(Продолжение, начало №2/2001)

Verilog — инструмент разработки цифровых электронных схем

Сергей Емец

yemets@javad.ru

Операторы

Синтаксис операторов в языке Verilog подобен синтаксису языка программирования C. К сожалению, отсутствуют операторы ++, -- и все операторы модификации вида (операция)=, например, *=, ^= и т. п. Но в то же время существуют логические операторы, полезные для моделирования цифровых схем: при одинаковом синтаксисе (\, ~|, ^, ~^, & ~&) данные операторы могут быть битовыми (bitwise) и работать с двумя операндами или операторами свертки (reduction) и с одним операндом. Тип выполняемой операции определяется по положению оператора в выражении. Кроме логических операций языком поддерживаются и арифметические операции, но, чтобы получить информацию о синтезируемости выражений с арифметическими операторами, следует ознакомиться с документацией на средства синтеза. Для эффективной реализации (синтеза) арифметических выражений в «железе» возможно потребуется приобретать специальные средства или библиотеки для datapath элементов.

Рассмотрим применение операторов на примере.

```
module op_test;
reg [3:0] D0, D1, D2, D3;
reg [3:0] A, B, C, D;          //Verilog case-sensitive
reg a, b, c, d;                // A и a — различные переменные
initial
begin
 D0=4'b0;
 D1=4'b1111;
 D2=4'b1010;
 D3=4'b01xz;
 A=D1~^D2;                     // bitwise операция — два операнда
 a=~^D2;   // reduction операция — один операнд
 B=D0^D3;
 b=^D3;
 C=D2&D3;
 c=&D3;
 D=D2|D3;
 d=|D3;
 $display(«A=%b a=%b B=%b b=%b C=%b c=%b D=%b d=%d», A, a, B, b, C,
c, D, d);
end
endmodule
```

Полученный результат будет выглядеть так:

```
Highest level modules:
op_test
A=1010 a=1 B=01xx b=x C=00x0 c=0 D=111x d=1
```

При выполнении логических и битовых операций состояние с высоким импедансом (z) воспринимается как неопределенное (x). При моделировании комбинаторная логика обычно способствует распространению x, но если, например, на один из входов элемента И (And) подан 0, то, независимо от значения на других входах, на выходе будет 0. Это иллюстрируют полученные в примере значения C и c.

В языке присутствует условный оператор ?:, который работает так же, как и в языке C. Таким образом, простейшей записью мультиплексора из 2 в 1 является:

```
assign Y=(SEL)?A:B;
```

Список операторов языка Verilog:

{}	concatenation
+ — * /	arithmetic
%	modulus
> >= < <=	relational
!	logical negation
&&	logical and
||	logical or
==	logical equality
!=	logical inequality
===	case equality *
!==	case inequality *
~	bit-wise negation
&	bit-wise and
|	bit-wise inclusive or
^	bit-wise exclusive or
^~ or ~^	bit-wise equivalence
&	reduction and
~&	reduction nand
|	reduction or
~|	reduction nor
^	reduction xor
~^ or ^~	reduction xnor
<<	shift left
>>	shift right
<<<	arithmetic shift left
>>>	arithmetic shift right
?:	conditional

Операторы, помеченные *, будут рассмотрены при описании поведенческих конструкций.

Часть операторов повторяется. Например, << и <<< выполняют одинаковое действие, а последовательное выполнение ~ и ^ имеет такое же значение, как и ^~ или ~^.

Применяться операторы могут как к цепям (wire), так и к сигналам (reg) и переменным. При этом используются различные типы присвоения. Для цепей, которые являются моделью физического соединения (провода), требуется подключение непрерывного воздействия, которое моделируется непрерывным (continuous) присвоением. Значения же регистров и

переменных могут изменяться в результате процедурных действий и сохраняться между воздействиями (так же, как и переменные процедурного языка программирования). Для моделирования этого используется процедурное (procedural) присвоение. Непрерывное присвоение употребляется вне процедурных блоков initial или begin и используется либо в описании цепи, либо с ключевым словом assign (существуют также процедурные непрерывные присвоения в блоках initial или always с ключевыми словами assign и deassign). Слева от оператора непрерывного присвоения (=) должен находиться объект типа цепь. При изменении значения какого-либо из объектов, входящих в выражение справа от =, данное выражение будет вычислено, и новое значение будет присвоено.

Ниже приведен пример, иллюстрирующий работу непрерывного присвоения.

```
module assign_test;
reg [3:0] data;
wire parity, forth;
wire gnd=1'b0;              //обьявление присвоение
wire y=(data[0])?data[1]:gnd;          //мультиплексор
assign parity=^data;        //непрерывное присвоение
assign forth=~|data[1:0];   //еще одно присвоение
initial
$monitor(data,, parity,, forth,, y);          //запускаем системную
функцию для индикации изменений
initial
for (data=0; data!=15; data=data+1'd1)    //переберем варианты
#1;          //задержка нужна, чтобы $monitor работал правильно
endmodule
```

Результат работы этого примера:

```
Highest level modules:
assign_test
0 0 1 0
1 1 0 0
2 1 0 0
3 0 0 1
4 1 1 0
5 0 0 0
6 0 0 0
7 1 0 1
8 1 1 0
9 0 0 0
10 0 0 0
11 1 0 1
12 0 1 0
13 1 0 0
14 1 0 0
15 0 0 1
```

Процедурные присвоения бывают двух типов: blocking (=) и nonblocking (<=). В большинстве случаев разработчики переводят эти названия на русский язык как «блочные» и «не блочные» присвоения, но по смыслу более подходящим кажутся термины «блокирующие» и «не блокирующие». Для того чтобы понять разницу между данными типами присвоения, нужно рассмотреть работу Verilog симулятора. В реальном объекте (цифровой схеме), который моделируется с помощью языка Verilog, события могут происходить одновременно: при изменении входного сигнала во всех элементах начинаются процессы, которые протекают одновременно и приводят к каким-либо изменениям выходных сигналов. Программа симулятора не может обрабатывать события одновременно и создает списки событий, которые должны выполняться последовательно. Когда все события из списка выполнены, симулятор переходит к следующему временному шагу: увеличивает значение текущего момента времени на некоторый временной интервал (второй параметр директивы `timescale) и выполняет список событий, которые должны произойти на этом шаге. Об исполнении событий в других временных шагах (механизме задержек) будет рассказано далее, а сейчас рассмотрим события, происходящие «одновременно» — в одном временном шаге.

Допустим, имеется следующее описание:

```
always @ (posedge CLK) a=b;
always @ (posedge CLK) b=a;
```

Пусть a и b — регистры единичной длины, и к моменту положительного фронта тактового сигнала CLK a = 0, b = 1. Какое значение будут иметь эти переменные после прохождения фронта? Это не определено в языке и зависит от того, в какой последовательности операции присваивания попадут в список. То есть получается конструкция, поведение которой непредсказуемо. Это значит, что либо обе эти переменные будут равны 0, либо 1. Конкретный результат зависит от симулятора и от последовательности, в которой будут обработаны строки исходного файла. «Блокирующее», или «блочное», присвоение (=) блокирует исполнение других операций в блоке последовательных операций (до тех пор, пока не будет выполнена данная операция). Использование «блокирующего» присвоения в подобной конструкции в конкурентно исполняемых блоках нежелательно, и его следует избегать. Но в то же время, если в блоке требуется провести последовательное исполнение операторов, следует применять данный тип присвоения.

Конструкция

```
always @ (posedge CLK)
begin
 a=0;
 b=a;
end
```

гарантирует, что после фронта CLK обе переменные — a и b — будут обнулены.

Если же предыдущие примеры переписать с использованием «неблокирующего», или «неблочного», присвоения (<=), то поведение модели изменится:

```
always @ (posedge CLK) a<=b;
always @ (posedge CLK) b<=a;
```

В этом случае в список событий, исполняемых во временном шаге после изменения CLK, эти операции будут помещены параллельно, то есть переменные обменяются своими значениями. После прохождения фронта CLK переменные изменят свои значения: a = 1, b = 0. Данная конструкция будет эквивалентна следующей:

```
always @ (posedge CLK)
begin
 a<=b;
 b<=a;
end
```

и описывает два D-триггера, выход каждого подан на вход другого, а на тактовые входы подан сигнал CLK. При этом последовательность записи: a<=b; b<=a или b<=a; a<=b — не играет роли, так как моделируются одновременно происходящие события.

Вторая конструкция в этом случае:

```
always @ (posedge CLK)
begin
 a<=0;
 b<=a;
end
```

описывает два D-триггера. На вход одного (a) подан 0, а выход подключен к входу другого (b). На тактовые входы подан сигнал CLK.

Если дополнить первый триггер сигналом асинхронной установки, получится схема, которая реально может быть использована для синхронизации и нормирования коротких импульсов.

```
always @ (posedge CLK or posedge SET)
begin
 if (SET) a<=1'b1;
 else
 begin
 a<=0;
 b<=a;
 end
end
```

Проверим работоспособность данной схемы следующим испытательным стендом (это упрощенный testbench — правильный пример создания испытательных стендов приведен в первой части статьи):

```
`timescale 1ns/10ps
module test;
reg CLK, SET, a, b;
always
#5 CLK=~CLK;              //100 МГц тактовый сигнал
always @ (posedge CLK or posedge SET)   //испытываемый    блок
begin
if (SET) a<=1'b1;
 else
 begin
 a<=0;
 b<=a;
 end
end
initial        //инициализируем переменные
begin
 CLK=0;
 SET=0;
 a=0; b=0;
 $monitor($time,,SET,,a,,b);
end
initial #100 $finish;   // через 100 нс завершить моделирование
initial        //подача «асинхронного» сигнала SET
begin
 #57 SET=1'b1;
 #1 SET=1'b0;
end
endmodule
```

Результат работы этого теста:

```
Highest level modules:
test
0 0 0 0
57 1 1 0
58 0 1 0
65 0 0 1
75 0 0 0
```

показывает, что асинхронный сигнал SET, длительностью 1 нс (фронт на 57 нс после пуска), был синхронизирован по фронту CLK в регистре b — длительность 10 нс (фронт 65 нс, срез 75 нс).

Для пользователей VHDL можно провести параллель между variable assignment (:= VHDL) и blocking assignment (= Verilog), signal assignment (<= VHDL) и nonblocking assignment (<= Verilog) соответственно. Но следует учесть, что в процедурных конструкциях Verilog различий между регистром и переменной не делается.

Временной и событийный контроль

Так как Verilog используется для моделирования физических систем, то большое внимание уделено привязке события к определенному моменту времени. Для этого используется механизм задержек.

Следует заметить, что средства синтеза (как для Verilog, так и для VHDL) игнорируют вре-

менные конструкции в исходном коде. Для правильной генерации цифровой схемы (нет листа) из библиотечных элементов в средства синтеза, наряду с описанием на языке высокого уровня вводятся файлы, содержащие описания «constrain». В этих файлах описываются временные ограничения распространения сигналов. Применяемый для этого синтаксис не стандартизован и определяется используемым средством синтеза. Профессиональная работа с HDL-языками высокого уровня подразумевает не только умение создавать грамотные поведенческие описания, но и умение правильно управлять средством синтеза с помощью «constrain».

Однако для моделирования временной контроль необходим, и в примерах этой статьи неоднократно использовались выражения вида #<число>. Вместо числа может использоваться выражение, содержащее целые и/или вещественные параметры. В синтаксисе языка определено несколько форм временных задержек для описания различных технологических условий, наиболее полной из которых является так называемая mintypmax форма — #(min, typ, max), например #(2, 3, 4). Но данные конструкции используются, как правило, для моделирования на уровне вентилей (нет листа), а более эффективным методом таких описаний является SDF-аннотация с применением специальных SDF-файлов. Поэтому в статье будет использоваться упрощенная форма задания задержки с одним параметром #typ. Для анализа исходных файлов «третьей стороны» можно считать, что всегда используется типовое (среднее) значение.

Рассмотрим применение задержки в непрерывном присвоении.

```
assign #10 c=a^b;
```

Данная конструкция описывает элемент «исключающее или» с задержкой распространения 10 нс (вернее, 10 единиц первого параметра директивы `timescale, который по умолчанию равен 1 нс). Все задержки в непрерывных присвоениях являются инерциальными. Это означает, что если сигнал A изменит свое состояние на время, меньшее 10 нс, а затем изменит еще раз, то изменения сигнала C не произойдет. Для того чтобы произошло изменение сигнала C требуется, чтобы сигнал A был зафиксирован в новом состоянии более 10 нс. Такая модель поведения соответствует распространению задержки при прохождении через элементы электронной схемы. Другой тип задержки — транспортная задержка, которая обеспечивает точное соответствие формы задержанного сигнала и сдвиг его по шкале времени. В отличие от VHDL (в котором существуют спецификаторы типа задержки inertial и transport) Verilog не позволяет использовать оба типа задержек в одном моделировании. Для переключения типа задержки служит ключ командной строки запуска Verilog-симулятора.

Рассмотрим задержки в процедурных блоках. Первая форма — простая задержка (она использовалась в примерах статьи). Она имеет вид:

```
#1 x=y;
```

Такая задержка вызывает остановку последовательного блока на 1 нс (не влияет на обработку конкурентно исполняемых блоков). Она не обязательно используется с оператором присваивания, может использоваться с пустым оператором #1. То есть #1 x=y может быть записано в такой форме #1; x=y или #0.5; #0.5 x=y. Поведение этих конструкций одинаково.

Также в процедурном блоке задержка может встречаться с другой стороны знака =. Это так называемая intra-assignment delay.

```
x=#1 y;
```

В этом случае вначале происходит вычисление выражения, затем задержка, а затем присвоение. То есть рассматриваемый пример эквивалентен

```
tmp=y;
#1;
x=tmp;
```

Кроме временного контроля существует событийный контроль (который можно считать другой формой временного контроля). Признаком событийного контроля является знак @. В рассмотренных ранее примерах событийный контроль использовался в блоках always. Отличие Verilog от VHDL в данном случае состоит в том, что для описания фронтов и срезов сигналов используются не специализированные атрибуты сигнала, а специальная конструкция языка. Это создает впечатление, что разработчики языка Verilog несколько лучше представляли себе процесс разработки цифровых схем. Событийный контроль используется в процедурных блоках так же, как и временной контроль. При этом задержка исполнения происходит не на фиксированный временной интервал, а до тех пор, пока не произойдет нужное событие. События бывают следующих типов:

- @(name) — изменение name, при этом name может являться цепью, регистром, переменной или переменной типа event;
- @(posedge A) или @(negedge A) — фронт или срез сигнала A, при этом A — однобитовый регистр или цепь;
- комбинация перечисленных событий с ключевым словом «or», например @(posedge CLK or posedge SET); в этом случае следует различать «or» с одноименной логической операцией; в событийном контроле «or» означает, что ожидается любое из перечисленных событий, а не определенный результат логической операции.

Для генерации синхронного сброса может использоваться такая конструкция:

```
nReset=0;
repeat (16) @(posedge CLK);
nReset=1'b1;
```

Конструкция во второй строке примера обеспечивает задержку на 16 тактов, вернее, задержку на прохождение 16 фронтов сигнала CLK.

(Продолжение следует)