач, поэтому такой подход обладает наибольшей гибкостью. Однако очевидно, что за универсальность нужно платить как в прямом, так и в переносном смысле. Стоимость покупного дизайна (IP) процессора общего назначения может быть сопоставима с общим бюджетом проекта, как в денежном эквиваленте, так и с точки зрения площади и потребляемой мощности. Кроме того, с помощью процессоров общего назначения, порой, невозможно добиться производительности, требуемой для определенной задачи.

Прямой противоположностью процессоров общего назначения являются интегральные схемы специального назначения (Application Specific Integrated Circuit – ASIC). Данный тип устройств спроектирован и оптимизирован для решения одной заданной функции и в большинстве случаев полностью лишен какой-либо

[1] Ведущий инженер-консультант, ООО "Синопсис".

гибкости. В случае изменения аппаратной функции или программного обеспечения требуется полный цикл разработки.

Проблемно-ориентированные процессоры (Application-Specific Instruction-set Processor – ASIP) занимают промежуточное положение в спектре микропроцессорных решений, сочетая гибкость процессоров общего назначения с эффективностью ИС специального назначения. ASIP способны выполнять инструкции из системы команд, разработанной для выполнения специфичных функций. Система команд является основной характеристикой проблемно-ориентированного процессора. Еще одной отличительной особенностью того или иного ASIP является то, как операции, входящие в систему команд, будут реализованы на аппаратном уровне или, другими словами, какие микроархитектурные решения наиболее оптимально покрывают спектр решаемых таким процессором задач.

Проблемно-ориентированный процессор может быть создан несколькими путями. Первый путь – попробовать взять за основу существующий на рынке процессор и изменить его под новые требования проекта. Преимущество этого подхода состоит в наличии исходной точки проектирования с хорошо известными параметрами, поскольку свойства исходного компонента заранее известны и протестированы. Кроме того, большинство поставщиков компонентов также предоставляют средства разработки программного обеспечения и драйверы. Однако существенный недостаток данного подхода состоит в том, что ограниченные возможности изменения архитектуры и предопределенные микроархитектурные решения существенно сужают сферу его применения, и в большинстве случаев необходимо полностью пройти цикл разработки, включающий в себя как создание аппаратных средств, так и средств разработки программного обеспечения. Отдельное внимание, при этом, стоит обратить на компоненты верификации и прототипирования, так как система зачастую создается с чистого листа.

В системах, построенных на базе процессоров общего назначения, реализация конкретной функции возложена большей частью на программное обеспечение при неизменной системе команд. В отличие от этого, в системах, использующих ASIP, изменения системы команд может происходить довольно часто, поскольку реализация функций системы возложена на аппаратуру, архитектура которой оптимизирована для выполнения инструкций из специализированной системы команд. Очевидно, что изменения в системе команд неизбежно влекут за собой изменения как в аппаратуре, так и в системном программном обеспечении, и требуют верификации.

```
// Start of structural skeleton
mem DM[1024]<num,addr>;
reg R[4]<num>;
pipe C<num>;
trn A<num>; trn B<num>;
fu alu;
...

// Start of instruction-set grammar
opn my _ core (alu _ inst | mac _ inst | shift _ inst);
...

opn alu _ inst (op:opcod, x:c2u, val:c16s, y:c2u) {
  action {
    stage EX1:
      A = R[x];
      B = val;
      switch (op) {
        case add : C = add(A, B) @alu;
        case sub : C = sub(A, B) @alu;
        case and : C = and(A, B) @alu;
        case or : C = or(A, B) @alu;
      }
    stage EX2:
      R[y] = C;
  }
  syntax : op " R" y ", R" x ", " val;
  image : "0"::op::x::y::val;
}
...
```

Рис.1. Фрагмент nML-описания процессора

Идеальным решением для проектирования проблемно-ориентированных процессоров могла бы стать система, позволяющая автоматизировать цикл проектирования как аппаратных, так и программных средств, а также инструментов верификации на основе описания системы команд, сохранив при этом возможность задавать описание микроархитектуры и деталей реализации аппаратуры.

Данная концепция была реализована командой разработчиков компании Target Compiler Technology, которая сейчас является частью компании Synopsys, в продукте под названием ASIP Designer.

В основу данного решения лег язык описания архитектуры (Architecture Description Language – ADL), называемый nML. nML используется как единое описание для автоматической генерации различных моделей, например, симулятора системы команд (Instruction Set Simulator – ISS), компилятора, генератора тестов и, наконец, описания аппаратуры на уровне регистровых передач (Register Transfer Level – RTL). Благодаря уникальной технологии компиляции, основанной на математическом аппарате ориентированных двудольных графов, широко применяемом в теории кодирования, сгенерированные модели являются взаимно эквивалентными.

ASIP Designer охватывает практически все аспекты логического проектирования ядра проблемно-ориентированного процессора – от исследования вариантов архитектурных решений с помощью автоматически генерируемого симулятора ISS до построения RTL-модели на языках VHDL или Verilog и средств разработки программного обеспечения (SDK). Причем, время, затраченное на полную итерацию, начиная от изменения системы команд или архитектуры до получения оптимизированных моделей (ISS, RTL, SDK), занимает всего несколько минут в отличие от длительного и сложного процесса ручного создания моделей с последующей их оптимизацией. Еще одно преимущество данного подхода состоит в едином входном описании проекта. С одной стороны, описание на языке nML сходно с описанием на естественном языке (в данном случаи, английском), с другой стороны, ADL-языки хорошо структурированы, что позволяет сделать описание архитектуры достаточно точным и избежать разночтений. В отрасли давно идет дискуссия о необходимости создания единого стандарта описания аппаратуры, способного обеспечить необходимую и достаточную информацию об устройстве как для создания аппаратуры, так и для построения программных моделей и верификации. Чтобы удовлетворить этому требованию, nML-описание состоит из двух основных частей: первая часть описывает аппаратные ресурсы ядра процессора, вторая – систему команд.

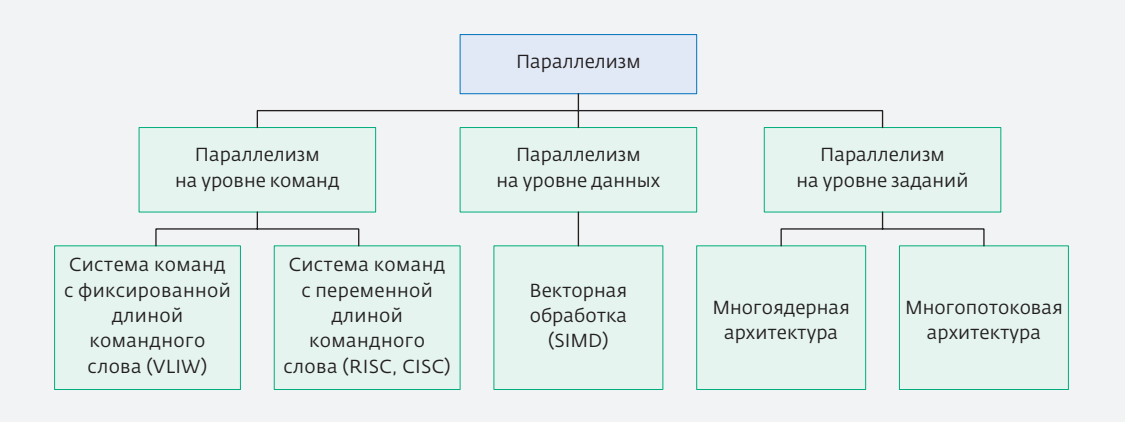

Рис.2. Опции параллелизма при проектировании ASIP

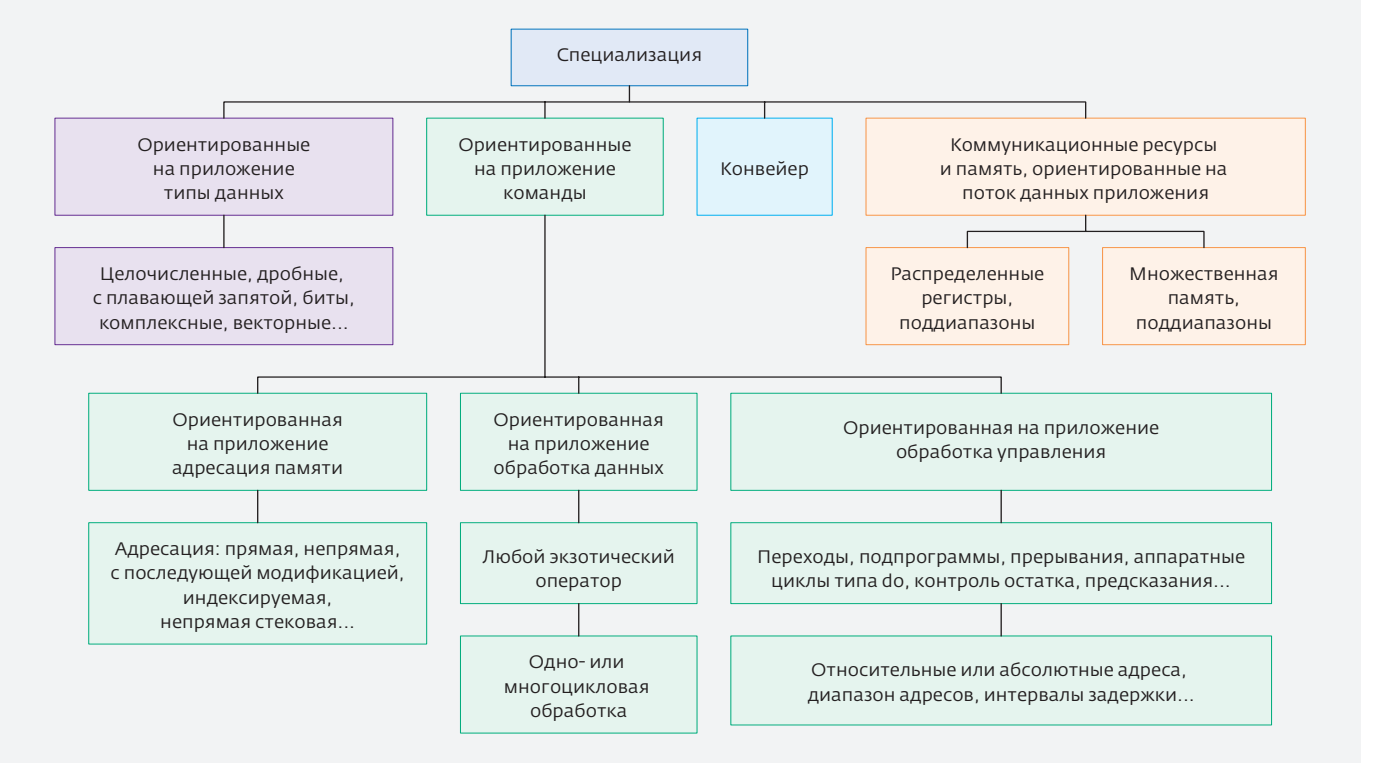

Рис.3. Опции специализации при проектировании ASIP

ASIP DESIGNER: ПРИНЦИПЫ

В ASIP Designer язык nML определяет как архитектуру процессора, так и систему команд. В первой части nML-описания, называемой структурным скелетом (structural skeleton), декларируются все модули памяти, связи и функциональные блоки проекта (рис.1). Во второй части nML-описания с помощью атрибутов определяется система команд. Интерфейс nML транслирует описание процессора в промежуточное представление, называемое графом системы команд (Instruction-Set Graph – ISG), который отображает все важнейшие структурные и временные параметры проекта.

В центре процесса проектирования СнК на базе ASIP – поиск и выбор оптимального архитектурного решения. При оптимизации архитектуры разработчики систем на базе ASIP применяют те же концепции, что и при разработке классических систем, а именно, параллелизм и специализацию, однако при реализации проекта на базе ASIP эти концепции приобретают особенности, связанные с гибкостью и программируемостью ASIP.

Параллелизм – это ключевой элемент практически всех проектов на базе ASIP с различными вариантами архитектур (рис.2). В ASIP используется несколько уровней параллелизма. Параллелизм на уровне команд применяется, например, в случае архитектур на основе сверхдлинного командного слова (Very Long Instruction Word – VLIW). Параллелизм на уровне данных используется в SIMD-архитектуре. Параллелизм на уровне заданий характерен для многоядерных/многопотоковых архитектур. Кроме того, возможна реализация любых комбинаций этих архитектур.

Еще одна важнейшая особенность ASIP – это специализация. Она означает способность выполнять заданные сложные функции с помощью одной или нескольких команд и возможность адаптировать к определенным задачам исполнительные конвейеры, внутренние/внешние коммуникационные шины и интерфейсы, а также ресурсы памяти (рис.3).

Для описания и оптимизации архитектуры процессора в ASIP Designer разработчику доступен широкий выбор параметров и опций. Среди них – поддержка разнообразных типов данных (целочисленных, дробных, с плавающей запятой, булевых, векторных (SIMD) и комплексных). В дополнение к встроенным функциям и операторам языка С поддерживаются определяемые пользователем функции. Возможны команды переменной длины, инструкции со сверхдлинным командным словом VLIW и т.п.

Возможен выбор как фон-неймановской, так и гарвардской архитектуры ядра (соответственно, с общей и с раздельной памятью данных и команд). Отсутствуют ограничения на количество определяемых блоков памяти данных или портов памяти. С помощью API-интерфейса памяти могут быть подключены модели внешней памяти и коммуникаций с зависящим от трафика данных

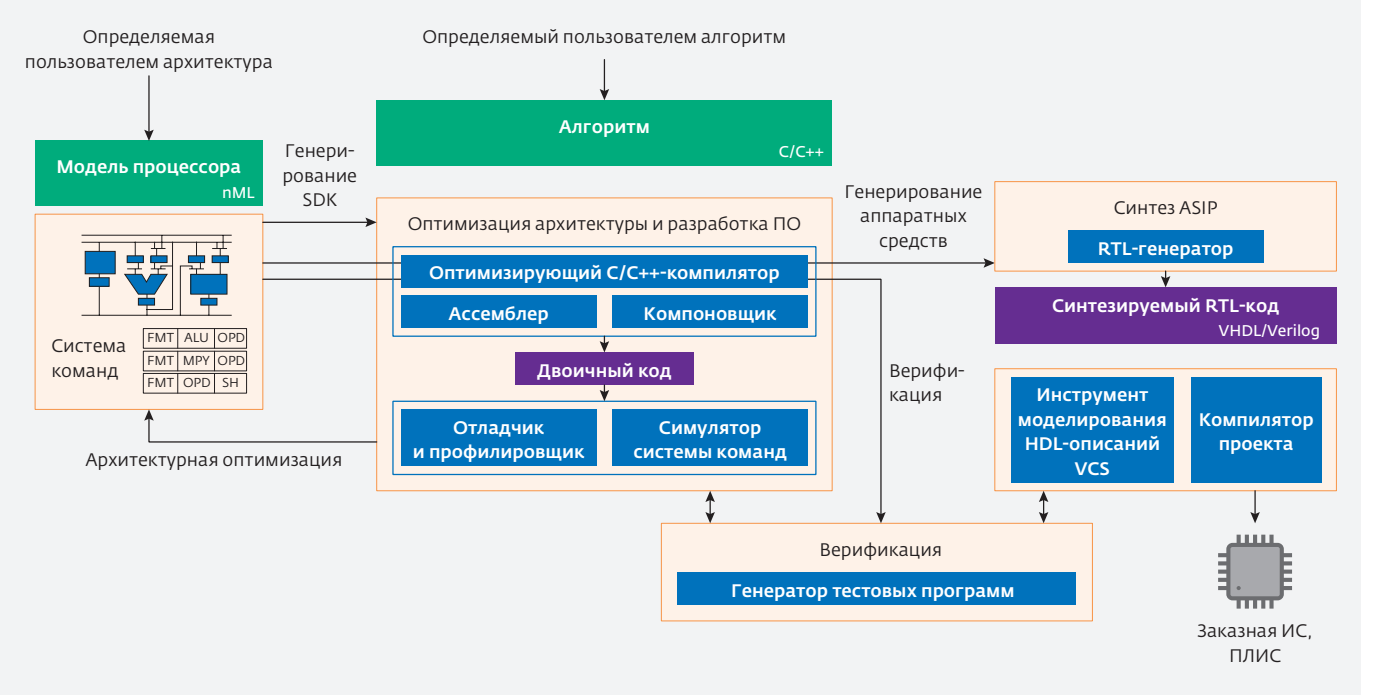

Рис.4. Набор инструментов ASIP Designer

временем ожидания. Компилятор поддерживает широкий набор режимов адресации. В nML точно описывается конвейер команд, в том числе способы разрешения конфликтов конвейера. Отсутствуют ограничения на число стадий конвейера.

Язык nML позволяет определять наборы регистров специального назначения и нестандартные схемы межсоединений, которые полностью поддерживаются компилятором, используя усовершенствованный метод распределения регистров. Компилятор гарантирует отсутствие конфликтов конвейера в сгенерированном коде.

Возможны все актуальные опции управления программой: вызов подпрограмм, прерывания, аппаратные циклы типа DO, контроль остатка, предсказания переходов и исполнение команд под условием.

ПРОЦЕСС РАЗРАБОТКИ

В ASIP Designer реализован базовый принцип, коренным образом отличающий его от традиционных подходов к созданию процессорных ядер. Он заключается в том, что процессор создается под конкретный пользовательский алгоритм. Соответственно, если традиционно средства разработки и отладки прикладных программ (SDK) – компиляторы, отладчики, среды разработки – создаются на последнем этапе разработки процессора, то в случае ASIP Designer – это процесс, непосредственно следующий за построением модели ядра.

Поиск оптимальной архитектуры ASIP – итерационный процесс, в котором обычно в качестве отправной точки используется описанная на языке nML простая базовая архитектура процессора. ASIP Designer предлагает несколько базовых процессоров, а именно, компактное 16-разрядное процессорное ядро, в котором используется всего несколько тысяч вентилей, но способное работать на частоте более 1 ГГц; производительное 16- или 32-разрядное процессорное ядро для приложений, ориентированных на управление, а также DSP-ядро, оптимизированное для приложений обработки сигналов с векторными типами данных и параллельным исполнением команд. Эти базовые процессоры могут быть полностью настроены и оптимизированы под определенное приложение, в результате чего достигается наилучшая производительность и/или энергоэффективность.

Процесс проектирования процессорных ядер с помощью ASIP Designer включает в себя следующие этапы (рис.4).

Создание модели процессора. Используя язык описания процессора nML, пользователь определяет архитектуру системы и структуру системы команд. Разработчикам доступен широкий набор архитектур ASIP – от процессора общего назначения до ядра цифровой обработки сигналов (DSP), вычислительного сопроцессора и т.п. Таким образом, формируется некая исходная модель ядра, которая затем оптимизируется под конкретные алгоритмы, для реализации которых и создается процессор.

Настройка средств разработки (SDK – Software Development Kit). После того, как сформирована

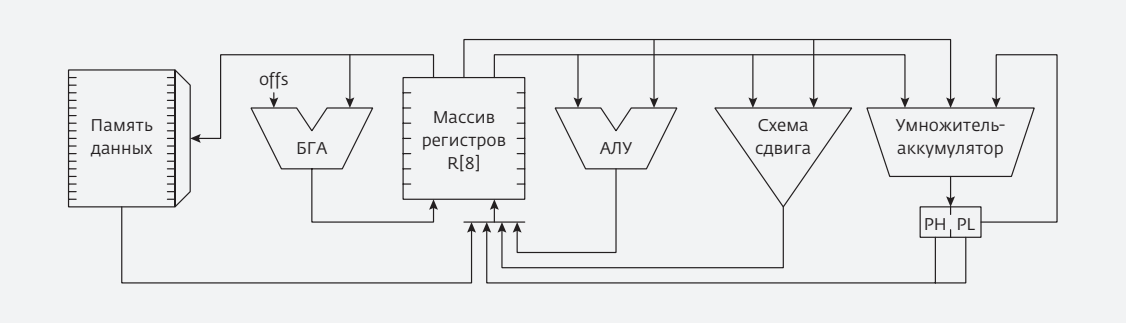

Рис.5. Структура ядра Tmicro

первичная модель процессора, необходимо проверить его эффективность на целевой задаче, написанной на языке высокого уровня – поскольку именно под них создается и оптимизируется ядро. Для этого необходим симулятор системы команд, средства разработки и отладки программ высокого уровня, а также комплект трансляции программы в машинные коды процессора. Это традиционный набор инструментов для разработки программ. Однако уникальная особенность ASIP Designer – перенастраиваемость всех инструментов под определенную архитектуру на основе описания процессора на языке nML. По сути, такое описание ASIP служит входной информацией для адаптации среды разработки.

Важнейший инструмент SDK – ***генератор симулятора системы команд*** (ISS). Он позволяет моделировать исполнение команд разрабатываемого процессора на основании nML-описания ядра с точностью до цикла или команды. Таким образом, создав nML-модель ядра, благодаря ISS разработчик может немедленно проверить работу системы команд, причем на конкретных прикладных программах.

Трансляцию прикладных программ с языка C/C++ в машинные коды обеспечивает другой инструмент ASIP Designer – ***C-компилятор.*** В отличие от традиционных компиляторов (например, GCC), C-компилятор из пакета ASIP Designer использует методы моделирования и оптимизации на основе графов, ориентированные на создание кода для специализированных архитектур. Такие структуры отличаются возможностью конвейерной обработки команд, гетерогенной структурой регистров, наличием блоков реализации специальных функции и параллелизмом на уровне команд. В состав пакета входят ассемблер и дизассемблер, а также компоновщик загрузочного кода (редактор связей).

Для удобства работы служит графический отладчик. Его можно использовать в том числе для моделирования системы команд и для внутрисхемной отладки.

Оптимизация архитектуры. На основе модели системы команд, сгенерированной ISS, а также с помощью средств SDK разработчик может отладить и оптимизировать архитектуру ASIP, чтобы она в максимальной степени соответствовала целевым алгоритмам, для реализации которых создается ASIP.

Генерирование аппаратных средств. Получив оптимизированную модель процессора на nML, ее можно автоматически транслировать в аппаратное представление на уровне модели регистровых передач (RTL-уровень). Для этого используется ***генератор аппаратных средств (RTL-генератор).*** Он синтезирует VHDL- или Verilog-описание ядра. Помимо функций синтеза, RTL-генератор способен оптимизировать схему для снижения энергопотребления, например, реализовав стробирование тактового сигнала для определенных регистров. Предусмотрена опция формирования необходимой внутренней инфраструктуры процессора для внутрисхемной отладки через JTAG-интерфейс. В результате формируется RTL-код, пригодный для синтеза конкретных схем или IP-блоков и их реализации в различных СБИС или ПЛИС.

В ASIP Designer входит еще один важный инструмент – ***генератор тестовых программ.*** Он способен генерировать тестовые программы на ассемблере целевого процессора с высоким покрытием возможных сбоев. Эти программы можно исполнять как на ISS-, так и на RTL-моделях. Кроме того, в процессе проектирования ASIP Designer генерирует программы предварительного тестирования и комплекты для регрессионного тестирования.

ПРИМЕР ОПТИМИЗАЦИИ АРХИТЕКТУРЫ ASIP: ПРИЛОЖЕНИЕ ДЛЯ ОЦЕНКИ ДВИЖЕНИЯ

Для иллюстрации процесса создания и оптимизации архитектуры с помощью набора инструментов ASIP Designer рассмотрим простой пример – приложение для реализации процедуры оценки движения (motion estimation). Этот алгоритм применяется во многих системах видеокодирования, таких как H.264/MPEG-4 AVC и H.265/HEVC. Это один из тех процессов в кодировании видеосигналов, которые требуют большого объема вычислений. Для оценки движения чаще всего используют метод сопоставления блоков в пространстве поиска, который отличается сравнительной

простотой аппаратной реализации и высокой вычислительной эффективностью.

Метод сопоставления блоков заключается в выполнении следующих действий. Текущий кадр разбивается на множество непересекающихся блоков: размером 16 x 16 пикселей для стандарта H.264 или размером от 8 x 8 до 64 x 64 пикселей – для H.265. Для каждого блока текущего кадра производится поиск наиболее похожего блока (реперного блока) в предыдущем кадре. Разность между позициями текущего и реперного блоков определяет вектор движения текущего блока. Для поиска оптимального вектора рассчитывается коэффициент подобия между текущим блоком и реперным блоком.

Степень подобия определяется суммой абсолютных разностей (sum of absolute differences – SAD):

$$SAD(x,y) = \sum_{i=0}^{15}\sum_{j=0}^{15}\left|search(i+x,j+x) - curr(i,j)\right|,$$

где search – пиксель текущего блока, curr – пиксель реперного блока.

При поиске оптимального решения для нашего приложения на каждом этапе итерационного процесса различные варианты архитектур будем сравнивать по:

- числу циклов, которые требуются для выполнения приложения и отдельно для расчета функции SAD (по результатам моделирования в ISS);
- числу команд, которые требуются для выполнения приложения и отдельно для расчета функции SAD (по результатам компилирования);
- числу вентилей, используемых в данной архитектуре, не считая память, для частоты тактового сигнала 500 МГц и 28-нм технологии от TSMC (по результатам моделирования аппаратных средств и логического синтеза);
- мощности, потребляемой процессорным ядром (по данным сгенерированного SAIF-файла);
- энергии, необходимой для выполнения SAD (произведение мощности потребления ядра на время выполнения функции).

Основным критерием уровня оптимизации в данном примере является количество командных циклов, которое требуется для исполнения приложения. Поскольку меньшее число циклов позволяет ASIP работать на более низкой частоте и с меньшей потребляемой мощностью, опосредованно оптимизируется также энергопотребление.

Базовый вариант: 16-разрядный микроконтроллер Tmicro

В качестве базовой архитектуры проекта выберем 16-разрядный микропроцессор, называемый Tmicro, который входит в состав библиотеки моделей, поставляемой вместе с ASIP Designer. Архитектура Tmicro доступна в виде nML-описания. Структура ядра Tmicro включает в себя арифметико-логическое устройство (АЛУ), умножитель-аккумулятор (УА), схему сдвига (СС), блок генерации адреса (БГА), восемь регистров данных и память данных на 64 кбайт (рис.5). Ядро Tmicro содержит отдельную память программ на 64

```
Program being simulated: /home/demo/asip _ labs/tmotion1/Release/motion
  Total cycle count           : 69567
  Total instruction count     : 68649

Function summary:
  Cycles   % of total Instruction % of total Function         Relative cycle
use in simulation
  -------- ---------- ----------- ---------- ---------------- -----------------
---------------
    67716      97.34%         66996      97.59% sad _ 16x16        *****************
************************
     1600       2.30%          1432       2.09% motion _ estimation*
       28       0.04%            19       0.03% main
Function detail: sad _ 16x16

  Low PC          :       4
  High PC          :     22
  Cycle-count           :      67716 (97.34%)
  Instruction-count     :      66996 (97.59%)

  PC   Instruction  Assembly             Exe-count Cycles  Relative cycle use
in simulation
  ---- ------------ ------------------ --------- -------- ---------------------
--------------
  4      3003          mvib r3,0           36        36
  5      3204          mvib r4,32          36        36
  6      3000          mvib r0,0           36        36
  7      2ee000150010 doi 16,21           36        108
  10     3107          mvib r7,16          576       576       ***
  11     2edf0014      do r7,20            576       1152      ******
  13     2e00          nop                 576       576       ***
  14     4a55          lbu r5,dm(r1++)     9216      9216      ******************
*********************************
  15     4a96          lbu r6,dm(r2++)     9216      9216      ******************
*********************************
  16     056e          sub r5,r5,r6        9216      9216      ******************
*********************************
  17     059d          sub r6,r3,r5        9216      9216      ******************
*********************************
  18     0e2b          lt r5,r3            9216      9216       *****************
*********************************
  19     2375          sel r5,r6,r5        9216      9216      ******************
*********************************
```

Рис.6. Фрагмент выходной информации симулятора ISS для кода приложения на базе Tmicro

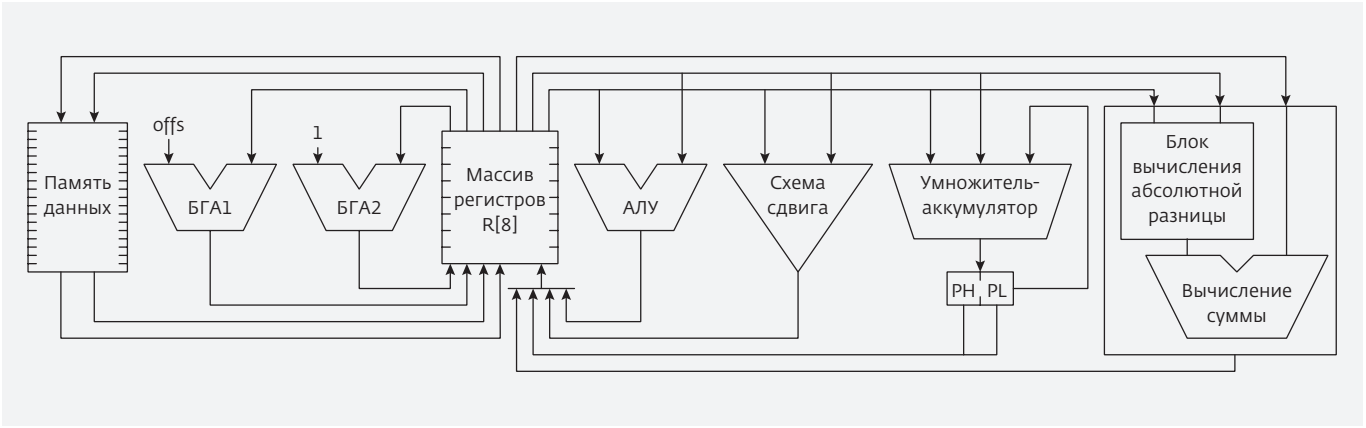

Рис.7. Структура процессора на базе ядра Tmicro со скалярным ускорителем

кслов по 16 бит. Разрядность команд – 16, 32 или 48 бит. Заметим, что пиксель изображения в нашем приложении занимает один байт.

После моделирования сгенерированного кода симулятор ISS предоставляет отчет, в котором отражается вся выходная информация (рис.6). Она содержит статистику по использованию команд и аппаратных ресурсов (регистров, шин и т.д.). Кроме того, для каждой функции C отображается сгенерированный машинный код с указанием времени, которое требуется для выполнения команды, и числа циклов, потраченных на каждую команду.

Анализ ассемблерного кода данного приложения показал, что для базовой архитектуры критичным участком кода является строки с 14 по 20 (см. рис.6), в которых реализуется внутренний цикл для вычисления функции SAD. Можно отметить, что для вычисления абсолютного значения SAD-функции требуется пять команд: две команды вычитания (sub), одна команда сравнения (lt), одна команда выбора (sel) и одна команда сложения (add). Кроме того, заметим, что параллелизм на уровне команд в базовом ядре Tmicro не доступен за исключением вычисления адреса (например, r2++) параллельно с загрузкой из памяти (lbu).

Вариант 1: микропроцессор со скалярным ускорителем

Для повышения производительности был предложен вариант реализации дополнительного специализированного функционального блока – скалярного ускорителя, который вычисляет значения функции SAD с последующим аккумулированием за один цикл. Такая модификация не требует значительного расширения аппаратного обеспечения. Кроме того, желательно было бы реализовать дополнительную параллельную организацию выполнения команд. Второй канал загрузки в память данных позволили бы организовать одновременную загрузку данных о пикселе реперного блока и пикселе текущего блока параллельно с вычислением и аккумулированием абсолютной разницы. Второй канал загрузки требует реализации еще одного блока генерирования адреса (рис.7).

После внесения необходимых изменений в nML-описание и запуска ISS для моделирования нового варианта процессора были получены оптимизированные результаты (см. табл.). Как видно из таблицы, суммарное число циклов снизилось на целых 79%, в то время как число команд осталось практически неизменным, а число вентилей увеличилось всего

Характеристики альтернативных вариантов проекта

Варианты реализации проекта	Число циклов (всего)	Число циклов для SAD-функции	Число команд (всего)	Число команд для SAD-функции	Число вентилей (тактовая частота 500 МГц)	Потребляемая мощность, мВт	Энергия, мкДж
Базовый вариант	69567	67716	193	19	11743	0,789	109,8
Вариант 1	14344	12492	193	18	12292	0,765	21,9
Вариант 2	5428	3780	185	13	25451	1,079	11,7
Вариант 3	3772	2124	186	14	29342	1,467	11,1
Вариант 4	3268	1620	187	15	26976	1,485	9,7

на 5%. Энергопотребление также снизилось на 80%. Благодаря организации программного конвейера в компиляторе тело внутреннего цикла вычисления SAD-функции теперь реализуется за одну высокопараллельную команду, для выполнения которой требуется всего один цикл на пиксель.

Для дальнейшего повышения скорости работы приложения можно рассмотреть альтернативный вариант архитектуры, способной обрабатывать множество пикселей за один цикл. Это возможно с помощью векторной обработки данных.

Вариант 2: микропроцессор с векторным ускорителем

Обработка множества пикселей параллельно требует введения параллелизма на уровне данных. Для этого ядро Tmicro было дополнено векторным ускорителем, использующим принцип SIMD-обработки (рис.8). Новый ASIP поддерживает вектор размером 16 элементов, каждый длиной в один байт. Это соответствует одной строке пикселей реперного и текущего блока (16×16 пикселей) рассматриваемого приложения. Память данных теперь может загружать и хранить сразу полный вектор. Введен также дополнительный массив регистров, состоящих из четырех векторных регистров.

Векторный ускоритель может теперь вычислять абсолютную разность для полных векторов за один цикл. Кроме того, введена отдельная команда вычисления векторной суммы, которая складывает все элементы вектора, содержащего абсолютные разности. Команды загрузки и хранения вектора работают над блоками данных, привязанными к 16-байтным границам в памяти. Однако блок текущего кадра может не всегда сохраняться в таком положении, поэтому введен также блок выравнивания вектора.

Компилятор С поддерживает векторную или SIMD-обработку, обеспечивая включение векторных типов данных и функций в исходный С-код. Стандартные операторы С могут быть переопределены на векторные типы данных. Кроме того, могут быть введены встраиваемые функции, такие как vadiff () (векторная абсолютная разница).

В результате такой модификации кода, одна векторная команда vadiff может теперь обрабатывать полную строку пикселей за один цикл, что позволяет эффективно использовать параллелизм на уровне данных.

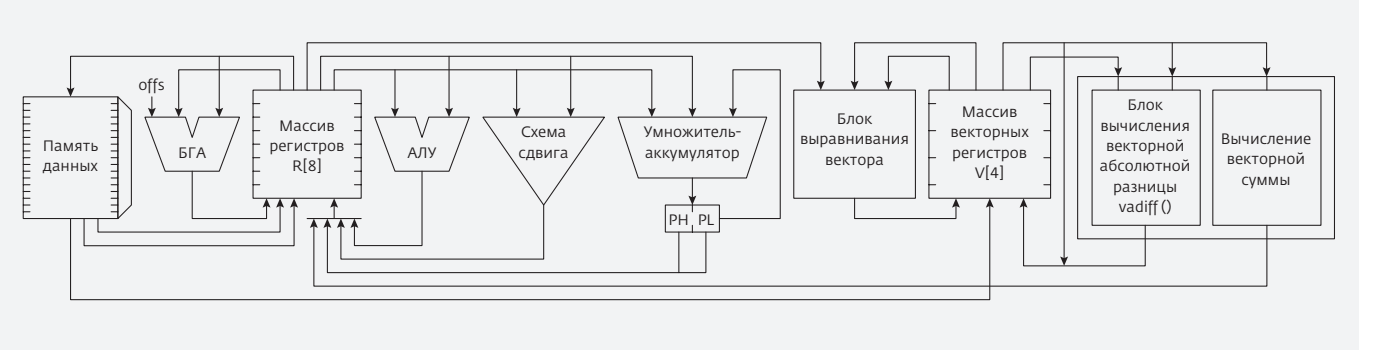

Рис.8. Структура процессора на базе ядра Tmicro с векторным ускорителем

По сравнению с предыдущей архитектурой, это позволило снизить число циклов на 62%, а энергопотребление на 47% (см. табл.). Однако, за исключением параллельного вычисления адреса и загрузки памяти, в таком варианте векторного ASIP отсутствует параллелизм на уровне команд. В результате введения дополнительного массива регистров и блоков реализации векторных функций общее число вентилей удвоилось. Чтобы дополнительно повысить скорость работы приложения, следующим шагом модификации архитектуры может стать введение дополнительного параллелизма на уровне команд.

Варианты 3 и 4: векторные ASIP с параллелизмом на уровне команд

С целью введения параллелизма на уровне команд были выполнены две дополнительные итерации проекта. Вариант 3 поддерживает выполнение одной команды векторной загрузки (ld) параллельно с векторным выравниванием (valign) и векторными арифметическими командами (vadiff, vsum). В варианте 4 добавлена вторая параллельная команда векторной загрузки. Это требует раздельной памяти для реперного блока и текущего блока. C-компилятор использует директивы размещения (allocate) переменных в специальной памяти.

В результате этих модификаций было достигнуто повышение скорости еще на 40%, что обеспечило экономию энергопотребления на 13% практически без использования дополнительных аппаратных ресурсов (см. табл.).

* * *

Очевидно, что традиционный подход, основанный на процессорах общего назначения, не способен удовлетворить всем требованиям системы-на-кристалле. Использование проблемно-ориентированных процессоров позволяет найти оптимальный баланс параметров микроархитектуры и создать процессорные ядра, наиболее полно отвечающие заданным требованиям проекта.

Уникальными возможностями ASIP Designer уже воспользовались более 250 команд разработчиков по всему миру. Спектр созданных с помощью ASIP Designer продуктов охватывает: сверхэнергоэффективные устройства для медицины, различные компоненты систем связи (от базовых станций до мобильных устройств), системы обработки изображения и видеосигнала, автомобильную электронику, индустриальные контроллеры, Интернет вещей.