

681.32 (075)

Ш....

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ  
Технологический институт  
Федерального государственного образовательного  
учреждения высшего профессионального образования  
«Южный федеральный университет»  
ПРИОРИТЕТНЫЙ НАЦИОНАЛЬНЫЙ ПРОЕКТ «ОБРАЗОВАНИЕ»**

**Шеболков В.В.**

# **Современные средства автоматизированного проектирования цифровых устройств на ПЛИС**

**Часть 2**

**Проектирование цифровых устройств в системе  
Active HDL 7.1**

**Учебное пособие**

**Таганрог 2007**

УДК 621.317

В.В. Шеболков. Современные средства автоматизированного проектирования цифровых устройств на ПЛИС. Часть 2: Проектирование цифровых устройств в системе Active HDL 7.1 - Таганрог: Изд-во ТТИ ЮФУ, 2007.- 60с.

Рассматривается технология схемотехнического проектирования цифровых устройств на ПЛИС в системе Active HDL 7.1. Описана технология создания проекта иерархической структуры в варианте “снизу-вверх” и на примере проектирования двоично-десятичного счетчика с дешифратором проиллюстрированы принципы проектирования цифрового устройства с иерархической структурой. Рассмотрены вариант технологии работы над проектом верхнего уровня при графическом вводе информации о проектируемом устройстве, а над проектами нижних уровней на языке VHDL.

Табл. 2. Илл. 42. Библиогр.; 2 назв.

Рецензенты:

© Технологический институт  
Южного федерального университета, 2007  
© В.В. Шеболков, 2007

## Содержание

1. ОБЩАЯ ХАРАКТЕРИСТИКА СИСТЕМЫ.....	4
2. СРЕДСТВА ПРОЕКТИРОВАНИЯ В ACTIVE HDL .....	10
3. ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ В ACTIVE HDL 7.1 ...	24
3.1. Основные принципы проектирования цифровых устройств в Active HDL .....	24
3.2. Технология проектирования на HDL-языках .....	25
3.3. Редактирование кода VHDL-файла.....	32
3.4. Технология тестирования проекта .....	45
3.5. Тестирование проекта с помощью испытательного стенда Test Bench .....	52
3.6. Технология графического проектирования .....	56
Заключение .....	60
Библиографический список.....	61

## 1. ОБЩАЯ ХАРАКТЕРИСТИКА СИСТЕМЫ

1.1. Active HDL 7.1 – современная система автоматизированного проектирования (САПР) цифровых устройств на ПЛИС, позволяющая решать проектные задачи как с помощью HDL-языков (VHDL и (или) Verilog), так и в графическом режиме. Система была Active HDL разработана американской фирмой Aldec в конце 90-х годов прошлого столетия и к настоящему времени выпущена уже ее 7-я модификация. Система принципиально не ориентирована на ПЛИС конкретной фирмы. Напротив, она поставляется с библиотеками компонентов, включающих широкий спектр производителей ПЛИС, и пользователь при инсталляции может выбрать тот состав компонентов, который необходим ему. Систему отличает удобный, интуитивно понятный интерфейс. В последних версиях системы проявляется тенденция интеграции инструментов проектирования с другими мощными программными средствами моделирования, в частности с пакетами MatLab и Simulink.

Рабочая среда Active-HDL основана на стандартном MFC-подобном (Microsoft Foundation Classes) окне графического интерфейса пользователя. Каждое окно может быть присоединено, перекрыто или свернуто. Главные компоненты Active-HDL:

- Окно просмотра проекта

- Окно проводника проекта

- HDL редактор

- Редактор автоматов

- Окно просмотра сигнала

- Окно консоли

- Менеджер библиотек

- Редактор принципиальных схем

Все компоненты Active VHDL объединены в единообразную графическую среду, являющуюся основным рабочим пространством проекта - workspace. Workspace организует рабочую область окна проектирования и связывает воедино все элементы среды проектирования. За исключением ядра моделирования, каждый инструмент Active HDL выполнен в форме отдельного окна. В табл.1.1 приведены компоненты Active-HDL 7.1.

Таблица 1.1

Компоненты Active HDL 7.1

Компонент	Описание
Console	Интерактивное окно, предназначенное для ввода - вывода текста. Обеспечивает ввод команд Active HDL, вывод сообщений сгенерированных инструментальными средствами Active HDL.
Design Browser	Окно просмотра - показывает текущее содержание проекта: файлы ресурсов, присоединенных к проекту; содержание файлов рабочих библиотек определяемых по умолчанию; структуру проектируемых устройств выбранных для моделирования; сигналы и переменные, объявленные внутри избранной области текущего проекта.
HDL Editor	HDL редактор - текстовый редактор для создания исходных VHDL (Verilog)-файлов проектируемых устройств. Он отображает определенные синтаксические конструкции различными цветами. Редактор интегрирован с моделирующей подсистемой, что позволяет упростить отладку исходных текстов программ.
Language Assistant	Языковой помощник – вспомогательный инструмент который содержит ряд типичных шаблонов

	конструкций языков VHDL и Verilog. Он объединен с HDL редактором так, чтобы можно автоматически вставить желательный шаблон в редактируемый файл. Языковый помощник позволяет определить собственный шаблон.
State Machine Editor	Редактор автоматов с конечными состояниями – графический инструмент, разработанный для редактирования диаграмм конечного автомата. Редактор осуществляет автоматический перевод графических примитивов в коды VHDL (Verilog).
Waveform Viewer	Просмотрщик формы сигнала показывает результаты моделирования действия тестовых сигналов на проектируемое устройство. Он позволяет графически редактировать форму сигнала, чтобы создать нужные испытательные цифровые сигналы
List	Окно списка показывает результаты моделирования, выполненные в форме таблицы текстового формата. Это позволяет анализировать результаты моделирования с точностью до дельта-циклов.
Watch	Окно просмотра показывает текущие значения выбранных сигналов и переменных в течение моделирования.
Processes	Окно процессов показывает текущее состояние одновременных процессов в разработанном проекте в течение моделирования.
Library Manager	Менеджер библиотек предназначен для управления VHDL (Verilog) библиотеками и их содержанием.
Perl (Tcl) Script	Редакторы сценариев – текстовые редакторы с вмонтированным отладчиком. Он разработаны, чтобы редактировать сценарии на языках Perl/Tcl и выполнять команды Active-HDL

SystemC (Handel-C, C++ Source)	Текстовые редакторы для создания исходных файлов компонентов проектируемых устройств на соответствующих языках программирования.
Design Explorer	Проводник проекта облегчает управление Active-HDL проектами. Он позволяет не запоминать физическую локализацию файлов проекта.

1.2. При реализации проекта в каталоге проекта создается ряд файлов. Это файлы конфигурации проекта, исходные файлы, файлы формы сигнала, файлы Active HDL проектов и т.д. Назначение и основные типы файлов указаны в табл. 1.2.

Таблица 1.2

## Типы файлов проекта

Тип файла	Назначение файла
<b>ADF</b>	design description file - файл описания проекта
<b>ASF</b>	state diagram source file - исходный файл диаграммы состояний
<b>AWF</b>	waveform file - файл формы сигнала
<b>BAS</b>	command source file - исходный командный файл
<b>BDL</b>	block diagram source file - исходный файл принципиальной схемы
<b>CFG</b>	various configuration files - различные конфигурационные файлы
<b>DAT</b>	various data files - файл данных
<b>DO</b>	script file - файл сценария
<b>DRC</b>	design rule check configuration file - файл проверки правил проектирования
<b>EPR</b>	list of source files for compilation - список исходных файлов для компиляции

<b>ERF</b>	compiler errors and messages - ошибки и сообщения компилятора
<b>FLW</b>	flow description file - файл описания потока
<b>LOG</b>	various log files - файл отчета
<b>LIB</b>	library index file - индексный файл библиотеки
<b>MFG</b>	library main file - главный библиотечный файл
<b>OID</b>	HDL wizard auxiliary file - вспомогательный файл Мастера HDL
<b>PL</b>	Perl script file - файл сценария на языке Perl
<b>SDF</b>	standard delay file - стандартный файл задержки
<b>TCL</b>	TCL/TK script file - файл сценария TCL/TK
<b>TXT</b>	text file - текстовый файл
<b>VHD</b>	VHDL source code - исходный VHDL код
<b>VHQ</b>	VHDL source code generated from the routed netlist - исходный VHDL код, сгенерированный из списка соединений
<b>VHDL</b>	VHDL source code - исходный VHDL код
<b>WSP</b>	workspace file - файл рабочей области окна

1.3. Проект представляет собой совокупность файлов, созданных для данного проекта и (или) включенных в него. Каждый проект сохраняется в отдельной папке, называемой папкой проекта. И папка проекта, и проект имеют одно и то же имя. Каждая папка проектов содержит вложенные папки Compile, Log, Src, файл описания проекта с расширением *.ADF*, имя которого совпадает с именем проекта, файл рабочей области с расширением *.WSP* и файл рабочей конфигурации библиотеки проекта с расширением *.CFG*.

Папка Compile содержит рабочие файлы, которые создаются и используются для компиляции и моделирования. Папка Src содержит либо исходный файл с расширением *.VHD*, либо файл диаграмм состояний с расширением *.ASF*.



Конфигурация проекта и его состояние описаны в файле описания проекта **ADF** (design description file), содержащем информацию о структуре проекта: исходных файлах и библиотеках, которые формируют **WSP** - (workspace file) файл рабочей области, содержащий данные о текущем состоянии проекта. Он восстанавливает состояние проекта, когда проект открывается вновь после того, как он был закрыт. Текущее состояние включает такую информацию о том, какие файлы открыты, какова текущая позиция курсора в каждом файле, какие закладки установлены, какие окна закреплены, и т.д.

Поскольку файл рабочего пространства описывает текущее состояние проекта, он может содержать и дополнительную информацию такую, как неустойчивость проекта, обусловленную наличием ошибки. В этом случае, проект, возможно, не удастся открыть из-за нестабильного состояния, записанного в файл рабочего пространства. Если это произошло, удалите нарушенный файл рабочего пространства – это даст возможность открыть проект с заданными по умолчанию параметрами настройки рабочего пространства.

1.4. Библиотеки Active-HDL включают библиотеки системы и библиотеки проектов. Библиотеки системы поставляются с программным обеспечением Active HDL и записаны в папке *VLIB*. Они имеют опции Read-only, установленные по умолчанию. Библиотеки проектов создаются в корневой папке проекта.

Библиотеки системы, записанные в подкаталоге *VLIB* содержат встроенные стандартные библиотеки (STD, IEEE), пре компилированные стандартные библиотеки (LPM, ALDEC, ALTERA\_MF, SIMPRIM и др.), в которых содержатся стандартные блоки, библиотеки производителей ИС, библиотеки синтеза (EXEMPLAR, SYNOPSIS, SYNPLIFY) и ряд вспомогательных библиотек. Модели всех базовых вентилях (gates) содержатся в библиотеке SIMPRIM.

## 2. СРЕДСТВА ПРОЕКТИРОВАНИЯ В ACTIVE HDL

2.1. Active HDL представляет комплекс программных средств, ориентированный на широкое применение HDL-языков описания проектируемых устройств и в сочетании с графическими средствами предоставляет пользователю выполнять все стадии проектирования от описания аппаратной части через синтез и отладку входящих в проект устройств до моделирования проекта.

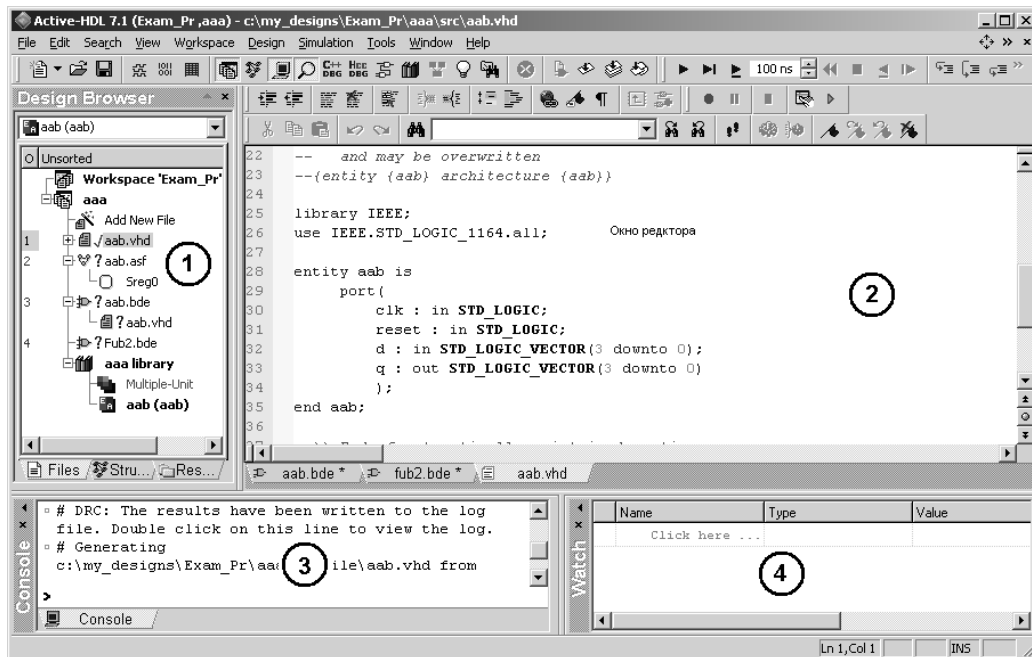


Рис.2.1

На рис. 2.1 показан интерфейс Active HDL в режиме ввода информации о проектируемом устройстве на языке VHDL. Его основными элементами являются окно обзора проекта Design Browser (1), рабочее окно редактора (2), окно Console (3) и окно Watch (4). Набор окон и их вид может изменяться: набор окон определяется опциями, устанавливаемыми в разделе View главного меню. Размеры окон могут изменяться пользователем по его усмотрению. Окно Design Browser содержит три вкладки: Files, Structure и Resource. На вкладке Files отображается иерархическое дерево файлов проекта. Переключаясь по нему можно двойным щелчком левой кнопки мыши открыть требуемый файл. Перейдя на вкладку Structure (щелкнув по

соответствующему значку в нижней части окна) можно просмотреть структуру этого файла: используемые библиотеки модулей, перечень и тип входов и выходов и т.д. Перейдя на вкладку Resource можно просмотреть ресурсы, которые обеспечивают работу проекта.

Окно Console предназначено для вывода сообщений генерируемых инструментальными средствами Active HDL и ввода команд Active HDL. Окно Watch используется для наблюдения за объектами программы при отладке проекта. В нем размещают имена переменных и сигналов, которые необходимо контролировать в процессе отладки.

Для ввода информации о проектируемых устройствах в Active HDL используются следующие инструментальные средства:

**HDL Editor** – текстовый HDL-редактор, который включает Language Assistant - помощник по языку с набором шаблонов языка VHDL (Verilog);

**State Machine Editor** – графический редактор автоматов с конечными состояниями - инструмент для создания цифровых схем, которые могут быть представлены в форме FSM (finite state machine – дискретных устройств с конечным множеством состояний, которые в отечественной литературе называют конечными автоматами) схем; описание устройства в форме FSM Active HDL автоматически преобразует в соответствующие HDL коды;

**Block Diagram Editor - редактор** блок-схем - предназначен для ввода информации о проектируемых устройствах в графической форме (в виде электрических схем); его удобно использовать для построения проектов иерархической структуры, что повышает наглядность представления проекта на верхних уровнях иерархии и является альтернативой непосредственному описанию иерархии в HDL кодах.

**2.2. HDL Editor** - редактор для ввода VHDL кода проектируемого устройства, ввода кода файлов испытательного стенда (Test

bench), а также других текстовых файлов. В Active HDL реализованы автоматизированные методы генерации VHDL кода, например New Source File Wizard – “мастер” нового исходного файла.

*Примечание.* В англоязычном компьютерном жаргоне Wizard - программа-разработчик, генерирующая программные элементы с заданными свойствами; ее принято называть “мастер”.

Окно редактора показано на рис. 2.1. Основные параметры редактора можно устанавливать в окне Preferences (команда Tools| Preferences в главном меню). Ключевые слова, комментарии и константы в HDL Editor отличаются от остальной части текста по цветам.

Новые VHDL-файлы в Active-HDL можно создавать, открыв с помощью команды Add New File в окне Design Browser, окно Add New File (рис.2.2) и выбрав один из следующих вариантов:

- VHDL Source Code (Исходный текст VHDL);
- Block Diagram (блок-схема);
- State Diagram (автомат с конечными состояниями);
- SystemC Source Code (исходный текст SystemC);
- Verilog Source Code (исходный текст Verilog);
- Add Existing File (добавить существующий файл).

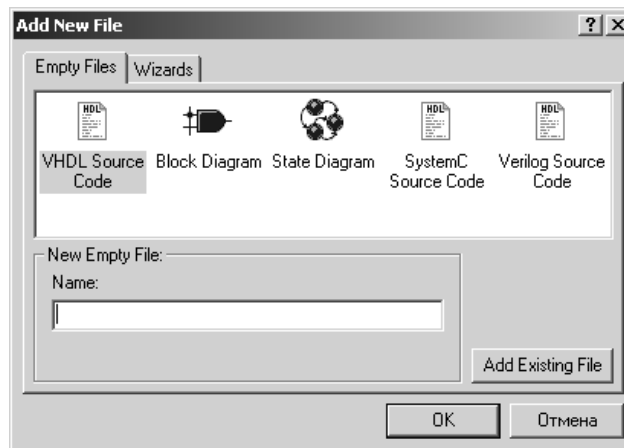
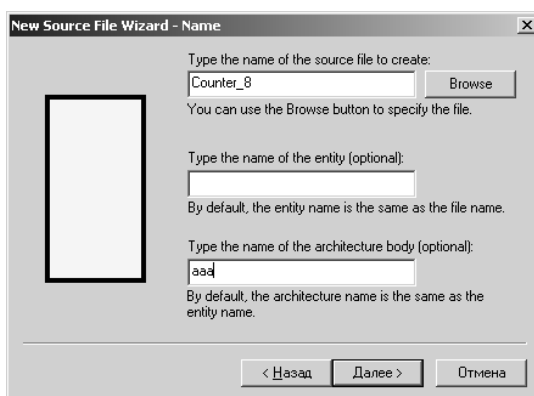


Рис. 2.2

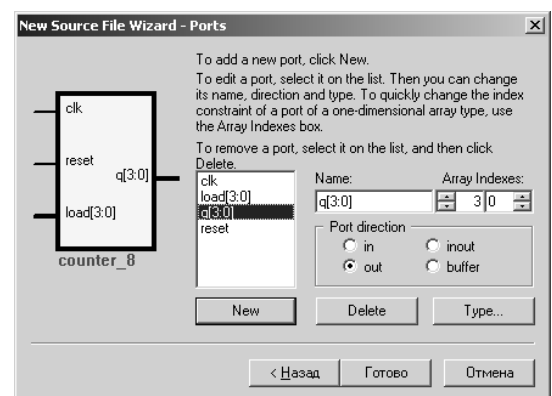
Для добавления нового файла в проект можно использовать и альтернативный вариант: вызвав через главное меню командой File|New|VHDL Source (Verilog Source, SystemC Source и т.д.) соот-

ветствующего “мастера” и ответив в интерактивном режиме на его вопросы.

“Мастера” облегчают ввод информации о проектируемом устройстве. Сначала необходимо указать, должен ли создаваемый код быть добавлен к текущему проекту, затем ввести имя файла для шаблона и (не обязательно) названия проектируемого объекта (entity) и варианта его реализации (architecture) (рис 2.3, а). Затем вводится информация о портах объекта (рис 2.3, б).



а)



б)

Рис. 2.3

Сгенерированный “мастером” VHDL-файл представляет собой заготовку файла VHDL-программы со всеми указанными пользователем и вставленными именами и типами портов. “Мастер” устанавливает прямую связь с Language Assistant (помощником по языку) и пользователь может выбирать желательный шаблон и скопировать его в свой или в исходный файл окна Language Assistant, что ускоряет написание VHDL-кода и сокращает число ошибок.

Language Assistant (помощник по языку) позволяет ускорить разработку исходного текста на VHDL. Это обеспечивается рядом шаблонов с подготовленными сегментами кода. Имеются несколько групп шаблонов:

**Code Auto Complete (автовод кода)** - завершает ключевые слова VHDL при печатании;

**Language templates (шаблоны языка)** - основные конструкции языка;

**Simulation templates (шаблоны моделирования)** - типовые процессы, полезные при создании тестовых панелей (Test Benches), типы сигналов синхронизации, доступ к файлам, и т.д.

**MatLab Interface (интерфейс с MatLab)** - синтаксические конструкции для конвертации VHDL-файла в m-файлы)

**Synthesis templates (шаблоны синтеза)** - ориентируемая на синтез реализация основных функциональных блоков, типа мультиплексоров, триггеров, счетчиков, и т.д.

**Tutorial, Training (обучающие программы)** с четырьмя шаблонами, которые включают счетчик, декодер, тестовую панель (test bench) и архитектуру высшего уровня.

**User templates (шаблоны пользователя)** - этот раздел предназначен для сохранения создаваемых пользователем шаблонов.

Language Assistant (рис.2.4) можно вызвать либо с панели инструментов либо из главного меню командой Tools|Language Assistant либо, вызвав щелчком правой кнопки мыши по полю редактора всплывающее меню и выбрав в нем соответствующую команду. Для размещения в файле проекта нужного шаблона необходимо щелчком левой кнопки мыши по полю редактора указать место, куда будет вставляться требуемый шаблон, затем найти его в левом окне Language Assistant, щелкнуть по нему правой кнопкой мыши и во всплывающем меню выбрать пункт Use (использовать, применить). Альтернативным вариантом является стандартная технология копирования кода шаблона из правого окна через буфер обмена.

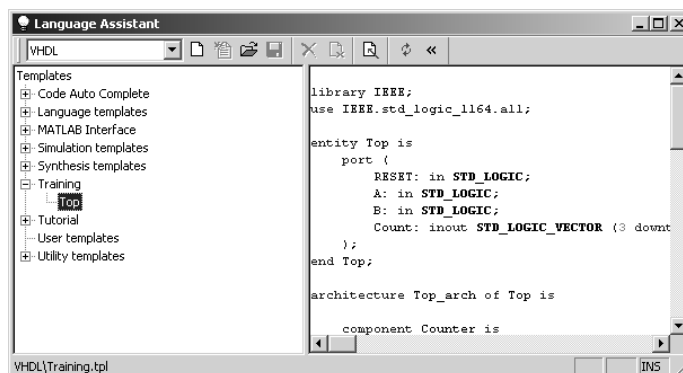


Рис.2.4

Для облегчения написания кода в HDL Editor реализована технология keywords auto-complete (автоматическое завершение ключевых слов) – при наборе начальных символов ключевого слова система выводит его полностью и ввод можно завершить нажатием клавиши Enter. Аналогично после набора имени шаблона нажатием клавиш Ctrl+Enter HDL-редактор вставляет шаблон из окна Language Assistant.

В процессе компиляции проекта HDL редактор сигнализирует об ошибках в HDL-файле. Все сообщения об ошибках отображаются в HDL-файле и одновременно в окне Compile.

Форматирование кода, улучшает его удобочитаемость. Панель форматирования текста показана на рис 2.5.

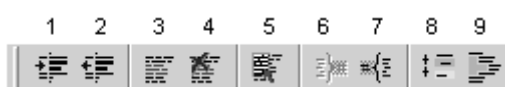


Рис. 2.5

Она содержит следующие инструменты (в скобках указан клавиатурный эквивалент команды):

- 1 – Indent (Tab) – сделать отступ выделенного текста;
- 2 – Outdent (Shift+ Tab) – удалить отступ выделенного текста;
- 3 – Comment (Ctrl+K) – преобразовать выбранный VHDL-текст в комментарий;
- 4 – Uncomment (Ctrl+ Shift+K) – преобразовать комментарий в VHDL-текст;

5 – Column selecting (Alt+C) – включить режим выделения по колонкам;

6 – Create group – сгруппировать выбранный текст для улучшения разборчивости кода. Созданная группа выделяется темным фоном и слева от нее появляется знак  $\oplus$ ;

7 – Remove groups – отменить создание всех групп;

8 – Generate structure – структурировать текст программы (раскрасить ее структуры). Для отмены раскраски ввести команду Remove groups.

9 – Autoformat text – автоматическое форматирование HDL-текста (установить отступы в соответствии со структурой операторов HDL-языка).

**2.3. State Machine Editor** - редактор состояний автоматов допускает графический ввод информации в проект в форме конечных автоматов. Конечный автомат (FSM) – устройство, которое характеризуется некоторым конечным количеством состояний и правилами перехода из одного состояния в другое. В форме конечных автоматов удобно представлять устройства управления и сложные цифровые устройства, обладающие внутренней памятью.

FSM-проект характеризуется следующим набором атрибутов:

- перечень состояний;
- перечень команд, которые могут быть выполнены во время переходов из одного состояния в другое;
- перечень действий, которые должны выполняться по каждой команде. Указанные действия могут быть следующими:
  - послать команду другому компоненту (асинхронно и синхронно);
  - выполнить определенную часть кода (пользовательская процедура);
  - установить определенное состояние FSM;
  - список условий для каждого состояния, которые вызывают описанные действия. Условия остаются активными до тех пор, пока



не возникнут изменения в состояниях перехода или когда состояние изменяется.

Для ввода информации о FSM-проекте на поле редактора необходимо разместить требуемые компоненты из набора примитивов конечного автомата и задать их параметры. В редакторе имеется достаточно обширный набор примитивов:

- State – состояние;
- Junction - соединение
- Transition – переход;
- Condition – условие
- Reset - сброс состояния;
- Entry action - действие при входе;
- Transition action - действие при переходе;
- State action - действие в состоянии;
- Exit action - действие при выходе.

Кроме этого в набор примитивов включаются такие традиционные компоненты HDL-языков как сигналы, переменные, константы, порты ввода-вывода и т.д. С полным набором примитивов можно ознакомиться в разделе FSM главного меню редактора состояний автоматов.

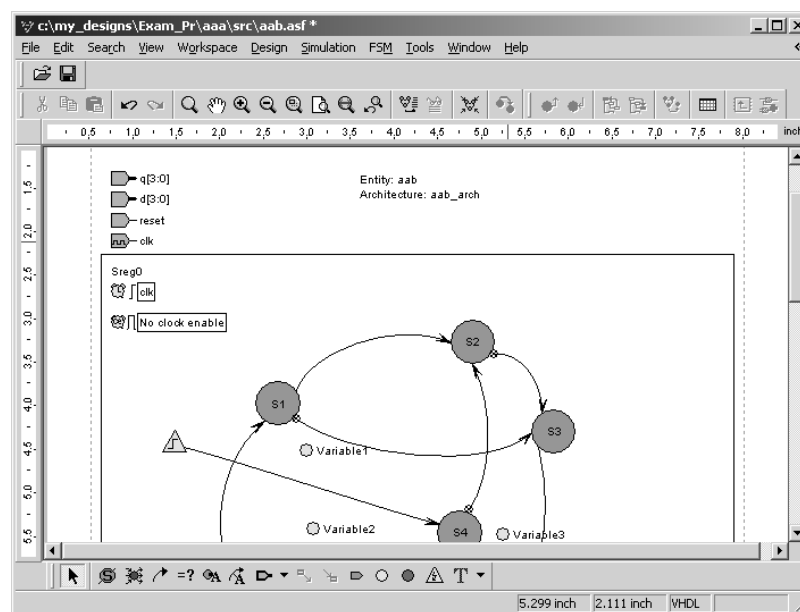


Рис. 2.6

Добавить в проект FSM-компонент можно двумя путями:

- 1) открыв с помощью команды Add New File в окне Design Browser, окно Add New File (рис.2.2) и выбрав вариант State Diagram;
- 2) вызвав через главное меню командой File|New| State Diagram “мастера” State Diagram и ответив в интерактивном режиме на его вопросы.

Окно редактора State Machine Editor показано на рис. 2.6. Он содержит стандартный для графических приложений Windows набор инструментов. В нижней части окна размещена панель, на которой расположены наиболее часто применяемые FSM-примитивы.

State Machine Editor позволяет изменять форму графа состояния по усмотрению пользователя, изменяя их размеры его вершин и (или) формы стрелок перехода и перемещать их в желательном направлении. Чтобы изменять свойства автомата, необходимо щелкнуть правой кнопкой мыши по полю редактора и выбрать требуемый пункт из контекстного меню Properties (Свойства). В окне свойств можно выбирать тип синхронизации автомата и определить момент смены состояния автомата (по переднему или по заднему фронту сигнала). Можно устанавливать опции декодирования, состояния автомата по умолчанию, условий сброса и поведения автомата, когда условия перехода не выполняются.

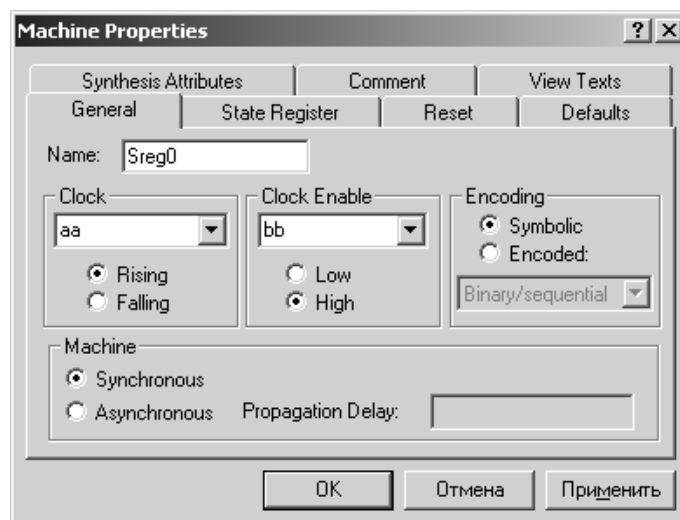


Рис. 2.7

Редактор может генерировать VHDL код для построенной диаграммы состояний конечного автомата. Чтобы сделать это, необходимо ввести команду опцию Generate HDL code через раздел FSM главного меню или, щелкнув по иконке Generate VHDL code на панели инструментов. Можно просмотреть сгенерированный VHDL код, выбрав пункт View code из FSM-меню. При необходимости в код можно добавить ссылки на библиотеки через пункт FSM-меню Set Generation code. Например, при выполнении математических действий можно добавить использование библиотеки IEEE.Std\_logic\_arith.all.

**2.4. Block Diagram Editor** (редактор блок-схем) - графический редактор для ввода информации о проекте в виде электрической схемы проектируемого устройства. Параллельно с вводом схемы устройства автоматически создается **VHDL**-код с DRC - контролем (design rule checking) за соблюдением проектных норм. Редактор блок-схем позволяет строить комбинированные проекты, в которых часть устройств проектируется с помощью HDL-редактора, часть с помощью редактора конечных автоматов, а объединять эти устройства на верхних уровнях иерархии можно с помощью редактора блок схем. Такой способ проектирования повышает наглядность проекта, делает его более “читабельным” для инженеров-электронщиков. Окно редактора блок схем показано на рис. 2.8.

Рабочее окно редактора расположено в центре. В нем с помощью инструментов редактора создается графическое изображение схемы проектируемого устройства. Для построения схемы могут использоваться графические символы стандартных логических вентилей (gates) типа И, ИЛИ, НЕ, Исключающее ИЛИ, И-НЕ, ИЛИ-НЕ и т. д., графические символы компонентов, созданных ранее в данном проекте, макромодели, фрагменты языка VHDL (переменные, константы, сигналы, некоторые операторы и т.д.), соединительные компоненты (провода и шины), порты ввода-вывода, графические и текстовые комментарии. Графические символы компонентов (Symbols)

вводятся с помощью окна Symbols Toolbox, остальные элементы схемы с помощью соответствующих инструментов размещенных на панели Diagram Items -VHDL, либо доступных через раздел Diagram главного меню.

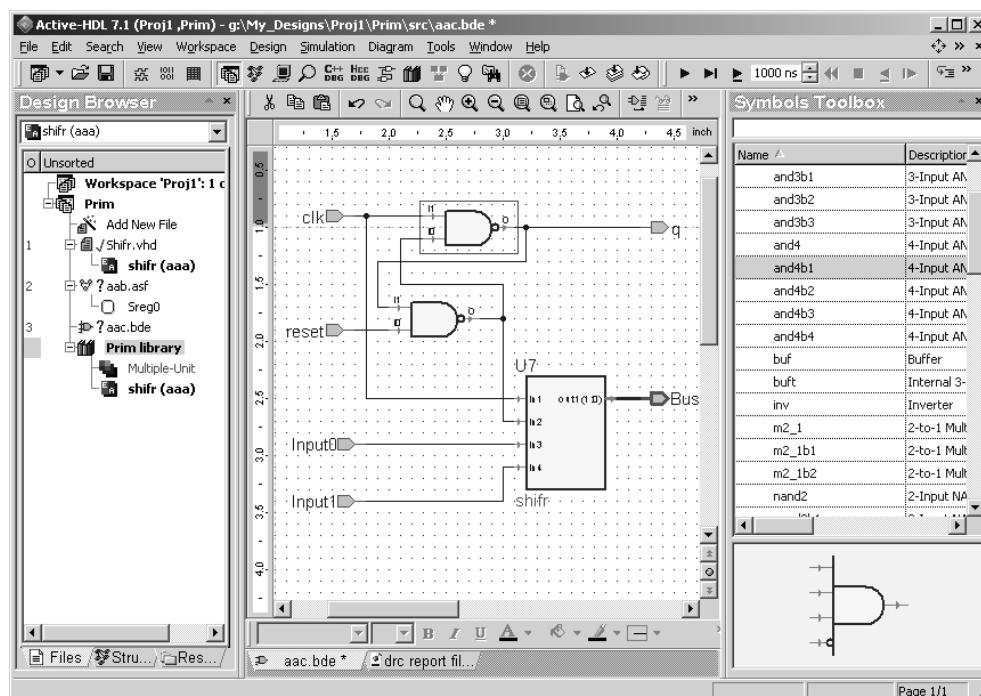


Рис. 2.8

Панель Diagram Items -VHDL с наиболее применяемыми инструментами расположена в верхней части окна (рис. 2.9).



Рис. 2.9

Select (Esc) – выделение объектов; перейти в этот режим из других режимов можно с помощью клавиши Esc;

Fub (F) – шаблон макромодели;

Process – макромодель оператора Process языка VHDL;

Show Symbols Toolbox (S)– показать/скрыть окно символов доступных компонентов Symbols Toolbox (рис. 2.8); в окне отображается набор символов стандартных вентилях (gate) и разработанных в проекте устройств в любом из редакторов; выделив требуемый символ компонента в этом окне, его можно перетащить на поле редактора и

использовать в проектируемой схеме; S – клавиша быстрого вызова этого окна, нажатие клавиши S при открытом окне скрывает его;

5, 6- Wire, Bus (W, B) – инструменты для соединения компонентов схемы с помощью проводов и шин и ввода их имен;

7- Input Terminal (I, O, Shift+I, Shift+O, 9, 0, Shift+9, Shift+0)– раскрывающийся список портов ввода-вывода (текст всплывающей подсказки и вид иконки зависят от вида выбранного в списке элемента);

8- Design Unit Header - раскрывающийся список конструкций языка VHDL (текст всплывающей подсказки и вид иконки зависят от вида выбранного в списке элемента);

9- VCC Symbol (P, G)- раскрывающийся список для размещения на поле чертежа символов питания (VCC) или земли (Ground);

10- Global Wire - раскрывающийся список для объявления глобальных сигналов и шин;

11- Attach Text (N) – инструмент для присоединения определенных текстовых конструкций к графическим символам (например, вывода названия символа, его обозначения в схеме и т.д.);

12- Line - раскрывающийся список для размещения на поле чертежа графических примитивов и текста, которые несут не функциональные, а представительские функции и вводятся обычно для улучшения “читабельности” проекта.

Вызов наиболее употребительных инструментов дублирован клавишами клавиатуры, наименования которых в приведенном списке указаны в скобках после названия инструмента. С помощью клавиш удобно переходить от одного инструмента к другому.

Подобно рассмотренным выше редакторам HDL Editor и State Machine Editor Block Diagram Editor позволяет генерировать VHDL-код проектируемого устройства (команда Diagram|Generate HDL code главного меню либо кнопка Generate HDL code на панели инструментов BDE Edit).

VHDL-код компонента, отображаемого на схеме соответствующим символом, можно редактировать. Для этого необходимо выде-

литель требуемый символ и ввести команду Push из контекстного меню. Альтернативный вариант доступа к VHDL-коду компонента – двойной щелчок по нему левой кнопкой мыши.

Как и большинство схемотехнических графических редакторов Block Diagram Editor позволяет размещать, вращать и упорядочивать графические изображения компонентов. Доступ к этим операциям осуществляется через команды Align, Mirror, Rotate, Order всплывающего меню, которое открывается щелчком правой кнопки мыши по требуемому компоненту или выделенной группе компонентов.

В редакторе блок-схем можно создавать макромодели двух типов:

Fub – макромодель части проекта, для которой не создается графическое изображение (символ компонента) и которая по этой причине не будет доступна для редактора блок-схем на верхних уровнях иерархии; такие макромодели создаются для улучшения логической структурированности проекта и если их не предполагается использовать повторно; внутреннее содержание Fub может представляться в любой из допустимых в Active HDL форм (на HDL-языке, в виде электрической схемы или диаграммы состояний конечного автомата);

Symbol - макромодель части проекта, для которой такое графическое изображение создается и которая будет доступна для редактора блок-схем на верхних уровнях иерархии.

Созданная в графическом редакторе макромодель (Fub) может быть преобразована в графический символ с помощью команды контекстного меню Create Fub to Symbol, которое вызывается щелчком правой кнопки мыши по графическому изображению макромодели Fub.

2.5. Анализируя средства ввода информации о проектируемых устройствах, отметим, что в системе Active HDL реализована концепция построения единой среды проектирования, не зависящей от того, какими средствами пользуется проектировщик для ввода ин-

формации о проекте. В качестве связующего элемента, объединяющего разнородные проекты, выбирается один из HDL-языков. Это обеспечивает возможность включения в проект компонентов, разработанных в других проектах и переносимость проектов из одной среды проектирования в другую, с одной платформы проектирования на другую.

Наличие трех вариантов редакторов позволяет объединять в одном проекте разные принципы описания проектируемого устройства и его компонентов. HDL-языки удобно применять на нижних уровнях иерархии для проектирования компонентов на основе алгоритмов их работы (поведения), не привязываясь к конкретному способу реализации компонента. Представление устройства в форме конечного автомата удобно применять при проектировании сложных компонентов с внутренней памятью. Конечные автоматы в большой степени упрощают проектирование цепей управления. Поток управления в них представляется в форме, подобной человеческому мышлению. Вследствие этого анализ их работы не вызывает затруднений. Кроме того, State Machine Editor автоматически генерирует HDL-код, который точно описывает поведение конечного автомата, что позволяет исключать любые непредвиденные состояния системы управления. Наконец, представление устройства в форме блок-схемы целесообразно применять на верхних уровнях иерархии проекта для повышения удобочитаемости проекта для инженеров, возможности применения библиотек макромоделей и при привязке реализации проекта к конкретному типу ПЛИС. Это дает возможность проектировщику более наглядно распределять ресурсы ПЛИС между компонентами проекта.

### 3. ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ В ACTIVE HDL 7.1

#### 3.1. Основные принципы проектирования цифровых устройств в Active HDL

Система Active HDL не является системой, позволяющей выполнить полный цикл проектирования от ввода проекта до программирования ПЛИС. Ее скорее можно рассматривать как хорошо развитую и удобную систему подготовки и испытания проекта. В ней можно выполнять следующие проектные операции:

- 1) ввод информации о входных и выходных сигналах и логике работы проектируемой схемы;
- 2) моделирование (симуляция) работы проектируемой схемы и анализ временных соотношений между сигналами в ней;
- 3) формирование файлов для испытания проекта в составе других систем (MatLab, Simulink);
- 4) формирование файлов для выполнения завершающих стадий проектирования в других САПР.

Для выполнения двух первых операций Active HDL располагает весьма развитыми и удобными средствами, которые описывались выше, а ниже будет описана технология их выполнения. Формирование файлов для испытания работы спроектированного устройства в составе систем MatLab и Simulink осуществляется после выполнения компиляции любого из компонентов проекта одной командой, доступной в контекстном меню, если были успешно выполнены предшествующие проектные операции. В качестве средства связи с другими САПР в системе может использоваться один из HDL-языков (VHDL или Verilog), выбираемый по усмотрению пользователя.

Все компоненты Active VHDL объединены в единообразную графическую среду, являющуюся основным рабочим пространством



проекта - workspace. Workspace организует рабочую область окна проектирования и связывает воедино все элементы среды проектирования. По умолчанию все объекты Workspace помещаются в одну папку.

Design – это отдельные проекты в пределах одного Workspace. В процессе проектирования активным является только один Design, который можно быть назначать по усмотрению пользователя.

Технология выполнения указанных выше проектных операций достаточно традиционна: добавление нового файла в проект, ввод в него информации о проектируемом устройстве, его входных и выходных сигналах, тестирование устройства и его корректировка.

### 3.2. Технология проектирования на HDL-языках

После загрузки системы Active VHDL появится диалоговое окно Getting Started (рис. 3.1). Для начала нового проекта необходимо выбрать опцию Create new workspace, для продолжения существующего – Open existing workspace и выделить требуемое название из списка проектов.

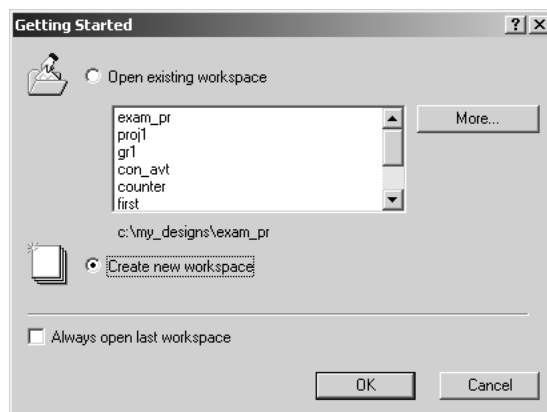


Рис. 3.1

После нажатия кнопки OK, окно Getting Started закрывается и открывается окно “мастера” создания нового проекта New Design Wizard (рис. 3.2)

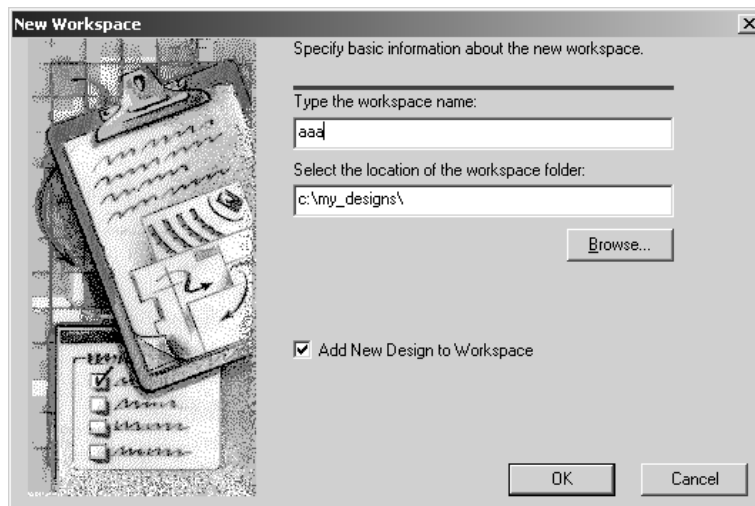


Рис. 3.2

В первом окне мастера необходимо ввести имя проекта и выбрать папку, в которой он будет размещен. После нажатия кнопки ОК открывается второе окно “мастера” New Design Wizard – окно ресурсов проекта (рис. 3.3), в котором необходимо выбрать один из вариантов продолжения диалога:

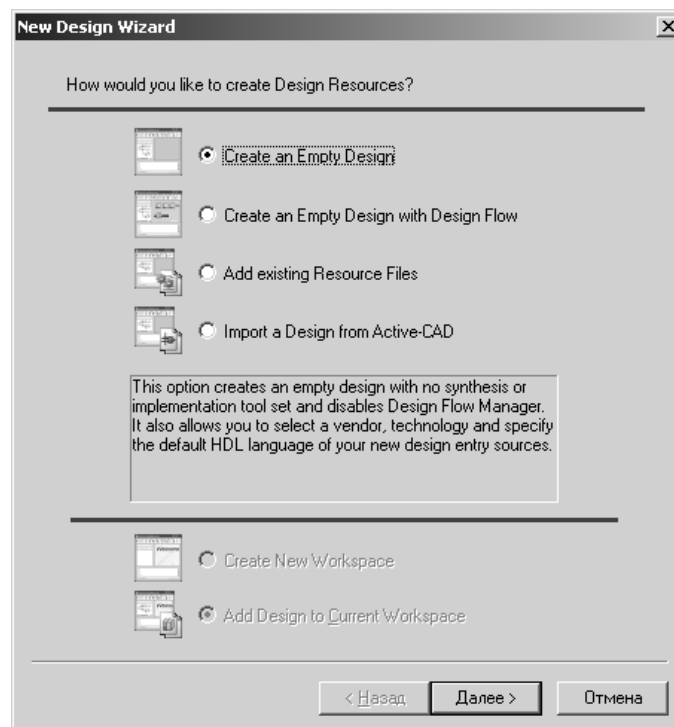


Рис.3.3

**Create an Empty Design** создать пустой проект;

**Create an Empty Design with Design Flow** создать пустой проект с оперативным добавлением новых файлов в него (это дает возможность сразу создать новые файлы, содержащие необходимые компоненты проекта);

**Add existing Resources Files** - добавить существующие файлы (в создаваемый проект могут быть импортированы существующие VHDL - файлы);

**Import design from Active-CAD** импортировать проект из Active-CAD - в этом случае сгенерированный в Active-CAD список соединений будет импортирован в проект.

В нижней части окна помещены варианты ответов при использовании “мастера” для продолжения ранее начатого проекта.

Выбрав Create an Empty Design и нажав кнопку Далее >, переходим к третьему окну “мастера”(рис. 3.4), в котором необходимо указать язык, на котором будут генерироваться файлы проекта для редактора блок схем и выбираемый по умолчанию HDL-язык для всех файлов проекта. В полях Vendor и Technology при необходимости указываются производитель ПЛИС и семейство ПЛИС, на которых будет реализовываться проект.

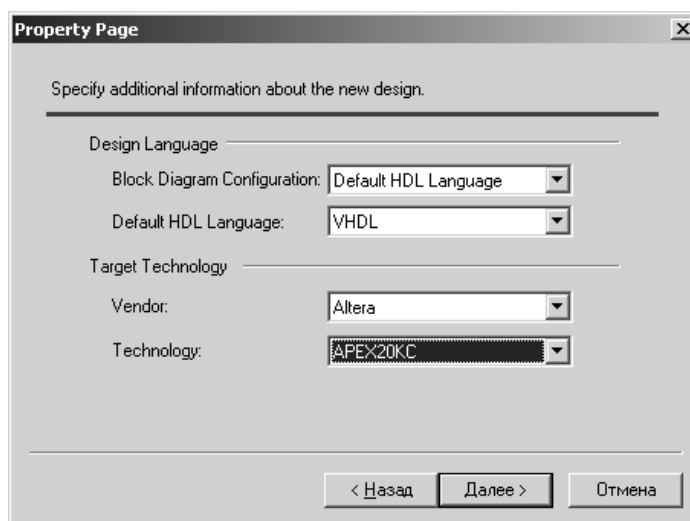


Рис.3.4.

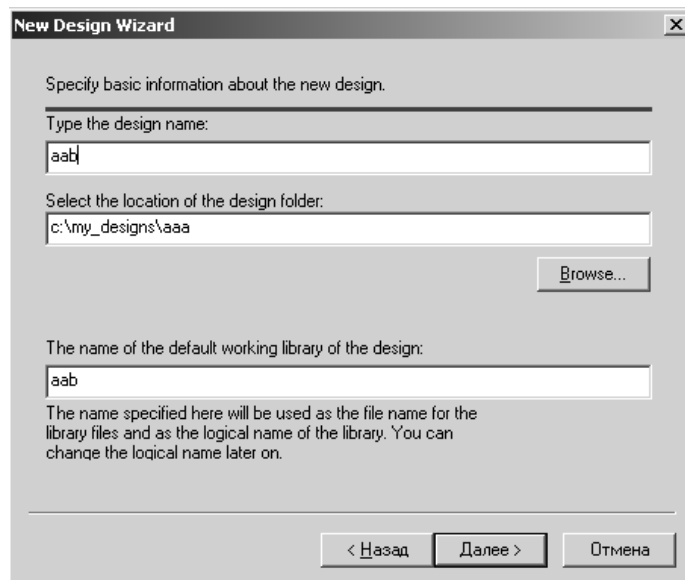


Рис.3.5

В следующем окне “мастера” (рис.3.5) необходимо указать имя создаваемого компонента проекта (design name) и имя рабочей библиотеки (working library) где будут размещаться файлы ресурсов этого компонента (по умолчанию эти имена совпадают).

Далее в открывшемся информационном окне подтверждаем правильность сделанных шагов или возвращаемся назад и корректируем их, после чего будет открыто основное окно интерфейса (рис. 3.6).

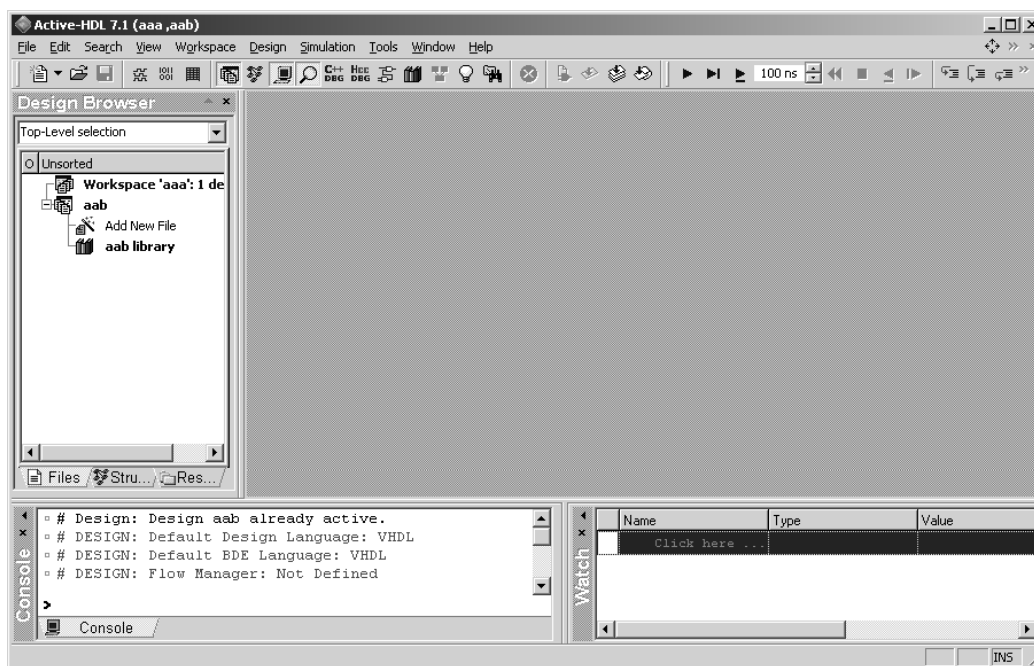


Рис. 3.6

Далее, щелкнув по ветви Add New File в окне Design Browser, откроем одноименное окно (рис.3.7) с двумя вкладками: Empty Files и Wizards.

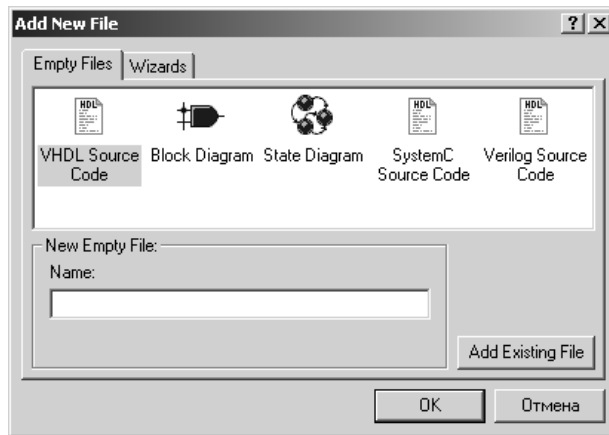


Рис. 3.7

Это окно предоставляет три варианта добавления нового файла в проект:

добавить пустой файл (Empty Files), предварительно указав его имя в поле Name и выбрав щелчком по соответствующей иконке его тип;

добавить существующий файл (кнопка Add Existing File);

добавить шаблон нового файла соответствующего типа с помощью “мастера” Wizard.

В первом варианте в проект добавляется пустой файл, и пользователь вводит в него информацию о новом компоненте проекта вручную, используя те или иные инструменты Active HDL. Во втором варианте Active HDL открывает стандартное окно выбора файлов, и проектировщик указывает с его помощью путь к требуемому файлу. В третьем варианте с помощью Wizard в интерактивном режиме создается заготовка нового файла, в которую Wizard сам добавит некоторые программные или графические конструкции с нужными пользователю свойствами.

Рассмотрим технологию добавления VHDL-файла в проект на примере проектирования двоичного четырехразрядного счетчика.



Рис. 3.8

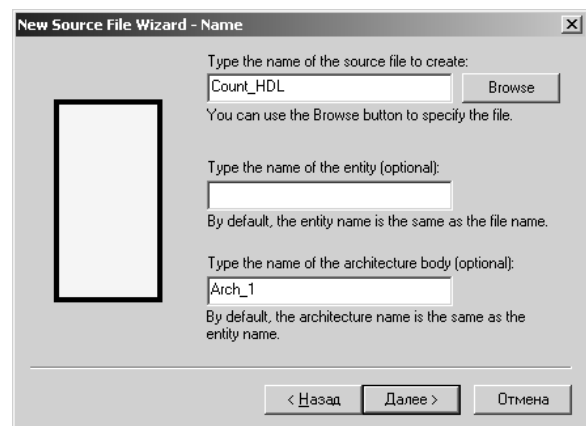


Рис. 3.9

Воспользуемся третьим вариантом добавления файла в проект. Перейдем в окне Add New File на вкладку Wizards и дважды щелкнем по иконке VHDL Source Code, чтобы добавить в проект файл на языке VHDL. В открывшемся первом окне “мастера” добавления новых файлов (рис.3.8) нажмем кнопку Далее > , после чего откроется второе окно мастера (рис.3.9).

Это окно предназначено для ввода имени файла - верхнее поле, имени объекта (entity) проектирования - среднее поле и имени варианта построения (architecture) проектируемого объекта – нижнее поле. Обязательным является заполнение только верхнего поля. Два других поля заполняются по желанию пользователя.

Введем в верхнее поле название файла проектируемого устройства (Count\_HDL), в нижнее – название варианта архитектуры проектируемого устройства Arch\_1, среднее поле оставим пустым и нажмем кнопку Далее >. В результате этих действий откроется окно “мастера” портов Wizard – Ports (рис.3.10).

Это окно предназначено для ввода информации о портах устройства – их названиях (Name), размерностях (Array index), видах (Port direction) и типах сигналов в них (Type).

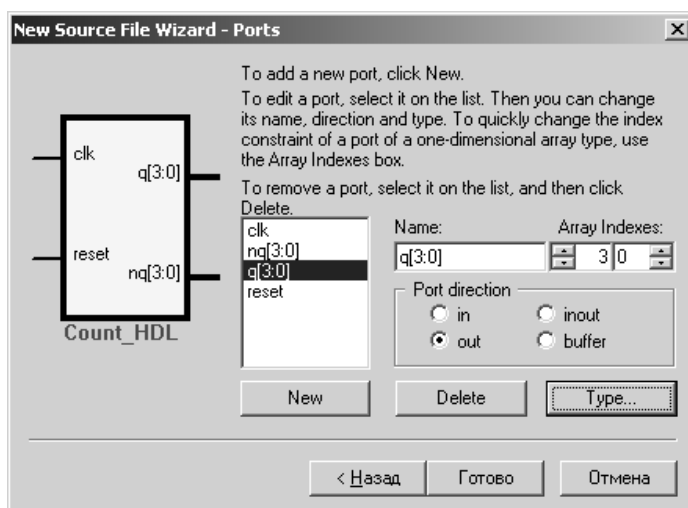


Рис. 3.10

Добавим в проектируемое устройство четыре порта:

clk -входной порт для тактовых импульсов, тип сигнала std\_logic;

reset - входной порт для сброса счетчика, тип сигнала std\_logic;

q[3:0] - выходной порт – 4-х разрядная шина размерностью [3:0], тип сигнала std\_logic\_vector (выходы всех разрядов счетчика);

nq[3:0] - выходной порт – 4-х разрядная шина размерностью [3:0], тип сигнала std\_logic\_vector (инверсные выходы счетчика);

Для добавления нового порта в устройство необходимо нажать кнопку New, ввести в поле Name имя порта, далее указать его вид (Port direction) и размерность (Array index), затем, нажав кнопку

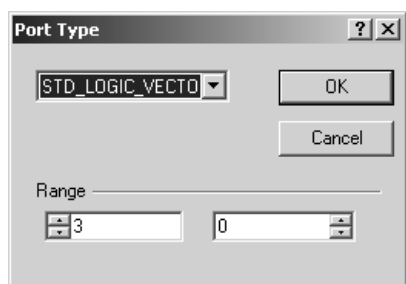


Рис. 3.11

Type..., в открывшемся окне Port Type (рис. 3.11) выбрать из раскрывающегося списка тип сигнала передаваемого через порт и при необходимости откорректировать его размерность (Range). После нажатия кнопки ОК возвращаемся в основное окно

Wizard – Ports. Если проводилась корректировка размерности сигналов в порте, то соответствующие изменения автоматически произойдут полях Array index. На любом шаге работы “мастера” можно вернуться назад и скорректировать введенные данные.

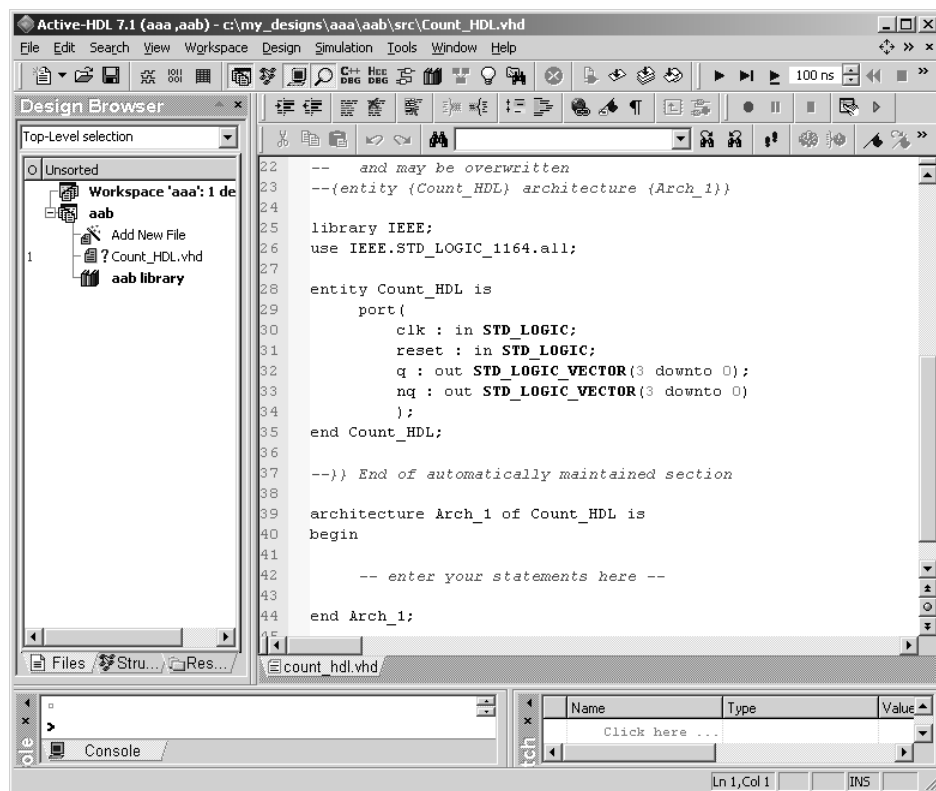


Рис. 3.12

Нажатием кнопки Готово работа “мастера” завершается, открывается HDL-редактор и в его окно выводится заготовка VHDL-файла с созданными “мастером” готовыми программными элементами (рис. 3.12).

### 3.3. Редактирование кода VHDL-файла

3.3.1 HDL редактор - это текстовый редактор, в котором поддерживается стандартная для Windows- приложений технология работы с файлами. Его дополнительными сервисными функциями являются распознавание и окраска ключевых слов используемого HDL-языка, возможность группировки, свертыванием и развертыванием при необходимости выделенных HDL конструкций, чтобы улучшить читаемость кода, а также возможность применения языкового помощника (Language Assistant) для редактирования и ввода текста HDL-кода.



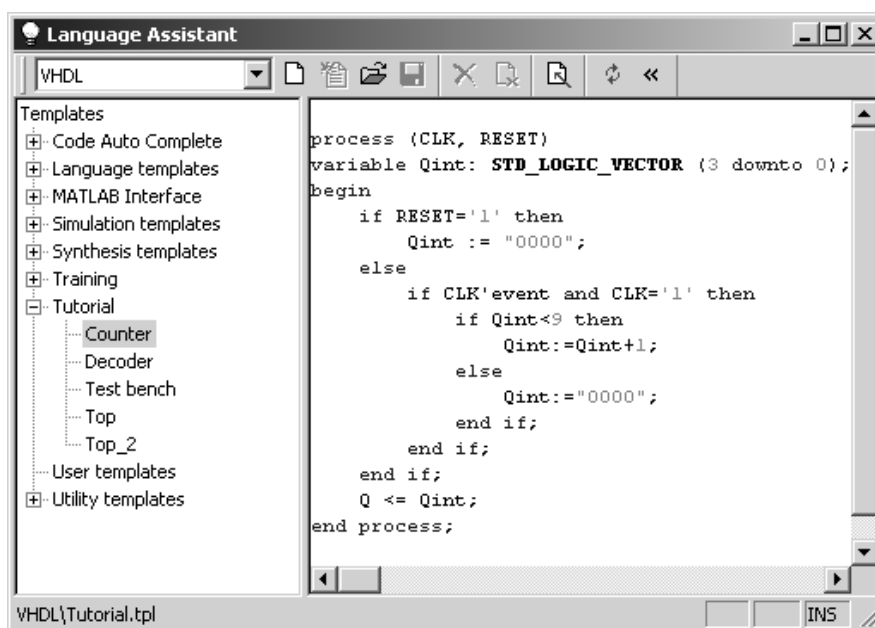


Рис. 3.13

Откроем Language Assistant (команда Tools | Language Assistant в главном меню) и воспользуемся шаблоном кода для проектируемого счетчика из раздела Tutorial (рис. 3.13). Для переноса кода шаблона укажем в окне HDL-редактора (рис. 3.11) место, куда необходимо вставить код (в секции architecture щелкнем левой кнопкой мыши в начале строки 42 -- enter your statements here -- ), затем перейдем в окно Language Assistant (рис. 3.12), щелкнем правой кнопкой мыши по пункту Counter в его левом окне и в открывшемся контекстном меню выберем команду Use. В результате этого текст шаблона, выведенного в правом окне, будет скопирован в указанное ранее место создаваемого VHDL-файла (рис. 3.14).

Теперь добавим ссылку на библиотеку IEEE.STD\_LOGIC\_UNSIGNED. Укажем в редактируемом VHDL-файле место, где должна размещаться эта ссылка, затем откроем Language Assistant, найдем в левом окне раздел Language templates, а в нем подраздел library, packages, раскроем его и выделим требуемую библиотеку (рис. 3.15). Щелкнув по выделенной библиотеке правой кнопкой мыши, выберем из контекстного меню команду Use – приведенный в правом окне текст ссылки на библиотеку будет вставлен в файл (рис. 3.16).

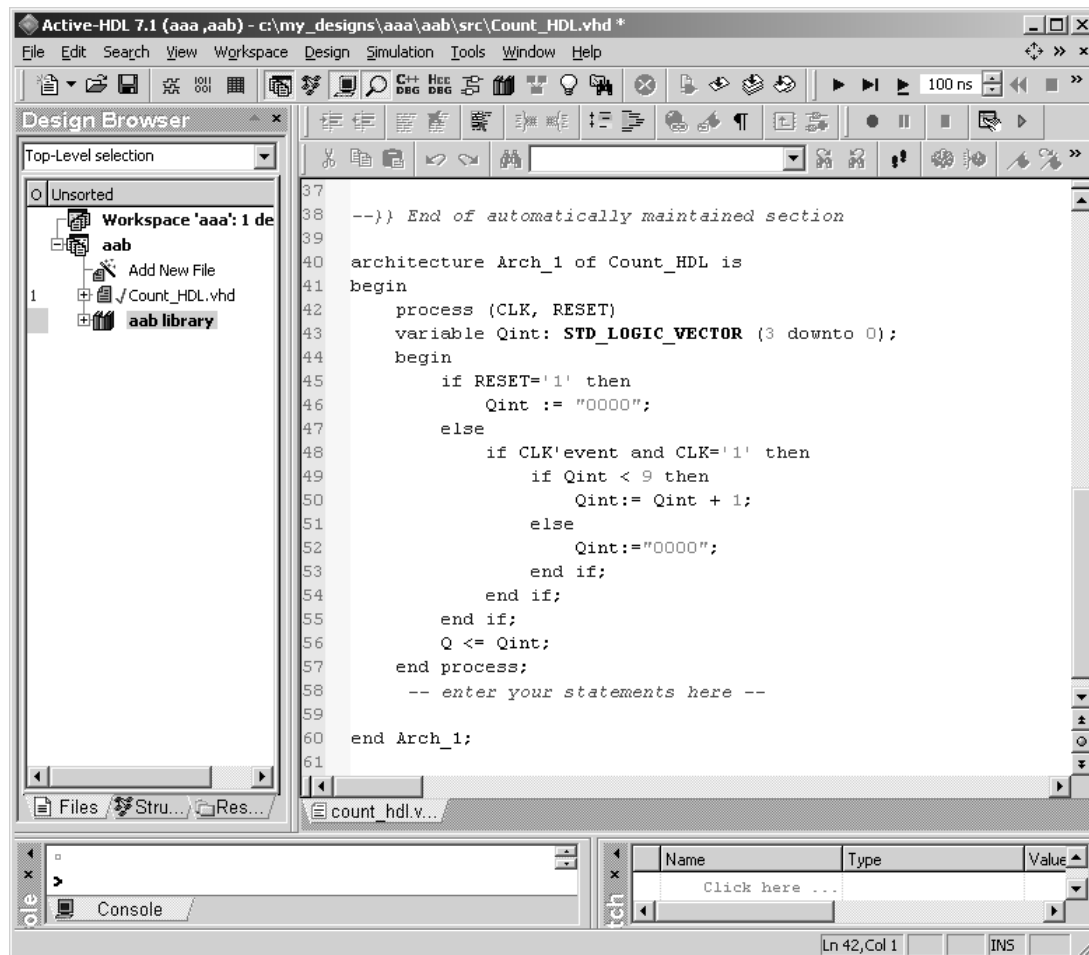


Рис. 3.14

Теперь добавим ссылку на библиотеку IEEE.STD\_LOGIC\_UNSIGNED.

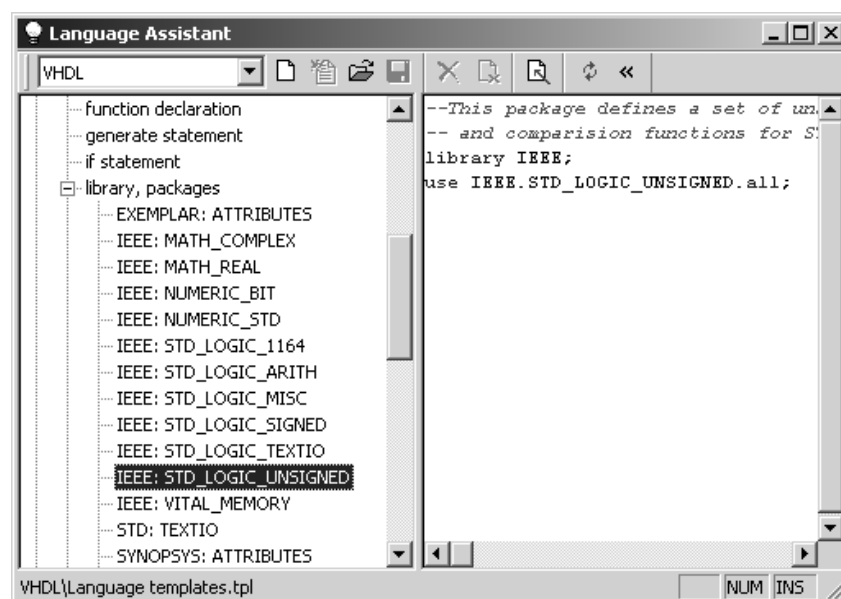


Рис. 3.15

Укажем в редактируемом VHDL-файле место, где должна размещаться эта ссылка, затем откроем Language Assistant, найдем в левом окне раздел Language templates, а в нем подраздел library, packages, раскроем его и выделим требуемую библиотеку (рис. 3.15). Щелкнув по выделенной библиотеке правой кнопкой мыши, выберем из контекстного меню команду Use – приведенный в правом окне текст ссылки на библиотеку будет вставлен в файл (рис. 3.16).

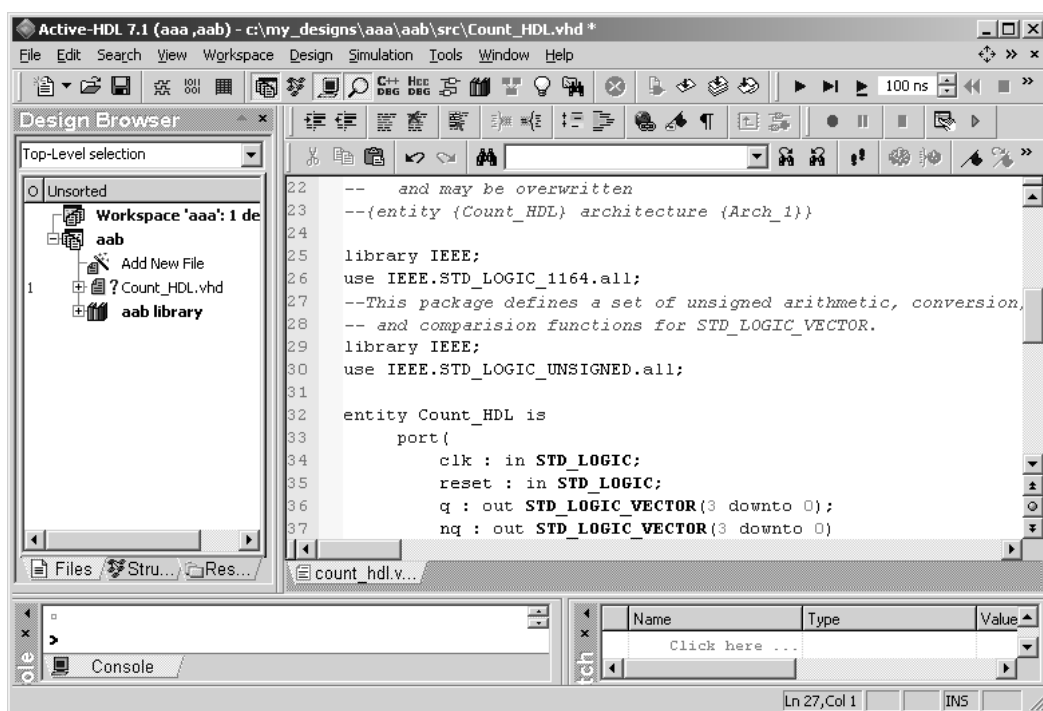


Рис. 3.16

Опишем инверсные выходы счетчика для чего в VHDL-файл в конце оператора process добавим строку “nq <= not Qint;” (рис. 3.17).

3.3.2. Выполним компиляцию VHDL-файла и его синтаксический контроль.

Команду для компиляции можно ввести несколькими способами:

- 1) щелкнув левой кнопкой мыши по полю HDL-редактора и нажав клавишу F11;
- 2) щелкнув правой кнопкой мыши по названию компилируемого файла в окне Design Browser и выбрав пункт Compile в контекстном меню;

- 3) вводом команды Design|Compile через главное меню;
- 4) нажатием кнопки Compile на панели инструментов.

Выполнив компиляцию, развернем иерархическое дерево проекта в окне Design Browser и заметим, что в нем после компиляции появились новые ветви count\_hdl (arch\_1), выделенные красным цветом.

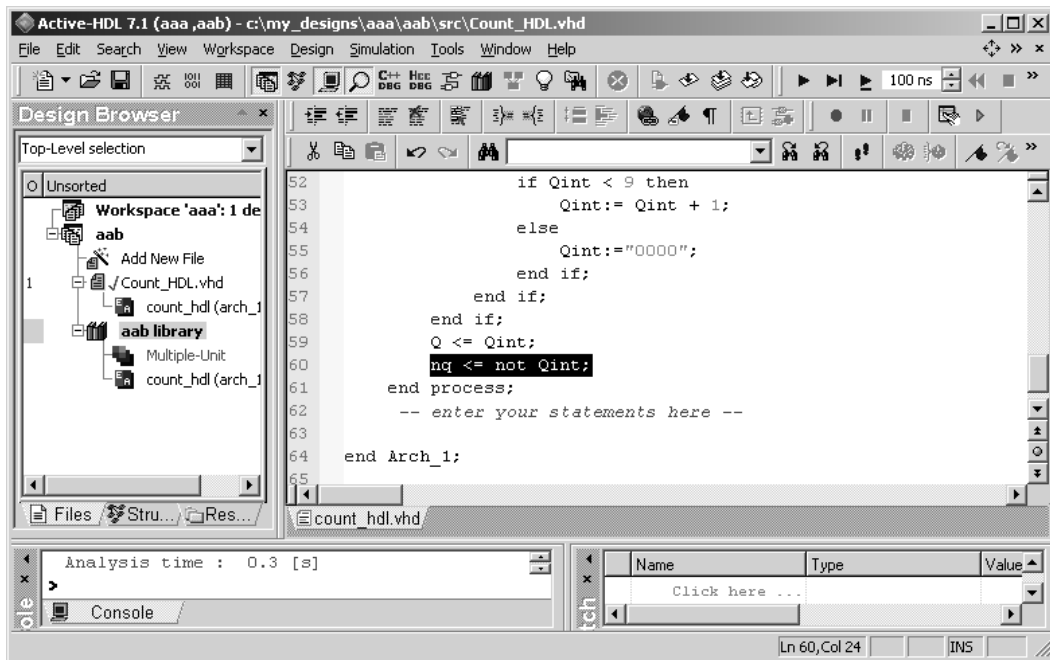


Рис. 3.17

Если в исходном файле допущены синтаксические ошибки, то процесс компиляции будет приостановлен, строки с ошибками в окне HDL-редактора будут подчеркнуты красными линиями, а напротив компилируемого файла в окне Design Browser появится знак ошибки (красный крест). Переместив курсор на строку с ошибкой, можно во всплывающем сообщении прочесть информацию об ошибке. Одновременно в окне Console будет выведен список ошибок, и до первой удачной компиляции новые файлы в иерархическом дереве не появятся. Проиллюстрируем это примером.

Увеличим размер окна Console. Для этого необходимо поместить курсор на линию, отделяющую это окно от окон Design Browser и HDL-редактора и после появления знака сплиттера нажать левую кнопку мыши и увеличить высоту окна до необходимых размеров.

Щелкнув правой кнопкой мыши в окне Console и выбрав во всплывающем меню пункт Clear log, удалим из него список с результатами предыдущих компиляций.

Перейдем в окно HDL-редактора найдем в нем строки раздела entity и в списке портов закомментируем строку с описанием одного из них, например clk. Это можно сделать либо выделив эту строку, затем щелкнуть по ней правой кнопкой мыши и выбрать команду Comment в контекстном меню (клавиатурный эквивалент Ctrl+K), либо просто ввести в начале строки признак комментария (два знака “минус” без пробелов между ними). Откомпилируем проект, перейдем в окне Console в начало списка ошибок и дважды щелкнем левой кнопкой мыши по его первой строке. Курсор в окне HDL-редактора переместится на строку, которая вызвала это сообщение, и слева от нее появится символ ошибки - красный крест (рис. 3.18). Перемещаясь по списку ошибок и выполняя над каждой его строкой указанные действия, можно определить, какой из строк HDL-файла вызвано то или иное сообщение об ошибке. Необходимо заметить, что не все ошибки, диагностируемые при компиляции как синтаксические, являются таковыми.

Иногда ошибки при компиляции возникают из-за того, что не подключена какая-либо из библиотек, конструкция которой используется в компилируемом файле. Характерным признаком такой ситуации является наличие большого количества сообщений в списке ошибок с указанием на недопустимость каких-либо операций над объектами программы, либо на то, что те или иные функции или объекты программы не известны.

Проиллюстрируем это примером. Закомментируем в компилируемом файле Count\_HDL.vhd строку со ссылкой на библиотеку (например, use IEEE.STD\_LOGIC\_UNSIGNED.all;) и откомпилируем файл. В окне Console будет выведен следующий список ошибок.

```
# Error: COMP96_0071: Count_HDL.vhd : (46, 9): Operator "<" is
not defined for such operands.
```

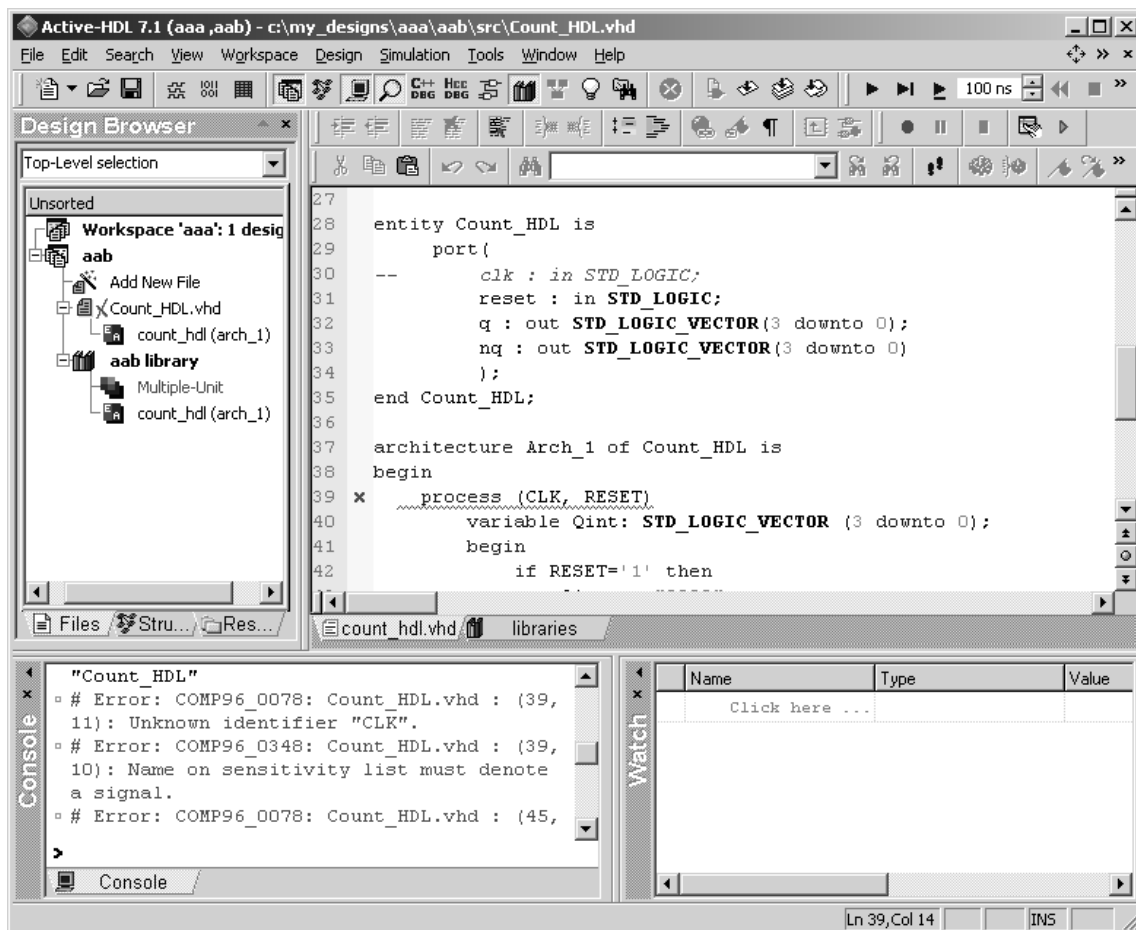


Рис. 3.18

# Error: COMP96\_0104: Count\_HDL.vhd : (46, 9): Undefined type of expression.

# Error: COMP96\_0098: Count\_HDL.vhd : (46, 9): Boolean type expected.

# Error: COMP96\_0077: Count\_HDL.vhd : (47, 14): Assignment target incompatible with right side. Expected type "std\_logic\_vector".

# Error: COMP96\_0071: Count\_HDL.vhd : (47, 14): Operator "+" is not defined for such operands.

# Error: COMP96\_0104: Count\_HDL.vhd : (47, 14): Undefined type of expression.

Большая часть сообщений в нем указывает на то, что не определены некоторые операции и функции, либо на недопустимость выполнения операций над операндами данных типов. Определение этих операций содержится в отключенной библиотеке. Уберем признак комментария со строки ссылки на указанную библиотеку и вновь от-

компилируем файл - убедимся, что компилятор не обнаружит в нем ошибок.

Следует иметь ввиду, что компилятор выявляет только синтаксические (формальные) ошибки. Вместе с тем в проекте могут содержаться алгоритмические ошибки смыслового характера. Диагностика таких ошибок наиболее трудна. Их можно выявить только в процессе моделирования работы устройства.

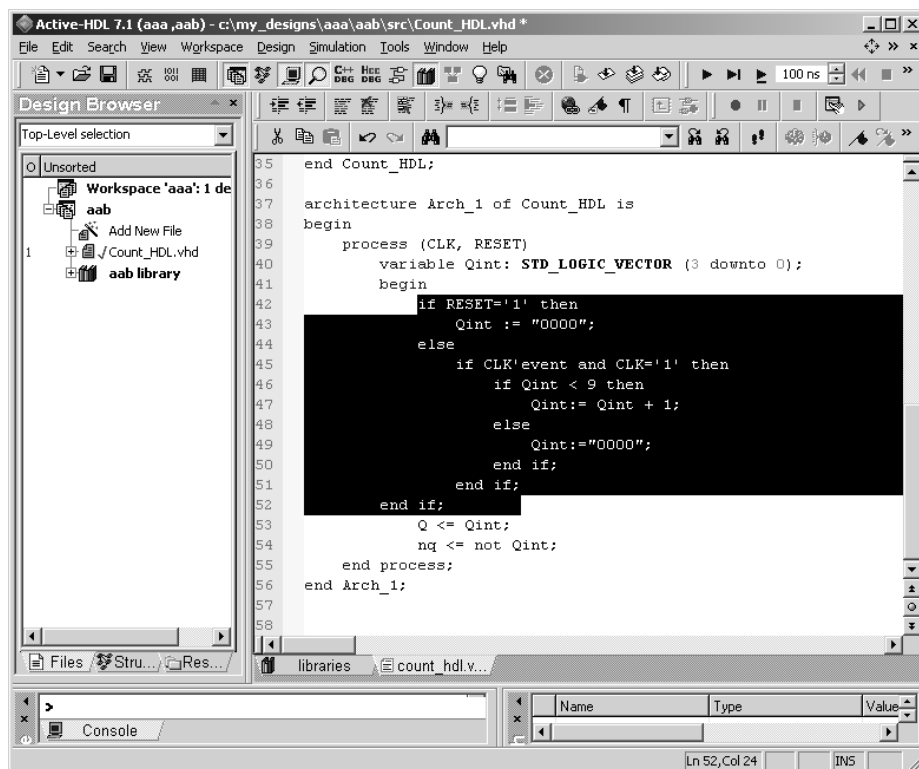


Рис. 3.19

После успешной компиляции проектируемого устройства HDL-файл целесообразно структурировать, что повысит его удобочитаемость. Выделим часть текста программы, которую необходимо “свернуть” (рис. 3.19), щелкнем по ней правой кнопкой мыши и во всплывшем меню выберем команду Document Structure|Create group. Выделенная часть текста окажется свернутой в одну строчку, слева от которой появится знак группы  $\oplus$  (рис. 3.20). Кроме того, она выделяется из обычного текста более темным фоном.

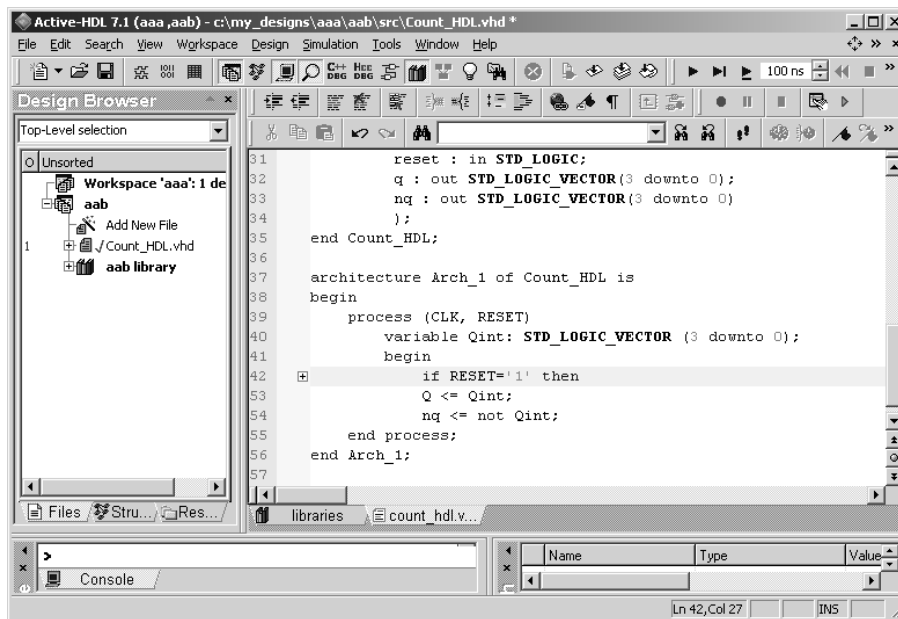


Рис. 3.20

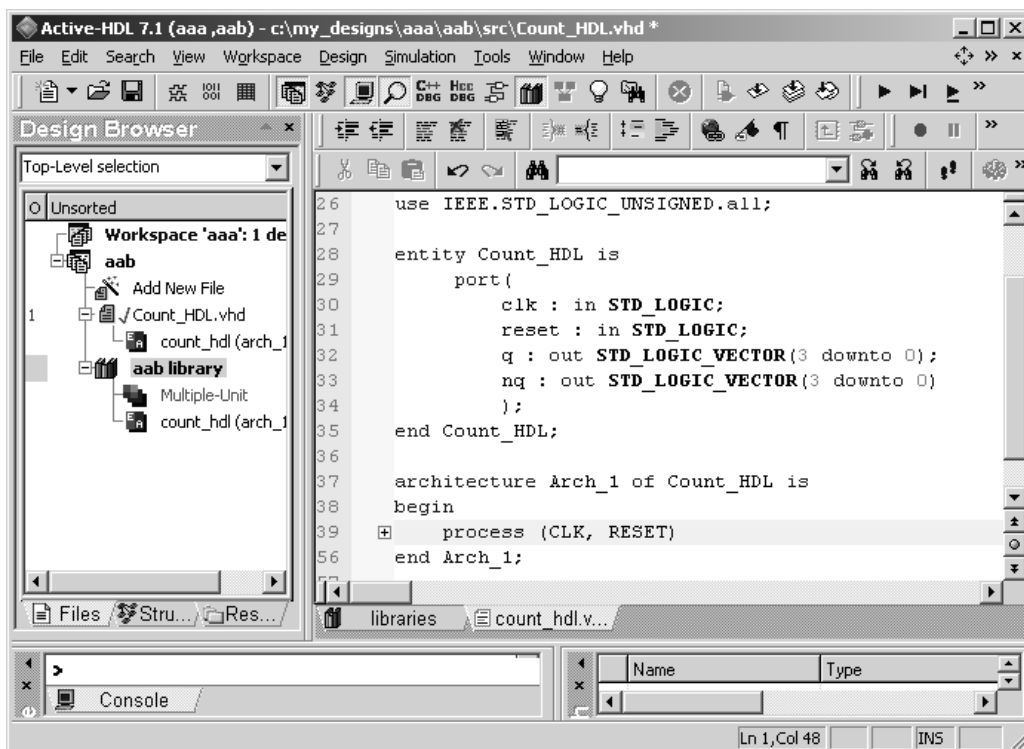


Рис.3.21

Щелкнув левой кнопкой мыши по знаку группы  $\oplus$  ее можно развернуть, значок группы изменит вид :  $\ominus$ . Повторный щелчок по значку группы вновь свернет текст. Группы можно делать вложенными. Выделив текст программы между операторными скобками begin и end Arch\_1 (рис. 3.19) и “свернув” его подобно описанному выше приведем его к виду, показанному на рис. 3.21.



Чтобы разгруппировать текст достаточно щелкнуть по сгруппированному тексту правой кнопкой мыши и во всплывшем меню выбрать команду Document Structure|Remove groups.

3.3.3. Создадим для проекта еще одно устройство нижнего уровня иерархии – декодер для разработанного счетчика. Он должен преобразовывать 4-х разрядный двоичный код с выхода счетчика в 10-разрядный позиционный код, следовательно, входы дешифратора будут подключаться к выходам  $q[3] - q[0]$  счетчика, а выходом являться 10-разрядная шина. Построим два варианта дешифратора с таблицами истинности, приведенными в табл. 3.1 и табл. 3.2

Таблица 3.1

Таблица истинности дешифратора для архитектуры bbc

x3	x2	x1	x0	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0

Щелкнем дважды по ветви Add New File в окне Design Browser и с помощью “мастера” подобно тому, как это делалось выше, создадим заготовку VHDL-файла Desh\_4\_10.vhd (рис. 3.22).

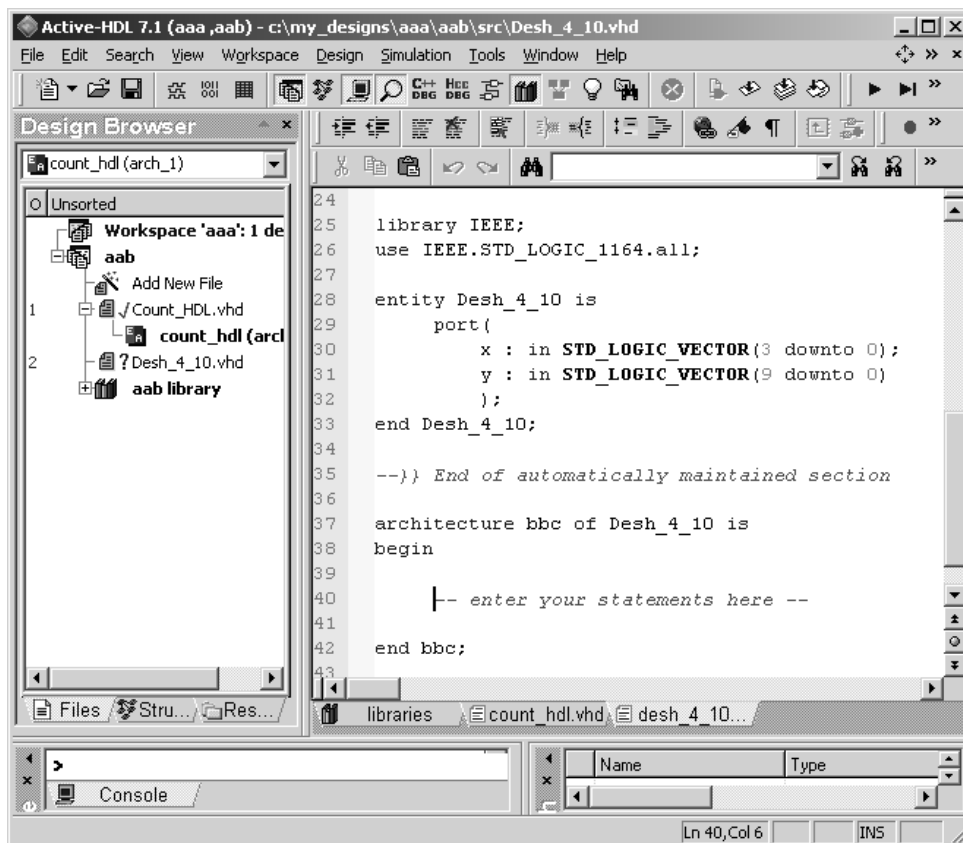


Рис.3.22

Вызовем Language Assistant, выделим в разделе Code Auto Complete пункт process (рис. 3.23) и скопируем шаблон кода, показанного в правом окне, в создаваемый файл (рис 3.23) в раздел architecture заменив им текст “-- enter your statements here --”.

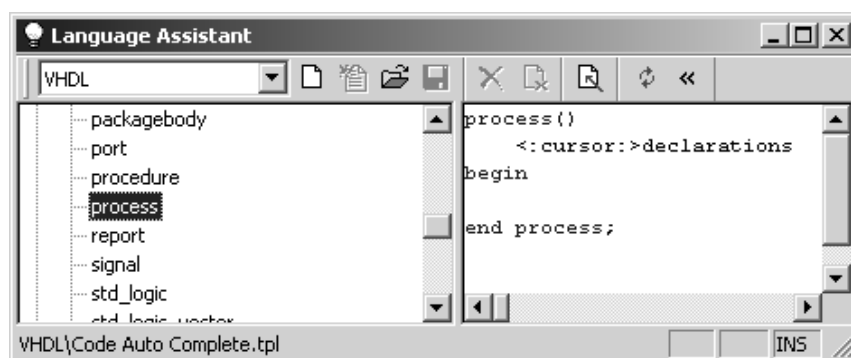


Рис.3.23

Аналогично, используя Language Assistant, вставим внутрь оператора process шаблон оператора case и затем введем текст описания работы дешифратора (рис.3.24).

Откомпилируем файл дешифратора, устраним синтаксические ошибки (если они допущены).

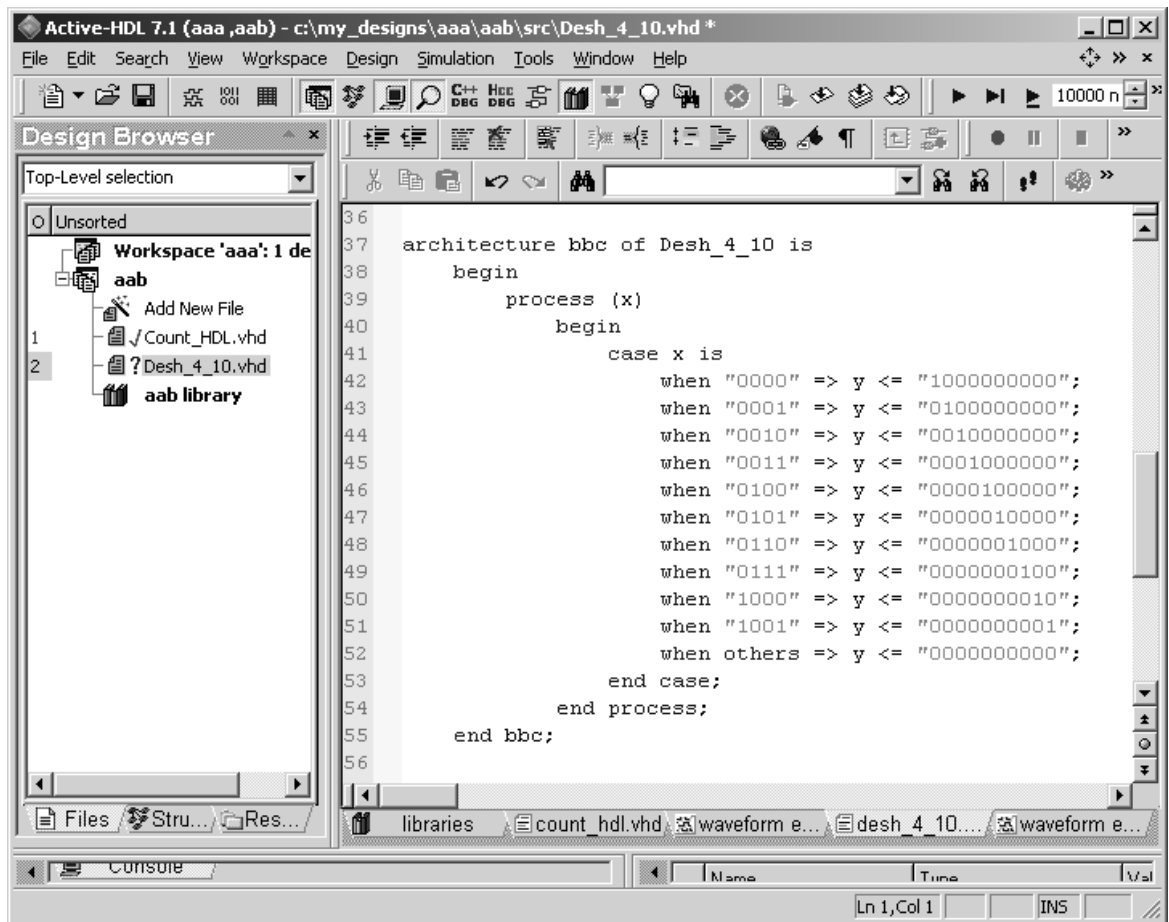


Рис. 3.24

Затем снова, используя Language Assistant, создадим второй вариант архитектуры дешифратора с несколько измененной таблицей истинности (табл.3.2), чтобы различать при моделировании один вариант архитектуры от другого.

Второй вариант архитектуры будем создавать на основе оператора with (рис. 3.25). Откомпилируем его. Если просмотреть окно Design Browser в окне после компиляции файла (рис. 3.25), можно заметить, что в иерархическом дереве проекта появилась еще одна ветвь desh\_4\_10.vhd (bbd), соответствующая второму варианту архитектуры дешифратора.

Таблица 3.2

Таблица истинности дешифратора для архитектуры bbd

x3	x2	x1	x0	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0

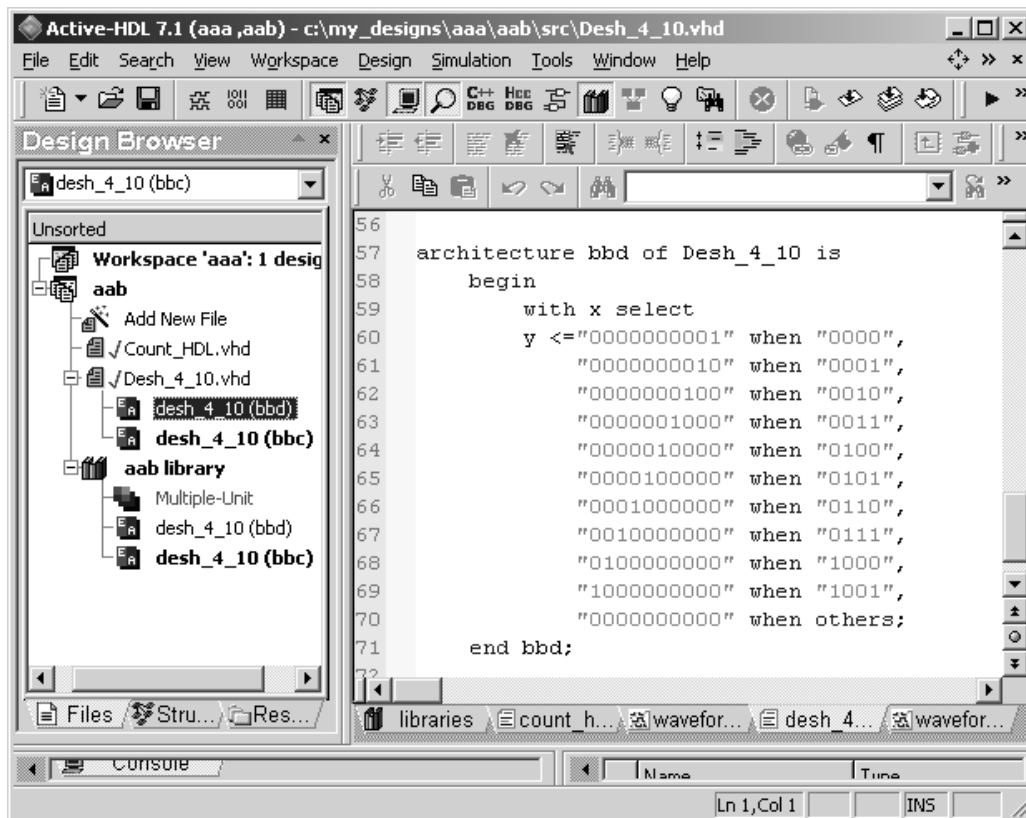


Рис. 3.25

Мы создали две отдельные разработки, содержащиеся в файлах *"Coun\_HDL.vhd"* и *"Desh\_4\_10.vhd"*. Двойным щелчком мыши в окне Design Browser по имени соответствующего файла можно открыть его в HDL-редакторе просмотреть и скорректировать VHDL код.

### 3.4. Технология тестирования проекта

3.4.1. Следующей задачей проектирования является тестирование работы спроектированных устройств. Эту задачу можно выполнить двумя путями:

1) назначить тестируемое устройство головным устройством проекта (временно установить для него Top-Level - верхний уровень иерархии), создать файл временных диаграмм, в котором будут содержаться входные сигналы, и куда будут выводиться результаты тестирования и провести моделирование работы устройства;

2) создать с помощью средств Active HDL файл испытательного стенда (Test bench) и провести тестирование с его помощью.

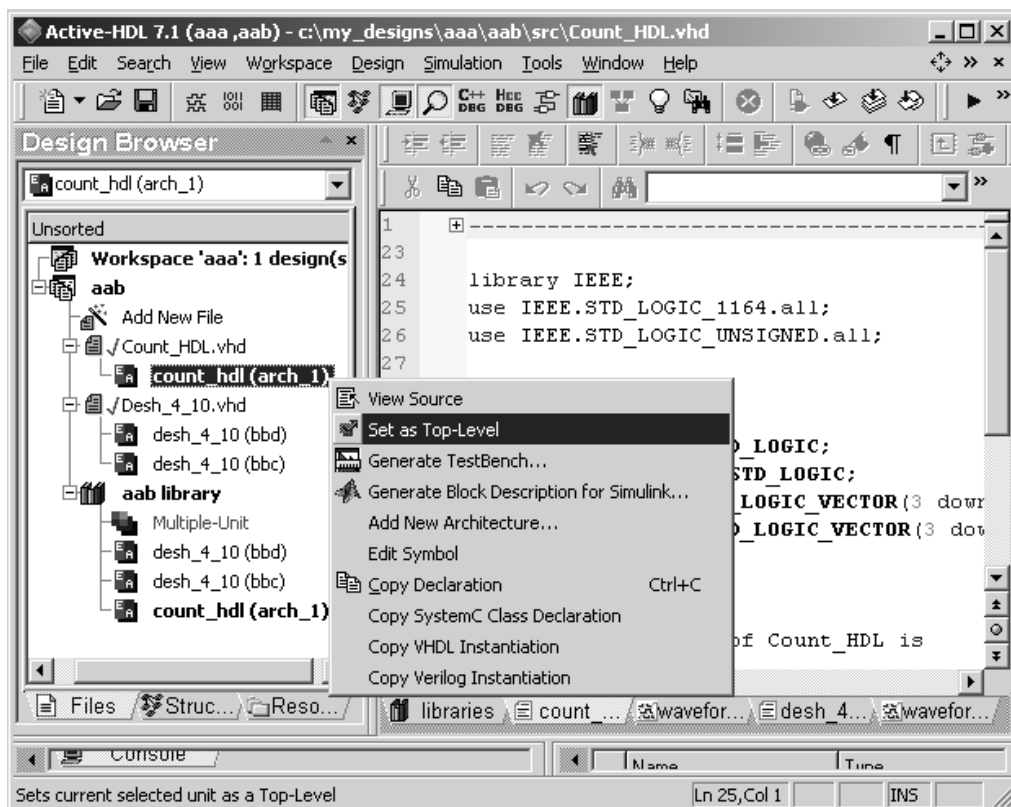



Рис. 3.26

3.4.2. Протестируем работу счетчика с помощью первого варианта технологии, а работу дешифратора - по второму варианту. Выполним компиляцию всех файлов проекта, затем в окне Design Browser раскроем дерево проекта и щелкнем правой кнопкой мыши по ветви count\_hdl (arch\_1), которая появится после компиляции файла счетчика Count\_HDL.vhd. В контекстном меню выберем команду Set as Top\_Level (рис. 3.26)

Затем создадим файл временных диаграмм (команда главного меню File|New|Waveform или кнопка  на панели инструментов). В открывшемся окне редактора временных диаграмм щелкнем правой кнопкой мыши левее вертикальной штриховой разделительной линии (либо просто нажмем Ctrl+I) после чего появится окно Add Signals.

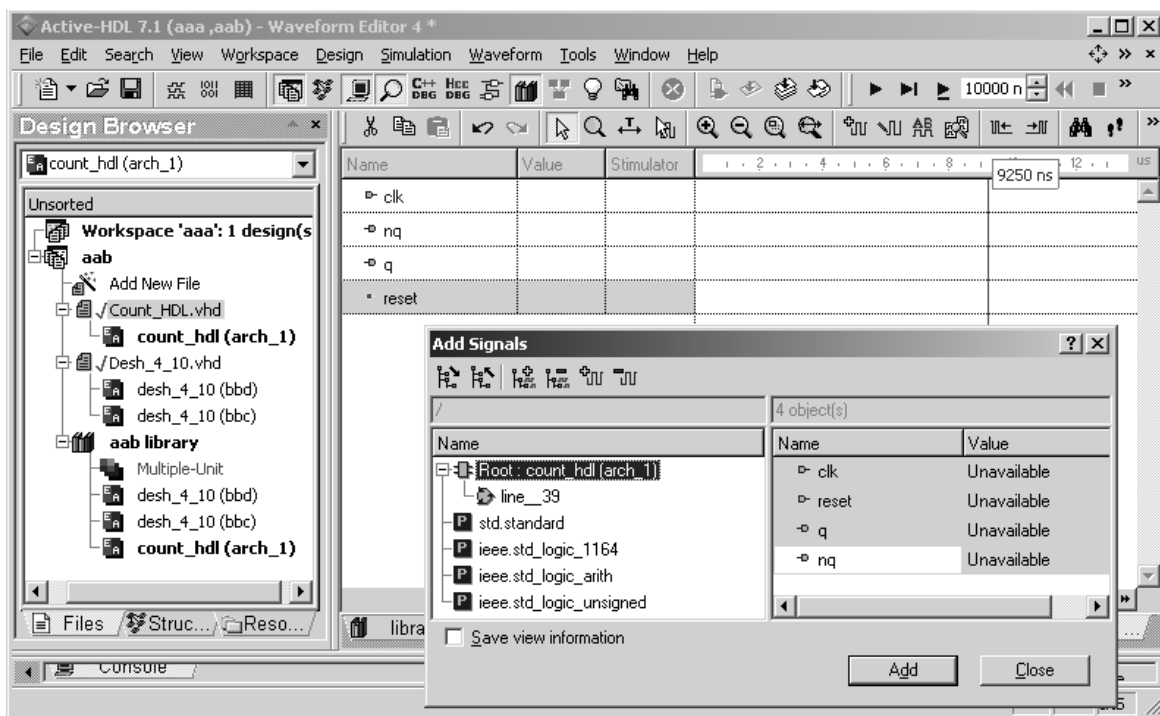


Рис. 3.27

В левой части окна выведен иерархический список модулей, содержащихся в тестируемом устройстве, в правой список – сигналов. Выделим все сигналы и нажмем кнопку Add – в окне редактора временных диаграмм появятся обозначения временных осей для ввода-вывода добавленных сигналов (рис. 3.27). Закроем окно Add Signals и

сформируем входные сигналы для счетчика: clk и reset. Дважды щелкнем левой кнопкой мыши по ячейке, находящейся на пересечении строки clk и колонки Stimulator – в результате этого будет выведено окно Stimulators, в котором можно задавать форму и параметры сигналов. Если выбрать левой кнопкой мыши требуемый тип сигнала в колонке Type, то справа от нее откроется окно для установки параметров выбранного сигнала.

Щелкнем левой кнопкой мыши по сигналу Clock (периодическая последовательность прямоугольных импульсов), откроется окно, в котором можно задать их начальную фазу, период повторения или частоту непосредственно вводя текст в выделенные поля, а также регулировать длительность, захватив и перемещая левой кнопкой мыши интерактивный регулятор – небольшой коричневый квадрат, справа от которого указывается относительная длительность импульса (рис. 3.28).

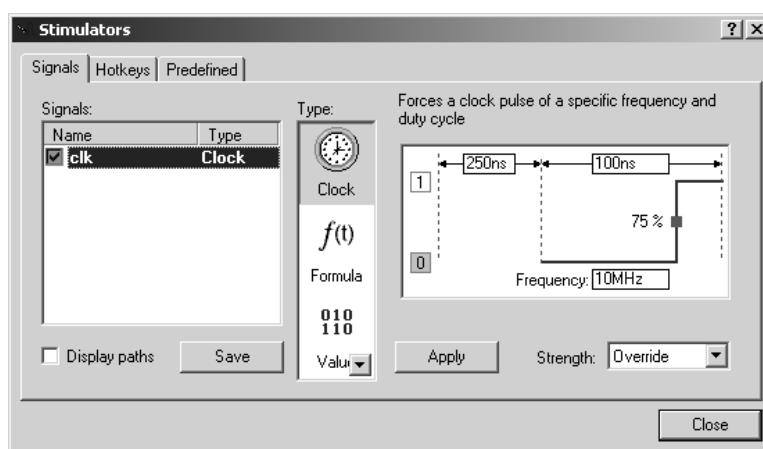


Рис. 3.28

Установив требуемые параметры, необходимо нажать кнопку Apply, после чего переходить к установке параметров сигнала reset.

В окне редактора временных диаграмм дважды щелкнем левой кнопкой мыши по ячейке, находящейся на пересечении строки reset и колонки Stimulator и в открывшемся окне Stimulators в колонке Type выберем пункт Formula, затем щелкнем левой кнопкой мыши по полю value, введем туда цифру 0. Затем заполним ниже появляющиеся

поля value и time offset как показано на рис. 3.29 и нажмем кнопку Apply. В результате этого будет сформирован сигнал reset – прямоугольный импульс длительностью 50 ns с задержкой 50ns от относительно начала координат. При желании этот сигнал можно описывать и далее, вводя информацию непосредственно с поле Enter formula в том же формате. Введем в конец этого поля текст “ , 1 1us, 0 1.1us” как показано на рис 3.30, вновь нажмем кнопку Apply и закроем окно. Мы добавили к ранее описанному импульсу еще один длительностью 0,1 мкс и задержкой 1 мкс относительно начала координат.

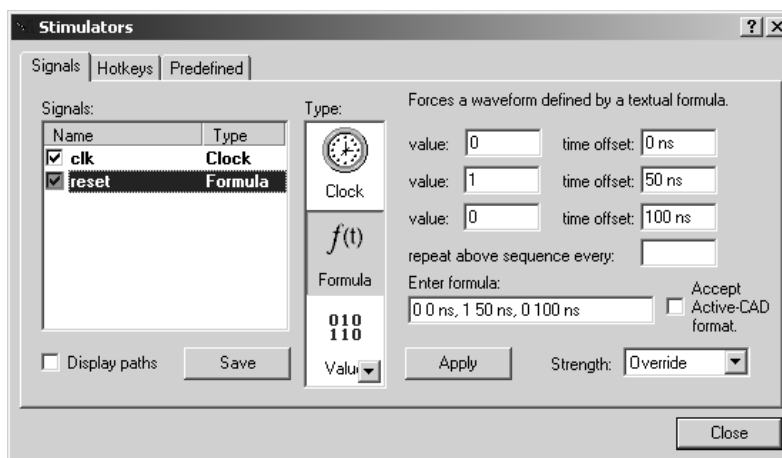


Рис. 3.29

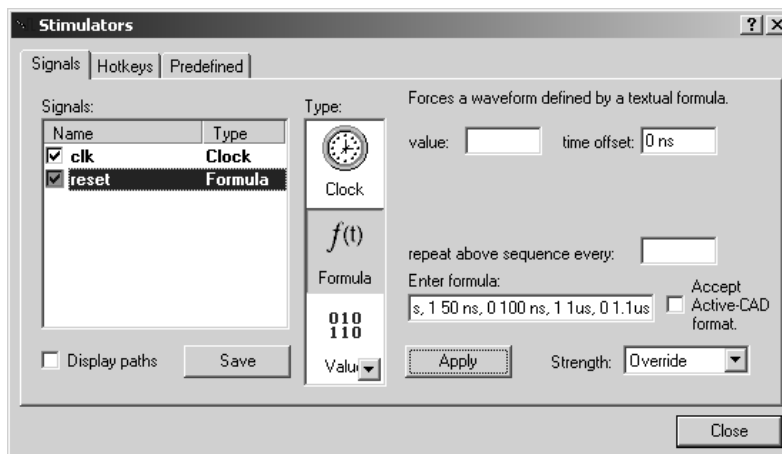


Рис. 3.30

3.4.3. На этом формирование входных сигналов для счетчика закончено и можно переходить непосредственно к моделированию его работы. Управление процессом моделирования можно выполнять



командами раздела Simulation главного меню, наиболее употребительные из которых выведены на панель управления (рис. 3.31).



Рис. 3.31

На ней размещены следующие инструменты:

- 1) Run (Alt+F5) – включить режим моделирования неограниченное время: результаты моделирования не выводятся до тех пор, пока не будет введена команда End Simulation (кнопка 6);
- 2) Run Until (F5) – моделировать в течение интервала времени, установленного окне Run Until, выводимом каждый раз после нажатия этой кнопки. По окончании указанного интервала выводятся результаты, а процесс моделирования приостанавливается до введения команды на продолжение моделирования;
- 3) Run For – моделировать в течение интервала времени, установленного окне 4, после чего выводятся результаты, а процесс моделирования приостанавливается до введения повторной команды.
- 4) Окно установки длительности моделирования;
- 5) Restart Simulation – повторить моделирование сначала;
- 6) End Simulation – закончить моделирование.

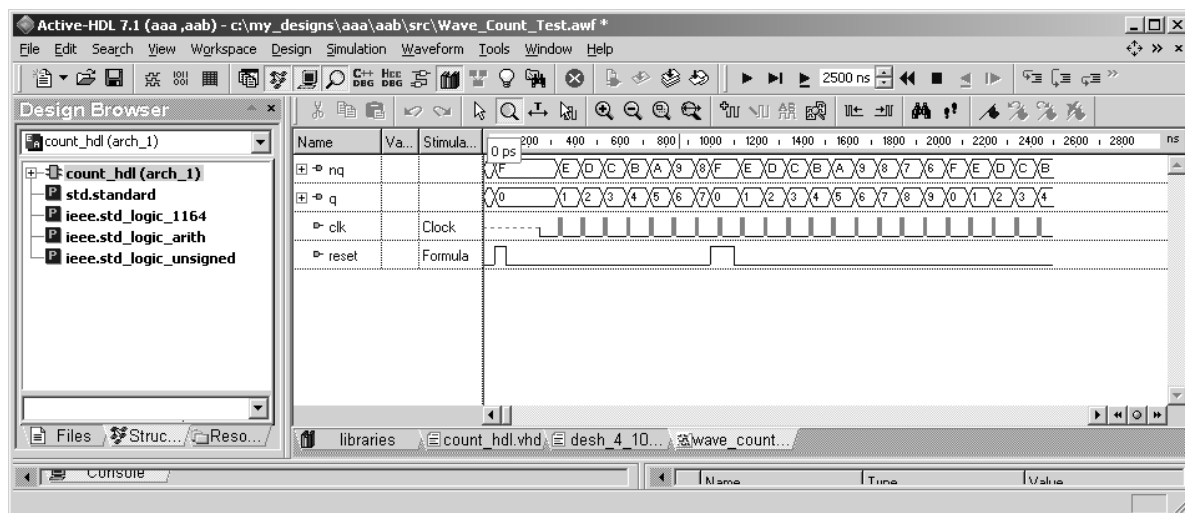


Рис. 3.32

Установим интервал моделирования равным 2500 ns и введем команду Run For. Результаты моделирования после выполнения этой команды показаны на рис. 3.32.

На двух верхних графиках показаны шестнадцатеричные коды временных диаграмм сигналов на шинах q[3..0] и nq[3..0]. Раскрыв шины двойным щелчком левой кнопки мыши по шине, эти сигналы можно представить более наглядно в двоичной форме (рис. 3.33).

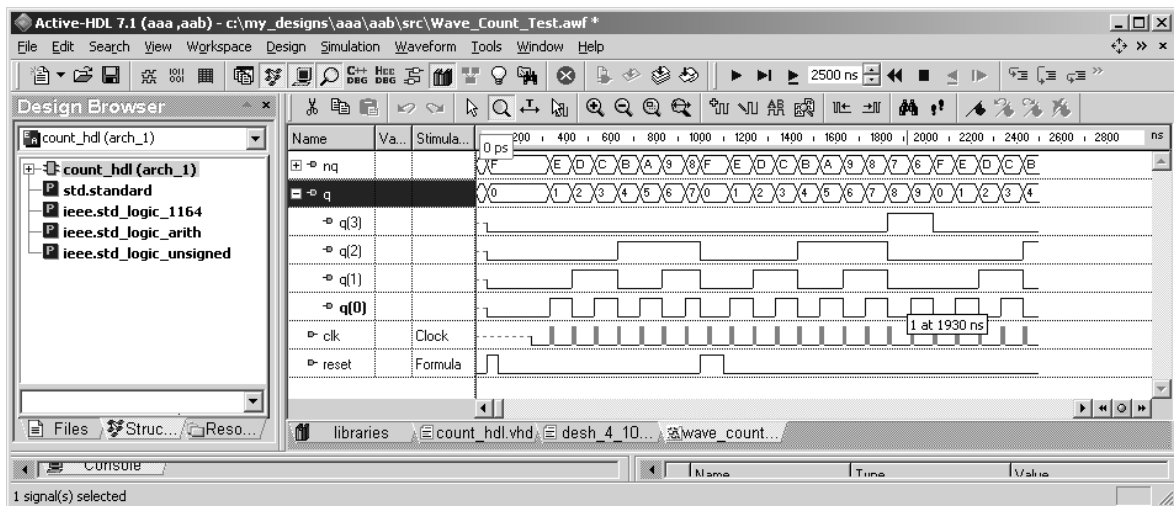


Рис. 3.33

Active HDL предоставляет ряд инструментов для анализа временных диаграмм и документирования результатов моделирования.

Некоторые из инструментов доступны из контекстного меню, которое вызывается щелчком правой кнопки мыши по названию сигнала в колонке Name (рис. 3.32, 3.33). Выделив требуемый сигнал или группу сигналов можно их скопировать в буфер обмена, удалить, очистить временную диаграмму и т.д. Следует иметь в виду, что некоторые пункты меню и инструменты становятся доступными только после ввода команды End Simulation (нажатия кнопки 6, рис. 3.31).



Рис. 3.34

3.4.4. Для детального анализа временных диаграмм можно воспользоваться инструментами на панелях анализа и редактирования временных диаграмм сигналов (рис. 3.34).

На них размещены следующие инструменты:

1) Zoom mode – включить режим масштабирования (электронной лупы). Выделив с помощью курсора требуемый участок временной диаграммы его можно растянуть по горизонтали на все окно. Чтобы вернуться к исходному изображению необходимо нажать кнопку **Zoom To Fit**;

2) Measurement mode – режим измерения временных интервалов. Для измерения требуемого временного интервала необходимо выбрать этот инструмент и выделить его левой кнопкой мыши;

3) Edit Mode - режим редактирования временных интервалов. Используя этот режим можно после моделирования откорректировать входные сигналы, затем в окне Stimulators (рис.3.29, 3.30) в колонке Type установить для этих сигналов тип Custom и нажать кнопку Apply. После этого можно выполнять моделирование с откорректированными сигналами;

4) Zoom In – увеличить масштаб по оси времени в 2 раза;

5) Zoom Out – уменьшить масштаб по оси времени в 2 раза;

6) Zoom To Fit – восстановить исходный масштаб;

7) Zoom Range – ввести нижнюю и верхнюю границы по временной оси;

8) Add Signals (Ctrl+I) – вывести окно добавления сигналов;

9) Stimulators - вывести окно конструктора сигналов Stimulators;

10) Insert comment – вставить комментарий к выделенному сигналу;

11) Compare waveforms – сравнить формы сигналов в текущем окне и ранее запомненном;

12) Previous event (Ctrl+F12) перевести визир (перемещаемую курсором вертикальную линию) к предыдущему событию (моменту изменения какого-либо из сигналов);

- 13) Next event (Shift+F12) - перевести визир к следующему событию;
- 14) Find (Ctrl+F) - найти величину или текст комментария;
- 15) Go To – перейти к моменту времени, указываемому во всплывающем окне;
- 16) Toggle bookmark(Ctrl+F2) - установить закладку в месте расположения визира;
- 17) Previous bookmark(Shift +F2) – перейти к предыдущей закладке;
- 18) Next bookmark (F2) – перейти к следующей закладке;
- 19) Clear all bookmarks – убрать все закладки.

### **3.5. Тестирование проекта с помощью испытательного стенда Test Bench**

3.5.1. Протестируем работу дешифратора с помощью формируемого Active HDL испытательного стенда Test Bench.

Испытательный стенд Test Bench представляет собой HDL-программу, которая связывает воедино все файлы проекта, определяет необходимые для испытания устройства входные и выходные порты и сигналы для них. Разработка программы для испытательного стенда требует аккуратности и достаточно высокой квалификации программиста. В Active HDL процедура создания испытательного стенда в большой степени автоматизирована и может быть выполнена с помощью “мастера” Test Bench Generator Wizard. Испытательный стенд может создаваться как для проекта в целом, так и для тестирования отдельных его компонентов.

Откомпилируем файл desh\_4\_10.vhd, найдем в окне Design Browser ветвь desh\_4\_10 (bbd), соответствующую варианту архитектуры bbd, щелкнем по ней правой кнопкой мыши. В контекстном меню выберем пункт Generate Test Bench (рис. 3.35) и, отвечая на вопросы “мастера” Test Bench Generator Wizard (рис.3.36), в диалоговом режиме создаем файл испытательного стенда desh\_4\_10\_TB.vhd.

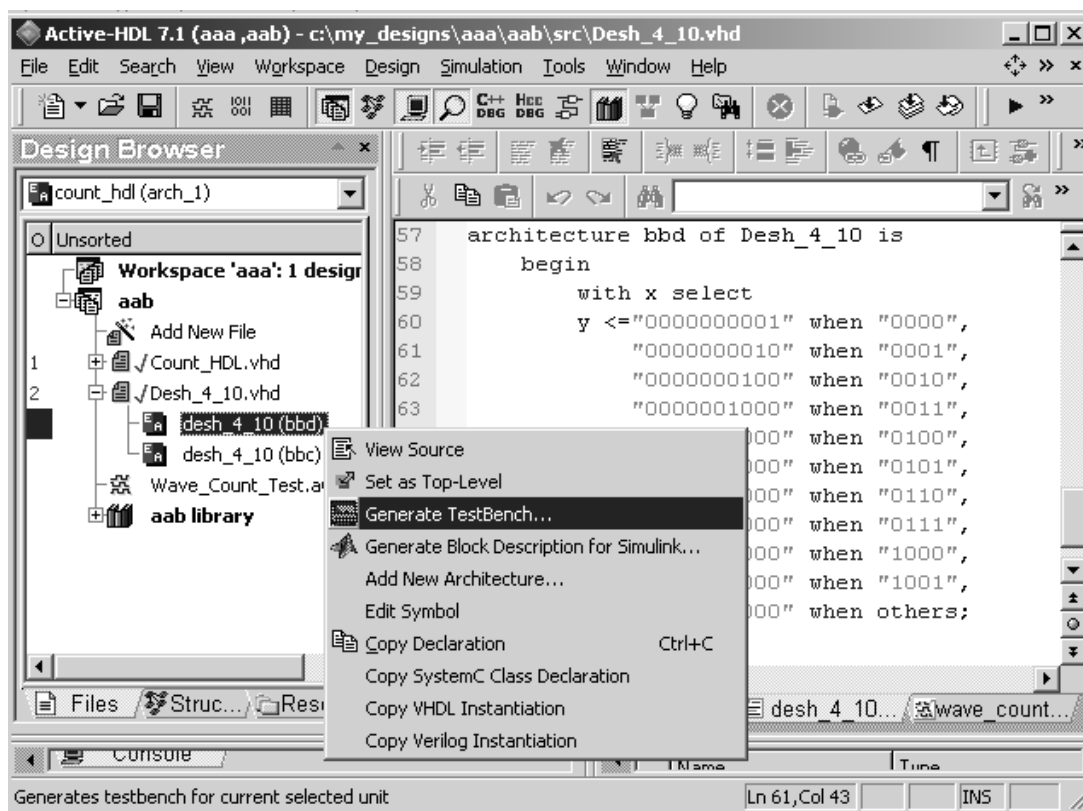


Рис. 3.35

В данном случае достаточно просто подтвердить ваше согласие с параметрами предлагаемыми “мастером”.

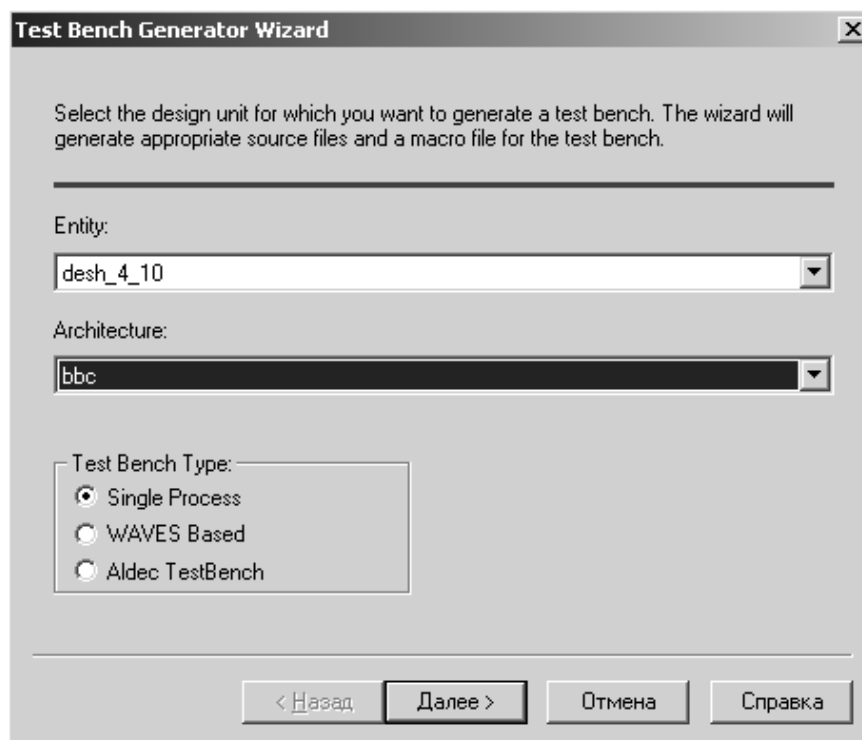


Рис. 3.36

Далее откроем в окне Design Browser папку TestBench, найдем в ней сгенерированный “мастером” файл испытательного стенда desh\_4\_10\_TB.vhd и командный файл desh\_4\_10\_TB\_runtest.do для его автоматической компиляции и моделирования. Далее можно переходить непосредственно к тестированию дешифратора. В окне Design Browser находим файл desh\_4\_10\_TB\_runtest.do, щелкнем по нему правой кнопкой мыши, затем во всплывающем меню выберем команду Execute (рис. 3.37) и выполним ее.

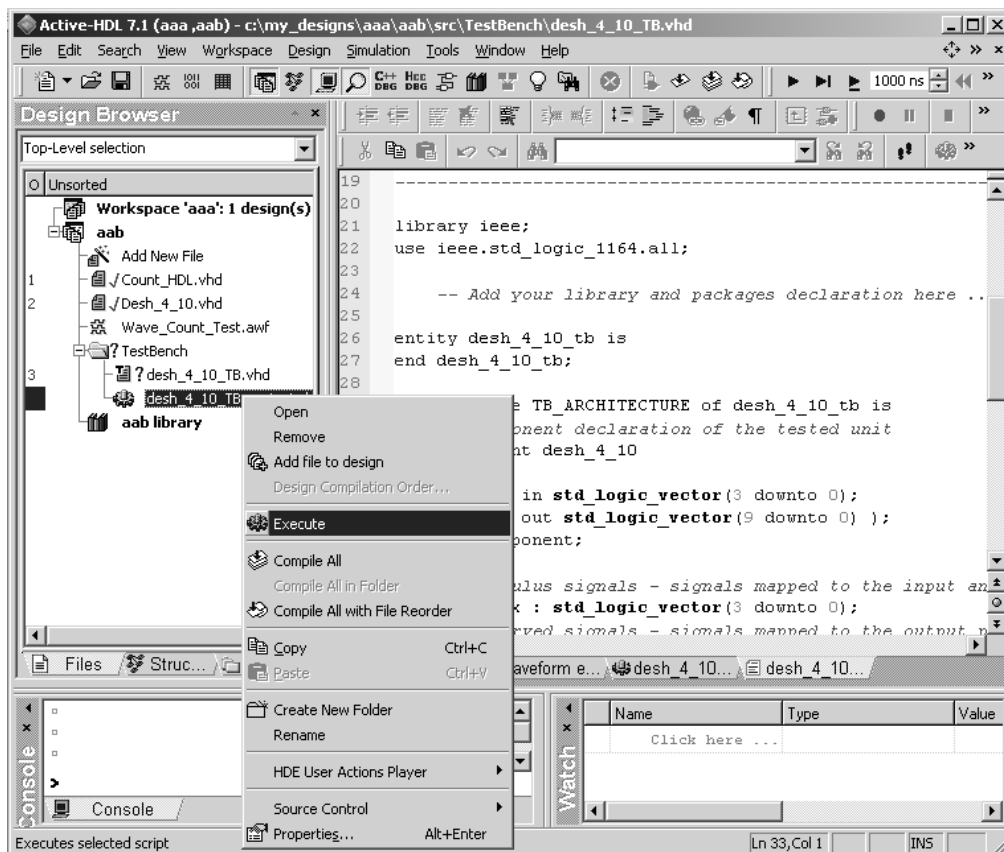


Рис. 3.37

В результате выполнения командного файла будут автоматически откомпилированы необходимые файлы проекта и тестируемому файлу будет назначен верхний уровень иерархии. Затем автоматически запустится Waveform Editor, и в его рабочем окне появятся временные оси для входных и выходных сигналов. Выделив сигнал *x* двойным щелчком левой кнопки мыши по ячейке Stimulator, открываем окно Stimulators и в колонке Type выбираем в качестве

типа сигнала Counter (сигнал с выхода двоичного счетчика). Нажмем кнопку Apply и закроем окно Stimulators.

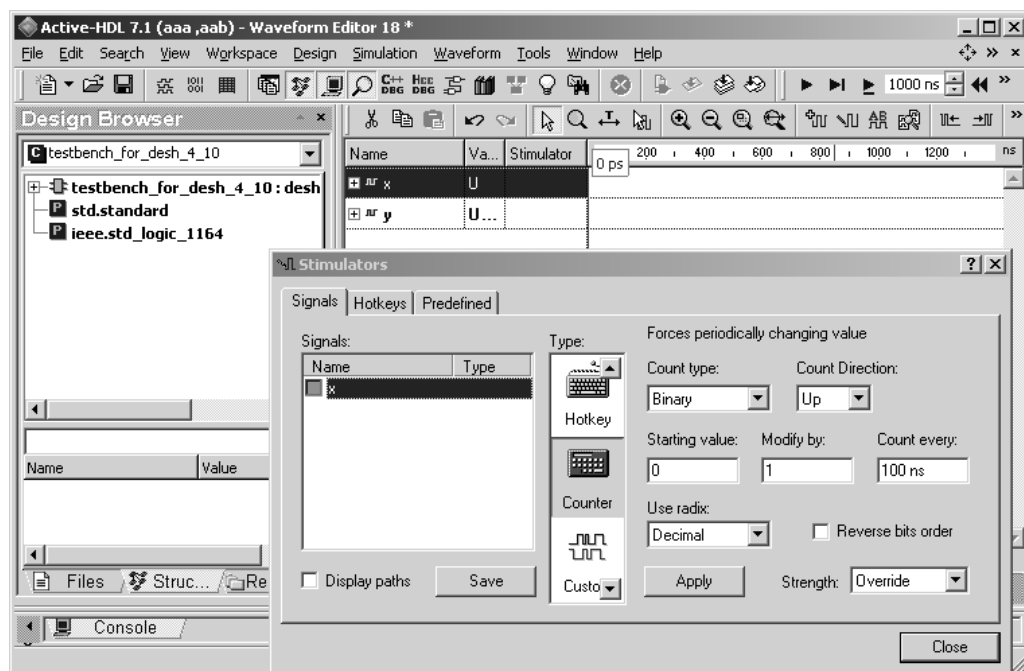


Рис. 3.38

Установим время цикла моделирования, равным 3000 ns и, нажав клавишу F5, промоделируем работу дешифратора. Результаты моделирования приведены на рис. 3.39.

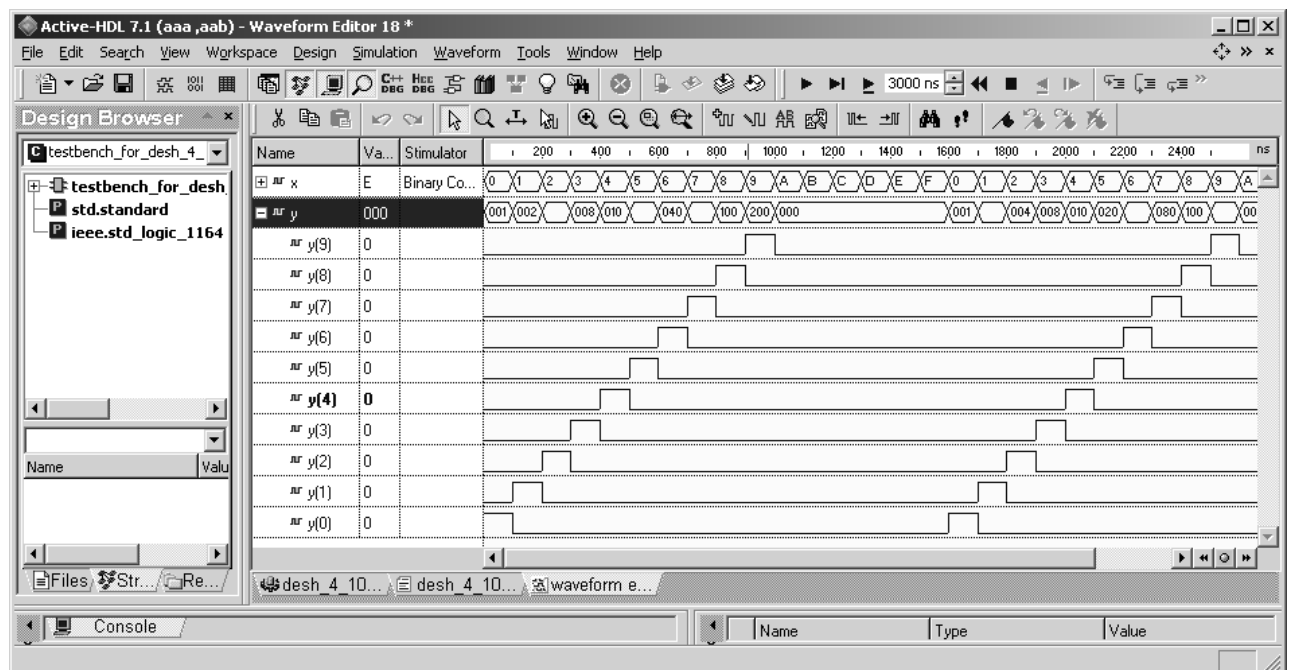


Рис. 3.38

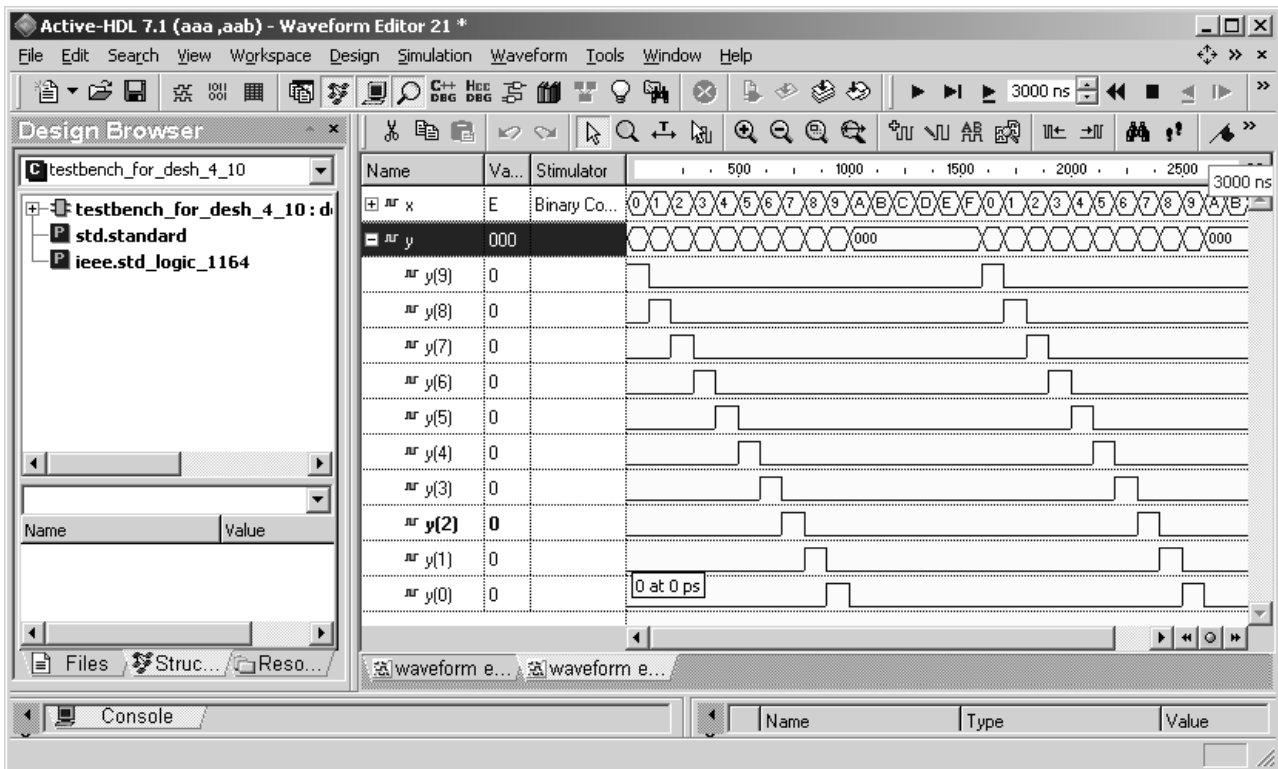


Рис. 3.39

На рис. 3.39 приведены полученные таким же путем результаты моделирования дешифратора для варианта архитектуры bbc.

### 3.6. Технология графического проектирования

3.6.1. Создадим с помощью графического редактора проект верхнего уровня иерархии, объединяющий счетчик и дешифратор.

Добавим в проект пустой файл Top.bde. Это можно сделать несколькими путями подобно тому, как это делалось выше для HDL-файла (см. п. 3.2, рис. 3.7 – 3.11) с тем лишь отличием, что в окне Add New File выбирается значок Block Diagram. Инструменты для работы с редактором блок-диаграмм описаны выше (п. 2.4).

Разместим на поле редактора блок-диаграмм необходимые компоненты. Нажмем клавишу I, разместим на поле два входных порта Input0 и Input1, затем нажмем комбинацию клавиш Shift+O и разместим два выходных порта для шин BusOutput0(7:0). Затем выберем инструмент Select (клавиша Esc) и, дважды щелкнув левой кнопкой мыши по названиям портов, переименуем их, как показано на рис.



3.40. Обратите внимание на то, чтобы при переименовании была изменена на нужную размерность выходных портов шин.

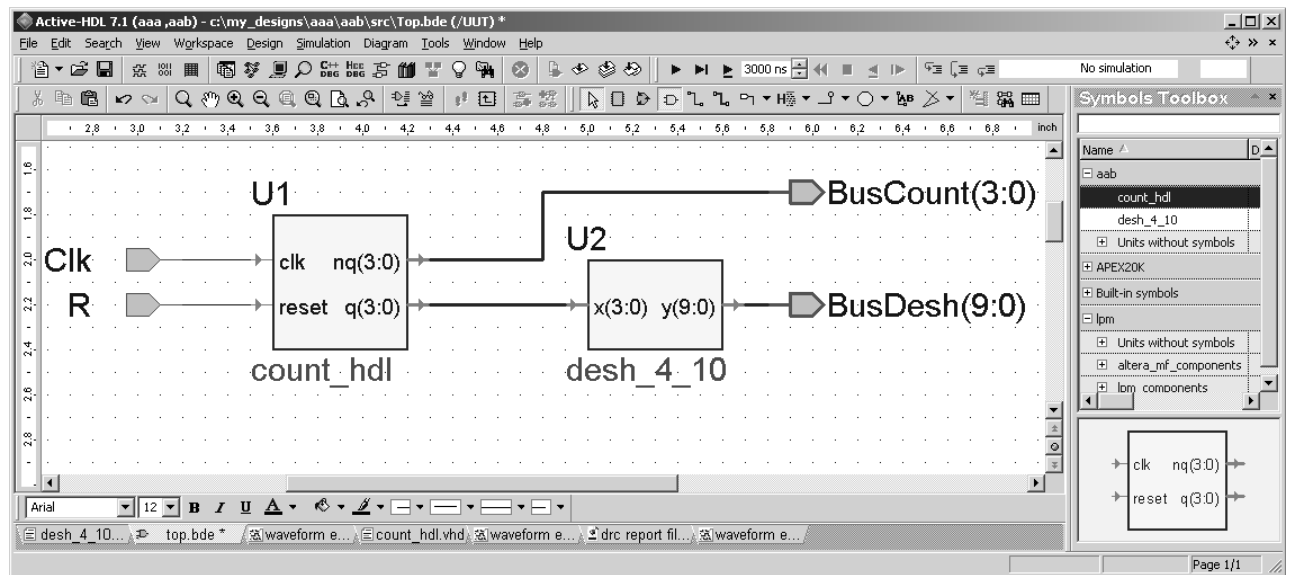


Рис. 3.40

Затем нажмем клавишу S и в появившемся окне Symbol Toolbox раскроем список aab, выберем по очереди символы компонентов count\_hdl и desh\_4\_10 и разместим их на поле редактора. Нажав клавишу W, соединим входные порты Clk и R со входами компонента count\_hdl, а затем, нажав клавишу B, выполним необходимые соединения с помощью шин. Готовый файл проекта Top.bde показан на рис. 3.40.

3.6.2. Протестируем проект в целом. Откроем окно Design Browser (Alt+1), найдем в нем ветвь Top.bde, щелкнем по ней правой кнопкой мыши и откомпилируем файл, чтобы проверить его на отсутствие формальных ошибок. Если таковые отсутствуют, удалим все ранее скомпилированные файлы (выделим ветвь aab library, щелкнем по ней правой кнопкой мыши и в контекстном меню выберем пункт Delete Simulation Data). Вновь выделим файл Top.bde, щелкнем по нему правой кнопкой мыши, в контекстном меню выберем пункт Compile all и выполним компиляцию всего проекта. Затем выделим откомпилированный файл для Top.bde (ветвь top(top)), на-

значим ему верхний уровень иерархии и создадим для него испытательный стенд.

После этого можно выполнить командный файл для испытательного стенда. Раскрыв список TestBench, найдем в нем ветвь top\_TV\_runtest.do, щелкнем по ней левой кнопкой мыши и выберем в контекстном меню пункт Execute. В результате этих действий откроется Waveform Editor и на экран монитора будет выведен пустой файл, содержащий заготовки временных диаграмм для входных и выходных сигналов проекта. Сформируем временные диаграммы входных сигналов подобно тому, как это делалось выше при тестировании работы счетчика (п. 3.3.7) и выполним моделирование работы устройства (нажав, например, клавишу F5). Результаты моделирования приведены на рис. 3.41.

Анализ результатов моделирования показывает, что устройство работает правильно.

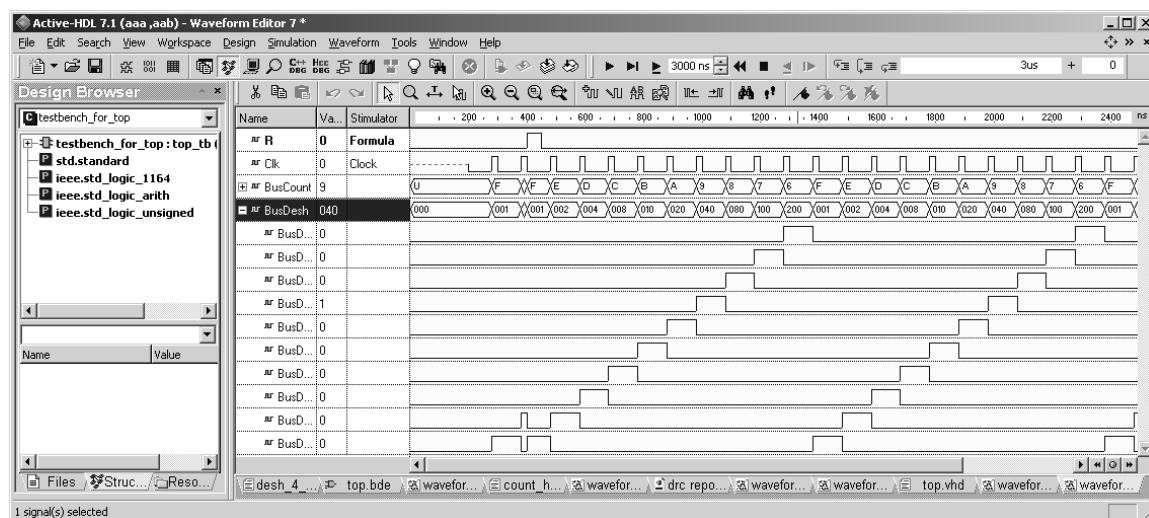


Рис. 3.41

**3.6.3. Трассировка работы программы.** Active HDL позволяет трассировать исходный текст HDL-файла во время моделирования: выполнять пошаговое моделирование работы программы в интерактивном режиме. Окно с текстом моделируемого файла автоматически открывается после начала моделирования с помощью команд пошагового моделирования. Строка, которая будет выполняться следующей, выделяется в HDL-редакторе желтым цветом.

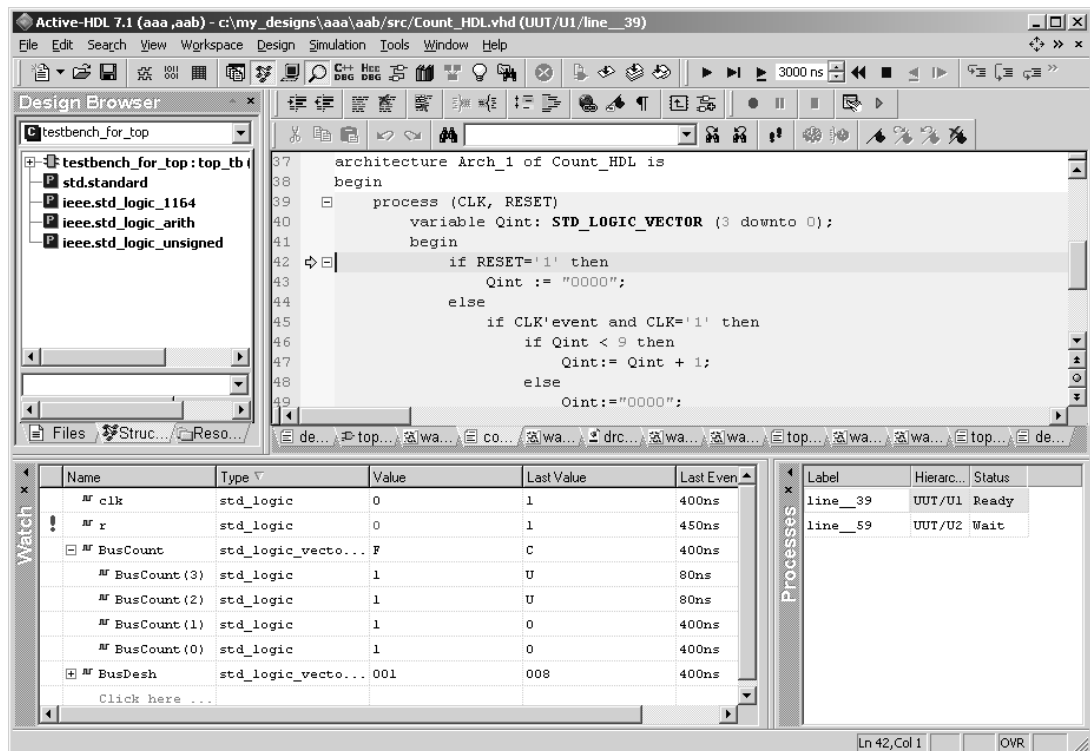




Рис.3.42


Сначала необходимо закончить выполнение предыдущего моделирования, выбрав в главном меню команду Simulation|End Simulation и повторно инициализировать начало моделирования, командой Simulation|Initialize Simulation. Затем сделаем видимыми окна Watch и Processes (команды главного меню View|Watch и View|Processes).

Окно Processes показывает статус процессов проекта. Окно Watch позволяет контролировать значения переменных и сигналов.

В режиме отладки доступны следующие команды:

 Trace into - выполняет одиночную VHDL инструкцию. Если при ее выполнении выполняется подпрограмма, трассировка переходит в тело подпрограммы.

 Trace over - выполняет одиночную VHDL команду. Если при ее выполнении выполняется подпрограмма, инструкции, содержащиеся в пределах тела подпрограммы, выполняются в за один шаг.

 Trace out - выполняет столько VHDL инструкций, сколько необходимо, чтобы завершить выполнение подпрограммы. Если под-

программы вложены друг в друга, команда завершает выполнение только самой внутренней подпрограммы.

Используя эти команды, можно наблюдать изменения значений сигналов в окне Watch. Знак восклицания, появляющийся перед именем сигнала, указывает на изменение значения сигнала в текущем цикле моделирования (сигнал CLK на рис. 3.42).

В окне Processes можно наблюдать список процессов и их текущее состояние (*ready* или *wait*). Состояние *ready* означает, что процесс в настоящее время активен, и он автоматически перемещается к вершине отображенного списка.

## ЗАКЛЮЧЕНИЕ

Во второй части пособия описана технология выполнения проектных операций в одной современных САПР цифровых устройств на ПЛИС Active HDL. Рассмотрены принципы проектирования цифровых схем на HDL-языках, а также с использованием графических средств: редактора блок-диаграмм и редактора состояний. Рассмотрены варианты ручного и автоматизированного тестирования спроектированных устройств с помощью включения в проект специальной программы – испытательного стенда. Проиллюстрирована технология создания комбинированного проекта иерархической структуры. За рамками пособия заключительные этапы проектирования цифровых устройств на ПЛИС – конструкторско-технологические аспекты (предпочтения при размещении отдельных компонентов проектируемого устройства в теле кристалла, распределение сигналов по выводам БИС и т.д.). Тем не менее, приведенный материал позволяет составить достаточно полное представление о технологии и принципах проектирования цифровых устройств в рассмотренной системе.

Пособие может оказаться полезным при изучении студентами дисциплин по цифровой схемотехнике и по освоению HDL-языков.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. [www.aldec.com](http://www.aldec.com)
2. Поляков А. К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. – М.: СОЛОН-Пресс, 2003. – 320 с: ил. – (Серия "Системы проектирования").

**Шеболков Виктор Васильевич**

**Современные средства автоматизированного  
проектирования цифровых устройств на ПЛИС**

Часть 2

**Проектирование цифровых устройств в системе  
Active HDL 7.1**

Учебное пособие

Ответственный за выпуск Шеболков В.В.

Редактор

Корректор

ЛР №020565 от 23.06. 1997г. Подписано к печати \_\_\_\_\_ г.

Бумага офсетная. Печать офсетная.

Формат 60х84<sub>1/16</sub>

Усл.п.л. – 4,0 Уч.-изд.л. –

Заказ № \_\_\_\_\_ Тираж экз.

”С”

---

Издательство Технологического института  
Южного федерального университета  
ГСП 17А, Таганрог, 28, Некрасовский, 44  
Типография Технологического института  
Южного федерального университета  
ГСП 17А, Таганрог, 28, Энгельса, 1