УДК 621.374.3; 004.4:004.9

Перепроектирование ПЛИС на основе трансформации моделей

Д.И. Черемисинов

Объединенный институт проблем информатики НАН Беларуси, cher@newman.bas-net.by

***Аннотация* — Рассматривается задача конвертации цифровой системы, реализованной на микросхемах FPGA семейства Spartan 3, в VHDL-описания, пригодные для синтеза в библиотеках проектирования заказных СБИС. Программа для решения этой задачи построена как средство автоматизации проектирования, управляемое моделями.**

***Ключевые слова* — FPGA, ASIC, перепроектирование, система перезаписи графов.**

I. Введение

По сравнению с заказными СБИС (ASIC) проектирование для программируемых пользователем вентильных матриц (FPGA), быстрее, с меньшими рисками разработки и дешевле (нет невозвращаемых затрат – non-recurring engineering charges, NRE). В то же время FPGA экономически не выгодны для массового производства дешевых устройств. Поэтому FPGA часто используются как средство прототипирования заказных СБИС. Создание FPGA прототипа позволяет снизить NRE ценой усложнения маршрута разработки заказной СБИС включением этапов проектирования FPGA и этапа *перепроектирования* (reverse engineering) FPGA на ASIC.

Перепроектирование является инверсией обычной разработки в смысле порядка процесса преобразований; его задача заключается в построении спецификации, анализируя продукт. Результат этапа проектирования на основе FPGA – bitstream – представляет собой структурное описание логической сети, заданное в формате, который является технологическим секретом фирмы производителя FPGA. При перепроектировании FPGA строится функциональное представление, пригодное для синтеза заказной СБИС, из bitstream или описания, из которого непосредственно строится bitstream. Исходное для проектирования FPGA функциональное описание нецелесообразно использовать для синтеза заказной СБИС, так как это приведет к потере результатов прототипирования.

Проектирование, управляемое моделями, (*model-driven development*) – это стиль проектирования (как правило, программного обеспечения), когда модели становятся основными артефактами (продуктами деятельности) разработки, из которых генерируются другие артефакты [1]. Под названием MDA (Model Driven Architecture) этот стиль обозначает технологию программирования, широко применяемую для разработки коммерческих программных продуктов. Термин «разработка, управляемая моделями», иногда используется для описания неавтоматизированных, но модельно-ориентированных подходов. В этой работе термин «управляемый моделями», обозначает подход, в котором применяется автоматизация. В стиле перепроектирования FPGA «разработка, управляемая моделями», разработчику предоставляется автоматизированный инструмент – программа, выполняющая преобразования моделей. Описание архитектуры этой программы и является предметом статьи.

В процессе проектирования устройств на основе Xilinx FPGA результирующее структурное описание представлено в формате NCD (Native Circuit Description) [2], который является технологическим секретом Xilinx. Для доступа к структурному представлению на этом уровне Xilinx предлагает текстовый формат XDL, частично описанный в [2]. Формат XDL является плоским структурным описанием в виде списка цепей, связывающих макроячейки FPGA. В перепроектировании, управляемом моделями, представление в формате XDL описывает исходную модель (analysis model). Результатом (artifact) перепроектирования является функциональная спецификация на языке VHDL. Таким образом, XDL является нотацией исходной модели, VHDL – это нотация результата разработки.

II. Модель структуры

Моделью в формате XDL структурного описания, запрограммированного в FPGA, выбран двудольный граф, одной долей которого являются порты (выводы) элементов FPGA – примитивов, а второй долей – цепи, соединяющие порты.

В качестве примера на рис. 1 изображен фрагмент логической схемы секции SLICEL, а на рис. 2 – модель этого фрагмента в виде неориентированного двудольного графа. В данном фрагменте XORG – логический элемент сумма по модулю 2, GYMUX – трехвходовый коммутационный элемент, DYMUX – двухвходовый коммутационный элемент.

На рисунках графов вершины, являющиеся выводами элемента, сгруппированы в прямоугольники (са-

ми прямоугольники не являются элементами графа). Обозначения выводов предваряются именем элемента, показанным на рисунках в фигуре элемента (см. рис. 2). Вершины цепей показаны кружками. На рис. 2 обозначения (имена) выводов и некоторых цепей для упрощения рисунка опущены.

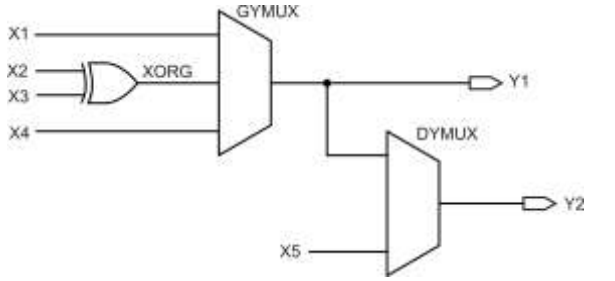

Рис. 1. Логическая схема (фрагмент секции SliceL)

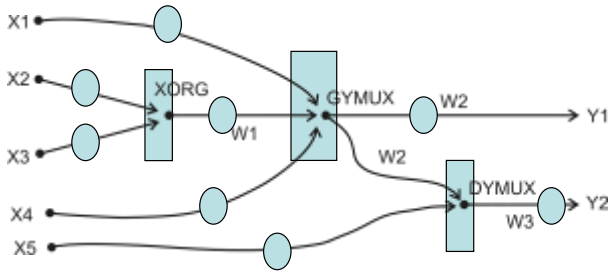

Рис. 2. Двудольный граф, соответствующий логической схеме

III. Трансформации моделей

Проектирование, управляемое моделями, концентрируется на решении задачи последовательностью шагов, представляющих собой трансформации моделей. Модели в памяти компьютера представлены некоторыми структурами данных. Таким образом, решение задачи состоит из, по крайней мере, трех шагов: преобразования текстового описания во внутреннее представление модели – структуру данных в памяти, трансформации исходной модели, и генерацию текстового описания по модели, являющейся результатом трансформации (рис. 3).

Для того чтобы описать трансформацию модели, в свою очередь, может быть использована модель, называемая моделью трансформации. Эту модель удобно задавать в операционной форме как некоторый алгоритм, обрабатывающий структуру данных в памяти. Шаги этого алгоритма представляют собой правила трансформации.

В качестве алгоритмического базиса модели трансформации удобно использовать подстановки графов (graph rewriting rule) – локальные трансформации графа с использованием правил перезаписи графов [3]. Алгоритмические системы на основе правил переписывания GRT (Graph Rewriting and Transformations) позволяют описывать любые алгоритмически представимые вычисления, так как в их основе лежит алгоритмически полная система нормальных алгоритмов Маркова. Функциональные возможности этой системы сосредоточены в использовании единственной операции, которая в теории алгоритмов Маркова называется *подстановкой слов*.

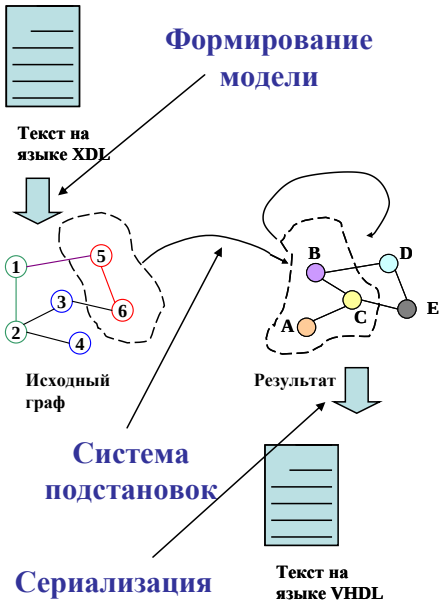

Рис. 3. Перепроектирование FPGA как разработка, управляемая моделями

В целом перепроектирование FPGA состоит из трех этапов. Сначала по описанию, из которого непосредственно получается файл программирования, строится двудольный неориентированный граф. Затем этот граф трансформируется в более детальное структурное описание. На последнем этапе перепроектирования FPGA уточненное структурное описание дополняется функциональными описаниями составляющих элементов, что превращает структурное описание в функциональное. Соответствующее преобразование – это оценка структурного описания. Фаза оценки представляет собой сериализацию (преобразование в текстовую форму) графа, полученного в результате второй фазы перепроектирования, в процессе его обхода.

IV. Перезапись графов

В данной ситуации наиболее подходящей является форма перезаписи графов с одиночным вытеснением (*Single PushOut - SPO*) [3]. В соответствии с данным подходом подстановка p: $L \rightarrow R$ состоит из шаблонного графа L (левой части правила), замещающего графа R (правой части правила) и частичного графового гомоморфизма $\rightarrow$ между L и R. Применение правила p к основному графу H называется *непосредственным выводом*, обозначаемым как $p::H \rightarrow H'$. Это требует частичного графового гомоморфизма $m:L \rightarrow H$, называемого *согласованием* (*match*). Согласование фиксирует вхождение L в граф H. Частичный морфизм описывает отношения объектов левой части к правой, детализируя процесс применения правила. Объекты, не описываемые морфизмом, удаляются, объекты, не изменяемые морфизмом, формируют контекст продук-

ции, объекты правой части продукции, не имеющие прообраза в левой части, создаются заново.

При применении правила *p* производится вывод графа *H'* из *H* путем замены вхождения *L* в *H* на *R*. Иными словами, подстановка *p* определяет частичное отношение между элементами ее левой и правой частей, определяя какие элементы *H* сохраняются, удаляются или создаются при ее применении. Чтобы применить *p* к графу *H*, нужно найти согласование *m:L→H,* отображающее каждый элемент *L* в элемент *H*. Применение *p* к графу *H* при согласовании *m* удаляет все элементы графа *H*, которые согласуются с элементами *L*, создает каждый элемент *R* в *H* и сохраняет все остальные элементы *H*. Следует отметить, что для шаблонных графов общего вида поиск согласования подграфов является *NP*-полной задачей.

Проектирование систем, управляемое моделями, основано на преобразования моделей в другие модели. Для создания преобразований требуются инструменты автоматизации их разработки. Мощной методикой для задания сложных преобразований является формализм GRT. Практическое использование формализма GRT зависит, с одной стороны, от существования математически обоснованного и удобного в работе программного обеспечения, и, с другой стороны, опирается на опыт использовании методики применения теории к задачам реального мира. После появления технологии разработки программ, управляемой моделями – MDA [4], GRT стала технологией, которая широко используется в промышленности разработки программного обеспечения. Известны ряд инструментальных сред MDA для разработки программ трансформации графов на основе GRT [5].

V. Система перезаписи графов

Известные в технологии MDA средства создания программ трансформации графов ориентированы на обработку визуальных представлений языка UML. В исследуемой задаче графы являются не визуальным представлением, а математическим формализмом. Для выполнения трансформаций графа структурного описания FPGA разработана специализированная система перезаписи графов. Эта GRT выполняет операцию замены структуры связи примитивов FPGA структурой связи составляющих примитив логических элементов – распаковку исходного структурного описания. В распакованном описании название типа элементов позволяет определить отношение вход-выход элемента.

A. *Подстановка распаковки*

Шаблонным графом L подстановки p: L→R распаковки примитива FPGA типа N (рис. 4) является граф, множество вершин которого состоит из выводов элемента N, а множество ребер пустое. Правая часть R подстановки p является двудольным графом, построенным в результате синтаксического анализа подсекции *N* секции *primitive_defs* из файла структурного описания микросхемы FPGA (с расширением .XDLRC).

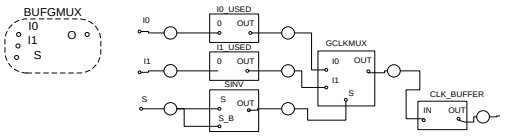

Рис. 4. Правило переписывания для распаковки элемента FPGA Spartan3 типа BUFGMUX

Структурное описание – netlist – микросхемы FPGA и описание – netlist – структуры, запрограммированной в FPGA, задаются на языке XDL [2], но эти описания отличаются по синтаксису и структуре. Netlist программирования в FPGA (user-design netlist), является описанием в виде списка цепей. Netlist микросхемы FPGA (FPGA fabric netlist) содержит раздел, называемый *primitive_defs*. В этом разделе имеются структурные описания примитивов FPGA в виде списка логических элементов (рис. 5). Список экземпляров задает структуру соединений, используемых в устройстве, указанием для каждого экземпляра элемента пар, содержащих вывод экземпляра и цепь, с которой связан этот вывод – списка подключения выводов элементов (port map).

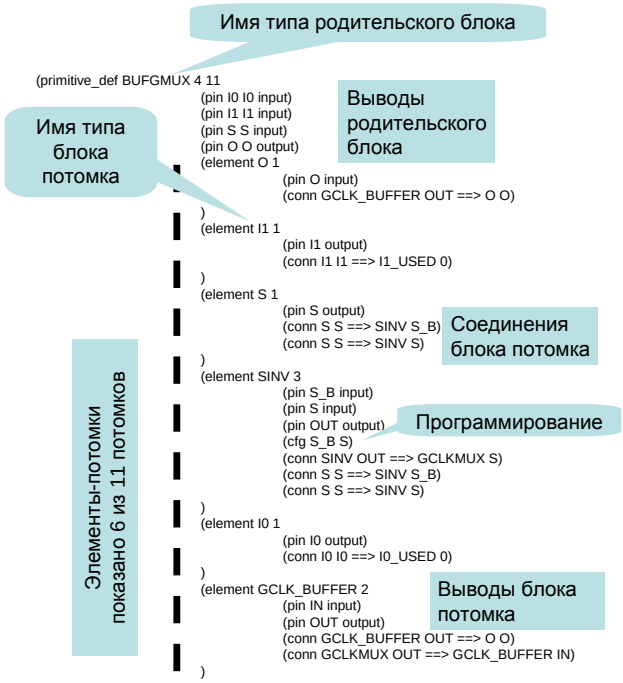

Рис. 5. Описания элемента BUFGMUX в файле описания микросхемы

Граф, показанный на рис. 7, демонстрирует результат применения правила перезаписи к графу на рис. 6. Этот граф получен склеиванием вершин правой части правила на рис. 3 к подграфу исходного графа, найденному с помощью левой части правила. Затем все вершины, полученные в результате склеивания, удаляются и склеиваются вершины соседних цепей.

Применение правила перезаписи – замена структурного описания программирования FPGA также структурным описанием, но более детализированным.

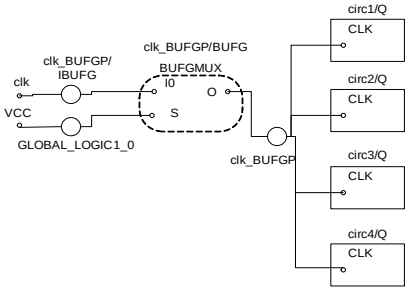

Рис. 6. Исходный граф

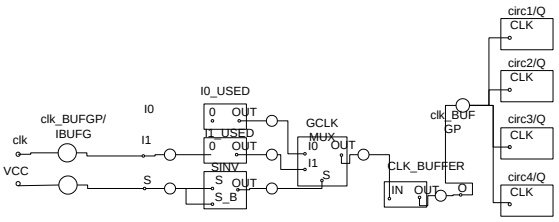

Рис. 7. Результат применения правила переписывания

Netlist программирования в FPGA строится программой xdl, входящей в состав САПР Xilinx. Программа xdl имеет опции, позволяющие построить netlist микросхемы FPGA – файл отчета (с расширением .XDLRC).

B. Подстановка запрограммированного элемента

Описание каждого экземпляра примитива FPGA (instance statement) в формате XDL содержит строку конфигурации, которая начинается с ключевого слова cfg. Строка конфигурации содержит несколько подстрок в формате name:logical name:value и каждая из этих подстрок описывает конфигурирование компонентов внутри описаний примитива. Член *name* указывает тип логического элемента (element в структурном описании элемента и primitive в файле отчета). Член *logical name* задает имена экземпляров элементов, а член value – программирование элемента.

В правиле перезаписи запрограммированного примитива часть R строится не по секции примитива из описания микросхемы FPGA, а по описанию экземпляра этого примитива из описания структуры, запрограммированной в FPGA [6]. Пример процесса формирования части R правила перезаписи из раздела конфигурации экземпляра примитива SLICEL, показанного ниже, приведен на рис. 8.

```
cfg " BXINV::BX BYINV::#OFF CEINV::#OFF
CLKINV::CLK COUTUSED::#OFF CY0F::#OFF
CY0G::#OFF CYINIT::#OFF CYSELF::#OFF
CYSELG::#OFF DXMUX::1 DYMUX::#OFF
F:circ0/circ1/d31:#LUT:D=(A1*(A4*(~A3+~A2)))
F5USED::#OFF FFX:circ3/Q:#FF
FFX_INIT_ATTR::INIT0 FFX_SR_ATTR::SRLOW
FFY::#OFF FFY_INIT_ATTR::#OFF
FFY_SR_ATTR::#OFF FXMUX::F5
FXUSED::#OFF
```

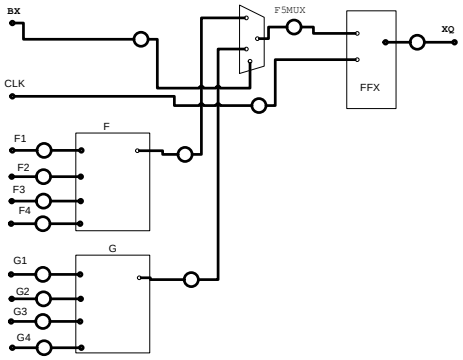

Рис. 8. Правая часть правила перезаписи экземпляра примитива SLICEL

Всего в этом экземпляре SLICEL используются восемь компонент: BXINV, CLKINV, DXMUX, F, FFX, FXMUX, G, F5MUX, из которых только четыре имеют не тривиальное программирование.

C. Программирование LUT

Роль основного логического элемента в примитивах FPGA SLICEL и SLICEM играет логическая таблица (LUT – look-up table). Элемент LUT имеет сложное внутреннее устройство: LUT может работать в режиме логической таблицы, в режиме сдвигового регистра, или в режиме оперативной памяти. Режим работы LUT задается в строке конфигурации значением параметра программирования: «#LUT», «#SHIFT-REG» или «#RAM». Элемент LUT является комбинационной схемой, если значение параметра его программирования есть «#LUT». В этом случае его функция задана в строке конфигурации (рис. 9).

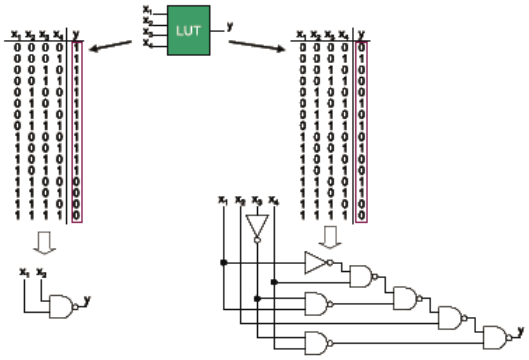

Рис. 9. Правые части правила переписывания при программировании LUT в режиме логической таблицы

LUT может быть сконфигурирована как 16-битовое синхронное ОЗУ. Две соседних LUT могут быть сконфигурированы как 16-битовое двухпортовое ОЗУ с записью и чтением по одному адресу и чтением по другому адресу. При этом для реализации синхронного режима записи входной бит данного, сигнал записи и адрес запоминаются в триггерах, а для чтения по второму адресу из блока второй LUT используется только мультиплексор чтения.

Можно конфигурировать LUTы как сдвиговые регистры. Это дает огромную экономию по сравнению со сдвиговыми регистрами, построенными из триггеров: один LUT может реализовать 16-битовый сдвиговый регистр. Целая макроячейка Spartan-3 может реализовать 32-битовый сдвиговый регистр. Когда LUT используется в режиме SRL, его структура представляет собой сдвиговый регистр, связанный с мультиплексором (рис. 11).

LUT в режиме сдвигового регистра реализует линию задержки на 1-16 тактов синхронизации. Число тактов задержки задается динамически значениями сигналов на линиях A0-A3. LUT может представлять собой и сдвиговой регистр фиксированной длины, если значения сигналов на линиях A0-A3 заданы константами. Работу LUT в режиме сдвигового регистра обеспечивают несколько элементов примитива SLICEM, показанных на рис. 10.

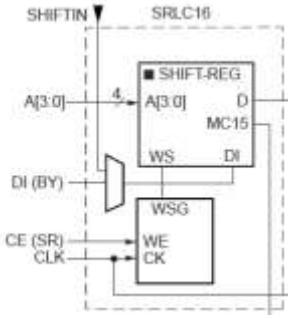

Рис. 10. Левая часть правила переписывания при программировании LUT в режиме сдвигового регистра

На этом же рисунке показано соответствие сигналов SLICEM сигналам сдвигового регистра. Этот рисунок задает левую часть правила переписывания. Правая часть правила переписывания LUT в режиме сдвигового регистра приведена на рис. 11.

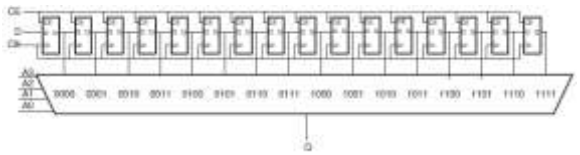

Рис. 11. Замена LUT при программировании LUT как сдвигового регистра

D. Константы

Примитивы CLB могут быть источником логических констант, например

```
inst "XDL_DUMMY_CLKB_VCC_X22Y0" "VCC",placed CLKB
VCC_X22Y0  ,
cfg "_NO_USER_LOGIC:: _VCC_SOURCE::VCCOUT "
```

Подстрока _NO_USER_LOGIC указывает, что эта секция не относится к рассматриваемому устройству и служит источником константы 1 (0), если именем элемента в следующей подстроке конфигурации является _VCC_SOURCE (_GND_SOURCE).

E. Триггеры

Примитивы SLICEL и SLICEM содержат по два триггера: FFX и FFY. Оба триггера функционально одинаковы. Функционирование триггеров задается программированием специальных управляющих элементов (FFX_INIT_ATTR, FFX_SR_ATTR для триггера FFX) и элемента SYNC_ATTR, который конфигурирует оба триггера. Строка конфигурирования экземпляра триггера задает тип триггера: FF (Flip-Flop) – триггер, управляемый фронтом тактового сигнала; Latch – триггер с потенциальным управлением (защелка) задается соответствующим значением в строке конфигурации триггера.

Элемент SYNC_ATTR задает тип сброса пары триггеров (RESET TYPE). Для FF он может быть синхронный (SYNC) либо асинхронный (ASYNC), для Latch – только асинхронный. Элемент FFX_INIT_ATTR (FFY_INIT_ATTR) программирует начальную установку (триггеры в FPGA устанавливаются в начальное состояние после этапа конфигурации FPGA). Значение программирования INIT0 (INIT1) вызывает установку в логический нуль (единицу). Если начальная установка не задана, тогда по умолчанию выбирается вариант INIT0. Элемент FFX_SR_ATTR (FFY_SR_ATTR) программирует уровень сброса, указывает состояние триггера после сброса либо в нуль (SRLOW), либо в единицу (SRHIGH), по умолчанию устанавливается в нуль. Все триггеры имеют пять входных портов (D,CE, CK, SR,REV) и один выходной (Q), который используется всегда. Из входных портов обязательными являются два (D и CK), с помощью которых можно построить триггер с минимальной функциональностью.

При выполнении подстановки распаковки примитивов SLICEL и SLICEM информация о конфигурировании триггера сохраняется в атрибутах портов.

Триггеры содержатся также в примитивах входных буферов IOB. Их программирование задается в конфигурационной строке аналогично конфигурационной строке SLICEL и SLICEM.

VI. О НАДЕЖНОСТИ ПРЕОБРАЗОВАНИЯ В VHDL

Алгоритм распаковки работает с неофициальным (недокументированным) описанием в формате XDL. Проблема распаковки формата XDL рассматривается в диссертации [8] (приложение В) в связи с задачей разработки инструментов трассировки и размещения элементов FPGA Xilinx альтернативных «фирменным». В этой задаче включение альтернативных инструментов в маршрут проектирования Xilinx происходит после выполнения «фирменных» инструментов. Трассированное и размещенное структурное описание распаковывается до уровня логических элементов. Формой представления распакованного описания является снова формат XDL, в такой форме оно обрабатывается (возможно, после ручной корректировки с целью устранения ошибок) альтернативными инструментами. Следствием выбора такого маршрута проектирования является требование «реверсивности» операции распа-

ковки. Результат распаковки должен быть представлен в форме, допускающей его обратное преобразование программой *xdl* в формат NCD. Главной проблемой обеспечения реверсивности является проблема именования типов распакованных элементов, потому что именование, предлагаемое в netlist микросхемы (FPGA fabric netlist), для этого не годится.

Проверка надежности разработанного алгоритма распаковки была выполнена путем эксперимента. Очевидно, эта проверка не может быть формальной, она выполнялась оценкой соответствия построенного в результате распаковки VHDL-описания с «эталонным» описанием. Нет необходимости выполнять эту проверку постоянно, достаточно выполнить ее один раз при разработке программы.

В качестве эталонного представления можно использовать структурное описание, являющееся результатом синтеза. Инструментом синтеза в САПР FPGA Xilinx является программа XST (Xilinx Synthesis Technology). Результатом работы XST является файл с расширением NGC, представляющий структурное описание, предназначенное для использования другими инструментами САПР FPGA Xilinx. Структура формата NGC не опубликована в документации Xilinx, и поэтому непосредственно результат синтеза не может быть использован в качестве эталона. Однако в САПР FPGA Xilinx включена программа Netgen, выполняющая перевод формата NGC в VHDL. Ее назначением является представление результата синтеза в форме, воспринимаемой инструментами моделирования.

Сравнение текстов на VHDL, полученных программой Netgen, и разработанной программой распаковки, обнаруживает разницу. Прежде всего, сравниваемые представления отличаются по наименованию элементов. Далее отличаются имена типов элементов и имена их выводов. VHDL-описание, полученное программой Netgen, ориентировано на моделирование на основе библиотеки UNISIMS, и использует имена, определенные в этой библиотеке. Распакованное VHDL-описание содержит имена типов элементов и имена их выводов, заданные в netlist микросхемы. Отличаются и наименования экземпляров элементов и цепей. Таким образом, буквальное сравнение этих двух файлов не позволяет выявить эквивалентность структуры соединений.

При ручной проверке можно установить соответствия имен типов элементов, имен их выводов, имен экземпляров элементов (instance) и имен цепей. Проверка на нескольких примерах показала, что эти соответствия представляют собой взаимно-однозначные отображения, т.е. структуры, задаваемые в обоих представлениях, эквивалентны.

Последний этап преобразования в VHDL – фаза оценки – выполняется в ходе обхода графа, построенного в результате применения системы правил перезаписи. При обходе наименования цепей становятся наименованиями сигналов в VHDL-описании. Порты всех элементов, за исключением триггеров и элементов памяти, заменяются функциональным описанием элемента на VHDL.

VII. Заключение

Разработанная программа-конвертер прошла экспериментальную проверку на 30 FPGA-проектах [7]. Исходными данными для реализации на микросхеме xc3s1000-4ft256 FPGA семейства Spartan 3 были алгоритмические VHDL-описания, в том числе и доступные по сети Internet, логические схемы в графическом редакторе системы ISE, либо описания в State-CAD, а также генерируемые IP-блоки системы ISE. Проверка правильности работы программы конвертации осуществлялась на основе сравнения результатов поведения исходных проектов и конвертированных VHDL-описаний. Исходные проекты моделировались как в системе ISE, так и отдельно в системе ModelSim. Моделирование конвертированных описаний осуществлялось только в системе ModelSim без использования системных библиотек ISE. В качестве синтезатора логических схем для заказных СБИС использовался синтезатор LeonardoSpectrum, который может использовать различные целевые библиотеки пользователя.

Литература

[1] France R B., Rumpe B. Model-driven Development of Complex Software: A Research Roadmap // International Conference on Software Engineering – ICSE. 2007. P. 37-54.

[2] Beckhoff, C. The Xilinx Design Language (XDL): Tutorial and Use Cases / C. Beckhoff, D. Koch, J. Torresen // Reconfigurable Communication-centric System-on_Chip (ReCoSoC'2011). 6th International Workshop. Montpellier. 2011. P. 1-8.

[3] Rozenberg G., ed. Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific. 1999. Vol. 1, 2, 3.

[4] The Model Driven Architecture / OMG, Needham, MA, 2002. URL: http://www.omg.org/mda/ дата обращения 08.05.2014.

[5] Agrawal A., Karsai G., Shi F. Graph Transformations on Domain-Specific Models // Technical report ISIS-03-403, Vanderbilt University. November, 2003.

[6] Черемисинов Д.И. Генерация выполнимых спецификаций цифровых систем из структурных описаний FPGA-проектов // Информатика. 2012. № 3(35). С. 111-123.

[7] Бибило П.Н. и др. Конвертация FPGA-проектов семейства Spartan 3 в заказные СБИС // Современная электроника. 2012. № 2. С. 58-61.

[8] Couch, J.D. Applications of TORC: An Open Toolkit for Reconfigurable Computing // Virginia Polytechnic Institute and State University [Electronic resource]. 2011. Mode of access : http://scholar.lib.vt.edu/theses/available/etd-08182011-165440/unrestricted/ Couch_JD_T_2011.pdf. – Date of access : 17.04.2012.