Операторы

Набор операторов (операций) в VHDL обеспечивает возможность работы с предопределнными типами данных.

Список операций приведен в табл. 1.

Строки табл. 2 располагаются в порядке старшинства (от низшего к высшему) операторов. Операторы, находящиеся в одной строке, обладают одинаковым старшинством (приоритетом). Таким образом, операции нижней строки табл. 2 обладают наибольшим приоритетом и выполняются первыми, в частности, логический оператор not выполняется прежде других логических операторов.

Таблица 1

Операции языка VHDL

Обозначение	Название
not	логическое НЕ
and	логическое И
or	логическое ИЛИ
nand	логическое И-НЕ
nor	логическое ИЛИ-НЕ
xor	исключающее ИЛИ
xnor	эквивалентность
=	равно
/=	не равно
<	меньше
<=	меньше либо равно
>	больше
>=	больше либо равно
+	сложение, присвоение знака +
−	вычитание, присвоение знака -
&	конкатенация
*	умножение
/	деление
mod	модуль
rem	остаток
**	возведение в степень
abs	абсолютное значение

Исходя из контекста VHDL-кода следует отличать оператор <= (назначение сигнала) и оператор <= (меньше либо равно). Следует

также отличать унарные операции присвоения знака +,− от соответствующих бинарных операций сложения и вычитания.

Замечание. Для устранения неоднозначностей трактовки старшинства операций используются скобки.

Например, выражение «A nand B nand C» неверно (синтаксическая ошибка). Данное выражение не представляет трехвходовый элемент И-НЕ (трехвходовую NAND-ячейку). Правильная запись «not (A and B and C)». Запись «A nand (B nand C)» не есть то же самое, что «(A nand B) nand C».

Таблица 2

Классификация операций

Класс операций	Операции					
Логические	and	or	nand	nor	xor	xnor
Сравнения	=	/=	<	<=	>	>=
Сложения и конкатенации	+	-	&			
Присвоение знака	+	—				
Умножения	*	/	mod	rem		
Смешанные	**	abs	not			

В табл. 3 сведены логические и арифметические операторы, а также указаны типы операндов и результатов выполнения операций.

Не все эти операторы могут быть автоматически синтезированы (т.е. превращены в реальную схему). Например, оператор деления \ не может быть синтезирован автоматически. При этом все операторы могут использоваться при моделировании.

За каждым оператором, который может быть синтезирован, стоят реальные аппаратные ресурсы, т.е. если вы ставите знак +(*), то в железе реализуется блок сумматора (умножителя).

По-умолчанию, большинство арифметических операторов (см табл. 3) применимы к сигналам типа integer. Это не всегда удобно, потому что в реальных схемах используются только два типа данных:

std_logic и std_logic_vector. При этом над ними также необходимо совершать арифметические операции. Для этого необходимо: перегрузить арифметические операторы, чтобы они применялись и к типам данных std_logic и std_logic_vector и/или ввести функции перевода сигналов из типа integer в std_logic_vector и наоборот. Для этого есть следующие варианты.

Использовать следующие пакеты из библиотеки ieee.

```
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

В этих библиотеках существуют переводные функции из integer в std_logic_vector и наоборот. Пусть:

signal a_int : integer range 0 to 255;

signal a_vec : std_logic_vector (7 downto 0);

Тогда можно писать:

a_int <= `conv_integer`(a_vec);

Или в обратную сторону:

a_vec <= `conv_std_logic_vector`(a_int,8);

Второй аргумент функции conv_std_logic_vector указывает, какой разрядности должен быть результат преобразования.

Пакет `ieee.std_logic_arith` содержит перегруженные определения арифметических операторов (+,-,*), а также функции перевода `conv_integer` и `conv_std_logic_vector`.

Пакет `ieee.std_logic_unsigned` позволяет применять эти операции к сигналам типа std_logic_vector, считая, они представляют беззнаковые целые числа. Это означает, что при использовании этих библиотек возможна следующая запись:

signal a : std_logic_vector(7 downto 0);

signal b : std_logic_vector(7 downto 0);

b <= a + 1;

Вместо пакета `ieee.std_logic_unsigned` можно использовать пакет `ieee.std_logic_signed`. Тогда сигналы типа std_logic_vector будут представлять числа со знаком. Одновременное использование обоих пакетов не допускается.

Примеры арифметических операторов.

Сложение (+)

```
RealX2 + 2.0 -- если RealX2 есть сигнал типа
             -- вещественный, то число 2 должно
```

-- быть записано как вещественное

1us + 3ns -- 1003ns

Таблица 3

operator	description	data type of operand a	data type of operand b	data type of result
a ** b	exponentiation	integer	integer	integer
abs a	absolute value	integer		integer
not a	negation	boolean, bit, bit_vector		boolean, bit, bit_vector
a * b	multiplication	integer	integer	integer
a / b	division			
a **mod** b	modulo			
a **rem** b	remainder			
+ a	identity	integer		integer
- a	negation			
a + b	addition	integer	integer	integer
a - b	subtraction			
a & b	concatenation	1-D array, element	1-D array, element	1-D array
a **sll** b	shift left logical	bit_vector	integer	bit_vector
a **srl** b	shift right logical			
a **sla** b	shift left arithmetic			
a **srl** b	shift right arithmetic			
a **rol** b	rotate left			
a **ror** b	rotate right			
a = b	equal to	any	same as a	boolean
a /= b	not equal to			
a < b	less than	scalar or 1-D array	same as a	boolean
a <= b	less than or equal to			
a > b	greater than			
a >= b	greater than or equal to			
a **and** b	and	boolean, bit, bit_vector	same as a	boolean, bit, bit_vector
a **or** b	or			
a **xor** b	xor			
a **nand** b	nand			
a **nor** b	nor			
a **xnor** b	xnor			

Вычитание (-)

8.33 - 5 -- неправильно, оба числа должны

-- быть одного типа

BusWidth - 1 -- допустимо, если BusWidth типа integer

Умножение (*)

4*SomeVal -- допустимо, если SomeVal
-- типа integer или time

Деление (/)

CLK/2 -- тип CLK - integer

5.0/2.0 -- результат - вещественное число 2.5

10ns/2ns — результат 5 типа integer, но не time

Модуль (mod)

6 **mod** 4 -- результат 2

6 **mod** (-4) -- результат -2

(-6) **mod** 4 -- результат 2

Остаток деления (rem)

6 **rem** 4 -- результат 2

6 **rem** (-4) -- результат 2

(-6) **rem** 4 -- результат -2

Экспонента (**)

c ** 0.5 -- неправильно в VHDL

A ** 2 -- эквивалентно A * A

B ** 3 -- эквивалентно B * B * B

Абсолютное значение (abs)

abs 1 -- результат 1

abs(-l) -- результат 1

abs(5 * (-2)) -- результат 10

В стандарте языка VHDL операции **rem** и **mod** определяются следующим образом.

Для операции L **rem** R должно выполняться соотношение

L = (L / R) * R + (L **rem** R),

где L/R - целая часть частного, (L **rem** R) - результат выполнения операции **rem** (остаток). Остаток имеет *знак операнда L.*

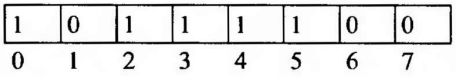
Для операции L **mod** R должно выполняться соотношение

L = N * R + (L **mod** R),

где N — некоторое целое число, (L **mod** R) - результат выполнения операции **mod.** Результат имеет *знак операнда R.*

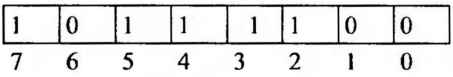
Следует быть внимательным при переходе от битовых векторов к числам:

Bit vector (0 **to** 7); -- возрастающий диапазон



Число 61 (десятичное), старший разряд справа

Bit vector (7 **downto** 0); -- убывающий диапазон



Число 188 (десятичное), старший разряд слева

Логические операторы выполняются для следующих типов данных:

- boolean;
- bit, bit_vector;
- std_logic, std_logic_vector;
- std_ulogic, std_ulogic_vector.

Логические операторы **and, or, xor** имеют одинаковое старшинство и выполняются слева направо в выражениях. Оператор **not** имеет более высокое старшинство и выполняется прежде других операторов. В сложных логических выражениях порядок выполнения операторов регулируется скобками. Рекомендуем читателю применять скобки в затруднительных случаях. Например, для выражения

Z <= A **and not** B **or** C;

будет только отрицание B, в выражении

Z <= A **and not** (B **or** C); будет отрицание подвыражения B **or** C, находящегося в скобках.

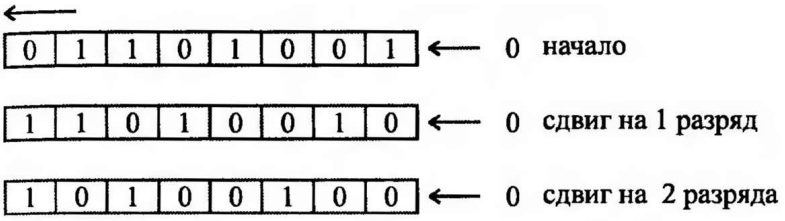
Операторы сдвига

Данные операторы поясним на примере семиразрядного битового вектора My Bus

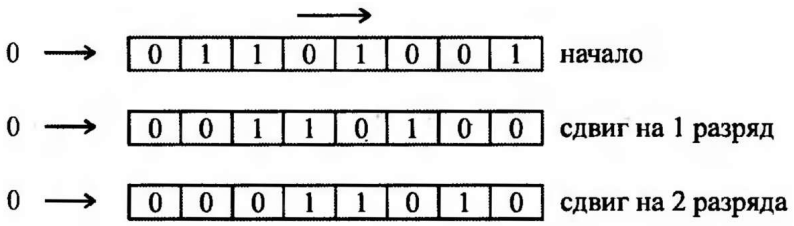
signal MyBus: bit_vector(7 **downto** 0) := "01101001";

Логический сдвиг влево (Shift Left Logical)

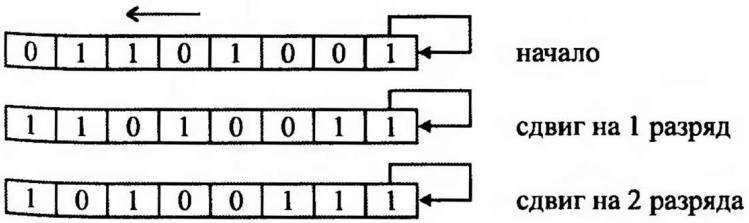
оператор **sll**



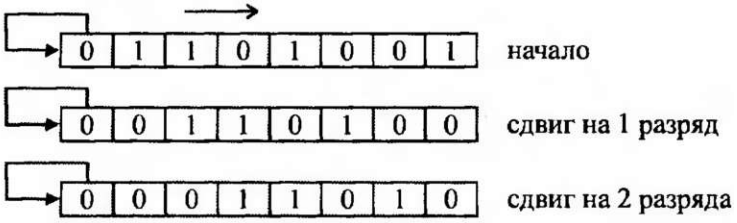
Логический сдвиг вправо (Shift Right Logical)
оператор **srl**



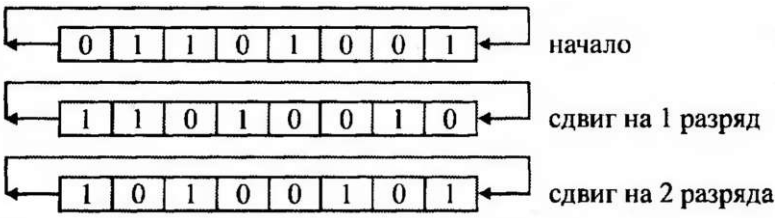
Арифметический сдвиг влево (Shift Left Arithmetic)
оператор **sla**



Арифметический сдвиг вправо (Shift Right Arithmetic)
оператор **sra**



Вращение логическое влево (Rotate Left Logical)
оператор **rol**



Вращение логическое вправо (Rotate Right Logical)
оператор **ror**