

Продолжение. Начало в № 3 `2009

Изучаем Active-HDL 7.1.

Урок 6. Инструменты, повышающие эффективность создания HDL-моделей

Александр ШАЛАГИНОВ
shalag@vt.cs.nstu.ru

Современные цифровые технологии автоматизированного проектирования в корне отличаются от традиционного подхода, при котором нередко вручную приходилось рисовать схемы, создавать спецификации и карты прошивки, выполнять синтез на картах Карно и чертить эпюры сигналов.

Разработчики аппаратуры «старой закалки» с недоверием, а порой и с неприязнью относились к САПР, как к большим и дорогостоящим «игрушкам». Заниматься написанием программного HDL-кода они считали ниже своего достоинства. «Пусть моделированием занимаются те, кто не способен к разработке», — говорили они (цитата из книги Дж. Армстронга).

Теперь эти позиции приходится пересматривать, и, хочешь — не хочешь, заниматься программированием, точнее, разработкой программных моделей на языках описания аппаратуры (чаще всего это языки VHDL и Verilog, а в последнее время — еще и Handel-C или SystemC).

Но схемотехник не хочет, а порой, и не может переквалифицироваться в программиста, ведь речь идет о смене образа мыслей разработчика.

Поэтому, чтобы сделать такой переход менее болезненным, создатели САПР идут на определенные компромиссы: предлагают пользователям альтернативные способы описания проекта — графическое представление в виде любимой разработчиком схемы или диаграммы состояний цифрового автомата.

А для облегчения текстового описания проекта на языках HDL, от которого в любом случае никуда не деться, предлагаются различные

встроенные инструменты, такие как **Language Assistant** (языковой помощник) или **IP Core Generator** (генератор интеллектуальных блоков-ядер).

Первый инструмент содержит некоторое множество шаблонов типовых конструкций языков HDL, из которых наподобие конструктора можно создавать исходный код, не вникая в тонкости используемого языка.

Второй инструмент вообще не требует знания языков описания аппаратуры, так как желаемый код генерируется автоматически. Вот об этих инструментах и пойдет разговор.

Языковой помощник


Языковой помощник можно вызвать в отдельном окне (рис. 1), отыскав его пиктограмму  в меню **Standard** или активизировав соответствующую команду из главного меню **Tools/Language Assistant** (рис. 2).





Рис. 2. Меню Tools/Language Assistant

Для выполнения одной услуги предпочтительнее вызывать помощник из контекстного меню, предварительно щелкнув правой кнопкой мыши в месте вставки шаблона на рабочей области редактора.

Если развернуть раскрывающийся список (рис. 1), то можно увидеть, что языковой помощник поддерживает много различных языков, однако мы ограничимся только шаблонами языка VHDL.

Впрочем, ранее упоминалось о возможности использовать этот инструмент при создании макрофайлов (урок 3, рис. 9). И сейчас мы, наконец, до него добрались.

Все множество шаблонов разбито на несколько разделов: **Code Auto Complete**, **Language templates** и т. д. Шаблоны каждого раздела хранятся в отдельном файле с тем же именем и имеют расширение *.tpl (от слова **template**).

Неиспользуемые разделы можно отсоединить (кнопка **Detach** ) , не удаляя их с диска, а когда они понадобятся, присоединить вновь (кнопка **Attach** ).

Развернув любой раздел, вы увидите список доступных в нем шаблонов. Выбрав желаемый шаблон, вы получите в правом окне помощника информацию о его содержимом (рис. 3).

Чтобы вставить текст выбранного шаблона в создаваемый или редактируемый документ, нужно указать в нем место вставки и нажать

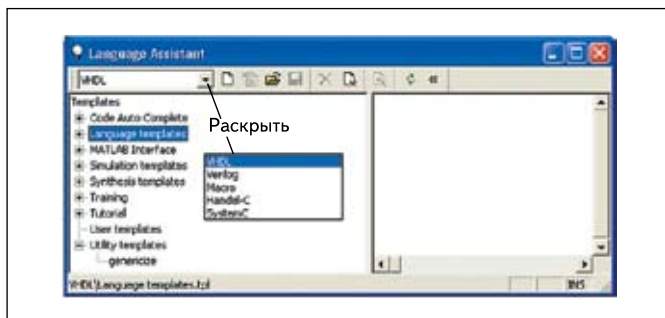


Рис. 1. Языковой помощник активизирован в отдельном окне пакета Active-HDL 7.1

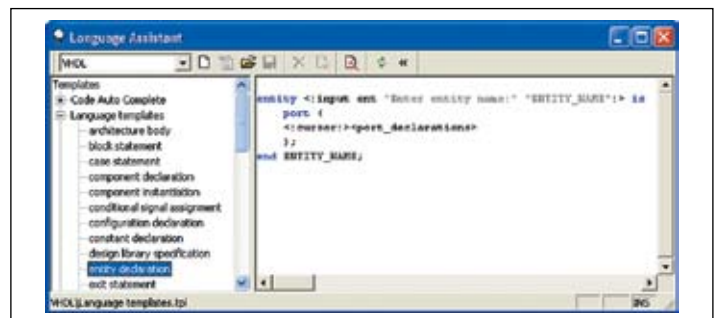


Рис. 3. В правом окне помощника Language Assistant показано содержимое выбранного шаблона

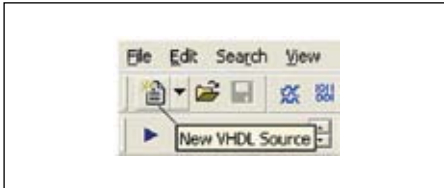


Рис. 4. Откроем пустой VHDL-документ

на кнопку **Use template**. Можно поступить иначе: щелкнуть правой кнопкой мыши на имени выбранного шаблона и исполнить команду **Use**.

Наконец, можно просто «прижать» указателем мыши нужный шаблон и отбуксировать его методом **drag & drop** в окно HDL-редактора.

Попробуем с помощью ассистента-знатока (это другое название **Language Assistant**) сконструировать VHDL-код уже знакомого нам по первому уроку мультиплексора **mux2_HDL**. Не обращаясь к услугам мастера **New Source File Wizard**, откроем пустой VHDL-документ, щелкнув на пиктограмме **New VHDL Source** (рис. 4). Мы уже представляем себе структуру VHDL-кода, который надо написать (урок 5, рис. 14). Сначала объявляются библиотека **library** и стандартный пакет **STD_LOGIC_1164**, потом задается объект проекта **entity** и, наконец, код архитектурного тела **architecture**. В таком порядке и будем действовать.

Вызовем программу языкового помощника и раскроем раздел **Language templates** (рис. 3). Развернем ветвь **library, packages** и найдем шаблон **IEEE: STD_LOGIC_1164**. В правом окне помощника появится содержимое выделенного шаблона (рис. 5). Перенесем его в рабочий документ. Можно выполнить эту работу и методом **drag & drop**.

Если вам не нужен сопровождающий шаблон комментариев, то следует поступить иначе. Выделите в шаблоне только нужный текст (в данном случае нижние две строки) и отбуксируйте его в окно HDL-редактора.

Затем найдите в этом же разделе шаблон **entity declaration** и методом **D&D** перенесите его в окно HDL-редактора. Прежде чем поместить шаблон в указанное курсором мыши место, редактор запросит его имя (рис. 6).

Вместо формального имени **ENTITY NAME** введем имя объекта проекта, в нашем примере **mux2_HDL**, и нажмем кнопку **OK**.

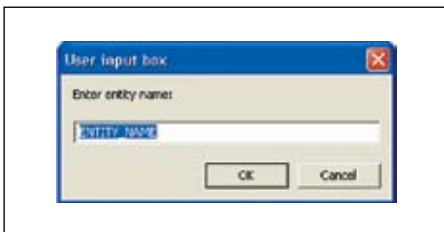


Рис. 6. Вместо формального имени ENTITY NAME напечатает свое имя — mux2_HDL

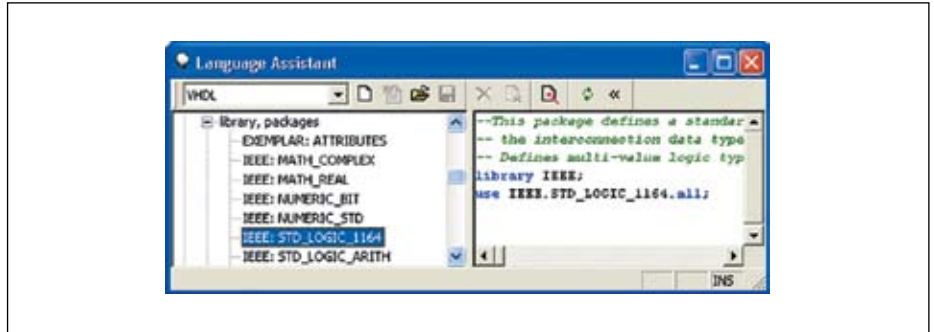


Рис. 5. Находим первый шаблон: библиотеку IEEE и стандартный пакет STD_LOGIC_1164

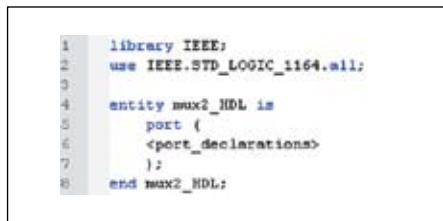


Рис. 7. В рабочем окне HDL-редактора появилось описание объекта проекта

В окне текстового редактора появится интерфейс объекта проекта с ключевым словом **entity** (рис. 7).

Аналогичным образом добавляем в код шаблон с декларацией архитектурного тела объекта — **architecture body**. Правда, сейчас диалог будет немного длиннее, потому что надо ответить на два запроса об имени архитектуры (**ARCH_NAME**) и об имени объекта проекта (**ENTITY NAME**). Назовем архитектурное тело другим словом, например **behavior** (поведение). С помощью кнопки **comment**, закоментируем еще не реализованные фрагменты кода (рис. 8), сохраним созданный файл и выполним пробную компиляцию (кнопка **compile**).

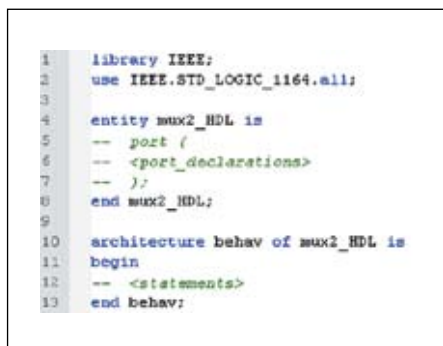


Рис. 8. Закомментируем не реализованные пока фрагменты кода

Если ошибок нет, пойдём дальше. В разделе **entity** нам предлагается декларировать входные и выходные порты (рис. 8, шестая строка). Однако подходящего для этих целей шаблона нет ни в одном разделе языкового помощника. Но мы его можем сделать сами.

Создание собственных шаблонов в языковом помощнике

Наверное, вы обратили внимание, что среди стандартных разделов помощника есть один пустой с названием **User templates** (рис. 1).

Вот в нем-то мы и создадим новый шаблон. Щелкнем правой кнопкой мыши на имени **User templates** и выполним команду **New template**. Введем имя нового шаблона, например, **port_my**, а в правое окно помощника скопируем описания портов **D0**, **D1**, **A** и **Y** из файла **mux2_HDL.hdl** (урок 5, рис. 6). Сохраним созданный шаблон и проверим его работу в деле. Снимем комментарий со строк 5–7 (рис. 8) и вместо текста **'port_declarations'** вставим только что созданный шаблон. Повторной компиляцией убедимся, что ошибок нет.

Вам не терпится высказать критические замечания в адрес выполненной работы? Согласимся — это решение нельзя назвать изящным. Полученный результат легко достигается простой операцией **Copy – Paste** через буфер обмена.

Обсудим иные возможные варианты. В пакете **OrCAD 9.1** тоже имеется языковой помощник, правда, у него другое название — **VHDL Samples**. В нем есть шаблон **Design Unit**, в котором объединены все рассмотренные нами конструкции (рис. 9). Обратите внимание, список портов не пустой, в него включен один входной **PORT1_NAME** и один выходной **PORT2_NAME** сигналы. Описания недостающих портов легко добавить простым копированием.

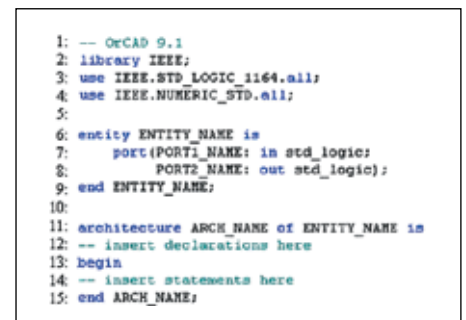


Рис. 9. Содержимое шаблона Design Unit из пакета OrCAD 9.1

В другой САПР Xilinx ISE 7.1 аналогичный инструмент называется **Language Templates**, но в нем нет таких громоздких конструкций, как в OrCAD.

Зато подробно представлены описания портов (входных, выходных и двунаправленных), как для одиночных, так и для шинных сигналов (от одного до 64 бит). На рис. 10 показаны примеры описания одиночных сигналов.

```
<port_name> : in std_logic;
<port_name> : out std_logic;
```

Рис. 10. Шаблоны описания одиночных сигналов (портов) в САПР Xilinx ISE 7.1

Выберем компромиссный вариант и создадим шаблон, включающий один входной и один выходной порты (рис. 9, строки 7 и 8). Но в отличие от пакетов OrCAD 9.1 и Xilinx ISE 7.1 у нас есть возможность сделать шаблон интерактивным (диалоговым).

Для этих целей в САПР Active-HDL 7.1 есть любопытный инструмент, названный загадочным словом **genericize**. Он находится в разделе **Utility templates** (рис. 2, внизу).

Сначала в HDL-редакторе напишем код, который должен сгенерировать разрабатываемый шаблон (рис. 11а). Затем выделим текст (имя, значение или выражение), который мы хотим заменить входным тегом, в нашем примере — это имя **PORT_INPUT_NAME**. Щелкнем на выделенном тексте (рис. 11б) правой кнопкой мыши и в открывшемся контекстном меню выполним команду: **Language Assistant/Utility templates/genericize**.

На трех последующих диалоговых панелях вводим необходимую информацию: метку **Label** (рис. 12а), описание запроса **Request caption** (рис. 12б) и значение по умолчанию **Default value** (рис. 12в).

По окончании диалога на месте выделенного текста появится входной тег **<input>** (рис. 13).

Обратите внимание на формат входного тега. Он выглядит так: **<input [label] "parameter_description" ["default_value"]>**.

```
a port {
  PORT_INPUT_NAME : in STD_LOGIC;
  PORT_OUTPUT_NAME : in STD_LOGIC
};

б port {
  PORT_INPUT_NAME : in STD_LOGIC;
  PORT_OUTPUT_NAME : in STD_LOGIC
};
```

Рис. 11. Создание нового шаблона port_interactive

```
port {
  <input 1 "Введите имя входного порта" "D0"> : in STD_LOGIC;
  PORT_OUTPUT_NAME : out STD_LOGIC
};
```

Метка Описание запроса Значение по умолчанию

Рис. 13. Выделенное имя PORT_INPUT_NAME заменяется входным тегом <input> интерактивного шаблона

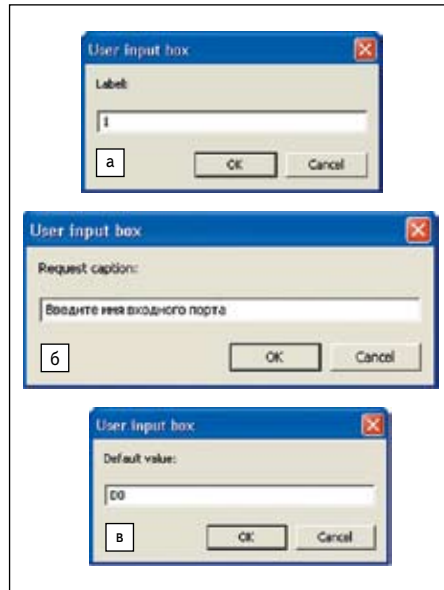


Рис. 12. В диалоговом режиме задаем параметры входного тега будущего шаблона

Сравните его с аналогичной строкой, показанной на рис. 13.

В описании входного тега опции **label** и **"default_value"** заключены в квадратные скобки. Это означает, что они не являются обязательными, то есть на первой и третьей диалоговых панелях мы могли не вводить имя метки и значение по умолчанию, оставив соответствующие поля пустыми.

Повторим описанную процедуру для выходного порта **PORT_OUTPUT_NAME**. В результате будет создан еще один входной тег (рис. 14). В принципе, входных тегов в шаблоне может быть сколько угодно.

Работа практически закончена, осталось сделать последний штрих, да и то только в том случае, если вас интересует позиция курсора редактирования, которую он займет после вставки шаблона.

В нашем примере (урок 5, рис. 6) надо ввести не один, а три входных порта **D0**, **D1** и **A**, поэтому курсор редактирования желательно позиционировать после строки с описанием первого входного порта.

```
port {
  <input 1 "Введите имя входного порта" "D0"> : in STD_LOGIC;
  <input 2 "Введите имя выходного порта" "Y"> : out STD_LOGIC
};
```

Рис. 14. Конструируем еще один входной тег для порта PORT_OUTPUT_NAME

Эту функцию в языковом помощнике выполняет другой тег, который так и называется — курсор **<cursor>**.

Поставьте его в то место, где должен оказаться курсор редактирования (не путайте с указателем мыши) после вставки шаблона, то есть в конец строки с описанием входного порта (рис. 15).

Вот и все, текст шаблона готов, осталось облечь его в соответствующую форму. Нам уже приходилось однажды создавать новый шаблон, так что эта операция знакома. Щелкнем правой кнопкой мыши на имени **User templates** и выполним команду **New template**.

Введем имя шаблона, например, **port_interactive**, а в правое окно помощника скопируем текст, показанный на рис. 15. Сохраним созданный шаблон и проверим его в работе. Обратите внимание на курсор редактирования, он находится именно там, где мы и хотели его видеть: в конце строки описания входного порта.

Вероятно, вас интересует вопрос о назначении необязательного параметра **label**. Где он находит применение? Мы рассмотрим только одну ситуацию, где польза от его применения не вызывает сомнений.

Во многих конструкциях языка VHDL одно и то же имя приходится задавать несколько раз. Такими конструкциями, в частности, являются известные нам блоки **entity** и **architecture** (рис. 8).

Чтобы дважды не вводить одно и то же имя, достаточно в различных входных тегах поставить одинаковые метки. Тогда данные, введенные в первом теге, автоматически распространятся и на все другие входные теги с той же самой меткой.

Прodelайте простой эксперимент, отредактируйте только что созданный шаблон **port_interactive**, заменив во втором теге значение метки 2 на 1. Вставьте скорректированный шаблон в VHDL-код и вы увидите результат: выходной порт получит такое же имя, что и входной порт:

```
port (
  D0: in STD_LOGIC;
  D0: out STD_LPGIC
);
```

```
port {
  <input 1 "Введите имя входного порта" "D0"> : in STD_LOGIC;<cursor>;
  <input 2 "Введите имя выходного порта" "Y"> : out STD_LOGIC
};
```

Тег <cursor> указывает позицию курсора редактирования после вставки шаблона

Рис. 15. Тег <cursor> определяет позицию курсора редактирования после вставки шаблона

Заметим мимоходом, что второго диалога не было вовсе, а сам эксперимент нельзя назвать корректным: компилятор не пропустит два одинаковых названия портов в одном блоке.

Генератор интеллектуальных блоков

В отличие от языкового помощника, который конструирует HDL-код из готовых шаблонов, IP Core generator в прямом смысле является генератором кода. Если шаблон — это полуфабрикат, небольшой отрезок исходного текста, то результатом работы генератора является полностью готовый для применения модуль.

Мало того, что такие модули могут использоваться в любой VHDL- или Verilog-системе, они построены на синтезируемом подмножестве языка, а значит, пригодны для инструментов синтеза и ПЛИС-технологий. Это действительно тот редкий случай, когда пользователь может не знать языка, на котором создается выходной продукт, что не мешает ему получить положительный результат.



Рис. 16. Выпадающее меню Tools

Для запуска программы IP Core generator достаточно исполнить аналогичную команду из выпадающего меню Tools инструментальной панели Standard (рис. 16).

Вообще-то в пакете Active-HDL имеется подходящая для этих целей иконка, но в стандартной настройке она скрыта. Тем не менее, вы можете отыскать ее, щелкнув правой кнопкой мыши на любой из инструментальных панелей и вызвав затем диалоговую панель Customize (рис. 17). Методом drag & drop отбуксируйте пиктограмму на удобную для вас позицию, например, рядом с иконкой Language Assistant.

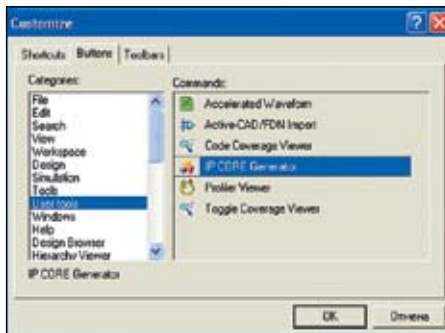


Рис. 17. Вызываем панель Customize, чтобы поместить кнопку IP Core Generator на инструментальную панель Standard

Начальное окно IP Core Generator содержит меню доступных для генерации модулей (на рис. 18 слева).

Кроме того, здесь же справа находятся три раздела с описанием программы: общее представление (IP Core Generator Overview), начало работы (Getting Started) и описание модулей (Module Descriptions).

Все множество модулей, которые может генерировать данный инструмент, тоже разбито на несколько разделов: Basic Elements, Arithmetical Functions, Sequential Logic, Memory Blocks и т. д. На рис. 18 первый раздел раскрыт, так что видны входящие в него простые вентили и комбинационные схемы.

Кстати, здесь же находится мультиплексор (Multiplexer), который мы рассматривали ранее. Щелкнем по названию модуля Multiplexer, чтобы вызвать его графическое изображение (рис. 19).

Познакомимся с основными разделами, видимыми на экране монитора. В разделе 1 выбирается язык описания аппаратуры (поле Language), задаются имена объекта проекта (поле Entity/Module name) и архитектурного тела (поле Architecture name).

В поле Vendor («Поставщик») по умолчанию установлено значение Independent («Независимый»), причем других значений в выпадающем списке обычно нет. Последнее означает, что большинство доступных модулей генерируется так, чтобы они не зависели от конкретного поставщика.

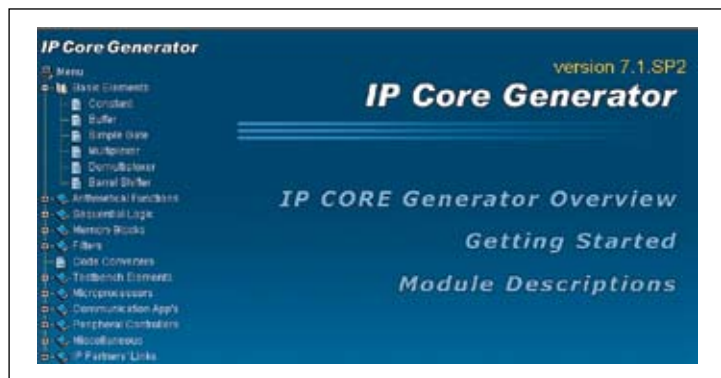


Рис. 18. Стартовое окно программы IP Core Generator

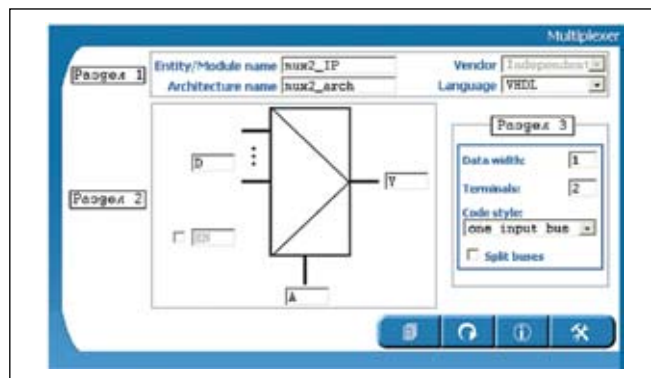


Рис. 19. Графическое представление модуля Multiplexer

Обратите внимание, в предыдущих версиях программы IP Core Generator по умолчанию предлагался язык VHDL, а в последней — Verilog. Это говорит о том, что популярность Verilog постоянно растет, и сейчас оба языка делят рынок почти поровну.

В разделе 2 показано условное графическое изображение генерируемого модуля, его входные и выходные сигналы (порты). Возле некоторых портов могут стоять флажки (check box).

Мультиплексор тоже имеет такой порт — это вход разрешения работы EN (от слова Enable). Если флажок установлен, то порт будет включаться в генерируемый код. Если флажок сброшен, то данный порт не будет участвовать в генерируемом коде (как будто этого входа нет).

Если вы хотите, чтобы мультиплексор имел не прямой, а инверсный выход, то надо подвести указатель мыши к требуемому выводу (в нашем случае к выходу Y), убедиться, что появилась подсказка Click here to change polarity («Щелкните здесь, чтобы сменить полярность»), и нажать левую кнопку мыши (рис. 20). Такую же операцию можно было выполнить и над входом EN, но мы его заблокировали.

Понятно, что раздел 2 разрешает заменять заданные по умолчанию имена портов на любые другие, лишь бы они не противоречили синтаксису рабочего языка описания

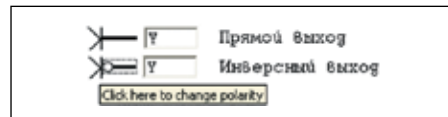


Рис. 20. Появление подсказки Click here to change polarity

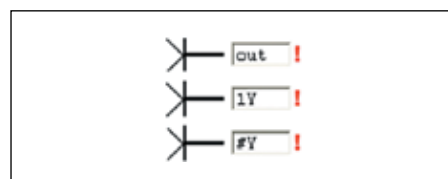


Рис. 21. Примеры неразрешенных к использованию имен



Рис. 22. Программа IP Core generator фиксирует ошибки при вводе имени порта

аппаратуры. В качестве имен нельзя, например, использовать зарезервированные слова, такие как **out** или **case**, имя также не должно начинаться с цифры или со служебного символа (рис. 21).

Впрочем, программа контролирует правильность ввода и обнаружит ошибку (рис. 22), как только вы покинете поле ввода. При этом рядом с неправильным именем появится восклицательный знак «!» красного цвета.

В разделе 3 объединены индивидуальные опции, задаваемые для каждого модуля. Для мультиплексора этот раздел очень простой. Параметр **Data Width** задает ширину шин коммутируемых данных (по умолчанию — одиночные сигналы). В поле **Terminals** указывается число коммутируемых каналов. В поле **Code Style** выбирается стиль кодирования.

Флажок **Split buses** («Разделенные шины») по умолчанию сброшен. Это означает, что шинные сигналы будут представляться в виде жгута, объединяющего все входящие в шину проводники. Шина будет представлена как вектор, то есть одним портом, имеющим тип **std_logic_vector**.

При установленном флажке **Split buses** шина предстанет разбитой на группу независимых проводников, входящих в шину, а соответствующие им порты будут иметь тип **std_logic**.

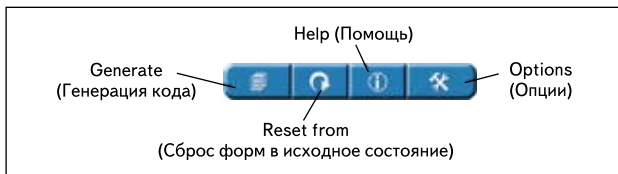


Рис. 23. Управляющие кнопки программы IP Core Generator

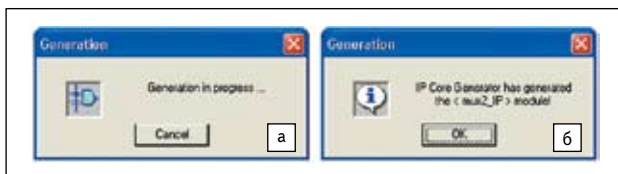


Рис. 25. IP Core генератор выполняет процесс генерации кода

В правом нижнем углу окна с графическим изображением генерируемого модуля (рис. 19) находятся четыре управляющие кнопки (рис. 23). Если при заполнении каких-то полей вы почувствуете затруднения, всегда можно воспользоваться услугами встроенной справочной системы, нажав кнопку **Help**.

Экспериментируя с заполнением различных полей, вы можете захотеть вернуть все на прежние места. Тогда вам понадобится кнопка **Reset form** (рис. 23). Единственное поле, в котором не восстанавливаются первоначальные данные, — **Language**.

Что означает «главная» кнопка **Generate**? Она запускает процесс генерации модуля, включающий три шага. Сначала **IP Core** генератор проверит, не осталось ли ошибок при вводе имен и значений параметров. Если вы были невнимательны, работая с программой в интерактивном режиме, **IP Core** генератор прекратит процесс генерации, выдав соответствующий отчет об ошибках (рис. 24).



Рис. 24. IP Core генератор обнаружил не устраненные при вводе ошибки

Если ошибок нет, начинается основная стадия — генерация кода. В течение этой стадии на экране видно окно **Generation** (рис. 25а). Успешное создание модуля подтверждается соответствующим сообщением (рис. 25б).

На третьей стадии **IP Core** генератор создает необходимые файлы и папки, куда и помещает результаты своей работы. По умолчанию создается папка **IP_CORE** в корне дерева активного проекта. Ее можно обнаружить в окне просмотра проекта **Design Browser** (рис. 26а).

В принципе можно создать несколько версий одного и того же модуля с различными конфигурациями. Разрешается даже остав-

лять имя модуля без изменения. Каждая новая версия модуля помещается в отдельные папки (**mux2_IP**, **mux2_IP0**, **mux2_IP1**, **mux2_IP2** и т. д.) внутри основной папки **IP_CORE** (рис. 26б).

Внутри каждой папки по умолчанию помещаются три файла (рис. 26а). Файл с расширением ***.vhd** содержит сгенерированный **VHDL**-код. Для нашего примера — это файл **mux2_IP.vhd** (рис. 27).

```

1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  entity mux2_IP is
4      port (
5          D : in std_logic_vector(1 downto 0);
6          A : in std_logic;
7          Y : out std_logic
8      );
9  end entity;
10 library IEEE;
11 use IEEE.std_logic_unsigned.all;
12 architecture mux2_arch of mux2_IP is
13 begin
14     Y <= D[0] when (A = '0') else D[1];
15 end architecture;

```

Рис. 27. VHDL-код, созданный программой IP Core Generator

В данном варианте решения функция мультиплексора реализуется с помощью оператора условного назначения (рис. 27, строка 36). Откомпилируйте полученный код, чтобы убедиться, что в нем нет ошибок.

Второй файл — **mux2_IP.bds** — содержит графическое изображение сгенерированного модуля (рис. 28). Однако если вы с помощью программы **Library Manager** откроете библиотеку проекта, то обнаружите, что в ней нет никакого символа.

Чтобы поместить графическое изображение компонента в библиотеку проекта, надо запус-

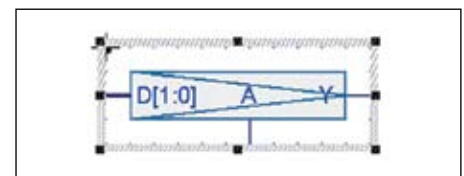


Рис. 28. Графическое изображение сгенерированного модуля

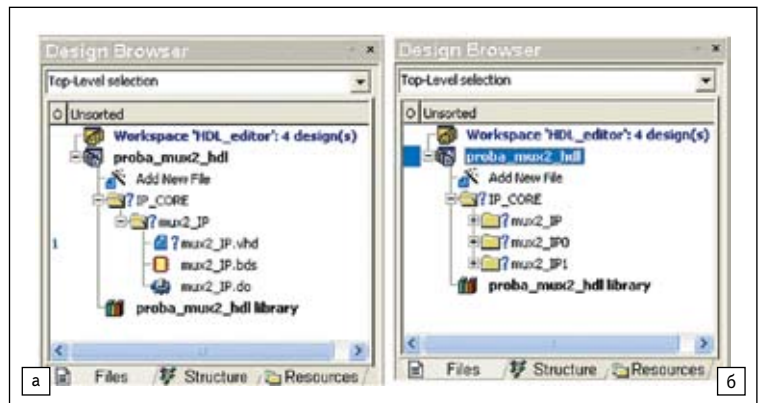


Рис. 26. IP Core генератор создает свою папку IP_CORE в файловой структуре активного проекта

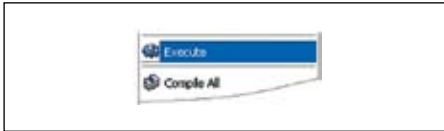


Рис. 29. Команда Execute

тить на исполнение третий файл `mux2_IP.do`, содержащий соответствующие макрокоманды (команда **Execute** на рис. 29).

По окончании работы в окне **Console** появится сообщение о том, что символ сохранен в рабочей библиотеке проекта:

```
# Symbol mux2_IP saved to library proba_mux2_hd1
```

Настройка параметров генератора интеллектуальных блоков

Если вас не устраивают заданные по умолчанию настройки **IP Core** генератора, то следует нажать на кнопку **Options** (рис. 23). Появится панель **IP Core Generator Configuration**, включающая три раздела (рис. 30).

Левую часть панели занимает раздел **Folders Options**, где можно изменить имя задаваемой по умолчанию папки **IP_CORE** на любое дру-

гое. Проследите, чтобы флажок **Specify the destination file** был сброшен. Напомним, папка **IP_CORE** создается внутри активного проекта (в папке **src**).

Если вы хотите вручную специфицировать месторасположение выходных файлов, то следует установить флажок **Specify the destination file**. Процесс генерации теперь не будет запускаться автоматически, так как программа должна уточнить, где вы собираетесь сохранять результаты генерации модуля.

При выборе режима **Set src design folder as root folder** (рис. 30) откроется диалоговая панель **File Browser for VHDL Modules** (рис. 31), где вы можете создать новую директорию или поместить генерируемые файлы (а их может быть несколько) в корневую папку **SRC** (от слова **source** — исходный).

Установив флажок **Compile file after generation**, вы автоматически получите и откомпилированный файл. В общем, в этом режиме появляется дополнительная свобода, и возможности ваших действий расширяются.

Наконец, вы можете выбрать режим **Set folder below as initial destination folder** и назначить папку в любом месте на жестком диске, в том числе и за пределами своего проекта.

В этом случае в ваш проект будет вставлена только ссылка на сгенерированный

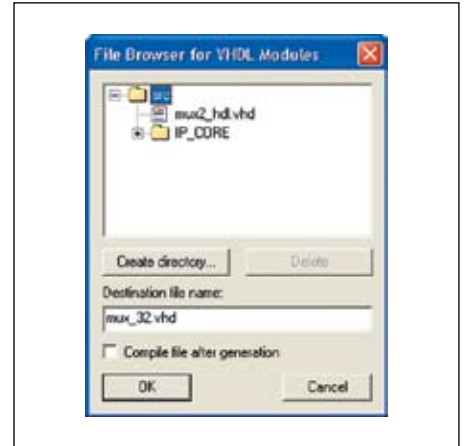


Рис. 31. Выбираем месторасположение для генерируемых IP Core генератором файлов

файл `mux2.vhd`. Щелкните на ссылке правой кнопкой мыши, выполните команду **Properties** и вы увидите путь к этому файлу.

Во втором разделе **Additional Files Options** диалоговой панели **IP Core Generator Configuration** (рис. 30) вы можете дополнительно разрешить генерацию графического изображения модуля (**Generate block diagram symbol**) и создание макрофайла для того, чтобы поместить его в рабочую библиотеку (**Generate macro**).

Установив флажок **Run macro after generation**, вы разрешите программе завершить процесс генерации модуля компиляцией и вставкой его графического изображения в рабочую библиотеку.

В третьем разделе **View Options** рекомендуется установить флажок **Show generated HDL file after generation**, и тогда автоматически будет открываться для показа созданный **IP Core** генератором файл. Надо сказать, что это довольно удобный режим завершения работы генератора и, по мнению автора, не стоит им пренебрегать.

Продолжение следует



Рис. 30. Диалоговая панель для настройки параметров IP Core генератора