

## 1.0 RTL Verilog

This document describes the RTL Verilog language elements and their organization as supported by Verifiable RTL principles.

- ✓ Check mark highlights Verifiable RTL rules.
- 📖 Book denotes language elements reserved for libraries.

## 2.0 Reserved Words

**bold** - supported for Verifiable RTL Design

**italic** - unsupported for Verifiable RTL Design ✓

<b>always</b>	<b>end</b>	<b>inout</b>	<b>posedge</b>
<b>assign</b>	<b>endcase</b>	<b>input</b>	<b>reg</b>
<b>begin</b>	<b>endfunction</b>	<b>module</b>	<b>tri</b>
<b>case</b>	<b>endmodule</b>	<b>negedge</b>	<b>tri0</b>
<b>casex</b>	<b>function</b>	<b>or</b>	<b>tri1</b>
<b>default</b>	<b>if</b>	<b>output</b>	<b>wire</b>
<b>else</b>	<b>initial</b>	<b>parameter</b>	

  

<i>and</i>	<i>highz0</i>	<i>pullup</i>	<i>table</i>
<i>buf</i>	<i>highz1</i>	<i>rcmos</i>	<i>task</i>
<i>bufif0</i>	<i>ifnone</i>	<i>real</i>	<i>time</i>
<i>bufif1</i>	<i>integer</i> 📖	<i>realtime</i>	<i>tran</i>
<i>casez</i>	<i>join</i>	<i>release</i>	<i>tranif0</i>
<i>cmos</i>	<i>large</i>	<i>repeat</i>	<i>tranif1</i>
<i>deassign</i>	<i>macromodule</i>	<i>rnmos</i>	<i>triand</i>
<i>defparam</i>	<i>medium</i>	<i>rpmos</i>	<i>trior</i>
<i>disable</i>	<i>nand</i>	<i>rtran</i>	<i>triereg</i>
<i>edge</i>	<i>nmos</i>	<i>rtranif0</i>	<i>vectored</i>
<i>endprimitive</i>	<i>nor</i>	<i>rtranif1</i>	<i>wait</i>
<i>endspecify</i>	<i>not</i>	<i>scalared</i>	<i>wand</i>
<i>endtable</i>	<i>notif0</i>	<i>small</i>	<i>weak0</i>
<i>endtask</i>	<i>notif1</i>	<i>specify</i>	<i>weak1</i>
<i>event</i>	<i>pmos</i>	<i>specparam</i>	<i>while</i>
<i>for</i> 📖	<i>primitive</i>	<i>strong0</i>	<i>wor</i>
<i>force</i>	<i>pull0</i>	<i>strong1</i>	<i>xnor</i>
<i>forever</i>	<i>pull1</i>	<i>supply0</i>	<i>xor</i>
<i>fork</i>	<i>pulldown</i>	<i>supply</i>	

## 3.0 Lexical elements

- **Names** - Begin with a- z A- Z or **\_**, and may contain numbers 0-9. Examples: *r\_ralph124c41*, *benya*, *krik*. *Names containing dollar signs are not supported.* ✓
- **Unsigned decimal integers** - Specify bit ranges, memory sizes, time. *High bit maximum: 511.* ✓ Examples: *[7:0]*, *[0:255]*, *#1*.
- **Sized integers** - use to represent bits. May be binary, octal, hex or decimal. *Maximum width: 512 bits.* ✓ Examples: *8'b0011\_1001*, *2'o2*, *13'h03f*, *9'd255*
- Use "z" values only in binary to directly drive tri- state **output** or **inout** ports. "x" values are not supported in verifiable RTL Verilog. ✓

## 4.0 Compiler Directives

- **'include** " <file> "
- **'define** <name> <text to comment or end-of-line>
- **'ifdef** <name>
- **'else**
- **'endif**
- **'timescale**

Unsupported compiler directives: *'default\_nettype*, *'unconnected\_drive*, *'nounconnected\_drive*, *'resetall*.

## 5.0 RTL Module

### 5.1 Module organization

**module** *asic8* (*port declarations*);

*port directions*

*port types* (" *reg*" for non- tri- state outputs)

*global reg declarations*

*function declarations*

**always** @( *sensitivity\_list* )

**begin**

*procedural statements (combinational logic)*

**end**

**assign** *statements (combinational logic)*

*tri- state port driving tri statements*

*storage element (flip- flop, latch, memory) instantiations*

**endmodule**

### 5.1.1 Port declarations

**module** *asic8* (*clk*, *scan\_in*, *scan\_ctl*, *bdp*, *scan\_out*);

**input** *clk*, *scan\_in*;

**input** [1: 0] *scan\_ctl*;

**inout** [0: 17] *bdp*;

**output** *scan\_out*;

**output** [43: 0] *r\_addr*;

### 5.1.2 Reg, wire declarations

**reg** *scan\_out*; // declare procedural block outputs "reg"

**reg** *c\_b0\_bnk0\_busy*; // one bit variable

**reg** [7: 0] *r\_p0\_l0\_zone*; // 8- bit variable

**reg** [0: 1] *M\_valid* [0: 4095]; // 4096x2 memory 📖

**wire** [43: 0] *r\_addr*; // declare module instance outputs, // assign statement targets "wire"

### 5.1.3 Functions

**function** [*< bit\_range >*] *function\_name*;

*inputs*

[*local reg declarations*]

**begin**

*procedural statements*

**end**

**endfunction**

Example:

```
function csa_s1;
input x;
input y;
input z;
begin
    csa_s1 = expression;
end
endfunction
```

### 5.1.4 Tristate port driving statements

- Use to drive all **inout** and tri- state **output** type ports.
- Set port name equal to a single term. Put logic expression driving that term back in *eval* or *eval\_out* task. Example:

*// inside procedural statement block*

*c\_bdp = (~ csa\_& ~wa) ? r\_a\_reg : 18'bz;*

**tri** [0: 17] *bdp* = *c\_bdp*; // preceding library module // instances

### 5.1.5 Library module instantiation

- All storage elements (flip- flops, memories, latches) must be instantiations of modules from libraries. 📖
- Library module instantiation content:  
*typename (bit\_width) instname (outname, clock, innames);*
- Example:

*DFFT\_X #(44) reg\_addr (r\_addr[ 43: 0], clock, c\_addr[ 43: 0]);*

## 5.2 Procedural statements

A *procedural block* is a statement or series of statements enclosed within a **begin end**

### 5.2.1 Control

**if** (*expression*)

*statement or procedural block*

**else** // if with no else is a latch; latches must be in library. 📖  
*statement or procedural block*

**case** (*scan\_ctl*)

*2'd0 : statement or procedural block*

*2'd1 : statement or procedural block*

**default** : *statement or procedural block*

**endcase**

**case**x ( { *i\_scan*, *c\_adv\_q*, *c\_q1\_act*, *c\_q0\_act* } )

*4'b1\_?\_??: c\_q0ry = r\_q1\_rwy;*

*4'b0\_0\_?0: c\_q0ry = {c\_ry\_qctl, c\_par\_data};*

*4'b0\_0\_?1: c\_q0ry = r\_q0\_ry;*

*4'b0\_1\_00: c\_q0ry = {2'h0, c\_ry\_qctl[ 2:8], c\_par\_data};*

*4'b0\_1\_01: c\_q0ry = {c\_ry\_qctl, c\_par\_data};*

*4'b0\_1\_1?: c\_q0ry = r\_q1\_ry;*

**endcase**

## 5.2.2 Assignments

- Non- blocking: targets must be type reg; use for flip- flop assignments only in libraries: `reg`  
`r_ icq_ 0 <= c_ icq_ 0;`
- Blocking: targets must be type reg; use in procedural blocks and functions for combinational variable and output assignments:  
`c_ i0_ btab_ active = c_ i0_ btab_ out[ 24];`
- Continuous assignments: targets must be type wire; outside procedural blocks:  
`assign c_ tab_ reset_ ctr = r_ tab_ reset_ ctr + 5'h01;`

## 5.3 Operators

### 5.3.1 Binary operators

Arithmetic (2's complement)	+	<b>a + b</b>	<b>addition</b> (note 1 )
	-	<b>a - b</b>	<b>subtraction</b> (note 1 )
	%	<b>a % b</b>	<b>modulo</b>
Bit-Wise (note 1 )	&	<b>a &amp; b</b>	<b>and</b>
		<b>a   b</b>	<b>or</b>
	^	<b>a ^ b</b>	<b>exclusive or</b>
	~^	<b>a ~^ b</b>	<b>exclusive nor</b>
Logical (note 2 )	&&	<b>a &amp;&amp; b</b>	<b>and</b>
		<b>a    b</b>	<b>or</b>
Relational (note 1 )	==	<b>a == b</b>	<b>equality</b>
	!=	<b>a != b</b>	<b>inequality</b>
	>	<b>a &gt; b</b>	<b>greater than</b>
	<	<b>a &lt; b</b>	<b>less than</b>
	>=	<b>a &gt;= b</b>	<b>greater or equal</b>
	<=	<b>a &lt;= b</b>	<b>less than or equal</b>
Shift	<<	<b>a &lt;&lt; b</b>	<b>logical shift left</b>
	>>	<b>a &gt;&gt; b</b>	<b>logical shift right</b>

- Notes
1. **a, b** widths must be identical. ✓
  2. **a, b** widths must be 1- bit, or relation/ logical operator subexpression. ✓

### 5.3.2 Unary operators

~	~a	<b>invert a</b>
^	^a	<b>parity of a</b>
~^	~^a	<b>not parity of a</b>
!	!a	<b>logical not</b>
&	&a	<b>unary and</b>
	a	<b>unary or</b>
~&	~&a	<b>unary nand</b>
~	~ a	<b>unary nor</b>

### 5.3.3 Miscellaneous operators

?:	<b>a ? b : c</b>	<b>conditional expression</b>
{.}	<b>{a, b, c}</b>	<b>concatenation</b>
{ }	<b>{a{ b}}</b>	<b>replication</b>

### 5.3.4 Library-only operators

For Verifiable RTL the following operators are supported only in libraries.

-	<b>-a</b>	<b>unary minus</b>
*	<b>a * b</b>	<b>multiply</b>
/	<b>a / b</b>	<b>divide</b>
==	<b>a == b</b>	<b>equality (0/ 1/ X/ Z)</b>
!=	<b>a != b</b>	<b>inequality (0/ 1/ X/ Z)</b>

### 5.3.5 Operator precedence

DON'T COUNT ON IT! Use parenthesis to force precedence. Verilog operator precedence is not 100% defined. In the absence of parenthesis, different tools may treat operator precedence differently. ✓

### 5.4 System tasks and functions

**\$display**("text, format specs", reg, reg, ... );  
**\$finish**;  
**\$time** //function  
**\$write** ("text, format specs", reg, reg, ... );  
**Supported format specs:** %b, %0b, %d, %0d, %h, %0h, %o, %0o, \t, \n, \l, \', \", %%

Copyright © 2001 Lionel Bening and Harry Foster  
 This document is Open Software, and may be freely copied under the rules of the "zlib/libpng License" as described in [http:// www.opensource.org/licenses/index.html](http://www.opensource.org/licenses/index.html)

For news about *Principles of Verifiable RTL Design* see [http:// verifiableRTL.com](http://verifiableRTL.com)

#### ORDER FORM

Please send me *Principles of Verifiable RTL Design*  
*Second Edition* by Lionel Bening and Harry Foster:  
 Copy(ies) of Hardbound, ISBN 0-7923-7368-5  
 EUR 127.00/ USD 110.00/ GBP 77. 00

- Payment enclosed in the amount of  
 Please invoice me     Please charge my credit card  
 Am. Ex.  Visa     Diners Club     Eurocard     Access

Name of Card Holder

Card No. Exp. Date:

Delivery address:

Name:

Address:

Date

Signature

European VAT Registration Number:

Return this order form to:

North and South America:  
 Kluwer Academic Publishers  
 P. O. Box 358 - Accord Station  
 Hingham, MA 02018-0358  
 Toll Free Phone:+ 866- 269- WKAP  
 Fax:+ 781- 681- 9045

Rest of World:  
 Kluwer Academic Publishers  
 P. O. Box 322, 3300 AH  
 Dordrecht, The Netherlands  
 Phone +31 786392 392  
 Fax:+ 31.786546474

# Verilog Instant Reference

for  
*Verifiable RTL  
Design*

Based on Kluwer  
bestseller  
*Principles of Verifiable  
RTL Design*

Rev. 1.0f