УДК 658.512.011:681.326:519.713

И.В. ХАХАНОВА

TLM-МОДЕЛЬ ПРОЦЕССОРА АНАЛИЗА ЛОГИЧЕСКИХ ОТНОШЕНИЙ

Предлагается TLM-модель спецпроцессора, ориентированные на повышение быстродействия анализа логических отношений за счет аппаратной реализации векторных операций. Дается пример анализа прилагательных для образования словоформ в целях построения правильных языковых конструкций русского языка. Спецпроцессор реализован в качестве функционального блока цифровой системы в кристалле FPGA.

Введение

Увеличение функциональной сложности и частоты процессора требует больших затрат энергопотребления, времени проектирования, а также высокой стоимости реализации цифровых систем. Компромиссным решением может быть стратегия проектирования, связанная с делением задач между несколькими ядрами процессора, процессорами, что приводит к созданию параллельных систем, использующих взаимосвязанную совокупность специализированных вычислителей. Такие структуры позволяют повысить производительность решения вычислительных задач, а также уменьшить энергопотребление и стоимость аппаратной реализации цифровых систем.

Особый интерес со стороны рынка электронных технологий вызывают научно-технические направления формализации мыслительной деятельности человека для создания компонентов искусственного интеллекта. Такие интеллектуальные средства, как экспертные системы, распознавание образов и принятия решений нуждаются в гетерогенном подходе при создании эффективных и быстродействующих двигателей (мульти- или специализированных процессоров). Характерным примером предметной области, для которой необходим специализированный процессор, служит анализ и синтез естественных языковых конструкций. При этом одним из центральных моментов проектирования текстовых процессоров является отображение взаимосвязей компонентов, существующих при синтезе и анализе языковых конструкций, в соответствующую аппаратную реализацию.

Цель – разработка TLM-модели спецпроцессора для аппаратной реализации векторных операций .

Задачи:

1. Анализ публикаций в области проектирования специализированных логических процессоров [1-4].
2. Анализ синтаксических и семантических моделей обработки текстов, реализованных на естественных языках [5-6].
3. Разработка архитектуры специализированного процессора для обработки логической сети анализа языковых конструкций [6].
4. Аппаратная имплементация TLM-модели грамматического анализа имен прилагательных.

В качестве прототипа разработки используется, реализовано в FPGA, специализированное устройство, которое выполняло грамматический анализ имен прилагательных [6]. Предлагаемая ниже модель обладает универсальностью и позволяет обрабатывать любые логические сети синтаксических и семантических отношений. Применение TLM-моделей и методов проектирования позволило сделать акцент на порядке обработки и передачи данных, уйти он несущественных деталей RTL-уровня для сокращения средств проектирования.

1. TLM-модели функциональных модулей цифровых систем на кристаллах

Современные технологии проектирования требуют применения более абстрактных, по сравнению с RTL, моделей описания поведения устройств. Этим условиям удовлетворяют структуры уровня транзакций или TLM-модели, которые оперируют понятиями передачи

данных между подсистемами SoC. На протяжение всего цикла проектирования SoC TLM обслуживает три стратегии: ранняя разработка программного обеспечения, архитектурный анализ, функциональная верификация. В совокупности они позволяют существенно повысить производительность средств моделирования и уменьшить время моделирования системы.

Практически полезная классификация моделей, разработанная Данном Гайски (Dan Gajski) и Луцием Каем (Lucai Cai), была представлена на конференции CODES (HW/SW Co-design Conference) в 2003 г. [1]. Согласно этой классификации основной концепцией процесса проектирования является независимая разработка средств коммуникации и функциональных блоков (подсистем). При этом при описание коммуникаций и функциональностей становится инвариантным по отношению к понятию времени (un-timed - UT). Кроме того, может использоваться аппроксимированное время (approximately-timed - AT) или синхронизированное модельное время (cycle-timed - CT)(рис. 1).

На одном полюсе такой классификации находится модель регистровой передачи данных (RTL), которая описывает коммуникации и функционирование системы по тактам. На другом - системная архитектурная модель (SAM), в которой вообще не используется понятие времени при представлении коммуникаций и функций. В зависимости от требований к системе модель с аппроксимированным временем может применять статистическую оценку, расчетное время или основываться на временных параметрах подсистем.

Модель с потактовым описанием коммуникаций и аппроксимированной оценкой времени для блоков называется функциональной моделью шины (Bus Functional Model - BFM). Три остальные модели в классификации авторов получили названия:

1. Модель объединения компонентов – коммуникации без использования времени, функциональность с аппроксимированным временем.
2. Модель арбитража шины - коммуникации с аппроксимированным временем, функциональность с аппроксимированным временем.
3. Вычисления с точным временем - коммуникации с аппроксимированным временем, функциональность с точностью до такта.

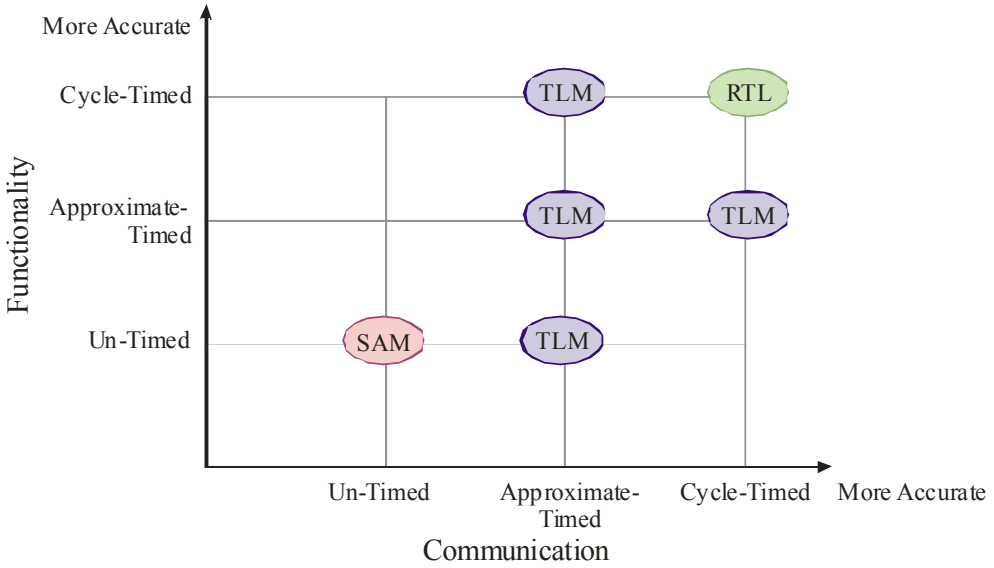


Рис. 1. Классификация моделей цифровых систем

Использование TLM для оценки возможностей SoC позволяет проектировщику выполнить начальный анализ системы до проектирования RTL или более точный анализ, имеющий место в описании временных параметров моделей. Ранняя оценка архитектуры и функциональности устройства дает возможность верифицировать реальное программное обеспечение на уровне TLM, до появления RTL-структуры, что позволяет уже на этих этапах выполнять существенную корректировку функциональных модулей hardware и software.

Для функциональной верификации системного уровня традиционно использовалось времязатратное моделирование Verilog или VHDL архитектур, в том числе и с применением стратегии эмуляции. По мере увеличения сложности проектируемых систем их обработка с помощью языков-симуляторов Verilog и VHDL существенно замедляет процесс проектирования. Поэтому моделирование уровня транзакций является решением задачи повышения быстродействия верификации.

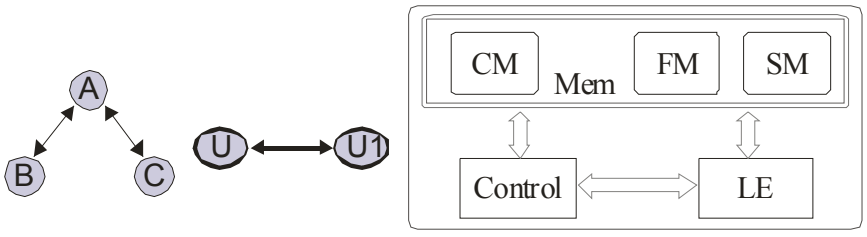
Кроме того, использование классов в аппаратных языках дает возможность повысить производительность ручного построения моделей путем применения объектно-ориентированных методов.

2. Модель спецпроцессора обработки логических отношений

Спецпроцессор предназначен для обработки логической сети, где вершинами выступают множества признаков A, B, C (рис. 2,а), а дугами – функциональные отношения между признаками, заданные в табличной форме. Например, это могут быть отношения между признаками словоформ «ударность» - «тип основы» из логической сети прилагательных [6] (рис. 2,б), где множества U ={б, у}, U1 = {ветхий, куцый, синий, седой, слабый, сухой, рыжий}, а функциональное отношение между ними представлено двоичной таблицей:

	ветхий	куцый	синий	седой	слабый	сухой	рыжий
б	1	1	1	0	1	0	1
у	0	0	0	1	0	1	0

Структура специализированного спецпроцессора представлена на рис. 2,в и состоит из трех основных блоков: управляющего (Control), логического (LE) и блока памяти (Mem). В свою очередь, память включает блоки хранения команд (CM – command memory); функций (FM – function memory), множеств (SM – set memory).



а б в

Рис. 2. Блок-схема спецпроцессора

Преимуществом предлагаемой архитектуры спецпроцессора является то, что он может обрабатывать логические сети любой размерности и не зависит от длины векторов множеств и размеров таблиц отношений. Ограничения связаны только физическими размерами памяти.

Логический блок принимает данные порциями фиксированного размера. Если таблица отношений или векторы множеств имеют большую длину, то они разбиваются на части. Таким образом, обработка всей таблицы отношений выполняется путем скроллинга данных с помощью окна (рис. 3), размер которого определяется разрядностью шины, передающей данные между памятью и логическим блоком. Обход таблицы начинается с левого верхнего угла и продолжается вправо вниз. Количество операций для реализации отношения при размерности окна обхода равной NxM определяется формулой $N_R = Nn \cdot Nm$. Каждая таблица отношений, поделенная на блоки в соответствии с размерностью окна обхода, записывается в память в форме одномерного вектора. Аналогично выполняется разбиение векторов множеств на отдельные слова памяти, при этом множества записываются в память фрагментами-словами (рис. 4). Два блока памяти, выделенных для хранения множеств и функций, дают возможность считывать за один такт фрагменты таблицы отношений и векторов множеств в целях последующего выполнения над ними элементарных логических операций.

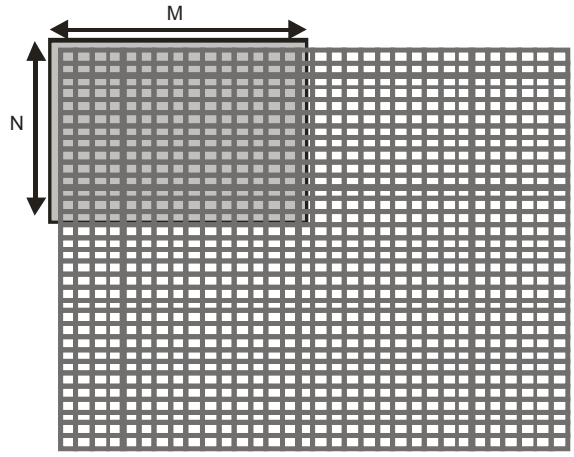


Рис. 3. Принцип обработки матрицы отношений

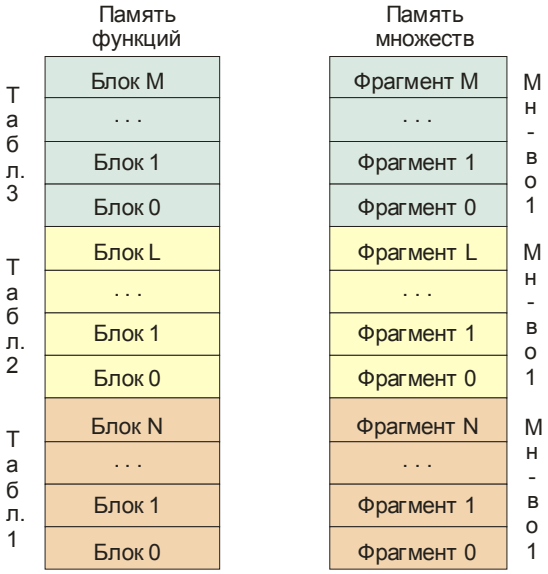


Рис. 4. Хранение информации о логической сети в памяти

Например, при окне обхода размерностью 4x4 таблица функции отношений «Окончание» – «классы окончаний» (x-y) логической сети анализа прилагательных [6] будет разбиваться на 28 блоков, как это показано на рис.5. Это означает, что для ее обработки потребуется выполнить 28 элементарных операций, составляющих цикл обработки таблицы.

Далее рассматривается анализ отношения между формами слов «ударность» - «тип основы». Пусть тип основы описывается множеством R1 = { ветхий, куцый, слабый, сухой, рыжий}, а ударность – множеством R2 = {б}. В блоке памяти множеств будет храниться двоичный вектор, соответствующий множеству признаков, где 1 означает присутствие элемента, а 0 – его отсутствие. Тогда при окне обхода размерностью 4x4 и учитывая, что длина векторов R1 и R2 равна 7 и 2 элементам, соответственно, вектор R1 будет занимать два слова памяти

R1	1	1	0	0	1	1	1	

, а вектор R2 – одно

R2	1	0		

. Для хранения системы отношений в пределах одной таблицы необходимо два слова памяти разрядностью 16 битов – таблица разбивается на два блока:

	ветхий	куцый	синий	седой	слабый	сухой	рыжий	
б	1	1	1	0	1	0	1	
у	0	0	0	1	0	1	0	
		1				2		

	ая	ого	ой	ое	ом	ому	ою	ую	ый	ые	ым	ыми	ых
ого	0	1	0	0	0	0	0	0	0	0	0	0	0
ому	0	0	0	0	0	1	0	0	0	0	0	0	0
ом	0	0	0	0	1	0	0	0	0	0	0	0	0
ое	0	0	0	1	0	0	0	0	0	0	0	0	0
ой	0	0	1	0	0	0	0	0	0	0	0	0	0
ою	0	0	0	0	0	0	1	0	0	0	0	0	0
его	0	1	0	0	0	0	0	0	0	0	0	0	0
ему	0	0	0	0	0	1	0	0	0	0	0	0	0
ем	0	0	0	0	1	0	0	0	0	0	0	0	0
ее	0	0	0	1	0	0	0	0	0	0	0	0	0
ей	0	0	1	0	0	0	0	0	0	0	0	0	0
ею	0	0	0	0	0	0	1	0	0	0	0	0	0
ую	0	0	0	0	0	0	0	1	0	0	0	0	0
ая	1	0	0	0	0	0	0	0	0	0	0	0	0
ым	0	0	0	0	0	0	0	0	0	0	1	0	0
ый	0	0	0	0	0	0	0	0	1	0	0	0	0
ые	0	0	0	0	0	0	0	0	0	1	0	0	0
ых	0	0	0	0	0	0	0	0	0	0	0	0	1
ыми	0	0	0	0	0	0	0	0	0	0	0	1	0
им	0	0	0	0	0	0	0	0	0	0	1	0	0
ий	0	0	0	0	0	0	0	0	1	0	0	0	0
ие	0	0	0	0	0	0	0	0	0	1	0	0	0
их	0	0	0	0	0	0	0	0	0	0	0	0	1
ими	0	0	0	0	0	0	0	0	0	0	0	1	0
ою	0	0	0	0	0	0	0	1	0	0	0	0	0
яя	1	0	0	0	0	0	0	0	0	0	0	0	0

Рис. 5. Система отношений для множеств «Окончание» и «Классы окончаний»

Карта памяти для хранения векторов и множеств рассмотренного выше примера представлена на рис 6. Память команд имеет следующий формат слова данных:

size_W	size_H	addr_R0	addr_R1	addr_R2

где size_W и size_H задают размер таблицы отношений, а addr_R0, addr_R1, addr_R2 – начальные адреса данных. Для представленного выше примера и согласно рис. 6 команда будет иметь вид

2	7	0	0	2

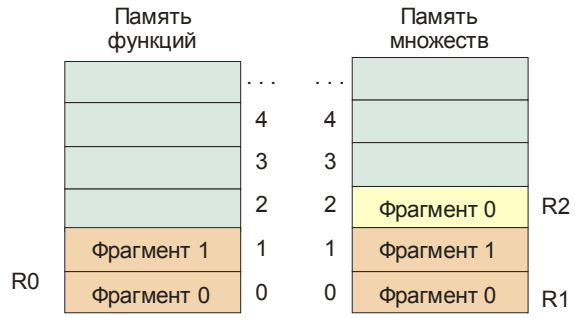


Рис. 6. Память для хранения данных об отношении «ударность» - «тип основы»

3. Создание TLM-модели процессора обработки логических отношений

По классификации [1] предлагаемая TLM-модель спецпроцессора имеет аппроксимированное время для передачи данных (для коммуникаций) и для внутренней работы блоков (функций) и, следовательно, является моделью с арбитражем шины.

Для разработки модели спецпроцессора был применен объектно-ориентированный подход. Отдельные блоки устройства описываются в виде классов, для связи между блоками используется виртуальный интерфейс. В процессе дальнейшего проектирования классы по отдельности могут быть заменены описанием RTL-уровня, а виртуальный интерфейс - обычным.

Для разработки TLM-модели используются объектно-ориентированные средства языка описания и верификации аппаратуры SystemVerilog. Структура классов представлена рис. 7.

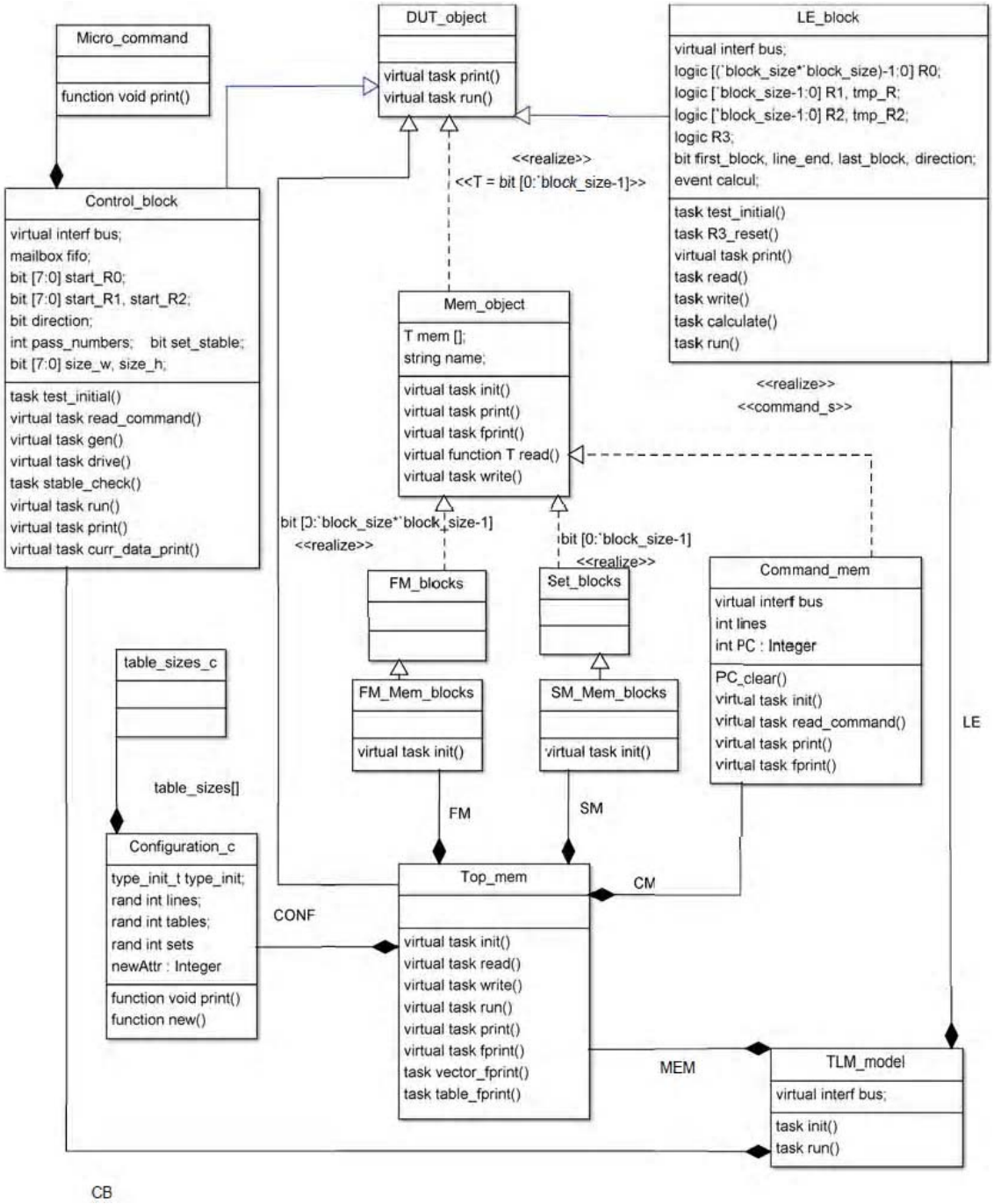


Рис.7. Структура классов

Класс TLM_model включает экземпляры объектов трех классов: Control_block, LE_block, Top_mem, которые соответствуют трем блокам структурной схемы устройства: управляющему (Control), логическому (LE) и блоку памяти (Mem) (см. рис. 2).

Базовым классом, от которого наследуются все остальные классы компонентов, является класс DUT_object. Класс Configuration предназначен для управления генерацией псевдослучайных тестовых данных, применяемых для отладки системы.

Класс Micro_command используется для представления команд процессора. Память строится на основе класса-шаблона Mem_object, который параметризируется типом дан-

ных T. Изменение типа данных при создании класса наследника позволяют создавать модели памяти различной конфигурации для создания трех блоков памяти модели спецпроцессора.

В классах, описывающих память, реализованы методы: инициализации исходного состояния экземпляра класса (read и write); вывода данных в консоль (print) и вывода данных в текстовый файл (fprint); запуска работы объекта. Кроме названных методов управляющий блок, представленный классом control_block, имеет также: чтение команды (read_command), генерация последовательности управляющих сигналов для памяти и логического блока (gen), передача управляющих сигналов в интерфейс (drive). Функция stable_check() проверяет момент завершения работы обработки сети.

Экземпляр класса TLM_model реализуется в модуле Top (листинг 1), который также подключает интерфейс bus, моделирующий работу шины специализированного процессора. SystemVerilog – код интерфейса представлен листингом 2. Он содержит поля для передачи данных, а также события, синхронизирующие работу отдельных блоков системы.

Листинг 1. Модуль верхнего уровня, реализующий TLM-модель устройства

```
Top.sv
‘include “TLM_model.sv”
module Top;
 //Interface
 interf bus();
 //Testbench(Memories)
 TLM_model TLM;
   initial begin
    TLM=new(bus);
   TLM.init();
   TLM.run();
  end
endmodule
```

Листинг 2. SystemVeriolog код интерфейса

```
interface interf;
 bit clk = 0;
 initial  forever #5 clk = ~clk;
  //command
  event command_request;
  event command_resp;
  event PC_clear_e;
  event pass_end; // pass all commands
  event work_stop; //
 bit [7:0] size_r1, size_r2;
 bit [7:0] start_R0, start_R1, start_R2;
   clocking cb@(posedge clk);
  // output size_r1, size_r2;
   // output start_R0, start_R1, start_R2;
 endclocking: cb
   always @(pass_end)
        $display(“1 pass command memmory have finished”);
  // memmory
  event data_read, data_send, pass_start;
 bit [7:0] addr_R0, addr_R1, addr_R2;
 bit first_block, line_end, last_block, direction;
 logic [(‘block_size*‘block_size)-1:0] R0;
 logic [‘block_size-1:0] R1;
 logic [‘block_size-1:0] R2;
 bit R3;
   event data_write;
 bit [7:0] addr_R2_w;
 logic [‘block_size-1:0] R2_w;
   // stop
 event simul_finish;
endinterface
```

4. Выводы

Научная новизна. Разработана TLM-модель спецпроцессора для логического анализа имен прилагательных. Модель позволяет имплементировать в аппаратуру любой граф логических отношений анализа конструкций естественного языка. Данное устройство требует больше времени на обработку системы, чем представленное ранее [6], однако превосходит его в универсальности.

Для уменьшения времени проектирования функциональных модулей были использованы технологии объектно-ориентированного программирования языка SystemVerilog для создания модели TLM-уровня. Современный подход к разработке цифровых систем позволил существенно повысить производительность процесса проектирования.

Дальнейшие исследования связаны с созданием генераторов сложных архитектур логических отношений, обладающих возможностью объединения нескольких логических блоков для повышения быстродействия работы функциональности.

Список литературы: 1. *David C. Black, Jack Donovan*. SystemC: from the ground up. Kluwer Academic Publishers, 2004. 263 p. **2.** *Bergeron, Janick.* Writing testbenches: functional verification of HDL models. Boston: Kluwer Academic Publishers, 2001. 354 c. 3. Donald E. Thomas. Philip R. Moorby. The Verilog Hardware Description Language.New York, Boston, Dordrecht, London, Moscow: Kluwer Academic Publishers. 2002. 404 p. **4.** *Хаханов В.И., Хаханова И.В.* и др. Verilog&SystemVerilog. Харьков: Новое слово, 2010. 528с. **5.** Общая алгебра. Справочник. Т.2. Под ред. Скорнякова. 1991. Глава 7. Категории. С.368-460. **6.** *Bondarenko M.F., Hahanova I.V*. Logic networks application for computing process organization. Радиоэлектроника и информатика, 2003. С. 150-156.

Поступила в редколлегию 11.03.2012

Хаханова Ирина Витальевна, д-р техн. наук, профессор кафедры АПВТ ХНУРЭ. Научные интересы: проектирование и верификация цифровых систем и сетей, цифровая обработка сигналов. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, E-mail: hahanova@mail.ru