Как создавать символы компонентов.

Урок 6

На принципиальной схеме компонент представляется в виде условного графического обозначения (УГО). Его принято называть символом. Символы компонентов объединяются в библиотеки, имеющие расширение *.olb. Об этом уже говорилось в предыдущей статье.

Александр Шалагинов, к. т. н.

shalag@vt.cs.nstu.ru

А сейчас речь пойдет о разновидностях символов. Они делятся на основные и вспомогательные (рис. 1). Вспомогательные символы представляют собой небольшую группу, куда входят соединители страниц (Off-Page Connector), порты иерархических блоков (Hierarchical Port), «угловые штампы» (Title Block) и символы подключения цепей питания (Power). Они создаются командой **New Symbol** из всплывающего меню или командой **Design/New Symbol…** из выпадающего меню.

Подавляющее же большинство относится к группе основных символов. Процесс их проектирования инициируется другой командой — **New Part**. Пользователей, знакомых с САПР PCAD, слово Part (дословно — часть, корпус) может ввести в заблуждение. В пакете OrCAD Part — это графическое изображение компонента на схеме. Однако, если компонент состоит из нескольких секций, то изображение любой из них будет называться другим словом — Symbol. Таким образом, перевод слова Symbol зависит от контекста: это может быть вспомогательный символ или графический образ одной секции основного компонента.

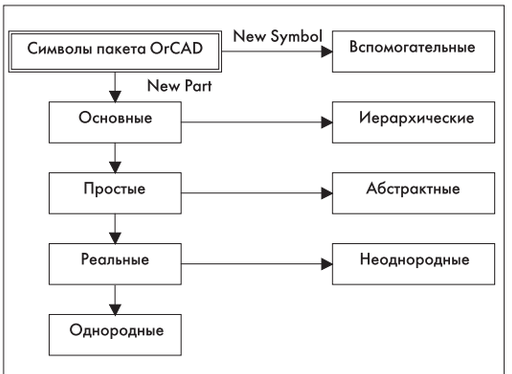


Рис. 1. Разновидности схемных описаний в пакете OrCAD

Основные символы подразделяются на простые и иерархические. Простые символы называют структурными *примитивами*, и в ходе проектирования они не подлежат какому-либо расчленению (детализации). Примитивы представляют собой объекты с известным поведением. Другими словами, они всегда имеют функциональное описание, часто в виде встроенных в систему поведенческих моделей. В пакете OrCAD — это VHDL- или SPICE-модели.

Иерархические символы в САПР OrCAD реализованы в виде иерархических блоков (Hierarchical Block) или не примитивных символов (Nonprimitive Part). С первой разновидностью мы познакомились на уроке 3, со второй — на предыдущем занятии.

Иерархические символы имеют внешнее и внутреннее описания.

При понижении уровня описания такого символа он превращается в структуру, показывая «спрятанную» в нем подсхему или VHDL-код. Внутреннее описание в большинстве случаев — это подчиненная схема, которую часто называют схемой замещения или эквивалентной схемой.

Среди простых символов можно, в свою очередь, выделить две разновидности:

- символы реальных компонентов;
- символы абстрактных компонентов.

Отличительным признаком реального компонента, например микросхемы 555LA3, является наличие упаковочной информации в его схемном описании, с указанием контактов питания и земли, и ссылка на конструкторское описание (тип корпуса, например DIP14 или SO14).

Основное отличие абстрактных компонентов заключается в том, что для них не указывается, а, возможно, и не существует конкретная физическая реализация. Например, мы можем создать символ компонента NAND2 (логический элемент 2И-НЕ), не утруждая себя описанием упаковочной информации и ссылками на тип корпуса. Никто не запретит вам нарисовать двухвходовой мультиплексор MUX2, описать его функцию и использовать затем в проекте, хотя в действительности такой микросхемы нет.

Из-за отсутствия данных о физической реализации, абстрактные компоненты не позволяют довести проект до реального воплощения. Тем не менее, они широко применяются на первых этапах функционального проектирования и при построении иерархических блоков.

И наконец, реальные символы делятся на однородные (Homogeneous) и неоднородные (Heterogeneous). Первые описывают компоненты, состоящие из секций (элементов) одного типа (например, 555LA3 содержит четыре *одинаковых* элемента 2И-НЕ), а вторые — из секций разного типа (например, 555LR11 включает два *различных* элемента: 2-2И-2ИЛИ-НЕ и 3-3И-2ИЛИ-НЕ). Заметим, что не каждая САПР предоставляет пользователю такие возможности. Например, в пакете DesignLab 8 вообще нельзя создавать неоднородные компоненты.

Проектирование абстрактных компонентов

Начнем с самого простого — с процесса создания абстрактных символов, которым не требуется задавать упаковочную информацию и определять физическую реализацию. Нет необходимости вводить контакты питания и земли.

Запустим графический редактор OrCAD Capture и выясним, устраивают ли нас его параметры, заданные по умолчанию. Исполним команду **Options/Design Template…**. После того как откроется диалоговая панель с одноименным названием, выберем закладку Page Size. Установим метрическую систему единиц (Millimeters), стандартный размер страницы A4 и шаг сетки (Pin-to-Pin Spacing), равный 2.50 millimeters. Нажмем кнопку OK. Эта информация запишется в файл Capture.ini (в раздел [Design Template]), и все новые проекты будут получать по умолчанию сделанные установки.

Создадим новую (или откроем уже существующую) библиотеку, в которую будем помещать проектируемые символы (команда **File/New/Library…** или **File/Open/Library…**) и определим ей уникальное имя, например my_lib.olb (см. урок 5).

Вновь щелкнем правой кнопкой мыши на имени библиотеки и выберем команду **New Part** — создать графический образ (УГО) нового компонента. Напомним, что команда **New Symbol** нам не подходит — она используется только для проектирования *вспомогательных* символов, например «углового штампа» или иерархических портов. Об этом мы уже говорили.

Для простоты выберем в качестве объекта проектирования двухвходовой вентиль 2И-НЕ и назовем его NAND2. После исполнения команды **New Part** на экране появится диалоговая панель **New Part Properties** (рис. 2), где предстоит немного поработать.

Прежде всего, в поле Name: надо ввести имя проектируемого символа NAND2. Затем в поле Part Reference Prefix: заменим префикс U на DD. Дело в том, что по отечественным стандартам позиционные обозначения цифровых элементов начинаются именно с этих букв, например DD1.1, DD3.6, DD18 и т. д.

В поле PCB Footprint задается тип корпуса, в который помещается реальный компонент. Для абстрактного символа такая информация не имеет смысла, поэтому оставим его пустым.

Установим флажок Create Convert View, чтобы показать, что для проектируемого элемента 2И-НЕ существует логически эквивалентное обозначение (рис. 3). Скажем больше, мы не только отмечаем этот отрадный факт, но и собираемся нарисовать оба изображения (нормальное Normal и конвертированное Convert).

Рис. 2. Диалоговая панель для задания основных параметров символа

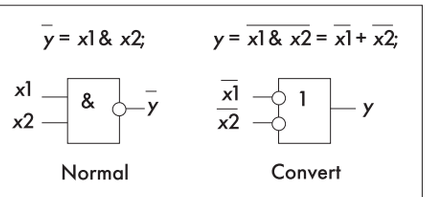


Рис. 3. Две логически эквивалентные формы изображения элемента 2И-НЕ

В следующем небольшом окне Parts per Pkg требуется установить число элементов, размещаемых (упаковываемых) в корпусе микросхемы. Оставим здесь цифру 1, заданную умолчанием. Для абстрактного символа эта информация не несет реального смысла.

В разделах Package Type (тип упаковки — однородные или неоднородные символы) и Part Numbering (способ нумерации символов в упаковке -буквенный или цифровой) оставим все как есть. В итоге диалоговая панель New Part Properties должна выглядеть так, как показано на рис. 2.

Если вам захочется впоследствии вернуться к рассмотренной панели, то придется воспользоваться командой **Options/Package Properties…**, хотя, судя по названию, она совсем не подходит для этих целей.

И еще один полезный совет. Первые эксперименты, как правило, бывают неудачными, и нередко появляется желание удалить сделанное и начать все заново. Если вы захотите удалить свой элемент, то предварительно надо закрыть окно с УГО проектируемого компонента, и только тогда станет доступной команда **Design/Delete…** или клавиша DEL. Сознаюсь, я и сам не сразу об этом догадался.

После нажатия на кнопку OK вы увидите рабочее окно графического редактора Part and Symbol Editor, на котором пунктиром изображен шаблон будущего символа (рис. 4).

Его габариты всегда можно изменить «буксировкой» углов. Необходимые размеры символа определяются числом входных (или выходных) контактов. Поэтому начинать проектирование лучше всего с размещения именно этих объектов.

В правой (по умолчанию) части рабочего окна находится палитра инструментов (Tool Palette), используемых для создания символа. Конечно, все они могут быть вызваны и из выпадающего меню Place.

Щелкнем на иконке Place pin, показанной справа, или активизируем команду с тем же названием.

На экране появится диалоговая панель для задания параметров контакта. Введем его имя IN1, номер 1, форму вывода Line и тип вывода Input (рис. 5).

Рис. 4. Рабочее окно редактора символов

Нажмем кнопку OK и поместим первый контакт слева от пунктирного прямоугольника. Вы обнаружите, что редактор не собирается бросать начатую работу и намерен продолжать размещение выводов. Их параметры могут не соответствовать вашим желаниям, но не противьтесь и определите место для всех остальных контактов. Потом мы их отредактируем. Такой стиль работы может показаться не очень элегантным, но он быстрее приводит к результату.

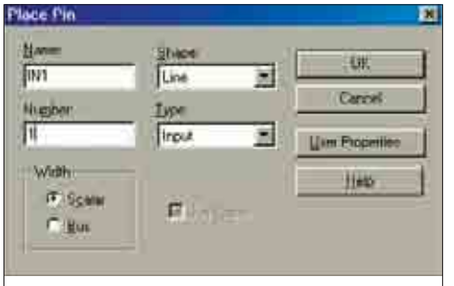


Рис. 5. Диалоговая панель для задания параметров контакта

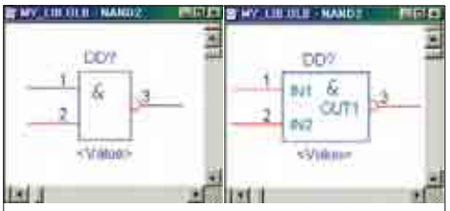
Закончив размещать выводы, дважды щелкнем на втором контакте. Убедимся, что там все правильно — редактор сам инкрементировал имя. Параметры третьего контакта придется изменить. Во-первых, это выходной контакт, поэтому вместо имени IN3 введем OUT1, а тип Input переправим на Output. Во-вторых, вывод OUT1 не прямой, а инверсный, поэтому заменим его форму на Dot (вместо Line).

Остается нарисовать графику символа. Активизируем команду **Place/Rectangle** и нарисуем контур символа «поверх» пунктирного прямоугольника. Командой **Place/Text…** укажем функцию &, выполняемую компонентом. Возможно, вам захочется изменить размеры текста, заданные по умолчанию.

При вводе текста рекомендуется отключать режим дискретного перемещения объектов по узлам сетки, чтобы более точно позиционировать его. Для этого достаточно щелкнуть на пиктограмме Snap To grid, показанной справа. Она находится на пане-

ли инструментов и приобретает красный цвет в режиме «плавного» перемещения курсора мыши. На самом деле в этом режиме шаг перемещения объектов остается дискретным, но уменьшается до величины, равной 0,1 шага сетки.

Законченный рисунок должен выглядеть так, как показано на рис. 6а.



а) б)

Рис. 6. Графическое описание компонента NAND2

Впрочем, мы упустили одну вещь: для простых логических элементов имена выводов очевидны. Они не добавляют к графическому описанию новой информации, а только перегружают его ненужными деталями. Поэтому на УГО простой логики имена контактов делают невидимыми. Это дает еще один плюс: теперь на размещение имен не требуется место, а потому символ можно перерисовать более компактно (рис. 6б).

Чтобы сделать имена невидимыми, надо дважды щелкнуть левой кнопкой мыши в зоне размещения символа, не попав при этом на какой-либо объект. Появится диалоговая панель User Properties, на которой следует выбрать строку Pin Names Visible и заменить значение True на False (рис. 7).

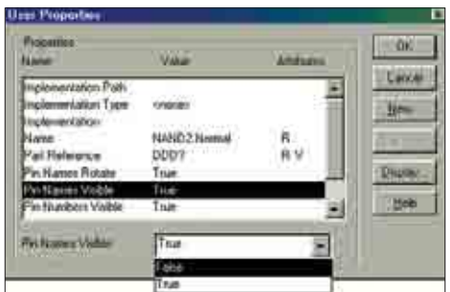


Рис. 7. Диалоговая панель редактирования параметров (свойств) символа

То же самое можно проделать и с номерами контактов, так как они не несут для абстрактных символов никакой важной информации.

Последнее, что предстоит сделать — это создать логически эквивалентную форму проектируемого символа (Convert). Активизируем команду **View/Convert** и отредактируем предлагаемую редактором заготовку в соответствии с рис. 3.

Прежде всего, изменим форму входных контактов IN1 и IN2, выбрав вид Dot, и выходного контакта OUT1, указав для него прямой выход Line. Затем поменяем функцию 2И (&) на 2ИЛИ (1). Остальные операции, связанные с созданием графики и отключением видимости имен и номеров контактов, комментировать не станем. В результате конвертированный символ должен принять вид, показанный на рис. 8.

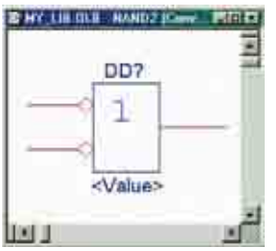


Рис. 8. Логически эквивалентная форма (Convert) символа NAND2

Строго говоря, при создании конвертированного символа требуется изменить не только форму контактов, но и проинвертировать их имена. Однако последнее ни в коем случае не следует делать. Иначе у вас возникнут проблемы при создании VHDL-моделей. Ну, а чтобы избавиться от ощущения не до конца выполненной работы, можно утешать себя тем, что имена контактов все равно не видны на схемном обозначении.

Сохраним символ NAND2 и закроем окно редактора Part and Symbol Editor. Заметим, что двойным щелчком на созданном символе всегда можно вернуться в режим его редактирования. Если все сделано правильно, закроем библиотеку my_lib.olb.

Теперь надо протестировать наше «детище» и посмотреть, как оно будет выглядеть на схеме рядом с фирменными компонентами. С этой целью создадим новый проект test_nand2, подключим к нему библиотеку my_lib.olb и разместим нормальное и конвертированное изображения символа NAND2 на схеме. Рядом поместим их зарубежный аналог 7400 из библиотеки ttl.olb (рис. 9). Для наглядности визуализируем свойство Graphic всех компонентов (по умолчанию оно не выводится на экран).

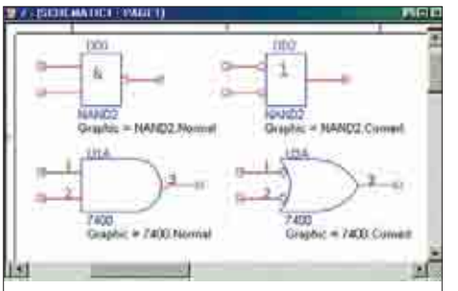


Рис. 9. Результаты проектирования символа NAND2

Сравнивая результат, отметим с некоторой долей самодовольства, что наши символы ничуть не хуже, они даже более компактны, а это немаловажно для больших схем.

Нам еще предстоит создать для нарисованного символа его VHDL-модель и верифицировать ее. Активизируем команду **File/New/VHDL File** и «ручками» напишем VHDL-код модели логического элемента NAND2.

Выполняя эту работу, удобно пользоваться «заготовками» допустимых конструкций языка VHDL и примерами их использования. Введите команду **Edit/Samples…** и вы получите доступ к этим образцам-заготовкам. Начинайте с конструкции Design Unit, которая определяет структуру VHDL-модели. Фактически дело сводится к тому, чтобы заменить стандартные имена шаблона реальными именами в нашем символе. Самое трудное (пока мы не знакомы с языком VHD — написать функцию элемента. В этой ситуации проще всего посмотреть, как это делают «умные люди».

Например, можно открыть текстовый файл Pldgates.vhd и найти VHDL-код для элемента NAND2 или командой **Edit/Samples…** выбрать шаблон Boolean Operators и посмотреть, как реализуется нужная нам логическая операция:

```
Y(3) <= A nand B;
```

В итоге текст VHDL-модели для нашего элемента должен выглядеть так, как показано на рис. 10.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;      -- подключение стандартных
библиотек
entity nand2 is                   -- описание интерфейса
 port (IN1, IN2: in std_logic;
       OUT1: out std_logic);
end nand2;

architecture MODEL of nand2 is    -- архитектурное тело
 begin
-- insert statements here
OUT1 <= IN1 nand IN2 after 10ns; -- описание функции
end MODEL;
```

Рис. 10. Текст VHDL-модели для логического элемента NAND2

Командой **Edit/Check VHDL Syntax** проверим, есть ли в модели синтаксические ошибки. Если они имеются, выявим и устраним их.

Нам еще предстоит убедиться, что созданный символ NAND2 действительно реализует логическую функцию 2И-НЕ. Эта процедура называется верификацией. Удалим из схемы, показанной на рис. 9, все ненужные элементы, кроме одного, подключим к нему внешние цепи и иерархические порты так, как показано на рис. 11.

Перейдем в окно менеджера проекта и выполним команду **Tools/Create Netlist..** (создать список цепей). Выберем закладку VHDL, так как мы хотим иметь список цепей в формате языка VHDL. Установим флажок View Output, чтобы увидеть список цепей на экране, и нажмем OK. Графический редактор OrCAD Capture автоматически сгенерирует описание схемы на языке VHDL. В нашем случае оно выглядит весьма незамысловато (рис. 12).

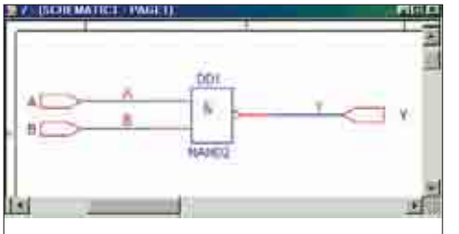


Рис. 11. Схема верификации логического элемента NAND2

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY SCHEMATIC1 IS PORT (    -- описание интерфейса
                A : IN std_logic;
                B : IN std_logic;
                Y : OUT std_logic
); END SCHEMATIC1;

ARCHITECTURE STRUCTURE OF SCHEMATIC1 IS -- архитектурное тело

-- COMPONENTS

COMPONENT NAND2     -- объявление компонента NAND2
                PORT (
                IN1 : IN std_logic;
                IN2 : IN std_logic;
                OUT1 : OUT std_logic
                ); END COMPONENT;

-- SIGNALS

-- GATE INSTANCES

BEGIN
DD1 : NAND2     PORT MAP( -- карта порта экземпляра компонента NAND2
    IN1 => A,      -- описание связей между формальными портами компонента
    IN2 => B,      -- и фактическими именами цепей
    OUT1 => Y
);
END STRUCTURE;
```

Рис. 12. VHDL-список цепей для схемы верификации элемента NAND2

Легко заметить, что схема представлена в виде объекта проекта с именем SCHEMATIC1, который имеет два входных (IN1, IN2) и один выходной (OUT1) порты. В архитектурном теле объявляется используемый компонент NAND2, а в исполняемом разделе создается его конкретный экземпляр DD1 и показывается, как его выводы подключаются к иерархическим портам.

Передадим наш проект на моделирование (команда **Tools/Simulate…**), выберем тип моделирования In Design и зададим временные диаграммы на входах A и B (команда **Stimulus/New Interactive…**). Сохраним их в файле test_nand2.stm.

Если сейчас заглянуть в журнал протокола Session Log, то вы увидите предупреждение:

Warning: [Load112] no entity or package named 'NAND2' found

Программа моделирования OrCAD Simulate не нашла VHDL-модель элемента NAND2. Исправим оплошность. Щелкнем правой кнопкой мыши на папке In Design менеджера проекта и исполним команду **Add File**.

Найдем файл nand2.vhd и подключим его к проекту, указав тип — VHDL SimModel (VHDL-модель). Теперь нас ждет успех. Перегрузим проект (команда **Simulate/Reload Project**), убедимся, что в журнале протокола нет никаких предупреждений, и запустим проект на моделирование (команда **Simulate/Run…**).

Результаты моделирования (рис. 13) говорят о том, что элемент NAND2 правильно выполняет свою функцию.

Рис. 13. Результаты верификации логического элемента NAND2

Проектирование реальных компонентов

Создание графических изображений (символов) реальных компонентов потребует дополнительных усилий. Нам придется повторить всю работу, которая выполнялась для абстрактного символа и, кроме того, задать упаковочную информацию, контакты питания и земли, указать тип корпуса, в котором размещается реальный компонент. Уже знакомую работу выполним конспективно, сосредоточив внимание на отличиях.

В качестве объекта проектирования возьмем компонент 555LA3, содержащий четыре одинаковых двухвходовых вентиля 2И-НЕ. Попытаемся «формализовать» технологию создания реального символа.

1. Открываем библиотеку my_lib.olb (команда **File/Open/Library…**). Вызываем диалоговую панель New Part Properties (команда **Design/New Part…**) и вводим следующую информацию:
1.1. Имя компонента: 555LA3.
1.2. Префикс позиционного обозначения: DD.
1.3. Типовой корпус: DIP14 (для абстрактного символа этот пункт не выполняется).
1.4. Устанавливаем флажок Create Convert View, чтобы создать логически эквивалентную форму графического изображения.
1.5. Устанавливаем число секций в корпусе: 4 (Parts per Pkg).
1.6. Оставляем тип упаковки Homogenous, так как в корпус помещаются одинаковые символы.
1.7. Выбираем цифровой (Numeric) способ нумерации секций компонента.
1.8. Щелкаем на кнопке OK.
2. В открывшемся рабочем окне редактора символов (Part and Symbol Editor) создаем графический образ компонента:
2.1. Вводим входные и выходной контакты IN1, IN2, OUT1 (команда **Place/Pin…**). Заметим, что номера контактов это не просто порядок их ввода. Они должны соответствовать номерам «ножек» микросхемы, к которым подключаются входы и выход *первой* секции. Для контакта OUT1 задаем форму вывода Dot (инверсный выход). Все логические контакты делаем не видимыми (команда **Options/Part Properties…**, свойство Pin Names Visible, значение False).
2.2. Вводим контакты питания VCC и земли GND. В реальной микросхеме они подаются на 14 и 7 «ножки», поэтому в поле Number указываем соответствующие цифры. Выводы имеют нулевую длину (Zero_Length) и тип Power. Флажок Pin Visible оставим сброшенным, так как выводы питания обычно не видны на схеме (рис. 14).
2.3. Задаем логическую функцию &, которую выполняет элемент (команда **Place/Text…**) и помещаем ее внутри символа.
2.4. Корректируем габаритные размеры символа («буксировкой» его углов).
2.5. Рисуем графику символа (команда **Place/Rectangle**).
2.6. Активизируем команду **View/Convert** и создаем логически эквивалентное изображение символа (детальное описание этого процесса рассмотрено ранее при проектировании абстрактного символа).

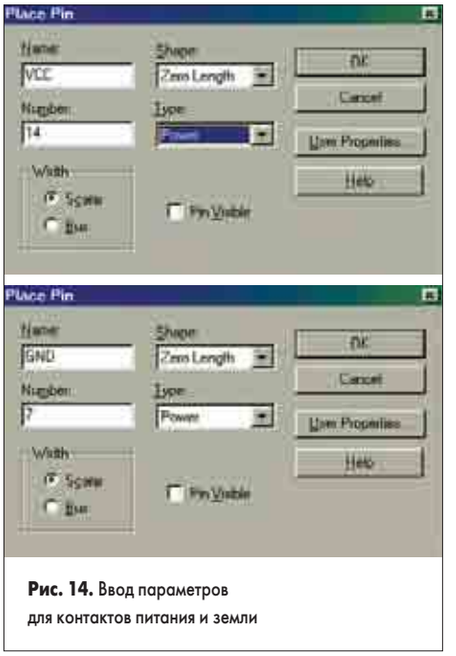


Рис. 14. Ввод параметров для контактов питания и земли

3. Возвращаемся к нормальному изображению символа (команда **View/Normal**) и вводим упаковочную информацию для второй секции (команда **View/Next Part**). Придется заглянуть в справочник по интегральным микросхемам, чтобы узнать, на какие «ножки» выводятся входы и выход второй секции (их номера соответственно 4, 5 и 6). Двойным щелчком на контакте вызываем его параметры и в пустое поле Number вводим нужный номер. *Не забудьте проделать эту операцию и для контактов питания и земли!* Аналогичную работу выполняем для всех оставшихся секций.
4. Сохраняем созданный символ и закрываем библиотеку.
5. Протестируем полученный символ. С этой целью создадим новый проект test_555LA3.opj и в рабочем окне редактора схем (Schematic Page Editor) разместим 5–6 копий нашего символа (число копий должно превышать число секций в компоненте). Выполним автоматическую упаковку (команда **Tools/Annotate…** менеджера проектов). Если при вводе упаковочной информации мы не наделали ошибок, то ре-

дактору понадобится всего два корпуса, чтобы разместить все элементы. При этом номера позиционных обозначений и секций будут возрастать в направлении слева направо и сверху вниз (рис. 15).

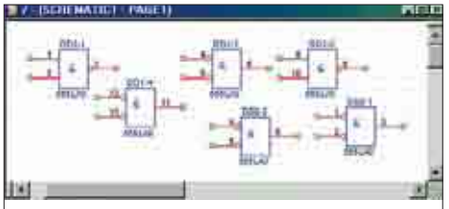


Рис. 15. Проверка правильности задания упаковочной информации для элемента 555LA3

6. Выполним функциональную верификацию нашего символа. Однако прежде надо создать его VHDL-модель. Эту проблему опять придется решать с помощью «умных людей». Посмотрим, как выглядит VHDL-модель для зарубежного аналога 7400. Мы найдем нужное описание в файле ttl.vhd и приспособим его для наших целей.

Придется заменить имя оригинала \7400\ на \555LA3\ во всех местах, где оно попадается. В компоненте 7400 имена контактов называются I0, I1 и O (от слов Input и Output), а секции именуются как A, B, C и D. У нас выводы имеют имена IN1, IN2 и OUT1, а секции не именуются, а нумеруются -1, -2, -3 и -4.

Хуже всего то, что в полное имя вывода добавляется еще и номер секции, например IN1_-1 (у фирменных элементов соответственно I0_A). Полученный идентификатор не удовлетворяет синтаксису языка VHDL. Чтобы компилятор не браковал такие «ошибочные» идентификаторы, они заключаются в backslash-скобки, вот так: \IN1_-1\. Кстати, сказанное касается и названий компонентов 7400 и 555LA3. Имена начинаются с цифры, что противоречит правилам языка VHDL. Поэтому они тоже заключены в backslash-скобки: \7400\ и \555LA3\.

Как видите, предстоит немало кропотливой работы по замене имен. Можно ли избежать этой работы? Оказывается, такая возможность есть. Удалите на схеме все элементы, кроме одного, и создайте VHDL-список цепей (команда **Tools/Create Netlist…** менеджера проектов). В нем вы без труда найдете нужный фрагмент модели. Остается скопировать его и вставить в свою модель. После всех перипетий должен получиться такой результат (рис. 16).

Сохраним созданный файл под именем 555LA3.vhd и присоединим его к проекту. С этой целью щелкнем правой кнопкой мыши на папке In Design и исполним команду Add File.

Оставшаяся часть работы нам уже знакома. Дорисуем схему верификации в соответствии с рис. 11 и передадим наш проект на моделирование (команда **Tools/Simulate…**). Выберем тип моделирования In Design и зададим временные диаграммы на входах A и B (команда **Stimulus/New Interactive…**). Сохраним их в файле test_555LA3.stm.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.orcad_prims.all;

ENTITY \555LA3\ IS PORT(
        \IN1_-1\ : IN std_logic;
        \IN2_-1\ : IN std_logic;
        \OUT1_-1\ : OUT std_logic;
        VCC : IN std_logic;
        GND : IN std_logic;
        \IN1_-2\ : IN std_logic;
        \IN2_-2\ : IN std_logic;
        \OUT1_-2\ : OUT std_logic;
        \IN1_-3\ : IN std_logic;
        \IN2_-3\ : IN std_logic;
        \OUT1_-3\ : OUT std_logic;
        \IN1_-4\ : IN std_logic;
        \IN2_-4\ : IN std_logic;
        \OUT1_-4\ : OUT std_logic
END \555LA3\;

ARCHITECTURE model OF \555LA3\ IS

 BEGIN
 \OUT1_-1\ <= NOT ( \IN1_-1\ AND \IN2_-1\ ) AFTER 22 ns;
 \OUT1_-2\ <= NOT ( \IN1_-2\ AND \IN2_-2\ ) AFTER 22 ns;
 \OUT1_-3\ <= NOT ( \IN1_-3\ AND \IN2_-3\ ) AFTER 22 ns;
 \OUT1_-4\ <= NOT ( \IN1_-4\ AND \IN2_-4\ ) AFTER 22 ns;
END model;
```

Рис. 16. VHDL-модель компонента 555LA3

В принципе можно использовать и ранее созданный файл test_NAND2.stm с описанием таких же сигналов A и B. Достаточно подключить его к проекту уже знакомым нам способом (командой Add File).

Заметим, что OrCAD не запрещает подключать к проекту несколько STM-файлов и тестировать работу схемы на различных наборах входных данных. В нашем случае можно добавить в проект оба файла: test_NAND2.stm и test_555LA3.stm.

Теперь при передаче схемы на моделирование OrCAD Simulate уточнит, какой из доступных ему STM-файлов следует загрузить для проведения текущего эксперимента (рис. 17). Укажите на желаемый файл и нажмите кнопку OK.

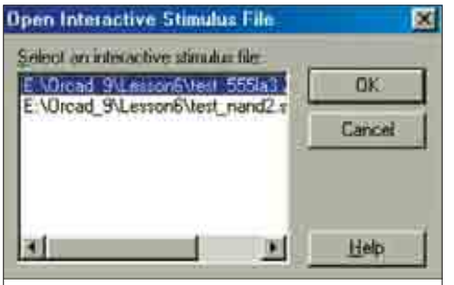


Рис. 17. Выбор конкретного STM-файла с описанием внешних воздействий

Выполнив моделирование, вы увидите результаты (рис. 18), которые подтверждают правильность спроектированного компонента и его VHDL-модели.

Больше нет оснований для сомнений, что он подведет нас на этапах функционального проектирования или разводки печатных плат.

Поэтому с чистой совестью можно поставить на нем «личное клеймо» — своеобразный знак качества. Мы сделаем это просто: добавим к символу еще одно свойство Author = Shalaginov. Конечно, это шутка. Там по праву должна стоять ваша фамилия, но и ответственность за его качество отныне будет лежать на вас. Изменения лучше внести в библиотечный символ, а затем обновить его в схеме командой Design/Update Cache (см. урок 5).

Рис. 18. Результаты функциональной верификации компонента 555LA3

Проектирование реальных компонентов с неоднородными символами

Напомню, что такие компоненты содержат в своей упаковке два или более *разных* по выполняемой функции или по числу внешних выводов элементов. Например, компонент 555LR11, содержит два элемента И-ИЛИ-НЕ с разным числом входов, а в микросхеме 564LP2 объединены два элемента 3ИЛИ-НЕ и один инвертор НЕ. В таких случаях говорят, что компонент содержит секции разного типа (Heterogeneous).

Заметим, что компонентов с неоднородными символами буквально единицы, поэтому их проектирование не является актуальной задачей. Мы уже отмечали, что в пакете DesignLab 8 такой режим вообще не предусмотрен.

К тому же у вас всегда есть альтернатива — несколько неоднородных элементов можно нарисовать как один обобщенный символ. Однако если вы все же решили исследовать данный режим, то помните, что вам придется нарисовать все неоднородные элементы, входящие в компонент.

VHDL-модели неоднородных компонентов *структурно* ничем не отличаются от аналогичных моделей для однородных компонентов. Небольшое отличие состоит лишь в том, что первые содержат *разные* функциональные описания (по числу неоднородных символов), а вторые — копии одного и того же описания (по числу секций).

Кроме того, если имена выводов неоднородных символов разные, то к ним не добавляется номер (имя) секции, а если они одинаковые, то все реализуется уже известным нам способом: к имени контакта добавляется имя или номер секции, например I0_A или IN1_-1. В следующей секции — I0_B или IN1_-2 и т. д.

И последнее. При упаковке элементов по корпусам по неизвестным мне причинам редактор фиксирует ошибку размещения неоднородных компонентов, если элементов не хватает, чтобы полностью укомплектовать корпус микросхемы.