附录　语法参考

本附录提供了Verilog HDL 语言的所有语法。[⊖]

关键词

以下是Verilog HDL 硬件描述语言的关键词。注意，只有小写的名字才是关键词。

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

⊖ 源自IEEE Std 1364-1995 ，Copyright © 1995, IEEE, All rights reserved.

语法规范

下列规范应用于语法描述，规则采用巴科斯—诺尔范式(BNF)书写：

1) 语法规则按自左向右非终结字符的字母序组织。

2) 保留字、操作符和标点标记是语法的组成部分，以粗体字表示。

3) 非终结名字前的斜体名字的语义表示与非终结名字相关联。

4) 非粗体的垂直符号（|）用于分离可替换的选项。

5) 非粗体的方括号（[...]）表示可选项。

6) 非粗体的大括号（{...}）表明某项可以重复 0次或多次。

7) 以粗体出现的方括号、圆括号、大括号（[...], (...){...},）以及其他符号（如；）表示符号是语法的组成部分。

8) 起始的非终结名字为 “ 源文本 （source_text）”

9) 此语法中使用的终结名字以大写形式出现。

文法

```
always_construct ::=
       always
         statement

binary_base ::=
       'b | 'B

binary_digit ::=
       x | X | z | Z | 0 | 1

binary_number ::=
       [ size ] binary_base binary_digit { _ | binary_digit }

binary_operator ::=
          + | - | * | / | %
       |  == | != | === | !== | && | || | < | <= | > | >=
       |  & | | | ^ | ^~ | ~^ | >> | <<
block_item_declaration ::=
          parameter_declaration
       |  reg_declaration
       |  integer_declaration
       |  real_declaration
       |  time_declaration
       |  realtime_declaration
       |  event_declaration

blocking_assignment ::=
       reg_lvalue = [ delay_or_event_control ] expression

case_item ::=
          expression { , expression } : statement_or_null
       |  default [ : ] statement_or_null

case_statement ::=
          case ( expression ) case_item { case_item } endcase
       |  casez ( expression ) case_item { case_item } endcase
       |  casex ( expression ) case_item { case_item } endcase

charge_strength ::=
```

(small)
| **(medium)**
| **(large)**

cmos_switch_instance ::=
[name_of_gate_instance] **(** output_terminal **,** input_terminal **,**
ncontrol_terminal **,** pcontrol_terminal **)**

cmos_switchtype ::=
cmos | **rcmos**

combinational_body ::=
table
combinational_entry { combinational_entry }
endtable

combinational_entry ::=
level_input_list **:** output_symbol **;**

comment ::=
short_comment
| long_comment

comment_text ::=
{ ANY_ASCII_CHARACTER }

concatenation ::=
{ expression { **,** expression } }

conditional_statement ::=
if (expression **)** statement_or_null [**else** statement_or_null]

constant_expression ::=
constant_primary
| unary_operator constant_primary
| constant_expression binary_operator constant_expression
| constant_expression **?** constant_expression **:** constant_expression
| string

constant_mintypmax_expression ::=
constant_expression
| constant_expression **:** constant_expression **:** constant_expression

constant_primary ::=
number
| *parameter*_identifier
| *constant*_concatenation
| *constant*_multiple_concatenation

continuous_assign ::=
assign [drive_strength] [delay3] list_of_net_assignments **;**

controlled_timing_check_event ::=
timing_check_event_control specify_terminal_descriptor
[**&&&** timing_check_condition]

current_state ::=
level_symbol

data_source_expression ::=
expression

decimal_base ::=
'd | **'D**

decimal_digit ::=

```
     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
decimal_number ::=
        [ sign ] unsigned_number
     |  [ size ] decimal_base unsigned_number
delay2 ::=
        # delay_value
     |  # ( delay_value [ , delay_value ] )
delay3 ::=
        # delay_value
     |  # ( delay_value [ , delay_value [ , delay_value ] ] )
delay_control ::=
        # delay_value
     |  # ( mintypmax_expression )
delay_or_event_control ::=
        delay_control
     |  event_control
     |  repeat ( expression ) event_control
delay_value ::=
        unsigned_number
     |  parameter_identifier
     |  constant_mintypmax_expression
description ::=
        module_declaration
     |  udp_declaration
disable_statement ::=
        disable task_identifier ;
     |  disable block_identifier ;
drive_strength ::=
        ( strength0 , strength1 )
     |  ( strength1 , strength0 )
     |  ( strength0 , highz1 )
     |  ( strength1 , highz0 )
     |  ( highz1 , strength0 )
     |  ( highz0 , strength1 )
edge_control_specifier ::=
     edge [ edge_descriptor [ , edge_descriptor ] ]
edge_descriptor ::=
        01
     |  10
     |  0x
     |  x1
     |  1x
     |  x0
edge_identifier ::=
     posedge | negedge
edge_indicator ::=
        ( level_symbol level_symbol )
     |  edge_symbol
edge_input_list ::=
     { level_symbol } edge_indicator { level_symbol }
```

edge_sensitive_path_declaration ::=
        parallel_edge_sensitive_path_description = path_delay_value
    | full_edge_sensitive_path_description = path_delay_value

edge_symbol ::=
    r | R | f | F | p | P | n | N | *

enable_gate_instance ::=
    [name_of_gate_instance] **(** output_terminal **,** input_terminal **,**
    enable_terminal **)**

enable_gate_type ::=
    bufif0 | bufif1 | notif0 | notif1

enable_terminal ::=
    *scalar*_expression

escaped_identifier ::=
    \ { ANY_ASCII_CHARACTER_EXCEPT_WHITE_SPACE } white_space

event_control ::=
        @ *event*_identifier
    | **@ (** event_expression **)**

event_declaration ::=
    event *event*_identifier { **,** *event*_identifier } **;**

event_expression ::=
        expression
    | *event*_identifier
    | **posedge** expression
    | **negedge** expression
    | event_expression **or** event_expression

event_trigger ::=
    -> *event*_identifier **;**

expression ::=
        primary
    | unary_operator primary
    | expression binary_operator expression
    | expression **?** expression **:** expression
    | string

full_edge_sensitive_path_description ::=
    ([edge_identifier] list_of_path_inputs ***>** list_of_path_outputs
    [polarity_operator] **:** data_source_expression **)**

full_path_description ::=
    (list_of_path_inputs [polarity_operator] ***>** list_of_path_outputs **)**

function_call ::=
        *function*_identifier **(** expression { **,** expression } **)**
    | name_of_system_function [**(** expression { **,** expression } **)**]

function_declaration ::=
    function [range_or_type] *function*_identifier **;**
        function_item_declaration { function_item_declaration }
        statement
    endfunction

function_item_declaration ::=
        block_item_declaration
    | input_declaration

```
gate_instantiation ::=
        n_input_gatetype [ drive_strength ] [ delay2 ] n_input_gate_instance
            { , n_input_gate_instance } ;
      | n_output_gatetype [ drive_strength ] [ delay2 ] n_output_gate_instance
            { , n_output_gate_instance } ;
      | enable_gatetype [ drive_strength ] [ delay3 ] enable_gate_instance
            { , enable_gate_instance } ;
      | mos_switchtype [ delay3 ] mos_switch_instance
            { , mos_switch_instance } ;
      | pass_switchtype pass_switch_instance { , pass_switch_instance } ;
      | pass_en_switchtype [ delay3 ] pass_en_switch_instance
            { , pass_en_switch_instance } ;
      | cmos_switchtype [ delay3 ] cmos_switch_instance
            { , cmos_switch_instance } ;
      | pullup [ pullup_strength ] pull_gate_instance { , pull_gate_instance } ;
      | pulldown [ pulldown_strength ] pull_gate_instance
            { , pull_gate_instance } ;

hex_base ::=
      'h | 'H

hex_digit ::=
        x | X | z | Z
      | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
      | a | b | c | d | e | f | A | B | C | D | E | F

hex_number ::=
      [ size ] hex_base hex_digit { _ | hex_digit }

identifier :=
      IDENTIFIER [ { . IDENTIFIER } ]
      /* The period may not be followed or preceded by a space */

IDENTIFIER ::=
        simple_identifier
      | escaped_identifier

init_val ::=
      1'b0 | 1'b1 | 1'bx | 1'bX | 1'B0 | 1'B1 | 1'Bx | 1'BX | 1 | 0

initial_construct ::=
      initial
        statement

inout_declaration ::=
      inout [ range ] list_of_port_identifiers ;

inout_terminal ::=
        terminal_identifier
      | terminal_identifier [ constant_expression ]

input_declaration ::=
      input [ range ] list_of_port_identifiers ;

input_identifier ::=
        input_port_identifier
      | inout_port_identifier

input_terminal ::=
      scalar_expression

integer_declaration ::=
      integer list_of_register_identifiers ;
```

level_input_list ::=
  level_symbol { level_symbol }

level_symbol ::=
  0 | 1 | x | X | ? | b | B

limit_value ::=
  constant_mintypmax_expression

list_of_module_connections ::=
  ordered_port_connection { , ordered_port_connection }
  | named_port_connection { , named_port_connection }

list_of_net_assignments ::=
  net_assignment { , net_assignment }

list_of_net_decl_assignments ::=
  net_decl_assignment { , net_decl_assignment }

list_of_net_identifiers ::=
  *net*_identifier { , *net*_identifier }

list_of_param_assignments ::=
  param_assignment { , param_assignment }

list_of_path_delay_expressions ::=
  *t*_path_delay_expression
  | *trise*_path_delay_expression , *tfall*_path_delay_expression
  | *trise*_path_delay_expression , *tfall*_path_delay_expression ,
    *tz*_path_delay_expression
  | *t01*_path_delay_expression , *t10*_path_delay_expression ,
    *t0z*_path_delay_expression , *tz1*_path_delay_expression ,
    *t1z*_path_delay_expression , *tz0*_path_delay_expression
  | *t01*_path_delay_expression , *t10*_path_delay_expression ,
    *t0z*_path_delay_expression , *tz1*_path_delay_expression ,
    *t1z*_path_delay_expression , *tz0*_path_delay_expression ,
    *t0x*_path_delay_expression , *tx1*_path_delay_expression ,
    *t1x*_path_delay_expression , *tx0*_path_delay_expression ,
    *txz*_path_delay_expression , *tzx*_path_delay_expression

list_of_path_inputs ::=
  specify_input_terminal_descriptor { , specify_input_terminal_descriptor }

list_of_path_outputs ::=
  specify_output_terminal_descriptor { , specify_output_terminal_descriptor }

list_of_port_identifiers ::=
  *port*_identifer { , *port*_identifier }

list_of_ports ::=
  (port { , port })

list_of_real_identifiers ::=
  *real*_identifier { , *real*_identifier }

list_of_register_identifiers ::=
  register_name { , register_name }

list_of_specparam_assignments ::=
  specparam_assignment { , specparam_assignment }

long_comment ::=
  /* comment_text */

loop_statement ::=
  forever statement

```
      | **repeat** ( expression ) statement
      | **while** ( expression ) statement
      | **for** ( reg_assignment ; expression ; reg_assignment ) statement
```

mintypmax_expression ::=
  expression
  | expression : expression : expression

module_declaration ::=
 module_keyword *module*_identifier [list_of_ports] ;
  { module_item }
 endmodule

module_instance ::=
 name_of_instance ([list_of_module_connections])

module_instantiation ::=
 *module*_identifier [parameter_value_assignment] module_instance
  { , module_instance } ;

module_item ::=
  module_item_declaration
  | parameter_override
  | continuous_assign
  | gate_instantiation
  | udp_instantiation
  | module_instantiation
  | specify_block
  | initial_construct
  | always_construct

module_item_declaration ::=
  parameter_declaration
  | input_declaration
  | output_declaration
  | inout_declaration
  | net_declaration
  | reg_declaration
  | integer_declaration
  | real_declaration
  | time_declaration
  | realtime_declaration
  | event_declaration
  | task_declaration
  | function_declaration

module_keyword ::=
 module | **macromodule**

mos_switch_instance ::=
 [name_of_gate_instance] (output_terminal , input_terminal ,
  enable_terminal)

mos_switchtype ::=
 nmos | **pmos** | **rnmos** | **rpmos**

multiple_concatenation ::=
 { expression { expression { , expression } } }

n_input_gate_instance ::=
 [name_of_gate_instance] (output_terminal , input_terminal
  { , input_terminal })

n_input_gatetype ::=
and | **nand** | **or** | **nor** | **xor** | **xnor**

n_output_gate_instance ::=
[name_of_gate_instance] **(** output_terminal { **,** output_terminal } **,**
input_terminal **)**

n_output_gatetype ::=
buf | **not**

name_of_gate_instance ::=
*gate_instance*_identifier [range]

name_of_instance ::=
*module_instance*_identifier [range]

name_of_system_function ::=
$identifier

name_of_udp_instance ::=
*udp_instance*_identifier [range]

named_port_connection ::=
. *port*_identifier **(** [expression] **)**

ncontrol_terminal ::=
*scalar*_expression

net_assignment ::=
net_lvalue = expression

net_decl_assignment ::=
*net*_identifier = expression

net_declaration ::=
net_type [**vectored** | **scalared**] [range] [delay3] list_of_net_identifiers **;**
| **trireg** [**vectored** | **scalared**] [charge_strength] [range] [delay3]
list_of_net_identifiers **;**
| net_type [**vectored** | **scalared**] [drive_strength] [range] [delay3]
list_of_net_decl_assignments **;**

net_lvalue ::=
*net*_identifier
| *net*_identifier [expression]
| *net*_identifier [*msb*_constant_expression **:** *lsb*_constant_expression]
| *net*_concatenation

net_type ::=
wire | **tri** | **tri1** | **supply0** | **wand** | **triand** | **tri0** | **supply1** | **wor** | **trior**

next_state ::=
output_symbol | **-**

non_blocking_assignment ::=
reg_lvalue <= [delay_or_event_control] expression

notify_register ::=
*register*_identifier

number ::=
decimal_number
| octal_number
| binary_number
| hex_number
| real_number

octal_base ::=

```
    'o | 'O
octal_digit ::=
    x | X | z | Z | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
octal_number ::=
    [ size ] octal_base octal_digit { _ | octal_digit }
ordered_port_connection ::=
    [ expression ]
output_declaration ::=
    output [ range ] list_of_port_identifiers ;
output_identifier ::=
        output_port_identifier
    |   inout_port_identifier
output_symbol ::=
    0 | 1 | x | X
output_terminal ::=
        terminal_identifier
    |   terminal_identifier [ constant_expression ]
par_block ::=
    fork
      [ : block_identifier
        { block_item_declaration } ]
      { statement }
    join
parallel_edge_sensitive_path_description ::=
    ( [ edge_identifier ] specify_input_terminal_descriptor =>
      specify_output_terminal_descriptor [ polarity_operator ] :
      data_source_expression )
parallel_path_description ::=
    ( specify_input_terminal_descriptor [ polarity_operator ] =>
      specify_output_terminal_descriptor )
param_assignment ::=
    parameter_identifier = constant_expression
parameter_declaration ::=
    parameter list_of_param_assignments ;
parameter_override ::=
    defparam list_of_param_assignments ;
parameter_value_assignment ::=
    # ( expression { , expression } )
pass_en_switchtype ::=
    tranif0 | tranif1 | rtranif1 | rtranif0
pass_en_switch_instance ::=
    [ name_of_gate_instance ] ( inout_terminal , inout_terminal ,
      enable_terminal )
pass_switch_instance ::=
    [ name_of_gate_instance ] ( inout_terminal , inout_terminal )
pass_switchtype ::=
    tran | rtran
path_declaration ::=
```

simple_path_declaration **;**
| edge_sensitive_path_declaration **;**
| state_dependent_path_declaration **;**

path_delay_expression ::=
constant_mintypmax_expression

path_delay_value ::=
list_of_path_delay_expressions
| **(** list_of_path_delay_expressions **)**

pcontrol_terminal ::=
*scalar*_expression

polarity_operator ::=
+ | **-**

port ::=
[port_expression]
| **.** *port*_identifier **(** [port_expression] **)**

port_expression ::=
port_reference
| **{** port_reference { **,** port_reference } **}**

port_reference ::=
*port*_identifier
| *port*_identifier **[** constant_expression **]**
| *port*_identifier **[** *msb*_constant_expression **:** *lsb*_constant_expression **]**

primary ::=
number
| identifier
| identifier **[** expression **]**
| identifier **[** *msb*_constant_expression **:** *lsb*_constant_expression **]**
| concatenation
| multiple_concatenation
| function_call
| **(** mintypmax_expression **)**

procedural_continuous_assignment ::=
assign reg_assignment **;**
| **deassign** reg_lvalue **;**
| **force** reg_assignment **;**
| **force** net_assignment **;**
| **release** reg_lvalue **;**
| **release** net_lvalue **;**

procedural_timing_control_statement ::=
delay_or_event_control statement_or_null

pull_gate_instance ::=
[name_of_gate_instance] **(** output_terminal **)**

pulldown_strength ::=
(strength0 **,** strength1 **)**
| **(** strength1 **,** strength0 **)**
| **(** strength0 **)**

pullup_strength ::=
(strength0 **,** strength1 **)**
| **(** strength1 **,** strength0 **)**
| **(** strength1 **)**

pulse_control_specparam ::=
    PATHPULSE$ = (*reject*_limit_value [, *error*_limit_value]) ;
  | **PATHPULSE$**specify_input_terminal_descriptor /**no space; continue**/
    $specify_output_terminal_descriptor = (*reject*_limit_value
    [, *error*_limit_value]) ;^{1}

range ::=
  [*msb*_constant_expression : *lsb*_constant_expression]

range_or_type ::=
  range | **integer** | **real** | **realtime** | **time**

real_declaration ::=
  real list_of_real_identifiers ;

real_number ::=
    [sign] unsigned_number . unsigned_number
  | [sign] unsigned_number [. unsigned_number] e [sign]
      unsigned_number
  | [sign] unsigned_number [. unsigned_number] E [sign]
      unsigned_number

realtime_declaration ::=
  realtime list_of_real_identifiers ;

reg_assignment ::=
  reg_lvalue = expression

reg_declaration ::=
  reg [range] list_of_register_identifiers ;

reg_lvalue ::=
    *reg*_identifier
  | *reg*_identifier [expression]
  | *reg*_identifier [*msb*_constant_expression : *lsb*_constant_expression]
  | *reg*_concatenation

register_name ::=
    *register*_identifier
  | *memory*_identifier [*upper_limit*_constant_expression :
      *lower_limit*_constant_expression]

scalar_constant ::=
  1'b0 | **1'b1** | **1'B0** | **1'B1** | **'b0** | **'b1** | **'B0** | **'B1** | **1** | **0**

scalar_timing_check_condition ::=
    expression
  | ~ expression
  | expression == scalar_constant
  | expression === scalar_constant
  | expression != scalar_constant
  | expression !== scalar_constant

seq_block ::=
  begin
    [: *block*_identifier
      { block_item_declaration }]
    { statement }
  end

seq_input_list ::=
  level_input_list | edge_input_list

sequential_body ::=

```
    [ udp_initial_statement ]
    table
      sequential_entry
      { sequential_entry }
    endtable

sequential_entry ::=
    seq_input_list : current_state : next_state ;

short_comment ::=
    // comment_text \n

sign ::=
    + | -

simple_identifier ::=
    [a-zA-Z][a-zA-Z_$0-9]

simple_path_declaration ::=
      parallel_path_description = path_delay_value
    | full_path_description = path_delay_value

size ::=
    unsigned_number

source_text ::=
    { description }

specify_block ::=
    specify
      { specify_item }
    endspecify

specify_input_terminal_descriptor ::=
      input_identifier
    | input_identifier [ constant_expression ]
    | input_identifier [ msb_constant_expression : lsb_constant_expression ]

specify_item ::=
      specparam_declaration
    | path_declaration
    | system_timing_check

specify_output_terminal_descriptor ::=
      output_identifier
    | output_identifier [ constant_expression ]
    | output_identifier [ msb_constant_expression : lsb_constant_expression ]

specify_terminal_descriptor ::=
      specify_input_terminal_descriptor
    | specify_output_terminal_descriptor

specparam_assignment ::=
      specparam_identifier = constant_expression
    | pulse_control_specparam

specparam_declaration ::=
    specparam list_of_specparam_assignments ;

state_dependent_path_declaration ::=
      if ( conditional_expression ) simple_path_declaration
    | if ( conditional_expression ) edge_sensitive_path_declaration
    | ifnone simple_path_declaration

statement ::=
        blocking_assignment ;
      | non_blocking_assignment ;
      | procedural_continuous_assignment ;
      | procedural_timing_control_statement
      | conditional_statement
      | case_statement
      | loop_statement
      | wait_statement
      | disable_statement
      | event_trigger
      | seq_block
      | par_block
      | task_enable
      | system_task_enable
statement_or_null ::=
      statement | ;
strength0 ::=
      supply0 | strong0 | pull0 | weak0
strength1 ::=
      supply1 | strong1 | pull1 | weak1
string ::=
      " { ANY_ASCII_CHARACTERS_EXCEPT_NEWLINE } "

                    | path_declaration
system_task_name ::=
      $identifier
      /* The $ cannot be followed by a space */
system_timing_check ::=
        $setup ( timing_check_event , timing_check_event , timing_check_limit
          [ , notify_register ] ) ;
      | $hold ( timing_check_event , timing_check_event , timing_check_limit
          [ , notify_register ] ) ;
      | $period ( controlled_timing_check_event , timing_check_limit
          [ , notify_register ] ) ;
      | $width ( controlled_timing_check_event , timing_check_limit ,
          constant_expression [ , notify_register ] ) ;
      | $skew ( timing_check_event , timing_check_event , timing_check_limit
          [ , notify_register ] ) ;
      | $recovery ( controlled_timing_check_event , timing_check_event ,
          timing_check_limit [ , notify_tegister ] ) ;
      | $setuphold ( timing_check_event , timing_check_event ,
          timing_check_limit , timing_check_limit [ , notify_register ] ) ;
task_declaration ::=
      task task_identifier ;
        { task_item_declaration }
        statement_or_null
      endtask
task_enable ::=
      task_identifier [ ( expression { , expression } ) ] ;
task_item_declaration ::=
        block_item_declaration
      | input_declaration
```

| output_declaration
| inout_declaration

time_declaration ::=
time list_of_register_identifiers **;**

timing_check_condition ::=
scalar_timing_check_condition
| **(** scalar_timing_check_condition **)**

timing_check_event ::=
[timing_check_event_control] specify_terminal_descriptor
[**&&&** timing_check_condition]

timing_check_event_control ::=
posedge
| **negedge**
| edge_control_specifier

timing_check_limit ::=
expression

udp_body ::=
combinational_body
| sequential_body

udp_declaration ::=
primitive *udp*_identifier **(** udp_port_list **) ;**
udp_port_declaration
{ udp_port_declaration }
udp_body
endprimitive

udp_initial_statement ::=
initial *udp_output_port*_identifier = init_val **;**

udp_instance ::=
[name_of_udp_instance] **(** output_port_connection **,** input_port_connection
{ **,** input_port_connection } **)**

udp_instantiation ::=
*udp*_identifier [drive_strength] [delay2] udp_instance { **,** udp_instance } **;**

udp_port_declaration ::=
output_declaration
| input_declaration
| reg_declaration

udp_port_list ::=
*output_port*_identifier **,** *input_port*_identifier { **,** *input_port*_identifier }

unary_operator ::=
+ | **-** | **!** | **~** | **&** | **~&** | **|** | **~|** | **^** | **~^** | **^~**

unsigned_number ::=
decimal_digit { _ | decimal_digit }

wait_statement ::=
wait (expression **)** statement_or_null

white_space ::=
space | tab | newline

参 考 文 献

Prentice Hall, NJ, 1998.

2. Bhasker J., *Verilog HDL Synthesis: A Practical Primer*, Star Galaxy Pu lishing, PA, 1998, ISBN 0-9650391-5-3.
3. Lee J., *Verilog Quickstart*, Kluwer Academic, MA, 1997.
4. Palnitkar S., *Verilog HDL: A Guide to Digital Design and Synthe* Prentice Hall, NJ, 1996, ISBN 0-13-451675-3.
5. Sagdeo V., *Complete Verilog Book*, Kluwer Academic, MA, 1998.
6. Smith D., *HDL Chip Design*, Doone Publications, 1996.
7. Sternheim E., R. Singh and Y. Trivedi, *Digital Design and Synthesis w Verilog HDL*, Automata Publishing Company, CA, 1993.
8. Thomas D. and P. Moorby, *The Verilog Hardware Description Langua* Kluwer Academic, MA, 1991, ISBN 0-7923-9126-8.

10. Open Verilog International, *OVI Standard Delay File (SDF) Format Manual.*