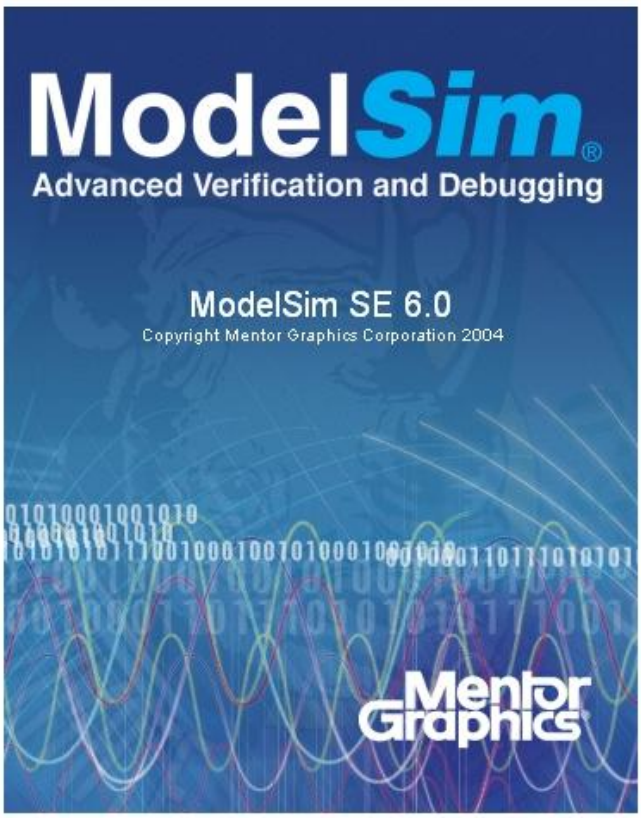
Севастопольский национальный технический университет
Кафедра радиотехники и телекоммуникаций

Методическое пособие

по применению программы ModelSim для моделирования цифровых интегральных схем



Севастополь
2010

ВВЕДЕНИЕ

Одна из характерных тенденций современного этапа развития технологии проектирования цифровых систем — применение языков описания аппаратуры HDL (Hardware Description Language) высокого уровня, среди которых наибольшее распространение получили Verilog и VHDL. Среди этих языков особое место принадлежит языку Verilog, получившему статус международного стандарта. Выполняя роль интегрирующего элемента в маршрутах проектирования электронных устройств, язык Verilog стал входным языком для многих программ моделирования и синтеза проектных решений, связывающим различные этапы проектирования в единый процесс.

При этом не только возрастает роль средств моделирования в процессе разработки устройств, но и меняются методы и требования, предъявляемые к процедурам верификации. Для своевременного обнаружения возможных ошибок средства моделирования должны обеспечивать возможность контроля результатов каждого этапа процесса проектирования: создания исходных HDL-описаний, синтеза, размещения и трассировки в кристалл. Такой подход обеспечивает минимальное время разработки устройства и сокращает стоимость этого процесса, так как цена ошибки возрастает с каждым последующим шагом проектирования.

Современные интегрированные САПР СБИС состоят из большого числа программ, различающихся ориентацией на различные проектные процедуры и разные типы схем. Наиболее известными разработчиками интегрированных САПР являются фирмы Synopsys, Cadence Design Systems, Mentor Graphics. Настоящее пособие предназначено для первоначального изучения основных этапов проектирования цифровых интегральных схем с использованием системы ModelSim компании Mentor Graphics.

1. НАЗНАЧЕНИЕ И ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПАКЕТА MODELSIM

Пакет программных средств ModelSim в настоящее время является одной из наиболее распространенных систем моделирования цифровых устройств. Программа ModelSim предназначена для симуляции и верификации проектов интегральных схем (ИС), созданных на основе их HDL-описаний. Отличительные особенности пакета:

— полная поддержка основных стандартов языков Verilog, VHDL, System C и их расширений;

— единое ядро моделирования пакета, обеспечивающее возможность полной отладки «смешанных» проектов, которые одновременно содержат модули, написанные на VHDL и Verilog;

— поддержка библиотек всех ведущих фирм-изготовителей как программируемых логических интегральных схем (ПЛИС) семейств FPGA (Field Programmable Gate Array) и CPLD (Complex Programmable Logic Device), так и ASIC (Application-Specific Integrated Circuit);

— высокая скорость компиляции и моделирования, обеспечивающая минимальное время отладки систем различного уровня сложности. Одним из главных факторов, повышающих производительность, является использование принципа оптимизированной прямой компиляции. В соответствии с этим принципом исходные VHDL- или Verilog-описания компилируются в машинно-независимый объектный код, исполняемый на любой поддерживаемой платформе;

— открытая архитектура программных средств ModelSim, обеспечивающая тесную интеграцию с пакетами САПР «третьих» фирм. Средства управления пользовательским интерфейсом Tcl (Tool command language) и Tk (Tool kit) предоставляют возможность организации прямого доступа к моделирующему ядру ModelSim, загрузки информации о выполнении процесса моделирования и его результатов в среду используемой САПР и управления работой системы ModelSim через интерфейс применяемых средств проектирования;

— расширенные отладочные возможности пакета, позволяющие пользователю не только быстро отыскать и идентифицировать ошибки, но и сразу же устранить причины их возникновения;

— возможность работы в различных режимах, в том числе и пакетном. Разработав и отладив некоторый сценарий моделирования в интерактивном режиме, можно оформить его для последующего использования в виде пакетного командного файла.

2. ПРИМЕНЕНИЕ ПАКЕТЕ MODELSIM ДЛЯ ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ ИНТЕГРАЛЬНЫХ СХЕМ

2.1. Этапы моделирования ИС в среде ModelSim

При работе в среде ModelSim файлы с HDL-описаниями блоков, а также системные файлы, которые создаются и взаимодействуют друг с другом в процессе моделирования, целесообразно объединять в единый проект. Создание проекта не является обязательным требованием, но часто позволяет сократить объем рутинных операций при моделировании.

В случае использования проекта этапы моделирования цифрового устройства могут быть отображены в виде последовательности процедур, приведенных на рис. 1.

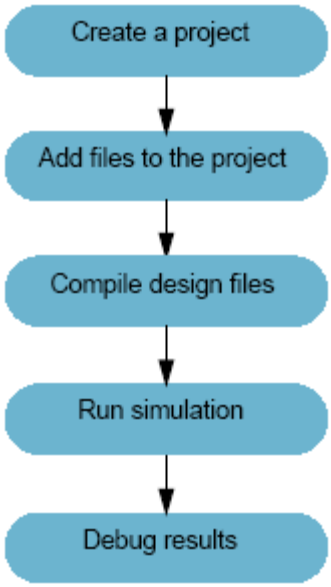


Рис. 1. — Последовательность моделирования в программе ModelSim

2.2. Пример моделирования цифрового устройства на примере четырехразрядного счетчика

Рассмотрим простой пример моделирования четырехразрядного счетчика с использованием его Verilog-описания. Проверка Verilog-описания счетчика сводится к формированию тестовых воздействий (сигналов) на входах отлаживаемого устройства, представленного в виде модуля, и наблюдении сигналов на выходах. Для создания таких воздействий можно, например, написать еще один модуль на языке Verilog (test_bench), который будет использовать исходный модуль как компонент.

При описании счетчика будем использовать блочно-иерархический подход, в соответствии с которым он может быть представлен как совокупность более простых устройств. Для начала рассмотрим схему счетчика, чтобы произвести его разбиение на составные части.

Как известно, четырехразрядный асинхронный счетчик строится на основе четырех Т-триггеров (T_FF), с изменением их состояний по заднему фронту счетных импульсов (рис. 2).

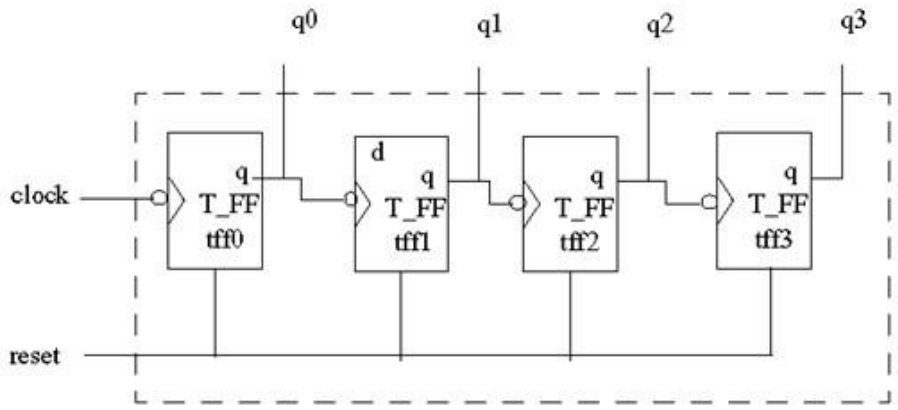


Рис. 2. — Четырехразрядный асинхронный счетчик

Каждый из Т-триггеров может быть выполнен на основе D-триггера (D_FF) и инвертора (принимая, что $\bar{q}$ выход отсутствует в D-триггере), как показано на рис. 3, б.

а)

reset	q_n	q_{n+1}
1	1	0
1	0	0
0	0	1
0	1	0
0	0	0

б)

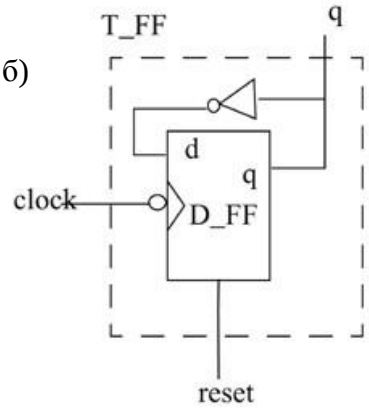


Рис. 3.— Таблица истинности а) и схема Т-триггера б)

Таким образом, проектируемый счетчик с точки зрения блочно-иерархического подхода может быть подвержен декомпозиции в соответствии с рис. 4. В нашем примере счетчик будет состоять из четырех Т-триггеров, каждый из которых в свою очередь будет состоять из одного D-триггера и одного инвертора (not gate).

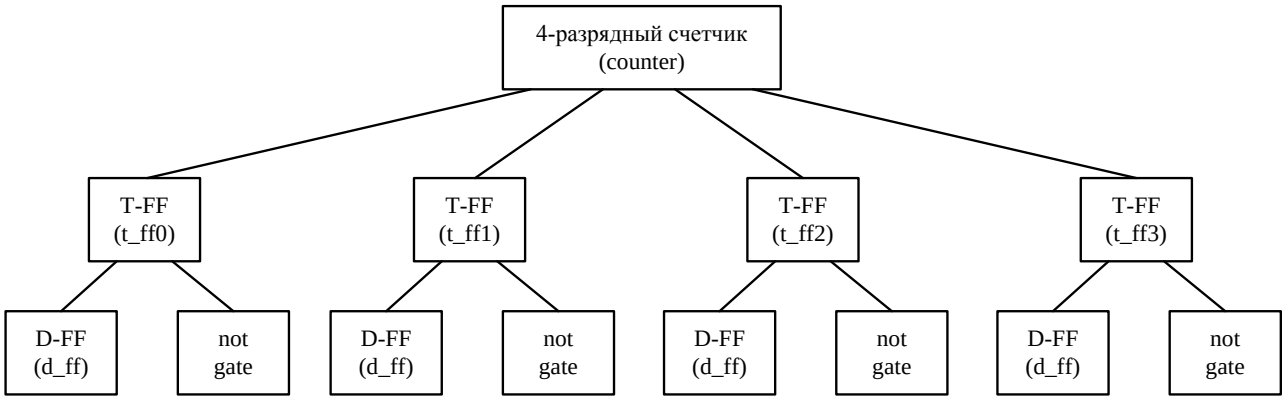


Рис. 4. — Иерархия проекта четырехразрядного счетчика

Традиционно при проектировании сложных цифровых систем используется нисходящая модель маршрута проектирования. При такой организации маршрута проектирования проект проходит различные фазы, постоянно увеличивая детализацию представления. В соответствии с нисходящим принципом проектирования сначала создается описание блока верхнего иерархического уровня, затем он разбивается на более простые блоки, а последние разделяются на еще более простые блоки и так далее до тех пор, пока декомпозиция завершится, например, на уровне вентильного представления блоков.

Следует отметить, что возможна и восходящая модель маршрута проектирования, которую мы будем использовать в нашем примере. В соответствии с этой моделью маршрута мы создадим Verilog-описание D-триггера (D-FF), который будет являться блоком нижнего иерархического уровня (low-level block). Затем мы построим T-триггер (T-FF) из D-триггера (D-FF) и инвертера not (Verilog примитива) в соответствии с рис. 3, то есть мы создадим блок более высокого иерархического уровня (T-FF). Далее, соединение четырех блоков (T-FF) в соответствии со схемой счетчика (рис. 2) в единый блок позволит нам получить Verilog-описание счетчика (counter), как модуля верхнего иерархического уровня.

2.3. Блок тестовых (испытательных) сигналов

После того как Verilog-описание проектируемого устройства будет завершено блок верхнего иерархического уровня должен быть протестирован путем подачи на его входы тестовых (испытательных) сигналов и оценки его выходных сигналов. Блок, формирующий тестовые сигналы, обычно называют stimulus block или test bench. Хорошей практикой считается, когда Verilog-описания блока испытательных сигналов и блока проектируемого устройства представлены в отдельных файлах.

В этом случае в блоке испытательных сигналов, который будем называть «test_bench», создают экземпляр блока верхнего иерархического уровня проектируемого цифрового устройства, формируют последовательности входных испытательных сигналов, и проводят проверку правильности его функционирования.

На рис. 5 показана структура модуля испытательных сигналов test_bench, который становится блоком самого высокого иерархического уровня (top-level block). Он формирует входной сигнал в виде последовательности счетных импульсов clk и сигнал асинхронного сброса reset. Для проверки правильности функционирования выходной сигнал q (состояние счетчика) должен быть определен в различные моменты времени в зависимости от количества поданных на его вход счетных импульсов clk и сигнала сброса reset.

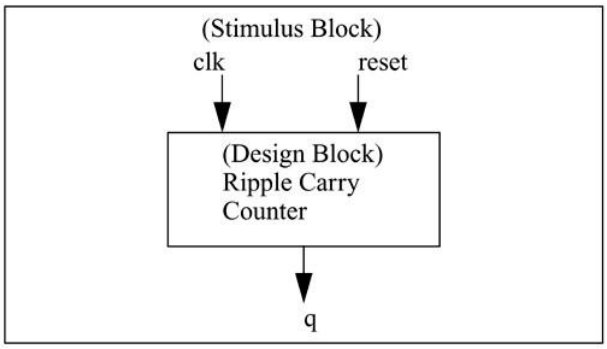


Рис. 5.— Структура блока испытательных сигналов

2.4. Verilog-описание четырехразрядного счетчика

Для иллюстрации примера моделирования цифрового устройства, рассмотренного в предыдущих параграфах, составим Verilog-описание счетчика и блока испытательных сигналов. Теперь мы воплотим иерархический подход в языке описания Verilog.

Основной структурной единицей описания цифровых устройств в Verilog является модуль **(module)**. Модуль является «строительным блоком» в Verilog. Модуль может быть самостоятельным элементом, отражающим свойства определенного блока, или совокупностью модулей более низкого иерархического уровня, каждый из которых отражает свойства соответствующего блока. Как правило, элементы группируются в модули для обеспечения общей функциональности, что позволяет их использования в различных метах описания проектируемого устройства. Модуль обеспечивает необходимую функциональность по отношению блокам более высокого иерархического уровня посредством его портов (входов и выходов), но при этом остается для них «черным ящиком», т.е. не показывает свою внутреннюю реализацию. Это позволяет проектировщику изменять внутреннее описание модуля без дополнительных изменений остальной части проекта.

Verilog является как языком описания структуры цифрового устройства, так и языком описания его поведения. Способ описания каждого модуля может быть представлен в виде четырех уровней абстракции в зависимости от задач проектирования. При любом уровне абстракции модуль ведет себя одинаково по отношению к окружающим объектам (другим модулям). Для них внутренность модуля является скрытой. Таким образом, уровень абстракции, используемый для описания модуля, может быть изменен без какого-любого изменения

описания остальных модулей. К упомянутым уровням абстракции Verilog описания относятся следующие.

Поведенческий или алгоритмический уровень (Behavioral or algorithmic level). Это самый высокий уровень абстракции, предусмотренный Verilog HDL. На этом уроне модуль может быть реализован в виде процедур и операторов, реализующих поведенческий алгоритм его функционирования без акцентирования на детали его аппаратной реализации. Проектирование на этом уровне подобно программированию на языке С.

Уровень информационного потока данных (Dataflow level). На этом уровне модуль проектируется в виде совокупности операторов, отражающих каким образом данные передаются между регистрами и как эти данные обрабатываются в проекте.

Вентильный уровень (Gate level). Модуль проектируется как совокупность логических элементов и межсоединений между этими элементами. Проектирование на этом уровне подобно описанию проекта в виде логических схем.

Уровень транзисторных ключей (Switch level). Это самый низкий иерархический уровень абстрактного описания цифровых устройств, предусмотренный Verilog HDL. На этом уровне модуль может быть реализован в виде совокупности транзисторных ключей и соединений между ними. Проектирование на этом уровне позволяет учитывать некоторые физические процессы при работе ключевых схем.

Verilog позволяет дизайнерам смешивать и согласовывать все четыре указанных уровня абстрактного описания проекта. Среди разработчиков цифровой аппаратуры сформировался термин **register transfer level (RTL)** - уровень межрегистровых передач, который широко используется при описании цифровых устройств. По сути, он представляет собой комбинацию поведенческого и потокового описания конструкций, поддерживаемую программами логического синтеза.

Например, если проект состоит из четырех модулей, то Verilog позволяет каждый из них описать при помощи различных уровней абстракций. Обычно, наивысший уровень абстракции (поведенческий) более гибкий и технологически независимый. В противоположность ему, уровень транзисторных ключей является технологически зависимым и поэтому менее гибким. На этом уровне небольшие изменения описания модуля могут вызвать необходимость вносить изменения в весь проект. В частности, D-триггер можно описать на вентильном либо уровне транзисторных ключей. Эти виды описания используют, как правило, разработчики библиотек элементов для различных технологий изготовления ИС. Мы же будем использовать поведенческое описание D-триггера.

Следует отметить, что в англоязычной литературе приняты термины «триггер» — для устройств, тактируемых фронтом, и «защелка» для устройств, тактируемых уровнем тактового импульса.

Для описания триггерных схем в Verilog используют **always**-блоки с ключевыми словами **posedge** и **negedge** для того, чтобы указать с каким фронтом сигнала связано изменение состояния триггера: с возрастающим (из лог. 0 в лог. 1 — posedge) или спадающим (из лог. 1 в лог. 0 — negedge).

Например, если для синхронного D-триггера с асинхронным сбросом его состояние будет изменяться под воздействием спадающего (заднего) фронта тактового сигнала **clk** и возрастающего (переднего) фронта сигнала сброса **reset**, то его RTL-описание будет иметь вид:

```
always @(posedge reset or negedge clk)
    begin
    if (reset)
        q <= 1'b0;
    else
        q <= d;
    end
```

В примере применен оператор ветвления **if**, который аналогичен оператору if языка C, и оператор **<=**, который носит название неблокирующего назначения (или присваивания). Смысл этого оператора заключается в том, что значение переменной «q» будет присвоено только в момент выхода из блока. Эти операторы не должны повторяться для одной и той же переменной в одной ветви алгоритма, чтобы избежать неопределенного состояния. Ветвью алгоритма называется путь, по которому алгоритм может завершиться. Попав на одну ветвь, процесс не может перейти на другую параллельную ветвь. Такими параллельными ветвями в алгоритме стали операторы присваивания под ключевым словом **if** и под словом **else**.

Следует отметить, что поведенческое описание D-триггера не учитывает временные задержки, возникающие в триггере при переключении транзисторов. Однако программы синтеза позволяют преобразовать это описание в вентильное описание с учетом задержек присущих выбранной технологии изготовления ИС.

В соответствии с иерархий проекта, показанного на рис. 4, проектируемый счетчик может быть представлен как модуль верхнего иерархического уровня **counter**. Четыре Т-триггера (**t_ff0, t_ff1, t_ff2, t_ff3**) могут быть реализованы как экземпляры модуля более низкого иерархического уровня **t_ff**. В свою очередь, модуль **t_ff** представим как соединение модуля еще более низкого иерархического уровня — D-триггера (**d_ff**) и Verilog примитива — инвертора (**not**).

Мы будем использовать восходящую модель маршрута проектирования, в соответствии с которой сначала составим Verilog-описание модуля самого низкого иерархического уровня — **d_ff** модуля, затем Т-триггера — **t_ff**. После этого, создав четыре экземпляра модуля **t_ff** и, соединив их в виде Verilog-описания, получим модуль счетчика **counter**. В завершении, для его тестирования создадим модуль **test_bench**. При этом каждый из указанных модулей будет описан в отдельном файле, имеющем название соответствующего модуля с разрешением ***.v.**

2.5. Запуск программы и создание проекта

2.5.1. Запуск программы

ModelSim SE 6.0 может быть запущен либо из стартового меню Windows, либо двойным щелчком левой клавиши мышки по ярлыку  , расположенном на рабочем столе. После запуска программы ModelSim SE 6.0 открывается основное окно программы (рис. 6).

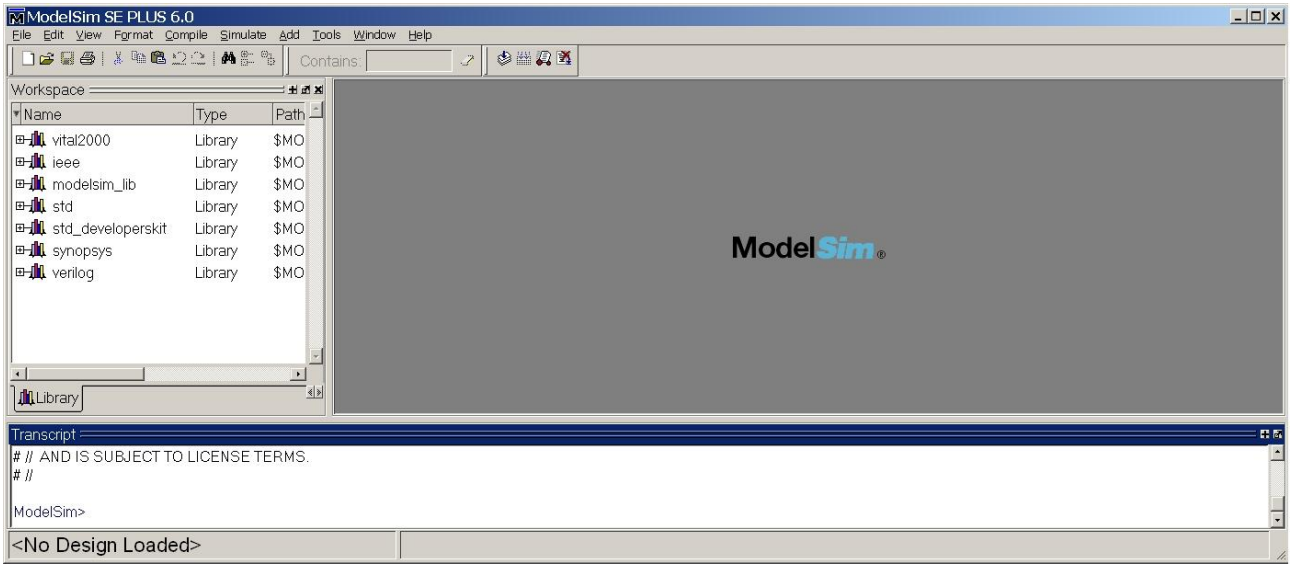


Рис. 6 — Основное окно программы ModelSim SE 6.0

2.5.2. Создание проекта

В строке меню основного окна **ModelSim** выбираем пункт **File → New → Project**. В результате появится окно **Create Project** (рис. 7).

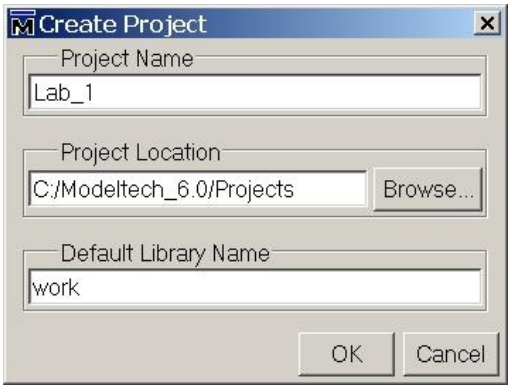


Рис. 7. — Окно создания проекта

В поле **Project Name** укажем имя проекта **Lab_1.** В поле **Project Location** укажем рабочую директорию **c:\Modeltech_6.0\Projects**, а в поле **Default Library Name** оставим **work** (название по умолчанию) в качестве названия рабочей библиотеки проекта. После нажатия клавиши **ОК** появится окно, предлагающее создать указанную директорию, если она предва-

рительно не была создана, или окно с сообщением, что указанный проект уже существует и предложением заменить его.

В результате создания проекта в директории **Projects** будет создана библиотека **work** и **Lab_1.mpf** конфигурационный файл проекта, по завершении чего откроется окно **Add Items to the Project** (рис. 8).

2.5.3. Добавление файлов в проект

В открывшемся окне **Add Items to the Project** выбираем "**Create New File**" (рис. 8).

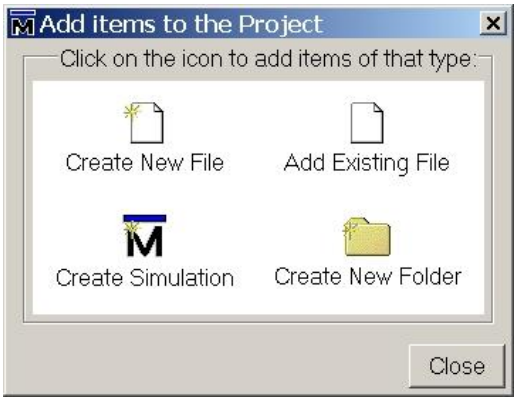


Рис. 8. — Окно добавления объектов в проект

В результате появляется окно **Create Project File** (рис. 9).

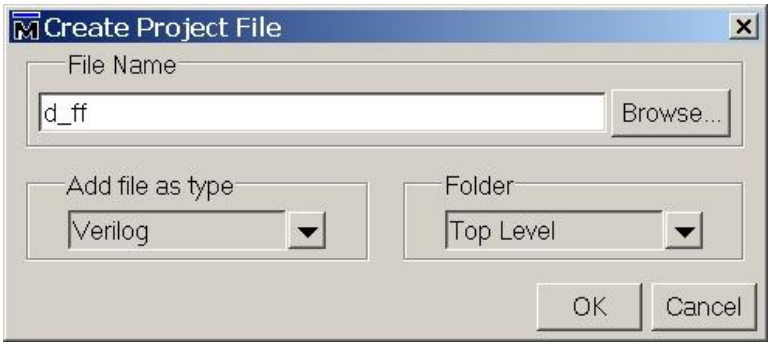


Рис. 9. — Окно создания нового файла в проекте

В поле **File Name** введем **d_ff** — название файла, в котором будет создано Verilog описание модуля нижнего иерархического уровня — D-триггера. В поле **Add file as type** выберем тип файла **Verilog,** а в поле **Folder** оставим **Top Level**, поскольку модуль более высокого иерархического уровня пока не существует.

После нажатия кнопки **OK** в рабочей директории проекта будет создан файл **d_ff.v**, а в окне **Workspace** появится его изображение.

2.5.4. Создание и редактирование файлов с Verilog описанием модулей проекта

Двойной щелчок левой клавишей мышки по имени файла «**d_ff.v**» в окне **Workspace** открывает его для редактирования в соседнем окне. Поскольку этот файл был только лишь создан, то он в открывшемся окне содержит всего две пустые пронумерованные строки. Вашей дальнейшей задачей является составление Verilog-описания указанного модуля.

В качестве примера на рис. 10 приведено окно с Verilog-описанием D-триггера, структура которого была рассмотрена в предыдущем разделе.

d_ff.v

```
ln #
 1  //////////////////////////////////////////////////////////////////////
 2  // Version    : 1.1
 3  // Module     : d_ff
 4  // Project    : Lab_1
 5  // File       : d_ff.v
 6  // Created    : 2010/02/23
 7  //       by   : Valeriy Vertegel
 8  // Language   : Verilog HDL 2001
 9  //
10  // Description: Behavioral description of the D-flip-flop with synchronous reset.
11  //
12  //
13  // Copyright (c) Sevastopol National Technical University
14  //
15  //////////////////////////////////////////////////////////////////////
16
17
18  `timescale 1ns/1ns
19
20  module d_ff
21  (
22     d,             // data input
23     clk,           // clock pulse input
24     reset,         // synchronous reset input
25     q              // data output
26  );
27
28     input d;
29     input clk;
30     input reset;
31     output q;
32
33     reg q;
34
35     // Behavioral description of the D-flip-flop
36  always @(posedge reset or negedge clk)
37     if (reset)
38        q <= 1'b0;
39     else
40        q <= d;
41
42  endmodule
```

Рис. 10 — Окно редактора с Verilog описанием D-триггера

Повторите это описание в окне для редактирования **d_ff.v** файла Вашего проекта, соблюдая синтаксис языка Verilog, принятые правила записи идентификаторов, операторов, строковых комментариев, отступов, табуляции и пробелов. Не забывайте периодически сохранять файл, чтобы избежать его потерю в результате ошибочных действий при наборе.

Следующим модулем, входящим в структуру проектируемого счетчика, является Т-триггер — модуль более высокого иерархического уровня. Для его Verilog описания создадим файл **t_ff.v** и включим его в проект проектируемого счетчика. Для этого выберем в пункте меню основного окна ModelSim пункт **File → Add to Project → New File..**

В результате появляется окно **Create Project File** (рис. 11).

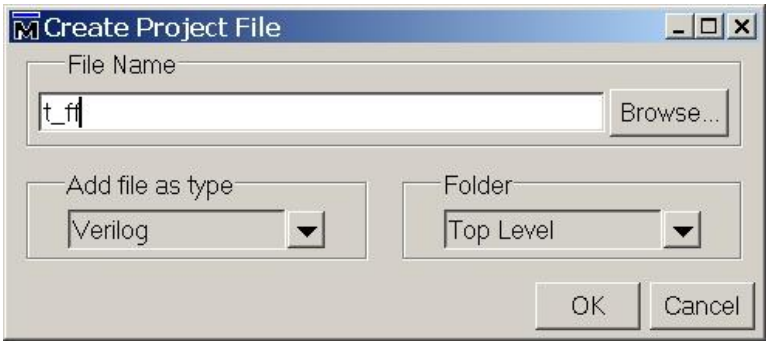


Рис. 11. — Окно добавления нового файла в проекте

В поле **File Name** введем **t_ff** — название файла, в котором будет создано Verilog описание модуля Т-триггера. В поле **Add file as type** выберем тип файла **Verilog,** а в поле **Folder** оставим **Top Level**, поскольку это модуль более высокого иерархического уровня.

Аналогично поступим, добавляя в проект файлы с Verilog описаниями оставшихся модулей: **counter.v** и **test_bench.v.** В результате последовательного добавления указанных файлов в проект они будут отображаться в окне **Workspace**, как показано на рис. 12.

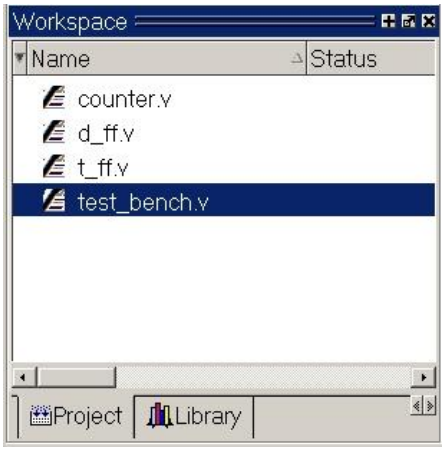


Рис. 12. — Окно Workspace, содержащее Verilog файлы проекта

Двойной щелчок левой клавишей мышки по имени файла в окне **Workspace** открывает каждый из файлов в отдельной закладке для редактирования в текстовом редакторе.

Дальнейшей задачей является составление Verilog-описаний всех модулей включенных в проект. В качестве примера воспользуйтесь описанием модулей на рис. 13, 14, 15.

t_ff.v*

```
ln #
 1  //*****************************************************************
 2  // Version     : 1.1
 3  // Module      : t_ff
 4  // Project     : Lab_1
 5  // File        : t_ff.v
 6  // Created     : 2010/02/23
 7  //       by    : Valeriy Vertegel
 8  // Language    : Verilog HDL 2001
 9  //
10  // Description: Define the module t_ff.
11  //              It instantiates a D-flipflop and inverter not.
12  //              We assumed that module D-flipflop is defined
13  //              by separated module.
14  //
15  // Copyright (c) Sevastopol National Technical University
16  //
17  //*****************************************************************
18
19  `timescale 1ns/1ns
20
21  module t_ff
22  (
23     clk,
24     reset,
25     q
26  );
27
28     input clk;
29     input reset;
30     output q;
31
32     wire d;    // wire which connects not get output with d input
33                // of the D flip-flop
34
35  // Instantiation of d_ff.
36  // ------------------------------------------------------------------
37     d_ff        i_dff0
38  // ------------------------------------------------------------------
39  (
40     .d          (d),
41     .clk        (clk),
42     .reset      (reset),
43     .q          (q)
44  );
45
46     not i_not (d, q);    // not gate is Verilog primitive.
47
48  endmodule
```

Рис. 13 — Окно редактора с Verilog-описанием Т-триггера

counter.v

```
ln #
 1  //*********************************************************************
 2  // Version    : 1.1
 3  // Module     : counter.v
 4  // Project    : Lab_1
 5  // File       : counter.v
 6  // Created    : 2010/02/23
 7  //      by    : Valeriy Vertegel
 8  // Language   : Verilog HDL 2001
 9  //
10  // Description: Ripple carry counter  behavioral model.
11  //              Define the top-level module called ripple carry counter.
12  //              It instantiates 4 T-flipflops.
13  //
14  // Copyright (c) Sevastopol National Technical University
15  //
16  //*********************************************************************
17
18  `timescale 1ns/1ns
19
20  module counter
21  (
22     clk,
23     reset,
24     q
25  );
26
27     input  clk;       // input clock signals
28     input  reset;     // input reset signals
29     output [3:0] q;   // output signals
30
31  // Four instances of the module t_ff are created.
32  // Each has a unique name (i_tff0, i_tff1, i_tff2, i_tff3).
33  // Notice, that each instance is a copy of the module t_ff.
34
35  // ---------------------------------------------------------------
36     t_ff        i_tff0
37  // ---------------------------------------------------------------
38  (
39     .clk        (clk),
40     .reset      (reset),
41     .q          (q[0])
42  );
43
44  // ---------------------------------------------------------------
45     t_ff        i_tff1
46  // ---------------------------------------------------------------
47  (
48     .clk        (q[0]),
49     .reset      (reset),
50     .q          (q[1])
51  );
52
53  // ---------------------------------------------------------------
54     t_ff        i_tff2
55  // ---------------------------------------------------------------
56  (
57     .clk        (q[1]),
58     .reset      (reset),
59     .q          (q[2])
60  );
61
62  // ---------------------------------------------------------------
63     t_ff        i_tff3
64  // ---------------------------------------------------------------
65  (
66     .clk        (q[2]),
67     .reset      (reset),
68     .q          (q[3])
69  );
70
71  endmodule
72
```

Рис. 14 — Окно редактора с Verilog-описанием 4-разрядного счетчика

test_bench.v

```
ln #
  1  //*********************************************************************
  2  // Version    : 1.1
  3  // Module     : test_bench
  4  // Project    : Lab_1
  5  // File       : test_bench.v
  6  // Created    : 2010/02/23
  7  //      by    : Valeriy Vertegel
  8  // Language   : Verilog HDL 2001
  9  //
 10  // Description: Stimulus module instantiates the design block
 11  //              and directly drives the signals in the design block.
 12  //
 13  // Copyright (c) Sevastopol National Technical University
 14  //
 15  //*********************************************************************
 16
 17  `timescale 1ns/1ns
 18
 19  module test_bench;
 20
 21     reg clk;
 22     reg reset;
 23     wire[3:0] q;
 24
 25  // instantiate the design block
 26  // ---------------------------------------------------------------
 27     counter      i_counter
 28  // ---------------------------------------------------------------
 29  (
 30     .clk         (clk),
 31     .reset       (reset),
 32     .q           (q)
 33  );
 34
 35  // ---------------------------------------------------------------
 36  // Clock generator
 37  // Control the clk signal that drives the design block.
 38  // Cycle time = 10 time units.
 39
 40  initial
 41     begin
 42     clk = 0;
 43     forever #5 clk = !clk;
 44     end
 45
 46  // ---------------------------------------------------------------
 47  // Control the reset signal that drives the design block.
 48  // Reset is asserted from 0 to 20 and from 200 to 220.
 49
 50  initial
 51     begin
 52     reset = 1'b1;
 53     #20 reset = 1'b0;
 54     #180 reset = 1'b1;
 55     #10 reset = 1'b0;
 56     #20 $stop;     //stop the simulation
 57     end
 58
 59  // ---------------------------------------------------------------
 60  // Monitor the outputs
 61  initial  $monitor($time, " Output q = %d",  q);
 62
 63  endmodule
```

Рис. 15 — Окно редактора с Verilog-описанием тестового модуля

2.6. Компиляция файлов проекта

Следующим этапом в маршруте моделирования цифрового устройства в системе ModelSim является компиляция файлов проекта. Для этого необходимо установить параметры компилятора и переопределить тип файлов, используемых в проекте. Выберете в окне **Workspace** первый файл из списка, затем нажав правую клавишу мышки, в раскрывающемся меню выберите пункт **Properties…**. В открывшемся окне **Project Compiler Settings** нажмите кнопку **Change Type** и в открывшемся окне **Change Type** выберите тип файла **Verilog**, отключите опцию **Do Not Compile** и выберите **Compile to library: work**. После нажатия клавиши ОК в окне **Workspace** в столбце **Status** появится вопросительный знак голубого цвета.

Повторите указанные действия последовательно для всех Verilog-файлов проекта.

Для компиляции файлов проекта в строке меню основного окна **ModelSim** выбираем пункт **Compile → Compile All**.

Если файлы проекта не содержат ошибки, то после компиляции в окне **Transcript** выдается сообщение.

```
# Compile of test_bench.v was successful.
# Compile of counter.v was successful.
# Compile of d_ff.v was successful.
# Compile of t_ff.v was successful.
# 4 compiles, 0 failed with no errors.
```

Если же файлы проекта содержат ошибки, то после компиляции об этом выдается сообщение, например,

```
# Compile of d_ff.v failed with 1 errors.
```

Двойной щелчок по сообщению об ошибке в окне **Transcript** открывает новое окно, в котором указывается модуль, содержащий ошибку, строка Verilog-кода (в скобках) и причина этой ошибки (рис. 16).

Рис. 16 — Окно результатов компиляции с указанием ошибки

Двойной щелчок по сообщению об ошибке в этом окне выдает строку Verilog-кода, в которой содержится ошибка.

После исправления ошибки файл, её содержащий, должен быть сохранен и заново перекомпилирован. Индикаторами результатов успешной компиляции файлов проекта будут «галочки» зеленого цвета в столбце **Status** окна **Workspace**, как это показано на рис. 17.

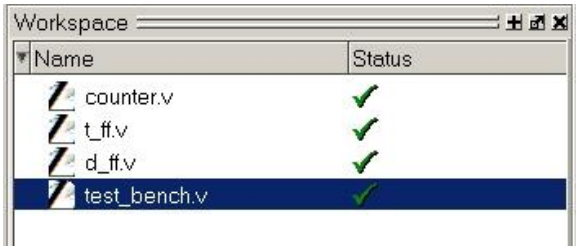


Рис. 17. — Окно Workspace с результатами успешной компиляция файлов проекта

2.7. Моделирование проекта

2.7.1. Задание параметров моделирования

Следующим этапом в маршруте проектирования цифрового устройства в системе ModelSim является моделирование проектируемого устройства. Для этого в строке меню основного окна **ModelSim** выбираем пункт **Simulate → Runtime Options**. В результате появится окно **Runtime Options**, позволяющее задать параметры моделирования (рис. 18).

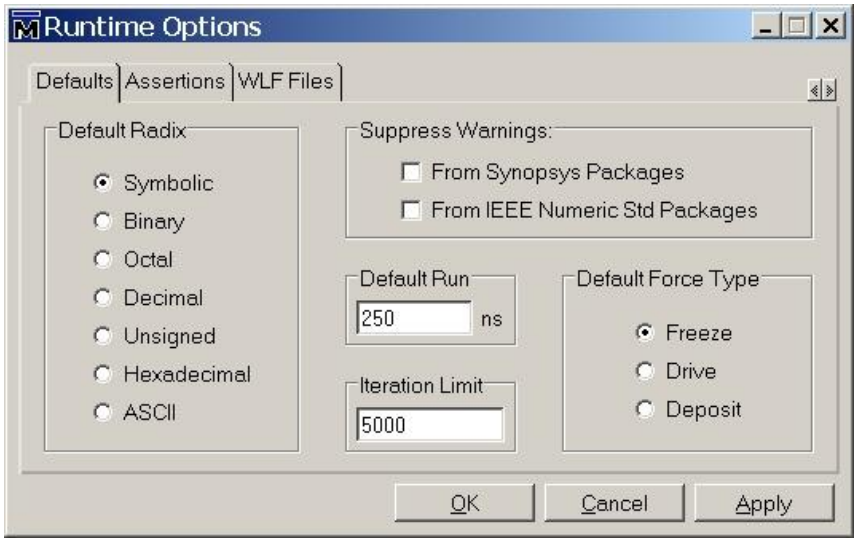


Рис. 18. — Окно задания параметров моделирования

В нем необходимо установить:

Default Radix → Symbolic (формат представления данных);

Default Run → 250 ns (время одного шага моделирования при пошаговом моделировании);

Iteration Limit → 5000 (число событий моделирования, событие — изменение сигнала).

Остальные параметры — по умолчанию, нажать **OK** (рис. 18).

2.7.2. Запуск моделирования

Для запуска моделирования проектируемого устройства необходимо в строке меню основного окна **ModelSim** выбираем пункт **Simulate → Start Simulation**. В результате откроется окно **Start Simulation** (рис. 19).

В этом окне необходимо раскрыть библиотеку **work**, нажав «+» слева от символа этой библиотеки и выбрать модуль верхнего иерархического уровня. В нашем примере этим модулем является **test_bench**. Нажать кнопку **ОК**.

В результате этого в окно **Transcript** будет выведена информация об успешном подключении объектов из библиотеки **work,** в закладке **Sim** окна **Workspace** будут отображены все экземпляры модулей, а в основном окне **ModelSim** появится окно **Objects,** содержащее сигналы **clk, reset, q** (рис. 20).

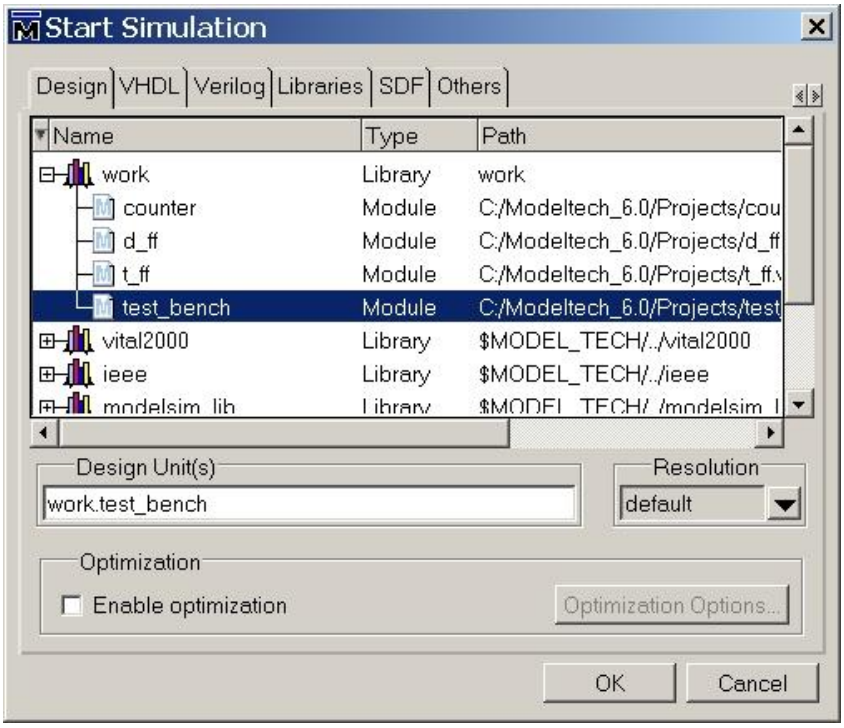


Рис. 19. — Окно Start Simulation с выделением модуля верхнего иерархического уровня

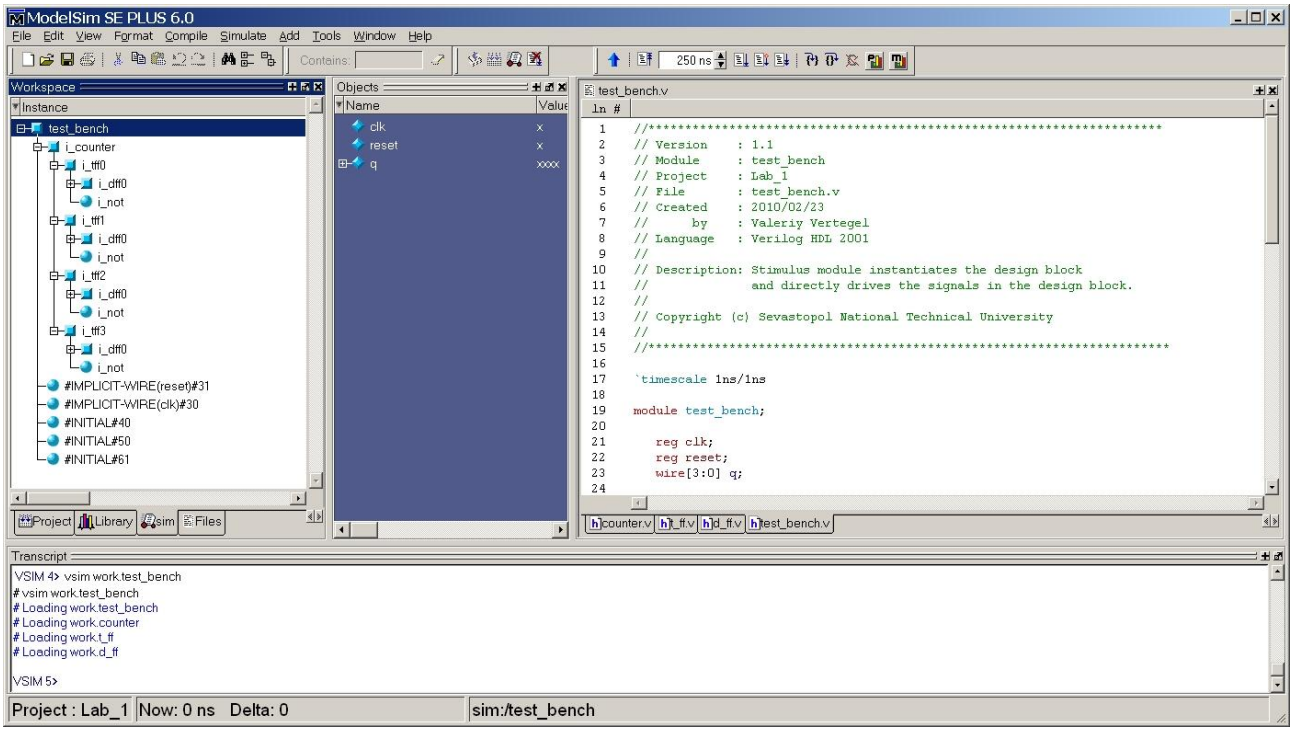


Рис. 20. — Основное окно ModelSim после начала моделирования

2.7.3. Отображение сигналов на временных диаграммах

Для отображения сигналов проектируемого устройства на временных диаграммах используется окно **Wave**. Это окно может быть открыто командой **View → Debug Windows → Wave**. Для отображения желаемых сигналов необходимо в окне **Objects** в раскрывающемся меню выбрать **Add → Wave → Signals in Region**. В результате все сигналы из окна **Objects** добавятся в окно **Wave**. При добавлении сигналов окно Wave, появятся соответствующие команды в окне **Transcript**. Это есть другой способ перемещения сигналов из окна **Objects** в окно **Wave**.

Открытое окно Wave, с выбранными для отображения сигналами, показано на рис. 21.

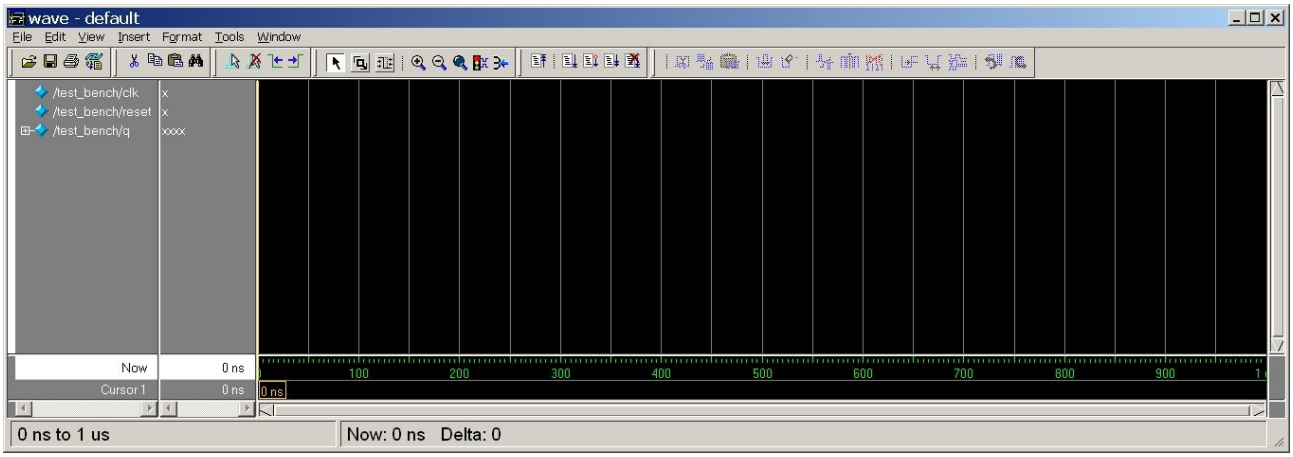


Рис. 21. — Окно Wave с выбранными для отображения сигналами

2.7.4. Отображения сигналов в окне Wave

Для отображения сигналов в окне **Wave** выберите в основном окне **ModelSim** выберите пункт меню **Simulate → Run → Run 250 ns** или нажмите кнопку (Run All). Симуляция будет выполнена для значения по умолчанию (250 ns). Результаты моделирования будут отображены в окне **Wave** в виде графиков выходных сигналов. Для изменения масштаба временных диаграмм, воспользуйтесь кнопками панели инструментов.

Окно **Wave** с диаграммами входных и выходного сигналов счетчика в измененном масштабе показано на рис. 22.

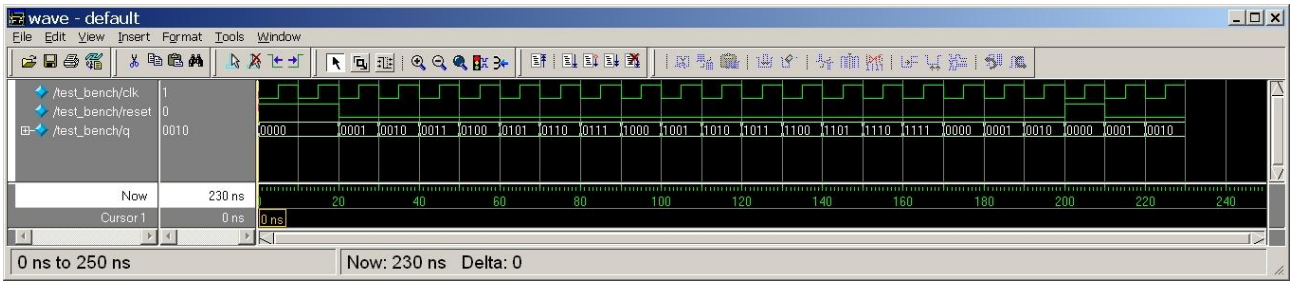


Рис. 22. — Диаграммы входных и выходного сигналов счетчика

По умолчанию формат отображения сигнала «q» в окне **Wave** является двоичным. Для того чтобы изменить этот формат на шестнадцатеричный, выберите в окне **Wave** строку соответствующую этому сигналу, и, нажав правую клавишу мышки, в открывшемся меню выберите пункт **Radix → Hexadecimal**.

Нажав на знак «+» слева от сигнала «**q**» можно наблюдать выходные сигналы **q[3], q[2], q[1], q[0]** каждого из триггеров, входящих в состав проектируемого счетчика. При этом следует отметить, что частота следования импульсов на выходе каждого Т-триггера в два раза ниже, чем частота следования импульсов на его входе. Поэтому такие счетчики еще называют «счетчиками-делителями частоты».

Из диаграмм также видно, что счетчик изменяет свое состояние по заднему фронту тактового импульса **clk** при значении сигнала сброса **reset = 0.** Асинхронный сброс счетчика при **reset** = 1 обнуляет значение выходного сигнала **q,** что обусловлено его Verilog-описанием.

Об этом также свидетельствует окно **Transcript** (рис. 23), в которое в процессе моделирования выводилась информация о времени моделирования и значение состояния счетчика **q** в десятичной системе счисления.

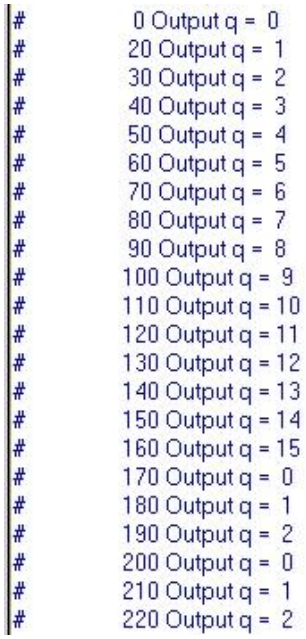


Рис. 23. — Информация, выводимая в окно Transcript в процессе моделирования

Следует обратить внимание, что перезапуск моделирования может привести к бесконечному циклу. Остановить бесконечное (зациклившиеся) моделирование можно командой **break** в консольной области (окно Transcript), либо командой главного меню **Simulate → Break**, либо нажатием кнопки  (**Break**) в панели инструментов окна Wave. В случае цикла текущее время моделирования непрерывно увеличивается. Текущее время выводится внизу окна Wave.

При выходе из системы моделирования в директории work будут храниться скомпилированные во внутреннее представление файлы проекта. К тому же в рабочей директории будут сохранены системный файл Lab_1.mpf сохраненного проекта; системный файл vsim.wlf временной диаграммы (двойной щелчок вызовет систему моделирования и откроет окно wave с сохраненной временной диаграммой), а также созданные нами Verilog файлы модулей.

При завершении работы в системе ModelSim директория проекта запоминается и в следующем сеансе можно начать работу с проектом, выполнив команду **File → Recent Directories** и выбрав необходимую рабочую директорию, либо выполнив команду **File → Recent Project**, выбрав ранее сохраненный проект