ИНФОРМАТИКА

2012 **июль-сентябрь** **№ 3**

УДК 621.374.3; 004.4:004.9

Д.И. Черемисинов

ГЕНЕРАЦИЯ ВЫПОЛНИМЫХ СПЕЦИФИКАЦИЙ ЦИФРОВЫХ СИСТЕМ ИЗ СТРУКТУРНЫХ ОПИСАНИЙ FPGA-ПРОЕКТОВ

Описывается алгоритм конвертации формата XDL в промежуточный формат описания реализованного на FPGA проекта. Моделью промежуточного представления служит двудольный граф. Алгоритмическим базисом операции конвертации является подстановка графов, задающая распаковку элемента FPGA в примитивные логические элементы. Проводится методика испытаний этого алгоритма, для реализации которой разработана программа конвертации промежуточного представления в формат EDIF.

Введение

Проектировщик цифрового устройства должен построить его функциональное и структурное описание. Обычно задается только одна форма спецификации, другая строится в ходе проектирования с применением САПР. Как правило, исходными данными для процесса проектирования является функциональное описание, по которому во время логического синтеза и технологического проектирования строится структурное описание. Нужда в преобразовании структурного описания в функциональное возникает при верификации результатов проектирования.

Структурная модель (netlist) представляет систему в терминах взаимосвязей составляющих компонентов и не содержит ничего кроме компонентов и цепей с их атрибутами. В своем основном значении структурное описание есть описание *внутреннего устройства* чего-либо. Структурное описание устройства является взглядом наблюдателя изнутри системы, характеризующим ее организацию через элементы (части) и их взаимозависимости. Функциональное описание системы является взглядом внешнего наблюдателя, определяющим его взаимодействие с окружающей средой через соотношения между входами и выходами. Структурное описание выполнимо (может моделироваться), если известны функциональные модели компонентов; функциональное описание – это описание законов функционирования, эволюции системы, алгоритмов ее поведения, оно моделируемо само по себе по определению. Таким образом, описания поведения при помощи понятий или структуры или функции содержат фундаментальные различия.

Для представления структурных моделей в современных САПР используются специальные текстовые языки описания данных, называемые форматами структурных описаний. Формальной моделью структурных описаний являются графы.

1. Форматы структурных описаний FPGA

Использование элемента в описании называют экземпляром элемента (instance). Структурное описание содержит (или ссылается на) описания элементов или блоков устройства, и, таким образом, каждый экземпляр имеет «определение». В этих определениях обычно перечисляются точки связи, используемые при подключении элемента к другим элементам устройства. Точки подключения связей называют выводами (port) элемента. Выводы считаются частью структуры экземпляра элемента. Цепи (net) символизируют «провода», которыми подключаются экземпляры элементов в устройстве.

Структурная модель может быть задана в виде списка экземпляров (instance based netlist) или в виде списка цепей (net based netlist).

Список экземпляров задает структуру соединений, используемых в устройстве, указанием для каждого экземпляра списка пар, содержащих вывод экземпляра и цепь, с которой связан этот вывод, – списка подключения выводов элементов (port map). В этом виде описания список

цепей нужно собирать, анализируя списки подключения выводов элементов. Примером задания структуры соединений списком экземпляров является структурный стиль VHDL [1].

Список цепей задает структуру соединений, используемых в устройстве, указанием, во-первых, всех экземпляров элементов и, во-вторых, всех цепей, которые задаются перечислением выводов экземпляров, связанных этой цепью. Примером задания структуры соединений списком экземпляров является формат EDIF [2].

EDIF – формат netlist общего назначения, принятый в качестве стандарта в области САПР интегральных схем. EDIF имеет Lisp-подобный синтаксис. В настоящее время наиболее широко используется версия 2 0 0 этого формата, позволяющая описывать структуру проектируемых устройств. Известные производители САПР электронных устройств, такие как Cadance, Mentor Graphics, Altera и др., в своих САПР предусматривают возможность использования EDIF.

Предложения языка EDIF имеют иерархическую структуру и построены из конструкций, называемых секциями. Все секции имеют стандартную форму. Текст секции заключен в скобки и состоит из названия секции, обычно заданного ключевым словом языка EDIF, и тела секции. Тело чаще всего представляет собой последовательность секций более низкого уровня иерархии. Тела секций нижнего уровня иерархии содержат непосредственно данные.

Содержательно описание проектного решения на EDIF состоит из множества библиотек (library), которые, в свою очередь, состоят из элементов (cell). Каждый элемент представлен в библиотеке в одном или нескольких видах (view). Обычно используется один тип view – netlist. Задание view состоит из интерфейса (interface) и содержания (contents). Секция интерфейса задает определение элемента – список его выводов, а секция содержания представляет собой структурное описание элемента в виде списка цепей. Вначале перечисляются экземпляры, а затем цепи.

Для описания устройств с большим количеством экземпляров общепринятой практикой является использование иерархии. Иерархическое описание разбито на блоки. Каждый блок становится «определением», экземпляры которого используются в структурном описании. Определение, которое не включает экземпляров, называют *примитивом* в данном структурном описании, тогда как определение, которое включает экземпляры других элементов, является иерархическим – детализованным. В «свернутом» (folded) иерархическом описании допускается использование нескольких экземпляров одного детализованного определения. В «развернутом» (unfolded) иерархическом описании не допускается больше одного экземпляра детализованного определения. Структурное описание называется «плоским» (flat), если все экземпляры являются примитивами.

В процессе проектирования устройств на основе Xilinx FPGA результирующее структурное описание представлено в формате NCD (Native Circuit Description) [3], описание которого является технологическим секретом Xilinx. Для доступа к структурному представлению на этом уровне Xilinx предлагает текстовый формат XDL, частично описанный в [4]. Формат XDL является плоским структурным описанием в виде списка цепей.

Обобщение (generalization) – это вид отношений между общим описанием и специфическим, которое основывается на общем и детализирует его. Специфическое описание полностью согласуется с общим (имеет то же поведение), но обладает также и некоторой дополнительной информацией. Обобщение связывает общую сущность (родителя) с ее конкретным воплощением (потомком). Обобщения иногда называют отношениями типа «является», имея в виду, что одна сущность является частным (детализованным) выражением другой, более общей. Обобщение означает, что специфическое описание может заменить общее описание, но не наоборот. Другими словами, потомок может быть подставлен вместо родителя. Операция замены общего структурного описания более детальным структурным описанием в соответствии с заданным отношением обобщения – это *распаковка* исходного описания.

Смысл распаковки состоит в построении описания, которое, оставаясь структурным, позволяет получить функциональное (выполнимое) описание. В детализованном описании названия типа элементов позволяют определить функцию – отношение вход-выход – элемента. Поэтому такое описание может быть превращено в выполнимое путем суперпозиции этих функций в соответствии со структурой соединений.

Информация об отношении обобщения структурного описания [5], соответствующего размещенной в FPGA схеме, содержится в файле отчета для соответствующего типа (part)

FPGA Xilinx. Программа *xdl* в САПР Xilinx имеет опции, позволяющие построить файлы отчета (с расширением .XDLRC). Отношение обобщения задано в файле отчета секцией *primitive_defs*, содержащей зависимости для каждого типа блока родительского описания (рис. 1).

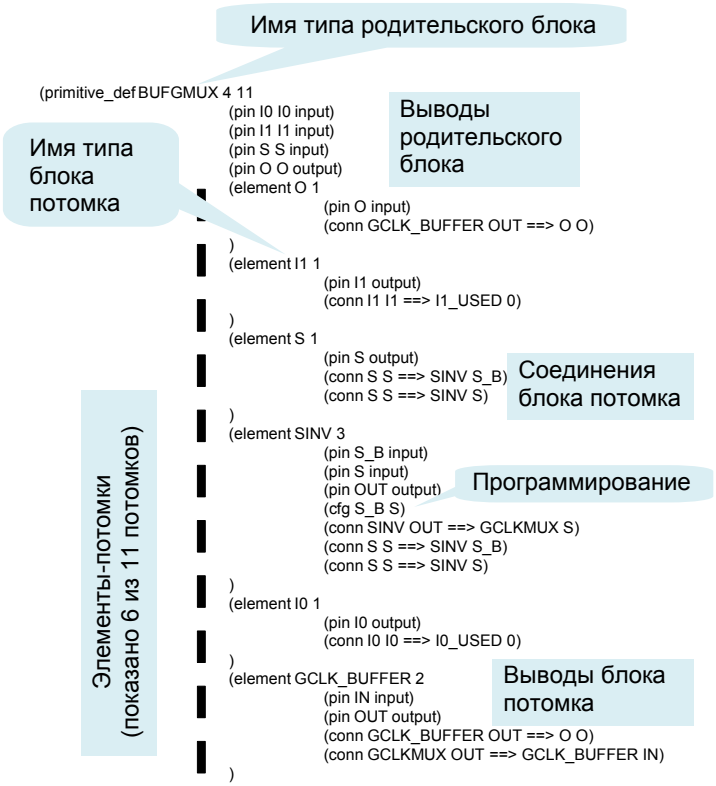

Рис. 1. Зависимость отношения обобщения для блока BUFGMUX из файла отчета

Описание примитива формата XDL в файле отчета – это плоское структурное описание в виде списка экземпляров. Иллюстрация отношения обобщения для блока SLICEL приведено в виде схемы [5, рис. 4-2] в руководстве пользователя FPGA Xilinx семейства Spartan-3.

2. Модель структуры устройства для операции распаковки

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

В качестве примера на рис. 2 изображен фрагмент логической схемы секции SLICEL, а на рис. 3 – 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.

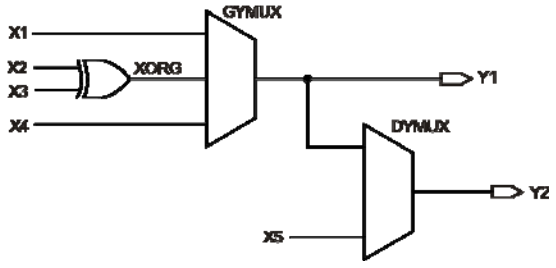

Рис. 2. Логическая схема (фрагмент секции SLICEL)

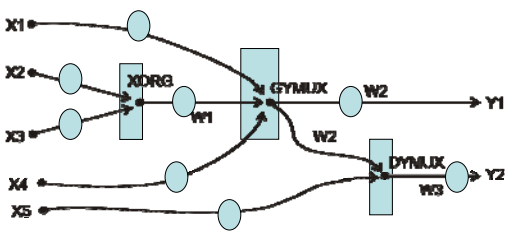

Рис. 3. Неориентированный двудольный граф, соответствующий логической схеме

На рисунках графов вершины, являющиеся выводами элемента, сгруппированы в прямоугольники (сами прямоугольники не являются элементами графа). Обозначения выводов предваряются именем элемента, показанным на рисунках в фигуре элемента (см. рис. 3). Вершины цепей показаны кружками. Обозначения (имена) выводов и некоторых цепей для упрощения рисунка опущены.

3. Распаковка блоков FPGA Spartan 3

В качестве алгоритмического базиса для решения задачи распаковки формата XDL удобно использовать подстановки графов, основой которых являются операции поиска и замены подграфов. Подстановка графов задает правило переписывания графа в форме «образец – действие» [6].

Алгоритмические системы на основе правил переписывания в форме «образец – действие» позволяют описывать любые алгоритмически представимые вычисления, так как в их основе лежит алгоритмически полная система нормальных алгоритмов Маркова [7]. Очевидно, свойства этой системы сохраняются при ее использовании для вычислений на графах. Функциональные возможности этой системы ограничиваются использованием единственной операции – сопоставления с образцом для построения любых алгоритмов. В теории алгоритмов Маркова эта операция называется *подстановкой слов*. Подстановка состоит из двух частей: левой и правой. Подстановка применяется к слову следующим образом: в слове ищется подслово, совпадающее с левой частью подстановки (образцом). Если такое подслово находится, то оно заменяется правой частью подстановки.

В основе алгоритма преобразования структурного описания, построенного на базе операций сопоставления с образцом, лежит та же идея, что и в алгоритме Маркова, когда берется список образцов (подсхем) с действиями замены и выполняются действия везде, где соответствующие образцы встречаются в некотором большом структурном описании.

Граф g – неориентированный двудольный граф, построенный в результате синтаксического анализа структурного описания в формате XDL. Вершины графа *g* помечены. Граф *H* называется *подграфом* (или *частью*) графа *G*, если $VH \subseteq VG$, $EH \subseteq EG$. Если множество вершин подграфа *H* есть *U*, а множество его ребер совпадает с множеством всех ребер графа *G*, концы которых принадлежат *U*, то *H* называется *подграфом, порожденным множеством U*. Определим двуместную операцию *склейки* вершин как отождествление двух заданных вершин пары графов, не имеющих общих элементов. Можно обобщить эту операцию как склейку графов, указывая множество пар склеиваемых вершин.

Число ребер, инцидентных некоторой вершине *v*, называется *степенью* вершины и обозначается deg *v*, вершина степени 1 называется *концевой* (*висячей*). Ребро, инцидентное концевой вершине, также называется *концевым*. В графе *g*, построенном в результате синтаксического анализа структурного описания в формате XDL, все вершины-выводы являются висячими, а вершины-цепи имеют $\deg v \geq 2$ (см. рис. 3).

Подстановка графов – это правило, заданное с помощью $g_j \rightarrow g_r$, где g_j и g_r – графы. Граф g_j называется образцом подстановки, а g_r – правой частью подстановки. Подграф графа *g*, изоморфный графу g_j, должен быть заменен графом, изоморфным g_r. Эта операция называется применением подстановки. *Операция применения подстановки* использует модель приклеивания, в которой механизм применения подстановки задается отображением склеивания. *Функция трансформации меток* вычисляет значения меток при применении операции подстановки. Задает процедуру вычисления меток в подграфе g_r^{host} по значениям меток в подграфе g_i^{host}.

Введем следующие определения:

g – исходный граф, к которому применяется правило подстановки;

g_j^{host} – подграф графа *g*, изоморфный g_j. *Остаток графа* – это граф $g - g_i^{host}$ (оператор «–» обозначает удаление из *g* всех вершин и ребер g_i^{host} и всех ребер, одна или обе конечные точки которых принадлежат g_i^{host});

g_r^{host} – подграф, изоморфный g_r и используемый при подстановке вместо g_i^{host}. *Исходное множество ребер связи* – множество ребер, соединяющих g_i^{host} с остатком графа (эти ребра

«подключают» g_i^{host} к g). *Множество ребер связи после подстановки* – множество ребер, соединяющих g_r^{host} с остатком графа.

Чередующаяся последовательность $v_1, e_1, v_2, e_2, ..., v_l, e_l, v_{l+1}$ вершин и ребер графа, такая, что $e_i = v_i v_{i+1}$ (i = 1, 2, …, l), называется *путем, соединяющим вершины* $\boldsymbol{v_i}$ *и* $\boldsymbol{v_{i+1}}$ (или ($\boldsymbol{v_i}$, $\boldsymbol{v_{i+1}}$)-путем). Очевидно, что путь можно задать последовательностью $v_1, v_2, ..., v_l, v_{l+1}$ его вершин, а также последовательностью его ребер $e_1, e_2, ..., e_l$. Граф называется *связным*, если любые две несовпадающие вершины в нем соединены путем. В графе g, построенном в результате синтаксического анализа структурного описания в формате XDL, число компонент связности равно числу цепей.

3.1. Подстановка распаковки

Выводы элементов при распаковке удобно считать висячими вершинами. Образцом подстановки распаковки является граф, множество вершин которого состоит из выводов элемента заданного типа, а множество ребер пустое. Правая часть подстановки является двудольным графом, построенным в результате синтаксического анализа соответствующей подсекции секцией *primitive_defs* из файла отчета (с расширением .XDLRC). Для иллюстрации на рис. 4 приведена подстановка распаковки элемента BUFGMUX [5].

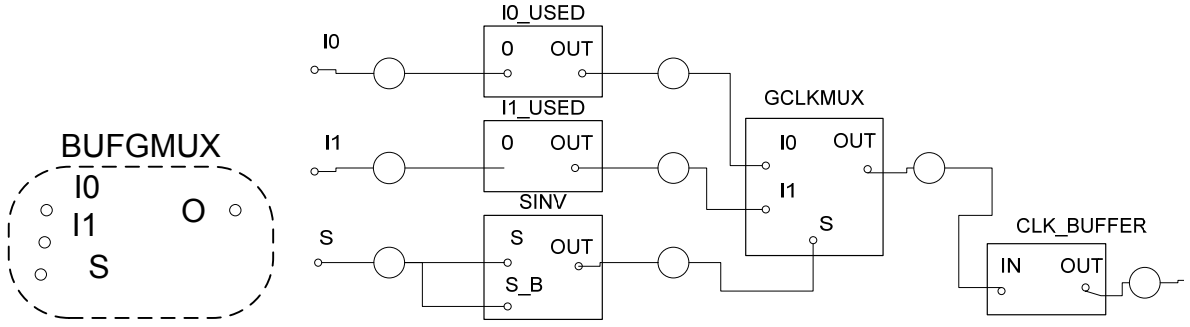

Рис. 4. Подстановка распаковки элемента BUFGMUX

Граф правой части подстановки строится в ходе анализа подсекции BUFGMUX секцией *primitive_defs* из файла отчета. Все выводы элементов включаются в долю выводов графа. Списки соединений преобразуются в долю цепей и задают ребра, связывающие выводы и цепи.

Операция применения подстановки распаковки выполняется в два этапа. На первом этапе все вершины подграфа в исходном графе g, выделенного образцом подстановки, склеиваются с одноименными внешними выводами графа g_r правой части подстановки. Не подсоединенные в графе g выводы элемента g_r не включаются в результат подстановки. Цепи с deg $v < 2$ тоже не включаются в результат подстановки. На втором этапе склеиваются вершины-цепи g, инцидентные некоторой вершине и полученные в результате склеивания на предыдущем этапе, а сама эта вершина удаляется из графа g. Данная операция выполняется для каждого вывода распаковываемого элемента.

Процесс применения подстановки распаковки элементов типа BUFGMUX к фрагменту схемы (рис. 5) состоит в следующем. Единственный подграф, соответствующий образцу подстановки распаковки элементов этого типа, в этом фрагменте выделен пунктиром. После приклеивания вершин-выводов из правой части подстановки к вершинам, выделенным образцом, граф фрагмента принимает вид, показанный на рис. 6. Затем одна за другой удаляются вершины-выводы и склеиваются инцидентные удаляемой вершине вершины-цепи. Для вывода I0 склеиваемые цепи показаны жирной дуговой стрелкой.

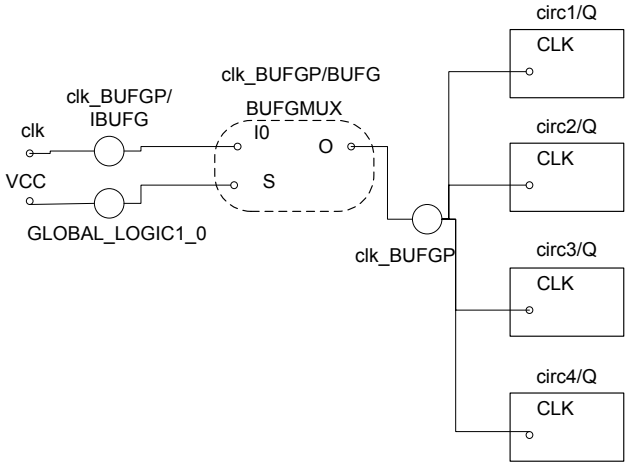

Рис. 5. Исходный фрагмент схемы устройства VLSI_1_D

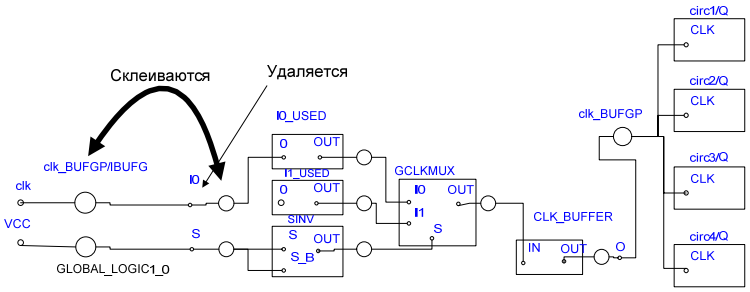

Рис. 6. Результат приклеивания провой части подстановки при распаковке элемента clk_BUFGP/BUFG

3.2. Учет параметров программирования

Описание каждого элемента 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 элемента и всегда содержит три члена, разделенные двоеточием. Например, для элемента clk_BUFGP/BUFG из устройства VLSI_1_D задана следующая строка конфигурации.

```
cfg " DISABLE_ATTR::LOW I0_USED::0 I1_USED::#OFF SINV::S_B GCLKMUX:clk_BUFGP/B
UFG.GCLKMUX: GCLK_BUFFER:clk_BUFGP/BUFG:"
```

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 подстроке `DISABLE_ATTR::LOW` первый член указывает, что описывается конфигурирование логического элемента `DISABLE_ATTR` элемента BUFGMUX. Имя экземпляра этого элемента не задано, а параметр программирования (value) равен `LOW`. Все возможные значения программирования элемента приведены в составляющей cfg описания `DISABLE_ATTR` в файле отчета. Подстрока `I0_USED::0` указывает, что элемент с именем `I0_USED` соединяет цепь его входа с цепью выхода, а подстрока `I1_USED::#OFF` указывает, что в элементе с именем `I1_USED` соединение цепи входа с цепью выхода отсутствует. Программирование `SINV::S_B` обозначает использование функции инвертирования в элементе `SINV`. Подстроки `GCLKMUX:clk_BUFGP/BUFG.GCLKMUX:` и `GCLK_BUFFER:clk_BUFGP/BUFG:` задают имена экземпляров элементов `GCLKMUX` и `GCLK_BUFFER`.

Для учета программирования при распаковке элемента в правой части подстановки удаляются все выводы элементов, для которых указано программирование `#OFF`, и удаляются неправильные цепи. Выводы программируемых коммутационных элементов удаляются, а инцидентные им цепи склеиваются в соответствии с настройкой элемента.

Анализируя строки конфигурации в элементах FPGA, можно заметить, что значительная часть составляющих элементов имеет значение #OFF в подстроке конфигурации, т. е. находится в выключенном состоянии. С точки зрения экономии вычислительной работы выгодно при реализации подстановки использовать инверсный процесс формирования правой части подстановки. В прямом процессе реализация распаковки начинается с полного графа элементов примитива и их связей с исключением в дальнейшем ненужного. В инверсном процессе правая часть подстановки вначале содержит используемые в данной конфигурации элементы, которые можно найти, анализируя строку конфигурации, затем эти элементы дополняются цепями в соответствии с описанием структуры примитива в файле отчета.

Покажем этот процесс формирования правой части подстановки распаковки на примере элемента circ3/Q типа SLICEL [5]. Для этого элемента задана следующая строка конфигурации:

```
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
G:circ0/circ1/d32:#LUT:D=((~A3*(A4*A1))+(A3*((~A2*(A4*A1))+(A2*(A4+A1)))))
GYMUX::#OFF REVUSED::#OFF SRINV::#OFF
SYNC_ATTR::ASYNC XBUSED::#OFF
XUSED::#OFF YBUSED::#OFF YUSED::#OFF
F5MUX:circ0/circ1/d3_f5: "
```

Подстроки активных элементов перечислены ниже. Всего в этом элементе используется восемь компонентов: BXINV, CLKINV, DXMUX, F, FFX, FXMUX, G, F5MUX.

```
BXINV::BX
CLKINV::CLK
DXMUX::1
F:circ0/circ1/d31:#LUT:D=(A1*(A4*(~A3+~A2)))
FFX:circ3/Q:#FF
FXMUX::F5
G:circ0/circ1/d32:#LUT:D=((~A3*(A4*A1))+(A3*((~A2*(A4*A1))+(A2*(A4+A1)))))
F5MUX:circ0/circ1/d3_f5: "
```

Эти компоненты в файле отчета имеют следующие описания. Цепи соединений с активными компонентами или с выводами элемента выделены жирным шрифтом.

```
(element BXINV 3
              (pin BX_B input)
              (pin BX input)
              (pin OUT output)
              (cfg BX_B BX)
              (conn BXINV OUT ==> CYINIT BX)
              (conn BXINV OUT ==> DXMUX 0)
              (conn BXINV OUT ==> CY0F BX)
              (conn BXINV OUT ==> F5MUX S0)
              (conn BX BX ==> BXINV BX_B)
              (conn BX BX ==> BXINV BX)
       )
(element CLKINV 3
              (pin CLK_B input)
              (pin CLK input)
              (pin OUT output)
              (cfg CLK_B CLK)
              (conn CLKINV OUT ==> FFX CK)
              (conn CLKINV OUT ==> FFY CK)
              (conn CLK CLK ==> CLKINV CLK_B)
              (conn CLK CLK ==> CLKINV CLK)
       )
```

```
(element DXMUX 3
                (pin 0 input)
                (pin 1 input)
                (pin OUT output)
                (cfg 0 1)
                (conn DXMUX OUT ==> FFX D)
                (conn BXINV OUT ==> DXMUX 0)
                (conn FXMUX OUT ==> DXMUX 1)
        )
(element F 5
                (pin A1 input)
                (pin A2 input)
                (pin A3 input)
                (pin A4 input)
                (pin D output)
                (cfg <eqn>)
                (conn F D ==> XORF 0)
                (conn F D ==> CYSELF F)
                (conn F D ==> FXMUX F)
                (conn F D ==> F5MUX F)
                (conn F1 F1 ==> F A1)
                (conn F2 F2 ==> F A2)
                (conn F3 F3 ==> F A3)
                (conn F4 F4 ==> F A4)
        )
(element FFX 6
                (pin CK input)
                (pin CE input)
                (pin D input)
                (pin Q output)
                (pin SR input)
                (pin REV input)
                (cfg #FF #LATCH)
                (conn FFX Q ==> XQ XQ)
                (conn CLKINV OUT ==> FFX CK)
                (conn CEINV OUT ==> FFX CE)
                (conn DXMUX OUT ==> FFX D)
                (conn SRINV OUT ==> FFX SR)
                (conn REVUSED OUT ==> FFX REV)
        )
(element FXMUX 4
                (pin F5 input)
                (pin F input)
                (pin FXOR input)
                (pin OUT output)
                (cfg F5 F FXOR)
                (conn FXMUX OUT ==> XUSED 0)
                (conn FXMUX OUT ==> DXMUX 1)
                (conn F5MUX OUT ==> FXMUX F5)
                (conn F D ==> FXMUX F)
                (conn XORF O ==> FXMUX FXOR)
        )
(element G 5
                (pin A1 input)
                (pin A2 input)
                (pin A3 input)
                (pin A4 input)
                (pin D output)
                (cfg <eqn>)
                (conn G D ==> XORG 0)
                (conn G D ==> CYSELG G)
                (conn G D ==> F5MUX G)
                (conn G D ==> GYMUX G)
                (conn G1 G1 ==> G A1)
                (conn G2 G2 ==> G A2)
                (conn G3 G3 ==> G A3)
                (conn G4 G4 ==> G A4)
        )
```

```
(element F5MUX 4
              (pin F input)
              (pin G input)
              (pin OUT output)
              (pin S0 input)
              (conn F5MUX OUT ==> F5USED 0)
              (conn F5MUX OUT ==> FXMUX F5)
              (conn F D ==> F5MUX F)
              (conn G D ==> F5MUX G)
              (conn BXINV OUT ==> F5MUX S0)
        )
```

Сформированный из выводов элементов и выделенных цепей двудольный граф показан на рис. 7.

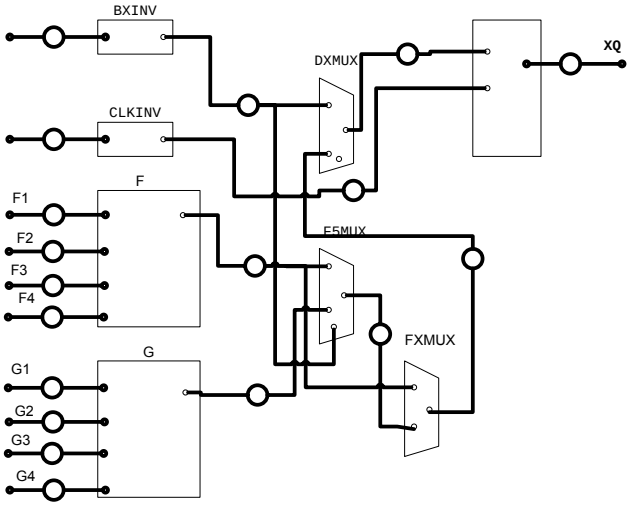

Рис. 7. Структура элемента circ3/Q

Непосредственно при построении графа на рис. 7 можно учесть программирование элементов. Вид правой части подстановки, используемой при распаковке элемента circ3/Q, изображен на рис. 8.

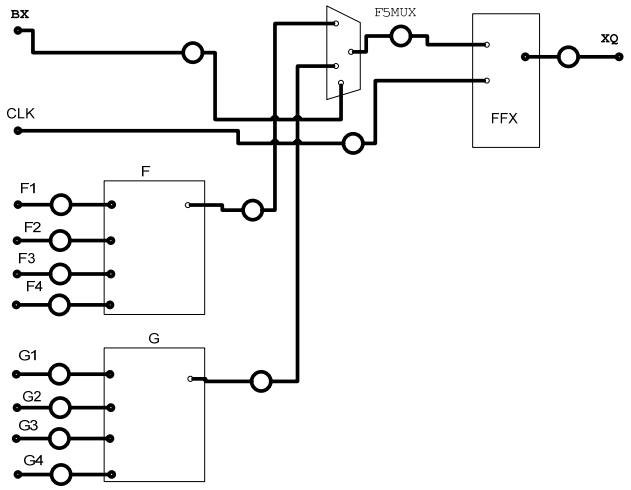

Рис. 8. Правая часть подстановки распаковки элемента circ3/Q

3.3. Алгоритм распаковки

1. Просматриваются предложения net из исходного описания в формате XDL и строится двудольный граф структурного описания.

2. Для каждого предложения inst из исходного описания выполняются следующие действия.

2.1. Используя строку конфигурации рассматриваемого предложения inst, с помощью инверсного процесса строится с учетом программирования двудольный граф правой части подстановки.

2.2. К выводам экземпляра с именем рассматриваемого предложения inst (к подграфу, составленному из этих выводов) применяется построенная на шаге 2.1 подстановка графов.

2.3. Подставленные при распаковке экземпляры элементов имеют имена типов, являющиеся именами составляющих элементов из описания примитива в файле отчета, используемого для построения правой части подстановки. В качестве имени экземпляра используется имя исходного экземпляра, дополненное именем типа. Например, распаковывается экземпляр *circ2/Q*, именем экземпляра LUT G будет *circ2/Q:G*. Если в строке конфигурации для элемента указан второй компонент параметра, то эта строка тоже добавляется к имени элемента. Например, LUT G *circ2/Q* имеет параметр программирования

```
G:circ0/circ1/d2_F:#LUT:D=((~A2*(A4@A1))+(A2*(A4*A3))).
```

Полное имя экземпляра, полученного в результате подстановки, будет иметь вид *circ2/Q:G:circ0/circ1/d2_F*. Имена вновь образуемых цепей строятся на основе имени элемента, являющегося источником в цепи. Это имя дополняется именем вывода выхода.

Уточненное структурное описание, полученное в результате распаковки исходного структурного описания, может быть дополнено функциональными описаниями составляющих элементов, что превращает структурное описание в функциональное. Соответствующее преобразование – это оценка (evaluation – вычислить числовое значение, выразить в цифровой форме).

Этап оценки состоит в назначении логических функций элементам структурного описания, полученного в результате распаковки. Для комбинационных элементов логическая функция определяется названием элемента. Элементы F и G являются комбинационными, если значение параметра программирования value из подстроки конфигурации начинается с «#LUT». Например, элемент G в конфигурации элемента circ3/Q задан следующей строкой:

```
G:circ0/circ1/d32:#LUT:D=((~A3*(A4*A1))+(A3*((~A2*(A4*A1))+(A2*(A4+A1))))).
```

Здесь `circ0/circ1/d32` – имя экземпляра элемента, а значением параметра программирования является строка, в которой логическое выражение задает функцию элемента

```
#LUT:D=((~A3*(A4*A1))+(A3*((~A2*(A4*A1))+(A2*(A4+A1))))).
```

Секции (слайсы) CLB могут служить 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)`.

4. Методика проверки работоспособности алгоритма распаковки

Алгоритм распаковки работает с неофициальным (недокументированным) описанием, правильность его работы основывается на предположении, что по названиям типов элементов можно правильно установить их функции. Смысл проверки заключается в том, что она позволяет подтвердить (опровергнуть) данное предположение. Очевидно, эта проверка не может быть формальной. Так как алгоритм распаковки работает со структурой в памяти, для оценки его работы требуется преобразование этой структуры в форму для вывода результатов. Моделью обрабатываемого структурного описания во внутреннем представлении служит двудольный граф, форма вывода которого должна допускать ручное и автоматическое сравнение с эталонным представлением.

В качестве эталонного представления можно использовать структурное описание, являющееся результатом синтеза. Инструментом синтеза в САПР FPGA Xilinx является програм-

ма XST (Xilinx Synthesis Technology) [3]. 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 NGC2EDIF, предназначенная для перевода формата NGC в EDIF. Ее назначением является представление результата синтеза в форме, воспринимаемой инструментами из других САПР. Программа NGC2EDIF по файлу с расширением NGC строит файл, содержащий структурное описание в формате EDIF и имеющий расширение NDF. Описание в файле NDF не предназначено для использования в процессе проектирования FPGA Xilinx. С учетом этих соображений удобно, чтобы форматом вывода для оценки работоспособности алгоритма тоже являлся EDIF (рис. 9).

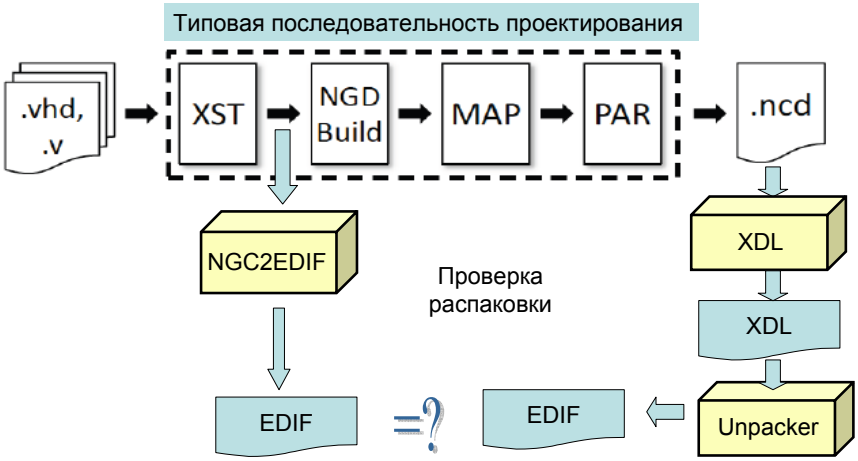

Рис. 9. Тест при оценке работоспособности алгоритма распаковки

Формат EDIF может быть преобразован в формат SF [8], для этого имеются соответствующие программы [9]. Описание в формате EDIF невыполнимо, но может стать выполнимым после преобразования в VHDL и замены элементов их функциональными эквивалентами.

Сравнение текстов в формате EDIF, полученных программой NGC2EDIF и программой распаковки, обнаруживает разницу. Прежде всего сравниваемые представления отличаются по элементам. В EDIF, полученном программой NGC2EDIF (файл NDF), присутствуют буферные элементы IBUF и OBUF. Так как цель распаковки состоит в том, чтобы служить промежуточным результатом при построении функционального описания, то включать в структуру элементы, реализующие только транспорт данных, нецелесообразно. Также отличаются имена типов элементов и имена их выводов. Описание в файле NDF ориентировано на моделирование, основанное на библиотеке UNISIMS [5], и использует имена, определенные в этой библиотеке. Распакованный EDIF содержит имена типов элементов и имена их выводов, заданные в файле отчета. Отличаются и наименования экземпляров элементов и цепей. Таким образом, текстовое сравнение этих двух файлов не позволяет выявить эквивалентность структуры соединений.

Оценка теста работоспособности требует ручного сравнения для выявления эквивалентности описаний. Работа по построению алгоритма преобразования внутреннего представления в формат EDIF, который эквивалентен EDIF, построенный программой NGC2EDIF, нецелесообразна, так как на суть самого алгоритма распаковки не влияет. Кроме того, когда сформировано функциональное описание, заменяющее структурное, тест на работоспособность можно проводить проверкой эквивалентности результатов моделирования.

Методика ручной проверки состоит в установлении соответствия имен типов элементов, имен их выводов, имен экземпляров элементов (instance) и имен цепей. Если эти соответствия представляют собой взаимно-однозначные отображения, тогда структуры, задаваемые в обоих представлениях, эквивалентны.

Заключение

Проблема распаковки формата XDL рассматривается в [10] 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 данного маршрута проектирования является требование «реверсивности» операции распаковки. 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 файлом отчета, для этого не годится.

В работе рассмотрена проблема конвертации структурного формата XDL проекта цифрового устройства, реализованного на FPGA семейства Spartan 3 (микросхема XC3S1000), в структурное промежуточное описание. По промежуточному описанию далее будет строиться моделируемое описание на языке VHDL. Описаны алгоритмы, разработанная программа конвертации и методика тестирования результата конвертации.

Список литературы

1. Суворова, Е.А. Проектирование цифровых систем на VHDL / Е.А. Суворова, Ю.Е. Шейнин. – СПб. : БХВ-Петербург, 2003. – 576 с.

2. Stanford, P. EDIF: Electronic Design Interchange Format Version 2 0 0 / P. Stanford, P. Mancuso. – Electronic Industries Association, ANSI/EIA-548-1988, 1988. – 484 p.

3. Зотов, Ю.В. Проектирование цифровых устройств на основе ПЛИС фирмы XILINX в САПР WebPack ISE / Ю.В. Зотов. – М. : Горячая линия – Телеком, 2003. – 624 с.

4. 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.

5. Xilinx Inc., Spartan-3 Generation FPGA User Guide – Extended Spartan-3A, Spartan-3E, and Spartan-3 FPGA Families UG331 (v1.5). – USA, 2009. – 522 p.

6. Черемисинов, Д.И. Анализ и преобразование структурных описаний СБИС / Д.И. Черемисинов. – Минск : Белорусская наука, 2006. – 275 с.

7. Марков, А.А. Теория алгорифмов / А.А. Марков, Н.М. Нагорный. – М. : Наука. Гл. ред. физ.-мат. лит., 1984. – 376 с.

8. Бибило, П.Н. Кремниевая компиляция заказных СБИС / П.Н. Бибило. – Минск : Ин-т техн. кибернетики АН Беларуси, 1996. – 268 с.

9. Черемисинов, Д.И. Конвертор структурных описаний для интеграции САПР // Междунар. конф. «Теория и практика логического управления» / Москва, 10–11 ноября 2003 г. – М. : Институт проблем управления им. В.А. Трапезникова, 2003. – С. 114–118.

10. Couch, J.D. Applications of TORC: An Open Toolkit for Reconfigurable Computing / J.D. Couch // 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.

Поступила 23.05.12

Объединенный институт проблем информатики НАН Беларуси,
Минск, Сурганова, 6
e-mail: cher@newman.bas-net.by

D.I. Cheremisinov

GENERATING EXECUTABLE CIRCUIT SPECIFICATIONS FROM STRUCTURAL FPGA DESCRIPTIONS

The problem of transformation of structural description Xilinx FPGA to the description which is possible to simulate without use of libraries of elements is considered. The algorithm of converting XDL format description of FPGA in an intermediate format is described. The model of intermediate representation is a bipartite graph. The algorithmic basis of the conversion operation is the graph substitution which unpacks an element of FPGA into the primitive logic elements. A technique is developed to verify this algorithm. This technique is based on the program that converts the intermediate format into the EDIF format.