

Министерство цифрового развития, связи и массовых коммуникаций
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

Кафедра инфокоммуникационных систем и сетей

ЛАБОРАТОРНАЯ РАБОТА 2

по дисциплине «Прототипирование телекоммуникационных систем»

**«Разработка и отладка параметризованного
модуля двоичного счетчика с разрешением счета»**

Мелентьев О.Г.

Методические указания
к лабораторной работе

Новосибирск, 2025

Цель работы:

Изучение основных этапов проектирования и тестирования модуля в ModelSim на примере двоичного счетчика с разрешением счета и изменяемой разрядностью.

Задание:

- 1) Ознакомиться с основными понятиями языка описания аппаратуры System Verilog.
- 2) Создать по техническому заданию стандартный модуль счетчика module counter_enbl с настраиваемым параметром, сделать листинг модуля проекта, подготовить соответствующий файл.

Техническое задание:

parameter N - разрядность счетчика;	
Входы: clk - тактовый вход; reset – сброс; enable - разрешение счета	Выход: N – разрядная шина значения счетчика.
Логика работы счетчика.	
По переднему фронту тактового импульса, если reset = 1, обнуляем счетчик, иначе если enable = 1, увеличиваем значение счетчика на 1.	

- 3) Изучить программу тестирования счетчика в ModelSim, листинг module cnt_en_testbench и подготовить соответствующий файл.
- 4) Выполнить симуляцию в ModelSim для 8 разрядного счетчика.

Отчет должен содержать:

- цель работы;
- код программы в виде скриншота;
- эпюры, полученные вследствие симуляции и тестирования;
- сделанные выводы о полученных результатах.

Методические указания

1. Ознакомиться с основными понятиями языка описания аппаратуры System Verilog, необходимыми для выполнения лабораторной работы.

- Существует четыре типа значений, которые может принимать сигнал – как wire, так и reg:
 - ✓ 0 – логический ноль;
 - ✓ 1 – логическая единица;
 - ✓ z – высокоимпедансное логическое состояние — состояние выхода логического устройства, при котором он обладает высоким сопротивлением (импедансом), то есть фактически отключен от подсоединённого к нему

проводника (относится к несинтезируемой части языка);

✓ x – неопределенное состояние (относится к несинтезируемой части языка);

Логические значения `0` и `1`, которые могут принимать сигналы wire и reg, относятся к типу logic (логический тип данных). Поэтому при объявлении переменной можно не выбирать между reg и wire, а использовать logic. Тип данных logic указывает на то, что *компилятор сам определяет*, что это проводник wire или переменная reg.

Тип сигнала не является раз и навсегда установленной величиной, а может меняться от модуля к модулю.

- **Блокирующие и неблокирующие присвоения.**

В Verilog *существует два типа процедурных присвоений*:

✓ **блокирующее присвоение** – происходит строго в порядке объявления, выполнение последующего присвоения не начинается, если не завершилась процедура вычисления текущего процесса

- $a = b$;

✓ **неблокирующее присвоение** – выполняется независимо от порядка расположения операций

- $a <= b$;

- **Создание экземпляра модуля:**

Прежде чем использовать модуль в качестве части цифровой системы, следует создать экземпляр (instance) модуля. Экземпляр модуля представляет некоторый аналог вызова функции в языках программирования. Но если в языках программирования вызов функции соответствует обращению к части кода программы, то создание экземпляра модуля соответствует установке на плате экземпляра микросхемы определенного номинала (модуля).

Для создания экземпляра модуля в коде другого модуля используется следующий синтаксис:

< имя модуля > < имя экземпляра > (< список портов >);

Подобно тому, как после установки микросхемы на плате к ее выводам необходимо подсоединить линии с передаваемыми сигналами, при создании экземпляра модуля следует указать сигналы, подсоединяемые к портам модуля. Имеется два способа указания подсоединения сигналов к портам экземпляра модуля: по местоположению и по имени порта (таб.1).

Таблица 1 – Способы подсоединения сигналов к портам модуля

module primer	Создание экземпляра модуля с подсоединением сигналов по местоположению	Создание экземпляра модуля с подсоединением сигналов по имени порта
<pre>module primer (input x, output y); wire a = x; assign y = ~ a; endmodule</pre>	<pre>primer position (X, Y);</pre>	<pre>primer port (.x(X), .y(Y));</pre>
	Порядок записи сигналов в экземпляре соответствует порядку записи сигналов в модуле	<p>или</p> <pre>primer port (.y(Y), . x(X));</pre> <p>Сигналы в экземпляре можно записывать в произвольном порядке</p>

Передача сигналов по имени порта требует больше символов кода, однако освобождает разработчика от знания порядка следования портов в объявлении модуля. Кроме того, данный способ уменьшает количество ошибок при написании кода проекта. Поэтому он используется при создании экземпляров модулей с большим количеством портов.

- **Параметры.** Часто необходимо создать стандартный модуль, *который можно настроить несколькими параметрами*, когда для этого модуля создается экземпляр (инстанциация модуля).

Ключевое слово *parameter* применяется для введения и назначения величины, которая не меняется в данной редакции текста, но может измениться в новой редакции.

Параметры в Verilog представляют собой константы, они могут быть изменены во время компиляции для того, чтобы принять новые значения. Это позволяет пользователю настраивать экземпляры (instance) модулей при их установке в проекте.

При создании модуля необходимо задать первоначальную величину параметра:

```
module X # (parameter N=10)
```

При установке в проект экземпляра модуля параметр настраивается заново:

```
X # (.N(param)) X1 // имя модуля # (назначаемый параметр) имя экземпляра
```

- Операторы:

Операторы сравнения	
используются для сравнения значений и возвращают булево значение `1` (истина) или `0` (ложь).	
<code>==</code>	равно
<code>!=</code>	не равно
<code>></code>	больше
<code><</code>	меньше
<code>>=</code>	больше или равно
<code><=</code>	меньше или равно

Логические операторы	
используются для выполнения логических операций над булевыми значениями	
<code>&&</code>	логическое И
<code> </code>	логическое ИЛИ
<code>!</code>	логическое НЕ

- Тестовый модуль **TestBench** и его особенности

Наряду с выполнением основной функции - описанием тестов *TestBench* может являться оболочкой проекта.

Первая строка содержит директиву масштаба модельного времени, которая задается в начале файла Verilog строкой вида:

``timescale 1ns/1ps`

1ns означает **шкалу времени** для симулятора, а **1ps** означает точность расчета времени для него. Единицы времени, указанные здесь, применяются при создании тестовых сигналов, подаваемых на вход тестируемой в симуляторе схемы. Оба параметра указываются в нано или пикосекундах и могут принимать значения только 1, 10 или 100. По умолчанию шкала времени назначается 1 ps.

В некоторых версиях моделирующих программ такая строка должна содержаться во всех файлах перед модулем.

В следующей строке тестового модуля указывается ключевое слово *module* и имя модуля с обязательной приставкой *testbench*, указывающей на то, что это тестовый модуль проекта. В этом случае список портов отсутствует.

Далее указывается **характер поведения выходов и входов** (*reg* или *wire*).

Моделирование осуществляется с помощью конструкций, определяемых ключевыми словами *initial begin...end*. Конструкции содержат описания начальных значений и дальнейшего поведения входных сигналов во времени.

Тестовый модуль без блока *initial* не работает.

- **Блок инициализации (*initial*)**

С помощью блока *initial* осуществляется собственно моделирование устройства. Блок инициализации включает в себя оператор или группу операторов, которые будут выполняться с момента старта моделирования. В первом случае конструкция блока имеет вид:

initial<оператор>;

а во втором:

initial begin

<группа операторов>;

end

Ключевые слова *begin* и *end* называются операторными скобками. Группы операторов всегда заключаются в операторные скобки. Так как большие проекты, как правило, редактируются и дополняются, то предпочтительнее использовать второй вариант.

Блок *initial* применяется в основном для явного назначения переменных (процедурное назначение), поэтому он используется главным образом для задания входных воздействий (в том числе начальных значений), т.е. при формировании тестов. Следует помнить, что переменные в этом случае должны быть объявлены как *reg*.

Все блоки *initial*, присутствующие в Verilog-проекте, выполняются одновременно, один раз в начале моделирования.

- **Always-блок**

Блок *always* означает, что действие, указанное в нем, выполняется до тех пор, пока процесс моделирования (или симуляции) не будет остановлен.

Блок *always* применяется только с данными типа *reg*. Отсюда следует и обратное положение. Если требуется использовать блок *always*, например, для определения значений выходов сумматора *s* и *p*, то их надо объявить, как данные типа *reg*.

В Verilog используется несколько конструкций с ключевым словом *always*.

Наиболее частое употребление имеет конструкция с так называемым списком чувствительности.

Конструкция имеет следующий формат:

always @ (<имя переменной 1> or <имя переменной 2> or...)

Эта запись означает, что указанные за ней внутри блока действия выполняются всегда при изменении хотя бы одной переменной (здесь - операнда). Список операндов для блока *always* называется списком чувствительности. Вместо *or* в конструкции можно использовать запятую. В конце данной конструкции точка с запятой не указывается.

Примеры записи рассмотренной конструкции:

always @ (in1 or in2 or in3 or in4)

always @ (data)
always @ (posedge clk)

Последняя запись применяется для описания устройств с динамическим управлением записью. При управлении переходом 0→1 применяется ключевое слово ***posedge***, при управлении переходом 1→0 - ключевое слово ***negedge***. Здесь речь идет об управлении устройством фронтом (или спадом) тактового сигнала. Если у одной переменной ставится *posedge* (или *negedge*), то и у всех переменных из списка чувствительности должно стоять *posedge* (или *negedge*).

Например:

always @ (posedge clk, negedge reset)

При вычислениях, которые должны производиться при изменении любой из используемых в блоке переменных, Verilog позволяет не перечислять имена списка, достаточно указать символ обобщенного списка чувствительности:

*always @ **

Если в вычислениях участвует одна переменная, то блок *always* применяется без списка чувствительности.

Например:

always # 10 clk =~ clk;

- **Операторы задержки.** Симуляция аппаратуры происходит в дискретные моменты времени. Чтобы выполнить действия в симуляции в определенном порядке, нам часто требуется представить генерируемые входные сигналы разными в разные моменты времени, вместо того, чтобы производить их одновременно. Для этого мы используем операторы задержки:

Общий синтаксис оператора:

#< n >, здесь n задает задержку в единицах времени.

#5; // задержка в 5 единиц времени

#100; // задержка в 100 единиц времени

2. Изучить техническое задание на создание стандартного модуля счетчика *module counter_enb1* с настраиваемым параметром и листинг модуля проекта, подготовить соответствующий файл.

Техническое задание:

parameter N - разрядность счетчика;	
Входы: clk - тактовый вход; reset – сброс; enable - разрешение счета	Выход: N – разрядная шина значения счетчика.
Логика работы счетчика.	
По переднему фронту тактового импульса, если reset = 1, обнуляем счетчик, иначе если enable = 1, увеличиваем значение счетчика на 1.	

Листинг модуля проекта, соответствующий техническому заданию:

```

module counter_enbl # (parameter N=5) // объявляем значение параметра равным 5
(
input clk,                // входы по умолчанию имеют тип wire
input reset,
input enable,
output reg [N-1:0] counter // объявляем тип выхода  шина reg
);
    always @(posedge clk) begin // по положительному фронту генератора clk
        if (reset) begin // если reset=1
            counter <= 0; // сброс счетчика в 0
        end
        else if (enable) begin // в противном случае если enable=1
            counter <= counter + 1; // увеличиваем значение счетчика на 1
        end
    end
end
endmodule

```

3. Изучить листинг *module trigger_x* и подготовить соответствующий файл.

Создать триггер активного состояния, который устанавливает на выходе значение set =1 и сбрасывает его в ноль при reset=1 (некая разновидность D - триггера).

Листинг module trigger_x - триггера активного состояния:

```
module trigger_x
  (input clk,
   input reset,
   input set,
   output reg out
  );
  always @(posedge clk) begin
    if (reset) begin
      out <= 0;
    end else begin
      if (set) begin
        out <= 1;
      end
    end
  end
end
endmodule
```

4. Изучить программу тестирования счетчика в ModelSim, листинг module cnt_en_testbench и подготовить соответствующий файл.

Перед тем, как пробовать логику на реальном устройстве, следует удостовериться, что она функционирует корректно. Проверка делается с помощью написания на Verilog кода теста (testbench) и применения программного обеспечения симуляции, чтобы запустить тест.

Программное обеспечение для моделирования позволяет упростить тестирование HDL-кода. Оно позволяет разработчику анализировать прохождение сигналов и помогает находить ошибки в проекте, не загружая конфигурацию в микросхему ПЛИС.

Программа тестирования двоичного счетчика с разрешением счета и изменяемой разрядностью на примере 5-разрядного экземпляра	
1.	Подать на вход счетчика тактовый сигнал
2.	Обнулить содержимое переменных
3.	Сформировать сигнал сброса счетчика в ноль после третьего значения тактового сигнала
4.	Сформировать сигнал разрешения работы счетчика после пятого значения тактового сигнала
5.	Пауза (остановка счета) на 10 и 15 тактовом импульсе

Листинг модуля тестирования:

//Тестируем счетчик с разрешением счета

```

/^ timescale 1ns / 100ps
module cnt_en_testbench;

localparam cnt_width = 4;

reg clk;
reg [cnt_width:0] cnt_taktov;

logic enable;
logic [cnt_width-1:0] out_counter;

initial begin
    clk = 0;
    cnt_taktov = 0;
end

always
    #10 clk = ~ clk;

always @(posedge clk) begin
    cnt_taktov <= cnt_taktov + 1;
end

// Управляющие сигналы
wire reset = (cnt_taktov == 3);
wire start = (cnt_taktov == 5);
wire active;

wire pause = (cnt_taktov == 10) || (cnt_taktov == 15);

//Триггер активного состояния
trigger_x tr (
    .clk(clk), .reset(reset),
    .set(start), .out(active));

assign enable = active & ~ pause;

//Тестируемые модули
counter_enbl # ( .N(cnt_width)) cnt_inst
(
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .counter(out_counter)
);

endmodule

```

5. Симуляция в ModelSim для 8 разрядного счетчика.

Выполнить следующие этапы симуляции:

1. Создать в индивидуальную папку для проекта и поместить в нее следующие файлы: counter_enbl.sv (тестируемый модуль), cnt_en_testbanch.sv (оболочка для тестирования) и trigger_x.sv (вспомогательный модуль).

Вышеназванные файлы можно создать в программе Visual Studio code, для этого надо:

- ✓ Запустить программу Visual Studio code
- ✓ Создать новый файл (**File → New Text File**) или (**Ctrl + N**)
- ✓ На рабочем поле создать текст программы и сохранить файл с расширением sv (**File → Save as**) или (**Ctrl + Shift + S**)
- ✓ Закрыть файл (**Ctrl + F4**)

Кроме того, файлы можно создавать в программе Блокнот.

2. Открыть программу ModelSim и создать новый проект:

- ✓ Выберите пункт меню **File → New → Project** или воспользуйтесь быстрым стартом **Jumpstart → Create a Project**
- ✓ В окне **Create Project** задайте имя проекта (Project Name) **cnt_en_testbanch**, выберите индивидуальную папку (Project Location), имя библиотеки по умолчанию (Default Library Name) **work**, дальнейшие настройки (Copy Setting Form) оставьте без изменения и нажмите ОК.
- ✓ В окне **Add items to the Project** нажмите на **Add Existing File**, выберите свою индивидуальную папку и добавьте файлы counter_enbl.sv cnt_en_testbanch.sv, trigger_x.sv в проект (выбрано **Reference from current location**).

3. Выполнить компиляцию проекта (**Compile → Compile All**). В случае наличия ошибок исправить их и выполнить компиляцию заново.

4. Перейти к процессу симуляции

(**Simulate → Start Simulation → work → cnt_en_testbanch.sv**)

5. Добавить сигналы, которые необходимо посмотреть в процессе симуляции (**Add → to wave → all item in region**)

6. Открыть вкладку **Wave** и запустить симуляцию

(**Simulate → Run → Run -All**)

6. Проверить правильность работы проекта.

Чтобы выполнить симуляцию проекта с новыми параметрами, сбросьте результаты предыдущей симуляции (**Simulate → Break**) и выполните пункты 3 – 7 заново.