Продолжение. Начало в № 3`2009

Изучаем Active-HDL 7.1. Урок 7. Проектирование схем: размещение компонентов

Александр ШАЛАГИНОВ
shalag@vt.cs.nstu.ru

Схемное описание проекта в среде Active-HDL 7.1 выполняется с помощью графического редактора Block Diagram Editor (BDE). С технологией проектирования схем мы немного знакомы по первому уроку. Теперь нам предстоит более подробное изучение этого вопроса.

Вызвать **BDE**-редактор можно многими способами (во всяком случае, их не менее семи). Конкретный выбор зависит от того, собираетесь вы пользоваться услугами «мастера» по созданию схем (**Block Diagram Wizard**) или полагаетесь на свой опыт.

Если вы только начинаете знакомство с этим пакетом, то проще всего активизировать закладку **Design Flow Manager**, навести указатель мышки на пиктограмму **BDE** (рис. 1) и щелкнуть левой кнопкой мыши. Именно этим приемом мы пользовались, вызывая схемный редактор на первом уроке.

Но сейчас мы поступим иначе. Щелкнем правой кнопкой мыши (**ПКМ**) в любом свободном месте на закладке **Design Flow Manager** и выполним команду **Design Entry/BDE…** (рис. 2).

Эффект будет такой же: активизируется «мастер» нового исходного файла **New Source File Wizard**. С его помощью введем имя схемного файла, например, **mux2_schema_mix**, и имена портов — **D0**, **D1**, **A**, **Y**.

Когда откроется рабочая область редактора, попробуйте нарисовать схему мультиплексора на 2 входа, используя логические элементы **and2**, **or2** и **inv** из встроенной библиотеки **Built-in Symbols**. Такая работа нам уже знакома по первому уроку. Желательно выполнить ее за 5–6 минут. Считайте, что это тест на «отлично».

Рис. 1. Схемный редактор BDE удобно запускать из менеджера маршрутов проектирования

Рис. 2. Команда Design Entry/BDE...

Проектирование схем на разнородных компонентах

По умолчанию **BDE**-редактору доступна только одна библиотека со встроенными символами **Built-in Symbols**. Но как только вы создадите и откомпилируете первый файл своего проекта, в «ящике символов» (в разделе **Units without symbols**) появится новый компонент, в данном случае — **mux2_schema_mix**. И выглядит он как логический примитив — черный ящик.

Это наводит на мысль, что в нашем проекте можно построить свои собственные элементы и использовать их для проектирования схемы. Подтвердим сказанное примером.

Создадим «самодельный» элемент, например **and2_my**. С этой целью активизируем закладку **Design Flow Manager** и щелкнем по иконке **HDE**. С помощью «мастера» введем имя файла **and2_my**, входные порты **I1**, **I2** и выходной порт **O**.

В автоматически созданный VHDL-шаблон добавим текст: **O <= I1 and I2;**. Место его включения подсказывается в шаблоне фразой: -- **enter your statements here** — («Введите ваши операторы здесь»). Вот и вся работа. Содержимое символа готово, но где же его графический образ?

Вы получите ответ на этот вопрос после того, как откомпилируете созданный файл и в схемном редакторе вновь откроете «ящик символов». Откройте раздел **Units without symbols** («Элементы без символов») и выделите в нем строку **and2_my**. В нижней части панели **Symbols Toolbox** появится графическое изображение только что созданного элемента (рис. 3).

На самом деле редактор показывает его будущую стандартную реализацию. Обратите внимание, в рабочей библиотеке **Lesson_7** графического образа созданного элемента **and2_my** еще нет. Это хорошо видно в окне менеджера библиотек **Library Manager** (рис. 4). С этим инструментом мы еще познакомимся.

Символ будет автоматически сгенерирован системой, как только вы сделаете попытку разместить на рабочей области **BDE**-редактора его первую копию. Проделайте названную операцию. Вы заметили небольшую задержку редактора на ваши действия? Это время было потрачено на то, чтобы создать графический образ элемента. Теперь и менеджер

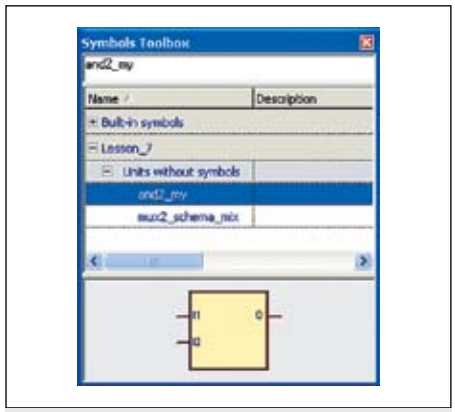

Рис. 3. В «ящике символов» появился созданный нами элемент and2_my

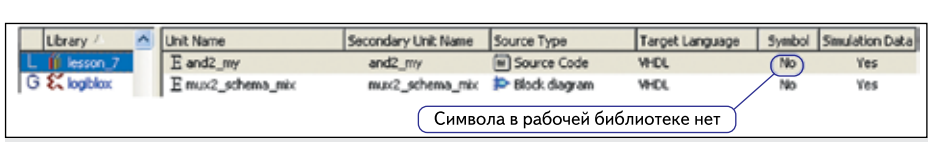

Рис. 4. Менеджер библиотек Library Manager показывает, что графического описания для элемента and2_my пока не существует

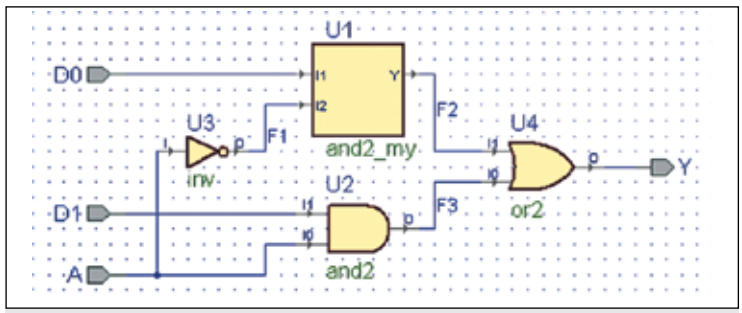

Рис. 5. Заменяем системный элемент and2 из встроенной библиотеки «самодельным» элементом and2_my

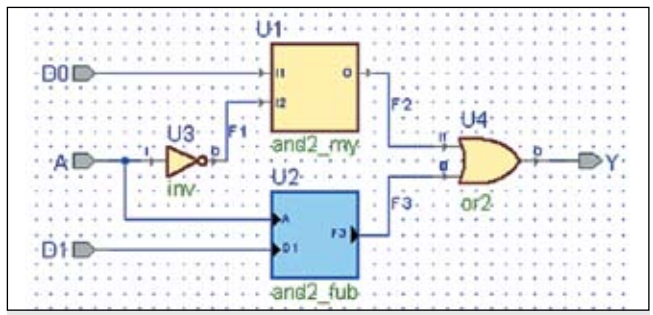

Рис. 7. Заменяем еще один встроенный элемент and2 иерархическим блоком and2_fub

```
entity mux2_schema_mix is
    port(
        D0,D1,A : in STD_LOGIC;
        Y : out STD_LOGIC);
end mux2_schema_mix;

architecture mux2_schema_mix of mux2_schema_mix is
    component and2_my
        port (I1,I2 : in STD_LOGIC;
              O : out STD_LOGIC);
    end component;
    signal F1 : STD_LOGIC;
    signal F2 : STD_LOGIC;
    signal F3 : STD_LOGIC;
begin
    U1 : and2_my     -- U1
    port map(
        I1 => D0,
        I2 => F1,
        O => F2);
    F3 <= D1 and A; -- U2
    F1 <= not(A);   -- U3
    Y <= F3 or F2;  -- U4
end mux2_schema_mix;
```

Рис. 6. VHDL-код, автоматически сгенерированный системой по схемному описанию мультиплексора mux2_schema_mix

библиотек покажет, что символ появился в рабочей библиотеке.

Заменим один из элементов **and2**, взятый из встроенной библиотеки, на «самодельный» **and2_my** (рис. 5) и вновь откомпилируем схему мультиплексора **mux2_schema_mix**.

Интересно посмотреть, каким получится ее **VHDL**-код (рис. 6). Сравните описания встроенного **and2** и нашего **and2_my** вентилей.

Не сомневаюсь, вы оцените лаконичность записи для встроенного символа U2 — всего одна строка кода: **F3 <= D1 and A;**. Для нашего вентиля U1 сначала потребовалось его объявить (строки 31–34), а затем в исполняемом разделе вызвать его конкретную реализацию (строки 39–43).

Продолжим «чудесные превращения» схемы мультиплексора и заменим еще один встроенный элемент — U2 — иерархическим блоком. Придется сначала удалить элемент U2, затем нажать на пиктограмму **Fub** и нарисовать внешний вид блока так, чтобы он касался «висячих» проводников (рис. 7).

Стандартное имя **Fub1** поменяем на пользовательское **and2_fub** и создадим внутреннее описание блока, выбрав для реализации **VHDL**-код. Шаблон описания будет создан автоматически, так что нам останется только вставить в него строку **F3 <= A and D1;**. Более подробно проектирование иерархических блоков будет рассмотрено в последующих уроках.

Чтобы завершить наш эксперимент, осуществим еще одно действие. Заменим встроенный элемент **or2** графическим процессом. Понятно, что элемент **or2** придется удалить, а на его место вставить новый загадочный объект, называемый графическим процессом. Мы еще вернемся к нему.

Операция напоминает вставку иерархического блока, только теперь нам нужна другая иконка (**Add Process**). Чтобы все получилось наилучшим образом, контур размещаемого объекта должен соприкасаться с концами «висячих» проводников (рис. 8). Стандартную метку процесса **Process_1** мы тоже поменяем на **or2_my**, а в тело процесса добавим строку **Y <= F2 or F3;**. Эту операцию раньше выполнял встроенный элемент **or2**.

Проведенный эксперимент показывает, что на одной схеме могут «уживаться», на первый взгляд, совершенно разнородные графические объекты: символы (системные и пользовательские), иерархические блоки и графические процессы ( ). Чтобы подчеркнуть этот факт, в название схемы **mux2_schema_mix** было добавлено слово **mix** (смешанный).

Мы уже отмечали, что по умолчанию к редактору схем подключена только одна встроенная библиотека символов. Символы из этой библиотеки имеют много особенностей.

Во-первых, они недоступны для редактирования (кроме системных настроек). Да и содержимое самой библиотеки, включающей несколько десятков компонентов, изменить нельзя.

Во-вторых, при генерации **HDL**-кода из схемы получается очень компактное потоковое представление проекта (так называемое **RTL**-описание), при котором каждый компонент записывается всего одним булевским выражением (рис. 6 или урок 1, рис. 13). Возможно, по этой причине в данной библиотеке отсутствуют триггеры.

Из-за отмеченных особенностей встроенная библиотека символов не видна в окне менеджера библиотек **Library Manager**.

В редких случаях реальные проекты удается выполнить, используя компоненты только встроенной и/или рабочей библиотеки. Как же подключить к редактору системные библиотеки или библиотеки из других проектов? А ведь это совершенно необходимое условие для повторного использования ранее выполненных разработок (технология **reuse**).

Подключение к проекту библиотек с символами

На деле все получается просто. Вызовите «ящик символов» (горячая клавиша **S**), откройте для него контекстное меню и выполните команду **Select Libraries**. Вы увидите список всех доступных редактору библиотек (рис. 9).

В начале списка находится встроенная библиотека **Built-in symbols**. Установленный рядом с ней флажок говорит о том, что по умолчанию она присоединена к схемному редактору.

Далее идут библиотеки пользователя, объединенные под общим названием **User Libraries**. По умолчанию в данный список входят только библиотеки проектов, включенных в активное рабочее пространство (в примере это **Workspace 'Lessons'**).

Для упрощения рисунка часть проектов временно удалена из рабочего пространства (команда **Remove from Workspace**).

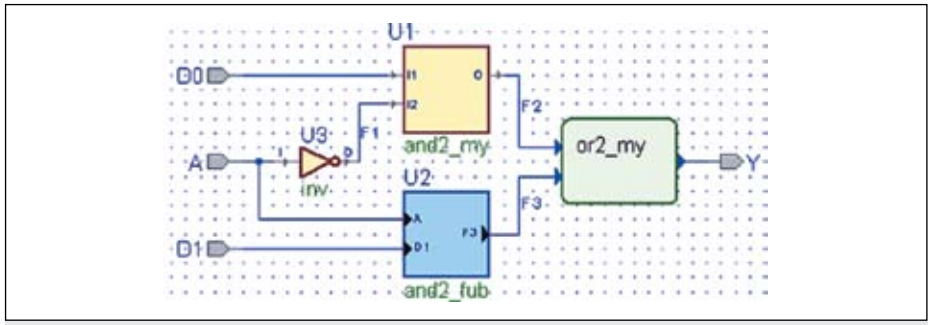

Рис. 8. Заменяем системный элемент or2 из встроенной библиотеки графическим процессом or2_my

Рис. 9. Панель Libraries показывает список доступных схемному редактору библиотек

На рис. 9 видно, что две библиотеки — **lesson_1** и **lesson_7** — подключены к «ящику символов», а библиотека **lesson_2** отключена от него. Об этом можно судить по сброшенному флажку lesson_2.

Ниже расположены глобальные библиотеки системы. Они отсортированы в алфавитном порядке по именам фирм, выпускающих соответствующие компоненты. Если имя выделено жирным шрифтом (на рис. 9 строка XILINX), то как минимум одна библиотека этой фирмы подключена к ящику символов. В нашем примере такой библиотекой является **spartan3**. Здесь тоже пришлось пожертвовать точностью и немного изменить рисунок.

Контекстное меню панели **Libraries** содержит команды **Select All** и **Deselect All**, позволяющие двумя щелчками мыши присоединить или отсоединить сразу все библио-теки. Вторая команда находит практическое применение, когда нужно «почистить» «ящик символов», а вот первую лучше не использовать: слишком долго придется ждать, пока загрузятся все библиотеки.

Рассмотрим еще одну реальную ситуацию. Допустим, у вас имеется библиотека собственных символов **my_lib**, и вы хотите сделать ее доступной в текущем проекте. Желаемого результата можно достичь двумя способами.

Вероятно, самый простой путь — подключить ее как обычный проект к рабочему пространству. Позднее (урок 15) мы узнаем, что любая библиотека организована «по образу и подобию» проекта, и, следовательно, с ней можно манипулировать как с обычным проектом.

А коль дела обстоят таким образом, то выделим в окне **Design Browser** строку **Workspace 'Lessons'**, щелкнем по ней правой кнопкой мыши и выполним команду **Add Existing Design to Workspace** («Добавить к рабочему пространству существующий проект»).

В файловой структуре своего компьютера найдем нужную библиотеку (например, файл **my_lib.adf**) и подключим ее к рабочему пространству.

Вы обнаружите, что на панели **Libraries** в раздел **User Libraries** добавилось имя вашей библиотеки. Остальное, как говорится, «дело техники». Установите рядом с ней флажок и наслаждайтесь результатами своего труда.

Другой способ немного сложнее, но только потому, что вы еще не знакомы с менеджером библиотек. В окне **Library Manager** выполните команду **Attach Library** (присоединить библиотеку) из меню **Library** (рис. 10).

Рис. 10. Команда Attach Library

Найдите нужную библиотеку (файл **my_lib.lib**) и откройте ее. Она должна появиться в списке доступных библиотек. Щелкните по ней **ПКМ** и выберите команду **Add to Symbols Toolbox**. Проверьте: библиотека должна стать доступной схемному редактору.

«Ящик символов» (Symbols Toolbox)

А теперь уделим более пристальное внимание «ящику символов» (рис. 11). По уроку 1 мы уже знаем, что он состоит из двух основных окон.

Верхнее окно панели **Symbols Toolbox** показывает список присоединенных к редактору библиотек. По умолчанию они находятся в свернутом виде ⊡ (**collapse**). При желании вы можете раскрыть ⊡ (**expand**) любую библиотеку и посмотреть ее содержимое. Команды контекстного меню **Expand All** и **Collapse All** позволяют выполнять названные операции одновременно со всеми библиотеками, присоединенными к «ящику символов».

Нижняя часть панели называется **Symbol Preview** и обеспечивает предварительный просмотр символа, который был выбран в списке символов. Окно предварительного просмотра может быть включено или выключено командой **Preview Visible** из контекстного меню.

Не является для нас новостью и то, что далеко не всегда библиотечные модули (имеются в виду компоненты) укомплектованы соответствующими символами. Это касается, прежде всего, компонентов, которые еще ни разу не использовались в схеме. Такие модули хранятся в специальном разделе библиотеки, называемом **Units without symbols** («Модули без символов»).

Несмотря на то, что символа еще нет, окно предварительного просмотра покажет его виртуальный образ. При первом вызове такого компонента на схему система автоматически сгенерирует его символ и удалит его из раздела «Модули без символов».

Заголовок списка компонентов (рис. 11) по умолчанию содержит три столбца: имя компонента (**Name**), имя библиотеки (**Library**), где он находится, и краткую информацию о его назначении (**Description**). В целях экономии места два из трех столбцов можно удалить.

Для облегчения поиска нужного компонента список библиотек можно сортировать по алфавиту, как в прямом, так и в обратном порядке. Независимо от этого процесса выполняется сортировка компонентов всех библиотек.

Более того, если вы выполните команду **Group by Libraries** («Сгруппировать библиотеки»), то получите сводный список, содержащий компоненты из всех библиотек, подключенных к «ящику символов».

Но быстрее всего вы получите результат, если напечатаете имя искомого объекта в окне быстрого поиска — **Quick Find box**. Оно расположено вверху, на панели **Symbols Toolbox**.

Здесь может оказаться полезным режим поиска с учетом регистра (команда **Case Sensitive Search**). Например, напечатав **and2**, вы получите символ из встроенной библиотеки, а если введете **AND2** — из библиотеки **spartan3** фирмы **Xilinx**.

Отметим еще одну полезную особенность «ящика символов». Он дает возможность «на лету» отредактировать нужный вам сим-

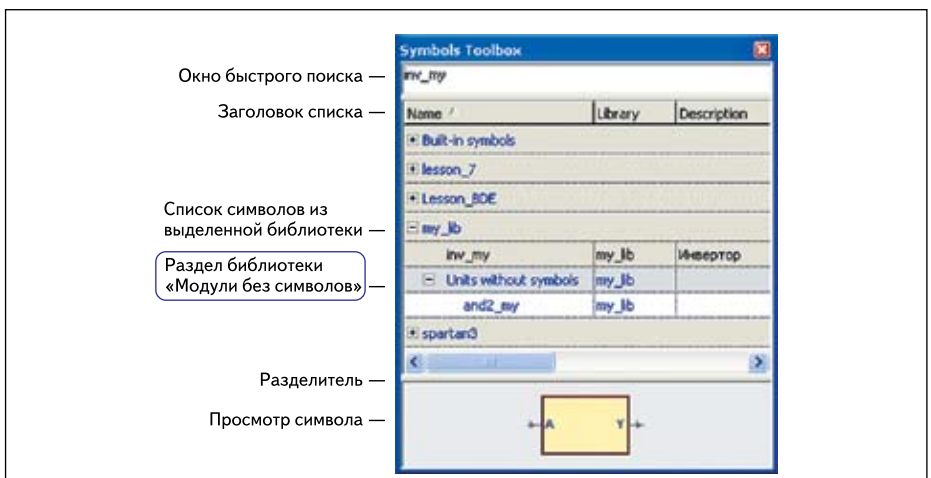

Рис. 11. Диалоговая панель «ящика символов» (Symbols Toolbox)

вол. Щелкните по нему **ПКМ** и выполните команду **Edit Symbol**.

Система перейдет в режим редактирования символа, но как только вы сохраните внесенные изменения и вернетесь на панель **Symbols Toolbox**, то обнаружите, что имеете дело уже с обновленной версией символа.

Проделаете то же самое с каким-нибудь встроенным символом, чтобы убедиться, что система не разрешает его редактировать. В контекстном меню вы просто не увидите команду **Edit Symbol**.

Если вызвать контекстное меню для компонента, у которого еще нет символа, то вместо команды **Edit Symbol** вы увидите другую команду — **Generate Symbol**. Значит, можно создать символ, не размещая его на схеме.

Массивы символов и иерархических блоков

Заканчивая разговор о размещении символов на схеме, нельзя умолчать об одной весьма полезной особенности, которая доселе не встречалась в других САПР, таких как **DesignLab 8.0** или **OrCAD 9.1**.

Нередко, проектируя схему, нужно разместить на ней много идентичных компонентов, например, триггеров при рисовании многоразрядного регистра или шинных драйверов, обеспечивающих высокоомный выход на системную шину.

Как вы поступите? Разместите один символ, а затем создадите нужное число копий? Это приемлемый вариант, если вы не забыли про команду **Duplicate**. Она позволяет разместить на схеме нужное число символов в «строку» или «столбец», а также создать целую матрицу однотипных элементов. Но место на схеме вам вряд ли удастся сэкономить. Между тем есть способ это сделать. Поместите на схеме первый символ, например, инвертор **inv** из встроенной библиотеки (рис. 12а), и вызовите для него панель **Symbol Properties**.

В разделе **Array of instances** установите флажок **Create array of…** и задайте нужное число экземпляров данного компонента, например восемь копий (рис. 13). Обратите внимание, возле символа **inv** появился «текст» *8 (рис. 12б), указывающий на то, что теперь это не один символ, а восемь одинаковых экземпляров, «положенных один на другой». Конечно, это образное выражение.

Подрисуйте к выводам массива символов отрезки шин подходящей ширины (рис. 12в). Заметим, что массивы можно создавать не только для встроенных символов, но и для любых других (системных и пользовательских).

Рис. 13. Задаем необходимое число экземпляров символа (фрагмент)

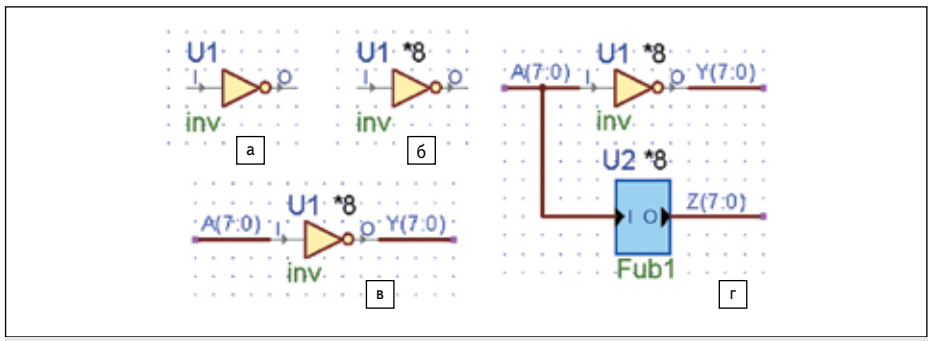

Рис. 12. а, б) Создание массива из восьми инверторов; в) подключение шины; г) добавление восьми копий иерархического блока Fub1

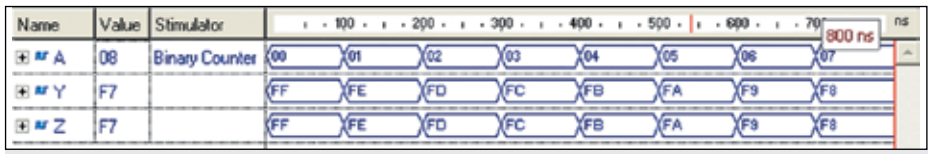

Рис. 14. Результаты моделирования массива инверторов и иерархических блоков

Более того, эта операция разрешена и для иерархических блоков. Единственное отличие: шины надо подводить не к выводам (их просто нет), а к портам иерархического блока (рис. 12г).

Промоделируйте эту простую схему. Все работает как надо (рис. 14): на выходах **Y** и **Z** наблюдается поразрядная инверсия данных, поступающих от входной шины **A**.

Загляните в файл с **VHDL**-кодом, автоматически сгенерированным по схемному описанию (рис. 15). Правда, ничего особенного вы там не увидите. В архитектурном теле объявлен компонент **Fub1** — иерархический блок (строки 29–32), а в исполняемом разделе вызываются восемь его экземпляров с метками реализации от **U2_ARRAY0** до **U2_ARRAY7** (строки 41–45). Эти метки тоже сгенерированы самой системой.

С экземплярами встроенного компонента inv дела обстоят еще проще: в исполняемый раздел вставляется столько булевских выражений, сколько проводников имеет шина (строки 37–39).

Кроме символов, иерархических блоков и графических процессов (рис. 8) на схеме обычно присутствуют входные и выходные клеммы (**Terminals**), обеспечивающие интерфейс с внешней средой, а нередко и специальные символы питания и «земли», исполняющие роль генераторов нуля и единицы.

```
architecture test_array_inv of test_array_inv is
    component Fub1
        port (I : in STD_LOGIC;
              O : out STD_LOGIC);
    end component;
    signal A : STD_LOGIC_VECTOR (7 downto 0);
    signal Y : STD_LOGIC_VECTOR (7 downto 0);
    signal Z : STD_LOGIC_VECTOR (7 downto 0);
begin
    Y(0) <= not(A(0));
-- аналогичным образом описаны остальные экземпляры
    Y(7) <= not(A(7));

    U2_ARRAY0 : Fub1
    port map(I => A(7),O => Z(7));
-- аналогичным образом описаны остальные экземпляры
    U2_ARRAY7 : Fub1
    port map(I => A(0),O => Z(0));
end test_array_inv;
```

Рис. 15. VHDL-код, сгенерированный системой по схемному описанию, содержащему массивы символов и иерархических блоков (после серьезной правки)

Символы питания и «земли»

По умолчанию они имеют имена **VCC** и **GND**. Понятно, что их функция более скромна, нежели символов компонентов или иерархических блоков. И тем не менее, они довольно часто применяются в качестве генераторов логических уровней — нуля (**GND**) и единицы (**VCC**).

Заметим, что заданные по умолчанию имена **VCC** и **GND** легко сменить на более удобные, например, **HI** (от слова **high** — высокий) и **LO** (от слова **low** — низкий). А выполнить такую операцию можно прямо на схеме.

Чтобы подать желаемый уровень на какой-либо проводник, его нужно подключить к единственному выводу символа питания (рис. 16а) или назвать проводник именем символа питания (рис. 16б).

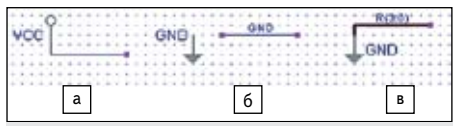

Рис. 16. Работа с символами питания

Следует добавить, что все правила, рассмотренные в следующем уроке для клемм, применимы и по отношению к символам питания. Но одну оговорку придется сделать. Оказывается, символы питания могут работать на шину.

Показанный на рис. 16в фрагмент схемы вполне работоспособный. Низкий уровень в данной ситуации будет назначен всем четырем проводникам шины, и в результате схема получается весьма удобной и компактной. ■

Продолжение следует