Знакомство с пакетом OrCad 9.1

Урок 5. Как создать свою личную библиотеку

Александр Шалагинов, к. т. н.

shalag@vt.cs.nstu.ru

В пакете OrCAD используется библиотечный метод проектирования. Он заключается в том, что проектируемое устройство «собирается» из отдельных *готовых* деталей, называемых компонентами (Component). Ими могут быть резисторы, конденсаторы, транзисторы, реле, разъемы, микросхемы и прочие радиоэлектронные изделия. Все компоненты объединяются в библиотеки, которые входят в состав пакета. Они называются *системными*.

Пользователь может создавать новые описания компонентов и включать их в уже существующие библиотеки. При желании можно организовать и свои личные библиотеки. Они называются *пользовательскими*.

Общее число библиотечных компонентов достигает в современных САПР нескольких десятков тысяч. Так, например, пакет OrCAD 9.1 содержит более 30 тысяч компонентов.

В библиотечном «хозяйстве» любого современного пакета автоматизированного проектирования не так-то просто разобраться, а для OrCAD эта задача осложняется еще и тем, что в нем фактически объединены библиотеки двух пакетов — собственно OrCAD и DesignLab 8 фирмы MicroSim, которая в 1998 году вошла в состав фирмы OrCAD.

Возможно, вы уже обнаружили, что в одном проекте нельзя объединять компоненты, поддерживаемые SPICE-моделями (проекты типа PCB — PSpice) и VHDL-моделями (PCB — Simulate). «Инородные» для данного типа проекта компоненты будут просто игнорироваться.

Другая сложность, характерная для всех современных САПР, заключается в том, что любой компонент имеет не одно, а несколько различных описаний и они хранятся в *разных* местах.

На принципиальной схеме компонент представляется в виде условного графического обозначения (УГО). Его принято называть *символом (Part, Symbol)*. Символы компонентов объединяются в библиотеки, имеющие расширение *.olb. Итак, запомним, в OLB-библиотеках хранятся *схемные образы* компонентов. В них отсутствует информация о форме и физических размерах (габаритах) реальных компонентов.

На печатной плате тот же компонент выглядит совсем иначе. Здесь главное передать его физические размеры и форму, чтобы выяснить, сколько места потребуется для его размещения. Такое описание, в отличие от схемного, называют *конструкторским или физическим*. Оно представляет собой габаритную проекцию корпуса компонента на печатную плату. Иногда его называют футпринтом, или «отпечатком» корпуса (от английского слова footprint — дословно: отпечаток ступни). Конструкторские описания хранятся в библиотеках с расширением *.llb и используются при трассировке печатных плат.

Для моделирования проекта требуется еще одно, третье по счету, описание компонента — описание его функции, поведения или алгоритма работы. Это описание обычно называют *функциональным или поведенческим*, иногда *математическим*. Оно задает правило, по которому вычисляется реакция компонента на входные воздействия. По этой причине третий вид описаний называют *моделями* (аналитическими или имитационными, поведенческими или структурными).

В пакете OrCAD для построения моделей используется стандартный язык описания аппаратуры VHDL, в пакете DesignLab 8 применяются так называемые SPICE-модели. Названные типы моделей не могут обрабатываться одной моделирующей программой, поэтому в OrCAD их две: Simulate для VHDL-моделей и программа PSpice A/D для SPICE-моделей (урок 8).

Понятно, что для каждого типа моделей должны быть свои библиотеки. VHDL-модели хранятся в библиотечных файлах с расширением *.vhd, а SPICE-модели — в библиотеках с расширением *.lib. В обоих случаях это текстовые файлы. Первые находятся в папке Orcad_9/Capture/Library, вторые — в папке Orcad_9/Capture/Library/Pspice.

Но и это еще не все. При проектировании принципиальных схем и разводке печатных плат требуется информация о том, каким образом и сколько символов компонента упаковывается (помещается) в его корпус. Например, в корпус компонента 555LA3 (аналог 74ls00) упаковано четыре вентиля 2И-НЕ, в корпус компонента 555TM2 (аналог 74ls74) — два D-триггера, а в корпус компонента 555LN1 (аналог 74ls04) — шесть инверторов.

Необходимо также указать, к каким выводам корпуса компонента подключены входы и выходы каждого символа, на какие «ножки» подается питание, какие контакты являются взаимозаменяемыми (логически эквивалентными).

Эти описания называются *упаковочной информацией* (Package) и в некоторых САПР хранятся в отдельных библиотеках. Например, в пакете DesignLab — в библиотеках с расширением *.plb. В САПР OrCAD упаковочная информация включена в библиотеки схемных описаний *.olb.

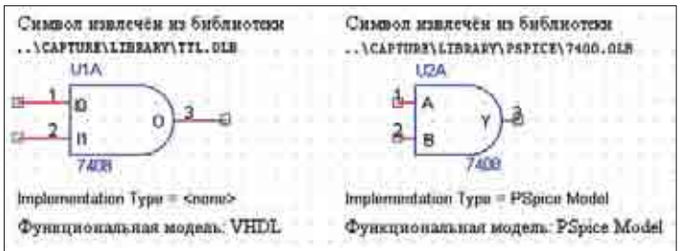


Рис. 1. Два типа описаний компонентов пакета OrCAD нельзя смешивать в одном проекте

а) б)

Рис. 2. Создание личной библиотеки пользователя

Для проектирования схем и их функциональной верификации необходимы только OLB- и VHD-библиотеки. Именно о них и пойдет дальнейший разговор.

Многие пользователи предпочитают работать не с системными библиотеками, а со своими личными, в которых нет ничего лишнего. Там обычно хранятся только те компоненты, которые встречаются в рабочем проекте. Кроме того, пользователь может отредактировать свойства компонентов под свои специфические требования, *не искажая* при этом системных библиотек.

Разработчики пакета OrCAD, предвидя такую потребность, добавили в файловую структуру каждого проекта специальный раздел, так называемый кэш проекта (Design Cache). Фактически это встроенная в проект библиотека, содержащая копии всех компонентов, используемых в проекте. Кроме того, кэш проекта выполняет еще одну роль. Он является своеобразным буфером, защищая системные библиотеки от неосторожного обращения с ними.

Кажется, что необходимость в создании личных библиотек отпала, ведь разработчики OrCAD обо всем позаботились. Но это не так, по крайней мере для российского пользователя. Дело в том, что фирменные библиотеки пакета OrCAD 9.1 содержат описания *только* зарубежных компонентов. Их имена и графические изображения, как правило, не соответствуют отечественным стандартам.

Поэтому российскому пользователю приходится создавать свои графические описания (или редактировать зарубежные), не ограничиваясь при этом рамками одного проекта. Ну а результаты такой работы удобнее всего сохранять в своей личной библиотеке.

Следует обратить внимание еще на одну особенность пакета OrCAD 9.1 (о ней мы уже вскользь упоминали): он содержит схемные библиотеки двух различных типов, компоненты из которых нельзя смешивать в одном проекте. Возможно, по этой причине даже однотипные компоненты имеют некоторые различия в графических описаниях. Прежде всего бросается в глаза разная длина выводов (рис. 1). Кроме того, одни и те же выводы имеют разные имена (сравните I0, I1, — слева, A, B, Y — справа), и уже по этой причине такие элементы не могут моделироваться одной функциональной моделью.

Обогащенные полученными сведениями, приступим к практической работе. Для начала создадим свою личную Olb-библиотеку (для схемных описаний) и назовем ее my_lib.olb.

Запустим графический редактор OrCAD Capture и активизируем команду **File/New /Library**. Откроется окно со стандартным названием библиотеки LIBRARY1 (рис. 2а). Нас не устраивает это имя, и мы сразу заменим его другим — my_lib.olb. С этой целью щелкнем правой кнопкой на строке с названием библиотеки, и когда откроется всплывающее меню, выполним команду **Save as…**. Введем нужное имя и сохраним библиотеку. Убедимся, что при повторном открытии личной библиотеки (команда **File/Open/Library…**) мы получили желаемый результат (рис. 2б).

Попробуем теперь скопировать в свою библиотеку my_lib.olb какие-нибудь компоненты из системной библиотеки, например из ttl.olb. Командой **File/Open/Library…** загрузим в редактор OrCAD Capture обе библиотеки и расположим их окна рядом. Выделим копируемый элемент, например 7404, и отбуксируем его в окно my_lib на строку с тем же названием. Операцию следует выполнять при нажатой клавише Ctrl, иначе вместо копирования будет выполнено перемещение элемента.

Мы уже знаем, что описанный способ копирования называется «Drag-and-drop». Более подробно о нем шел разговор на уроке 3 (рис. 6). Понятно, что тот же результат будет получен, если копирование компонента выполнять через буфер обмена Clipboard.

Результат проведенного эксперимента может обескуражить любого новичка. Мы копировали один элемент, а в итоге в библиотеке их оказалось более двух десятков. В чем же дело? За объяснениями далеко ходить не надо: вслед за копируемым элементом в новую библиотеку отправились и все *графически подобные* элементы (алиасы), а таких для компонента 7404 оказалось много — 27 штук.

В действительности в библиотеке ttl.olb хранится один графический образ компонента 74LS04 (так решили разработчики пакета OrCAD), а все остальные подобные ему компоненты только ссылаются на своего «родителя».

Иначе говоря, компонент 74LS04 имеет 26 псевдонимов (алиасов, дополнительных имен). Такое решение существенно экономит объем библиотечного Olb-файла, так как вместо 27 рисунков-копий в библиотеке хранится только один оригинал. Ну а с издержками такого решения вы уже ознакомились.

Добавлю, что у всех графически подобных элементов к тому же одинаковая упаковочная информация, так что выигрыш оказывается еще более значительным.

Чтобы убедиться, что все сказанное — чистейшая правда, выделите любой элемент, щелкните на нем правой кнопкой и выполните команду **Edit Part**. Для редактирования откроется окно с изображением компонента 74LS04, а не того, на котором вы щелкали мышью.

Проделайте эту операцию для нескольких элементов — результат останется прежним: во всех случаях вызываться будет компонент 74LS04. Выполните команду **View/Package**, и вы увидите данные об упаковке все того же «родителя» — компонента 74LS04. Если вы все еще сомневаетесь, наберите команду **Options/Package Properties…**.

Откроется диалоговая панель Edit Part Properties, на которой надо нажать кнопку Part Aliases… Появится новая панель с аналогичным названием (рис. 3), и вы увидите все псевдонимы компонента 74LS04. Выделите весь список (с помощью клавиши Shift) и удалите алиасы, нажав на клавишу Delete.

Вы увидите, что из библиотеки my_lib.olb исчезли все элементы, кроме 74LS04. Кстати, его можно переименовать (командой **Rename** из всплывающего меню) в элемент 7404, получив наконец-то результат, к которому мы так долго стремились.

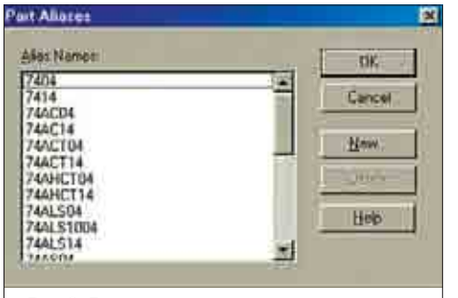
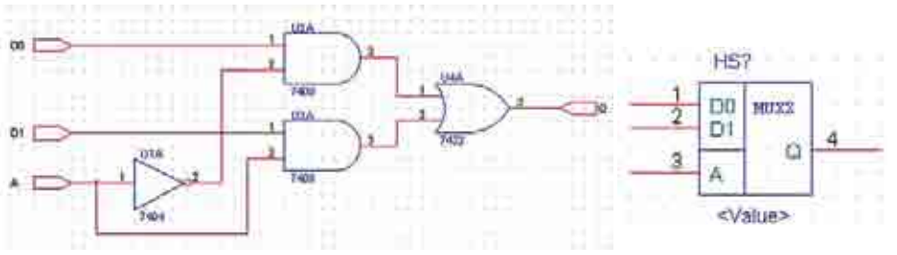


Рис. 3. Диалоговая панель для задания и удаления псевдонимов (алиасов) редактируемого компонента

Описанным выше приемом скопируем в личную библиотеку my_lib.olb компоненты 7408 и 7432. Избавимся от всех ненужных алиасов и сохраним результат. Чуть позже убедимся, что все элементы в созданной библиотеке вполне работоспособны.

В Olb-библиотеки пакета OrCAD, в отличие от многих других САПР такого же назначения, можно записывать не только символы (условные графические изображения), но и целые схемы. С этой особенностью мы уже знакомы по уроку 3 (рис. 6). Таким способом мы сохраняли внутренние описания (схемы замещения) иерархических блоков.

Этот же прием удобно использовать и при проектировании иерархических (не примитивных) символов. В отличие от иерархических блоков такой символ легко выполнить с



а) б)

Рис.4. Внутреннее (а) и внешнее (б) описания иерархического символа

соблюдением всех требований ГОСТ, что является его весьма важным достоинством.

Нарисуем схему замещения проектируемого иерархического символа, выбрав для простоты хорошо знакомый узел — мультиплексор на два входа mux2 (урок 1, рис. 1). Откроем новый проект с тем же названием, подключим к нему библиотеку my_lib.olb и создадим в нем схему мультиплексора mux2. Так как это будет «внутренняя начинка» иерархического символа (рис. 4а), подрисуем к ней входные и выходные порты (команда **Place/Hierarchical Port…**), чтобы образовать вертикальные связи с внешним описанием (рис. 4б).

Назовем открытую по умолчанию папку SCHEMATIC1 другим именем, например mux2_HS, иначе ее будет трудно отыскать в библиотеке. Сохраним проект и закроем рабочее окно со схемой замещения. Последняя операция совершенно необходима, иначе не сработает метод «Drag-and-drop», да и вообще любое копирование или перемещение.

Теперь можно открыть библиотеку my_lib.olb и скопировать в нее папку mux2_HS. Убедитесь, что операция выполнена успешно: в файловой структуре библиотеки появилась нужная папка, а в кэше библиотеки (Library Cache) — соответствующие компоненты, входящие в схему замещения.

Нам еще предстоит создать внешнее описание иерархического символа (рис. 4б). Подробно процесс проектирования символов будет обсуждаться на следующем уроке, а сейчас мы лишь поверхностно познакомимся с этой работой. Заметим лишь, что иерархический символ, как и любой абстрактный компонент, создавать гораздо проще, чем символы реальных компонентов. Не надо рисовать контакты питания и земли, нет необходимости задавать упаковочную информацию.

Библиотека my_lib.olb уже открыта. Щелкнем на папке с ее названием правой кнопкой мыши и выполним команду **New Part**. Откроется панель New Part Properties. Введем в поле Name имя символа mux2_HS, в следующем окне префикс U изменим на HS и перейдем к самому важному: нам предстоит сделать ссылку на схему замещения иерархического символа. Нажмем кнопку Attach Implementation… (Подключить реализацию, то есть дополнительное описание). В нашем случае — это схема замещения, хранимая в папке mux2_HS.

Откроется диалоговая панель с тем же названием (рис. 5). В верхнее поле Implementation Type введем тип внутреннего описания: Schematic View — схема замещения, в следующее поле Implementation (реализация) — имя папки, где она хранится, нижнее поле оставим пустым, так как схема замещения находится в той же самой библиотеке, что и символ. Разместим контакты и нарисуем графику символа в соответствии с рис. 4б (урок 6).

Протестируем наш символ. Создадим новый проект test_mux2_HS.opj, подключим к нему личную библиотеку my_lib.olb и файл ttl.vhd с функциональными моделями компонентов, используемых в проекте. Разместим в окне графического редактора одну копию иерархического символа mux2_HS, подведем к его выводам проводники и проименуем их.

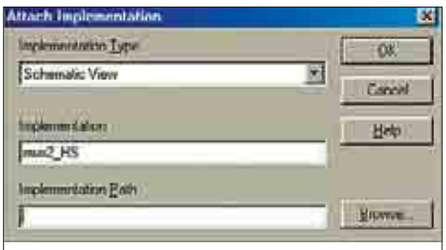


Рис. 5. Подключение к непримитивному символу внутреннего описания — схемы замещения

Теперь предстоит самая важная работа: надо сообщить редактору OrCAD Capture, что символ mux2_HS не является примитивом, что это иерархический символ.

Дважды щелкнем на нем левой кнопкой мыши, и, когда откроется окно редактора свойств Property Editor, изменим значение свойства Primitive с DEFAULT (по умолчанию символ считается примитивом) на NO. Убедимся, что символ теперь имеет подчиненную схему. Для этого достаточно выполнить команду **View/Descend Hierarchy** или нажать комбинацию клавиш Shift+D. Редактор должен понизить иерархию описания и показать схему замещения непримитивного символа mux2_HS.

Если в схеме много непримитивных символов, то проще выполнить групповую операцию по изменению их статуса. В менеджере проекта выделим проект (имя файла с расширением *.dsn) и выполним команду **Options/Design Properties…**. В открывшейся диалоговой панели выберем вкладку Hierarchy и в разделе Part установим опцию Nonprimitive.

Остается промоделировать нашу простую схему, содержащую всего один иерархический символ. С этой работой вы уже хорошо знакомы, и, надеюсь, она не вызовет никаких затруднений.

А теперь поговорим более подробно о внутренней библиотеке проекта, называемой кэшем (Design Cache). Есть у нее и другое имя — внедренный архив. Создадим новый проект Cache.opj, откроем папку Design Cache и убедимся, что, кроме символа TitleBlock0 там ничего нет.

Разместим в рабочем окне редактора схем элемент 7404 из библиотеки my_lib.olb. Активизировав окно менеджера проекта, вы увидите, что имя только что нарисованного на схеме элемента 7404 появилось в списке Design Cache. Проделаем то же самое для элементов 7408 и 7432. Их копии тут же появятся в кэше проекта. Отсюда вывод — кэш как в зеркале показывает все элементы, которые встречаются в проекте. Если в проекте много схем, а в каждой — много страниц, то все равно кэш будет общим для всех.

Кстати, если вам понадобятся элементы, которые уже есть в кэше, то их можно вызывать в точности так же, как и из других библиотек.

Попробуем удалить элемент 7404 из схемы. Как отреагирует на эти действия кэш, ведь теперь таких элементов в схеме нет? Увы, обратную операцию он не выполняет так же лихо, как прямую. Нам придется самим побеспокоиться о «мусоре» во встроенной библиотеке. Выделим папку Design Cache и выполним команду **Design/Cleanup Cache**. Копии элементов, отсутствующих в схеме, исчезнут из списка хранимых в кэше символов.

Проделаем еще одну интересную (и полезную) вещь. Поместим на схеме элемент 7410 из системной библиотеки ttl.olb. Понятно, что его копия тут же появится в кэше. Откроем личную библиотеку my_lib.olb и методом «Drag-and-drop» отбуксируем в нее копию элемента 7410 из кэша. Обратите внимание, нам не пришлось «бороться» с алиасами, заниматься редактированием элемента, и тем приятнее отмечать, что он без всяких трудов оказался в нашей библиотеке.

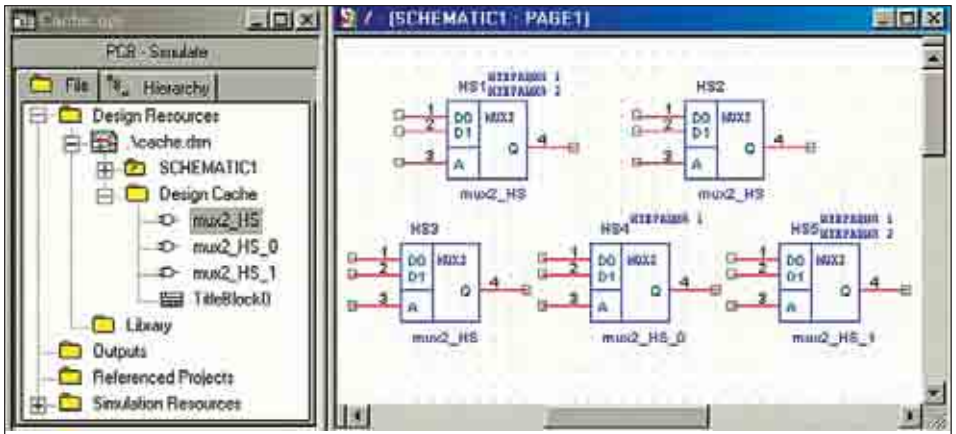
Описанный окольный путь использует кэш в роли «перевалочной базы» при копировании элементов из системной библиотеки в пользовательскую. На деле он оказывается гораздо эффективнее, чем тот неуклюжий метод, которым мы пользовались в начале урока.

В процессе проектирования возникает порой необходимость отредактировать уже размещенный на схеме элемент. Вспомним, что его оригинал находится в библиотеке, откуда он был извлечен, а в кэше проекта хранится его копия.

Следовательно, у нас есть две возможности:

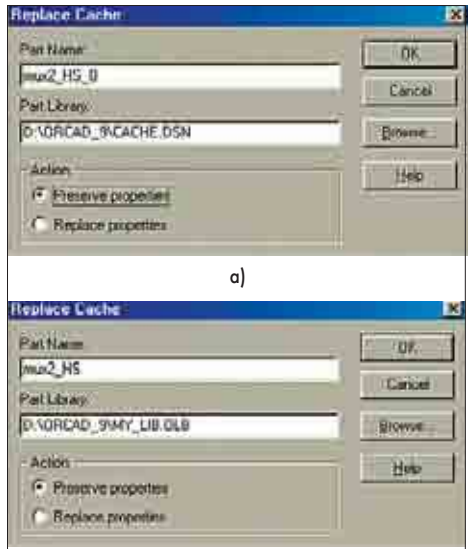
- Внести изменения в копию и, убедившись, что получен желаемый результат, заменить старый библиотечный символ новой версией.
- Отредактировать библиотечный символ непосредственно в библиотеке, а затем обновить в проекте все его устаревшие копии.

Первый способ кажется более логичным и безопасным, с него и начнем. Разместим на странице схемы две копии символа mux2_HS из библиотеки my_lib.olb. Выделим первую из них и выполним команду **Edit/Part**. Редактор



а) б)

Рис. 6. Редактирование элемента mux2_HS непосредственно в схеме проекта



а)

б)

Рис. 7. Диалоговая панель для замены старой копии элемента в проекте:
а) замена на новую версию из библиотеки;
б) восстановление потерянной с библиотекой связи

OrCAD Capture перейдет в режим редактирования компонентов.

Внесем в наш символ какие-нибудь изменения, например добавим текст «ИТЕРАЦИЯ 1» и сохраним новую версию. Редактор поинтересуется, хотим ли мы распространить изменения на все такие же элементы схемы (Update All) или только на текущий символ (Update Current). Можно вообще отказаться от неудачно выполненной работы, нажав кнопку Discard.

Выберем режим Update Current. Окно редактора компонентов закроется, и вы обнаружите, что на схеме появился отредактированный элемент, а его новая копия под именем mux2_HS_0 добавилась в кэш проекта.

Выполните еще одну итерацию с этим же элементом. Добавьте текст «ИТЕРАЦИЯ 2» и сохраните результат. В кэше проекта обнаружится еще одна строчка — mux2_HS_1 (рис. 6а). Как видно, кэш накапливает все редакции элемента, и вы можете при необходимости вернуться к промежуточным версиям.

Убедимся, что такая возможность действительно есть. Инициируем команду **Place/Part** (или Shift+P), в списке библиотек выделим строку Design Cache и один за другим разместим на схеме все три версии нашего элемента: mux2_HS, mux2_HS_0 и mux2_HS_1 (рис. 6б, внизу). Неудачные варианты редактирования всегда легко удалить, а затем командой **Design/Cleanup Cache** очистить кэш от мусора. Рабочий вариант можно скопировать в свою библиотеку, удалив старый символ и восстановив имя mux2_HS для новой редакции. Впрочем, не будем торопиться с этой простой работой, ведь наши эксперименты еще не закончены.

Рассмотрим другой способ редактирования элемента (на примере все того же многострадального символа mux2_HS). Теперь будем редактировать элемент непосредственно в самой библиотеке my_lib.olb, где он хранится. Откроем эту библиотеку (команда **File/Open/Library**), найдем в ее каталоге нужный символ и, щелкнув на нем правой кнопкой, выполним команду Edit Part. Внесем в него какие-нибудь изменения, например добавим текст «NEW». Сохраним отредактированный символ и вернемся в наш проект Cache.opj.

Выделим в кэше имя mux2_HS и выполним команду **Design/Update Cache** (Модифицировать кэш). Мы увидим, что элементы HS2 и HS3 заменены новой копией из библиотеки: это легко заметить — рядом с ними появился текст «NEW». Однако попытка проделать то же самое с другими версиями символа (mux2_HS_0 и mux2_HS_1) ни к чему не приведет: у них другие имена, мы их редактировали в проекте и, увы, потеряли связь с библиотекой.

Как же с честью выйти из создавшейся ситуации и восстановить потерянную связь с библиотекой? Для этого надо воспользоваться командой **Replace Cache…** из меню Design. Выделим в кэше строку mux2_HS_0 и выполним эту команду. Откроется диалоговая панель Replace Cache (рис. 7а).

В верхнем поле панели вы увидите имя элемента из кэша — его-то мы и собираемся заменить. В нижнее поле вписано имя проекта (Cache.dsn), где надо выполнить эту работу.

Итак, мы видим, «*что надо заменить*». Теперь укажем, «*чем это заменить*». Введем в верхнее поле имя библиотечного элемента mux2_HS, а в нижнем укажем библиотеку my_lib.olb, из которой надо взять элемент. Найдем ее с помощью кнопки Browse (рис. 7б).

В разделе Action (Действие) можно уточнить, как выполнять замену: с сохранением свойств копии (Preserve properties) или с заменой их на свойства оригинала (библиотечного элемента). Если вы выберете второй вариант (Replace properties), то, нажав OK, обнаружите, что редактор сменил имена элементов: вместо mux2_HS_0 теперь стоит имя mux2_HS. Сменились, правда, и позиционные обозначения элементов, но это легко исправить.

В заключение заметим, что командой **Design/Replace Cache…** можно заменять не только устаревшие копии одного и того же элемента, но и совершенно разные элементы — например, таким приемом можно сменить все вентили 7408 (2И-НЕ) на элементы 7432 (2ИЛИ), сохранив при этом электрические связи между их выводами и проводниками.