

Окончание. Начало в № 3`2009

Александр ШАЛАГИНОВ  
shalag@vt.cs.nstu.ru

# Изучаем Active-HDL 7.1.

## Урок 15. Менеджер библиотек

Для организации и управления «библиотечным хозяйством» в среде **Active-HDL** имеется специальная программа, называемая менеджером библиотек **Library Manager**. Она упрощает работу с библиотеками, освобождая пользователя от необходимости углубляться в файловую организацию библиотек на физическом уровне.

Менеджер библиотек позволяет создавать библиотеки, изменять содержимое существующих библиотек, задавать им статус глобальной или локальной библиотеки, изменять режим доступа, присоединять и отключать их от активного проекта, выполнять поиск и редактирование компонентов в библиотеках.

Запустить программу можно несколькими способами. В нашем распоряжении имеется команда главного меню **View/Library Manager**, иконка  на стандартной панели инструментов (рис. 1) и горячая клавиша **Alt+7**.

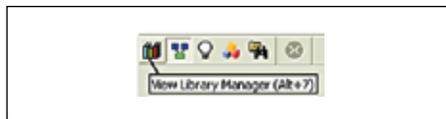


Рис. 1. Запуск менеджера библиотек Library Manager

Окно менеджера библиотек (рис. 2) состоит из двух панелей. На левой панели приводится список библиотек, подключенных в настоящий момент к программе, и их основные параметры: статус (**G** — глобальная библиотека или **L** — локальная), логическое имя **Library**, поставщик **Vendor**, режим доступа **Mode (R/O** — только чтение или **R/W** — полный доступ), комментарий **Comment** и местонахождение **Directory**.

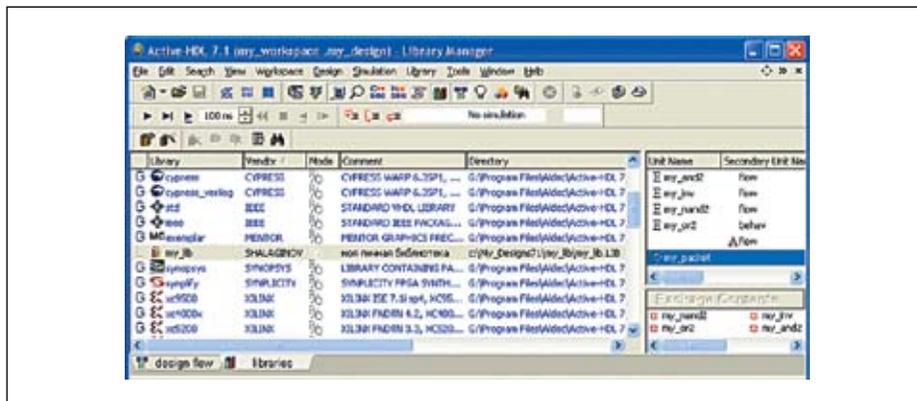


Рис. 2. Окно менеджера библиотек Library Manager

Правая панель отображает содержимое выбранной библиотеки — список и параметры входящих в нее модулей. Обычно это откомпилированные внутренние описания компонентов: исходный **HDL**-код, схемы или диаграммы состояний цифровых автоматов, а также список цепей в формате **EDIF**.

Кроме того, в библиотеках хранятся внешние описания компонентов — символы. В схематехнике их называют условными графическими обозначениями (УГО).

Если библиотека содержит описания иных объектов (функций, процедур, объявления типов, констант, сигналов), то они объединяются в специальный модуль, называемый пакетом **Package**.

Библиотека может содержать несколько пакетов, и их имена тоже отображаются на правой панели менеджера библиотек. Пакет легко отличить от компонента, так как слева от его имени помещается буква **P**, например **Prog\_pkt**; на рис. 2.

Чтобы увидеть содержимое пакета, следует щелкнуть на нем ЛКМ. В нижней части панели откроется дополнительное окно со списком входящих в пакет объявлений (окно **Package Contents** на рис. 2).

Заметьте, на левой панели приводятся логические имена библиотек, которые по умолчанию совпадают с физическими именами, но об этом мы поговорим позднее.

Если в среду проектирования **Active-HDL** не загружен ни один проект, то в окне **Library Manager** видны только системные библиотеки (**system libraries**). Они автоматически подключаются к менеджеру библиотек в процессе установки системы. Это — глобальные библиотеки, о чем говорит символ **G** (от слова **Global**) в левом столбце таблицы (рис. 2).

Системные библиотеки предоставляют пользователю ограниченный доступ к своим ресурсам. Запись **R/O** в четвертом столбце таблицы говорит о том, что библиотеки открыты только для чтения данных.

С системными библиотеками лучше не экспериментировать или делать это весьма осторожно, чтобы не повредить среду проектирования **Active-HDL** и не нарушить ее целостности. Недаром они закрыты для редактирования.

Список всех глобальных библиотек проекта хранится в текстовом файле конфигурации с названием **library.cfg**. Он находится в папке **Vlib**, где, кстати, располагаются и все системные библиотеки.

Пока в систему не загружен ни один проект, вы не можете создать новую библиотеку. Соответствующая команда **Create New Library** из меню **Library** недоступна для исполнения.

Но стоит нам создать или открыть какую-нибудь проект, и в окне менеджера библиотек появится новая локальная библиотека. Это рабочая библиотека проекта. По умолчанию ее имя будет таким же, как имя активного проекта.

Для активного проекта можно создать не одну, а несколько новых библиотек. Все они локальные и будут принадлежать тому проекту, в котором созданы. Отсюда происходит их название — библиотеки проекта (**design libraries**). Еще раз подчеркнем: только одна из них может быть рабочей.

В рабочую библиотеку по умолчанию помещаются все откомпилированные модули текущего проекта. Список локальных библиотек проекта находится в соответствующем файле, который называется **projlib.cfg**. Для каждого нового проекта автоматически создается свой файл конфигурации **projlib.cfg** и помещается в рабочую папку проекта.

Закончив небольшую теоретическую разминку, перейдем к практической работе. Попробуем создать свою личную библиотеку **my\_lib** и наполнить ее различными данными.

Один способ нам уже известен. Нужно создать проект с именем **my\_lib**, и тогда в нем автоматически появится рабочая библиотека с тем же названием. Но мы поступим иначе.

Создадим новое рабочее пространство **my\_workspace**, а в нем — новый проект **my\_design**. Как только такая работа будет закончена, в окне менеджера библиотек мы увидим новую рабочую библиотеку проекта **my\_design**. Она генерируется автоматически самой системой. Но нам-то нужна другая



Рис. 3. Из окна Library Manager вызываем «мастер» создания новой библиотеки

библиотека, которую мы собираемся создать самостоятельно и назвать **my\_lib**.

Выполним команду **Create Library...** из меню **Library** или **Design**. Можно щелкнуть и по иконке (**Create New Library**), расположенной на инструментальной панели **Library Manager**. Во всех случаях активизируется работа «мастера» создания новой библиотеки **New Library Wizard**. Введем имя создаваемой библиотеки **my\_lib**, укажем место ее расположения, например, **C:\My\_Designs71**, и установим флажок **Create empty library** (рис. 3).

Созданную библиотеку вы обнаружите в окне **Library Manager** (рис. 4). Она действительно пустая, так как на правой панели отсутствуют какие-либо библиотечные данные.

Но почему созданная библиотека **my\_lib** не видна в окне просмотра проекта **Design Browser**? Ответ прост: любой проект может иметь только одну рабочую библиотеку, и таковой является библиотека **my\_design**.

Если быть абсолютно точным, то создаваемая библиотека **my\_lib** в течение короткого времени присутствовала в окне просмотра проекта. Более того, она даже становилась активной на время ее автоматической генерации. Однако по окончании процесса система удалила ее из рабочего пространства.

Вы не успели ничего такого заметить? Тогда повторите эксперимент, предварительно удалив только что созданную библиотеку с жесткого диска. Сделать это легко: выделите библиотеку **my\_lib** в окне менеджера библиотек и нажмите на клавиатуре клавишу **Delete**.

Будьте внимательны. Повторно запустив процесс автоматической генерации новой библиотеки **my\_lib**, вы должны увидеть в окне просмотра проекта следующий текст (рис. 5). Через некоторое время он исчезнет.

Подводя итог сказанному, заметим, что система выполнила в автоматическом режиме всю ту работу, которую мы посчитали скучной и не желали делать.

А теперь создадим еще одну библиотеку **my\_lib2**, но в отличие от предыдущего примера не станем устанавливать флажок **Create empty library**. Другими словами, при созда-

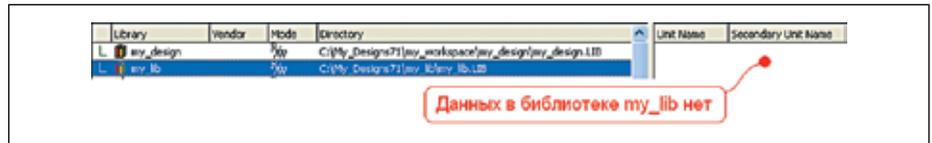


Рис. 4. В окне менеджера библиотек появилась пустая библиотека проекта **my\_lib**



Рис. 5. На время создания проектной библиотеки она загружается в рабочее пространство как обычный проект. По окончании процесса проект удаляется из рабочего пространства

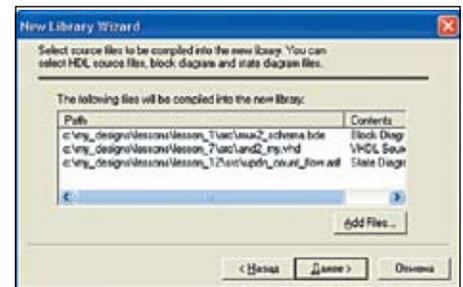


Рис. 6. «Мастер» предлагает задать список исходных файлов, которые должны быть откомпилированы в новую библиотеку **my\_lib2**

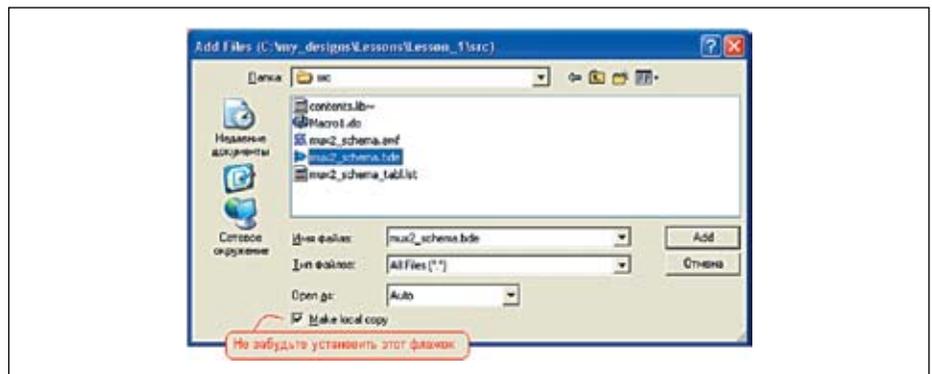


Рис. 7. Добавляем в библиотеку схемный файл **mux2\_schema.bde**

нии новой библиотеки мы хотим сразу же загрузить в нее некоторые данные.

«Мастер» создания новой библиотеки открывает панель (рис. 6), которую мы не видели в первом случае, создавая пустую библиотеку.

Нам предлагается указать исходные файлы, которые система будет компилировать в новую библиотеку. Об этом говорит следующий текст сверху панели: **Select source files to be compiled into the new library**.

Здесь же дается подсказка, какие типы файлов можно помещать в библиотеку: **You can select HDL source files, block diagram and state diagram files**. Выясняется, что файлы могут содержать исходный HDL-код, схемы или диаграммы состояний.

Их месторасположение не ограничивается рамками текущего проекта. Для полноты картины добавим в создаваемую библиотеку все возможные варианты исходных описаний компонентов. В свое время они были созданы как результат освоения предыдущих уроков.

Нажмем на кнопку **Add Files...** и с помощью следующей панели мастера **Add Files** (рис. 7) найдем файл со схемой мультиплектора **mux2\_schema.bde**, созданный на уро-

ке 1. Кнопкой **Add** добавим его в формируемый список.

Выполняя данную операцию, не забудьте установить флажок **Make local copy**. В противном случае в библиотеку добавится не локальная копия файла, а только ссылка на него, и мы рискуем потерять автономность создаваемой библиотеки.

Аналогичным образом скопируем в библиотеку **VHDL**-модель логического элемента **and2\_my** (урок 7, рис. 5) и диаграмму состояний реверсивного счетчика **UpDn\_count\_flow** (урок 12, рис. 19).

После того как список компилируемых в библиотеку **my\_lib2** объектов будет сформирован, следует нажать на кнопку «Далее» (рис. 6) и разрешить «мастеру» выполнить всю запланированную им программу (рис. 8). Она состоит из четырех шагов (**steps**).

Что же собирается «мастер» сделать? Для начала — создать временный проект, содержащий заданные исходные файлы (step 1). Затем откомпилировать их в рабочую библиотеку проекта по умолчанию (step 2). После чего удалить из рабочего пространства временный проект с сохранением его рабочей



Рис. 8. «Мастер» готов создать библиотеку, выполнив программу из четырех пунктов

библиотеки (step 3) и вернуться к ранее открытому проекту (step 4).

Заметим, что, создавая пустую библиотеку **my\_lib**, «мастер» действовал в соответствии с той же самой программой, только компилировать ему было нечего.

По окончании работы в окне менеджера библиотек появится вновь созданная библиотека **my\_lib2** со всем ее содержимым (рис. 9).

Обратите внимание на столбец **Source Type**. В нем указывается тип исходных данных: исходный код (**Source Code**), схема (**Block diagram**) и диаграмма состояний (**State diagram**).

Любое из этих описаний доступно для просмотра и даже для редактирования. Достаточно щелкнуть ПКМ на нужной строке и выполнить в контекстном меню команду **View Source: Text** или **View Source: Diagram** (рис. 10). Прделайте эту простую работу, чтобы удостовериться в сказанном.

Обратите внимание, локальные библиотеки активного проекта менеджер библиотек выделяет черным шрифтом, в то время как глобальные библиотеки отмечаются синим

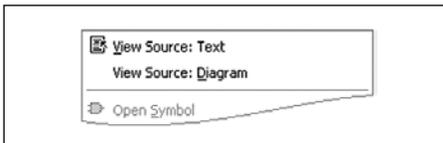


Рис. 10. Для просмотра и редактирования исходных описаний удобно использовать команды View Source: Text или View Source: Diagram

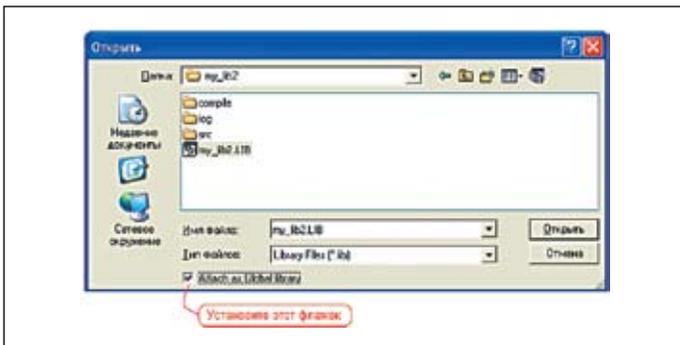


Рис. 11. Установив флажок Attach as Global library, вы присоедините библиотеку к активному проекту со статусом глобальной

Library	Unit Name	Secondary Unit Name	Source Type	Target Language	Symbol	Simulation Data
my_lib2	E and2_my	and2_my	Source Code	VHDL	No	Yes
my_lib	E mux2_schema	mux2_schema	Block diagram	VHDL	No	Yes
my_design	E updn_count_flow	updn_count_flow...	State diagram	VHDL	No	Yes

Рис. 9. В окне менеджера библиотек появилась созданная библиотека my\_lib2, содержащая три модуля: and2\_my, mux2\_schema и updn\_count\_flow

цветом. Кстати, рабочие библиотеки неактивных проектов, принадлежащие тому же самому рабочему пространству (в примере это **my\_workspace**), временно получают статус глобальных библиотек.

Чтобы убедиться, что созданные библиотеки **my\_lib** и **my\_lib2** принадлежат тому проекту, в котором они были созданы (проекту **my\_design**), проведем еще один простой эксперимент.

В том же рабочем пространстве **my\_workspace** создадим новый проект, например **my\_design2**. Он автоматически получит статус активного проекта. Убедимся, что в окне **Library Manager** библиотеки **my\_lib** и **my\_lib2** не видны.

А вот рабочая библиотека **my\_design** оказывается доступной и из нового проекта. Главное условие видимости — проекты должны находиться в одном рабочем пространстве.

Обратите внимание, система автоматически сменила статус библиотеки **my\_design** — она стала глобальной. Это наталкивает на мысль, что, сделав библиотеку **my\_lib** или **my\_lib2** глобальной, мы сможем работать с ней из любого проекта.

Если вы хотите, чтобы какая-либо локальная библиотека была доступна активному проекту, достаточно присоединить ее, выполнив команду **Attach Library** из меню **Library** или щелкнув по соответствующей иконке , расположенной на инструментальной панели **Library Manager**.

Если вы пожелаете, чтобы какая-либо локальная библиотека была доступна для любого проекта по умолчанию, надо установить для нее статус глобальной. Один из вариантов — поменять ее статус в процессе присоединения к активному проекту (рис. 11).

Другой вариант предлагает сменить статус библиотеки после того, как она будет присоединена к проекту. Достаточно щелкнуть

на строке с присоединенной библиотекой ПКМ и выполнить в контекстном меню команду **Make global library**.

Здесь возникает законный вопрос: а можно ли глобальную библиотеку опять сделать локальной? В контекстном меню подходящей команды вы не обнаружите. Отсутствует она и в главном меню.

И, тем не менее, проблема решается достаточно просто. Нужно просто отсоединить глобальную библиотеку от менеджера проекта, выполнив команду **Detach** из меню **Library** или из контекстного меню. Другая возможность, дающая тот же результат, — щелкнуть по иконке  и подтвердить выполняемую операцию.

В теоретической части урока говорилось о файлах **library.cfg** и **projlib.cfg**, в которых находятся списки библиотек, загружаемых в менеджер библиотек. Познакомимся с их содержимым.

Найдем на диске папку **my\_workspace**, в которую вложены папки с проектами **my\_design** и **my\_design2**. Раскроем папку **my\_design** и познакомимся с содержимым файла **projlib.cfg** (рис. 12).

Здесь вроде бы все понятно. Рабочая библиотека **my\_design** была автоматически сгенерирована системой в процессе создания проекта **my\_design** и получила то же самое имя. Две другие библиотеки **my\_lib** и **my\_lib2** — результат наших экспериментов.

Откроем файл **library.cfg**, расположенный в папке **my\_workspace** (рис. 13). И здесь нет загадок. Мы видим список рабочих библиотек всех проектов, подключенных к данному рабочему пространству.

Кроме того, в первой строке файла указывается на необходимость присоединить к указанному списку все глобальные библиотеки, перечисленные в файле с таким же названием из папки **Vlib** (к нему задается полный путь).

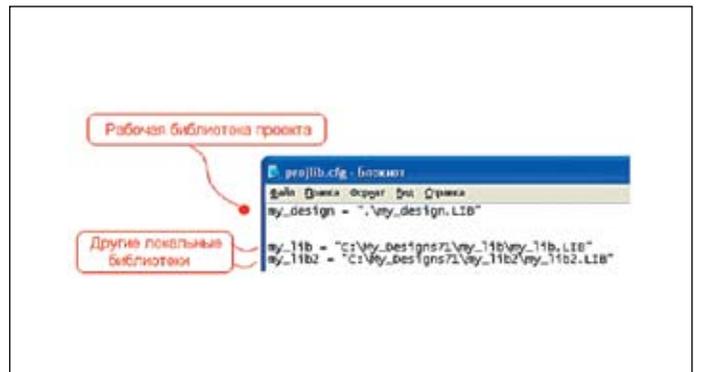


Рис. 12. Список локальных библиотек проекта my\_design в файле projlib.cfg

```

$include = "D:\Program Files\Aldec\Active-HDL 7.1\vl1b\
my_design = ".\my_design\my_design.lib"
my_design2 = ".\my_design2\my_design2.lib"

```

Рис. 13. Список рабочих библиотек всех проектов, подключенных к рабочему пространству my\_workspace

Найдем этот файл по указанному пути и посмотрим его «начинку». Изначально (при установке пакета) он включает все глобальные библиотеки среды **Active-HDL 7.1**, а это примерно 200 наименований. По понятным причинам на рис. 14 приведен только его небольшой фрагмент.

Вы можете сделать копию данного файла и удалить из него все библиотеки, которые не собираетесь использовать. Тем самым будут созданы более комфортные условия для работы с менеджером библиотек. А резервная копия позволит вам при необходимости восстановить первоначальную конфигурацию глобальных библиотек.

Интересно посмотреть, что произойдет с содержимым рассмотренных файлов, если библиотеку **my\_lib2** (или любую другую) сделать глобальной. Прodelайте эту операцию и посмотрите на результат.

Вы увидите, что ссылка на библиотеку **my\_lib2** исчезла из файла **projlib.cfg** и появилась в файле **library.cfg**, который находится в папке **Vlib**. Содержимое файла **library.cfg**, расположенного в папке **my\_workspace**, останется без изменений.

Если библиотеку **my\_lib2** снова сделать локальной, то обнаружится, что она потеряла родовую связь с тем проектом, в котором была создана.

Впрочем, не всегда дела обстоят описанным образом. Если локальную библиотеку присоединить к «чужому» проекту и затем сделать ее глобальной, то связь с «родителем» сохранится.

Прodelайте этот эксперимент, и вы убедитесь, что ссылка на библиотеку **my\_lib2** имеется теперь в двух файлах: в файле со списком глобальных библиотек **library.cfg** и в файле **projlib.cfg** проекта **my\_design**.

Заметим, что **Library Manager** первым читает файл со списком локальных библиотек, а потому уже найденное логическое имя **my\_lib2** в списке глобальных библиотек будет просто игнорироваться.

Вы не задавались вопросом, что будет делать менеджер библиотек, если предложить

```

xc9500 = "D:\Program Files\Aldec\Active-HDL 7.1\vl1b\XC9500\XC9500.LIB"
xc5200 = "D:\Program Files\Aldec\Active-HDL 7.1\vl1b\XC5200\XC5200.LIB"
xc4000x = "D:\Program Files\Aldec\Active-HDL 7.1\vl1b\XC4000X\XC4000X.LIB"
xc4000e = "D:\Program Files\Aldec\Active-HDL 7.1\vl1b\XC4000E\XC4000E.LIB"
xc3000 = "D:\Program Files\Aldec\Active-HDL 7.1\vl1b\XC3000\XC3000.LIB"

```

Рис. 14. Содержимое файла конфигурации library.cfg, находящегося в папке Vlib (фрагмент). Файл содержит список глобальных библиотек системы, загружаемый в окно менеджера библиотек Library Manager

ему присоединить библиотеку, которая уже имеется в его списке? Станет ли он выполнять эту «бессмысленную» работу? Впрочем, не станем гадать, а проверим все на практике.

Библиотека **my\_lib2** уже присоединена к активному проекту **my\_design**. Попробуем присоединить ее еще раз (команда **Attach Library**) и посмотрим на результат (рис. 15).

Мы видим, что теперь две локальные библиотеки с разными логическими именами — **my\_lib2** и **my\_lib21** — ссылаются на одну и ту же физическую библиотеку **my\_lib2**. Менеджер библиотек автоматически изменил имя повторно подключаемой библиотеки.

Последнее означает, что и мы можем редактировать логические имена библиотек по своему усмотрению, правда, только в том случае, когда к библиотеке имеется полный доступ. Соответствующие команды **Rename**, **Read/Write** или **Read Only** активизируются из контекстного меню.

### Физическая организация библиотек

Мы уже знаем, что системные библиотеки пакета **Active-HDL** размещаются в папке **Vlib**. Отыщем названную папку на диске, например по стандартному пути **C:\Program Files\Aldec\Active-HDL 7.1**. Откроем содержимое нескольких библиотек и проанализируем их структуру (рис. 16).

Прежде всего, обратим внимание на файл с расширением **\*.adf** — это текстовый файл описания проекта. Значит, библиотека организована «по образу и подобию» проекта, а следовательно, ее можно загрузить в среду **Active-HDL** как обычный проект. Впрочем, это для нас уже не новость.

Во всех библиотеках мы видим папку **src**. Название происходит от слова **source** — «источник», «исходный». В ней хранятся исходные файлы библиотеки или проекта и другие ресурсные файлы. Они обычно отображаются

в окне просмотра проекта **Design Browser**. Во всяком случае их туда можно добавить.

Еще одна папка с именем **compile** хранит промежуточные файлы, автоматически генерируемые системой из схемных описаний или диаграмм состояний. Они могут быть представлены на любом из возможных языков описания аппаратуры **VHDL**, **Verilog** или в формате **EDIF**. В некоторых библиотеках, например, в библиотеках **ieee** или **simprim**, папка **compile** отсутствует (рис. 16).

В принципе из библиотеки можно удалить папки **src** и **compile** без катастрофических последствий. Но способность моделировать свои проекты вы не потеряете, так как откомпилированные модули из библиотеки не пропали. Однако обновить библиотеку вы уже не сможете. Кроме того, станет недоступным и просмотр исходных описаний компонентов (HDL-кодов, схем и диаграмм состояний).

Откомпилированные данные о компонентах (их рабочие модели), а также графические изображения (УГО), находятся в файлах с фиксированными именами:

```

0<library_name>.mgf
1<library_name>.mgf
2<library_name>.mgf
3<library_name>.mgf

```

Названные имена генерируются автоматически, а их число динамически отслеживается системой в зависимости от типа размещаемых в библиотеке данных. Например, для графических изображений компонентов зарезервирован файл **2<library\_name>.mgf**. Если вы удалите из библиотеки все символы, исчезнет за ненадобностью и этот файл.

Некоторые библиотеки, в частности, библиотека **ieee**, вообще не содержат подобной информации, а значит, в структуре такой библиотеки не будет и файла с именем **2ieee.mgf**.

Еще один файл заслуживает пристального внимания — это так называемый индексный файл библиотеки, имеющий расширение

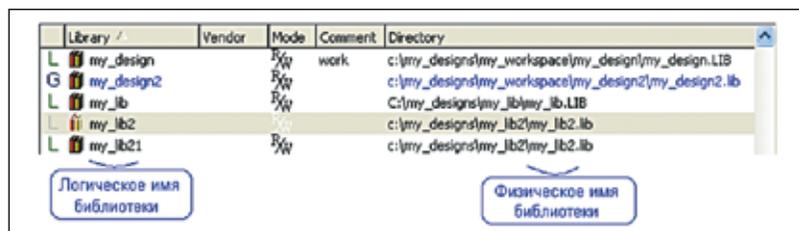


Рис. 15. Менеджер библиотек присоединил к активному проекту ту же самую физическую библиотеку my\_lib2 под новым логическим именем my\_lib21

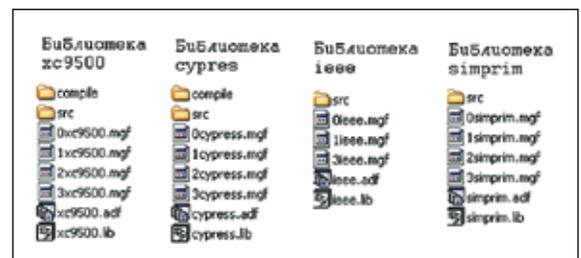


Рис. 16. Типовое содержимое библиотеки



Рис. 17. Файловая структура вновь созданной пустой библиотеки

\***lib**. В нем хранится список элементов (библиотечных модулей), включенных в библиотеку. Кроме того, здесь задаются все ссылки на исходные файлы ресурсов библиотеки или проекта.

Кратко обсудив теоретические вопросы организации библиотек, перейдем к практической работе. Попробуем сделать свою библиотеку, повторяющую организацию и возможности системных библиотек САПР **Active-HDL**.

В рабочем пространстве **my\_workspace** проекта **my\_design** у нас уже имеется пустая библиотека **my\_lib**. Найдем ее расположение на диске и посмотрим на файловую структуру (рис. 17).

Откройте папку **src** и убедитесь, что в ней нет никаких исходных данных. Поэтому первое, что предстоит сделать, — это наполнить базу данных библиотеки **my\_lib**.

Загрузим созданную библиотеку **my\_lib** в рабочее пространство **my\_workspace** (откроем файл **my\_lib.adf**) и будем работать с ней как с проектом. Найдем на диске нужные исходные файлы, например, **and2\_my.hdl**, и добавим их к формируемой библиотеке. Похожим образом мы наполняли данными библиотеку **my\_lib2** (рис. 6).

Еще раз напомним, что при добавлении файлов надо следить за тем, чтобы был установлен флажок **Make local copy**. Иначе в библиотеку будут вставлены только ссылки, что лишит ее автономности.

Отсутствующие описания, например, VHDL-модели компонентов **or2\_my** и **inv\_my**, можно создать непосредственно в рабочем проекте **my\_lib**.

Выполним компиляцию всех исходных файлов (команда **Compile All**) и убедимся, что в библиотеке **my\_lib** для всех имеющихся компонентов появились данные для моделирования. Столбец **Simulation Data** в правом окне менеджера библиотек (рис. 18) должен содержать во всех строчках слово **Yes**.

Проверим, что созданной библиотекой уже можно пользоваться. Активизируем проект **my\_design**, соберем из библиотечных элементов какую-нибудь простую схему, например, **mux2\_schema**, и промоделируем ее. Все должно получиться.

Некоторые претензии можно высказать лишь в отношении графики использованных элементов. Она имеет стандартный вид и может не удовлетворить вкус пользователя. Но это легко исправить, заглянув в урок 10 «Как проектировать символы».



Рис. 18. Для всех компонентов библиотеки **my\_lib** имеются данные для моделирования

Убедитесь, что вам доступно содержимое исходных файлов всех компонентов, включенных в библиотеку. Щелкните пару раз на УГО любого символа, и вы должны увидеть его исходный код. Проверьте, что эта же операция «работает» и из окна менеджера библиотек.

А теперь проделайте простой эксперимент: переименуйте папку **src** своей библиотеки (или на время переместите ее в другое место). Исходные файлы станут недоступны.

Следующим шагом наведем порядок в своем библиотечном хозяйстве. Для начала посмотрим, как это сделано у разработчиков **Active-HDL** и проделаем то же самое у себя.

Откроем папку **src** библиотеки **xc9500** и выясним ее содержимое. Мы увидим много схемных файлов и всего два VHDL-файла: **vcomponents.vhd** и **XC9500.VHD**. Поинтересуемся их содержимым. В файле **XC9500.VHD** хранится исходный код всех моделей компонентов, включенных в библиотеку. Его основная особенность состоит в том, что он объединяет в себе модели всех компонентов библиотеки. А у нас модель каждого компонента хранится в отдельном файле.

Устранить это различие не представляет особого труда. Создадим новый файл (команда **File/New/VHDL Source**), назовем его **my\_gates.vhd** и перенесем туда модели компонентов из отдельных файлов. Последние больше не нужны, и их лучше удалить.

Второй файл системной библиотеки **xc9500** называется **vcomponents.vhd**. В нем находятся два пакета — **VPKG** и **VCOMPONENTS**, но только один из них, а именно **VCOMPONENTS**, интересует нас в данный момент. В нем собраны объявления всех компонентов, включенных в библиотеку.

Здесь придется сделать небольшое пояснение. Язык VHDL унаследовал от языка программирования ADA механизм пакетов. Пакеты фактически являются средством расширения языка описания аппаратуры. Вы можете использовать стандартные пакеты, собранные в библиотеке **ieee** и содержащие описания типов, констант, процедур и функций, расширяющих возможности языка VHDL.

Например, использование пакета **std\_logic\_1164** увеличивает число допустимых значений цифрового сигнала с двух до девяти, а значит, расширяет и возможности моделирования. Благодаря пакетам эффективно решается проблема повторно (**reuse**) и совместно используемых данных при коллективных разработках.

Чтобы информация, содержащаяся в пакете, стала доступной для использования, надо

сделать ссылку на соответствующий пакет и библиотеку, в которой он находится, например вот так:

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

Если библиотека небольшая, то модели компонентов и их определения удобно помещать в одном файле. Так, например, поступили разработчики библиотеки **cypress** (рис. 19). Мы последуем их примеру.

Откроем файл **my\_gates.vhd** и в его начало вставим шаблон пакета с именем **my\_packet**, снабдив его ссылкой на библиотеку **ieee** и стандартный пакет **std\_logic\_1164**. На помощь можно привлечь языковой помощник (урок 6).

Добавим в созданный пакет декларации компонентов. С этой целью выделим в окне менеджера библиотек первый компонент из библиотеки **my\_lib** и щелкнем на нем ПКМ. В контекстном меню выполним команду **Copy Declaration**. Определение копируемого компонента окажется в буфере обмена. Остается только перенести его в только что созданный пакет **my\_packet**, удалив последнюю строку с конфигурацией.

Аналогичным образом следует перенести в пакет объявления остальных компонентов библиотеки. Откомпилируем файл **my\_gates.vhd** и убедимся, что в окне менеджера библиотек

```
library ieee ;
use ieee.std_logic_1164.all ;

PACKAGE cygates is

component cy_and2
  port(a,b : in std_logic;
        y : out std_logic);
end component;
-- объявления других компонентов
end cygates ;
-----

library ieee ;
use ieee.std_logic_1164.all ;
entity cy_and2 is
  port (a,b : in std_logic;
        y : out std_logic);
end cy_and2;
architecture archRTL of cy_and2 is
begin
  y <= a AND b ;
end archRTL;
-- VHDL код других компонентов
```

Рис. 19. Содержимое файла **cy\_gates.vhd** системной библиотеки **cypress** (фрагмент)

Library	Unit Name	Secondary Unit Name	Source Type	Target Language	Symbol	Simulation Data
my_lib	and2_my	and2_my	Source Code	VHDL	Yes	Yes
mach10	inv_my	inv_my	Source Code	VHDL	Yes	Yes
cydonell	or2_my	or2_my	Source Code	VHDL	Yes	Yes
at10k	my_packet		Source Code	VHDL		Yes
at_vtl						
ad2						
ad00k						
ad0mx						
ad3200dx						
ad70w						

Рис. 20. В библиотеку my\_lib добавлен пакет my\_packet с декларациями компонентов

лиотек появился созданный нами пакет, а внизу справа показано его содержимое — объявления компонентов (рис. 20).

Если открыть папку **src** библиотеки **cypress**, то вы наверняка обратите внимание на необычный файл `update_cypress.do`. Судя по названию, этот файл предназначен для обновления (**update**) библиотеки, внесения в нее изменений в пакетном режиме.

Скопируем его в свою библиотеку под именем **update\_my\_lib.do** и приспособим для своих целей (рис. 21).

Первая команда **setlibrarymode** устанавливает режим «чтение-запись», разрешая модифицировать библиотеку. Следующая

команда **clearlibrary** очищает ее от старых данных для моделирования. Третья команда **acom** выполняет компиляцию обновленного файла **my\_gates.vhd**, а четвертая — восстанавливает режим «только чтение».

В окне просмотра проекта **Design Browser** щелкнем на вновь созданном командном файле **update\_my\_lib.do** и выполним команду `|@ Execute`. Установим библиотеке **my\_lib** статус глобальной. Для этого выделим ее в окне менеджера библиотек, вызовем контекстное меню и выполним команду **Make global library**.

Откроем индексный файл **my\_lib.lib** и добавим в раздел `[~A]` две строки:

```
[~A]
Comment=моя личная библиотека
Vendor=SHALAGINOV
```

Ну вот, кажется все. Осталось убедиться, что наша библиотека обладает теперь всеми свойствами системной библиотеки.

Вновь активизируем проект **my\_design** и нарисуем в нем две одинаковые схемы, в одной используем элементы из системной

библиотеки **cypress**, в другой — элементы из нашей библиотеки **my\_lib** (рис. 22).

Конвертируем графические описания обеих схем, чтобы получить VHDL-код (команда **Generate Code** всплывающего меню или иконка .

Сравнив результаты, приходится констатировать, что еще не все сделано: наша схема не видит пакета **my\_packet** из библиотеки **my\_lib**. Это проявляется в том, что объявления используемых в схеме компонентов вставлены в текст модели, хотя они есть в пакете **my\_packet**.

Чтобы все получилось, нужно сделать видимым данный пакет. С этой целью выполним команду **Code Generation Setting...** из главного меню **Diagram** и активизируем закладку **Packages List** (рис. 23). В верхнее окно **Libraries** добавим библиотеку **my\_lib**, а в среднее окно **Package** — имя имеющегося в ней пакета **my\_packet**.

Повторно конвертируйте схему, построенную на элементах нашей библиотеки, и убедитесь, что теперь в полученном VHDL-коде присутствует оператор **use** со ссылкой на пакет **my\_packet**:

```
library my_lib;
use my_lib.my_packet.all;
```

Но самое большое достижение — в сгенерированном тексте больше нет деклараций компонентов.

```
1 # File: update_my_lib.do
2 # Copyright Shalag.
3
4 setlibrarymode -rw my_lib
5 clearlibrary
6 acom -work my_lib my_gates.vhd
7 setlibrarymode -ro my_lib
```

Рис. 21. Создаем командный файл update\_my\_lib.do для обновления библиотеки my\_lib

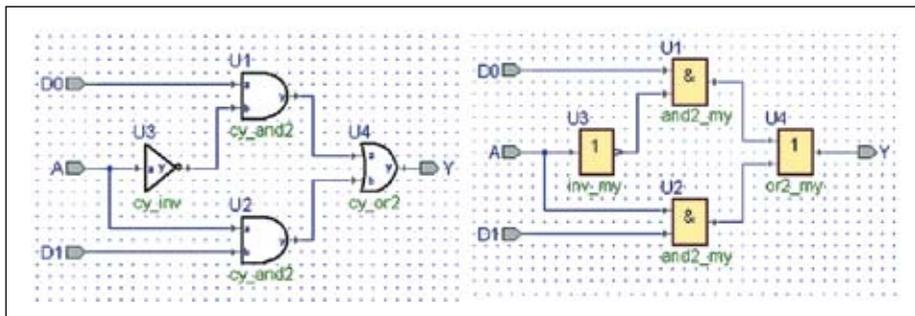


Рис. 22. Проверяем на простой схеме качество нашей библиотеки my\_lib

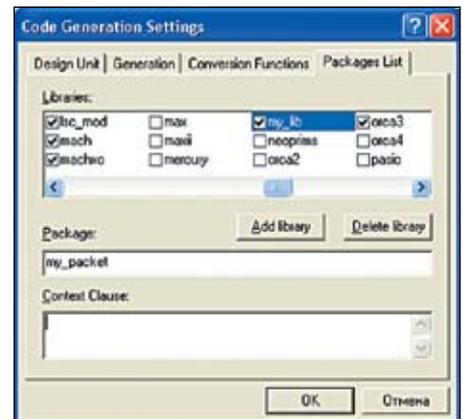


Рис. 23. В настройки программы Code Generation добавляем нашу библиотеку my\_lib и пакет my\_packet