

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

## **КОНТРОЛЬНАЯ РАБОТА**

по дисциплине «Прототипирование телекоммуникационных систем»

**«Реализация сдвиговых регистров с последовательным и параллельным  
входами»**

Мелентьев О.Г.

Методические указания

Новосибирск, 2025

### **Цель работы:**

Реализация и тестирование параметризованных модулей сдвиговых регистров с последовательным и параллельным входами.

### **Задание:**

- 1) Ознакомиться с основными понятиями языка описания аппаратуры System Verilog.
- 2) Создать по техническому заданию стандартный модуль сдвигового регистра с последовательным входом и параллельным выходом, сделать листинг модуля проекта, подготовить соответствующий файл.

Техническое задание:

parameter M - разрядность регистра;	
Входы: clk - тактовый вход; reset – сигнал синхронного сброса ячеек регистра; bit_in – последовательный вход в старший разряд; shift – сигнал записи в старший разряд входного бита со сдвигом содержимого в сторону младшего разряда.	Выход: byte_out – M разрядная шина значения регистра
Логика работы регистра	
По переднему фронту тактового импульса если reset = 1 регистр обнуляется, иначе если shift = 1 производим сдвиг и записываем вход в старший разряд.	

- 3) Создать по техническому заданию стандартный модуль сдвигового регистра с параллельным входом и последовательным выходом из младшего разряда, создать листинг модуля проекта, подготовить соответствующий файл.

Техническое задание:

parameter M - разрядность регистра;	
Входы: clk - тактовый вход; reset – сигнал синхронного сброса ячеек регистра; bus_in – входная M разрядная шина; set – сигнал записи в регистр; shift – сигнал сдвига содержимого регистра в сторону младшего разряда.	Выход: bit_out – последовательный выход
Логика работы регистра	
По переднему фронту тактового импульса если reset = 1 регистр обнуляется, иначе если set = 1, пишем в регистр из входной шины, если shift = 1 пишем в старший разряд единицу и сдвигаем содержимое регистра в сторону младшего разряда.	

- 4) Создать модуль тестового окружения для верификации сразу двух регистров в соответствии с заданной структурой.

- 5) Изучить программу тестирования счетчика в ModelSim, листинг module cnt\_en\_testbench и подготовить соответствующий файл.
- 6) 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 контрольной работы.**

- **Числа в Verilog**

Язык Verilog может работать со следующими величинами:

- ✓ константами, которые хранят фиксированную информацию, и
- ✓ переменными, которые могут изменяться во время моделирования.

И константы, и переменные могут быть как целыми, так и действительными числами.

Наиболее часто применяются целые числа. Формат записи целого числа имеет следующий вид:

$\langle \text{разрядность} \rangle' \langle \text{основание} \rangle \langle \text{число} \rangle$

Разрядность указывается в десятичной системе и определяет количество бит под представление числа.

У констант и десятичных чисел разрядность и основание могут не указываться.

В этом случае разрядность рассчитывается из величины числа.

- **Основание определяет выбранную систему счисления:**

- ✓ *B* или *b* - двоичную;
- ✓ *D* или *d* - десятичную;
- ✓ *H* или *h* - шестнадцатеричную;
- ✓ или *o* - восьмеричную.

Например:

✓ 5'b10011      // 5 бит отводится под представление числа 19

✓ 4'd 12      // 4 бита отводится под представление числа 12

Приведем некоторые способы записи числа 8:

✓ 4'b1000 // запись сделана в двоичной системе

✓ 4'd8      // запись сделана в десятичной системе

- ✓ 8 // десятичная система может применяться без указания разрядности и основания.

Разработчик сам выбирает удобную для работы систему счисления.

В любом месте числа может быть записан знак подчеркивания в качестве разделителя разрядов, улучшающий читабельность.

Например:  $12'b0001\_1010\_1000$ .

Если разрядность заказана больше, то число дополняется слева либо нулями, либо знаком  $x$  (неопределенность) в зависимости от значения старшего разряда.

Например:

- ✓ запись  $a = 4'b010$ ; означает  $a = 0010$ ,
- ✓ запись  $c = 4'bx10$ ; означает  $c = xx10$ .

- Для обозначения поведения сигналов в Verilog, кроме 0 и 1, предусмотрен **учет неопределенного и высокоимпедансного состояний**. Эти состояния обозначаются соответственно символами  $x$  и  $z$ . Символ  $x$  используется также для обозначения безразличного состояния.

В начале работы устройства появление  $x$  является естественным, так как из-за наличия задержек элементов не во всех узлах сразу устанавливаются определенные значения сигналов. Но в процессе моделирования появление  $x$  может свидетельствовать о конфликтной ситуации.

Появление высокоимпедансного состояния  $z$  (так называемого третьего состояния) говорит о наличии обрыва в цепи. В зависимости от особенностей проекта состояние  $z$  может использоваться в качестве рабочего, но может быть и признаком конфликта.

Как  $x$ , так и  $z$  указываются в числе вместо цифры в том месте, которое соответствует сигналу, принимающему эти значения.

- **Условные операторы**

К условным относятся три оператора:

- ✓ оператор условного перехода *if*;
- ✓ оператор выбора *case*;
- ✓ оператор ветвления (тернарный оператор).

Первые два оператора применяются, как правило, в блоке *always*; оператор ветвления может применяться и в блоке *always*, и в конструкции *assign*.

**Оператор условного перехода *if***

В этом операторе вычисляется условие, и в зависимости от результата выполняется либо первый указанный оператор (или группа операторов), либо второй.

Конструкция оператора имеет следующий вид:

```
if (условие)
  begin [:<имя>] // выполняется, если результат равен 1 (истина)
    <группа операторов>
  end
else
  begin [:<имя>] // выполняется, если результат равен 0 (ложь)
    <группа операторов>
  end
```

В качестве условия могут использоваться простые и сложные функции, которые записываются с помощью операторов Verilog. Оператор логического равенства с единичной правой частью ( $== 1$ ) может не указываться; тогда в качестве условия фигурирует одна независимая переменная.

*Например,* записи *if (clk)* и *if (clk == 1)* тождественны. Результатом проверки условия должна быть однобитная величина (истина или ложь).

Допускается вложенность операторов *if*.

Таким образом, оператор условного перехода требует выполнения каких-либо действий при условии "истина" или других действий при условии "ложь". Если результат "ложь" маловероятен или невозможен, то Verilog допускает при очевидности ситуации не указывать *else*. Однако нарушение пары *if* - *else* может вызвать синтез нежелательной логики. Чтобы указать, что ничего не должно происходить при условии "ложь", используется пустой оператор *else* и точка с запятой:

```
else ;
```

### **Оператор выбора case**

В операторе выбора *case* вычисления производятся так же, как и в операторе *if*, в зависимости от условия. Однако условие может быть только численной величиной или, в частном случае, параметром.

Конструкция оператора имеет вид:

```
case ({переменная 1, переменная 2, ...})
  <условие 1>:<оператор 1>;
  <условие 2>:<оператор 2>;
  .....
default: <оператор>;
endcase
```

Независимые переменные в операторе *case* указываются с помощью конкатенации (склеивания):

```

case({in1, in2}) // in1, in2 – однобитные переменные, например, входы
устройства
2`b01: y=0 // y – выход устройства
default: y=1 // во всех остальных случаях
endcase

```

В примере оценивается величина оператора {in1, in2}. Если он равен 2`b01 (in1 = 0, in2 = 1), то выходу присваивается значение нуля. В случае несовпадения значения входных переменных с условием (ключевое слово *default*) выходной сигнал должен быть равен единице. Во избежание проблем при моделировании устройства оператор *case* всегда должен включать случай несовпадения, даже если он пустой:

```
default: ;
```

### Оператор ветвления

Оператор ветвления является тернарным (тройным) оператором. Конструкция оператора имеет вид:

```
<переменная> = <условное выражение>? <выражение 1> : <выражение 2>;
```

Если условное выражение истинно, то переменная принимает значение выражения 1, иначе - значение выражения 2.

Условное выражение (иначе - проверяемая величина) может быть переменной или параметром, а также может включать в себя проверку результата выполнения различных операторов.

Например:

```
flag = (in == 4'b1100)?1:0;
```

В данном случае, если векторная величина *in* принимает значение 1100, *flag* = 1, иначе *flag* = 0.

По умолчанию условное выражение сравнивается с единицей.

- **Векторы**

Как цепи, так и регистры могут иметь произвольную разрядность. В этом случае есть возможность объявить их как векторы.

Общий формат объявления векторов имеет следующий вид:

```
reg/wire[N2:N1]<имя порта1>,<имя порта2>...;
```

где

N2 - старший (или значимый) бит;

N1 - младший бит.

Например:

`reg[3:0]q;` // выход 4-битного (4-разрядного) регистра. Здесь порты `q[3]`, `q[2]`, `q[1]` и `q[0]` являются отдельными элементами вектора; причем `q[3]` - старший бит (нумерация от большего к меньшему).

С каждым отдельным элементом вектора (портом) можно работать и выполнять различные операции независимо от других элементов.

Примеры записи векторов:

`wire [31:0]data;` // 32-разрядная шина

`reg [3:0]a,b;` // здесь объявлены 4-разрядные переменные `a` и `b`.  
Последняя запись эквивалентна двум следующим:

`reg [3:0]a; reg [3:0]b;`

Если в характеристике `reg/wire` порты объявлены как векторы, то и в назначении `output / input` они должны быть объявлены как векторы (формат аналогичен).

Например: `output [3:0]q;`

Перечисляемые в выходах или во входах отдельные элементы вектора указываются в порядке от старшего бита - к младшему.

Например:

`output test_out[6], test_out[3], test_out[0];`

Лучший стиль - присваивать векторы только равной длины, поскольку разные моделирующие программы, а также разные версии программ неодинаково реагируют на разную длину векторов.

Если присваиваются векторы разной длины, то Verilog либо дополняет нулями, либо обрубает правую часть оператора присваивания (=) в зависимости от того, меньше или больше разрядов в правой части, чем в левой.

Например:

`reg [7:0]a; reg [3:0]c;`

`a = c;` // переменная `c` дополняется четырьмя нулями в старшие разряды;  
// неявное дополнение; некорректная запись

`a = {4'h0,c};` // применение конкатенации, явное дополнение нулями;  
// корректная запись.

Для сдвига можно использовать операцию шинирования:

```
reg [7:0] a;  
a <= {1'b1, a [7 : 1]} // сдвиг на 1 разряд вправо, в старший разряд  
//записывается 1.
```

**2. Изучить техническое задание на создание стандартного модуля сдвигового регистра с последовательным входом и параллельным выходом, создать листинг модуля проекта, подготовить соответствующий файл.**

Техническое задание:

parameter M - разрядность регистра;	
<p>Входы:</p> <p>clk - тактовый вход;</p> <p>reset – сигнал синхронного сброса ячеек регистра;</p> <p>bit_in – последовательный вход в старший разряд;</p> <p>shift – сигнал записи в старший разряд входного бита со сдвигом содержимого в сторону младшего разряда.</p>	<p>Выход:</p> <p>byte_out – M разрядная шина значения регистра</p>
Логика работы регистра	
По переднему фронту тактового импульса если reset = 1 регистр обнуляется, иначе если shift = 1 производим сдвиг и записываем вход в старший разряд.	

Вариант реализации комбинационной схемы подобного регистра на D-триггерах показан на рисунке 1.

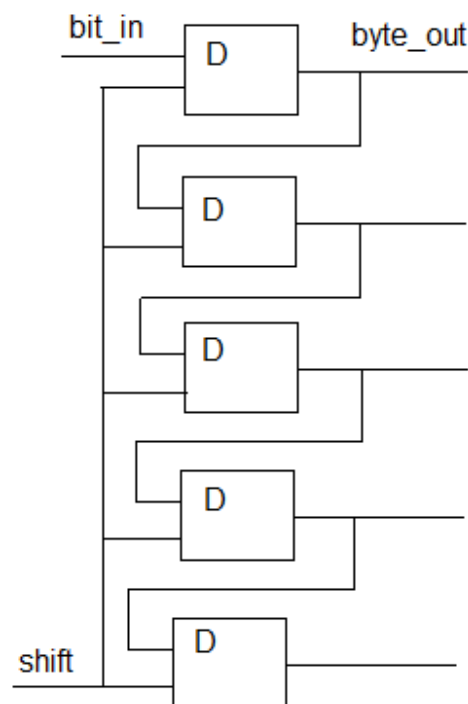


Рисунок 1- Комбинационная схема регистра на D-триггерах



**3. Изучить техническое задание на создание стандартного модуля сдвигового регистра с параллельным входом и последовательным выходом из младшего разряда, создать листинг модуля проекта, подготовить соответствующий файл.**

Техническое задание:

parameter M - разрядность регистра;	
Входы: clk - тактовый вход; reset – сигнал синхронного сброса ячеек регистра; bus_in – входная M разрядная шина; set – сигнал записи в регистр; shift – сигнал сдвига содержимого регистра в сторону младшего разряда.	Выход: bit_out – последовательный выход
Логика работы регистра	
По переднему фронту тактового импульса если reset = 1 регистр обнуляется, иначе если set = 1, пишем в регистр из входной шины, если shift = 1 пишем в старший разряд единицу и сдвигаем содержимое регистра в сторону младшего разряда.	

**4. Создать модуль тестового окружения для верификации сразу двух регистров в соответствии со структурой, приведенной на рисунке 2.**

Программа тестирования регистров на примере 8-разрядных экземпляров	
1.	Подать на вход счетчика тактовый сигнал
2.	Обнулить содержимое переменных
2.	Сброс регистров в ноль по третьему значению тактового сигнала
3.	Запись байта в регистр с параллельной загрузкой после 5 значения тактового сигнала

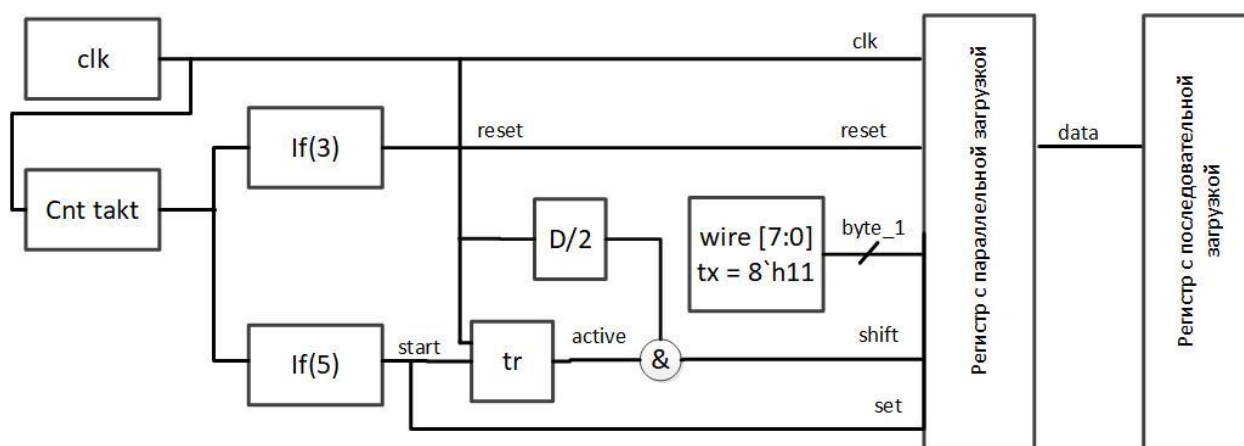


Рисунок 2 – Структурная схемы тестового модуля

Этапность создания тестового модуля:

- Создайте переменные clk и [4:0]cnt\_taktov, объявите тип переменных reg.

- b) В блоке `initial` обнулите содержимое переменных `clk` и `[4:0]cnt_taktov`.
- c) Задайте тактовый сигнал `clk` с периодом 20 единиц времени.
- d) В блоке `always` осуществите инкрементирование счетчика тактов по переднему фронту сигнала `clk`.
- e) Объявите необходимые провода и шины, в соответствии со схемой и соедините с их помощью создаваемые элементы.
- f) Подайте на провода `reset` и `start` условия равенства содержимого счетчика тактов 3 и 5, а на шину `byte_1` – значение байта.
- g) Создайте экземпляр триггера активного состояния.
- h) Создайте экземпляры тестируемых модулей и соедините их входы и выходы с объявленными проводами и шинами.

### **5. Проведите симуляцию проекта в ModelSim. Время симуляции от 500 ps.**

1. Создайте папку для проекта и поместите в нее следующие файлы: модули сдвиговых регистров (тестируемые модули), модуль тестового окружения и `trigger_x.sv` (триггер активного состояния, взять из лабораторной работы №2).
2. Откройте программу ModelSim и создайте новый проект ( задайте имя проекта, выберите свою индивидуальную папку и добавьте находящиеся в ней файлы).
3. Выполните компиляцию проекта (`Compile` → `Compile All`). В случае наличия ошибок исправьте их и выполните компиляцию заново.
4. Перейдите к процессу симуляции (`Simulate` → `Start Simulation` → `work` → `cnt_en_testbanch.sv` )
5. Добавьте сигналы, которые необходимо посмотреть в процессе симуляции ( `Add` → `to wave` → `all item in region` )
6. Откройте вкладку Wave и запустите симуляцию (`Simulate` → `Run` → `Run - All` )
7. Убедитесь в правильности работы проекта, результаты приведите в отчёте.