Язык параллельного программирования CAPER как средство для эффективного архитектурного моделирования современных СБИС.

Акопджанян, В.А., Вартанов. С.Р.

В статье обсуждаются вопросы использования языка параллельного программирования Caper в качестве средства эффективного архитектурного моделирования современных СБИС. В работе проведено сравнение методов программирования и моделирования на языках Caper и SystemC. В работе продемонстрировано, что программирование на языке Caper не уступает по эффективности программированию на языке SystemC, а в отдельных случаях является более простым и удобным. Также показано, что скорость симуляции для отдельных классов задач на языке Caper значительно превосходит (в разы для так называемых SC_THREAD) скорость симуляции на языке SystemC.

1. Введение

Современное архитектурное моделирование СБИС [1] становится все более актуальной темой для исследований благодаря очевидным преимуществам, которое оно приносит. К таким преимуществам относятся

- возможность ранней диагностики функциональных проблем в СБИС, таких как, несовместимость различных устройств и протоколов, а также различных ошибок поведения;
- возможность получать приблизительное представление о быстродействии проектируемой СБИС на этапе раннего проектирования;
- возможность более быстрой симуляции архитектурных моделей по сравнению с аналогичными RTL моделями.

Область моделирования интегральных схем отличается тем, что в процессе моделирования одновременно выполняются несколько функций в различных участках схемы, следовательно, наиболее естественной парадигмой моделирования СБИС является параллельное программирование. С другой стороны, наиболее распространенные на данный момент средства архитектурного описания, такие как язык SystemC, с описательной точки зрения основаны на языке C++ - языке последовательного программирования, - что не позволяет адекватно и эффективно программировать параллельное поведение элементов схем. В силу того, что проектирование СБИС происходит на композиционной основе, а также того что сложность СБИС постоянно растет, то постоянно растут и требования к средствам эффективного описания схем и симуляции их поведения. Это приводит к чрезмерному усложнению программирования и к сложностям отладки параллельных программ на C++.

По сложившемуся мнению, средства архитектурного описания и симуляции СБИС должны позволять:

1) эффективно описывать структуры схем;
2) адекватно программировать параллельную работу элементов схем;
3) эффективную программную симуляцию описанных СБИС.

Кроме того, перечисленные выше качества определяют дополнительные требования к языку описания схем на архитектурном уровне и на уровне системы симуляции, такие как:

1) широкие средства контроля поведения СБИС;
2) возможности отладки;
3) удобство программирования моделей;
4) совместимость с другими языками описания СБИС;
5) переносимость и масштабируемость между различными платформами.

На сегодняшний день к распространенным средствам архитектурного моделирования относят язык SystemC. Однако, моделирование СБИС средствами языка SystemC имеет определен-

ные проблемы [1,3]. Так, отсутствуют полноценные отладчики, ориентированные для архитектурного моделирования. Практика указывает на неочевидное, усложненное программирование модулей (используются неудобные и непрозрачные конструкции языка C++). Кроме того, известны проблемы с масштабируемостью программ для многопроцессорных систем.

В этой связи исследовался вопрос о свойствах языка программирования, необходимых для эффективного программирования задач представления и симуляции работы СБИС. Представлялось очевидным, что таким языком должен быть язык параллельного программирования. Одновременно, возникала задача исследования принципов и свойств параллельного программирования в контексте решения поставленной задачи симуляции СБИС. Для этих целей был выбран язык императивного параллельного программирования Caper [4, 5].

В данной статье представлены результаты этих исследований, а именно:

1) результаты сопоставления возможностей программирования моделей СБИС;
2) результаты симуляции работы СБИС на языке SystemC с аналогичными возможностями языка Caper.

Последний создавался с целью разрешения следующих концептуальных положений программирования:

- структурированность программ с возможностью динамического самоизменения и самоорганизации;
- возможность динамической компиляции и исполнения программы, динамической компоновки исполняемого кода;
- параллельное исполнение элементов структуры программы с возможностью поддержания всех возможных схем инициирования параллельных вычислений;
- управление процессом параллельных вычислений на основе событий (программирование событиями);
- переносимость и независимость межу различными платформами;
- встроенная синхронизация любых типов.

Вычисления программ на Caper поддерживаются комплексом виртуальных машин, в том числе собственной параллельной виртуальной машиной, не зависящей от средств операционной системы.

Далее будут рассмотрены возможности языка Caper по отношению к перечисленным критериям. Одновременно, будет проведено сопоставление с аналогичными средствами языка SystemC. В статье будет продемонстрировано, что:

1. Программирование на языке Caper не уступает по эффективности программированию на языке SystemC, а в отдельных случаях является более простым и удобным.

2. Скорость исполнения симулирующих программ для отдельных классов задач на языке Caper значительно превосходит (в разы для так называемых SC_THREAD) скорость исполнения программ на SystemC. В других случаях (SC_METHOD) скорость исполнения не уступает программам на SystemC.

Кроме того, в статье предложен метод описания схем, который состоит в разделении описания самой схемы на уровне блоков-подсхем и описания поведения самой схемы, что в сочетании с другими свойствами языка Caper делает возможным четкое разделение описательной и исполнимой частей программы. Данное разделение делает возможным конструирование полного каркаса схемы, а потом уже насыщение каркаса реальными процедурами симуляции и анализа поведения системы. Такое разделение прозрачно и позволяет сохранять описательную часть, модифицировать исполнимую часть, и наоборот.

2. Описание СБИС на языке CAPER.

Для сопоставления возможностей языка Caper и языка SystemC нами были созданы СБИС модели, запрограммированные на языке SystemC и Caper, причем обе программы реализуют не только одинаковую функциональность, но в них на этапе симуляции инициируется одинаковое количество параллельных процессов и определен одинаковый комплекс событий. В рассматриваемых ниже моделях использованы базовые элементы типа “процессор”, “память” и “межкомпонентное соединение” (см. Рис.3).

Как известно, в SystemC и в других родственных языках [1,2] для описания цифровых схем используется базовый элемент типа “модуль”, который абстрагирует понятие схемы. Модуль может состоять из нескольких других модулей так же, как схема состоит из подсхем. Каждый модуль имеет список параметров-переменных называемых списком портов модуля. Поведение модуля определяется набором параллельно работающих процессов на общем пространстве данных. Общее пространство данных определяется списком портов и внутренних переменных модуля.

Далее определяются объекты уже описанных модулей и их соединения между собой. Этот этап принято называть этапом уточнения (“elaboration”). После удачно пройденного этапа уточнения реализуется этап симуляции системы посредством старта параллельных процессов, описанных в различных модулях.

Для того чтобы наглядно продемонстрировать разницу методов программирования на SystemC и Caper, взят пример описания простой схемы на SystemC из [4](см рис. 3) и представлено ее описание на языке Caper(см рис. 4). Эта схема моделирует простое устройство **adder** сложения двух чисел. **adder** состоит из одноименного модуля и трех портов: двух входных портов **a** и **b,** и одного выходного порта **sum**. Также имеется устройство **tester** также состоящее из одноименного модуля и трех портов: состоит из одноименного модуля и трех портов: входного порта **sum,** и выходных портов **a** и **b.** Смысл устройства **tester** состоит в верификации модуля **adder,** постредсвом предоставления ему входных векторов. Заметим также что, одноименные порты модулей **tester** и **adder** присоединены друг к другу.

Итак, ниже представлен код вышеописанной схемы на SystemC:

```
SC_MODULE(adder)            // определение модуля
{
  sc_in<int> a, b;          // определение портов
  sc_out<int> sum;

  void do_add()             // процесс моделирующий сумматор
  {
    sum = a + b;
  }

  SC_CTOR(adder)            // конструктор
  {
    SC_METHOD(do_add);      // определение процесса и
    sensitive << a << b;    // список событий реагирования
  }
};

SC_MODULE(tester)           // определение модуля
{
  sc_out<int> a, b;         // определение портов
  sc_in<int> sum;
  void test()               // процесс моделирующий тестирование
  {
       a.write(1);          // запись в порт a
       b.write(1);          // запись в порт b

       wait();              // ждем информации на порте sum
       cout<<"sum = "
        << sum<<endl;

       sc_stop();           // конец симуляции
  }

  SC_CTOR(adder)            // конструктор
  {
    SC_THREAD (test);       // определение процесса
```

```
    sensitive << sum;       // список событий реагирования
  }
};

void sc_main ()
{
    adder adder1;           // определения обьектов типа adder и tester
    tester tester1;
    adder1.a(tester1.a);    //присеодинение портов a и b модуля adder к
                            //соответствуюшим портам модуля tester
    adder2.b(tester1.b);
    sc_start(-1);           //старт симуляции
}
```

Рис. 1. Программа схемы суммирования на языке SystemC

Из-за использования языка С++ для описания схем на SystemC, возникают следующие неочевидные ошибки программирования. Использование шаблонов для определений типов портов (или сигналов), а также использование перегруженных операторов (в частности **operator()**) для выражения операции присоединения портов друг к другу (**adder1.a(tester1.a))**, ведет к неочевидным ошибкам во время программирования. Кроме того операции синхронизации допустимы только для типов SystemC, что ведет либо к усечению программирования только типами SystemC (что противоречит принципам архитектурного моделирования), либо ведет к неочевидному и избыточному коду.

Опишем аналогичную схему на языке Caper. Основной процедурной единицей языка является понятие блока команд [5] (по существу – логически идентифицируемая процедурная единица). Каждый блок команд может иметь подблоки. Для представления схемы используются блоки, параметры которых представляют порты, с помощью которых происходит обмен информации с другими модулями. Далее определяются подблоки (соответственно подсхемы) и внутренние переменные, которые и определяют дальнейшее поведение блока (то есть описываемой им схемы).

При программировании в Caper используется следующая парадигма:

1. Определяются типы процедурных блоков с помощью оператора PROTOTYPE, обеспечивающего т.н. двойственную интерпретацию типа [7]: и в качестве описателя процедурного блока, и в качестве структуры данных, описывающей процедурный блок. Данные типы будут использованы для описания цифровой схемы с последующей привязкой к схеме конкретных процедур симуляции поведения отдельных элементов.

2. Задаются реальные процедурные блоки, описывающие функционирования симулируемых устройств.

3. Осуществляется имплементация (связывание) реальных блоков в схему.

Далее описываются основные конструкции языка Caper, которые используются в программе суммирования, и отражающие логику описания структур схем конструкциями языка с последующей привязкой интерпретирующих процедур.

Как отмечалось выше, к порождаемым типам в Caper относятся и процедурные блоки. Для типизации процедур необходимо использовать инструкцию PROTOTYPE (в угловых скобках указываются типы или агрегаты типов). К примеру, тип процедур, которые ничего не возвращают и имеют единственный параметр типа <channel>:

```
prototype <null>  simpleInterface ( <channel> port )
```

Заметим, что количество и тип портов не ограничены, т.е. конструкции типа

```
prototype <null>  richInterface ( <channel1> port1, <channel2>
                                  port2,…)
```

возможны и применимы.

Реальные процедурные блоки симуляции элементов схемы описываются еще одним оператором прототипирования **internal**:

```
internal <as simpleInterface> adderModel
```

Здесь использована опосредованная форма определения прототипа процедуры: посредством агрегата AS указывается тип процедур, определенный в PROTOTYPE, которому соответствует определяемый реальный блок, т.е `adderModel` имеет параметр `port` типа channel и не возвращает значений.

Соединения между блоками схемы задаются прозрачно через построение общих переменных для портов. Для этого используются операторы динамического построения структур данных BUILD:

```
build private <channel> bundle
build private <simpleInterface> adderModule, testerModule
```

Ключевое слово PRIVATE определяет область видимости указываемой переменной в процедурном блоке и его подблоках. Первый BUILD создает структуру типа `channel`, второй – две структуры с двойственной интерпретацией `adderModule` и `testerModule` типа `simpleInterface`.

```
// Определение сигналов интерфейса комуникации посредством структуры
// описания соедининия
struct channel { <caperSignal> a, <caperSignal> b, < caperSignal > sum }

// Определение типов интерфейсов симулируемых устройств
prototype <null> simpleInterface( <channel> port )

// Обявления моделей симуляции устройств
internal <as simpleInterface > adderModel, testerModel

build private < simpleInterface > adderModule, testerModule

// Построение канала комуникации типа channel
build private <channel> bundle

// Связывание порта устройства с каналом комуникации
adder.port  := bundle
// Назначение конкретной модели симуляции усттойства
adder.body  := adderModel
tester.port := bundle
tester.body := testerModel

// Асинхронный запуск моделей
do asynch adderModule, testerModule
return ;* завершение данного параллельного процесса

// описание поведенческой модели
block adderModel (<channel> port)
   while true
        // ожидание изменинbackwards сигналов a и b
        wait port.a.event && port.b.event
        // запись в порт sum
        port.sum.write(a+b)
   endw
endblock

block testerModel ( <channel> port )
      // запись в порты a и b
      port.a := 1
      port.b := 1
      // ожидание изменинbackwards сигнала sum
      wait port.sum.event
      OutText( “sum is ”, ntofs(port.sum) )
Endblock
```

Рис. 2. Программа схемы суммирования на языке Caper.

И наконец, окончательное связывание с реальными процедурными блоками осуществляется следующими присвоениями имен процедур:

```
adderModule.body := adderModel
testerModule.body := testerModel
```

Далее описанные блоки запускаются на параллельное выполнение с помощью оператора параллельного старта:

do asynch `adderModule, testerModule`

Здесь компилятором языка Caper определяется, что переменные `adderModule`, `testerModule` имеют процедурный тип, процедуры указаны в предопределенных членах body и что параметры вызова заданы одноименными членами структур.

Те же принципы можно использовать для описания подсхем, т.е. реализовывать поведение подсхем схемы через подблоки уже определенных блоков.

Как уже отмечалось, основным преимуществом данного способа представления схем состоят в разделении описания самой схемы на уровне блоков - подсхем и описания поведения схемы. Также следует отметить модульность подхода к моделированию соединений между элементами схемы. Так как соединение представляет собой структуру Caper, то оно автономно может содержать в себе необходимые функции для паковки/распаковки данных, проверки легальности трафика и т.д., что также повышает возможности верификации схем в Caper, и позволяет осуществлять поддержку TLM (transaction level modelling)[3].

Заметим, что в силу интегрируемости Caper с другими средствами разработки средств описания и симуляции схем блокам типа adderModel можно сопоставлять процедуры, созданные на других языках.

Кроме того, ввиду эффективной поддержки асинхронных параллельных вычислений Caper является привлекательным языком для моделирования следующего поколения цифровых схем - так называемых асинхронных схем, в которых отсутствует понятие синхронизации сигналов интерфейса по сигналу Clock [1].

3. Скорость симуляции СБИС на CAPER.

Для сопоставления скоростей симуляции СБИС было промоделировано соединение нескольких процессоров и модулей памяти через устройство связи (interconnect) (см. рис. 1) с помощью программ, разработанных на языке Caper и языке SystemC. Для организации сообщений внутри СБИС, было выбрано подмножество протокола коммуникации AHB [].

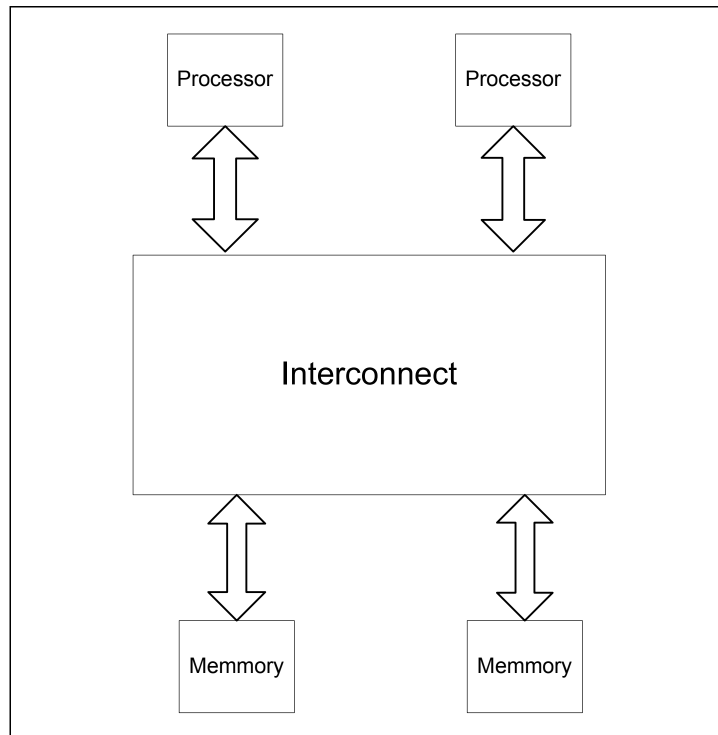

Рис.3. Схема конфигурируемой модели на языке Caper.

Была создана конфигурируемая модель, в которой количество процессоров и модулей памяти можно задавать на начальном этапе симуляции. Выбранная схема состоит из следующих подсхем: модуля моделирующего процессор, модуля моделирующего память, и модуля связи. Каждый процессор реализует процедуры посылки данных, получения данных, поддержания порта данных и реализации некоей программы. В свою очередь, каждая из перечисленных процедур реализуется самостоятельным параллельным процессом.

Модуль памяти поддерживается единственной процедурой и обеспечен одним портом. Задачей процедуры поддержки является получение запросов от процессора и выдача ответов.

Блок устройства связи отвечает за корректную маршрутизацию запросов от процессоров к блокам памяти. Количество его портов определяется количеством присоединенных к нему процессоров и модулей памяти. Каждому принимающему и оправляющему порту соответствует собственный параллельный процесс. Каждый “принимающий” процесс отвечает за транспортировку запроса от процессора к памяти, а каждый “отвечающий” процесс за транспортировку данных из памяти к процессору, инициировавшего запрос к памяти.

Для сравнения скорости работы симулятора СБИС на языке CAPER со скоростью симулятора на языке SystemC была создана функционально эквивалентная вышеописанной СБИС модель на языке SystemC.

Под функциональной эквивалентностью мы понимаем не только то, что модели реализуют одинаковую функциональность, но и что в них на этапе симуляции будет инициировано одинаковое количество параллельных процессов.

Так как в SystemC существуют две модели вычислений: SC_THREADS и SC_METHODS [2] и ввиду их существенных различий, было проведено следующее: вышеописанная СБИС была описана на языке SystemC дважды: с использованием SC_THREADS и с использованием SC_METHODS. Далее были запрограммированы аналогичные по поведению модели, средствами языка Caper.

Эксперименты с оценкой скорости показали следующие результаты, которые в виде таблиц и графиков показаны ниже (эксперименты были проведены в операционных системах Windows XP и Red Hat Enterprise Linux 3.0).

Для SC_THREADS:

Количество параллельных процессов	SystemC Время симуляции в секундах	CAPER Время симуляции в секундах
10	1,145s	0,849s
50	2,656s	1,67s
100	3,99s	2,004s
500	4,65s	2,135s
700	5,04s	2,407s
1000	5,416s	3,006s
5000	1min, 1s	20,91s
10000	2min	40,7 s
20000	Hang of simulation	2 min

Таблица 1. Результаты симуляции на основе SC_THREADS

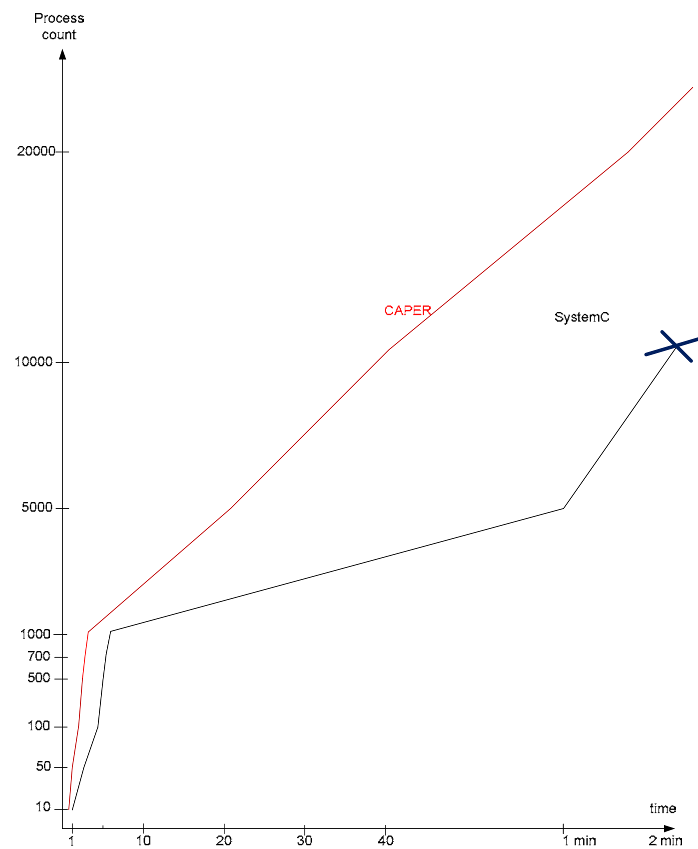

Рис. 4. Результаты симуляции на основе SC_THREADS

Как видно из рисунка, при увеличении количества параллельных процессов скорость симуляции на SystemC резко падает, а после 10000 параллельных процессов программа симуляции в SystemC завершается аварийно, в то время как программа на CAPER продолжает симуляцию (количество процессов доводилось до 200000, практика показывает, что предел определен архитектурными размерами оперативной памяти компьютера).

Анализ причин такого опережения программ на языке Caper по отношению к программам на SystemC показал следующее:

1. Скорость симуляции на SystemC сильно зависит от конкретной вычислительной среды. Это объясняется использованием библиотеки qthreads в качестве поставщика параллелизма, которая, в свою очередь, опирается на средства распараллеливания операционных систем.

В то же время, язык CAPER использует собственный механизм параллельной реализации программ (виртуальную машину), которая позволяет иметь большое количество активных легковесных параллельных процессов (сотни тысяч).

2. Из-за того что SystemC представляет собой надмножество C++, он не предоставляет средств контроля и синхронизации для всех типов данных (в частности, встроенные типы C++ не синхронизируемы), что приводит к определенным затратам во время программирования и реализации программ, и вынуждает производить синхронизацию через средства операционной системы, что замедляет симуляцию. В языке Caper используются собственные средства синхронизации.

Одновременно, были проведены сопоставления и временные замеры симуляции для SC_METHODS.

Количество параллельных процессов	SystemC Время симуляции в секундах	CAPER Время симуляции в секундах
10	1,15s	1,01s
50	2,65s	2,20s
100	3,00s	3,10s
500	5,00s	4,97s
700	5,09s	5,00s
1000	5,71s	5,82s
5000	9,5s	9,91s
10000	30,6s	33,1s
20000	50,3s	57,4s

Таблица 2. Результаты симуляции на основе SC_METHODS

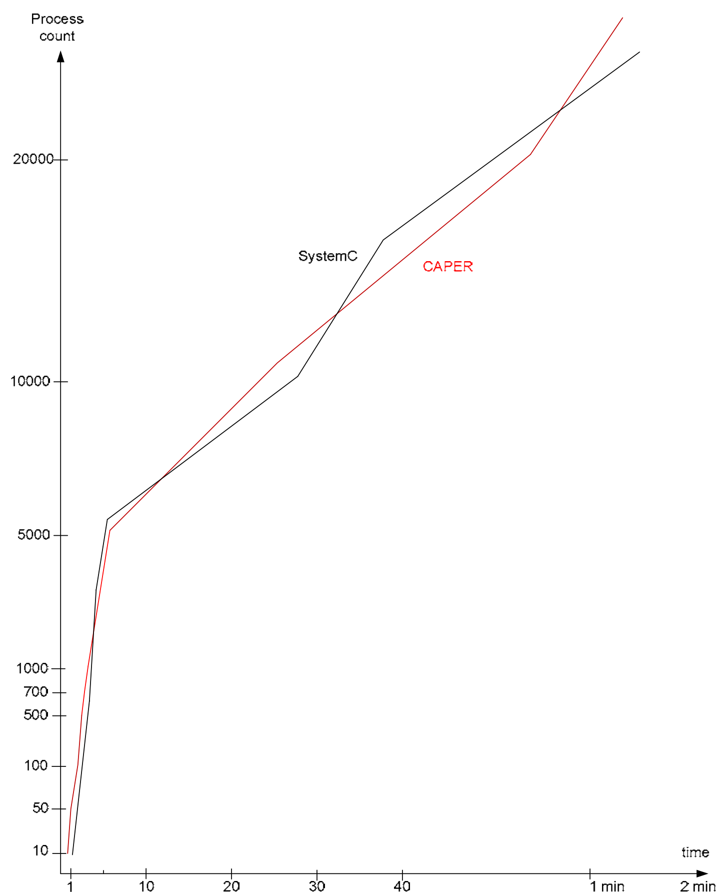

Рис. 5. Результаты симуляции на основе SC_METHODS

Как видно из графика и таблицы, Caper и SystemC имеют почти одинаковые скорости обработки процессов типа SC_METHODS.

Заключение

Таким образом, поставленные в работе цели по сопоставлению средств и методов программирования симуляторов СБИС на распространенном языке SystemC и языке параллельного программирования Caper показали следующее: с помощью языка Caper возможны эффективные решения ряда актуальных задач проектирования СБИС, таких как эффективная симуляция, повышенные уровня абстракции при верификации и описании схем. Более того, было продемонстрировано, что программы симуляции СБИС на языке Caper в отдельных случаях многократно превосходят по быстродействию современные средства программирования симуляторов. Тем самым можно заключить, что язык параллельного программирования Caper более адекватен задачам программирования симуляторов СБИС как в части эффективного описания структур схем, так и в частях адекватного программирования параллельной работы элементов схем и эффективной программной симуляции СБИС.

==================

Несмотря на то, что скорость симуляции на SystemC превышает скорость симуляции на SystemVerilog (компилируемый код против интерпретируемого) скорость симуляции на SystemC сильно зависит от конкретной вычислительной среды. Это объясняется использованием библиотеки qthreads в качестве поставщика параллелизма. Тем не менее, различие в скоростях настолько велико, что симулятор Veriator [], созданный специально для симуляции больших схем на SystemVerilog, добивается скорости посредством перевода описания схемы с SystemVerilog на SystemC.

Литература

1. Thorsten Grötker, Stan Liao, Grant Martin, Stuart Swan. System Design with SystemC. – Kluwer Academic Publishers, New York, 2002.
2. Brian Bailey, Grant Martin and Andrew Piziali. ESL Design and Verification: A Prescription for Electronic System Level Methodology. – Morgan Kaufmann/Elsevier, 2007.
3. Functional specification for SystemC 2.0., www.systemc.org
4. S.R. Vartanov. On Parallel Programming Language Caper. Lect. – Notes in Computer Sci., HCPN-2001, 501-503.
5. Вартанов С.Р. Язык программирования CAPER. – Киев, 1997 (Препр. 97-5, Националь-ная Академия Наук Украины, Институт Кибернетики им. Глушкова).
6. Вартанов С.Р. Методы конструирования и сопровождения данных средствами языка параллельного программирования Caper. – Труды 4-ой международной конференции “Параллельные вычисления и задачи управления” PACO-2008, Москва, 2008
7. S.R. Vartanov. A Mass Parallel Starts In Parallel Programming Language CaperA Mass Parallel Starts In Parallel Programming Language Caper. – Информационные технологии и управление. Том 4-1, Ереван, 2006