Продолжение. Начало в № 3 `2008

Иосиф КАРШЕНБОЙМ
iosifk@narod.ru

Краткий курс HDL.
Часть 2. Описание языка

Начало описания языка Verilog и немного о VHDL для сравнения

В этом разделе будет дано вступление к описанию языка Verilog. Для сравнения также будет приведена небольшая часть описания языка VHDL. Это сделано для того, чтобы читатель смог оперативно сравнивать конструкции этих двух языков. Для дополнительного изучения можно порекомендовать список литературы.

VHDL

Для VHDL приняты следующие соглашения об именах, применяемых пользователями в своих проектах:

- VHDL не отличает строчные и прописные буквы.
- Два знака «--» используются для строчного комментария.
- В именах могут использоваться цифры и символ подчеркивания — «_».
- Имена могут начинаться с цифры или буквы.
- Нельзя использовать два символа подчеркивания «__» или использовать символ подчеркивания в конце имени.
- Имена компонентов схемы должны быть уникальными. Например, нельзя назвать сигнал именем «A» и дать шине имя «A(7 downto 0)».

Зарезервированные слова VHDL приведены в таблице 1.

Таблица 1. Зарезервированные слова VHDL

abs	downto	library	postponed	Subtype
access	else	linkage	procedure	then
after	elsif	literal	process	to
alias	end	loop	pure	transport
all	entity	map	range	type
and	exit	mod	record	unaffected
architecture	file	nand	register	units
array	for	new	reject	until
assert	function	next	rem	use
attribute	generate	nor	report	variable
begin	generic	not	return	wait
block	group	null	rol	when
body	guarded	of	ror	while
buffer	if	on	select	with
bus	impure	open	severity	xnor
case	in	or	shared	xor
component	inertial	others	signal	
configuration	inout	out	sla	
constant	is	package	sra	
disconnect	label	port	srl	

Verilog

Для Verilog приняты следующие соглашения об именах, применяемых пользователями в своих проектах:

- Verilog отличает строчные и прописные буквы.
- Два знака «//» используются для строчного комментария. Набор знаков «/*» используется для начала блочного комментария, а «*/» — для конца комментария.
- В именах можно использовать цифры, буквы, символ подчеркивания «_» и символ «$».
- Имена могут начинаться с цифры, буквы или символа подчеркивания «_».
- В именах нельзя использовать символ пробела.

Зарезервированные слова Verilog приведены в таблице 2.

Таблица 2. Зарезервированные слова Verilog

always	endmodule	medium	reg	tranif0
and	endprimitive	module	release	tranif1
assign	endspecify	nand	repeat	tri
attribute	endtable	negedge	rnmos	tri0
begin	endtask	nmos	rpmos	tri1
buf	event	nor	rtran	triand
bufif0	for	not	rtranif0	trior
bufif1	force	notif0	rtranif1	trireg
case	forever	notif1	scalared	unsigned
casex	fork	or	signed	vectored
casez	function	output	small	wait
cmos	highz0	parameter	specify	wand
deassign	highz1	pmos	specparam	weak0
default	if	posedge	strength	weak1
defparam	ifnone	primitive	strong0	while
disable	initial	pull0	strong1	wire
edge	inout	pull1	supply0	wor
else	input	pulldown	supply1	xnor
end	integer	pullup	table	xor
endattribute	join	remos	task	
endcase	large	real	time	
endfunction	realtime	macromodule	tran	

Сигналы или переменные могут быть представлены следующими логическими уровнями:

- 0: ноль, логический низкий уровень, «ложно», «земля» (zero, logic low, false, ground);
- 1: единица, логический высокий уровень, «истина», «питание» (one, logic high, power);
- X: неизвестное значение (unknown);
- Z: высокий импеданс, неподключенный сигнал, третье состояние (high impedance, unconnected, tri-state).

Операторы

Операторы могут быть трех типов: унитарные, бинарные и тернарные. Унитарные операторы производят действия над единственным операндом. Бинарные операнды производят действия над двумя операндами. Тернарные операторы имеют два оператора, которые делают выбор для трех операндов. Примеры этих операндов:

```
a = ~ b;        // ~ унитарный оператор. b — операнд
a = b && c;     // && бинарный оператор. b и c — операнды
a = b ? c : d;  // ?: тернарный оператор. b, c и d — операнды
```

В языках VHDL и Verilog используются следующие арифметические и битовые операторы:

- арифметические(Arithmetic);
- конкатенации (объединения) (Concentration);
- повторения (Replication);
- условные (Conditional);
- равенства (Equality);
- логические побитовые операторы (Logical Bit-wise);
- логического сравнения (Logical Comparison);
- свертки (Reduction);
- отношения (Relational);
- сдвига (Shift);
- унитарные арифметические операторы (Unary Arithmetic (Sign)).

Операнды (Operands)

Есть несколько типов операндов, которые могут быть определены в выражениях. Самый простой тип — это цепь или регистр, и, если при этом указывается на полную разрядность цепи или регистра, то можно приводить только название цепи или регистра. В этом случае все биты, составляющие цепь или выходы регистра, используются как операнд.

Как цепи, так и регистры могут иметь несколько разрядов. При этом, если используется только единственный бит из вектора цепи или регистра, то необходимо использовать операнд выбора бита. Если при обращении к группе смежных битов в векторной цепи или в регистре используется только часть битов, то необходимо применить операнд выбора части битов из вектора.

На элемент памяти можно сослаться как на операнд. Конкатенация других операндов (включая вложенные конкатенации) может быть определена как операнд. Запрос функции — это тоже операнд.

Операторы и возможность их применения в VHDL или в Verilog приведены в таблице 3. Приоритеты выполнения операторов в Verilog приведены в таблице 4.

Битовые операторы (Bit-Wise Operators)

Битовые операторы позволяют производить побитовые операции над двумя или более операндами (табл. 5–9).

Обратите внимание на то, что необходимо отличать битовые операторы & и | от логических операторов && и ||. Например, если x = 1, и y = 2, то битовая операция x & y в результате даст 0, в то время как x && y — в результате даст 1. Когда операнды имеют неравное число битов, то операнд, имеющий меньшее число битов, будет заполнен нулями в старших разрядах.

Таблица 3. Операторы и возможность их применения в VHDL или в Verilog

Операция	Оператор		Описание
	VHDL	Verilog	
Арифметические операторы (Arithmetic Operators): exponential multiplication division addition subtraction modulus remainder absolute value	 ** * / + – mod rem abs	 * / + – % 	 Возведение в степень Умножение Деление Сложение Вычитание Модуль (остаток от деления) Остаток от деления Абсолютная величина
Concentration and Replication Operators: concentration replication	 &	 { } {{ }}	 Объединение Повторение
Conditional Operator: conditional		 ?:	 Условный оператор
Операторы равенства (Equality Operators): equality inequality	 = /=	 == !=	 Равно Не равно
Битовые логические операторы(Logical Bit-wise Operators): unary negation NOT binary AND binary OR binary NAND binary NOR binary XOR binary XNOR	 not and or nand nor xor xnor	 ~ & | ^ ^~ or ~^	 Побитовая инверсия Поразрядное двоичное «И» Поразрядное двоичное «ИЛИ» Поразрядное двоичное «И-НЕ» Поразрядное двоичное «ИЛИ-НЕ» Поразрядное двоичное «Исключающее ИЛИ» Поразрядное двоичное «Исключающее ИЛИ-НЕ»
Операторы логического сравнения (Logial Comparison Operators): NOT AND OR	 not and or	 ! && ||	 Поразрядное логическое «НЕ» Поразрядное логическое «И» Поразрядное логическое «ИЛИ»
Операторы свертки (Reduction Operators): AND OR NAND NOR XOR XNOR		 & | ~& ~| ^ ^~ or ~^	 Побитовая свертка
Операторы отношения (Relational Operators): less than less than or equal to greater than greater than or equal to	 < <= > >=	 < <= > >=	 Меньше Меньше или равно Больше Больше или равно
Операторы сдвига (Shift Operators): logical shift left logical shift right arithmetic shift left arithmetic shift right logical rotate left logical rotate right	 sll srl sla sra rol ror	 << >> 	 Логический сдвиг влево Логический сдвиг вправо Арифметический сдвиг влево Арифметический сдвиг вправо Циклический сдвиг влево Циклический сдвиг вправо
Идентичность		=== !==	Идентично Не идентично

Таблица 4. Приоритеты выполнения операторов

Приоритет выполнения операторов	
! ~ * / % + – << >> <<= >>= == != === !== & ^ ^~ | && || ?: (ternary operator)	Высший приоритет исполнения Низший приоритет исполнения

Таблица 5. Поразрядная инверсия

~	
0	1
1	0
x	x

Таблица 6. Поразрядная операция «И» (AND)

&	0	1	x
0	0	0	0
1	0	1	x
x	0	x	x

Таблица 7. Поразрядная операция «ИЛИ» (Or)

|	0	1	x
0	0	1	x
1	1	1	1
x	x	1	x

Таблица 8. Поразрядная операция «Исключающее ИЛИ» (exclusive Or)

^	0	1	x
0	0	1	x
1	1	0	x
x	x	x	x

Таблица 9. Поразрядная операция «Исключающее ИЛИ-НЕ» (exclusive NOR)

^~	0	1	x
0	1	0	x
1	0	1	x
x	x	x	x

Операторы приведения

Операторы приведения выполняют операции по-битно над операндом, чтобы провести результат, имеющий разрядность 1 бит. Первый шаг операции выполняется над первым и вторым битами операнда, в соответствии с таблицами 10–12. Вторые и последующие шаги выполняются аналогично с однобитовым результатом предшествующего шага и следующими битами операнда.

Таблица 10. Оператор приведения по «И» (AND)

&	0	1	x
0	0	0	0
1	0	1	x
x	0	x	x

Таблица 11. Оператор приведения по «ИЛИ» (Or)

|	0	1	x
0	0	1	x
1	1	1	1
x	x	1	x

Таблица 12. Оператор приведения по «Исключающему ИЛИ» (exclusive Or)

^	0	1	x
0	0	1	x
1	1	0	x
x	x	x	x

Необходимо отметить, что оператор приведения по «И-НЕ» и оператор приведения по «ИЛИ-НЕ» выполняет те же операции, что и операторы приведения по «И» и «ИЛИ» соответственно, но результаты их работы при этом будут инверсными. Результаты работы операторов приведения даны в таблицах 13, 14.

Таблица 13. Результаты работы оператора приведения для «И», «ИЛИ», «И-НЕ» и «ИЛИ-НЕ»

Результат работы оператора приведения для &, |, ~&, и ~|		
Операнд	& |	~& ~|
биты не установлены в все биты установлены в часть битов, но не все, установлены в	0 0 1 1 0 1	1 1 0 0 1 0

Таблица 14. Результаты работы оператора приведения для «Исключающего ИЛИ» и «Исключающего ИЛИ-НЕ»

Результат работы оператора приведения для ^ и ~^		
Операнд	^	~^
Значения нечетных битов Значения четных битов, если они есть	1 0	0 1

Ограничения синтаксиса

В языке Verilog есть два ограничения синтаксиса, предназначенные для того, чтобы защитить текст описания от ошибки, которая возникает при неправильной записи выражения. Эти ошибки бывает достаточно трудно найти, потому что они возникают из-за неправильного положения пробела и символа в выражении. Для примера можно рассмотреть два выражения. Выражения строк 1 и 2 внешне почти похожи, но дают совершенно разный результат (пример 1).

```
1. a & &b a ! !b
2. a && b a !! b
```

Пример 1. Тексты описаний, которые почти похожи, но дают совершенно разный результат

Как защитить пользователей от этого типа ошибок? Необходимо использовать круглые скобки для того, чтобы отделить сокращение ИЛИ или И (reduction or or and) оператор от поразрядного ИЛИ или И (bit-wise or or and) оператора. В примере 2 показан синтаксис, который требует круглых скобок.

Неправильный синтаксис	Правильный синтаксис
a & &b	a & (&b)
a | |b	a | (|b)

Пример 2. Примеры неправильного и правильного синтаксиса

Операторы сдвига

Операторы сдвига, << и >> выполняют сдвиг операнда, находящегося слева от оператора, влево или вправо на то число позиций двоичного разряда, которое указано справа от оператора сдвига. При сдвиге данных оба оператора производят заполнение освободившихся позиций двоичного числа нулями. Пример 3 иллюстрирует действия, выполняемые этими операциями.

```
module shift;
reg [3:0] start, result;
initial
        begin
        start = 1; // Start is set to 0001
        result = (start << 2); // Result is set to 0100
        end
endmodule
```

Пример 3. Пример работы операторов сдвига

В 3-м примере 4-битовому регистру start присвоено значение 0001. После сдвига влево на два разряда регистр будет содержать число 0100.

Условный оператор

Условный оператор имеет три операнда, отделенные двумя операторами, записанными в следующем порядке (пример 4).

```
cond_expr ? true_expr : false_expr
```

Пример 4. Условный оператор, который имеет три операнда

Если **cond_expr** оценивается как ложное — **false**, то для результата будет использовано выражение **false_expr**. Если условное выражение истинно — **true**, то для результата будет использовано выражение **true_expr**.

Если **cond_expr** или **true_expr** и **false_expr** содержат символы неоднозначности — X, то для того чтобы вычислить конечный результат, используйте таблицу 15.

Таблица 15. Работа условного оператора

?:	0	1	x	z
0	0	x	x	x
1	x	1	x	x
x	x	x	x	x
z	x	x	x	x

Если разрядность операндов различна, то операнд с меньшей разрядностью будет слева дополнен нулями (старшие разряды) так, чтобы соответствовать разрядности операнда с большей разрядностью.

В примере 5 показано то, как можно использовать условный оператор для того, чтобы выполнить шину с третьим состоянием.

```
wire [15:0] busa = drive_busa ? data : 16'bz;
```

Пример 5. Применение условного оператора для того, чтобы выполнить шину с третьим состоянием

Шина **busa** управляется сигналом **drive_busa**. В том случае, когда этот сигнал 1, на шину передаются данные — **data**, в противном случае шина переключается в третье состояние. Если значение сигнала **drive_busa** неизвестно, то и значение данных на шине **busa** тоже неизвестно.

Конкатенация — Concatenation

Конкатенация — объединение битов, находящихся в двух или более выражениях. Для записи конкатенации используются символы фигурных скобок { и }, с запятыми, отделяющими выражения, находящиеся в этих скобках. Пример 6 объединяет четыре выражения.

```
{a, b[3:0], w, 3'b101}
```

Пример 6. Применение оператора конкатенации для того, чтобы объединить четыре выражения

Пример 6 эквивалентен другой записи, которая приведена в примере 7.

```
{a, b[3], b[2], b[1], b[0], w, 1'b1, 1'b0, 1'b1}
```

Пример 7. Применение оператора конкатенации

Постоянные числа, разрядность которых не указана, не должны быть использованы в объединениях, потому что разрядность каждого операнда в конкатенации необходима для того, чтобы вычислить полный размер шины для конкатенации. Конкатенации могут быть записаны при использовании множителя повторения так, как показано в примере 8.

```
{4{w}} // This is equivalent to {w, w, w, w}
```

Пример 8. Применение оператора конкатенации с использованием множителя повторения

В примере 9 показаны вложенные конкатенации. Число, указывающее, сколько раз надо повторить конкатенацию, должно быть постоянной величиной.

```
{b, {3{a, b}}} // This is equivalent to
// {b, a, b, a, b, a, b}
```

Пример 9. Применение оператора конкатенации с использованием вложенных конкатенаций

Числа

Постоянные числа могут быть определены в десятичном (**d** or **D**), шестнадцатеричном (**h** or **H**), восьмеричном (**o** or **O**) или в двоичном (**b** or **B**) формате.

Они могут произвольно начинаться со знака + или — и могут быть написаны в одном из следующих форматов.

1. <size>'<base><number>: самая полная форма для записи числа.
2. <base><number>: эта форма записи использует разрядность по умолчанию и является машинно-зависимой, но, по крайней мере, равняется 32 битам.
3. <number>: эта форма записи использует разрядность по умолчанию для десятичного формата.

Наиболее простой формат 3 — беззнаковое десятичное число, содержащее, в любом порядке, последовательность цифр от 0 до 9. Хотя пользователь, возможно, не определил размерность написанного числа, Verilog сам вычисляет разрядность числа, примененного в выражении. Обычно в выражениях разрядность примененных в операторах чисел одинакова. И для использования выбирают требуемое число битов, начиная с наименьшего значащего бита.

Формат 1 определяет постоянный размер для разрядности и записывается следующим образом:

```
aa...a'sf nn…n.
```

При этом неизвестные и высокоимпедансные значения можно использовать во всех числах, кроме десятичных. В каждом случае применение символов **x** или **z** представляет данное число битов, имеющих состояние **x — неизвестное значение** или **z — высокий импеданс**. При записи чисел, кроме **z**, может быть использован символ **?**. Его применяют в операторе ***case***, для того, чтобы улучшить читаемость текста.

Если мы имеем два числа, и одно из них имеет меньшую разрядность, чем другое, то в таком случае для меньшего числа производится заполнение старших разрядов нулями так, чтобы оба числа имели одинаковую разрядность. Если же в числе с большей разрядностью старшие разряды заполнены значениями **x** или **z**, то и число с меньшей разрядностью тоже будет заполнено значениями **x** или **z**.

Символ подчеркивания может вставляться в числа с любым основанием для того, чтобы улучшить читаемость кода. Но, конечно, символ подчеркивания не может быть первым символом в записи числа. Приведем примеры двоичных чисел: 12'b 0x0x_1101_0zx1 и 12'b 0x0x11010zx1. Как мы видим, первое число читать значительно легче. А это пример неправильной записи: 8'b_0001_1010.

Примеры записи чисел, в которых не определена разрядность:

- 792 — десятичное число;
- 'h 7d9 — шестнадцатеричное число;
- 'o 7746 — восьмеричное число.

В примере 10 показаны формы записи чисел, в которых определена разрядность.

```
12'h x    // 12-битовое число с неопределенным значением
8'h fz    // 8-разрядное шестнадцатеричное число, эквивалентное
          // 8-разрядному двоичному числу 8 'b 1111_zzzz
10'd 17   // 10-разрядное число, равное 17
```

Пример 10. Примеры записи чисел, в которых определена разрядность

В примере 11 показаны формы записи чисел со знаком и отрицательных чисел. А в примере 12 показаны формы записи чисел со знаком «**?**».

```
–6 'd 5   // 6-разрядное число, равное -5 (то есть 111011)
6 'd –5   // такая запись — неправильна!
3 'sd 7   // знаковое число, имеющее разрядность в 3 бита, равное
          // –1, то есть децимальное число 7, представленное
          // тремя битами, как 111. Буква s указывает на то, что
          // число представлено как знаковое (signed),
          // соответственно, значение такого числа равно минус 1.
4 'sh F   // знаковое число, имеющее разрядность в 4 бита, равное
          // –1, то есть шестнадцатеричное число F, представленное
          // четырьмя битами, как 1111. Буква s указывает на то, что
          // число представлено как знаковое (signed),
          // соответственно, значение такого числа равно минус 1.
```

Пример 11. Примеры записи чисел со знаком и отрицательных чисел

```
12'd?   // 12-битовое десятичное число с записью высокого
        // импеданса как 'don't-care'
```

Пример 12. Примеры записи чисел со знаком «**?**»

Вещественные числа могут быть записаны в десятичном или научном формате. Если число записано в десятичном формате, оно должно иметь, по крайней мере, одну цифру после десятичной точки (пример 13).

```
1.8
3_2387.3398_3047
3.8e10 // e or E для основания экспоненты
2.1e-9
3.      // неправильная запись
```

Пример 13. Примеры записи вещественных чисел, которые могут быть записаны в десятичном или научном формате

Преобразование вещественного числа в целое число

Язык Verilog преобразовывает вещественные числа в целые, округляя вещественное число к самому близкому целому числу, а не отбрасывая десятичную часть. Например, вещественные числа 35,7 и 35,5 и округляются до значения 36, а 35,2 становится 35. Неявное преобразование имеет место, когда вы назначаете вещественное число как целое число (real to an integer).

Строки

Строка — это последовательность символов, ограниченная слева и справа кавычками. Они должны быть расположены на одной строке. В строке можно использовать специальные ESC-символы, для чего необходимо применять следующим образом символ «\»:

- **\n** печатать текст с новой строки. Аналогично применению клавиши RETURN.
- Символ табуляции **\t**. Эквивалентно тому, что происходит при нажатии на клавишу табуляции.
- Применение двух символов \\ является эквивалентом применения символа «\».
- \« является символом «.
- **\ddd** — символ ASCII, определяющий цифру. (В этой записи использовано от одной до трех восьмеричных цифр).
- **%%** — это печать знака %.

В примере 14 показаны правильная и неправильная формы записи строки.

```
«hello world»;          // правильно написанная строка
«good
  b
     y
        e
           wo
              rld»;  // неправильно написанная строка
```

Пример 14. Примеры записи строки (правильная и неправильная формы записи)

Параметры (Parameters)

В языке Verilog параметры не принадлежат к какому-либо регистру или группе цепей. Параметры не переменные, они — константы. Параметры могут быть как местные, локальные, так и глобальные. Синтаксис для объявлений параметров показан в примере 15.

```
<parameter_declaration>
::= parameter
<list_of_assignments>;
```

Пример 15. Синтаксис для объявлений параметров — <parameter_declaration>

Примечание: *в некоторых случаях при использовании параметров можно определять диапазон при объявлении параметра.*

<list_of_assignments> — список назначений, разделенный запятыми, где правая сторона назначения должна быть постоянным выражением, то есть выражением, содержащим только постоянные числа и предварительно определенные параметры. В следующем примере 16 показано объявление параметров.

```
parameter msb = 7;          // defines msb as a constant value 7
parameter e = 25, f = 9;    // defines two constant numbers
parameter r = 5.7;          //declares r as a 'real' parameter
parameter byte_size = 8, byte_mask = byte_size — 1;
parameter average_delay = (r + f) / 2;
```

Пример 16. Объявление параметров

Хотя параметры в Verilog представляют собой константы, они могут быть изменены во время компиляции для того, чтобы принять новые значения, которые отличаются от определенных в назначении объявления. Это позволяет пользователю настраивать инстансы модулей при их установке в проекте. Вы можете изменить параметр утверждением defparam или изменить параметр в утверждении инстанса модуля. Обычно параметры используются для задания значения времени задержки или разрядности (ширины) переменных.

Функции

Определение функции должно начаться с ключевого слова **функция — function**, сопровождаемого дополнительным ключевым словом **автоматическая — automatic** и дополнительным указателем **signed**, диапазоном или типом возвращаемого значения из функции, затем идет само название функции, потом или точка с запятой, или список портов функции, заключенных в круглые скобки, и точка с запятой. И, наконец, определение должно закончиться ключевым словом **endfunction**. Дополнительно можно использовать параметр *range_or_type*. Если функция определена по умолчанию, то есть без диапазона или типа, то ее представляют как функцию, возвращающую значение от однобитового регистра. Если используется параметр *range_or_type*, то необходимо определить, что представляет собой возвращаемое значение функции — реальное, целое число, время, реальное время или значение с диапазоном битов — [n:m]. Функция должна иметь по крайней мере один объявленный вход (одну входную переменную).

Ключевое слово **automatic (автоматический)** объявляет рекурсивную функцию со всеми функциональными объявлениями, распределенными динамически для каждого рекурсивного обращения. К автоматическим вызовам функции нельзя обратиться при помощи иерархических ссылок. Автоматические функции могут быть вызваны по их иерархическому названию.

Входы функции могут быть объявлены двумя способами. В первом способе необходимо дать название функции, сопровождаемое точкой с запятой. После точки с запятой должны следовать одно или более объявлений входов, которые могут быть произвольно установлены при объявлении тем блока. После объявлений тем для функции должна следовать «начинка» функции, то есть поведенческие утверждения (behavioral statement) и затем ключевое слово **endfunction**. Пример 17 определяет функцию, названную *getbyte*, используя при этом спецификацию диапазона.

По второму способу (пример 18) необходимо дать название функции, сопровождаемое открывающей скобкой, и одним или более входными объявлениями, которые отделяются запятыми. После всех объявлений входов должна быть закрывающая круглая скобка и точка с запятой. После точки с запятой могут идти объявления тем блока, сопровождаемые поведенческими утверждениями и затем ключевым словом **endfunction**.

```
function [7:0] getbyte;
input [15:0] address;
begin
    // code to extract low-order byte from addressed word
    . . .
    getbyte = result_expression;
end
endfunction
```

Пример 17. Определение функции getbyte при использовании спецификации диапазона

```
function [7:0] getbyte (input [15:0] address);
begin
    // code to extract low-order byte from addressed word
    . . .
    getbyte = result_expression;
end
endfunction
```

Пример 18. Определение функции getbyte при использовании перечисления портов в скобках

Значения, возвращаемые из функции

Определение функции неявно объявляет и переменную, которая используется в проекте. Переменная должна иметь то же самое название, что и функция. По умолчанию, эта переменная имеет то же самое значение, что и 1-битовый регистр. Или, в общем случае, переменная имеет то же самое значение, что и тип, определенный в объявлении функции. Определение функции устанавливает возвращаемое из функции значение, результат работы этой функции на внутренней переменной будет иметь то же самое название, что и функция. Попытка объявить другой объект с таким же названием, как у уже объявленной функции, и в тех пределах, где данная функция уже объявлена, приведет к ошибке. То же самое происходит и в пределах функции. В каждой функции есть своя переменная с названием функции, которая может использоваться в выражениях. Поэтому и в пределах данной функции неправильно объявлять другой объект с тем же самым названием, как и данная функция.

В следующем примере иллюстрируется это понятие. Для функции *getbyte*, определенной в примере 18, возвращаемое значение будет выглядеть следующим образом:

```
getbyte = result_expression;
```

В примере 18, использующем ту же функцию *getbyte*, создается новое слово, представляющее собой конкатенацию результатов двух вызовов функции:

```
getbyte : word = control ? {getbyte(msbyte), getbyte(lsbyte)}:0;
```

Правила работы с функциями

При использовании функции имеют больше ограничений, чем задачи. Сформулируем шесть правил использования функций:

1. Определение функции не должно содержать утверждений, которые связаны с управлением по времени, то есть не должно содержать любых утверждений, содержащих **#**, **@** или **wait**.
2. Функции не могут запускать задачи — **task**.
3. Определение функции должно содержать, по крайней мере, один входной аргумент.
4. Определение функции не должно иметь аргументов, объявленных как выход или как двунаправленный вход/выход.
5. Определение функции должно включать в себя назначение результата работы функции на внутреннюю переменную, которая имеет то же самое название, что и имя функции.
6. Функция не должна иметь неблокирующих назначений.

В примере 19 показано определение функции с именем *factorial*, которая возвращает целочисленное значение. Функция *factorial* вызывается итерационно, и результаты ее работы выводятся на монитор.

```
module tryfact;
// define the function
function automatic integer factorial;
input [31:0] operand;
integer i;
  if (operand >= 2)
      factorial = factorial (operand — 1) * operand;
  else
      factorial = 1;
endfunction

// test the function
integer result;
integer n;
initial
  begin
  for (n = 0; n <= 7; n = n+1)
      begin
          result = factorial(n);
          $display(«%0d factorial=%0d», n, result);
      end
  end
endmodule // tryfact
```

Пример 19. Пример вызова функции factorial

Результаты симуляции будут следующие:

```
0 factorial=1
1 factorial=1
2 factorial=2
3 factorial=6
4 factorial=24
5 factorial=120
6 factorial=720
7 factorial=5040
```

Использование постоянных функций (constant functions)

Вызовы функций постоянных значений (Constant function calls) используются для того, чтобы поддержать возможность проведения вычислений значений параметров во время разработки. Вызов функций постоянных значений должен быть вызовом функции для вычисления постоянного значения, которое находится локально в том модуле, где выполнен вызов, и аргументы функции представляют собой также постоянные выражения.

Функции постоянных значений представляют собой подмножество нормальных функций Verilog, которые должны иметь следующие ограничения:

- Они не должны содержать иерархические ссылки.
- Любая функция, вызванная в пределах функций постоянных значений, должна быть вызовом функции для вычисления постоянного значения, которое находится в текущем модуле, в котором и выполнен вызов. Системные же функции не должны вызываться.
- Все системные задачи в пределах постоянной функции должны игнорироваться. Единственная системная задача, которая может быть вызвана, это команда $display, но и она должна игнорироваться при вызове во время разработки.
- Все значения параметра, используемые в пределах функции, должны быть определены перед вызовом функции. Все идентификаторы, которые не являются параметрами или функциями, должны быть объявлены локально для данной функции. Если эти идентификаторы используют какое-нибудь значение параметра, которое затронуто прямо или косвенно, при использовании утверждения **defparam**, то результат будет не определен. Это может привести к ошибкам, или постоянная функция может возвратить неопределенное значение.
- Они не должны быть объявлены в пределах выполнения оператора **generate**.
- Они сами не должны использовать постоянные функции в любом контексте, требующем постоянного выражения. Вызовы функций постоянных значений выполняются во время разработки. Их выполнение не имеет никакого воздействия на начальные значения переменных, используемых или во время моделирования, или среди многократных вызовов функции во время разработки. В каждом из этих случаев переменные будут инициализированы, так же как и при обычном моделировании. В примере 20 представлено определение функции, названной clogb2, которая возвращает целое число, равное округленному в большую сторону значению логарифма по основанию 2.

```
module ram_model (address, write, chip_select, data);
parameter data_width = 8;
parameter ram_depth = 256;
localparam adder_width = clogb2(ram_depth);
input [adder_width — 1:0] address;
input write, chip_select;
inout [data_width — 1:0] data;
//define the clogb2 function
function integer clogb2;
  input depth;
  integer i,result;
  begin
  for (i = 0; 2 ** i < depth; i = i + 1)
      result = i + 1;
      clogb2 = result;
  end
endfunction
reg [data_width — 1:0] data_store[0:ram_depth — 1];
//the rest to the ram model

// Вызов функции ram_model с параметрами:
ram_model #(32,421) ram_a0(a_addr,a_wr,a_cs,a_data);
```

Пример 20. Определение функции, названной clogb2, которая возвращает целое число, равное округленному в большую сторону значению логарифма по основанию 2

Цепи, регистры, память, представление чисел и времени (Wire Registers, Memories, Integers and Time)

Цепь — wire

Цепи на схеме соответствуют физическим проводам, которые подключаются ко всем компонентам схемы. Разрядность цепей по умолчанию — один бит. Цепи не хранят значения сигнала, и для того, чтобы состояние сигнала в цепи было определено, какой-либо источник сигнала должен непрерывно управлять ею. Если цепь имеет несколько источников сигнала — драйверов (например, два выхода вентилей подключены к одной цепи), то значение результирующего сигнала в цепи имеет значение, согласно тому, какой тип цепи используется. Названия цепей и выполняемые ими функции приведены в таблице 16.

Таблица 16. Названия цепей и выполняемые ими функции

Название	Функция	Описание
wire		Для этой цепи, если все драйверы имеют одно и то же значение, тогда **wire** принимает это же значение. Если все драйверы, кроме одного, имеют значение **Z**, тогда **wire** принимает значение того драйвера, который имеет значение выходного сигнала, не равное **Z**. Если два или больше не-**Z** драйвера имеют различную мощность выходного каскада, то **wire** принимает значение более сильного драйвера. Если два драйвера равной силы имеют различные значения, то **wire** принимает значение **x**.
wand	AND	Проводное «И» wand/triand: 0 | 1 | x | z 0: 0 0 0 0 1: 0 1 x 1 x: 0 x x x z: 0 1 x z
wor	OR	Проводное «ИЛИ» wor/trior: 0 | 1 | x | z 0: 0 1 x 0 1: 1 1 1 1 x: x 1 x x z: 0 1 x z
tri	Tri	Цепь с третьим состоянием
triand	Tri + AND	Цепь с третьим состоянием, выполняющая проводное «И» wand/triand: 0 | 1 | x | z 0: 0 0 0 0 1: 0 1 x 1 x: 0 x x x z: 0 1 x z
trior	Tri + OR	Цепь с третьим состоянием, выполняющая проводное «ИЛИ» wor/trior: 0 | 1 | x | z 0: 0 1 x 0 1: 1 1 1 1 x: x 1 x x z: 0 1 x z
tri0	Tri + pulldown	Когда эта цепь не управляется, тогда ее значение — 0. Моделируется функция **pulldown**
tri1	Tri + pullup	Когда эта цепь не управляется, тогда ее значение — 1. Моделируется функция **pullup**
trireg		Цепь ведет себя как **wire**, за исключением того, что, когда все драйверы сети находятся в высоком импедансе — **Z**-состоянии, тогда цепь сохраняет свое последнее значение, которое было, когда эта цепь управлялась. **trireg** используются для того, чтобы моделировать емкость в данной цепи
supply0		Моделируют цепи, которые связаны с «землей»
supply1		Моделируют сети, которые связаны с питанием

Регистры

Регистры — это термин, применяемый для названия класса устройств, применяемых в цифровых схемах. Эти устройства могут запоминать информацию и хранить ее. Они определяются ключевым словом **reg** и могут иметь произвольную разрядность. Размер значения по умолчанию — 1 бит. Пример записи такого регистра: **reg My1BitReg** — определяет регистр, названный **My1BitReg** и имеющий разрядность в 1 бит, в то время как запись **reg [15:0] My16BitReg** определяет 16-разрядный регистр, названный **My16BitReg**. Регистры, имеющие разрядность, равную 1 биту, называют ***скаляром*** *(scalar)*, а регистры, имеющие разрядность более 1 бита , называют ***вектором*** *(vector)*. Название разрядов шины начинается со старшего бита (в этом случае это бит, имеющий номер 15) и заканчивается младшим битом шины. Но регистр может быть объявлен и так — **reg [0:15] My16BitReg**, при этом старший бит в названии шины будет иметь номер 0. Соответственно, и все другие биты шины будут пронумерованы по-другому.

Далее, регистр может быть объявлен как знаковый — **reg signed [15:0] My16BitReg**. Такая форма записи указывает на то, что при работе с этим регистром данные должны быть обработаны как число со знаком (двоичное дополнение).

Общая форма записи регистра показана в примере 21.

```
reg_declaration reg [signed] [range] list_of_variable_identifiers;
list_of_variable_identifiers variable_type { , variable_type }
variable_type
variable_identifier [ = constant_expression]
| variable_identifier dimension {dimension}
variable_identifier
identifier
dimension
[dimension_constant_expression : dimension_constant_expression ]
```

Пример 21. Общая форма записи регистра

В выражениях языка можно использовать как один бит, так и несколько соседних битов векторного регистра (или шины). Выбор одного бита из регистра или шины называют ***выбором бита*** *(bit-select)*, а выбор нескольких соседних битов называется ***выбор группы*** *(part-select)*.

Следующий пример показывает выбор бита из *acc*, этот бит адресуется операндом *index*:

```
acc[index]
```

Выбор части битов из регистра или цепи показан на следующем примере:

```
vect[ms_expr:ls_expr]
Форма записи регистров показана в примере 22.
reg [10:0] counter; reg a; reg [2:0] b;
reg [-5:7] c
a = counter [7]; // бит номер 7 из counter загружен в a
b = counter [4:2]; //биты номер 4, 3, 2 из counter загружены в b
```

Пример 22. Примеры записи регистров

Для выбора бита, можно пользоваться выражениями или литералами. Группы битов, выбранные как часть регистра или шины, должны быть определены выражениями или литералами, имеющими определенную разрядность, и эти значения могут быть положительными или отрицательными.

Блоки памяти

Память описывается как массив регистров. Вот синтаксис для описания такого массива:

```
reg [<memory_width>] <reg_name> [<memory_depth>];
```

Блоки памяти определяются при использовании объявления регистра (пример 23):

```
reg [10:0] lookUpTable [0:31];
```

Пример 23. Пример объявления блока памяти

Так определяется массив из 32 слов, названный **lookUpTable**, где каждое слово имеет разрядность в 11 битов. Память используется в выражениях следующим образом, например,

```
lookUpTable [5] = 75;
```

Это запись показывает, что пятое слово в памяти **lookUpTable** загружено значением 75. Блоки памяти могут быть многомерными. Вот так объявляются двух- и трехмерные массивы (пример 24).

```
reg [7:0] twoDarray [15:0] [63:0];
reg [9:0] threeDarray [31:0] [31:0] [43:0];
```

Пример 24. Примеры объявлений двух- и трехмерных массивов

Если 8-битовый регистр **a_reg** уже объявлен, то запись данных в этот регистр выглядит следующим образом:

```
a_reg = twoDarray[3][2];
```

Группы битов, выбранные как часть регистра или шины, и выбор бита могут быть применены и к блокам памяти. Чтобы выбрать часть битов памяти, к ней, в конце записи, добавляется еще один индекс. Таким образом, выбор битов от бита 7 и до бита 0 из памяти threeDarray может быть записан так (пример 25):

```
a = threeDarray [21] [4] [40] [7:0];
```

Пример 25. Выбор битов от бита 7 и до бита 0 из памяти threeDarray

Чтение элементов массива памяти вне диапазона, определенного для чтения, возвращает неизвестное значение — **x**. Запись в элементы массива памяти вне диапазона, определенного для записи, не имеет никакого эффекта. Выбрать одновременно больше чем одно слово из памяти невозможно.

Целые числа и время

Регистры используются для того, чтобы моделировать аппаратные средства. Иногда бывает полезно выполнить дополнительные вычисления в целях моделирования. Например, мы можем захотеть выключить какое-либо управление в проверяемом узле после того, как прошло определенное время при симуляции. Если мы используем регистры с этой целью, то операции на них могут быть перепутаны с действиями фактических аппаратных средств. ***Целые числа*** *(Integer)* и переменные ***времени*** *(time)* обеспечивают возможность для описания вычислений, необходимых для моделирования. Они делают описание более наглядным и самодокументируемым и облегчают сопровождение проектов. Для объявления целых чисел используется ключевое слово ***integer*** и далее идет список переменных. Объявление времени — то же самое, за исключением того, что используется ключевое слово ***time*** (пример 26).

```
integer a, b;          // два целых числа
integer c [1:100];  // массив целых чисел
time q, r;             // две переменные для времени
time s [31:0];       // массив времен
```

Пример 26. Объявления целых чисел и времени

Целое число обычно имеет разрядность в 32 бита. Операции с целым числом, как правило, выполняются в двоичном коде с дополнением, поэтому старший бит указывает признак знака. Переменная времени — это обычно 64-битовая переменная, используемая с системной функцией $time.

Системные задания и функции (Display and Write Tasks)

Стандартные системные задания и функции и их краткие описания приведены в таблице 17.

Вывод на монитор

Основное системное задание — это $display (табл. 18). С его помощью можно просмотреть переменные, строки, выражения. Оно чаще всего употребляется на практике.

Пример использования:

```
$display(p1, p2, p3,....., pn);
```

где p1, p2, p3,..., pn могут быть текстовыми строками, переменными или выражениями. Формат системного задания $display аналогичен printf в языке Си. По умолчанию системное задание $display добавляет новую строку при выводе текста на монитор. $display без параметров используется для перехода на новую строку.

В отличие от $display, системное задание $write при выводе данных на монитор не выполняет переход на новую строку.

Инициализация памяти

Представим себе, что мы применяем в модели память. Как было уже сказано, память описывается как массив регистров. Вот синтаксис для описания массива регистров:

```
reg [<memory_width>] <reg_name> [<memory_depth>];
```

Как же записать данные при инициализации проекта в такую память? Системная функция $readmemb как раз и позволяет читать бинарные данные из файла и загружать эти данные в моделируемую память. Синтаксис этой функции:

```
$readmemb («<file_name>», <reg_name>, <start_address>, end_address>);
```

где <file_name> — имя и путь к файлу, содержащему двоичные данные, параметр <reg_name> — это 2D-массив регистров, в котором хранятся данные, и, как опции, две цифры, которые указывают начальный и конечный адрес данных.

Файл инициализации может содержать только двоичные данные, пробелы и комментарии. Функцию $readmemb обычно вызывают при инициализации памяти в блоке initial. Функция $readmemh аналогична функции $readmemb, но работает с шестнадцатеричными данными.

Необходимо отметить, что если ранее программа XST, поставляемая фирмой Xilinx,

Таблица 17. Стандартные системные задания и функции

Формат записи	Описание	
$display, $write	Вывод информации на монитор	utility to display information
$fdisplay, $fwrite	Запись информации в файл	write to file
$strobe, $fstrobe	Вывод информации на монитор/запись данных	display/write simulation data
$monitor, $fmonitor	Вывод информации на монитор, запись информации в файл	monitor, display/write information to file
$time, $realtime	Текущее время симуляции	current simulation time
$finish	Выход из режима симуляции	exit the simulator
$stop	Останов в режиме симуляции	stop the simulator
$setup	Установить проверку времени	setup timing check
$hold, $width	Установить проверку по времени для параметров hold/width	hold/width timing check
$setuphold	Установить проверку по времени для параметров hold and setup	combines hold and setup
$readmemb/$readmemh	Читает память данными из файла симуляционного паттерна	read stimulus patterns into memory
$sreadmemb/$sreadmemh	Загружает память данными из файла симуляционного паттерна	load data into memory
$getpattern	Обработка файла симуляционного паттерна	fast processing of stimulus patterns
$history	Распечатать историю выданных команд	print command history
$save, $restart, $incsave	Сохранение, перезапуск, частичное сохранение	saving, restarting, incremental saving
$scale	Подстройка параметров временной шкалы по параметрам из другого модуля	scaling timeunits from another module
$scope	Переключение на соответствующий уровень иерархии	descend to a particular hierarchy level
$showscopes	Полный список поименованных блоков, заданий, модулей	complete list of named blocks, tasks, modules
$showvars	Показать переменные	show variables at scope

Таблица 18. Формат команды Display

Формат записи	Описание	
%d or %D	Вывод на монитор переменной в десятичном формате	Display variable in decimal
%b or %B	Вывод на монитор переменной в двоичном формате	Display variable in binary
%s or %S	Вывод на монитор строки	Display string
%h or %H	Вывод на монитор переменной в шестнадцатеричном формате	Display variable in hex
%c or %C	Вывод на монитор символа в коде ASCII	Display ASCII character
%m or %M	Вывод на монитор иерархического имени	Display hierarchical name (no argument required)
%t or %T	Вывод на монитор параметра времени	
%v or %V	Вывод на монитор параметра strength	Display strength

не имела возможности выполнять инициализацию памяти таким образом, но теперь такая возможность уже есть. Пример, показывающий чтение двоичных данных из файла:

```
reg [31:0] prom_data[1023:0];
initial
    $readmemb(«../data/mem_file.dat», prom_data);
```

Кроме бинарных или шестнадцатеричных данных, пробелов и комментариев, файлы инициализации могут также содержать адресные метки:

```
label (сопровождаемая выражением @<address>).
```

В примере 27 показана форма записи данных в файле инициализации памяти.

```
// This is a comment — комментарии

1111000011110000        // Запись 16-битового числа в первый адрес
1010_0101_1010_0101     // запись числа во второй адрес,
                        // символы подчеркивания применяются
                        // для облегчения чтения данных

// Если мы хотим поменять текущий адрес, то
@025
// теперь в этот адрес — 025, будет записано число
11111111_00000000

// Адрес также может быть помещен в ту же строку
@035 00000000_11111111
```

Пример 27. Форма записи данных в файле инициализации памяти

Продолжение следует

Литература

1. IEEE Standard Verilog Hardware Description Language, IEEE Computer Society, IEEE, New York, NY, IEEE Std 1364-2001.
2. IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language, IEEE Computer Society, IEEE, New York, NY, IEEE Std 1364-1995.
3. Bergeron J. Writing Testbenches. Functional Verification of HDL Models. Kluwer Academic Publishers, 2000.
4. Thomas D. E., Moorby P. R. The Verilog® Hardware Description Language. Fifth Edition. Kluwer Academic Publishers, 2002.
5. Palnitkar S. Verilog HDL: A Guide to Digital Design and Synthesis. Second Edition. Prentice Hall PTR, 2003.
6. HDL Chip Design. Douglas Smith, 3rd edition, 1997.
7. Madhavan R. Quick Reference for Verilog HDL. AMBIT Design Systems, Inc.
8. Емец С. Verilog — инструмент разработки цифровых электронных схем // Схемотехника. 2001. № 1–4.
9. Поляков А. К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. М.: Солон, 2003.
10. Стешенко В. Б. ПЛИС фирмы ALTERA: элементная база, система проектирования и языки описания аппаратуры. М.: Додэка-XXI, 2007.
11. Xilinx Synthesis Technology (XST) User Guide. 1991-2000 Xilinx.
12. Using Verilog to Create CPLD Designs. XAPP143 (v1.0) August 22, 2001.
13. Verilog Tutorial. Deepak Kumar Tala. 2005. **http://www.asic-world.com**